Improving Internal Software Reuse in the Context of Contemporary Software Engineering Practices

Xingru Chen

Department of Software Engineering
Blekinge Institute of Technology

Licentiate Dissertation Series no. 2023:09
Improving Internal Software Reuse in the Context of Contemporary Software Engineering Practices

Xingru Chen
Improving Internal Software Reuse in the Context of Contemporary Software Engineering Practices

Xingru Chen

Licentiate Dissertation in Software Engineering

Department of Software Engineering
Blekinge Institute of Technology
SWEDEN
“It does not matter how slowly you go as long as you do not stop.”
- Confucius
Abstract

Context: Companies adopt many software reuse practices, such as software product line, reuse verbatim, and systematic reuse, to improve their internal software development and maximize the benefits. Contemporary software engineering (SE) practices, such as microservices and InnerSource, influence internal software reuse.

Objective: In this thesis, we aim to improve internal software reuse in the context of contemporary SE practices. To do that, we want to 1) understand the state-of-the-art and the state-of-the-practice of software reuse costs and benefits and the challenges that companies are currently facing and 2) identify interventions to improve internal software reuse.

Methods: We conducted a systematic literature review to understand the state-of-the-art of software reuse costs and benefits. We performed two exploratory case studies to understand the state-of-the-practice of software reuse costs and benefits, challenges, and improvement areas in the context of contemporary SE practices. We performed another follow-up improving case study to investigate the medium-sized case company’s readiness of adopting InnerSource for software reuse.

Results: Existing literature reported more software reuse benefits than costs. The most reported software reuse benefits are better product quality and improved productivity. Verbatim reuse and systematic reuse result in more reuse benefits. Most of the included primary studies are of moderate quality, with only four having high quality. Practitioners think that software reuse costs in developing reusable assets will be paid off when developers start to reuse them. Challenges in software reuse in the context of contemporary SE practices differ between medium-sized and large-sized companies. Both of the companies perceive that InnerSource can help improve internal software reuse. Asking practitioners about both current and desired InnerSource reuse status helps
identify the needed InnerSource improvements, thus helping companies succeed in adopting InnerSource for reuse.

Conclusion: Both existing literature and our two case studies investigating software reuse in the context of contemporary SE practices showed that software reuse improves quality and productivity and has costs in developing and integrating reusable assets. However, the overall benefits outweigh the costs. Both case companies faced challenges in improving their internal reuse, where the most common challenges were about developing and maintaining reusable assets. The results showed that InnerSource helps develop and maintain reusable assets and further improves internal software reuse.

**Keywords:** *Software reuse, Costs and benefits, Contemporary software engineering practices*
Acknowledgements

I would like to express my sincere gratitude to my supervisors, Deepika Badampudi and Muhammad Usman, for their unwavering support, patience, and invaluable guidance throughout the journey of completing this thesis. Their expertise and dedication have been instrumental in shaping this work.

I am deeply grateful for the time and effort both Deepika and Muhammad have invested in me, allowing me to grow not only as a researcher but also as an individual. Their guidance has been invaluable, and I am fortunate to have had the privilege to work under their supervision.

I would also like to thank my family and friends for their support and encouragement throughout this endeavor. This thesis would not have been possible without their love and understanding.
Overview of Papers

Papers in this Thesis

- **Chapter 2: Xingru Chen**, Muhammad Usman, and Deepika Badampudi. “Understanding and Evaluating Software Reuse Costs and Benefits from Industrial Cases - A Systematic Literature Review” To be submitted to a software engineering journal.


Contribution Statement

We use the Contributor Roles Taxonomy (CRediT) [70] to describe the author contributions for the included papers.
Chapter 2

**Xingru Chen:** Conceptualization (supporting), data curation (lead), formal analysis (lead), investigation (lead), methodology (lead), visualization (lead), writing - original draft (lead), writing - review & editing (supporting)

**Muhammad Usman:** Conceptualization (lead), data curation (supporting), formal analysis (supporting), investigation (supporting), methodology (supporting), supervision (lead), visualization (supporting), writing - original draft (supporting), writing - review & editing (lead)

**Deepika Badampudi:** Conceptualization (supporting), data curation (supporting), formal analysis (supporting), investigation (supporting), methodology (supporting), visualization (supporting), writing - review & editing (supporting)

Chapter 3

**Xingru Chen:** Conceptualization (supporting), data curation (equal), formal analysis (equal), investigation (equal), methodology (equal), project administration (equal), visualization (lead), writing - original draft (lead), writing - review & editing (supporting)

**Deepika Badampudi:** Conceptualization (lead), data curation (equal), formal analysis (equal), funding acquisition (equal), investigation (equal), methodology (equal), project administration (equal), resources (equal), supervision (lead), visualization (supporting), writing - original draft (supporting), writing - review & editing (lead)

**Muhammad Usman:** Conceptualization (lead), data curation (equal), formal analysis (supporting), funding acquisition (equal), investigation (equal), methodology (equal), project administration (equal), resources (equal), supervision (supporting), visualization (supporting), writing - review & editing (supporting)

Chapter 4

**Deepika Badampudi:** Conceptualization (lead), data curation (equal), formal analysis (equal), funding acquisition (equal), investigation (equal), methodology (equal), project administration (equal), resources (equal), supervision (equal), visualization (lead), writing - original draft (lead), writing - review & editing (supporting)

**Muhammad Usman:** Conceptualization (lead), data curation (equal), formal analysis (equal), funding acquisition (equal), investigation (equal), method-
ology (equal), project administration (equal), resources (equal), supervision (equal), visualization (supporting), writing - original draft (supporting), writing - review & editing (lead)

**Xingru Chen:** Conceptualization (supporting), data curation (supporting), investigation (supporting), methodology (supporting), project administration (supporting), writing - original draft (supporting), writing - review & editing (supporting)

**Muhammad Usman:** Conceptualization (lead), data curation (supporting), formal analysis (equal), funding acquisition (equal), investigation (equal), methodology (equal), project administration (supporting), supervision (lead), visualization (supporting), writing - original draft (supporting), writing - review & editing (supporting)

**Deepika Badampudi:** Conceptualization (supporting), data curation (supporting), formal analysis (supporting), funding acquisition (equal), investigation (supporting), methodology (supporting), supervision (supporting), visualization (supporting), writing - review & editing (supporting)

**Other Papers not in this Thesis**


**Funding**

We acknowledge that this thesis was supported by the Knowledge Foundation through the OSIR project (reference number 20190081) at Blekinge Institute of Technology, Sweden.
# Contents

Abstract vii  
Acknowledgements ix  
Overview of Publications xi  
  Papers in this Thesis xi  
  Other Papers not in this Thesis xiii  

1 Introduction 1  
  1.1 Overview 1  
  1.2 Background 3  
  1.3 Research gaps and questions 5  
  1.4 Research methodology 8  
  1.5 Validity threats 10  
  1.6 Study results and synthesis 12  
  1.7 Conclusion and future work 18  

2 Paper 1: Understanding and Evaluating Software Reuse Costs and Benefits from Industrial Cases - A Systematic Literature Review 21  
  2.1 Introduction 22  
  2.2 Related work 24  
  2.3 Research methodology 27  
  2.4 RQ1 results and analysis: What are the characteristics of the primary studies on software reuse costs and benefits? 38  
  2.5 RQ2 results and analysis: What software reuse costs and benefits have been reported in the primary studies? 50
2.6 RQ3 results and analysis: How software reuse costs and benefits are measured in the primary studies? 61
2.7 RQ4 results and analysis: What is the strength of reported evidence on software reuse costs and benefits in the primary studies? 77
2.8 Discussion 86
2.9 Conclusion and future work 91

3 Paper 2: Reuse in Contemporary Software Engineering Practices – An Exploratory Case Study in A Medium-sized Company 93
3.1 Introduction 94
3.2 Related Work 97
3.3 Study Design 98
3.4 Results 105
3.5 Discussion 121
3.6 Conclusion and Future Work 127

4 Paper 3: Large Scale Reuse of Microservices Using DevOps and InnerSource Practices - A Longitudinal Case Study 129
4.1 Introduction 130
4.2 Background and Related Work 131
4.3 Research Method - Case study design 135
4.4 Case Description 140
4.5 Results RQ1: Software reuse costs and benefits 148
4.6 Results RQ2: Contemporary reuse challenges and improvement suggestions 153
4.7 Discussion 168
4.8 Conclusion 173

5 Paper 4: Using InnerSource for Improving Internal Reuse: An Industrial Case Study 175
5.1 Introduction 176
5.2 Related work 177
5.3 Research methodology 179
5.4 Results 185
5.5 Discussions 194
5.6 Conclusion and future work 198

References 201
List of Figures

1.1 Mapping between thesis objective, goals, research gaps, research questions, and chapters ........................................ 6
1.2 Chapter connections and contributions to the thesis objective ... 16
2.1 Combined search strategy and primary study selection process . 30
2.2 Number of papers per year ............................................... 39
2.3 Software reuse benefits identified from the primary studies ...... 51
2.4 Software reuse benefits in years ....................................... 52
2.5 Software reuse benefits and reuse approaches .................... 53
2.6 Software reuse benefits, reuse type, and project size .......... 54
2.7 Software reuse benefits, reuse purpose and measurement level .. 55
2.8 Software reuse benefits and reuse methods ...................... 55
2.9 Software reuse costs ..................................................... 57
2.10 Software reuse costs and reuse approaches ...................... 57
2.11 Software reuse costs, reuse purpose, and measurement level .. 58
2.12 Software reuse costs, reuse type, and project size ............ 59
2.13 Software reuse costs and reuse methods ......................... 59
2.14 Software reuse benefits and quality assessment results ....... 79
2.15 Software reuse costs and quality assessment results .......... 80
3.1 Software reuse and contemporary SE practices in S-Group Solutions AB ......................................................... 106
3.2 Costs and benefits of the software reuse in the context of contemporary SE practices ............................................ 108
3.3 Challenges and improvements of the software reuse in the context of contemporary SE practices ........................... 114
4.1 Research methodology followed in this study .................... 135
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>Example of the coding process</td>
<td>141</td>
</tr>
<tr>
<td>4.3</td>
<td>ADP Architectural framework</td>
<td>143</td>
</tr>
<tr>
<td>4.4</td>
<td>Different types of microservices in the ADP ecosystem</td>
<td>144</td>
</tr>
<tr>
<td>4.5</td>
<td>Maturity and reusability staircases (Reused with permission from Ericsson)</td>
<td>145</td>
</tr>
<tr>
<td>4.6</td>
<td>Costs and benefits involved in contemporary reuse</td>
<td>148</td>
</tr>
<tr>
<td>4.7</td>
<td>Causes resulting in a bottleneck</td>
<td>154</td>
</tr>
<tr>
<td>4.8</td>
<td>Contribution trend between 2020-2022</td>
<td>156</td>
</tr>
<tr>
<td>4.9</td>
<td>InnerSource as a reuse facilitator</td>
<td>157</td>
</tr>
<tr>
<td>4.10</td>
<td>Reasons inhibiting reuse of common microservices</td>
<td>164</td>
</tr>
<tr>
<td>5.1</td>
<td>Questionnaire example</td>
<td>182</td>
</tr>
<tr>
<td>5.2</td>
<td>Participants’ view on the current and desired status of the IS and reuse practices in the case company</td>
<td>186</td>
</tr>
<tr>
<td>5.3</td>
<td>What the participants think about the IS and reuse practices in the case company</td>
<td>187</td>
</tr>
<tr>
<td>5.4</td>
<td>Workshop prioritization results</td>
<td>191</td>
</tr>
</tbody>
</table>
## List of Tables

2.1 Secondary studies related to software reuse costs and benefits... 25  
2.2 Search database and search string. 29  
2.3 Number of primary studies in different study types. 40  
2.4 Publication type and channels. 40  
2.5 Contextual characteristics of the primary studies. 43  
2.6 Reuse specific characteristics of the primary studies. 47  
2.7 Better quality metrics, definitions and results. 62  
2.8 Better productivity metrics, definitions and results. 65  
2.9 Better maintainability metrics, definitions and results. 68  
2.10 Cost saving metrics, definitions and results. 69  
2.11 Faster time-to-market metrics, definitions and results. 70  
2.12 Other benefits, metrics, definitions and results. 70  
2.13 Costs, metrics, definitions, and results. 72  
2.14 No significant impact, metrics, definitions and results. 75  
2.15 Quality assessment results. 78  
2.16 Mapping between quality assessment and software reuse results. 81  
3.1 Aims and corresponding research questions of the data collection methods. 100  
3.2 Overview of the interview participants. 101  
3.3 Mapping between the interview questions and RQs. 102  
4.1 Evidence collection methods used and their purpose. 137  
4.2 Details of the interviewees. 139  
5.1 Proposed IS readiness instrument: categories and areas. 181  
5.2 Questionnaire participants’ demographics and experience in the case company. 185
Chapter 1

Introduction

1.1 Overview

Software reuse refers to the practice of utilizing existing software assets, such as components, modules, and libraries, to build new software systems or enhance existing ones [86]. Reusing existing software assets reduces the development time from scratch, thus increasing productivity. In addition, reusable software assets are often well-tested, which leads to better product quality. To attain the aforementioned advantages, companies implement software reuse within their projects or systems [48, 90, 133], which we refer to as internal software reuse.

