Understanding and evaluating software reuse costs and benefits from industrial cases—A systematic literature review

Xingru Chen ∗, Muhammad Usman, Deepika Badampudi

Department of Software Engineering, Blekinge Institute of Technology, Sweden

ARTICLE INFO

Dataset link: https://zenodo.org/records/10649264

Keywords:
- Software reuse
- Software reuse costs
- Software reuse benefits
- Systematic literature review

ABSTRACT

Context: Software reuse costs and benefits have been investigated in several primary studies, which have been aggregated in multiple secondary studies as well. However, existing secondary studies on software reuse have not critically appraised the evidence in primary studies. Moreover, there has been a relatively less focus on how software reuse costs and benefits were measured in the primary studies, and the aggregated evidence focuses more on software reuse benefits than reuse costs.

Objective: This study aims to cover the gaps mentioned in the context above by synthesizing and critically appraising the evidence reported on software reuse costs and benefits from industrial cases.

Method: We used a systematic literature review (SLR) to conduct this study. The results of this SLR are based on a final set of 30 primary studies.

Results: We identified nine software reuse benefits and six software reuse costs, in which better quality and improved productivity were investigated the most. The primary studies mostly used defect-based and development time-based metrics to measure reuse benefits and costs. Regarding the reuse practices, the results show that software product lines, verbatim reuse, and systematic reuse were the top investigated ones, contributing to more reuse benefits. The quality assessment of the primary studies showed that most of them are either of low (20%) or moderate (67%) quality.

Conclusion: Based on the number and quality of the studies, we conclude that the strength of evidence for better quality and improved productivity as reuse benefits is high. There is a need to conduct more high quality studies to investigate, not only other reuse costs and benefits, but also how relatively new reuse-related practices, such as InnerSource and microservices architecture, impact software reuse.

1. Introduction

Software reuse helps companies improve product quality [1–6] and development productivity [2–5]. Studies also reported that software reuse helps faster time-to-market [7,8] and better performance [3,9]. Software reuse has benefits but also comes with additional costs. In particular, developing reusable assets may incur additional costs, such as asset design [10] and team coordination [11]. However, these additional costs pay off when the reusable assets are reused in multiple products and solutions [5,11].

Several approaches have been used to implement software reuse, such as software product lines [10,12,13], reuse from repository [4, 14,15] and component reuse [16]. In addition, depending on whether companies have a well-thought-out strategy to develop for and with reuse, software reuse can be classified into systematic, ad hoc, and controlled reuse [10]. According to Tomer et al. [10], systematic reuse refers to the scenario wherein companies have planned to mine and catalog the reusable assets into a shared repository and design them for later reuse. On the other hand, ad hoc reuse (also referred to as opportunistic reuse) represents the cases wherein companies do not have a shared repository for reusable assets, and teams need to look for them and use them independently. Lastly, companies may have a shared repository for reusable assets but have yet to design them for later reuse, referred to as controlled reuse. Studies also investigated different types of reuse, namely verbatim reuse (also referred to as black-box reuse or reuse as-is) and modified reuse (also referred to as white-box reuse).

Primary studies, such as Succi et al. [1] and Morisio et al. [2], concluded software reuse benefits from empirical evidence. Such primary studies were further aggregated and synthesized in secondary studies (e.g., literature reviews or mapping studies) to draw new conclusions or identify patterns on software reuse costs and benefits. Mohagheghi and Conradi [17] performed a systematic literature review (SLR) on industrial quantitative studies investigating the effects of software reuse on quality, productivity, and economics. Barros et al. [20]...
conducted a systematic mapping study (SMS) investigating what software reuse benefits have been brought to the industry. Bombonatti et al. [19] conducted an SMS investigating the relationship between non-functional requirements and software reusability. Younoussi and Roudies [18] conducted an SLR on software reuse with one of their research questions on its benefits.

The aforementioned systematic secondary studies are mainly about software reuse benefits, with a minimal focus on software reuse costs. Furthermore, except Mohagheghi and Conradi [17], none of them [18–20] report how their included primary studies measured the identified software reuse benefits. Likewise, except for Mohagheghi and Conradi [17], their results are not based on industrial cases only, such as reporting practitioners’ general opinions on the usefulness of software reuse practices or studies based on an analysis of some open-source systems. Moreover, Mohagheghi and Conradi [17] included the primary studies up to 2005 in their review published in 2007, which indicates the need for updating their review. Among the four identified systematic secondary studies on software reuse costs and benefits, only Barros et al. [20] investigated the relationship between different reuse practices and reuse benefits, e.g., which reuse methods (systematic reuse and ad hoc reuse) may lead to more reuse benefits. Lastly, none of the existing systematic secondary studies on software reuse costs and benefits performed a rigorous quality assessment on their included primary studies, e.g., assessing the quality of reporting (see details in Section 2). Without a rigorous quality assessment, it is unclear how strong the primary evidence behind the software reuse costs and benefits results reported in these systematic studies. We aim to address these gaps by identifying and appraising the evidence on software reuse costs and benefits reported in the empirical studies based on industrial cases only. To do that, we conducted an SLR with the following specific contributions:

- Identified the software reuse costs and benefits from industrial cases.
- Identified the context and reuse-specific characteristics of the industrial cases reported in the included studies and associated them with software reuse costs and benefits.
- Identified the metrics used in the primary studies to measure software reuse costs and benefits.
- Assessed the quality of the primary study to gauge evidence for software cost and benefit results.
- Identified research gaps in the existing literature and suggested future research directions.

The remainder of the paper is structured as follows: Section 2 presents the related work. Section 3 describes the research methods and the validity threats. Section 4, Section 5, Section 6, and Section 7 answer the four research questions, and Section 8 discusses the findings. Section 9 concludes the SLR and proposes future work.

### 2. Related work

Many studies have reported impacts when conducting software reuse, such as performance decay [21], better product quality [1,7,9,22], and improved productivity [2,16,23,24].

We identified seven secondary studies that are directly related to [17–20] or mentioned [25–27] software reuse costs and benefits. Table 1 summarizes the characteristics of four directly related secondary studies [17–20] together with their corresponding research gaps (note: Gx means Gap no. x). The identified gaps indicate a need for a new secondary study on software reuse costs and benefits. All gaps are discussed in Section 1. Here, we further elaborate on the quality assessment gap (G2). Mohagheghi and Conradi [17] did not assess the quality of the primary studies but discussed the reported validity threats. Younoussi and Roudies [18] performed a quality assessment based on the research questions to check if the primary study contains the needed information. Bombonatti et al. [19] performed quality assessments on primary studies using bibliometric impact information. However, they

<table>
<thead>
<tr>
<th>Related work (RW)</th>
<th>Year</th>
<th>Software reuse costs and benefits focus</th>
<th>Gap (G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW1[17]</td>
<td>2007</td>
<td>Impacts in quality, productivity, and return on investment</td>
<td>G1: the year of the included primary studies is up till 2005. G2: the study lacks quality assessment on primary studies. G3: the study did not discuss the relationship between reuse-specific characteristics and software reuse costs/benefits.</td>
</tr>
<tr>
<td>RW2[18]</td>
<td>2015</td>
<td>Mainly benefits and few costs</td>
<td>G2: the study lacks quality assessment on primary studies. G3: the study did not discuss the relationship between reuse-specific characteristics and software reuse costs/benefits. G4: not all the included primary studies include an industrial case. G5: the study did not provide the metrics used to measure software reuse costs/benefits.</td>
</tr>
<tr>
<td>RW3[19]</td>
<td>2016</td>
<td>Reusability influence on non-functional requirements</td>
<td>G2: the study lacks quality assessment on primary studies. G3: the study did not discuss the relationship between reuse-specific characteristics and software reuse costs/benefits. G4: not all the included primary studies include an industrial case. G5: the study did not provide the metrics used to measure software reuse costs/benefits.</td>
</tr>
<tr>
<td>RW4[20]</td>
<td>2018</td>
<td>Only benefits</td>
<td>G2: the study lacks quality assessment on primary studies. G4: not all the included primary studies include an industrial case. G5: the study did not provide the metrics used to measure software reuse costs/benefits.</td>
</tr>
</tbody>
</table>

Table 1

Secondary studies related to software reuse costs and benefits.
lack aspects related to content quality since the authors only consider the venue rank and citation number. Barros et al. [20] had a research question on validity threats, but the authors only checked whether the validity threats were reported. Thus, it is hard to understand how reliable the evidence is reported in the primary studies.

The main objectives of the other three secondary studies are not on software reuse costs and benefits. However, they mentioned software reuse costs and benefits in their results. Chen and Babar [25] mentioned software reuse benefits when introducing one primary study about the effect of the variability management approach. Neto et al. [26] mentioned software reuse benefits when answering the research question about the testing strategies adopted in software product line testing approaches. As for Flemstrom et al. [27], they mentioned vertical test reuse benefits when answering the research question about the justification or motivation for using vertical test reuse.

Our study aims to investigate the state of the art of reported software reuse costs and benefits from industrial cases published until 2021. We want to identify how software reuse costs and benefits were measured in the primary studies and understand which reuse practices benefit companies more. In addition, we plan to assess and analyze the strength of the evidence behind the software reuse costs and benefits claims in the included primary studies.

3. Research methodology

We followed the systematic literature review guidelines proposed by Kitchenham et al. [28] to conduct our study.

3.1. Research questions

We addressed the following research questions (RQs) in this study:

RQ1: What are the characteristics of the primary studies on software reuse costs and benefits?

RQ2: What are the software reuse costs and benefits reported in the primary studies, and how are they related to reuse practices?

RQ3: How are software reuse costs and benefits measured in the primary studies?

RQ4: What is the strength of reported evidence on software reuse costs and benefits in the primary studies?

The characteristics help refine the understanding of how companies practiced software reuse in the included primary studies. The characteristics include general information about the included primary studies (e.g., year and venue), contextual characteristics (e.g., development methodology), and reuse-specific characteristics (e.g., reuse types).

3.2. Search and study selection processes

We used a combination of automated and snowballing search strategies. Snowballing search strategy needs a good start set [29], which we identified from the automated search strategy. To identify the primary studies on software reuse costs and benefits using automated search, we tried Scopus using the keywords “software reuse”, “cost, benefit” and retrieved 3482 primary studies. It was challenging to manage such a large number of primary studies. Therefore, we decided to follow an indirect approach — extract the primary studies that secondary studies included to synthesize software reuse costs and benefits. To make our search more inclusive, we first used automated search to identify secondary studies on software reuse. And within the identified secondary studies on software reuse, we included the ones about software reuse costs and benefits. Then, we extracted all relevant primary studies on software reuse costs and benefits from the included secondary studies. We followed the snowballing guidelines from Wohlin [29] to identify more primary studies on software reuse costs and benefits. To conclude, we performed the search in three phases (see Fig. 1).

• Phase 1: Identification of secondary studies using automated search.

• Phase 2: Identification of the primary studies from the secondary studies identified in Phase 1.

• Phase 3: Forward snowballing on the primary studies identified in Phase 2.

3.2.1. Phase 1: Identification of secondary studies using automated search

In Phase 1, we aimed to identify all secondary studies on software reuse. We identified the search string keywords - “software reuse”, “literature review”, and “mapping study” and ran the automated search in four databases: Scopus, IEEE, Web of Science, and ACM. We conducted the search string execution in May 2022 and limited the year to 2021. After removing all the duplicates, we retrieved 427 secondary studies on software reuse.

To avoid missing relevant secondary studies on software reuse and validate the reliability of the automated search results, we included two tertiary studies on software reuse [30,31] as an additional data source. Ultimately, we had 481 secondary studies on software reuse.

3.2.2. Phase 2: Identification of the primary studies from the secondary studies identified in Phase 1

Phase 2 consists of two steps — Step 1: identify secondary studies on software reuse costs and benefits from the identified studies in Phase 1, and Step 2: identify primary studies on software reuse costs and benefits from the identified studies in Step 1.

In Step 1, after all authors had a shared understanding of Secondary Study Selection Criteria (see Section 3.3), we identified the criteria on the title and abstract of ten randomly selected secondary studies, resulting in no disagreements. Subsequently, the first author independently applied the criteria to all secondary studies and included 79 secondary studies and seven doubtful secondary studies. The other two authors reviewed the selection results, including 81 secondary studies on software reuse. The first author independently read each research question (RQ) from the included secondary studies and identified 13 relevant secondary studies on software reuse costs and benefits. All authors independently read the identified 13 secondary studies. In a joint meeting, we discussed the RQs and agreed to include seven secondary studies on software reuse costs and benefits [17–20,25–27], which four [17–20] are directly on software reuse costs and benefits and three [25–27] have at least one RQ about software reuse costs and benefits.

In Step 2, we extracted 146 relevant primary studies, which were used to answer RQs about software reuse costs and benefits from the included seven secondary studies in Step 1 after removing the duplicates. We developed Primary Study Selection Criteria (see Section 3.3) to include primary studies investigating software reuse costs and benefits in industrial cases and reached a shared understanding of the criteria through a joint discussion. The first and second authors piloted Primary Study Selection Criteria on ten randomly selected primary studies, finding no disagreements in the pilot results. The first author read the full text and independently applied Primary Study Selection Criteria to all candidate primary studies, including 29 primary studies and eight doubtful primary studies. The second author reviewed the selection results, and ultimately, we included 29 primary studies on software reuse costs and benefits.

3.2.3. Phase 3: Forward snowballing on the secondary and primary studies identified in Phase 2

In Phase 3, we conducted snowballing on the included four secondary studies directly on software reuse costs and benefits, and 29 included primary studies from Phase 2.

Since the primary studies are extracted from the secondary studies, which went through a thorough search strategy, we applied forward

---

1 The detailed search string for each database is available at https://zenodo.org/records/10649264.
snowballing to cover the missing years from the included secondary studies. The included secondary studies include the primary studies published until the end of 2016, resulting in missing years from 2017 to 2021. We used Google Scholar as our forward snowballing search engine because of its comprehensive coverage. The new snowballing iteration starts when we finalize the included primary studies in the previous iteration. The snowballing iteration ends when no new studies are found.

We had two iterations and retrieved 360 primary studies and 25 new secondary studies in Iteration 1 and three primary studies in Iteration 2. The first author read the full text and independently applied the selection criteria to the retrieved secondary and primary studies. We included one more primary study from forward snowballing. Eventually, we included 30 primary studies on software reuse costs and benefits.

3.3. Selection criteria

In our study, we developed two selection criteria to identify secondary and primary studies on software reuse costs and benefits. For the Secondary Study Selection Criteria, we included the secondary studies that directly investigated software reuse, followed a systematic and structured process (i.e., SLR and SMS), were written in English, were not duplicated, and were published until 2021.

For the Primary Study Selection Criteria, we included the primary studies meeting all the following criteria.

1. Studies that are published in peer-reviewed workshops, conferences, and journals. And
2. Studies where software reuse costs and benefits are directly investigated. And
3. Studies that report quantitative or qualitative results based on cases from industry. And
4. When several studies reported the same study, we included only the most recent (normally the Journal version) paper.

3.4. Quality assessment

We used the quality assessment checklist developed by Dybå and Dingsøyr [32] as our quality assessment base, which is also recommended by Kitchenham et al. [33] and is originally from the quality assessment criteria on the critical appraisal skills program (CASP) [34].
We modified some questions in the quality assessment checklist² to fit our research topic.

The criteria of whether it is a research paper and the aims of the research in the quality assessment checklist are used to screen out irrelevant papers. The remaining criteria of the checklist were assessed on a four-point scale: “Yes”, “Partial”, “No”, and “Not applicable”. The partial scale was further elaborated in four levels: close to no - 0.25, close to yes - 0.75, and between yes and no - 0.5. We had four iterations for quality assessment piloting until we were satisfied and agreed on the conceptual understanding in the checklist. The changes we made in each iteration are as follows.

**Iteration 1:** We randomly selected three primary studies to pilot the quality assessment checklist. However, we noted some differences and made three improvements as follows.

- **Improvement 1:** Assess both the criteria and their sub-questions to enhance the traceability of the assessment decision.
- **Improvement 2:** Update context criteria using five context facets — product, processes, people, organization, and practices/tools/techniques, based on Petersen and Wohlin’s work [35]. We omitted the context facet — market, as it is not frequently reported in industrial software engineering research [35] and is less relevant to our study objectives.
- **Improvement 3:** Add one sub-question about the appropriateness of the research method used in the primary studies.

**Iteration 2:** We updated the quality assessment checklist and piloted it on a randomly selected primary study, excluding the ones we selected in Iteration 1. However, the first author excluded the study since it was an experience report, while the second author did not. We discussed this misalignment and agreed to include experience reports since they include industrial cases and are focusing on software reuse costs and benefits. As such, we did not exclude any primary studies in Iteration 2. We also agreed to treat the experience report differently, including their quality assessment criteria — less restriction on methodology-related criteria.

**Iteration 3:** We updated the quality assessment checklist and piloted it on another randomly selected primary study, excluding the ones we selected in Iterations 1 and 2. Both the first and second authors agreed on the quality assessment results. The second author reviewed the checklist again and agreed that we had a shared understanding, and the first author could start independently applying it.

**Iteration 4:** When the first author completed the quality assessment, the third author verified the results by applying the quality assessment checklist to three randomly selected papers. There were minor differences in quality assessment results, and the third author gave feedback on assessing the study motivation, sample size, researchers’ bias on the study, and contribution to the field. According to the feedback, the first author reapplied the quality assessment checklist to all included primary studies.

### 3.5. Data extraction

We designed the data extraction form³ based on the general items (e.g., year and author) and items corresponding to each RQ (e.g., reuse type). Two authors piloted the data extraction on three randomly selected primary studies to see if we could extract all the needed data and check the extracted data consistency. After discussing the piloting results, the first author independently applied the data extraction form to the rest of the primary studies. The second and third authors verified the extraction results by applying the data extraction form to three randomly selected primary studies. We discussed the verification results in a joint meeting, finding no major disagreements. However, we noticed that we extracted the metrics but not their results, e.g., we extracted metric – defect density but missed extracting its results – the investigated case got 24% defect reduction after adopting software reuse practices.

We then included data about the reuse comparison scheme, measurement level, and measurement results to address the missing data item. The first author independently extracted the new data items. The second author reviewed the extraction results and found inconsistent granularity issues in reporting the measurement results. Some of the extracted measurement results are at a high level, such as the productivity increased after reuse, while others are at a finer level, such as the productivity increased 25% after reuse. The first and second authors discussed and agreed that a finer level of data extraction provides more information to the data analysis. The first author re-extracted the data to a finer level, which the second author reviewed.

### 3.6. Data analysis

We synthesized the extracted data to answer the RQs. For RQ1, we analyzed the data according to three categories: general information, context characteristics, and reuse-specific characteristics. For RQ2, we used thematic analysis [36] to code the software reuse costs and benefits. We also identified patterns by mapping the software reuse costs and benefits to different characteristics, such as reuse methods. We grouped and analyzed the metrics according to each software reuse cost and benefit in RQ3. For RQ4, we analyzed the quality assessment results based on different types of study — qualitative study, quantitative study, and experience report. We used narrative synthesis [37] to understand how validity threats are reported in the primary studies and analyzed them based on four categories: construct validity, conclusion validity, internal validity, and external validity. The first author conducted the preliminary data analysis, and the results were reviewed and discussed iteratively with the second author. The third author reviewed the final results, and we addressed the review comments in a joint meeting.

### 3.7. Validity threats

We followed the guidelines developed by Amapatzioglou et al. [38] on reporting the validity threats of secondary studies in software engineering.

- **Study selection validity** concerns the threats that occur in the study search and selection processes. We used automated search to identify secondary studies on software reuse costs and benefits and extracted the primary studies on software reuse costs and benefits from industrial cases as our start set for snowballing. To mitigate the threat of missing secondary studies, we used four databases and searched for all secondary studies on software reuse to make our search more inclusive. We also used two tertiary studies as our additional data source to include more papers. As for the threat of missing primary studies, we used forward snowballing on Google Scholar to cover the missing years by the selected secondary studies. 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, the chances of missing primary studies before 2017 are low. To avoid including the wrong papers, all three authors had a shared understanding of the selection criteria and piloted the selection criteria on randomly selected secondary/primary studies. The first author applied the selection criteria to all primary studies, and the second author reviewed all selection results.

---

² The final quality assessment checklist is available at https://zenodo.org/records/10649264.
³ The detailed data extraction form is available at https://zenodo.org/records/10649264.
Communication and analysis phase. The development of the data extraction form and the accuracy of the extracted data are our threats in the data extraction phase. To mitigate that, we involved two authors in developing the data extraction form according to the RQs. All authors have piloted the data extraction results on three randomly selected primary studies to have a shared understanding. The first and second authors customized the quality assessment checklist based on several piloting results. When the first and second authors had a consensus on the quality assessment checklist, the third author also piloted the quality assessment checklist on three randomly selected primary studies with the first author. In addition, we tried our best to provide an unbiased review. The conclusion reflects our knowledge of software reuse costs and benefits. The first author performed the main analysis, and the results were discussed with the other authors.

• **Research validity** involves threats that can be identified in all phases and concern the overall research design. We faced threats related to the study's replicability, the gap that the study addressed, the publication bias of the existing primary studies on software reuse costs and benefits, and the authors' familiarity with the research field. To ensure the study's replicability, we developed the study protocol according to Kitchenham et al. [35]. Based on our literature search on software reuse secondary studies, we identified a need to investigate the state of the art of software reuse costs and benefits. We found relatively fewer studies on software reuse costs, which may result in an unclear tradeoff between software reuse costs and benefits. All authors have worked with industrial software reuse, and we tried our best to provide an objective review when analyzing the studies.

### 4. Characteristics of the included primary studies (RQ1)

The results are based on 30 primary studies. This section describes the primary studies from three perspectives: (1) general information of the primary studies, (2) contextual characteristics, and (3) reuse-specific characteristics.

#### 4.1. General information of the primary studies

The general information of the primary studies includes publication type, publication year, publication channels, and study type. 14 out of 30 primary studies were published at conferences, while the remaining 16 were published as journal papers. Table 3 shows the publication type and channels of the selected primary studies. Fig. 2 shows the number of primary studies over the years. We found that software reuse costs and benefits have been investigated in industrial cases most between 2001 and 2015. There was a publication gap between 2017 and 2020. However, we did find two primary studies [39, 40] specifically investigated software reuse costs and benefits and were published in 2019 and 2020 during our snowballing search process. We excluded them because the studies did not include industrial cases.

All primary studies have one or more industrial cases where software reuse was practiced. We categorized them into three study types: quantitative studies, experience reports, and qualitative studies. In total, we had 17 quantitative studies (QN), ten experience reports (ER), two qualitative studies (QL), and one both qualitative and quantitative (QNL) study (see Table 2). 4.2. Contextual characteristics

Contextual characteristics include the context information about the investigated industrial case (i.e., the type of development methodology and programming languages and the size of the company and the project). We summarized the contextual characteristics results in Table 4.

The results show that only five out of 30 primary studies reported the development methodology. Mohagheghi et al. (QN9 [13]), Mohagheghi and Conradi (QN13 [42]), Kolb et al. (QN23 [44]), one of the divisions in Lim (ER2 [7]) explicitly mentioned they followed incremental development. Gupta et al. (QN21 [43]) mentioned the waterfall process for one of the studied industrial cases.

Fifteen primary studies (50%) have described the company sizes, of which seven investigated large-sized companies, six investigated medium-sized companies, and three investigated small-sized companies. The results show that most studies investigated large- and medium-sized companies (more than 80%) instead of small-sized companies. More empirical studies are needed to investigate whether software reuse benefits small-sized companies the same as large- and medium-sized companies.

Twenty-four primary studies (80%) have mentioned their project sizes. The results show that software reuse costs and benefits were evaluated relatively uniformly across all ranges of project sizes — seven small-scale projects, nine medium-scale projects, and eight large-scale projects.

Seventeen primary studies (57%) reported the programming languages used in the investigated reuse cases, of which C (eight primary studies) and Java (four primary studies) were studied the most.

#### 4.3. Reuse-specific characteristics

Reuse-specific characteristics refer to the information typically related to software reuse in industrial cases investigating costs and benefits, which includes reuse approach, reuse type, reuse purpose, reuse method, reuse comparison scheme, and measurement level. Table 5 presents the reuse-specific characteristics.

---

**Table 2**

<table>
<thead>
<tr>
<th>Study types</th>
<th>Primary studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>QN</td>
<td>17 (QN1 [1], QN3 [12], QN4 [2], QN5 [23], QN7 [41], QN9 [13], QN10 [4], QN11 [3], QN12 [15], QN13 [42], QN15 [21], QN21 [43], QN23 [44], QN25 [45], QN26 [46], QN28 [47], QN29 [48])</td>
</tr>
<tr>
<td>ER</td>
<td>10 (ER2 [7], ER6 [14], ER8 [10], ER16 [5], ER17 [9], ER18 [49], ER19 [8], ER22 [50], ER24 [6], ER27 [51])</td>
</tr>
<tr>
<td>QL</td>
<td>2 (QL14 [16], QL30 [52])</td>
</tr>
<tr>
<td>QNL</td>
<td>1 (QNL20 [53])</td>
</tr>
</tbody>
</table>

* The mapping between the paper ID and primary studies is available at [https://zenodo.org/records/10649264](https://zenodo.org/records/10649264)
The reuse approach refers to the technical methods used by industrial cases to implement software reuse. We identified eight reuse approaches from the primary studies: software product line (SPL), reuse from repository, compositional reuse, component reuse, frame-based black-box reuse, and modified reuse. The results show that both reuse types are widely used in the primary studies. Nineteen primary studies reported verbatim reuse, and 14 reported modified reuse. In addition, industrial cases in ten primary studies adopted both verbatim and modified reuse.

The reusable assets can be used for different purposes, i.e., infrastructure reuse, domain reuse, and architecture reuse [17]. The results show that 75% reported reusable assets are designed for a specific domain, and fewer are about the infrastructure (21%) and architecture (4%). This result is expected since the most reported reuse approach is SPL, which comprises a product family within a specific application domain.

The reuse method refers to whether the company follows a systematic way to implement software reuse, including systematic reuse, ad

---

X. Chen et al.  Information and Software Technology 171 (2024) 107451

Table 3

<table>
<thead>
<tr>
<th>Publication type</th>
<th>Publication channels</th>
<th>Primary studies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IEEE Software</td>
<td>1995 [ER2], 1995 [ER19]</td>
</tr>
<tr>
<td></td>
<td>Journal of Systems and Software</td>
<td>1997 [QN3], 2001 [QN7]</td>
</tr>
<tr>
<td></td>
<td>Software Process: Improvement and Practice</td>
<td>1998 [ER27]</td>
</tr>
<tr>
<td></td>
<td>Decision Sciences</td>
<td>1999 [QN28]</td>
</tr>
<tr>
<td></td>
<td>ACM Transactions on Software Engineering and Methodology</td>
<td>2008 [QN13]</td>
</tr>
<tr>
<td></td>
<td>Empirical Software Engineering</td>
<td>2009 [QN21]</td>
</tr>
<tr>
<td></td>
<td>Journal of Software Engineering and Applications</td>
<td>2011 [QN30]</td>
</tr>
<tr>
<td></td>
<td>Requirements Engineering</td>
<td>2015 [QN20]</td>
</tr>
<tr>
<td></td>
<td>Total Quality Management and Business Excellence</td>
<td>2015 [QN25]</td>
</tr>
<tr>
<td></td>
<td>Business &amp; Information Systems Engineering</td>
<td>2021 [QN29]</td>
</tr>
</tbody>
</table>

Conference

| Conference              | IEEE International Conference on Software Engineering | 2004 [QN9] |
|                        | IEEE International Conference on Software - Science, Technology and Engineering | 2005 [ER6] |
|                        | IEEE International Conference on Software Maintenance | 2005 [QN10] |
|                        | Springer International Conference on Software Product Line | 2005 [QN11] |
|                        | IEEE International Software Product Line Conference | 2006 [QN23], 2007 [ER18], 2011 [ER16] |
|                        | IEEE International Conference on Management of Innovation and Technology | 2012 [QN12] |
|                        | Springer International Conference on Software Reuse | 2008 [ER17] |
|                        | Springer International Conference on Computational Science and its Applications | 2014 [QN15] |
|                        | ACM International Software Product Line Conference | 2011 [ER24], 2014 [ER22] |
|                        | IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing | 2016 [QN26] |

Table 4

<table>
<thead>
<tr>
<th>Contextual characteristics</th>
<th>Results: are presented as - Extracted characteristic (Number of primary studies reported) [References]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development methodology$^a$</td>
<td>- Incremental development (4) [ER2, QN9, QN13, QN23]</td>
</tr>
<tr>
<td></td>
<td>- Waterfall development (1) [QN21]</td>
</tr>
</tbody>
</table>

Company size$^b$

| Project size$^c$ | - Large-sized companies (7) [QN9, QN13, QL14, QN21, QN23, ER24, QN7] |
|                 | - Medium-sized companies (6) [QN1, ER6, QN7, QN10, ER18, QN28] |
|                 | - Small-sized companies (3) [QN12, ER17, QN25] |
|                 | - Small-scale projects (7) [QN4, ER6, QN7, ER8, QN11, QN12, QN15] |
|                 | - Medium-scale projects (9) [ER2, QN1, QN10, QN21, QN25, QN28, QL14, ER17, ER22] |
|                 | - Large-scale projects (8) [QN3, QN5, QN9, QN13, QN26, ER19, QNL20, ER24] |

| Programming languages | - C (8) [ER2, QN7, QN9, QN12, QN13, QN23, ER24, QN25] |
|                       | - Java (4) [QN4, QN13, QL14, QN21] |
|                       | - RPG (2) [QN1, QN11] |
|                       | - C++ (2) [QN7, ER19] |
|                       | - Erlang (2) [QN9, QN13] |
|                       | - Ada (1) [QN3] |
|                       | - Pascal (1) [ER2] |
|                       | - Fortran (1) [QN5] |
|                       | - Perl (1) [QN9] |
|                       | - COBOL (1) [QN10] |

$^a$ The development methodology stands for what software development process is followed in the case companies, such as agile development, continuous integration, and incremental development.

$^b$ We divided the reported companies into three scales in accordance with the European Commission’s classification of micro, small, and medium-sized companies [54]: large-sized companies have more than 250 employees; medium-sized companies have more than 50 but less than 250 employees; small-sized companies have less than 50 employees.

$^c$ We followed the same project size category as Mohagheghi and Conradi [17]. Small-scale objects: a few reused software assets or small products; medium-scale objects: larger than the small-scaled ones, but the objects are still few, or the software size is less or around 100 KLOC; large-scale objects: the objects are quite a few, or the software size is more than 100 KLOC.
the same projects using different reuse methods. Two primary studies (ER2 [7], QN9 [13], QN13 [42]) compared the costs and benefits followed a reuse approach and those that did not. Three primary studies focused on systematic reuse, while only four and three studies reported primary study investigated both controlled reuse and systematic reuse on different projects (QN1 [1], ER17 [9], ER19 [8]), and one

Ad hoc reuse, on the other hand, follows no formalized process or guidelines when implementing software reuse. Systematic reuse aims to develop new systems using existing software assets in an organized and planned way.

Domain-specific reuse stands for reusing common assets with a specific domain. Infrastructure reuse refers to reusing common assets across different domains, such as the login function. Engine reuse, or transformation-based reuse, refers to reuse engines that can produce outputs by transforming appropriate inputs. The reused template has a basic structure of the assets, which includes a pre-determined part that is independent from the system and a blank part that engineers use to develop

The framework contains pre-written design patterns and components of common functionalities that provide a foundation for building the system. Compositional reuse stands for reusing both assets that are specific to a single product and generic to various projects. Software product line (SPL) uses domain engineering to develop domain-specific reusable assets, and all products derived from the SPL reuse the same architecture. Selecting

The reused template has a basic structure of the assets, which includes a pre-determined part that is independent from the system and a blank part that engineers use to develop explicitly for the system.

Engine reuse, or transformation-based reuse, refers to reuse engines that can produce outputs by transforming appropriate inputs.

Infrastructure reuse refers to reusing common assets across different domains, such as the login function.

Domain-specific reuse stands for reusing common assets with a specific domain.

Architecture reuse refers to reusing the structures to build a system. 19 primary studies reported the purpose of developing or using reusable assets.

Systematic reuse aims to develop new systems using existing software assets in an organized and planned way.

Ad hoc reuse, on the other hand, follows no formalized process or guidelines when implementing software reuse.

Controlled reuse involves a more structured approach than ad hoc reuse but is less formal than systematic reuse.

One primary study investigated both controlled reuse and systematic reuse on different companies (ER8 [10]). Twenty primary studies (74%) focused on systematic reuse, while only four and three studies reported ad hoc reuse and controlled reuse.

In our sample, we found authors have used six different reuse comparison schemes to evaluate to what extent software reuse results in benefits or costs in the industrial cases. 14 primary studies compared software reuse costs and benefits between projects, of which eight compared the original project (which developed from scratch) and the projects reused the original ones, and six compared the projects that followed a reuse approach and those that did not. Three primary studies (ER2 [7], QN9 [13], QN13 [42]) compared the costs and benefits of reused and new assets in the same project. Three primary studies (ER6 [14], ER8 [10], ER17 [9]) compared the costs and benefits of the same projects using different reuse methods. Two primary studies (QN3 [12], QN5 [23]) compared the costs and benefits of the assets with different reuse modification levels. Five primary studies (QN7 [41], QN12 [15], QN15 [21], QN25 [45], QN28 [47]) used the reuse rate as the basis of comparison to investigate the costs and benefits.

The costs and benefits of software reuse can be measured at the high (project) and low (component) levels. The results show that the industrial cases have measured software reuse costs and benefits more on the high level than the low level - 22 compared to 7.

RQ1 Key findings: We included 30 primary studies and extracted four context characteristics and six reuse-specific characteristics from our study sample. The key findings about the extracted characteristics are as follows.

• Most primary studies on software reuse costs and benefits were conducted within medium- and large-sized companies (13 primary studies), while only a few investigated small-sized companies (three primary studies).

• The primary studies have investigated software reuse costs and benefits almost equally in all types of project sizes: small-scale projects (seven primary studies), medium-scale projects (nine primary studies), and large-scale projects (eight primary studies).
• Programming Languages C (eight primary studies) and Java (four primary studies) were studied the most in our sample.
• Only five primary studies in our sample reported the development methodology. Therefore, it is unclear in which context the other included studies practiced software reuse.
• SPL (14 primary studies) and reuse from a repository (six primary studies) are the most investigated reuse approaches in our sample.
• Most primary studies used verbatim reuse (nine primary studies) or both verbatim and modified reuse (ten primary studies). Only a few used modified reuse (four primary studies).
• Systematic reuse (20 primary studies) was investigated the most in our sample, and only a few primary studies investigated ad hoc (four primary studies) and controlled reuse (three primary studies).
• Our sample investigated more domain reuse (18 primary studies) than infrastructure (five primary studies) and architecture reuse (one primary study). This finding aligns with the finding in reuse approaches that the most investigated reuse approach is SPL.
• The most investigated comparison schemes in our sample are the comparison of original projects versus projects that reused them (eight primary studies), assets following a reuse approach versus those that did not (six primary studies), and assets with varying reuse rates (five primary studies).
• Most primary studies measured reuse impacts on high levels — e.g., project level (22 primary studies) rather than low levels — e.g., module level (seven primary studies).

5. Software reuse costs and benefits and their relation with reuse practices (RQ2)

We identified nine software reuse benefits and six software reuse costs from primary studies (see Fig. 3). