e-Informatica Software Engineering Journal, Volume 16, Issue 1, 2022, pages: 220110, DOI: 10.37190/e-Inf220110

Reuse in Contemporary Software Engineering Practices – An Exploratory Case Study in A Medium-sized Company

Xingru Chen*, Deepika Badampudi*, Muhammad Usman*
*Department of Software Engineering, Blekinge Institute of Technology, SE-37179, Karlskrona, Sweden
xingru.chen@bth.se, deepika.badampudi@bth.se, muhammad.usman@bth.se

Abstract

Background: Software practice is evolving with changing technologies and practices such as InnerSource, DevOps, and microservices. It is important to investigate the impact of contemporary software engineering (SE) practices on software reuse.
Aim: This study aims to characterize software reuse in contemporary SE practices and investigate its implications in terms of costs, benefits, challenges, and potential improvements in a medium-sized company.
Method: We performed an exploratory case study by conducting interviews, group discussions, and reviewing company documentation to investigate software reuse in the context of contemporary SE practices in the case company.
Results: The results indicate that the development for reuse in contemporary SE practices incurs additional coordination, among other costs. Development with reuse led to relatively fewer additional costs and resulted in several benefits such as better product quality and less development and delivery time. Ownership of reusable assets is challenging in contemporary SE practice. InnerSource practices may help mitigate the top perceived challenges: discoverability and ownership of the reusable assets, knowledge sharing and reuse measurement.
Conclusion: Reuse in contemporary SE practices is not without additional costs and challenges. However, the practitioners perceive costs as investments that benefit the company in the long run.

Keywords: software reuse, contemporary SE practices, software reuse costs and benefits, software reuse challenges and improvements, InnerSource

1. Introduction

Software reuse is commonly practiced in organizations and is described as "*the systematic use of existing software assets to construct new or modified ones or products*" [1]. The benefits of software reuse such as improved product quality, faster time-to-market and reduced development costs [1–3] are well-acknowledged. Although software reuse has been studied for more than five decades, with the constant changes in architecture patterns and styles (e.g., microservices), and processes (e.g., InnerSource [4]), the research in software reuse still remains relevant. In 2019, Barros-Justo et al. [5] conducted a tertiary study to investigate the trends in software reuse research. They identified many software reuse

Submitted: 08 Apr. 2022; Revised: 02 Jul. 2022; Accepted: 28 Jul. 2022; Available online: 02 Sep. 2022

research proposals related to 1) requirements engineering, testing, and design activities, 2) evolution/maintenance and variability management in project and process management, and 3) other general reuse topics, such as decision-making based on systematic software reuse and metrics to evaluate the reuse performance. Capilla et al. [6] also identified new software reuse research opportunities in the context of new application domains, new software reuse techniques and methods.

The growing popularity of open source use has also impacted software reuse. Mikkonen and Taivalsaari [7] identified the growing popularity of opportunistic design, which is "*developing new software systems by routinely reusing and combining components (open source components and modules online) that were not designed to be used together*". Xu et al. [8] also identified a trend in increased library reuse of Maven libraries in Maven Central[1]. Although opportunistic reuse is the opposite of systematic reuse as Barros-Justo et al. [2] and Capilla et al. [9] stated, in the long run, there is a need to systematize the maintenance of the external reusable assets from the open source. In addition to the open source software, the open source "way of working" is adopted by various software organizations, e.g., GlobalSoft/SoftCom [10], and IBM (CMS) [11, 12]. Inspired by the open source way of working, Tim O'Reilly coined the term InnerSource [4] as "*the use of open source development techniques within the corporation*" in 2000. Vitharana et al. [12] further conceptualized InnerSource, particularly consumer contributions to reusable assets, as participatory reuse. The authors described participatory reuse as "the scenario in which potential reusers participate in the entire development process (e.g., analysis, design, development, testing) to ensure that the project assets meet their reuse needs."

In addition to the open source and IS practices, the changing technology also impacts software reuse, i.e., the unit of reuse changed from components to microservices. Organizations are increasingly adopting microservices, together with DevOps practices (e.g., continuous integration and deployment) and container-based solutions (e.g., Docker) to improve the delivery time and scalability of their products and systems (cf. [6, 13–15]).

Studies have investigated the impact of opportunistic reuse [9], microservices [16], and InnerSource [17] on software reuse. Capilla et al. [9] found negative impacts of opportunistic reuse on software reuse. Their results indicate that the integration of reusable assets found opportunistically increases the number of smells and issues in most cases. Gouigoux and Tamzalit [16] identified increased reuse as one of the main benefits of migrating from monolith to microservices based architecture solutions. InnerSource (IS) practices facilitate software reuse [17]. When consumers of reusable assets also participate in developing and maintaining the reused assets, it further promotes reuse. The above mentioned software engineering (SE) practices can be referred to as contemporary SE practices.

While studies investigate the impact of individual contemporary SE practices on software reuse, there is no empirical investigation of software reuse in a combination of contemporary SE practices. We refer to *software reuse in contemporary SE practices* as organizations practicing both opportunistic reuse – leveraging open-source assets and libraries wherever possible, and participatory reuse with the help of IS practices for collaboratively developing reusable assets, together with the adoption of new technical solutions such as microservices-based architectures and DevOps practices.

It is important to investigate if the previously well-known challenges of software reuse are still applicable in the context of contemporary SE practices and discover the new implications of software reuse. For example, when developing reusable assets in participatory

[1]Statistics for the Maven Repository, https://search.maven.org/stats

reuse, additional coordination may be required when accepting contributions from other teams. Furthermore, other teams may need additional documentation to understand how they can contribute. The ownership of the reusable assets can be complicated when different teams are involved in development. Opportunistic reuse involves additional integration effort due to differences in the architectural style and technology in the target project and the reusable asset from open source [9].

Barros-Justo et al. [18] pointed out the empirical evidence of software reuse in practice, particularly in medium-sized companies, is limited. Also, existing systematic literature studies on software reuse highlight the need for more empirical studies [1–3]. Therefore, we contribute by investigating the state-of-the-practice of software reuse in the context of contemporary SE practices in S-Group Solutions AB[2], a medium-sized Swedish IT company.

In our study we characterize software reuse in contemporary SE practices. We conducted an exploratory case study to investigate how practitioners practice software reuse with contemporary SE practices and how they perceive its costs, benefits, challenges, and improvements. We collected the data using in-depth interviews, group discussions and document analysis. Reduced time and improved product quality are the main identified benefits. The study participants perceived additional coordination with stakeholders as a cost in both, development and use of a reusable asset. The study participants identified discoverability and ownership of the reusable assets, knowledge sharing and reuse measurement as the top focused challenges and improvement areas. The participants were in consensus on adopting IS patterns[3] in order to address the top listed challenges and improvement areas.

The remainder of the paper is structured as follows: Section 2 presents the related work; Section 3 describes the study design; Section 4 provides the study results; Section 5 discusses the results in comparison to the related works and provides discussion on threats to validity; Section 6 concludes the paper and proposes the future work.

2. Related work

Companies adopt software reuse practices to achieve certain benefits (e.g., better productivity), which leads to additional costs. Likewise, adopting software reuse practices also results in some challenges, which researchers try to solve by proposing improvement suggestions. This section will present an overview of the related works on software reuse costs, benefits, challenges, and improvements.

Many studies identified increased development productivity and better product quality as software reuse benefits, which includes both internal [19–21] and opportunistic reuse [19, 22, 23]. Furthermore, less maintenance effort [18, 19, 23], standardized architecture [21] and higher documentation quality [18] have also been identified as benefits of software reuse.

Relatively fewer studies investigated software reuse costs than benefits. Kruger and Berger [20] discovered that the majority of the additional reuse costs relate to the development for reuse phase. They noted that developing assets for reuse is generally more costly than developing for single use. However, Mohagheghi et al. [21] investigated the relation

[2]https://sgroup-solutions.se/

[3]https://patterns.innersourcecommons.org/

between software reuse and increased rework and did not find a cause-effect relationship between them.

The implementation of the software reuse initiative is not without challenges. Barros-Justo et al. [18] replicated Bauer et al.'s study [19], investigating the challenges and problems related to software reuse practices. Both studies [18, 19] identified the same software reuse related challenges including licensing issues, "not invented here" syndrome, inadequate granularity of reusable assets, accessibility of reusable assets, decrease of code understandability and difficulties in modifying the code due to software reuse. Mäkitalo et al. [23] and Barros-Justo et al. [18] also found that fixing compatibility and dependency related issues is particularly challenging in case of reusable assets. In addition to these technical challenges, coordination among the teams working on the development of the reusable assets is also a challenge [20].

Some improvement suggestions have also been proposed in the literature to address the challenges associated with the development of reusable assets. For example, using a written reuse guidebook to improve the understandability of the reusable assets [24], tools to help improve the discoverability of the reusable assets [24, 25] and allocating developers a separate time budget to develop or maintain the reusable assets [24].

Barros-Justo et al. [18] pointed out that few empirical studies on software reuse exist in small to medium-sized companies, therefore, they conducted a survey study in a medium-sized company to fill the gap. Our study further contributes to the medium-sized company context. We conducted an exploratory case study to cover the topic in more depth, using interviews and group discussions. In our study, we collected data about software reuse costs and benefits as well as about reuse related challenges and improvements in the context of contemporary SE practices. Moreover, we also discussed the feasibility of adopting selected IS patterns to address the identified challenges and improvement areas.

3. Study design

This section presents the details of the study design.

3.1. Research method

The study is part of a research project aimed at improving the internal reuse practices of the partner companies. In a joint discussion involving both company and research team members, it was decided to start with an initial study to understand the current state-of-the-reuse practice at the case company. We chose an exploratory case study [26] as our research method to investigate the current reuse practice in the company. We used three data collection methods (see Section 3.4 and Table 1): interviews, group discussions and company documentation.

3.2. Research questions

We formulated the following three research questions to guide our study:
RQ1: How software reuse is conducted in the context of contemporary SE practices in a medium-sized company? Motivation: To understand the software reuse strategies in contemporary SE practices, we aim to characterize the reuse process. We investigated the company's reuse related activities, roles and workflows.

RQ2: What are the costs and benefits of practicing software reuse in the context of contemporary SE practices in a medium-sized company? Motivation: To understand practitioners' perceptions of software reuse costs and benefits in the context of contemporary SE practices. Cost is the extra/additional effort required to develop, maintain or use the reusable assets.
RQ3: What are the challenges in practicing software reuse in the context of contemporary SE practices in a medium-sized company, and how can they be improved? Motivation: To understand what challenges, issues or problems the practitioners in a medium-sized company encountered in software reuse in the context of contemporary SE practices. To collect the improvements from the practitioners view and discuss other possible interventions with practitioners that can facilitate software reuse.

3.3. Case company and unit of analysis

The case company, S-Group Solutions AB, is a private Swedish IT company that focuses on developing spatial information and geographical information systems (GIS). The company offers its solutions to the public sector and the target customers are mainly local governments and authorities. S-Group Solutions AB has 65 employees and can therefore be classified as a medium-sized company [27]. The software development organization of the company consists of 29 people and it is divided into three teams corresponding to four solution areas. Each development team consists of an average of five developers each, with one senior developer acting as the tech lead. Each solution area has a corresponding project manager, a product owner and a tester. The development organization has one software architect who oversees and guides all teams and is responsible for maintaining the integrity of the overall software design and architecture. In addition, the company also has a support team, a UX engineer and a technical writer. The development teams follow agile practices (e.g., daily standup and sprint planning) to manage their work. S-Group Solutions AB uses Azure DevOps and has continuous integration and delivery (CI/CD) pipeline, which updates every midnight. Currently, S-Group Solutions AB is migrating some codes from a monolithic architecture to a microservices-based architecture. The unit of analysis is the software reuse practice at the case company. Currently, two of the three teams are more involved in the development and use of the reusable assets, while the other team is relatively new in this reuse journey.

3.4. Data collection

The data is collected through semi-structured interviews, multiple group discussions and company documentation. The aim of each data collection method and its corresponding research questions (RQs) are presented in Table 1. We used group discussions and the company documentation to validate and triangulate the interview data (see aims in Table 1). The software architect was our contact person at the case company, who has a long working experience (12 years) at the company and has a leading role in introducing the software reuse related practices. We used semi-structured interviews to collect data since it allows improvisation and exploration of the studied objects [26] and captures unexpected information on the studied topic [28]. The group discussions are used since it collects in-depth perspectives through interactive conversations with multiple participants, not only with the moderator/interviewer as interviews do. Allowing multiple opinions provides more descriptive and elaborated data.

Table 1. Aims and corresponding research questions of the data collection methods

Data collection methods	Aims	RQ
Interviews	Understand how software reuse is practiced in the company, and collect the practitioners' perceptions on software reuse costs, benefits, challenges and improvements	RQ1, RQ2, RQ3
Group discussions	1) Validate the interview results, and get additional inputs	RQ1, RQ2, RQ3
	2) Collect the challenge prioritization results from the company's perspective 3) Discuss the feasibility and application of the interventions which were proposed by the authors and collect the feedback from the company	RQ3
Company documentation	Understand the company structure in a written form and triangulate it with the interview results	RQ1

Selection of the interview participants. To select the right person as participant candidates, we shared with the software architect a list of candidate roles related to software reuse at the company, including producers and consumers of the reusable assets and their managers and team leads. The software architect helped us identify four participants initially – the software architect himself, two tech leads (representing two different teams) involved in the development and consumption of reusable assets and one product owner. During the interviews with these four participants, we also identified the need to cover the role of a tester and a project manager. With the help of the software architect, we managed to interview one tester and one project manager. Table 2 shows the summary of the participants. In total, we have interviewed 20 percent of the population (6 out of 29), which covers all teams, and both technical and non-technical roles.

Table 2. Overview of the interview participants

PID	Team	Current role	Exp[a]	Interview duration
P1	Team 2	Product owner (PO)	27	1h10mins
P2	Team 1	System developer and tech lead (TL1)	3	1h10mins
P3	Team 3	System developer and tech lead (TL2)	7	1h30mins
P4	All teams	Software architect (SA)	12	1h20mins
P5	Team 1	Tester (T)	4	55mins
P6	Team 1	Project manager (PM)	6	1h

[a] Experience in number of years the practitioner is working with the current company.

Interview design. As mentioned in Section 3.1, we chose to conduct semi-structured interviews. The second author developed the interview guide (see Table 3) and it was reviewed independently by the other two authors and a senior researcher from the research project, which resulted in minor reformulations. We also performed a pilot interview with a practitioner from another company to test the interview guide. The interview guide contains seven aspects: introduction, participants' background, reuse practices, costs, benefits, challenges and improvements. The mapping between the interview questions and RQs are presented in Table 3. We used the interview questions as a guide and followed semi-structured interview format which allowed for flexibility in the interview.

Table 3. Mapping between the interview* questions and RQs

Interview questions	Corresponding RQs
1. Overview of the current reuse practices and your role a) Details on role and experience: i: What is your role? Please provide a short overview of the tasks that you perform in your role. ii: Which other roles do you interact with and why? iii: What is your overall experience and experience with development for and with reuse?	Demographic information
b) Details on the product, team/s and shared assets: i: What is the size of your team? ii: Which software artefacts (requirements, test cases, code, etc.) you work with? Format of requirements? iii: Do you prefer development of assets from scratch or reuse? existing/available assets? What is the motivation behind your preference? iv: Which software assets do you/your team share (across site?)? What solution do they offer? Can you give an example? v: Do you produce and/or consume the shared assets? Give examples of the shared assets your are involved in?	RQ1
2. Is there a company/project/unit wide strategy/policy/goal to develop with reuse?	RQ1
3. Is there a company/project/unit wide strategy/policy/goal to develop for reuse (i.e., developing assets, e.g., code, with the aim to make them reusable)?	RQ1
4. How is the funding of shared assets done?	RQ1
5. What is your experience regarding developing for and with reuse? What reuse[4] related activities/tasks/initiatives are you involved in?	RQ1
6. What activities, if any, are performed in your company to: a) Identify the reusable assets b) Develop/adapt reusable assets. i: What are the unique activities in development of reusable assets? c) Use reusable assets or replace existing assets with reusable assets. d) Maintain reusable assets. e) Share reusable assets or make reusable assets available. i:[–] **For all activities ask the following questions:** ii: Is there anyone response for this activity? If yes, who? If not, should there be any one responsible? iii: Who or what triggers this activity and how often? iv: What information/input is needed for the activity? v: Who provides the information needed for the activity or how is it obtained?	RQ1
7. Benefits of development for reuse and with reuse (what are the reuse benefits and how are they measured?) How reuse benefits – i: the organization, ii: your role (**incentive**), iii: the product, iv: business/customers, v: the team?	RQ2
8. Costs of development for reuse and with reuse (what are the reuse costs and how are they measured?) How reuse costs affect – i: the organization, ii: your role, iii: the product, iv: business/customers, v: the team?	RQ2
9. What are the challenges and improvement areas with respect to development for and with reuse?	RQ3

* In this interview, we want to know your view on software reuse. In particular, we want to know your experiences of developing and/or using reusable assets. Assets include components, microservices, APIs etc. developed either in-house or acquired from open source projects that could be reused within the company.

Due to the Covid-19 pandemic, we conducted the interviews online using Microsoft Teams and the interview duration was set to approximately one and half hours. To provide

[4]Reuse could also mean using the same open source component that someone else at the company has adopted.

some context and background information, we shared high-level interview questions with the participants before the interviews. The authors distributed their tasks mainly in three parts during the interview: lead the interview, ask follow-up questions and take notes. All three authors were involved in all the parts by switching their tasks in different interviews. We requested participants' permissions to audio record the interviews.We guaranteed that the data will only be stored in a local drive and will only be used in an aggregate form during the analysis and presentation of results. All interview participants gave their consent for audio recording of the interviews.

Selection of the group discussions participants. As described previously, the study is part of a research project. Keeping in view the relevance of the topics covered in the study and the long-term goals of the project, we formed a discussion group to take the study and the project forward. The discussion group consists of five members – including two participants from the case company – the software architect and the project manager – and the authors representing the project's research team. The software architect has a leading role in software reuse practice in the company and interacts with all development teams about the technical issues. Therefore, his involvement was necessary for the study and project. The project manager's role is also important as he is responsible for planning and managing the more active teams in developing and maintaining the reusable assets. Including the project manager ensures coverage of project management and planning-related perspectives in the discussions.

Group discussions design and company documentation review. Similar to the interviews, group discussions were also held online through Microsoft Teams due to the Covid-19 pandemic. We conducted discussions to validate interview results, prioritize the challenges, and discuss the potential improvements to address the prioritized challenges. In the group discussion on validating interview results, we presented the interview results to the company contact person. The results of the interview data validation are provided in Section 3.5. Prior to the next group discussion, we asked the company contact person to conduct an internal discussion with the team members to prioritize the challenges identified in the interviews. In addition, we investigated the possible solutions for the identified challenges from the existing literature. Then in a group discussion, the company contact mentioned the prioritized challenges (reported in Section 4.3.3). In the same group discussion meeting, we provided potential solutions to mitigate the prioritized challenges. We then discussed the feasibility of the proposed solutions with the contact person and the project manager (see Section 4.3.3). The project manager provided company documentation that included information about the people involved in reuse practices, reuse context and the reuse activities, which we used to triangulate the interview results. On average, the discussions lasted for an hour. The research team took extensive notes during the discussions. The discussions concluded with one of the authors sharing the summary and confirming the next steps (e.g., who is expected to do what before the next group discussion).

3.5. Interview data analysis approach

To enable the data analysis, the first author transcribed word to word of approximately seven hours of audio recordings from the interviews. The other two authors did a preliminary analysis by extracting and analyzing the relevant data from the transcripts. The authors held a joint meeting to discuss the interview credibility, the transcription quality and the preliminary analysis findings. At the end of the discussion, we reached a consensus on the findings and agreed that all six interviews are eligible for the study. We presented the

results from the preliminary analysis to the software architect. Apart from a few minor corrections, the software architect agreed and was able to relate to the results.

Taking inspiration from the recommended thematic synthesis steps [29] and four-steps data analysis process [30], we used the following integrated approach (both inductive and deductive approaches) to code and analyze our data. The entire data analysis process is described as follows:

Generating start list and clustering. We created a general start list for clustering according to the research questions and context information. The start list acted as a preliminary theme to group the raw data according to the general domain instead of content-specific, enabling inductive coding. The start list contained seven aspects: personal background, reuse context, reuse activities, reuse costs, reuse benefits, reuse challenges and reuse improvements. The first author extracted the relevant text segments from the transcripts and grouped them according to the generated start list.

Inductive coding within clusters. We followed the descriptive coding method according to Saldana's coding manual book [31]. The first and second authors agreed on a code naming style and then they independently coded the text segments from the reuse benefits cluster to pilot the code naming style. During the coding process, more text segments from the transcripts were extracted when needed. The piloting results showed that we were consistent in the code meanings and the corresponding text segments. However, we needed to agree on a common name for each code. For example, we named the same text segments "save time" and "reduced development time". We eventually used "reduced development time" and agreed that the code names should be more explicit. After the piloting, the two authors independently coded clusters related to costs, challenges and improvements, and arranged several meetings to address the disagreements. Apart from refining the code names, the coding results showed that the first and second authors had similar opinions on the codes and text segments. We merged two codes into one for the benefits cluster related to maintenance, and merged another two codes into one for the costs cluster related to team coordination. To enhance comprehension, we also cleaned the text segments jointly. All the changes are logged to ensure code traceability. A codebook was generated when the two authors reached a consensus on the codes and code descriptions. Using the codebook as a reference, the first author went through all six transcripts again to ensure we did not miss any relevant text segments. The first author extracted 15 new text segments, which resulted in two code modifications. When there was no disagreement between the first and second authors, they asked the third author for review. The review contains four parts: the suitableness of the unit (broad or brief) of the extracted text segments, the relevance between the codes and the text segments, the coverage of the codes and the clarity of the code descriptions. We discussed the review results and addressed the disagreements in a joint meeting among all authors. In the review process, the third author suggested the removal of one code for benefits and three codes for challenges due to their low relevance to software reuse. The relevance of these codes was discussed jointly and all three authors agreed to remove them. In addition, we further refined the code names and extracted additional text segments according to the review suggestions.

Translate codes into themes. We used pattern coding [31] to generate themes according to the codes relations and thematic map to visualize and organize the codes and themes. The first author independently came up with the themes for four clusters. Then the second and third authors individually reviewed the appropriateness of the themes. The disagreements were addressed iteratively through multiple discussions.

Validation of the interview findings. To reduce the risk of researchers' bias in data interpretation, we shared the study report with the company contact person for review. The contact person commented that one of the challenges was about technical incompatibility than a challenge caused by software reuse. Therefore, we removed this challenge from our results. Overall, the contact person confirmed that the results reflected the reality. In addition, the results were also presented, discussed and agreed upon in one of the group discussions.

4. Results

This section presents the results per research question.

4.1. RQ 1. Reuse practice and contemporary SE practices in a medium-sized company

The software reuse process in S-Group Solutions AB consists of both participatory reuse and opportunistic reuse, as presented in Figure 1. The potential reusable assets at the case company are code (packages and microservices), requirements and automated test cases. To better explain how software reuse is practiced in the contemporary software engineering practices environment, we characterize different reuse activities in participatory reuse and opportunistic reuse.

In participatory reuse (see the solid-lined box on top in Figure 1), the development teams that use the reusable assets also participate in the development process. The software architect, product owners, project managers and tech leads conduct a solution vision meeting to propose reusable candidates before the project initiates and share the knowledge among the teams. To facilitate the communication across different solution areas, people in the same role sit together, i.e., testers at one place and product owners at one place, which also facilitates exploring potential reuse opportunities across different teams. After identifying the internal reusable candidates in solution vision, the software architect and tech leads perform technical analysis to discuss overall design, such as the API design. Once the technical analysis is completed, the relevant team develops the reusable code assets themselves. When the consumers or other developers want to contribute to the internal reusable code assets, they need to coordinate with the owner of the reusable assets to align the needs from both sides (consumers and producers) and understand the code commit requirements. There are two types of contributions: One is to add new functionalities to the reusable assets and the other is to fix the bugs in the reusable assets. The reusable assets include npm packages[5], NuGet packages[6] and microservices, which are stored in the internal DevOps repository (Azure DevOps server). The internal DevOps repository has the capability of keyword searching, which helps to look up the shared reusable assets. For the most reused packages, a read-me file provides a short description of the package. All developers are potential producers and consumers of the shared reusable assets, i.e., all developers within the company can reuse the shared packages, and if the consumers identify the need to fix or add something, they need to do that on their own. After the update, a new version number is assigned to the revised package. The case company has reached nearly 40% of the reuse rate, as a ratio of reusable assets from previous projects and the

[5]https://www.npmjs.com/
[6]https://www.nuget.org/

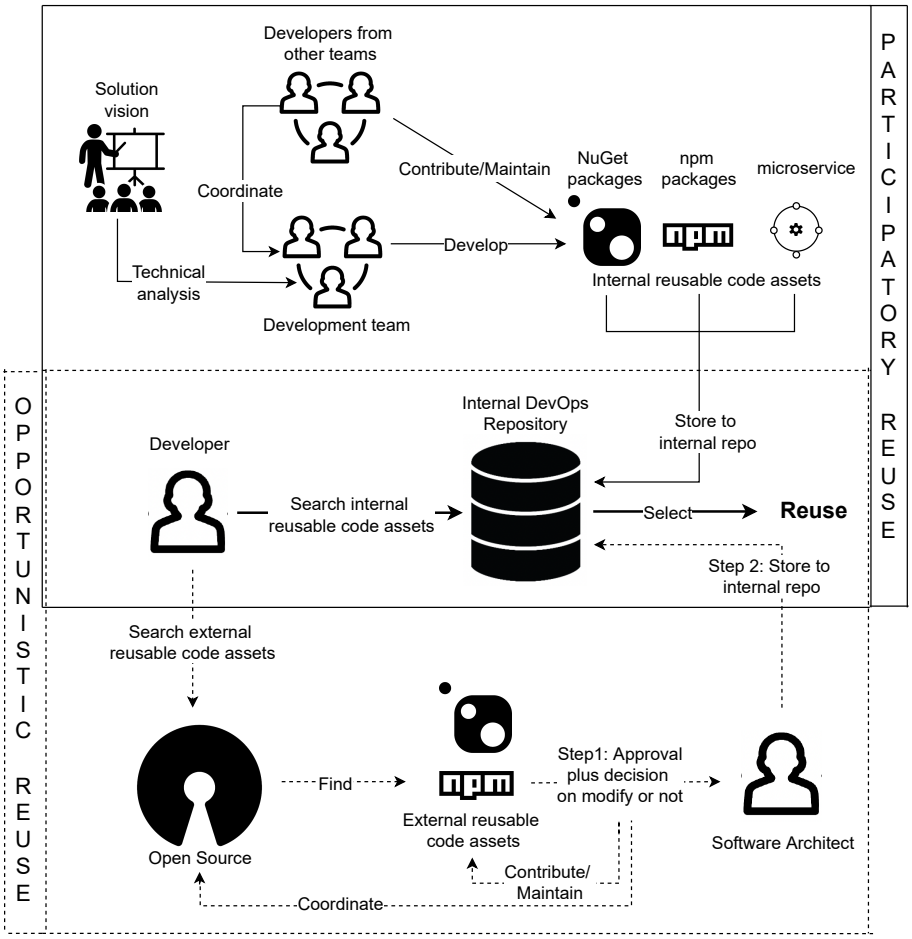

Figure 1. Software reuse and contemporary SE practices in S-Group Solutions AB

newly developed project. The company aims to increase the participatory reuse since it supports the development and maintenance of the reusable assets and also facilitates the company-wide adoption of the reuse practices.

In opportunistic reuse (see the dotted-lined box at bottom in Figure 1), the company reuses from the open-source community. However, developers need to get approval from the software architect before importing the external reusable assets into the internal DevOps repository and reusing them. The approval includes risk analysis, such as the fit to purpose check and the associated community activeness check. The developers need to coordinate with the open source community for bug fixes and new feature requests. They also make upstream contributions. If the open source maintainers do not respond in time, the software architect and developers need to decide whether to modify the external reusable code assets themselves and store the modified ones into the internal DevOps repository (provided that the OSS license permits). The opportunity reuse in the company is limited in package reuse only.

According to the software reuse process description in S-Group Solutions AB, InnerSource is practiced by accepting other teams' developers participation in the development and maintenance of the reusable assets. All code is stored and shared organization-wide in the internal DevOps repository, expect for some sensitive code which is kept private within the team that developed it.

RQ1 – Summary. We characterize the reuse in contemporary SE practices in S-group Solutions AB, which follows both participatory reuse and opportunistic reuse. All reusable assets are managed in the internal DevOps repository. Developers from different teams collaborate in developing and maintaining the internal reusable assets. Developers also retrieve external reusable assets from Open Source community and make upstream contributions for bug fixes or maintain the assets locally.

4.2. RQ2. Software reuse costs and benefits

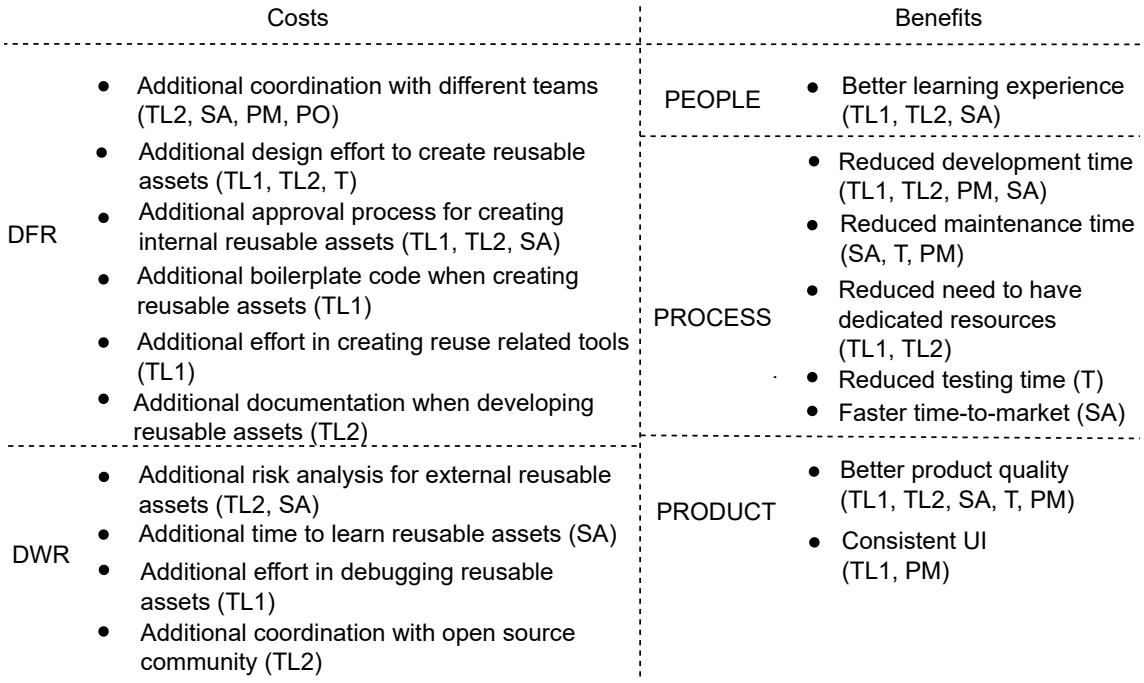

Figure 2. Costs and benefits of the software reuse in the context of contemporary SE practices

Software reuse includes an upfront cost in creating reusable assets, which pays off when reusable assets are integrated in new solutions. Figure 2 provides the classifications of the practitioner perceived costs and benefits, mapping with the participants by the role abbreviations (see Table 2). The listed codes follow the order of their coverage among the participants – from more to less common, reflecting which costs and benefits are considered relevant by different study participants. Sections 4.2.1 and 4.2.2 discuss the reuse costs and benefits perceived by the participants, respectively.

4.2.1. Reuse costs perceived by the participants

This section describes the identified costs related to development for reuse (DFR) and development with reuse (DWR). DFR contains all activities for "creating, acquiring or re-engineering reusable assets", while DWR contains all activities for "using reusable assets in the creation of new software products" [32]. We identified six costs in DFR and four costs in DWR.

In theme DFR, the software reuse costs are as follows:

1. **Additional coordination with different teams** is perceived as a cost by four participants (one of the tech leads, the software architect, the project manager and the product owner). The producers and consumers need additional synchronization and communication to develop or maintain the internal reusable assets, especially in participatory reuse. The software architect described it as "*developing something that fits a few other people, you have to take their needs into consideration and integrate that into the specific product*" and it usually "*ends up in a prioritize discussion*" for the purpose of matching release time as described by the product owner.
2. When creating reusable code assets, **additional design effort to create reusable assets** is needed to make reusable assets easy to use, less error-prone and avoid breaking changes. Such cost is reflected in technical analysis process in the participatory reuse followed in the company (see Figure 1), which was shared by two tech leads and the tester. The additional design effort is not limited to reusable code assets but also reusable automated test cases. One tech lead described it as "*usually you care more about the design of the (reusable) component. But as soon as it is common, you need to design it better so that it is easier for other teams to use as well.*". And the tester shared that creating auto test using page object "*adds small overhead in the short term but will probably be time-saving in the long term.*"
3. **T**hree participants mentioned that additional approval process for creating internal reusable assets is needed before the implementation. The product owner approves the reusable asset functionality, and the project leader approves from the workload and time perspective. One tech lead said reusable microservices need to "*go through, from the product owner, project leader, all of that, before you create the (reusable) microservices*". The additional approval process is part of the solution vision activity in the participatory reuse (see Figure 1).
4. When reusing, some technical problems might constraint the developers from efficient reuse and they need to take additional actions to achieve reuse. One tech lead discussed the cost that developers have to write **additional boilerplate code when creating reusable assets**. He described this cost as "*every time we need to create a new package to address a lot of boilerplate code that we need to implement*". Boilerplate code is code that is copy-and-pasted without modification, e.g., the definitions of getting and setting instance variables method in object-oriented programs.
5. Tools can help in promoting reuse. However, if the developers need to **create the reuse related tools** themselves then it involves **additional effort**. One of the developers added – "*create stuff (reuse tool) that is easy for reuse requires additional effort. However, it can only take a very little time in the long run. It will take some time in the beginning to set everything up and to get it working.*"
6. **Additional documentation when developing reusable assets** is brought up by one tech lead and he described it as "*if you develop some reusable components, you try to add more documentation, describing what component is, so the developers who reuse it will understand.*"

   In theme DWR, the software reuse costs are as follows:

1. One tech lead and the software architect pointed out **additional risk analysis for external reusable assets** in opportunistic reuse. To acquire an external reusable asset, the tech lead said that "*when you have some package candidates, you need to check (if they fit) requirements, you need to do like prototyping and testing*". The software architect added that "*every time we choose to do something like picking a new open-source framework or open-source tool, it has to go through that process*

(risk analysis) where it goes through a few lines within the company to assure that, for example, licenses, agreements actually meets the terms for including this in the product and so on."

2. The software architect highlighted the cost of **additional time to learn reusable assets**, and he said that "*understanding is going to take a bit longer if you are not familiar with that specific project or that specific component.*"
3. One tech lead pointed out the cost of **additional effort in debugging reusable assets**. The debugging process jumps over the reused code and developers have to copy the source code into the project to enable the debugging process. He described this cost as "*it is kind of a little bit of a hassle to debug that, because you need to remove the package and use the actual source code as a reference instead*".
4. **Additional coordination with open source community** is a cost in opportunistic reuse. One of the tech lead explained that "*If it is a new bug (in the external reusable components), then sometimes you need to request for the fix and sometimes it is a problem because you need to wait for such fix or if possible you need to do some workarounds.*"

4.2.2. Reuse benefits perceived by the participants

Although the participants pointed out costs in both DFR and DWR, most of them stated that the benefits outweigh the costs. We identified eight benefits of software reuse in the case company (see Figure 2), which were classified into three themes according to the context facets [33]: people, process and product.

The software reuse benefits the engineers that are involved in the development, integration and maintenance of the reusable assets. In the people theme, we identified **better learning experience** as a reuse benefit. The software architect shared better learning experience as a benefits for the developers that are involved in software reuse. Such benefit is gained from the additional time that developers spend in learning reusable assets. During the learning, the developers will understand what the reusable assets are about and how they were built. A well-designed reusable asset will help developers grasp knowledge faster than development from scratch. The software architect perceives the value in "*getting a much broader understanding of things*" and "*learning much faster than doing it all by yourself*". Such knowledge gaining will also help the company develop better-skilled teams.

In addition to the above benefit for people, the participants also shared the following process related benefits of practicing software reuse:

1. As a result of the additional costs in the DFR, reusing the assets **reduces time in development, maintenance, testing and delivery** (see first four codes for theme PROCESS in Figure 2). The software architect highlighted that software reuse helps "*faster time-to-market*" since developers do not need to develop everything. The project manager, the software architect and two tech leads perceive the main benefit of reusing software is that "*it will save a lot of time instead of we have it (the code) from scratch*", namely, reduced development time. Meanwhile, as a result of reuse, changes or fixes can be propagated easily, which helps reduce the maintenance time. One tech lead said software reuse "*gain (benefits) from a maintainability point of view where you can fix things in one place and reflect all over the entire product*". On the other hand, the tester added the reusable code requires less testing effort and described it as "*when developers reuse stuff, because they reuse something that we know how good quality is, we do not have to spend the same amount of time on testing that specific code once*

again." Overall, the company can benefit from the time-saving perspective and be more competitive in the market.

2. Two tech leads highlighted the benefit of **reduced need for dedicated resources** for consumers because they can rely on the producers of the reusable assets, who have competence in a particular area. One of them described that it is "*a good thing that if it (reusable asset) is more specific area, all the developers do not need to learn such area. So other teams (consumers), they just reuse with such kind of component.*" Teams, even the company could benefit from the reduced resources and further reduce the costs.

Lastly, we also identified the following product related benefits of practicing software reuse:

1. Due to the careful design in producing reusable assets and evolution after several reuses, **better product quality** is discussed by five participants. They said that the software with more reused content has better quality (reduced defects, bugs, deficiencies) as "*such kind of components (reusable components), they are more or less tested. And they contain less bugs than in the components we just developed*". Good product quality could gain reputation for the company and increase the competitiveness.
2. Two participants mentioned benefits in **consistent UI**, however, from different perspectives. One tech lead emphasized the company brand value because of the **consistent UI** and said "*we will share the same, maybe header, sidebars, dashboard, so the users or the customers will recognize our product by whichever application they are using.*" The project manager also mentioned a direct benefit to the customers, i.e., **a consistent UI** leading to a consistent user experience for the customers across different products and modules from the case company, and he described this benefit as "*if you reuse a component that has UI artifacts, it will also look and feel the same and work the same way. You can help create consistency in our UIs.*".

RQ2 – Summary. Costs in DFR result mostly from designing, developing, coordinating for creating reusable assets and their documentation. However, it pays off when developers start to reuse more. Costs in DWR result from learning, analysing, and coordinating for using the reusable assets. The main benefits of software reuse are related to time-saving, better product quality and improved learning experience.

4.3. RQ3. Software reuse challenges and improvements

From interview, the participants also shared the challenges they face in practicing software reuse in the context of contemporary SE practices and the improvements they would like to implement. In total, we identified 14 challenges, which are divided into the following two groups.

– The challenges with improvement suggestions: In this group there are five challenges for which the participants also shared some improvement suggestions (see Section 4.3.1 for details).
– The challenges without improvement suggestions: In this group there are nine challenges, without any specific improvement suggestions by the study participants (see Section 4.3.2 for details).

In addition to the challenges above, we also identified three improvement suggestions (generic improvements) that could not be mapped to any of the challenges, which are described at the end of Section 4.3.1.

In the group discussions, the software architect and project manager prioritized the challenges based on the company's needs in the group discussions. We proposed IS related improvements to address the top concerning challenges, namely discoverability and ownership of reusable assets, knowledge sharing and reuse measurement. The prioritized challenges and IS related improvements that the authors suggested are described in Section 4.3.3.

Figure 3 provides the classifications of the practitioner perceived challenges and improvements, mapping with the participants by the role abbreviations (see Table 2).

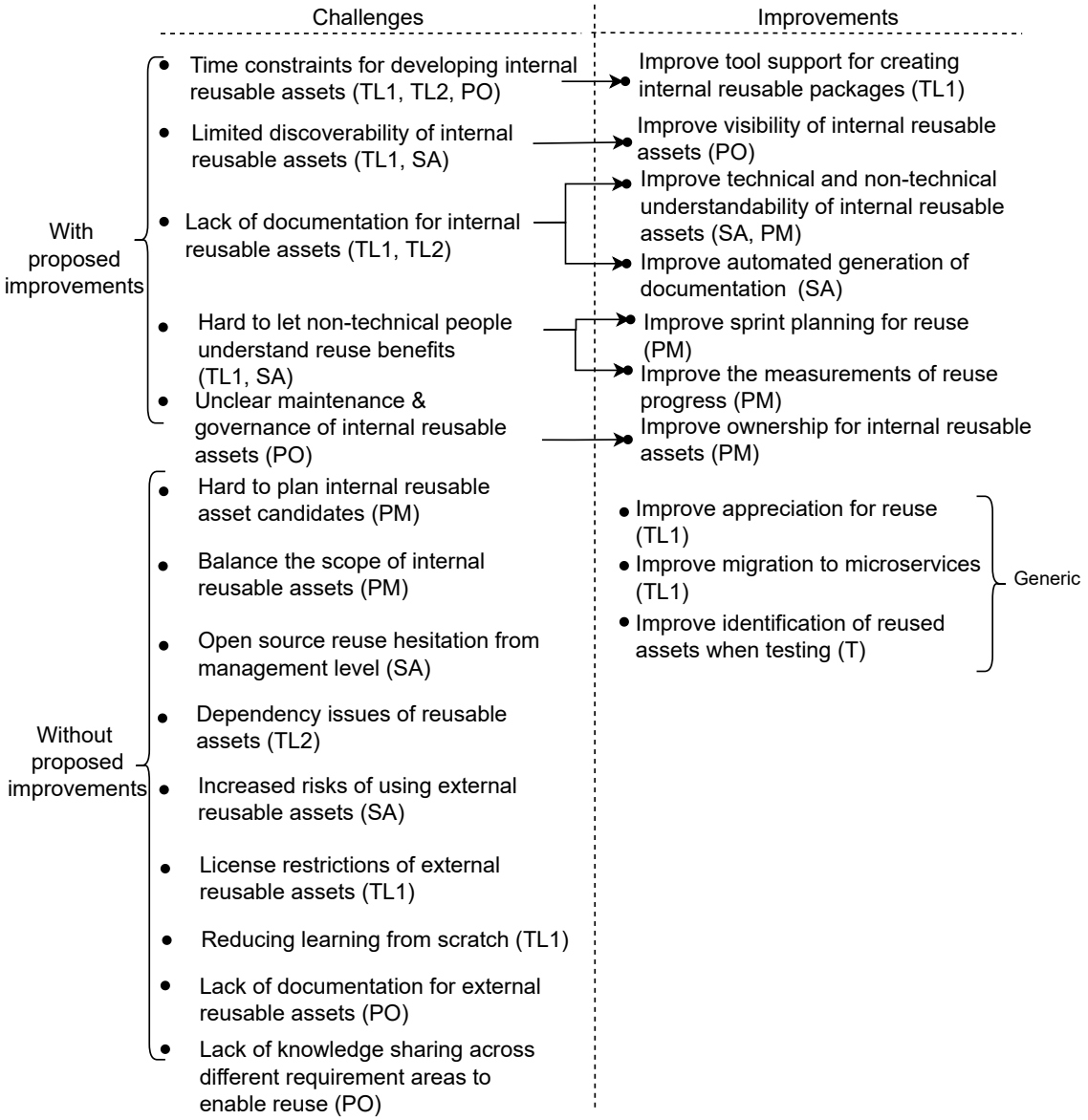

Figure 3. Challenges and improvements of the software reuse in the context of contemporary SE practices

4.3.1. Challenges along with improvement suggestions

We elicited five out of fourteen challenges with participants' proposed improvements. In addition, the participants mentioned three generic improvements which are also described in this section. The challenges with proposed improvements are as follows:

1. **Time constraints for developing internal reusable asset** is a big concern raised by two tech leads and the product owner. Tight release schedule is always a constraint for software product delivery in the industry. Adding DFR into an existing development life cycle is even more demanding. One of the tech leads gave an example that "*we have a notification system, which were very easily could be made to a microservice. But for now, since we are close to release, we will keep it in our application for now, but it will probably become a requirement and as new application or service of itself in the future*".

   Improvement suggestion – Improve managing time constraints

   Improving *tool support* for creating internal reusable packages is necessary to manage the time constraints for developing reusable assets. One of the tech lead suggested supporting tools for creating reusable packages – "*to have a lot of tools that do things for us or to create packages more easily*". To enable development of reusable assets while ensuring timely delivery, the project manager wanted to **improve *sprint planning* for reuse** and suggested the following: "*a good approach from my perspective is to have the developers give me two estimates, one where we do not develop it as a reusable component, and one where we do. So I can discuss reuse priorities with the product owner. If the product owner does not agree to prioritize reuse tasks, then my suggestion is to implement it for that specific area, but then we are allocated time afterwards for converting it to a reusable component.*"

2. Two participants perceived **limited discoverability of internal reusable assets** as a challenge. Reuse will not happen if the developers cannot find the available reusable assets. One of the tech lead and the software architect identified the challenges that people in the company are not able to discover all the existing reused assets within the company. The software architect emphasized "*it (the challenge) is the discoverability of the things for developers to know what actually exists internally*".

   Improvement suggestion – Improve visibility of internal reusable assets

   To improve the discoverability challenge, the product owner wanted to **improve *visibility* of internal reusable assets** by managing and grouping the similar user stories representing a workflow. The right categorization helps to improve visibility and hence facilitate software reuse. He added "*we probably have thousands of user stories, but to be able to get that in a manageable way, you probably need to step up a little bit, maybe on workflow level.*"

3. **Lack of documentation for internal reusable assets** was mentioned by two tech leads. One of the tech lead said "*usually they (internal packages) do not have good*

documentation". And low quality of documentation or missing documentation, may hinder understandability of internal reusable assets.

Improvement suggestion – Improve documentation for internal reusable assets

The participants identified concrete improvement suggestions to improve documentation. The software architect wanted to **improve *automated generation* of documentation** since they considered manually writing the documentation for reusable assets is as an overhead and suggested automating this process: "*since we want to spare the developers from writing too much documentation, we are looking into how to automate it entirely.*"
The software architect and project manager both suggested on how to **improve *technical and non-technical* understandability of internal reusable assets**. The software architect suggested that they need to improve documentation of reusable assets in a way that helps developers understand the capability of the reusable assets and "*how to use the package (reusable assets)*". The project manager suggested that they need to improve documentation so that "*other people within the company know what is available for reuse than only the developers.*"

4. One of the tech leads and the software architect found it was **hard to let non-technical people understand reuse benefits** which led to a lack of management support. The software architect brought up that "*sometimes it is a challenge to get them (non-technical people) to understand what is the actual benefit for it (software reuse)*".

Improvement suggestion – Improve management's perception on reuse benefit

The project manager wanted to **improve the *measurement* of reuse progress** to demonstrate the reuse rate to the management. He said that "*when we develop a new web application, I want to be able to see that in this new web application, how much did we reuse. So basically how much of the code base that's in this web application is from reuse. And that could be one version of kind of measuring how much implementation time is saved.*"

5. The product owner pointed out the challenge of **unclear maintenance/governance of internal reusable assets**. He raised the following question – "*when we have written it (the reusable assets), who should maintain it (the reusable assets)*".

Improvement suggestion – Improve maintenance/governance of internal reusable assets

The project manager identified the need to **improve *ownership* for internal reusable assets** and suggested that they need "*a clear strategy of who is responsible and who owns this (reusable) component*".

Improvement suggestion – Generic improvements

Improvements in reward are mentioned by one of the tech leads to scale reuse instead of providing suggestions to mitigate the listed challenges. It is important to **improve *appreciation* for reuse**, which motivates the developers to not only produce but also consume reusable assets. One tech lead mentioned "*not incentive or maybe like said appreciation that we make time to create something that will save time later on.*"

The tester wanted to **improve *identification of reused assets* when testing**. He suggested enhancing the traceability of reused code in the system under test – "*I think from my (tester) point of view, one area for improvement is to clarify when we reuse stuff, because it is not always very clear*".

One of the tech lead identified a need in having a clear vision to **improve *migration* to microservices**, he described it as "*from the architecture point of view, what components, to plan, extract (for migration)*".

4.3.2. Challenges without proposed improvements

In this section we discuss nine out of fourteen challenges that participants brought up without any associated improvement suggestions.

1. The project manager found it is **hard to plan internal reusable assets candidates**. He added that: "*identify is this functionality that should be implemented as a reusable component, and taking that decision, that is hard to get in black or white. And usually you have to move down some grey area to kind of take that decision.*"
2. After identifying the reusable asset candidates, practitioners are faced with a dilemma of the scope of the reusable assets. The project manager found it challenging to balance how general and specific the reusable assets should be, namely **balance the scope of internal reusable assets**. He got this input from developers that "*they (developers) find very hard when it comes to reusable components, to find the right level of how generic the component should be*".
3. The software architect identified **open source reuse hesitation from management level**. He said "*people that started as a developer and he now have another role in higher-up management*" are hesitant towards open-source reuse. He added that "*mentioning open-source to a person who worked with proprietary systems and closed was not really easy. And open-source is misunderstood in many ways, I would say.*" The management can hinder opportunistic reuse if they are not willing to take in the open source software.
4. When performing reuse, one tech lead identified **dependency issues of reusable assets**. Many dependencies need to be taken care of when reusing the package and this dependency overhead creates lots of work for developers and is not good for users as well. He added: "*some (reusable) components are good for us, but it has a lot of dependencies.*"
5. The software architect highlighted the **increased risks of using external reusable assets**. The company relies on the quality of the reused external assets. And he added that "*it is a bit more risky to include things from the outside*".

6. **License restrictions of external reusable assets** may prevent the reuse of external assets. One of the tech leads pointed that sometimes they "*cannot reuse because it depends on licensing*".
7. With increased software reuse, one of the tech lead highlighted an issue in **reducing learning from scratch**. He described that "*if we keep reusing stuff and not code anything ourselves, that might be an issue. Get experience that way, to do things from scratch as well.*" However, the software architect viewed learning benefits in terms of understanding and knowledge gained from reusing as described in Section 4.2.2.
8. The product owner and one tech lead raised a concern in **lack of documentation for external reusable assets**. The product owner stated that developers also need to know what exists externally, and what can be brought into the company. Incomplete documentation in external reusable assets hinders the opportunistic reuse practice: "*to find and also to see can it (reusable asset) reports in that way that we want to utilize it and maybe incorporate it in our product as the way it is or something else with the license agreement. That is hard to find it on that level.*" And the tech lead also said sometimes "*the real read-me files have none*" in external packages.
9. **Lack of knowledge sharing across different requirement areas to enable reuse** indicates limited transparency. The product owner explained that this knowledge sharing problem occurred because different teams work on their specific requirement areas and they lacked central communication for sharing. However, he emphasized that "*when every part is developed, it is very important that we need to share knowledge, so several people know about this functionality.*"

4.3.3. Prioritized challenges along with improvement suggestions -

Although practitioners perceived some challenges with software reuse, they considered reuse to be important and wanted to invest in further improving the software reuse process in the company. We presented the overall interview findings and asked the software architect to prioritize the challenges and improvements they would like to implement. Based on the company's requirements and internal discussions with the relevant stakeholders, the software architect prioritized discoverability and ownership of reusable assets, knowledge sharing and reuse measurement as the focus areas for further investigation. We identified some IS patterns from the InnerSource Commons[7] that could address the top challenges and improvement areas. The IS patterns that we discussed with the product manager and the software architect are discussed below:

1. **Discoverability of the reusable assets.** *InnerSource Portal* pattern[8] aims to create an intranet portal that allows the project owners to advertise their projects to the entire organization. Though the case company does not have shared IS projects, they can use the portal to find all the reusable assets in an efficient way.
2. **Ownership of the reusable assets.** The participants emphasized there is a need to have a clear strategy about the ownership. To complement the participant's suggestion, we proposed two InnerSource patterns – *30 Days Warranty* pattern[9] and *Trusted Committer* pattern[10], to address the maintenance/governance challenge. *Thirty Days Warranty* pattern assigns the contributors the responsibility to pass the knowledge and

[7] https://innersourcecommons.org/
[8] https://patterns.innersourcecommons.org/p/innersource-portal
[9] https://patterns.innersourcecommons.org/p/30-day-warranty
[10] https://patterns.innersourcecommons.org/p/trusted-committer

solve the problems about their contributions within a certain period. It creates a buffer time for the one responsible for the reusable assets to understand the contributed code and gain the ability to maintain them. *Trusted Committer* pattern aims to assign a trusted committer role to the most active contributors and allocate bandwidth to facilitate the maintenance/governance of the reusable assets.

3. **Knowledge sharing of the reuse related information**. To facilitate knowledge sharing, we suggested to improve work and decision transparency in the group discussions, namely, 1) ask all teams to publish their roadmaps and backlog planning, 2) publish decisions and allow for discussions from other teams. Such suggestions were generated from IS pattern – *Transparent Cross-team Decision Making Using RFCs*[11], which helps increase the chance of other teams' participation by publishing internal requests for comments (RFCs) documents.
4. **Reuse measurement**. We suggested *Cross-team Project Valuation* pattern[12] to further address the reuse measurement improvement. Such a pattern aims to create a model to calculate the value of cross-team projects (in our case, the shared reusable assets) and demonstrate the increased productivity when people from other teams are also involved in development or maintenance.

We discussed the feasibility of the above proposed IS patterns with the product manager and the software architect. In the discussions, we concluded that many patterns from InnerSource Common[13] could help address the top concerning challenges. Moreover, the project manager agreed to conduct a follow-up investigation to check the company's readiness for adopting more InnerSource practice to improve the development and maintenance of the reusable assets.

RQ3 – Summary. In the group discussions, S-Group Solutions AB rated discoverability and ownership of the reusable assets, knowledge sharing of the reuse related information and reuse measurement as the major improvement areas of the software reuse practice. Apart from the improvements proposed by the participants, IS patterns help address a lot of software reuse challenges.

5. Discussion

This section further discusses and compares our results in software reuse costs, benefits, challenges, and improvements with the related works.

5.1. Software reuse costs in the context of contemporary SE practices

In our study, we identified that practicing software reuse results in additional costs – more in case of development for reuse as compared to development with reuse. Our findings are in line with the results reported by Kruger and Berger [20] and Agresti [24]. We found additional coordination in participatory reuse as we anticipated. Additional coordination is needed in the case of opportunistic reuse as well. Kruger and Berger [20] also found that practicing software reuse results in additional synchronization and coordination among different teams. They found additional coordination when handing over reusable assets

[11] https://patterns.innersourcecommons.org/p/transparent-cross-team-decision-making-using-rfcs
[12] https://patterns.innersourcecommons.org/p/crossteam-project-valuation
[13] https://innersourcecommons.org/

to different functional teams, such as development teams and quality assurance teams. In comparison, our identified additional coordination occurred when other consumer teams wanted to participate and contribute to developing the reusable assets. It could also be argued that additional coordination helps increase the transparency between different teams and enhance the internal collaboration.

Our study and Agresti [24] found extra costs in understanding the reusable code. However, Agresti [24] identified extra cost when the reused assets need extensive modification. Our participants did not bring up such a cost. Comparing Agresti's study [24] with our study, we think the reason could be that our case company follows a relatively more systematic process when performing software reuse, such as using the solution vision process and the technical analysis before developing the reusable assets (see Figure 1). Moreover, our case company did not mention additional integration effort in opportunistic reuse as we assumed.

5.2. Software reuse benefits in the context of contemporary SE practices

We identified better product quality and time-saving in development, maintenance, testing and delivery as the main software reuse benefits in the case company. Multiple secondary studies (cf. [1–3]) and primary studies [18, 20, 21, 24, 25, 34] on software reuse also identified that the software reuse practices contribute to better product quality and time saving in one or more phases of the software development cycle.

Bauer et al. [25] and our study found that software reuse helps in improving the consistency of the product. However, in our study, software reuse is found to contribute to consistent user interface experience across different modules, while in Bauer's et al. [25] study, software reuse contributes to feature consistency over the range of products. Literature related to internal reuse [21, 34] and our study found the learning benefit in software reuse, however, from different perspectives. We identified that internal reuse practice also offers some learning opportunities to the developers – they could learn more from understanding and reusing well-designed reusable assets. Goldin et al. [34] also found that requirement management and reuse help the new employees complete the onboarding process easier and quicker from the learning perspective.

Barros-Justo et al. [18] identified higher documentation quality as a benefit of software reuse. We did not find higher documentation quality as a benefit of software reuse in our study. However, the participants pointed out that reusable assets require additional documentation (for details, see Section 4.2) as we anticipated. This upfront cost may contribute to higher document quality later. There maybe two reasons for not having higher documentation quality due to the reuse practices in the case company. First, the case company is still at the beginning (about two years) of their software reuse journey. They need more time to adapt to the reuse approaches. Second, in a medium-sized company, it is difficult to invest extra resources to create additional documentation.

The participants also brought up that due to the availability of the reusable assets, the consumer teams do not need to dedicate resources in those domain areas that are already covered by the reusable assets. However, with this benefit placed, the consumers may take things for granted and start to ask for more features in the reusable assets. Riehle et al. [35] reported a similar scenario wherein the producer teams were over-burdened due to the large number of change requests from the consumers of the reusable common assets.

5.3. Software reuse challenges in the context of contemporary SE practices

In our study, the participants were positive about having internal reusable assets. However, they also pointed out some challenges related to the management of the internal reusable assets, including discoverability, knowledge sharing and the ownership of the internal reusable assets. Barros-Justo et al. [18] and Bauer and Vetro [19] also reported that finding the relevant reusable asset is a common problem. Due to the boundaries between projects, reusable assets become unavailable for the developers across projects [19] – such a way of working potentially constraints software reuse and it represents the same challenge that we also identified in our case company – namely, lack of knowledge sharing across different teams to enable reuse. According to our study, Bauer and Vetro [19], and Barros-Justo et al. [18], practitioners rely on the repository search and communication with their colleagues as the main methods for finding the relevant internal reusable assets.

The question of who will own and maintain the reusable assets in participatory reuse is important. In our study, the participants brought up this question as an important challenge to deal with as we anticipated. Kruger and Berger [20] also identified the challenges in coordinating in and between teams, especially when the responsibilities are not clear.

Some participants in our study also shared that it is hard to explain the benefits of practicing software reuse to the non-technical persons (e.g., senior management), which may hinder the organization-wide adoption of software reuse. Morisio et al. [36] and Kolb et al. [37] also found that the senior management support is essential for promoting the software reuse process to the entire organization.

In our study, we found it is difficult to define the scope of the reusable assets at the initial stage. Kolb et al. [37] also shared a similar finding, however in their case, the challenge was about adding new features to an existing reusable component.

As discussed previously, software reuse practices offer learning opportunities to the developers. However, interestingly some participants cautioned that too much reliance on reuse may have a negative impact on the capability of the developers to write own code. Bauer et al. [25] also discussed the challenge of trying to strike a balance between acceptable level of reuse and excessive reuse.

Our study identified that the reuse of packages and components may lead to additional dependencies that need to be taken care of. Bauer et al. [25] identified dependency explosion was the major issue for Google in software reuse, especially the ripple effects caused by changes in reused code assets. Barros-Justo et al. [18] also noted dependency issues when reusable assets are integrated into the new solutions. We suggest practitioners could adopt and follow the practices proposed by Gustavsson [38] for managing the open source dependencies, e.g., establishing a forum for conscious decisions on open source dependencies, maintaining a dependency list and scanning for security issues. For opportunistic reuse, dealing with license restrictions was also shared as a challenge by the participants in our study, which is in line with some related works [18, 19].

5.4. Software reuse related improvements in the context of contemporary SE practices

First, we discuss those improvements that the case company has already implemented as a result of this study. The improvements are aimed at improving the development, integration and documentation of the reusable assets:

1. Additional boilerplate code: To remove the need to write additional boilerplate code while developing a reusable package, the company has developed a mechanism to create a template that includes all startup code required for initiating the development of a reusable package.
2. Additional effort in debugging: In cases when the bugs are related to the reused shared packages, the participants shared that they need to spend some additional time on debugging as they need to copy the code of the reusable package to a new project to perform the debugging. The company has now developed the support to address this issue.
3. Lack of documentation: Lack of documentation for reusable packages was identified as one of the challenges. Some documentation for reusable packages is now automated, thus saving the time and effort spent on manually creating the reusable package documentation.

In addition to the three implemented improvements discussed above, we also agreed to investigate the feasibility of adopting more IS practices and patterns to improve the development, maintenance and governance of the reusable assets in software reuse.

In the case company, the reusable assets are maintained in a repository with some keyword searching options. We suggested the case company to adopt the *InnerSource Portal* pattern to enhance the discoverability of reusable assets. For the same discoverability purpose, Agresti et al. [24] suggested cleaning up the reusable code library, setting criteria to qualify the reusable code, finding a manager to look after the library, and having an online keyword-search capability across different sources. Moreover, Bauer et al. [25] suggested that the reusable assets should be listed in the marketplace and the reusable assets from different libraries should be merged to avoid the duplicates. The similarity of the discoverability improvements among our case company, related InnerSource pattern and the discussed related works [24, 25] is that we all focused on the management and the search facility of the reusable assets.

The ownership of reusable assets affects the developers and the project managers. However, we did not find ownership related improvements in the selected related works. The patterns – *30 Day Warranty* and *Trusted Committer* pattern that we introduced to the company, could help in solving the ownership issues, reducing the effort in locating people, and synchronizing the meeting schedules and release plans [4]. As for the reuse measurements, the case company started using the reuse rate to track the percentage of the reused code assets. We also suggested *Cross-team Project Valuation* pattern to the company. Mohagheghi and Conradi [1] conducted a literature review, investigating the metrics about software reuse quality, productivity and economic benefits. They aggregated and categorized different metrics from 11 studies from 1994 to 2005. We argue that there is a need to extend such a literature review since the reusable assets (e.g., microservices) and the reuse type (e.g., participatory reuse) have evolved since 2005.

Compared to our study, Agresti et al. [24] also provided suggestions for improving the understandability of reusable assets, such as better comments in the code, better structured software modules and a written reuse guidebook. However, they did not mention the need for non-technical people, e.g., managers to understand the value of reuse.

The development and maintenance of reusable assets also have budgetary implications. Like our study, Agresti et al. [24] also discussed the need for improvements in resource planning to facilitate developers that are working on the reusable assets in addition to other tasks. They [24] suggested allocating additional budget for the developers to facilitate them for contributing to the reusable assets.

5.5. Threats to validity

We discuss threats to validity in two phases using the validity threats categorization proposed by Peterson and Gencel [39]: (1) study design and data collection, and (2) data analysis.

5.5.1. Study design and data collection

Theoretical validity. The theoretical validity is concerned with construct definition, evaluation comprehension and the selection of subjects. We decided the study objective based on the company's needs through a joint discussion with the company contact person. The interview guide is developed and reviewed iteratively among authors, and a pilot semi-structured interview is performed before the actual interviews to evaluate the interview questions' comprehension. As for the recruitment strategy, we provided the reuse related role descriptions to help the contact person identify the relevant people for the interview. The sample size is small, but we managed to cover at least 20% of the population, all teams, and related roles. The sample size of the group discussions is small and the participants are from the interviews. However, the selected two participants are the most relevant and experienced people in software reuse practice in the company. In addition, we asked the two participants to gather opinions from their colleagues and prepare documentation before they came to the discussions.
Descriptive validity. The descriptive validity is concerned with factual accuracy. We transcribed the interviews word to word and tried to use the actual text segments to describe the results as much as possible. Moreover, we presented the preliminary study results and shared the study report to the software architect. And he confirmed that the results captured the reality.

5.5.2. Data analysis

Interpretative validity. The interpretive validity is concerned with capturing the relevant information and researchers' bias in interpretation. We transcribed the interviews word to word to avoid misinterpretations. We followed the Cruzes's and Dybå's [29] recommended steps of thematic analysis to analyze the transcripts. The first and second authors independently analyzed and generated the code to confirm the results. The third author validated the data credibility as mentioned in Section 3.5, which resulted in some minor changes regarding code names and code descriptions. We also presented the results to the company to eliminate misinterpretation.
Generalizability. The generalizability is concerned with the context information which influences the study transferability. Our focus is medium-sized companies and we introduced the company context information in detail (see Section 3.3 and Section 4.1), so that other relevant companies could relate our case to their context and get some useful insights. Furthermore, the detailed context information helps the researchers to include the details when reporting findings on software reuse in contemporary SE practices.

6. Conclusion and future work

In this paper, we reported the results of an exploratory case study on software reuse practice in the context of contemporary SE practices conducted in a medium-sized company. The reported study covers the software reuse process, costs, benefits, challenges and improvements. We obtained the data from six semi-structured interviews, four group discussions and relevant documentation, followed by a rigorous process to analyze the collected data.

The study elaborates how the case company is practicing software reuse, including participatory reuse and opportunistic reuse. Participatory reuse is an organizational-wide reuse collaboration between the producers and consumers of the reusable assets, while opportunistic reuse relates to the reuse of external assets from open source communities or other third parties. The results show that the software reuse costs mainly relates to the development of the reusable assets, their documenting and the time spent in additional coordination between the teams working on the common reusable assets. In our study, the participants perceived that the benefits of software reuse outweigh the associated costs, thus were in favor of further improving the software reuse practices. Software reuse benefits many stakeholders in terms of people, process and products. The main perceived benefits are related to time-saving and product quality, which are highly aligned with the investigated related works. The study participants were aware of the software reuse challenges and suggested some concrete improvements. According to the interviews and group discussions results, discoverability and ownership of the reusable assets, knowledge sharing and reuse measurement are the top concerning challenges and improvements for the case company, which have a great potential to be addressed by certain InnerSource patterns and practices.

The case company is interested in adopting InnerSource patterns and practices to systematize the software reuse process. We are planning a follow up investigation at the case company to ascertain the company's readiness for adopting InnerSource practices for improving the development and maintenance of the reusable assets. With the help of the relevant stakeholders, the idea is to assess the application of the proposed improvements in terms of costs and importance and select specific InnerSource practices for implementation in the case company. In the long term, we are interested in evaluating the effectiveness of the adopted practices for improving the state of software reuse in the case company.

Acknowledgements

We would like to acknowledge that this work was supported by the Knowledge Foundation through the OSIR project (reference number 20190081) at Blekinge Institute of Technology, Sweden. We would also like to thank the practitioners from the case company for collaborating with us. Lastly, we would like to thank Prof. Claes Wohlin for participating in the initial discussion with the case company and providing valuable feedback on the study design and the initial version of the study report.

References

[1] P. Mohagheghi and R. Conradi, "Quality, productivity and economic benefits of software reuse: A review of industrial studies," *Empirical Software Engineering*, Vol. 12, No. 5, 2007, pp. 471–516.

[2] J.L. Barros-Justo, F. Pinciroli, S. Matalonga, and N. Martínez-Araujo, "What software reuse benefits have been transferred to the industry? A systematic mapping study," *Information and Software Technology*, Vol. 103, 2018, pp. 1–21.

[3] D. Bombonatti, M. Goulão, and A. Moreira, "Synergies and tradeoffs in software reuse – A systematic mapping study," *Software: Practice and Experience*, Vol. 47, No. 7, 2017, pp. 943–957.

[4] D. Cooper and K.J. Stol, *Adopting InnerSource*. O'Reilly Media, Incorporated, 2018.

[5] J.L. Barros-Justo, F.B. Benitti, and S. Matalonga, "Trends in software reuse research: A tertiary study," *Computer Standards and Interfaces*, Vol. 66, 2019, p. 103352.

[6] R. Capilla, B. Gallina, C. Cetina, and J. Favaro, "Opportunities for software reuse in an uncertain world: From past to emerging trends," *Journal of Software: Evolution and Process*, Vol. 31, No. 8, 2019, p. e2217.

[7] T. Mikkonen and A. Taivalsaari, "Software reuse in the era of opportunistic design," *IEEE Software*, Vol. 36, No. 3, 2019, pp. 105–111.

[8] B. Xu, L. An, F. Thung, F. Khomh, and D. Lo, "Why reinventing the wheels? An empirical study on library reuse and re-implementation," *Empirical Software Engineering*, Vol. 25, No. 1, 2020, pp. 755–789.

[9] R. Capilla, T. Mikkonen, C. Carrillo, F.A. Fontana, I. Pigazzini et al., "Impact of opportunistic reuse practices to technical debt," in *IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, 2021, pp. 16–25.

[10] M. Höst, K.J. Stol, and A. Oručević-Alagić, "Inner source project management," in *Software Project Management in a Changing World*. Springer, 2014, pp. 343–369.

[11] S. Fox, "IBM internal open source bazaar." *Presentation at the IBM Linux Technology Center in November 2007*.

[12] P. Vitharana, J. King, and H.S. Chapman, "Impact of internal open source development on reuse: Participatory reuse in action," *Journal of Management Information Systems*, Vol. 27, No. 2, 2010, pp. 277–304.

[13] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," *IEEE Software*, Vol. 33, No. 3, 2016, pp. 42–52.

[14] P. Jamshidi, C. Pahl, N.C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, Vol. 35, No. 3, 2018, pp. 24–35.

[15] J. Soldani, D.A. Tamburri, and W.J. Van Den Heuvel, "The pains and gains of microservices: A systematic grey literature review," *Journal of Systems and Software*, Vol. 146, 2018, pp. 215–232.

[16] J.P. Gouigoux and D. Tamzalit, "From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture," in *IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 62–65.

[17] M. Capraro and D. Riehle, "Inner source definition, benefits, and challenges," *ACM Computing Surveys (CSUR)*, Vol. 49, No. 4, 2016, pp. 1–36.

[18] J.L. Barros-Justo, D.N. Olivieri, and F. Pinciroli, "An exploratory study of the standard reuse practice in a medium sized software development firm," *Computer Standards and Interfaces*, Vol. 61, 2019, pp. 137–146.

[19] V. Bauer and A. Vetro, "Comparing reuse practices in two large software-producing companies," *Journal of Systems and Software*, Vol. 117, 2016, pp. 545–582.

[20] J. Krüger and T. Berger, "An empirical analysis of the costs of clone-and platform-oriented software reuse," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 432–444.

[21] O.P.N. Slyngstad, A. Gupta, R. Conradi, P. Mohagheghi, H. Rønneberg et al., "An empirical study of developers views on software reuse in statoil asa," in *Proceedings of the ACM/IEEE international symposium on empirical software engineering*, 2006, pp. 242–251.
[22] S.A. Ajila and D. Wu, "Empirical study of the effects of open source adoption on software development economics," *Journal of Systems and Software*, Vol. 80, No. 9, 2007, pp. 1517–1529.
[23] N. Mäkitalo, A. Taivalsaari, A. Kiviluoto, T. Mikkonen, and R. Capilla, "On opportunistic software reuse," *Computing*, Vol. 102, No. 11, 2020, pp. 2385–2408.
[24] W.W. Agresti, "Software reuse: developers' experiences and perceptions," *Journal of Software Engineering and Applications*, Vol. 4, No. 01, 2011, p. 48.
[25] V. Bauer, J. Eckhardt, B. Hauptmann, and M. Klimek, "An exploratory study on reuse at Google," in *Proceedings of the 1st international workshop on software engineering research and industrial practices*, 2014, pp. 14–23.
[26] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, Vol. 14, No. 2, 2009, pp. 131–164.
[27] E.C. (2003), *Commission Recommendation of 6 May 2003 concerning the definition of micro, small and medium-sized enterprises, C (2003) 1422*. [Online]. http://data.europa.eu/eli/reco/2003/361/oj.[Accessed:Apr.16,2021]
[28] C.B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, Vol. 25, No. 4, 1999, pp. 557–572.
[29] D.S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *International Symposium on Empirical Software Engineering and Measurement*, 2011, pp. 275–284.
[30] K. Petersen and C. Wohlin, "A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case," *Journal of Systems and Software*, Vol. 82, No. 9, 2009, pp. 1479–1490.
[31] J. Saldaña, *The coding manual for qualitative researchers*. SAGE Publications Limited, 2021.
[32] *IEEE Standard for Information Technology – System and Software Life Cycle Processes – Reuse Processes*, IEEE Std. 1517–2010, Aug. 2010.
[33] K. Petersen and C. Wohlin, "Context in industrial software engineering research," in *3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2009, pp. 401–404.
[34] L. Goldin and D.M. Berry, "Reuse of requirements reduced time to market at one industrial shop: A case study," *Requirements Engineering*, Vol. 20, No. 1, 2015, pp. 23–44.
[35] D. Riehle, M. Capraro, D. Kips, and L. Horn, "Inner source in platform-based product engineering," *IEEE Transactions on Software Engineering*, Vol. 42, No. 12, 2016, pp. 1162–1177.
[36] M. Morisio, M. Ezran, and C. Tully, "Success and failure factors in software reuse," *IEEE Transactions on Software Engineering*, Vol. 28, No. 4, 2002, pp. 340–357.
[37] R. Kolb, I. John, J. Knodel, D. Muthig, U. Haury et al., "Experiences with product line development of embedded systems at testo ag," in *10th International Software Product Line Conference (SPLC'06)*. IEEE, 2006, pp. 10–pp.
[38] T. Gustavsson, "Managing the open source dependency," *Computer*, Vol. 53, No. 2, 2020, pp. 83–87.
[39] K. Petersen and C. Gencel, "Worldviews, research methods, and their relationship to validity in empirical software engineering research," in *2013 joint conference of the 23rd international workshop on software measurement and the 8th international conference on software process and product measurement*. IEEE, 2013, pp. 81–89.