Companies explored various practices to implement software reuse. Software product line (SPL) can be defined as a structured approach that aims to create new applications by defining and managing the commonality and variability among all software assets from a specific domain [107]. Component-based reuse [59] is another reuse approach that aims to develop self-contained, reusable code assets that can be easily assembled into larger systems. Service-oriented reuse [127] reuses simple services that could be integrated to produce new and more complex ones. Companies also used various reuse methods depending on whether they adhered to a planned or systematic reuse program, such as systematic reuse [124], ad hoc reuse [48] and controlled reuse [57]. Depending on the needs, companies may choose to reuse assets that are developed internally (e.g., reuse from the repository of reusable assets within the company) [133], external company (e.g., open source software) [25] or both [90]. When using the reusable assets, developers can reuse them as-is (also referred to as verbatim
reuse) if they fit the need or modify them to fit the need [124]. Researchers and practitioners also evaluated the benefits that different software reuse practices brought to companies, e.g., verbatim reuse resulting in better quality [135] and systematic reuse resulting in better productivity [90].

Although software reuse has been studied for over five decades, there is still a need to investigate how software reuse has evolved with the recent changes in software technologies (e.g., microservices, DevOps, and cloud-native) and processes (e.g., InnerSource). Companies are progressively adopting microservices, along with DevOps methodologies such as continuous integration and deployment, as well as container-based solutions like Docker, in order to improve the delivery time and scalability of their products and systems [10, 25, 61, 130]. Companies also found InnerSource helps promote software reuse via a common space for sharing reusable assets as it provides better transparency of the reusable assets and avoids duplicated work [54, 76, 141]. The growing popularity of open source reuse, namely opportunistic reuse, has also impacted software reuse. Mäkitalo et al. [77] found that reusing from open source software helps improve software quality and productivity. We refer to these new software engineering (SE) practices and technologies, including microservice-based architecture, DevOps, cloud-native, InnerSource, and opportunistic reuse, as contemporary SE practices.

In this thesis, we aim to investigate how software reuse can be improved within the context of contemporary SE practices. To achieve that, we formulated two goals (Gs): G1 - Understanding internal software reuse, both from literature and practice, and G2 - Improving internal software reuse. For G1, we conducted a systematic literature review (Chapter 2) on software reuse costs and benefits in industrial cases to identify what software reuse practices implemented by industrial cases lead to what benefits, how the costs and benefits are measured, and their strength of evidence. We also conducted two exploratory case studies (Chapters 3 and 4) in two companies of different sizes. In these case studies, we investigated how companies practice software reuse within the context of contemporary SE practices. In addition, we also identified costs, benefits, challenges, and improvements related to software reuse practices. For G2, following up on the case study reported in Chapter 3, we conducted an improving case study (Chapter 5) to investigate how InnerSource can be used to improve internal software reuse.

The remainder of the thesis is structured as follows. Section 1.2 describes the background information about software reuse. Section 1.3 presents the thesis objective, the identified research gaps, and the corresponding research questions. We explain the research methods used in our studies in Section 1.4 and discuss
the validity threats of the studies in Section 1.5. Section 1.6 presents an overview of the chapters’ results and a synthesis of how each chapter contributes to the thesis before we conclude the thesis and propose future work in Section 1.7.

1.2 Background

1.2.1 Traditional software reuse

Software reuse was introduced in the 1960s when Mcilory et al. [82] presented component reuse at a NATO Software Engineering Conference. Since then, studies have investigated various topics in software reuse, such as software reuse approaches (like component-based development [59] and software product lines [107]), and reusable assets (like code [135], requirements [50] and knowledge gained from existing test cases [62]). Barros et al. [12] revealed that the most frequently reported topics are requirements and testing in software product lines and systematic reuse for decision making.

Understanding the costs and benefits associated with software reuse practices is crucial for aiding companies in decision-making regarding their software development. Many case studies [48, 90, 133, 135] have investigated software reuse costs and benefits. Furthermore, these primary findings have been reviewed in various systematic secondary studies. Both Mohagheghi et al. [86] and Barros et al. [14] conducted review studies and identified a consensus that software reuse contributes to enhanced productivity and product quality.

1.2.2 Contemporary SE practices and software reuse

In this section, we describe the connection between contemporary SE practices (i.e., opportunistic reuse, InnerSource, and microservices together with DevOps and cloud-native) and software reuse.

Opportunistic reuse

Open source software is protected by copyright and is distributed under licenses specifically crafted to ensure the source code availability [23]. Pre-tested libraries are often used as building blocks for larger software systems, providing reusable functionality and improving the release time [63]. Mikkonen and Taivalsaari [85] refer to “developing new software systems by routinely reusing and combining components (open source components and modules online) that were not designed to be used together” as opportunistic design.
Opportunistic reuse brings both benefits and drawbacks. For benefits, Morgan and Finnegan [92] found that open source software reuse benefits in better software quality, flexible use, a good base of developers and testers, low costs, increased collaboration, and increased business functionalities. In addition to what Morgan and Finnegan [92] reported, Mäkitalo et al. [77] found it helps increase productivity, faster development, avoid common security risks, and have stable initial prototypes. For drawbacks, Capilla et al. [26] found that integrating reusable assets, which were found opportunistically, increases the number of smells and issues in most cases. Morgan and Finnegan [92] and Mäkitalo et al. [77] also investigated the drawbacks of opportunistic reuse, and they found it causes problems in compatibility issues, poor documentation, lack of support, uncertain ownership, dependency issues, security and licenses issues.

InnerSource and software reuse

Inspired by the open source way of working, Tim O’Reilly coined the term InnerSource [34] as “the use of open source development techniques within the corporation”. InnerSource has four development practices: participatory reuse, self-selection of tasks, volunteering, and collaborative development projects [28]. Participatory reuse, as one of the InnerSource practices, helps to improve internal reuse by letting the consumers of the reusable assets contribute to the reusable assets development phases. Vitharana et al. [141] 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.”

The adoption of InnerSource practices has helped companies in improving their internal reuse [28, 76, 131]. Many companies have adopted InnerSource [76, 76, 83, 138, 141] and found InnerSource practices improved the transparency across teams via a central space of sharing common reusable assets and therefore helped not only avoid duplicated work but also improved software reuse. To support companies in adopting InnerSource, InnerSource Commons¹, as the world’s largest community of InnerSource practitioners, have created a lot of InnerSource patterns² and shared the knowledge worldwide. For example, the pattern - InnerSource Portal helps companies build an intranet website to advertise reusable assets to increase the participatory reuse contribution.

¹https://innersourcecommons.org/
²https://patterns.innersourcecommons.org/
1.3 Research gaps and questions

Microservices and software reuse

The adoption of microservices-based architecture, together with the cloud-native and DevOps practices, has been on the rise. Microservices architecture comprises a number of smaller, independently deployable services [30]. Microservices are decentralized and are built and released with automated deployment processes [96]. Many technologies and cloud vendors have emerged to support microservices in better application development, testing, and deployment, such as Docker and Kubernetes. Transitioning to microservices and using DevOps helps improve portability, reuse, and maintainability [36]. In addition, microservices are language-independent and can be easily adjusted to almost any technology.

The individually encapsulated characteristic of microservices, together with the DevOps and cloud-native practices, helps increase software reuse. Microservices help increase reuse [51, 136] and decrease time when developing similar applications, resulting in financial gain [51]. Similar findings were made by Shadija et al. [126] who found more “fine-grained” microservices enable greater reuse when the microservices are located in the same container or different containers but on the same host. Schneider [121] found that using DevOps helps quickly deploy reusable microservices. Though microservices are language-independent, Wang et al. [142] found that the benefit of using multiple languages limits the reuse opportunity since understanding the microservices in different languages takes a longer time. In addition, the authors also found it challenging to manage the common code shared by multiple microservices.

In this thesis, we focus on investigating software reuse in the context of mixed contemporary SE practices. Chapter 3 investigates software reuse impacts in opportunistic reuse and InnerSource. Chapter 4 investigates the impact of reusing microservices in a cloud-native environment while adopting InnerSource practices.

1.3 Research gaps and questions

The overall goal of the thesis is to improve internal software reuse within the context of contemporary software engineering practices. We derived two goals as described in Section 1.1 and identified three gaps. To address the gaps, we raised three corresponding research questions. Figure 1.1 shows the connections between the thesis’s overall goal, goals, gaps, research questions, and chapters. We explain the research gaps as follows.
Overall goal: Improving internal software reuse in the context of contemporary software engineering practices

Goal 1: Understanding internal software reuse

<table>
<thead>
<tr>
<th>Gaps</th>
<th>Research Questions</th>
<th>Chapters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gap 1: Lack of systematic review studies on existing software reuse practices and their impacts in industrial cases</td>
<td>What is the state of the art of software reuse practices and their impacts in industrial cases?</td>
<td>Chapter 2</td>
</tr>
<tr>
<td>Gap 2: Lack of research in internal software reuse within the context of contemporary software engineering (SE) practices</td>
<td>What is the state of the practice of internal software reuse within the context of contemporary SE practices?</td>
<td>Chapters 3 &amp; 4</td>
</tr>
</tbody>
</table>

Goal 2: Improving internal software reuse

<table>
<thead>
<tr>
<th>Gaps</th>
<th>Research Questions</th>
<th>Chapters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gap 3: Lack of research in improving internal software reuse using InnerSource within the context of contemporary SE practices</td>
<td>How to improve internal software reuse using InnerSource within the context of contemporary SE practices?</td>
<td>Chapter 5</td>
</tr>
</tbody>
</table>

Figure 1.1: Mapping between thesis objective, goals, research gaps, research questions, and chapters

Gap 1: Lack of systematic review studies on existing software reuse practices and their impacts in industrial cases  Companies have adopted various software reuse practices and reported many costs and benefits from practicing them. Barros et al. [14], Mohagheghi et al. [86], Younoussi and Roudies [146], and Bombonatti et al. [21] performed review studies to investigate software reuse costs and benefits. However, the above existing secondary studies have the following gaps: 1) they are less focused on software reuse costs, 2) they have gaps in reporting software reuse costs and benefits measurements from 2006, 3) only one secondary study investigated the relationship between existing software reuse practices and their impacts, 4) not all primary studies in these secondary studies include industrial cases, and 5) they have not done a rigorous quality assessment. Therefore, we performed a systematic literature review (Chapter 2) to address the above mentioned gaps.
1.3 Research gaps and questions

**Gap 2: Lack of research in internal software reuse within the context of contemporary software engineering (SE) practices**  
As discussed in Section 1.2.2, studies show that open source software, microservice, and InnerSource help improve software reuse. However, according to the findings from Chapter 2, we did not find the existing literature explicitly reporting the reuse benefits from Contemporary SE practices. In addition, there is a lack of research on software reuse costs, benefits, challenges, and improvements in the context of a combination of different contemporary SE practices. To address this gap, we conducted two exploratory case studies (Chapters 3 and 4) in two companies of different sizes. Chapter 3 investigates the costs, benefits, challenges, and improvement areas of participatory and opportunistic reuse. Chapter 4 investigates the costs, benefits, challenges, and improvement areas of reusing microservices in a cloud-native environment while adopting InnerSource practices.

**Gap 3: Lack of research in improving internal software reuse using InnerSource within the context of contemporary SE practices**  
In Chapter 3, the main software reuse challenges we identified are the discoverability and ownership of the reusable assets. The study results showed that InnerSource helps address the challenges related to the development, maintenance, and ownership of the reusable assets. Therefore, we conducted a follow-up improving case study (Chapter 5) to help the medium-sized case company scale InnerSource to improve internal software reuse. Bauer [16] mentioned adopting InnerSource practices requires a good understanding of the company’s needs and context. Existing IS frameworks [49, 131] and maturity model [41] help the companies understand their needs and IS context. The existing literature often reports IS adoption in large companies [76, 113]. However, there were no clear guidelines on which InnerSource practices can be selected and prioritized by a medium-sized company. To tackle this gap, in Chapter 5, we developed a questionnaire by customizing the InnerSource maturity model pattern ³ to understand the case company’s InnerSource needs in software reuse and select the needed InnerSource improvement areas. In addition, we also conducted a workshop for prioritizing the InnerSource improvement areas and brainstorming possible solutions for the prioritized InnerSource improvement areas.

³https://patterns.innersourcecommons.org/p/maturity-model
1.4 Research methodology

To achieve different research goals, we used a combination of research methodologies: systematic literature review and case study, which are explained as follows.

1.4.1 Systematic literature review

A systematic literature review is a method used to aggregate and analyze the data and findings of other authors about a specific research question or questions through a scientific and repeatable process. We followed the guidelines from Kitchenham et al. [66] and conducted a systematic literature review on software reuse costs and benefits (Chapter 2). We conducted the systematic literature review in the following steps.

1. Study identification. In this step, we search and select primary studies. We used a combination of automated search and snowballing search strategies. We followed the automated search guidelines from Kitchenham et al. [66] and the snowballing search guidelines from Wohlin [144]. We used the automated search strategy to identify an appropriate start set since snowballing needs a good start set [144]. In addition, We used two tertiary studies as another data source to validate our search results and include more relevant secondary studies. We developed two selection criteria to identify secondary and primary studies on software reuse costs and benefits.

2. Data extraction. We developed a data extraction form based on the research questions.

3. Quality assessment. This step evaluates the quality of the primary studies. We used the quality assessment checklist developed by Dybå and Dingsøyr [40], which is originally from the quality assessment criteria on the critical appraisal skills programme (CASP) [109] as our quality assessment base.

4. Analysis. This step analyzes the data extracted from the primary studies. We categorized the study characteristics into three categories: general information, context characteristics, and reuse specific characteristics. Thematic analysis [35] was used to code the software reuse costs and benefits. Extracted metrics are grouped into different software reuse costs and
1.4 Research methodology

benefits. We analyzed the quality assessment results based on different types of studies. Narrative synthesis [116] was used to understand how the threats to validity are reported in the primary studies and analyze the reported validity threats.

1.4.2 Case study

A case study is used to study a contemporary phenomenon in its natural context. According to the study purpose, case studies can be categorized into four types [115]: exploratory, descriptive, explanatory, and improving. In our case, we conducted exploratory case studies in two companies of different sizes (Chapters 3 and 4) since we wanted to understand the state of practice of software reuse within the context of contemporary software engineering practices. We also conducted a follow-up improving case study (Chapter 5) to help the medium-sized company identify and prioritize needed InnerSource practices, further improving their internal reuse. In addition, Chapter 4 is a longitudinal case study, where we conducted a follow-up group discussion after a year to understand the status of the identified challenges and improvements.

Data collection methods  

*Semi-structured interview* In Chapters 3 and 4, we selected semi-structured interviews since they allow improvisation and exploration of the studied objects [118] and capture unexpected information on the studied topic [123].

*Group discussion* In Chapters 3 and 4, group discussions are conducted since they collect in-depth perspectives through interactive conversations with multiple participants, providing more descriptive and elaborated data. In addition, the group discussions were used to validate the interview results.

*Additional company material* Additional material from companies is collected for data triangulation. We collected company documentation in Chapters 3 and 4 and got access to the company’s intranet website in Chapter 4.

*Questionnaire* The questionnaire consists of a set of questions that helps gather information more quickly and cost-effectively [128]. We followed the survey guidelines from Linäker et al. [74] to create the questionnaire in Chapter 5.

*Workshop* Workshops allow the participants to share their opinions and discuss and exchange ideas among multiple stakeholders. In Chapter 5, we followed the workshop guidelines from Brem [22].
Data analysis methods  Thematic analysis In Chapters 3 and 4, taking inspiration from the recommended thematic synthesis steps [35], we coded and analyzed our data in two phases: 1) code the extracted text segments that related to the research questions, 2) translate the codes into themes.

Principle of survey research We followed the principles of survey research proposed by Kitchenham and Pfleeger [64] to analyze the questionnaire results in Chapter 5.

1.5 Validity threats

Validity threats occur in two phases [103]: (1) study design and data collection, and (2) data analysis. In the following, we discuss our studies’ validity threats according to the four scheme validity from Runeson and Höst [118]: construct validity, internal validity, external validity, and reliability.

Construct validity concerns whether the studied measurements answered the research questions and performed as the researchers expected. In Chapters 3 and 4, 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. We provided the reuse related role descriptions to help the contact person identify the relevant people for the interview. In both studies, we tried to involve as many participants as possible and cover all the necessary roles. In Chapter 3, we developed the questionnaire iteratively and reviewed it internally. We also asked for an additional review from the company contact person. We provided definitions for specific terms in the questionnaire so the respondents understood the meaning. We also provided open-text fields in the questionnaire to collect additional comments from the respondents.

Internal validity concerns whether other factors influence the causal relationship under test. Though we have not investigated the causal relations in the thesis, we ensured factual accuracy by asking the contact person to review the results constantly. In Chapters 3 and 4, 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 with the company contact person to ensure the results captured reality. Similarly, in Chapter 5, we presented the study participants with the questionnaire and workshop results multiple times to ensure the accuracy of the results.
1.5 Validity threats

We piloted the study selection criteria, quality assessment checklist, and data extraction form many times to ensure that all authors have the same understanding in Chapter 2. We involved more than one researcher during the study selection phase. In addition, the second and third authors reviewed three randomly selected primary studies on the quality assessment and data extraction results.

**External validity** represents the generalizability of the findings. In Chapters 3 and 4, we applied the interview guide to two different sizes of companies. We may not be able to generalize the identified software reuse costs, benefits, challenges, and improvement areas to all types of companies since different companies have their unique context. However, we have reported the context of the studies where we performed the case studies. The results are relevant to the companies that are working in a similar context. In Chapter 5, we only applied the questionnaire to one medium-sized company. Due to the availability of the teams, the workshop only included producers, and the proposed solutions were limited to producers’ perceptions. Our questionnaire can potentially provide other companies with a new way of assessing their company situation for IS and reuse practices. Our findings, especially the proposed IS solutions, still benefit the companies.

We developed the study protocol according to Kitchenham et al. [65] in Chapter 2. We used a combination of automated search and snowballing search strategies to identify the primary studies related to software reuse costs and benefits with industrial cases. We used an indirect approach for the automated search: identify primary studies using secondary studies. To make our search more inclusive, we searched for secondary studies on software reuse on four databases. We also used two tertiary studies as an additional data source and used them to validate the automated search results. Then, we use the secondary studies selection criteria to include secondary studies on software reuse costs and benefits. Primary studies on software reuse costs and benefits are extracted from the included secondary studies on software reuse costs and benefits. Since the identified secondary studies have gone through a thorough search process - automated search, manual search, and snowballing, which covers the primary studies published before 2017, we chose to apply only forward snowballing on the identified primary studies to include more recent primary studies. In Chapter 2, we may be unable to extract many software reuse costs related to primary studies since software reuse success stories are probably published more frequently than failure stories.

**Reliability** refers to the researchers’ bias on the data and the analysis. In Chapters 3 and 4, we followed Cruzes and Dybå’s [35] recommended steps of
thematic analysis to code and analyze the transcripts. Multiple authors were involved in coding and analyzing the transcripts. The authors had extensive discussions and reviews on the thematic analysis result to ensure its trustworthiness. We also presented the results to the company to eliminate misinterpretation. In Chapter 5, we developed the questionnaire instrument based on the IS maturity model pattern. We presented the questionnaire results to the company contact person, who agreed that the results reflected reality. We presented the workshop results at the end of the workshop and shared the results in the form of a report with the company contact person, which did not lead to any major changes.

1.6 Study results and synthesis

This section provides an overview of the study findings in each chapter (see Section 1.6.1) and synthesizes how each study contributes to the thesis objective (see Section 1.6.2).

1.6.1 Study results

Software reuse costs and benefits (Chapter 2)

To address Gap 1 (see Section 1.3), we conducted a systematic literature review, following the guidelines provided by Kitcheham et al. [66]. In total, we identified 30 primary studies from 506 secondary studies and 509 primary studies.

We identified nine software reuse benefits and six software reuse costs. The most mentioned software reuse benefits are better quality and improved productivity. Few studies investigated software reuse costs. The results also showed that verbatim reuse and systematic reuse lead to more software reuse benefits. Software product line and reuse from the repository are the most common reuse approaches. Though software reuse is improved with contemporary software engineering practices, we found no industrial cases investigating software reuse costs and benefits in the context of contemporary software engineering practices, such as microservice reuse and participatory reuse.

For quality assessment, we found some of the criteria in the quality assessment checklist do not apply to certain study types, i.e., experience report has less obligation in reporting a rigorous research method. The results also show that most of the included primary studies are of moderated quality, with only four having high quality. Better product quality was observed in all types of quality studies, indicating that software reuse is highly likely to benefit better
1.6 Study results and synthesis

Product quality in most cases. Improved productivity - the second most cited benefit, is mainly noted in moderate quality studies.

One thing worth noting is we found that software reuse may impact an attribute differently depending on different metrics. For example, the results showed that software reuse helped improve performance in terms of memory size but reduced performance regarding processing time.

Software reuse in contemporary software engineering (SE) practices (Chapters 3 and 4)

To address Gap 2 (see Section 1.3), we conducted two exploratory case studies on two companies of different sizes to understand the state of practice of software reuse in contemporary software reuse practices.

Chapter 3 We conducted six interviews and four group discussions. We also collected relevant documentation from the medium-sized company. At the time of investigation, the case company was still at the beginning (about two years) of its software reuse journey.

The company follows both participatory reuse and opportunistic reuse. They manage all reusable assets in the internal DevOps repository. Different development teams collaborate in developing and maintaining internal reusable assets. In opportunistic reuse, developers maintain the external reusable assets locally or make upstream contributions to the Open Source community for bug fixes.

We categorized the identified benefits into three context facets: people, process, and product. The reuse practices in the medium-sized company help the reusable assets consumers have a better learning experience, reduce the development, maintenance, and testing time, faster time-to-market, reduce the need for dedicated resources, improve product quality, and have consistent UI. We categorized the costs in two reuse views [1] - development for reuse (DFR) and development with reuse (DWR). In DFR, the costs occur most in designing, developing, documenting the reusable assets, and coordinating for creating reusable assets. Costs in DWR result from learning, analyzing, and coordinating for using the reusable assets. However, the costs pay off when developers start to reuse more.

The medium-sized company faced challenges related to external open source software (i.e., open source reuse hesitation from the management level, increased risk, lack of documentation, and license restrictions), internal reusable assets (i.e., lack of development time, limited discoverability, lack of documentation, ownership issues, reusable assets planning and scoping issues), knowledge sharing, unaware of reuse benefits from non-technical people and reduced learning
from scratch. The interviewed practitioners also provided many improvement suggestions regarding internal reusable assets, such as using tools to support creating internal reusable assets and automating document generation. The top concerned software reuse challenges are discoverability and maintenance of the reusable assets, knowledge sharing of the reuse related information, and lack of reuse measurement. The case company has already practiced some InnerSource practices, and the case study results showed that scale InnerSource could further address most identified software reuse challenges.

**Chapter 4** We interviewed 12 people about the costs, benefits, challenges, and improvements of software reuse in contemporary SE practices. We also had a follow-up group discussion after a year to understand the status of the challenges and improvements identified in the first phase.

The investigated company is transitioning towards a cloud-native approach to developing software applications and microservices. To improve internal reuse, the case company made the following efforts before the investigation: 1) established a platform with clear responsibility of technology, tools, practices, and principles support for developing and reusing common microservices, 2) developed an internal portal - marketplace, that hosts all common microservices and their necessary information, 3) the common microservices are developed and maintained centrally, 4) adopted a modern cloud-native architecture using Kubernetes\(^4\) and Helm\(^5\), 5) had clear InnerSource roles and responsibilities, and 6) one of the goals of the platform is to encourage the consumers of the common microservices also to become producers and make contributions.

The main benefits the practitioners perceive are better quality, improved productivity, improved customer experience, improved competence, and improved way of working (e.g., reuse practices). With regards to costs, we categorized them in two reuse views, as in Chapter 3. In development for reuse, additional costs occur in implementing design rules - the requirements for handling issues such as configuration, integration, and backward compatibility. Documentation, deployment, and testing of the microservices also required additional effort. As for development with reuse, it costs more to understand and integrate the microservices. In addition, the microservices consumers need to update their competence (i.e., knowledge and infrastructure) to use microservices. Nonetheless, the investment in creating common microservices is returned when various users within the organization start to reuse them.

\(^4\)http://kubernetes.io/
\(^5\)http://helm.sh
The company also faced challenges in software reuse under contemporary SE practices. The central team that develops and owns the common microservices may have a bottleneck due to the large number of queries and requests by the users of the common microservices and the lack of contributions from the potential contributors. Developers may misinterpret certain requirements, such as design rules, which could delay the releases. The common microservices are released much more frequently, which results in the misalignment between the release cycle of producers and consumers. In some cases, the frequent updates in the microservices may break the backward compatibility. Sometimes, it is difficult to reuse common microservices since they lack documentation and take time to sustain their maturity when new changes are added. The case company mitigated the above mentioned challenges through the adoption of InnerSource, automating the implementation and checking of design rules, and enhancing the maturity of common microservices.

Identify InnerSource improvement areas (Chapter 5)

In Chapter 3, the main challenge is about how to improve the development, maintenance, and ownership of reusable assets. One of the proposed improvements in Chapter 3, which the practitioners discussed, was the adoption of InnerSource practices. Therefore, we conducted a follow-up improving case study (Chapter 5) to help the medium-sized company identify and prioritize the needed InnerSource practices using questionnaires and a workshop. In total, we received seven questionnaire responses and had a workshop with five practitioners.

In the questionnaire, we asked for demographic information and opinions on InnerSource and reuse practices. The demographic information includes the role, the team information, working experience, and the involvement of the reuse practices (development, consuming, or maintaining reusable assets). In total, we have 19 areas for the InnerSource and reuse practices, covering six categories: reusable assets, reuse process, reuse collaboration, the measurement for reuse, rewards for reuse, and culture regarding reuse. We asked the respondents both the current and desired status of InnerSource reuse practices to help the company identify the improvement areas. The current status results showed that the transparency of the reusable code assets and the plans/roadmaps for overall projects are relatively mature. The desired status results showed that all practitioners want to achieve systematic and mature InnerSource and reuse practices in the future for most areas. Combining the results of current and desired status, we found that the practitioners see the need for improvements in five particu-
lar areas: discoverability of reusable assets, plans/roadmaps for reusable assets, contribution/maintenance support for reusable assets, communication channels for reuse, and measuring/monitoring reuse.

In the workshop, the participants prioritized the improvement areas according to the importance and costs to the company. The results showed that the top prioritized areas are the discoverability of reusable assets, communication channels for reusable assets, and contribution/maintenance support for reusable assets. The participants also proposed some concrete InnerSource practices as solutions for the prioritized improvement areas, such as better structured repositories for storing and searching the reusable assets and standardized documentation for the reusable assets.

1.6.2 Synthesis

![Diagram showing chapter connections and contributions]

Figure 1.2: Chapter connections and contributions to the thesis objective

Each chapter in this thesis reports an individual study, contributing to the overall thesis objective: improve internal software reuse in the context of contem-
1.6 Study results and synthesis

Temporary software engineering (SE) practices. Figure 1.6.2 presents the connection between chapters and how they contribute to the thesis objective.

Before providing a solution to improve internal software reuse in contemporary SE practices, it is important to understand the area. Chapter 2 enriches the understanding of software reuse costs and benefits reported in the existing literature through five contributions: 1) Identified existing software reuse costs and benefits. 2) Identified the relationship between software reuse practices and software reuse costs and benefits. 3) Identified the metrics for measuring software reuse costs and benefits. 4) Assessed the strength of evidence on software reuse costs and benefits. And 5) Identified research gaps in the existing literature and suggested future research directions.

We also conducted two exploratory case studies (Chapters 3 and 4) to understand how software reuse is practiced with contemporary SE practices in two different sizes of companies and what software reuse costs, benefits, challenges, and improvement areas in contemporary SE practices are perceived by practitioners. The knowledge acquired from the literature and practices allowed us to have a better understanding of internal software reuse in contemporary SE practices.

According to the literature review and two case studies, we found that existing software reuse practices and software reuse in the context of contemporary SE practices benefit both in better product quality and improved productivity. Practitioners in two case studies also mentioned the benefits of learning and customer satisfaction, which were measured in some of the existing literature. We did not find some benefits identified in the case studies in our systematic literature review: reduced need for dedicated resources, consistent UI, improved competence, and improved way of working. Practitioners have also mentioned a lot of upfront costs, such as additional documentation. However, these upfront costs were not explicitly measured in the existing literature. In general, the existing literature reports less cost measurement results. In addition, we did not find existing literature measuring the software reuse costs and benefits in the context of contemporary SE practices.

In the context of contemporary SE practices, both case companies faced challenges in developing and maintaining reusable assets. Documentation was challenging for both case companies. The medium-sized company lacked documentation because of time constraints. The large-sized company had documentation but lacked descriptions of reusable assets’ purpose, use cases, and limitations. Due to the size differences, the medium-sized company found it hard to allocate time to develop reusable assets. Though the large company has a central team that is responsible for the development and maintenance
of reusable assets, they may become a bottleneck due to frequent queries and the lack of contributions from consumers. In addition, the large-sized company established requirements for developing the reusable assets, while the medium-sized company used meetings to discuss the design of the reusable assets. Due to the different stages of the software reuse journey, the medium-sized company faced challenges in planning and scoping reusable assets and discovering existing reusable assets, while the large-sized company already established a relatively mature platform and intranet website for developing and hosting reusable assets. For the improvements, both case companies used automation and InnerSource to improve their internal software reuse.

After having a good understanding of internal software reuse in contemporary SE practices, we conducted a follow-up improving case study (Chapter 5) to help the medium-sized company select and prioritize the InnerSource improvement areas. This improving case study is for the collaborated medium-sized company only. Thus, the InnerSource readiness instrument is limited in the medium-sized company and early reuse journey. In the future, more investigation is needed to improve the generalizability of the instrument.

1.7 Conclusion and future work

In this thesis, we aim to improve internal software reuse in the context of contemporary software engineering practices. A systematic literature review (Chapter 2) is conducted to investigate the state of the art of software reuse costs and benefits and how they are measured. We also investigated the relation between software reuse practices and the identified software costs and benefits and assessed the study quality to understand the strength of the evidence. Two exploratory case studies (Chapters 3 and 4) are conducted to investigate the state of the practice of software reuse costs, benefits, challenges, and improvement areas in contemporary software engineering practices for medium-sized and large-sized companies. The findings of Chapters 3 and 4 show that InnerSource can help address the challenges faced in software reuse. In Chapter 5, we helped the medium-sized case company identify and prioritize InnerSource improvements through InnerSource readiness check questionnaires and a workshop.

We had the following findings based on the studies conducted in the thesis. The most reported software reuse benefits are better product quality and improved productivity. In contemporary software engineering practices, software reuse benefits the same as traditional software reuse in product quality and productivity. Sharing reusable assets company-wide to maximize the software reuse
1.7 Conclusion and future work

benefits become a common goal for most companies. InnerSource practices help mitigate the challenges encountered during the internal software reuse process, such as the discoverability of reusable assets. To ensure success in InnerSource reuse, InnerSource solutions should be designed to fit the company’s needs and context.

In Chapter 5, we developed an instrument to assess the company’s need of adopting InnerSource. However, it is only applied to a single company. In the future, we want to improve and use the instrument in other companies. Based on that, we want to propose guidelines that help companies in selecting and prioritizing InnerSource practices for the improvement of internal software reuse.
Chapter 3

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

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

### 3.1 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” [86]. The benefits of software reuse such as improved product quality, faster time-to-market and reduced development costs [14, 21, 86] 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 [34]), the research in software reuse still remains relevant. In 2019, Barros-Justo et al. [12] conducted a tertiary study to investigate the trends in software reuse research. They identified many software reuse 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. [25] 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 [85] 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. [145] also identified a trend in increased library reuse of Maven libraries in Maven Central. Although oppor¬

---

1 Statistics for the Maven Repository, https://search.maven.org/stats
3.1 Introduction

tunistic reuse is the opposite of systematic reuse as Barros-Justo et al. [14] and Capilla et al. [26] 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 [60], and IBM (CMS) [46, 141]. Inspired by the open source way of working, Tim O’Reilly coined the term InnerSource [34] as “the use of open source development techniques within the corporation” in 2000. Vitharana et al. [141] 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. [10, 25, 61, 130]).

Studies have investigated the impact of opportunistic reuse [26], microservices [51], and InnerSource [28] on software reuse. Capilla et al. [26] 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 [51] identified increased reuse as one of the main benefits of migrating from monolith to microservices based architecture solutions. InnerSource (IS) practices facilitate software reuse [28]. 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 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 [26].

Barros-Justo et al. [13] 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 [14, 21, 86]. 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 3.2 presents the related work; Section 3.3 describes the study design; Section 3.4 provides the study results; Section 3.5 discusses the results in comparison to the related works and provides discussion on threats to validity; Section 3.6 concludes the paper and proposes the future work.

\(^2\)https://sgroup-solutions.se/
\(^3\)https://patterns.innersourcecommons.org/
3.2 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 [18, 69, 129] and opportunistic reuse [5, 18, 77]. Furthermore, less maintenance effort [13, 18, 77], standardized architecture [129] and higher documentation quality [13] have also been identified as benefits of software reuse.

Relatively fewer studies investigated software reuse costs than benefits. Kruger and Berger [69] 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. [129] investigated the relation 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. [13] replicated Bauer et al.’s study [18], investigating the challenges and problems related to software reuse practices. Both studies [13, 18] 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. [77] and Barros-Justo et al. [13] 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 [69].

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 [4], tools to help improve the discoverability of the reusable assets [4, 17] and allocating developers a separate time budget to develop or maintain the reusable assets [4].

Barros-Justo et al. [13] 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.

3.3 Study Design

This section presents the details of the study design.

3.3.1 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 [118] as our research method to investigate the current reuse practice in the company. We used three data collection methods (see Section 3.3.4 and Table 3.1): interviews, group discussions and company documentation.

3.3.2 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.

3.3.3 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 [2]. 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.

3.3.4 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 3.1. We used group discussions and the company documentation to validate and triangulate the interview data [see aims in Table 3.1]. 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 [118] and captures unexpected information on the studied topic [123]. 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 3.1: Aims and corresponding research questions of the data collection methods

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

**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 3.2 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 3.2: Overview of the interview participants

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

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

**Interview design:** As mentioned in Section 3.3.1, we chose to conduct semi-structured interviews. The second author developed the interview guide (see Table 3.3) 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 3.3. We used the interview questions as a guide and followed semi-structured interview format which allowed for flexibility in the interview.

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 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
Table 3.3: Mapping between the interview questions and RQs

<table>
<thead>
<tr>
<th>Interview questions</th>
<th>Corresponding RQs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Overview of the current reuse practices and your role -</td>
<td>Demographic information</td>
</tr>
<tr>
<td>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?</td>
<td>RQ1</td>
</tr>
<tr>
<td>b) Details on the product, team/s and shared assets: i: What is the size of your team? ii: Which software artifacts (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? iv: Which software assets do you/your team share (across site)? What solution do they offer? v: Can you give an example? vi: Do you produce and/or consume the shared assets? Give examples of the shared assets you are involved in?</td>
<td>RQ2</td>
</tr>
<tr>
<td>2. Is there a company/project/unit wide strategy/policy/goal to develop with reuse?</td>
<td>RQ1</td>
</tr>
<tr>
<td>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)?</td>
<td>RQ1</td>
</tr>
<tr>
<td>4. How is the funding of shared assets done?</td>
<td>RQ1</td>
</tr>
<tr>
<td>5. What is your experience regarding developing for and with reuse? What reuse related activities/tasks/initiatives are you involved in?</td>
<td>RQ1</td>
</tr>
<tr>
<td>6. What activities, if any, are performed in your company to: a) Identify the reusable assets b) Develop/adapt reusable assets. c) What are the unique activities in development of reusable assets? d) Use reusable assets or replace existing assets with reusable assets. e) Maintain reusable assets. f) Share reusable assets or make reusable assets available.</td>
<td>RQ1</td>
</tr>
<tr>
<td>i: For all activities ask the following questions: i: Is there anyone response for this activity? If yes, who? If not, should there be any one responsible? ii: Who or what triggers this activity and how often? iii: What information/input is needed for the activity? iv: Who provides the information needed for the activity or how is it obtained?</td>
<td>RQ2</td>
</tr>
<tr>
<td>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, iv: the team?</td>
<td>RQ3</td>
</tr>
<tr>
<td>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, iv: the team?</td>
<td>RQ3</td>
</tr>
<tr>
<td>9. What are the challenges and improvement areas with respect to development for and with reuse?</td>
<td>RQ3</td>
</tr>
</tbody>
</table>

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 3.3.5. 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 3.4.3). 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 3.4.3). 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).

**3.3.5 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 [35] and four-steps data analysis process [104], 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 [120]. 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
3.4 Results

This section presents the results per research question.

3.4.1 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 3.1. 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.

Figure 3.1: Software reuse and contemporary SE practices in S-Group Solutions AB

In participatory reuse (see the solid-lined box on top in Figure 3.1), 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. Af-
3.4 Results

ter 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, NuGet packages 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 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 3.1), 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.

5https://www.npmjs.com/
6https://www.nuget.org/
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, except 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.

### 3.4.2 RQ2 Software reuse costs and benefits

<table>
<thead>
<tr>
<th>Costs</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DFR</strong></td>
<td><strong>PROCESS</strong></td>
</tr>
<tr>
<td>• Additional risk analysis for external reusable assets (TL2, SA)</td>
<td>• Better learning experience (TL1, TL2, SA)</td>
</tr>
<tr>
<td>• Additional time to learn reusable assets (SA)</td>
<td>• Reduced development time (TL1, TL2, PM, SA)</td>
</tr>
<tr>
<td>• Additional coordination with different teams (TL2, SA, PM, PO)</td>
<td>• Reduced maintenance time (SA, T, PM)</td>
</tr>
<tr>
<td>• Additional design effort to create reusable assets (TL1, TL2, T)</td>
<td>• Reduced need to have dedicated resources (TL1, TL2)</td>
</tr>
<tr>
<td>• Additional approval process for creating internal reusable assets (TL1, TL2, SA)</td>
<td>• Reduced testing time (T)</td>
</tr>
<tr>
<td>• Additional boilerplate code when creating reusable assets (TL1)</td>
<td>• Faster time-to-market (SA)</td>
</tr>
<tr>
<td>• Additional effort in creating reusable assets (TL1)</td>
<td></td>
</tr>
<tr>
<td>• Additional documentation when developing reusable assets (TL2)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.2: 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 3.2 provides
the classifications of the practitioner perceived costs and benefits, mapping with the participants by the role abbreviations (see Table 3.2). 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 3.4.2 and 3.4.2 discuss the reuse costs and benefits perceived by the participants respectively.

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” [1]. 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 3.1), 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. Three 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 3.1).

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,
3.4 Results

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.”

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 3.2), which were classified into three themes according to the context facets [105]: 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 3.2). 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.

### 3.4.3 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 3.4.3 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 3.4.3 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 3.4.3.

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 3.4.3.

Figure 3.3 provides the classifications of the practitioner perceived challenges and improvements, mapping with the participants by the role abbreviations (see Table 3.2).

Figure 3.3: Challenges and improvements of the software reuse in the context of contemporary SE practices
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”.

   **Transformed 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”.

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.

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].

Transformed **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].”

Transformed **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”.

Transformed **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]”.

---

**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 3.4.2.

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.”
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 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

---

\(^7\)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
3.5 Discussion

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

3.5.1 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 [69] and Agresti [4]. 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 [69] also found that practicing software reuse results in additional synchronization and coordination among different teams. They found additional coordination when handing over reusable assets 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 [4] found extra costs in understanding the reusable code. However, Agresti [4] identified extra cost when the reused assets need extensive modification. Our participants did not bring up such a cost. Comparing Agresti’s study [4] 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 3.1). Moreover, our case company did not mention additional integration effort in opportunistic reuse as we assumed.

3.5.2 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. [14, 21, 86]) and primary studies [4, 13, 17, 50, 69, 129] 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. [17] 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 et al. [17]’s study, software reuse contributes to feature consistency over the range of products. Literature related to internal reuse [50, 129] 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. [50] also found that require-
3.5 Discussion

ment management and reuse help the new employees complete the onboarding process easier and quicker from the learning perspective.

Barros-Justo et al. [13] 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 3.4.2) 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. [112] 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.

3.5.3 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. [13] and Bauer and Vetro [18] 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 [18] — 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 [18], and Barros-Justo et al. [13], 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 [69]
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. [94] and Kolb et al. [68] 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. [68] 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. [17] 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. [17] 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. [13] 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 [56] 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 [13, 18].

3.5.4 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 de-


3.5 Discussion

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. [4] 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. [17] 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 [4, 17] 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 [34]. 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. Mohaghegh and Conradi [86] 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. [4] 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. [4] 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 [4] suggested allocating additional budget for the developers to facilitate them for contributing to the reusable assets.

### 3.5.5 Threats to validity

We discuss threats to validity in two phases using the validity threats categorization proposed by Peterson and Gencel [103]: (1) study design and data collection, and (2) data analysis.

**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.

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 and Dybå’s [35] 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 3.3.5, 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 3.3.3 and Section 3.4.1), 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.

3.6 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.
Chapter 5

Using InnerSource for Improving Internal Reuse: An Industrial Case Study

Abstract

Background: InnerSource consists of the use of open source development techniques within the corporation. It helps improve software reuse through increased transparency and inter-team collaboration. Companies need to understand their context and specific needs before deciding to adopt any specific InnerSource practices since they cannot apply all InnerSource practices at once. Aim: This study aims to support the case company in assessing its readiness for adopting InnerSource practices to improve its internal reuse, identify and prioritize the improvement areas, and identify suitable solutions. Method: We performed a case study using a questionnaire and a workshop to check the current and desired status of adopting InnerSource practices and collect potential solutions. Results: The study participants identified that the company needs to prioritize the improvements related to the discoverability, communication channels, and ownership of the reusable assets. In addition, they identified certain InnerSource practices as solutions for the prioritized improvement areas, such as better structured repositories for storing and searching the reusable assets and standardized documentation of the reusable assets. Conclusion: The question-
naire instrument aids the case company in identifying the improvement areas related to InnerSource and reuse practices. InnerSource practices could improve the development and maintenance of reusable assets.

Keywords: InnerSource, software reuse, readiness

5.1 Introduction

Software reuse has been investigated for more than 50 years since McIlroy introduced component-based reuse in the late 1960s [82]. Studies found software reuse is associated with many benefits, such as increased product quality and reduced development time [14, 86]. To improve internal reuse, companies use approaches such as software product line (SPL), component- and model-based development [14]. Companies also use commercial-off-the-shelf/government-off-the-shelf (COTS/GOTS) software in their products and solutions[14]. Companies started to practice opportunistic reuse, which refers to “developing new software systems by routinely reusing and combining components (open source components and modules online) that were not designed to be used together” [85].

There are mainly two types of stakeholders in software reuse: producers responsible for developing reusable assets and consumers who reuse and integrate reusable assets in their solutions. Platform-based internal reuse has a dedicated team (producers) for developing reusable assets. However, they face challenges when the consumers want changes (e.g., adding new features and requiring bug fixes) in reusable assets, resulting in producers needing more bandwidth to complete the tasks within the consumers’ asking deadline [91]. Such a scenario may lead to a situation wherein the producer teams become a bottleneck due to a large number of change requests and bug fixes in the reusable assets [111]. In open source software development, this bottleneck issue is addressed by making the code openly available and encouraging all stakeholders to contribute with new features and bug fixes. Inspired by the open source way of working, Tim O’Reilly coined the term InnerSource (IS) [34] as “the use of open source development techniques within the corporation”. IS way of working has the potential to improve the development and maintenance of reusable assets due to its focus on transparency and collaboration among different teams and developers (e.g., producers and consumers of reusable assets). To address the transparency issues, Lucent Technologies [54] and Hewlett-Packard Company [38] used IS practices to develop a central space for sharing common reusable assets, which further promoted their internal reuse practices.
This study is part of a project on open source inspired reuse. The project aims to help companies improve their internal software reuse practices. In our previous study [32], the case company identified the reuse challenges in discoverability, transparency, and ownership of reusable assets. We also found such challenges can be addressed by adopting InnerSource practices. Bauer [16] suggested that understanding the companies’ needs and context is one of the success factors for adopting IS. Existing IS frameworks [49, 131] and maturity model [41] help the companies understand their needs and IS context. However, the studies did not investigate the selection and prioritization of IS improvement areas. In addition, the existing literature often report IS adoption in large companies [76, 113], while small- and medium-sized companies are less investigated. In this study, we investigated the readiness of a medium-sized company to adopt IS practices for improving internal reuse practices. We made the following contributions:

1. We developed and used an instrument to help the case company assess its current and desired status regarding adopting IS practices related to the development and maintenance of reusable assets.

2. We developed and used a two-dimension scheme – based on the importance and cost of implementing the improvements to help the case company prioritize the improvement areas.

The remainder of this study is structured as follows: Section 5.2 presents the related works. We introduced the research methodology in Section 5.3. Section 5.4 consists of the results related to the questionnaire and the workshop. Section 5.5 presents discussions of our findings and the validity threats of the study. Section 5.6 presents the study conclusion and future work.

5.2 Related work

Edison et al. [42], and Capraro and Riehle[28] conducted literature reviews and found IS helps avoid duplicated work and promote software reuse. Such benefits motivated many large-scale companies as well to adopt IS, e.g., Lucent Technologies [54], Philips [76], and Hewlett-Packard [83]. The case company also showed interest in adopting IS patterns, such as Trusted Committer\(^1\), de-

\(^1\)https://patterns.innersourcecommons.org/p/trusted-committer
Paper 4: Using InnerSource for Improving Internal Reuse: An Industrial Case Study

Developed by InnerSource Commons\(^2\), to improve the development, maintenance, and ownership of reusable assets.

Adopting IS is not without challenges. In the early phase, culture - hierarchical organizational structure [38, 83], proprietary mindset [80], and silo mentality [98, 113] hinder the IS adoption. During the adoption and practicing phase, integrating the IS practices into the existing development process and building organizational-wide infrastructure are challenging [38].

The IS teams may lack domain knowledge to build the shared assets [132] and may not provide necessary documentation [113, 132]. The developers outside the IS teams may not be able to follow the IS teams’ contribution process [54], not have time to contribute to the common reusable assets [91], submit unfit contributions [132], may be reluctant to contribute [113, 132], and treat IS team as a provider only [132]. After the other developers submit the contributions, the IS teams may be reluctant to accept the contribution [132]. In addition, the IS teams need to balance the work between code review for the contributions and the implementation of new functionalities [54]. It is also hard to determine who should be responsible for the maintenance of the contributed part if it needs some bug fixing in the future [32]. Therefore, the ownership of the shared reusable assets becomes really important [55].

Before adopting IS practices in any company, it is important first to understand its context, motivation, and readiness - i.e., to what extent and which IS practices are appropriate for the company’s needs and context. Bauer [16] reported a failed IS adoption case and concluded that adopting IS requires a good understanding of the company’s needs and context. Gaughan et al. [49] proposed a framework to help firms understand when and how to adopt IS. Stol et al. [131] proposed another framework with three themes - software product, practices & tools, and organization & community, covering nine key factors supporting inner source adoption. Linåker et al. [73] used the framework from Stol et al. [131] and successfully assessed the inner source practices between two small development teams. Riehle [111] created an example IS charter to guide the companies adopting IS. InnerSource Common developed a maturity model pattern\(^3\) to help teams to self-assess their status with regards to different practices - i.e., at what level the teams are following different IS practices. Eckert et al. [41] also proposed a maturity model to assess IS implementation in a large medical diagnostics corporation.

\(^2\)https://innersourcecommons.org/
\(^3\)https://patterns.innersourcecommons.org/p/maturity-model
In this study, we selected the maturity model pattern as our base to develop a questionnaire instrument to help the case company assess the current and desired IS and reuse practices and identify the improvement areas. We chose the maturity model pattern because it covers more areas compared to the existing frameworks [49, 131], charter [111] and maturity model [41]. In addition, we conducted a follow-up workshop to help the company prioritize the identified improvement areas based on the importance and costs to the company. During the workshop, we also collected the potential solutions for the prioritized improvement areas.

5.3 Research methodology

We performed a case study to investigate the case company’s need to adopt IS practices and identify the InnerSource-related practices to improve internal reuse. We used a questionnaire and a workshop to collect the data.

5.3.1 Research questions

To guide the study, we formulated the following research questions (RQs):

RQ1: Which InnerSource-related improvements are needed to enhance the development, maintenance, and ownership of the reusable assets in the case company?

RQ2: Which identified InnerSource-related improvement areas should be prioritized and how to implement them?

RQ1 aims to identify those InnerSource-related improvement areas that the study participants desire to implement in the case company for improving the development, maintenance, and ownership of reusable assets. RQ2 aims to develop and use a mechanism to prioritize the improvements identified in RQ1. Furthermore, RQ2 also focuses on identifying the specific solutions for implementing the prioritized improvement areas.

5.3.2 Case company and unit of analysis

The case company is S-GROUP Solutions⁴, which is a medium-sized Swedish company [2]. The company develops geographical information systems (GIS) to help digitize the city development plan, traffic control, water and sewage. The targeted customers are mainly local governments and authorities.

⁴https://www.sgroup-solutions.se/
At the time of the study, the company had three development teams responsible for four solution areas. Each team has about five developers, a corresponding project manager, a product owner, and a tester. The development teams are based in Sweden and Lithuania. The case company works with different lead roles, such as lead web developer and lead backend developer, to guide the reuse practices. The development teams follow agile practices, use Azure DevOps and perform continuous integration and delivery (CI/CD). S-Group Solutions AB is migrating some of its monolithic proprietary software to a microservices-based architecture for better reuse. The reusable code assets in the case company are npm packages and NuGet packages. We focus on analyzing the internal software reuse practices, especially the collaboration among teams. The case company is in the initial phases of the reuse journey, and two development teams are active in reuse collaboration, and we refer to them as Team 1 - producers and Team 2 - consumers in this paper. Furthermore, the case company aims to expand the reuse collaboration to Team 3 - consumers in the future. In our previous study, the case company also showed interest in adopting IS practices to improve the development, maintenance, and ownership of reusable assets.

5.3.3 Data collection

We used two methods to collect the data: a questionnaire and a workshop.

Questionnaire

**Questionnaire design:** We used the questionnaire to collect the practitioners’ views about the case company on the current and desired state of IS and reuse practices. The questionnaire consists of a set of questions that helps gather information more quickly and cost-effectively [128]. We followed the survey guidelines from Linåker et al. [74] to create the questionnaire. The questionnaire consists of two parts: demographic questions and reuse-specific questions.

The demographic information includes the role, the team information, working experience, and the involvement of the reuse practices (development, consuming or maintaining reusable assets). We followed the IS maturity model pattern to check the company’s readiness to adopt IS as explained in Section

---

5 https://www.npmjs.com/
6 https://www.nuget.org/
5.3 Research methodology

Table 5.1: Proposed IS readiness instrument: categories and areas

<table>
<thead>
<tr>
<th>Categories</th>
<th>Areas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assets</td>
<td>Code repository (all code), reusable code repository, documentation of reusable assets, discoverability of reusable assets, support for contributing/maintaining, the reusable code assets, plans/roadmaps for all projects, plans/roadmaps for reusable assets, sprint/release planning for reusable assets, reusable test cases, traceability for reusable assets, other knowledge</td>
</tr>
<tr>
<td>Process</td>
<td>Code review process of reusable assets, ownership and maintenance of the reusable assets, continuous integration of reusable assets</td>
</tr>
<tr>
<td>Collaboration</td>
<td>Communication channels</td>
</tr>
<tr>
<td>Measurement</td>
<td>Measuring/Monitoring reuse</td>
</tr>
<tr>
<td>Rewards</td>
<td>Rewards for contributing to reusable assets</td>
</tr>
<tr>
<td>Culture</td>
<td>Attitude about collaboration on reusable assets, managers’ views on reuse collaboration</td>
</tr>
</tbody>
</table>

5.2. The authors are familiar with the company context from previous work on the project. The first and second authors customized the IS maturity model pattern based on the case company’s reuse context and its scale: medium-sized company. In total, we have 19 areas for the IS and reuse practices, covering six categories: reusable assets, reuse process, reuse collaboration, the measurement for reuse, rewards for reuse, and culture regarding reuse (see Table 5.1). Each area has two forms, asking respondents about the current and desired status of software reuse (see an example in Figure 5.1). Under each area, we have four options, ranging from Status 1 to Status 4. The lower status number means the area is less systematic and less mature. The higher status number means the area is more systematic and has additional requirements. We asked about the current and desired status of the IS and reuse practices per area so that the participants do not need to read the four options twice. We also provided an open text field for respondents to comment on for each area. We also provided definitions for specific terms to help respondents understand their meaning.

The third author reviewed the questionnaire based on the suitability of the study, the distinguishability and the understandability of the scale, and the inclusion relationship between the categories and areas. We addressed the disagreements in a joint meeting. We also send the questionnaire to the contact person - the project manager, for an expert review. Once we got the approval, we emailed the questionnaire invitation to the candidates. We used Google Forms
Category - Assets (code assets)

Questions about code repository (ALL CODE IN GENERAL)

What is the CURRENT STATUS in the company? Please select one of the following four options.

- Status 1: The teams have their own code repositories, which are not shared with others.
- Status 2: The teams have their own code repositories and they share it with certain stakeholders outside their team.
- Status 3: The teams have centralized code repositories and anyone in the organization can ask for access.
- Status 4: Teams have centralized code repositories, which, by default, are accessible to everyone in the organization.

What should be the DESIRED STATUS in your view? Please select one of the following four options.

- Status 1: The teams have their own code repositories, which are not shared with others.
- Status 2: The teams have their own code repositories and they share it with certain stakeholders outside their team.
- Status 3: The teams have centralized code repositories and anyone in the organization can ask for access.
- Status 4: Teams have centralized code repositories, which, by default, are accessible to everyone in the organization.

If you have any additional comments (e.g., if you cannot select only one option, or if the options are not clear) about the above questions/options, you can add them here.

---

Figure 5.1: Questionnaire example

as our questionnaire tool. The questionnaire\(^7\) was estimated to take about 25 -30 mins. The questionnaire was open for about a month (from January 11th, 2022, to February 9th, 2022).

**Questionnaire participants selection:** The project manager had an internal discussion with the head of system development and identified the producers and consumers of the reusable assets are potential candidates for the questionnaire. In total, we identified eight questionnaire candidates, of which seven answered the questionnaire. We assured the candidates that we would protect their personal information and use the data in an aggregated format for analyzing and reporting.

---

\(^7\)The questionnaire is available at https://doi.org/10.5281/zenodo.7849269.
Workshop

Workshop design: The workshop aims to discuss and prioritize the improvement areas selected from the questionnaire results and brainstorm possible solutions. We conducted a workshop so the participants could share their opinions and also discuss and exchange ideas among multiple stakeholders. We followed the guidelines from Brem [22] and planned a 90 mins workshop. The workshop took place at the university the research group works since the external environment influences the practitioners less and thus allows more ideation [22].

To prioritize the selected improvement areas, we prepared a two-dimension scheme - importance and cost. The importance dimension represents how important the improvement area is for the case company. The cost dimension reflects how costly it is for the case company to implement the solutions for the improvement area. We used a four-point Likert scale for each dimension. The first author also identified solutions from the literature, such as InnerSource patterns\(^8\), to aid the discussions, which the second author reviewed.

Before the workshop, each participant received a card book comprising several paper cards. Each paper card has two questions about one selected improvement area related to the importance and the cost. Our workshop has three phases as follows:

Phase 1: We explained our workshop’s purpose and process to the participants. Moreover, we gave a brief presentation about the questionnaire results, explaining to the participants why the areas were selected for the workshop. Phase 2: We collected the participants’ views on the importance and the costs to the company for each selected area. For every selected area, we started with a group discussion about its questionnaire results so that the participants could have a shared understanding of the selected area and brainstorm the corresponding potential solutions. We had 10 mins for each area. Once the time is up, the participants are required to give their rates on the paper card. The first author collected the paper card and plotted the paper cards on the whiteboard according to the answers. The plot diagram showed how different participants viewed the area, and the participants could give their reflections based on the results. Phase 2 ends when all the selected areas have been discussed and prioritized. We also encouraged the participants to lead the discussion to make them more involved. Phase 3: We asked the feedback from the participants on the workshop activities and summarized the workshop.

\(^8\)https://patterns.innersourcecommons.org/
Workshop participants selection: As described previously, we aim to prioritize the improvement areas according to their importance to the case company and the costs of implementing the solutions. Therefore, it is essential to include both technical and management views. We suggested to the project manager a list of roles and teams and requested him to include as many participants as possible. Due to the candidates’ availability, we eventually got five participants - four developers and one project manager, who were all from Team 1. In total, we had seven participants, including the five selected participants from the case company and the first and second authors from the research group. The first and second authors only acted as a note-taker and a moderator during the workshop. Before the workshop, we asked the project manager to share the questionnaire results with the workshop candidates.

5.3.4 Data analysis

Questionnaire data analysis

We followed the principles of survey research proposed by Kitchenham and Pfleeger [64] to analyze the questionnaire results.

First, we validated the questionnaire results and sorted out the incomplete answers. Then, we investigated why the participants left the answer empty and decided whether to include their data for later analysis. Second, we used the frequency analysis method to count the number of different statuses selected for each area and evaluate the company’s overall situation. Third, we partitioned the responses based on the reuse role of the participants - producers and consumers, to evaluate which areas different roles were satisfied with or wanted improvements. Satisfied with the current status means that the respondents selected the same options for the current and desired status, and wanted improvements means that the respondents selected higher-level status for the desired one than the current.

To reduce the researchers’ bias, we presented the questionnaire results to the project manager and asked for feedback. The project manager agreed with the results related to the questionnaire and agreed to conduct a workshop for a detailed improvement areas discussion.

Workshop data analysis

We prioritize the selected improvement areas based on the two-dimension scheme. For the proposed solutions, we identified the problems and analyzed the context. We shared the workshop results in the form of a report with the project
manager, and he commented that the report concluded the workshop without missing any important information.

## 5.4 Results

This section provides the results of the questionnaire and the workshop.

### 5.4.1 RQ1

We received seven out of eight potential participants’ responses. Table 5.2 shows the questionnaire participants’ demographics and experience in the case company. The main reusable asset is code. Few developers also reuse requirements and test cases. We got two responses that had incomplete answers. One respondent left one question empty since he was not involved in such reuse practices. The least experienced (four months) respondent did not answer eight areas, and we understand that it takes time for a newly recruited employee to grasp the company software practices. Each area stands for different IS and reuse practices, and the answers will not interfere with other areas, so we decided not to remove any data from the analysis. In the following subsections, we present the overview of the company IS and reuse practices status (both current and desired) and then detail the findings by partitioning the answers into different categories.

### Table 5.2: Questionnaire participants’ demographics and experience in the case company

<table>
<thead>
<tr>
<th>Participants ID</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role</td>
<td>Developer</td>
<td>Developer</td>
<td>Lead web developer</td>
<td>Developer</td>
<td>Developer</td>
<td>Tech lead developer</td>
<td>Developer</td>
</tr>
<tr>
<td>Team</td>
<td>Team 1</td>
<td>Team 1</td>
<td>Team 1</td>
<td>Team 1</td>
<td>Team 2</td>
<td>Team 2</td>
<td>Team 3</td>
</tr>
<tr>
<td>Experience</td>
<td>8 years</td>
<td>5 years</td>
<td>4.5 years</td>
<td>4 months</td>
<td>6 years</td>
<td>14 years</td>
<td>8 years</td>
</tr>
</tbody>
</table>

### Overview of the current IS and reuse practices status

Figure 5.2 - left side, shows the overview of participants’ views on the current status of the reuse practices in the case company. We used gradient colors to present different statuses; lighter colors represent lower status, and darker colors represent higher status. The number on each bar represents the number of respondents who selected the corresponding status. Most areas have seven
responses, except for the eight areas, which have one or two responses less. In addition, we sorted the areas in order of status numbers - from high to low.

The following seven areas are less systematic and less mature according to the respondents’ answers - the number of Status 1 and Status 2 responses is more than the number of Status 3 and Status 4: rewards for reuse contribution, measuring/monitoring reuse, reusable test case, attitude about reuse collaboration, communication channels for reuse, sprint/release planning for reusable assets, and discoverability of reusable assets. On the other hand, the following four areas are relatively more mature according to the respondents’ answers: reusable code repository, plans/roadmaps for all projects, other knowledge and continuous integration of reusable assets. Moreover, continuous integration of reusable assets is appropriately taken care of based on the responses. We also found three areas of the most diverse answers that need company-wide consensus: communication channels for reusable assets, sprint/release planning for reusable assets, and plans/roadmaps for reusable assets.
5.4 Results

The results indicate that currently, the transparency of the reusable code assets and the plans/roadmaps for overall projects are relatively mature. To increase the reuse collaboration between teams, the case company needs to focus on increasing the reuse incentive, building communication bridges between teams, monitoring the internal reuse progress, and enhancing the discoverability and maintenance of the reusable assets.

**Overview of the desired IS and reuse practices status**

Figure 5.2 - right side, shows the overview of participants’ views on the desired status of the IS and reuse practices in the case company. All respondents seek higher status for the IS and reuse practices, except for the rewards for reuse contribution. There are five areas - four about the assets and one about the process, that all respondents agreed to have the highest status: **documentation of reusable assets, discoverability of reusable assets, plans/roadmaps for all projects, other knowledge** and **continuous integration of reusable assets**. Respondents held the most different opinions on six areas: **reusable code repository, code review process of reusable assets, code repository (all code), manager’s view on**
reuse collaboration, attitude about reuse collaboration and measuring/monitoring reuse, which indicates a need of shared understanding.

The results indicate that the respondents want to achieve systematic and mature IS and reuse practices in the future for most areas. Compared to the current status of the IS and reuse practices, we can see that there are many areas that the respondents want to improve.

Producers’ and consumers’ view on the IS and reuse practices

Figure 5.3 reflects the participants’ views on what the case company needs to improve to achieve better IS and reuse practices. We used two colors to represent the producers and the consumers, and the data label represents the number of participants. The data are divided into two parts: satisfied with the current status and wanted improvements. We ranked the areas according to the number of participants who wanted improvement - from the largest to the smallest. A significant proportion of participants wanted improvements in support for contributing/maintaining the reusable code assets, documentation of reusable assets, plans/roadmaps for reusable assets, discoverability of reusable assets, communication channels for reuse, other knowledge and measuring/monitoring reuse. They are more satisfied with the reusable code repository, rewards for reuse contributions, continuous integration of reusable assets and code repository (all code). In addition, there are three areas that the participants voted equally in having changes in the future or staying as it is: traceability for reusable assets, attitude about collaboration on reusable assets and the managers’ view on reuse collaboration.

The equal number of votes drove us to further break down the answers into different roles. We found that producers wanted more improvements in IS and reuse practices than the consumers, e.g., ownership and maintenance of the reusable assets and measuring/monitoring reuse. Consumers are more eager to see the changes in support for contributing/maintaining the reusable code assets. For the areas of plans/roadmaps for all projects and sprint/release planning for reusable assets, we got the same proportion of producers satisfied with the current situation and wished for improvements. All producers wanted more management support, while all the consumers were pleased by the current manager’s status. If we only focus on the producers, we found five more improvement areas: traceability for reusable assets, attitude about reuse collaboration, manager’s view on reuse collaboration, ownership and maintenance of the reusable assets and reusable test cases.
5.4 Results

Figure 5.3 shows that the producers are less satisfied with the current situation than the consumers. Except for the areas in that all respondents wanted changes, the producers also wanted to see changes in the other four areas – reusable test cases, traceability for reusable assets, ownership and maintenance of the reusable assets and manager’s view on reuse collaboration.

According to the questionnaire results, we sorted out nine areas that most participants wish to have changes and aim for a higher status in the future - initially, we identified eight areas. We added the area of ownership and maintenance of the reusable assets since it is closely related to the top selected area: support for contributing/maintaining the reusable code assets. The identified improvement areas cover all categories except rewards. We present the nine areas and their corresponding categories as follows:

Assets: Documentation of reusable assets, Discoverability of reusable assets, Plans/Roadmaps for reusable assets, Support for contributing/maintaining the reusable code assets, and Other knowledge.

Process: Ownership and maintenance of the reusable assets.

Collaboration: Communication channels for reuse.

Measurements: Measuring/Monitoring reuse.

Culture: Manager’s view on reuse collaboration.

We selected six areas for later prioritization and omitted areas related to the documentation, other knowledge, and the manager’s view because of the following reasons:

- Documentation and other knowledge will be reflected in discoverability. In addition, we were informed by the case company that they have partially addressed the documentation problem.

- The current status for the manager’s view is already in a relatively high-level stage.

5.4.2 RQ2

The workshop took approximately 100 mins. According to the questionnaire results, we selected six areas for prioritization. During the workshop, support for contributing/maintaining the code of reusable assets and ownership and maintenance of the reusable assets were discussed together and merged into one area as contribution/maintenance support.

The workshop results show that the top prioritized areas are discoverability of reusable assets, communication channels for reusable assets and contribution/
We present the prioritized results based on importance, costs and both in the following subsections (see Figure 5.4).

Figure 5.4a and Figure 5.4b show the prioritization results of the selected six areas based on importance and cost. According to Figure 5.4a, there are no areas that were considered less important by the workshop participants. The majority thought discoverability of reusable assets was the most important one and communication channels for reusable assets were the second. Plans/Roadmaps for reusable assets and contribution/maintenance support are in third place and have the same prioritization rate. Based on Figure 5.4b, there are no areas perceived as very costly by the workshop participants. The majority thought contribution/maintenance support had the lowest cost. The discoverability of reusable assets, measuring/monitoring reuse, and communication channels for reusable assets had a similar cost level - the second place. Plans/Roadmaps for reusable assets had relatively higher costs than the others.

Figure 5.4c shows the final plot results for all the selected areas. The results show that all participants had similar opinions on each improvement area. We did not ask the participants to finalize their answers to a single opinion since the workshop’s purpose is to help participants discuss the differences and have a shared understanding of the improvement areas. In addition, the differences between answers are not drastic. Overall, we can see there are no areas that were less important and very costly to implement. If we focus on the four cells in the bottom right corner, we can see that discoverability of reusable assets, communication channels for reusable assets, and contribution/maintenance support were the most important to the company and less costly to fix, which indicates they are potentially the low-hanging fruits to initiate.

During the workshop, we also asked the participants for potential solutions for each selected area. In the following subsections, we will first elaborate on the problems in the reuse area identified in the company and explain the context. Then we will provide the solutions that the respondents provided. We discuss the connection between the proposed solutions and existing IS reuse practices in Section 5.5.2.

**Discoverability solutions**

**Problem:** The developers do not have a clear picture of which reusable assets exist in the company and how to use them. **Current situation:** 1) All the reusable packages are stored in the Azure DevOps repository. 2) There are two types of reusable packages: npm packages are properly documented and have associated readme files, while NuGet packages are not. 3) It is easier to
Figure 5.4: Workshop prioritization results
identify the reused npm packages since they are React packages and are more visible in reuse. However, NuGet packages are more backend packages and less visible in reuse.

Solutions proposed by the participants:

1. The company should collect the information and knowledge about the repository, such as creating a list of existing (and planned) reusable assets, and briefly describe what they do and where they are located by adding necessary links.

2. The repository should be restructured so that the developers can find the reusable assets easier, e.g., in addition to the basic search function, the repository could have some filtering functions based on, such as the type of reusable packages.

3. The developers should enhance the documentation of the NuGet package, such as readme files and changelogs.

4. The developers should enhance the traceability between the reusable assets and the application and modules that reused them.

Plans/roadmaps for reusable assets solutions

Problem: The teams do not have plans/roadmaps for reusable assets or plans for the reusable assets upgrade in existing applications. Current situation: 1) The reusable assets are initiated by developers when they realize the assets can be reused in multiple applications. 2) The company has a shared prioritized backlog; however, it does not include the reusable assets plan. 3) There is no policy for upgrading the reusable assets in existing applications.

Solutions proposed by the participants:

1. Instead of a bottom-up strategy (developers initiate the development of the reusable assets), the product owner and project leader should plan the reusable assets for development based on the requirements.

2. The team should have a plan for upgrading the reusable assets in the existing applications to the latest version, even though it will cost more in testing.

3. There should be an owner or manager to maintain the roadmaps of the reusable assets.
5.4 Results

Contribution/Maintenance support solutions

Problem: When there is a need for a fix, where should the developer modify — the reusable code assets or application? The owner of the reusable assets is unknown. Even though the contribution and maintenance guidelines (step-by-step guide) are in place, people still have many issues. No clear roles and responsibilities are defined. The teams lack knowledge sharing about reusable assets. Current situation: 1) It depends on the developers to decide whether to fix bugs in the application or modify the reusable assets. 2) Team 1 is responsible for developing npm packages, and Team 2 is responsible for developing NuGet packages. 3) It takes effort for developers to find the right person for contribution and maintenance support. 4) For the code review, one developer from the corresponding team should review the pull request. For the reusable assets code review, two developers responsible for different solution areas should review the pull request.

Solutions proposed by the participants:

1. The reusable assets should have an owner responsible for their maintenance and review.

2. The company should prepare and enhance the review process for later when reuse scales up. For example, write a clear policy about who should review the reusable assets.

3. The reusable assets information should be written in the readme file, and someone should be responsible for maintaining it.

4. The developers should write the contact person in the readme file.

Communication channels for reusable assets solutions

Problem: The company lacks communication between the teams when creating reusable assets. Current situation: 1) No cross-team communication for the reusable assets. 2) The company uses Teams and webhooks for communication within and across teams. 3) The pull request is announced in Teams. 4) It is hard to find the changelogs which are stored in the wiki.

Solutions proposed by the participants:

1. When there is a new release for the reusable packages, developers should use Teams to notify others.
2. The reusable assets might be released multiple times in a day and have many versions. To save time, the developers should only communicate the major changes.

3. The reusable assets should have a guardian so the stakeholders know whom to contact.

4. The wiki changelogs should have a better search facility.

5. The Teams channels should have an easy-follow structure for storing readme files and communicating the new major updates.

Q16 Measuring/Monitoring reuse solutions

**Problem:** The company lacks information about reuse measurement to track the progress of reuse. **Current situation:** Around two years, the company used the reuse rate (lines of code for reusable assets/lines of code for the application) to measure the reuse progress.

**Solutions proposed by the participants:**

1. The developers should tag the bugs related to the reusable assets to enable future measurements.

2. The developers should have traceability links between reusable assets and the applications that reuse them.

5.5 Discussions

Section 5.5.1 discusses the identified improvement areas and compares our instrument with related works. Section 5.5.2 discusses the possible improvement solutions with related works, and Section 5.5.3 presents the threats to the validity of this study.

5.5.1 The improvement areas

According to the questionnaire results, the current IS and reuse practices in the case company are not systematic, and respondents seek higher status in most areas, except rewards. In addition, we found that the producers wanted to improve more areas than the consumers. That is because the case company is at the early stage of the IS and reuse journey. The producers are more involved in the IS and reuse practices than the consumers.
5.5 Discussions

Although in the case company, all developers have access to the reusable code assets, discoverability is still a challenge - developers face problems in knowing which reusable assets exist in the company and how to search for the needed ones. In addition, to increase the reuse contribution and facilitate team collaboration, the case company needs to improve the transparency of the reusable assets plans/roadmaps, the contribution support, ownership of the reusable assets, and build communication channels between teams. The reuse measurements need enhancements so that the case company understands the benefits of doing IS and reuse practices.

The participants understand the current situation regarding IS and reuse practices covered by our questionnaire instrument. However, low-level status does not mean the area needs improvements since it might be the best situation according to the company’s needs. Therefore, we introduced the concept of the current and desired status in our instruments, which helped the case company to identify the needed improvement areas by understanding also where and what they want to achieve.

Existing frameworks [49, 131], charter [111] and maturity model [41] can help companies understand their IS context. Like our instrument, the above related works comprise the IS areas related to shared assets/transparency, process, collaboration, culture and incentive. For shared assets, only Stol et al. [131] and Gaughan et al. [49] mentioned the selection of the IS seed product. We have reusable assets as the initial seed product in the case company. Therefore, we omitted this part. Stol et al.’s framework [131] and the IS maturity model pattern mentioned the standardization of tools. However, our instrument did not include tool standardization since the case company is medium-sized and teams use the same tools. We think it is necessary to consider tooling standardization when adopting IS in large-scale companies that use many different tools. Existing works also investigate the transparency of reusable assets. In addition, our instrument asks about the discoverability and the traceability of reusable assets, which helps the practitioners understand more about the existing reusable assets in the company and how to search for them. Collaboration is more related to the communication channel, especially between teams. The culture consists of the managers’ and individuals’ views on IS reuse collaboration. In addition to the related works, we asked the case company about measurements in reuse monitoring. Such measurement is important since it provides persuadable facts to the managers about the benefits of software reuse and IS collaboration.
5.5.2 InnerSource solutions

The case company aims to create a centralized shared space for reusable assets, with the necessary documentation, good search facilities, and tracking functions for improving discoverability. Such infrastructures are widely used in large companies and resulted in better internal reuse, such as Hewlett-Packard Company [38], SAP [113], Nokia [76], Ericsson [140]. We found the infrastructure systems reported in existing literature are usually web-based, have good search facilities, and contain all reusable code assets and documentation, wiki page, mailing lists, and monitoring metrics. For example, Ericsson’s marketplace [140] advertises the top contributors on the homepage to encourage more contributions, categorize the microservices according to the maturity level, and has a good search and filtering mechanism. Companies can learn from the existing infrastructure systems and customize them according to their needs. Linden [75] also suggests using open source software to reduce the effort of building the infrastructure systems.

Communication channels are essential for IS practices, especially when communicating the changes. To reduce the impact of significant changes, we suggest utilizing the existing communication tools to achieve the desired goals. Currently, the case company uses Microsoft Teams and its channels to communicate updates, pull requests, and releases within the teams. We noticed from the questionnaire results that teams had different opinions on communication tools. Therefore, a company-wide communication channel is also needed. Besides, the communication results should be logged systematically and regularly maintained, especially for the decisions. Mailing list and wiki can be used to facilitate company-wide communication and log the necessary documentation [132]. In addition, specific roles [111] should be introduced to maintain the documentation.

So far, the case company did not have major problems in contribution and maintenance support. However, the participants are concerned about the contributions and maintenance when IS and reuse scale up to all teams. The participants found that the contributor tends to contribute without reading the readme file, resulting in more problems. We suggest the practitioners learn from the Standard Base Documentation pattern 9 developed by InnerSource Commons. InnerSource Commons also stated that explicit roles such as guardian and trusted committers would ease communication. InnerSource Common also

---

9https://patterns.innersourcecommons.org/p/base-documentation
developed a 30-Day Warranty pattern ¹⁰ to deal with the issues from the contributed work.

The case company used the reuse rate to measure the reuse progress. To improve that, the participants suggested tracking the applications that used reusable assets to increase traceability and identify the dependencies. In addition, measuring defects for reusable assets could also enhance the quality of the reusable assets and reduce the risks for other applications to use them. Capraro et al. [27] developed a patch-flow method to measure the IS collaboration. Buchner and Riehle [24] used metrics related to worked time to calculate the costs of IS collaboration.

Plans/Roadmaps for reusable assets are not only limited to identifying or designing the reusable assets but include the maintenance of the old versions of the reusable assets. To enhance the planning, the participants suggested involving the product owners and project managers in planning the reusable assets, adding views from both customers and project managers. The tracking from the reuse monitoring helps developers identify the projects with reusable assets, making the maintenance plan easier.

We also found solution overlaps between improvement areas. For example, discoverability and contribution/maintenance support both involve improving information provided in readme files. Traceability improvements between reusable assets and applications that use them are required in both improvement areas of discoverability and communication channels. Such overlap offers a starting point for determining which solutions to prioritize.

### 5.5.3 Threats to validity

We followed the four scheme validity from Runeson and Höst [118]: construct validity, internal validity, external validity, and reliability.

**Construct validity:** The construct validity demonstrates whether the studied operational measurements answered the research questions and performed as the researchers expected. The questionnaire was developed iteratively by the first and second authors and reviewed by the third author and the project manager from the case company. We provided definitions for specific terms used in the questionnaire so that the respondents understood the meaning. We also provided open text fields in the questionnaire so the respondents could comment on each area. In addition, we presented the questionnaire results before the

¹⁰https://patterns.innersourcecommons.org/p/30-day-warranty
workshop so that the participants understood how we selected the improvement areas for prioritization.

**Internal validity:** We have not investigated the causal relations in this study. To ensure the findings are valid and consistent, we asked the study participants - the project manager and the workshop participants, to review the questionnaire and workshop results multiple times.

**External validity:** The external validity represents the generalizability of the findings. In our study, we only applied the instruments in one medium-sized company. However, we described the company context in detail. We only included one team in the workshop since the other teams were not available. The proposed solutions are perceptions from producers only. Companies similar to the case company context can reuse our instrument to assess their company situation for IS and reuse practices. Companies could also learn from our findings, especially the proposed IS solutions.

**Reliability:** The reliability validity refers to the researchers’ bias on the data and the analysis. The first and second authors developed the questionnaire instrument based on the IS maturity model pattern. The third author and the project manager reviewed the instrument. We presented the questionnaire results to the project manager, who agreed that the results reflected reality. The questionnaire results are presented again to the workshop participants to help them understand how we selected the improvement areas for prioritization. We presented the workshop results at the end of the workshop and shared the results in the form of a report with the project manager. In addition, the project manager also reviewed this paper which did not lead to any major changes. Though two authors were involved in the workshop, they only acted as facilitators and had little influence on the discussions and the prioritization results.

### 5.6 Conclusion and future work

In this study, we developed and used an instrument to help the case company assess its readiness to adopt IS for improving the development and maintenance of the reusable assets.

The customized instrument helped assess the current and desired status of the IS and reuse practices and identify the improvement areas. The highest InnerSource status may not be the best case for companies in adopting InnerSource practices for reuse. The identified improvement areas for the case company are documentation, discoverability, support for contributing/maintaining the reusable code assets, plans/roadmaps for reusable assets, other knowledge,
ownership and maintenance of the reusable assets, communication channels for reuse, measuring/monitoring reuse, and managers’ view on reuse collaboration. The prioritized improvement areas are discoverability, communication channels, and ownership and maintenance of the reusable assets. The possible solutions indicate that IS practice can help improve the internal reuse of the case company.

In the future, we plan to investigate which IS solutions the case company adopted and measure whether IS helps to improve internal reuse using productivity and quality related metrics. In addition, we also want to enhance the generalizability of our questionnaire instrument by applying it to more companies that wish to adopt IS to improve internal reuse.

Acknowledgement

We acknowledge that this work was supported by the Knowledge Foundation through the OSIR project (reference number 20190081) at Blekinge Institute of Technology, Sweden. We also want to thank the practitioners from the case company for collaborating with us.
References


References


[33] Commons, I. InnerSource Patterns. InnerSource Commons.


[76] Lindman, J., Rossi, M., and Marttinen, P. Open source technology changes intra-organizational systems development—a tale of two companies.


[109] Public Health Resource Unit, v.. *Critical Appraisal Skills Programme*, 2008 (accessed on 06.01.08).

[111] Riehle, D. An example charter for inner source programs.


Context: Companies adopt many software reuse practices, such as software product line, reuse verbatim, and systematic reuse, to improve their internal software development and maximize the benefits. Contemporary software engineering (SE) practices, such as microservices and InnerSource, influence internal software reuse.

Objective: In this thesis, we aim to improve internal software reuse in the context of contemporary SE practices. To do that, we want to 1) understand the state-of-the-art and the state-of-the-practice of software reuse costs and benefits and the challenges that companies are currently facing and 2) identify interventions to improve internal software reuse.

Methods: We conducted a systematic literature review to understand the state-of-the-art of software reuse costs and benefits. We performed two exploratory case studies to understand the state-of-the-practice of software reuse costs and benefits, challenges, and improvement areas in the context of contemporary SE practices. We performed another follow-up improving case study to investigate the medium-sized case company’s readiness of adopting InnerSource for software reuse.

Results: Existing literature reported more software reuse benefits than costs. The most reported software reuse benefits are better product quality and improved productivity. Verbatim reuse and systematic reuse result in more reuse benefits. Most of the included primary studies are of moderate quality, with only four having high quality. Practitioners think that software reuse costs in developing reusable assets will be paid off when developers start to reuse them. Challenges in software reuse in the context of contemporary SE practices differ between medium-sized and large-sized companies. Both of the companies perceive that InnerSource can help improve internal software reuse. Asking practitioners about both current and desired InnerSource reuse status helps identify the needed InnerSource improvements, thus helping companies succeed in adopting InnerSource for reuse.

Conclusion: Both existing literature and our two case studies investigating software reuse in the context of contemporary SE practices showed that software reuse improves quality and productivity and has costs in developing and integrating reusable assets. However, the overall benefits outweigh the costs. Both case companies faced challenges in improving their internal reuse, where the most common challenges were about developing and maintaining reusable assets. The results showed that InnerSource helps develop and maintain reusable assets and further improves internal software reuse.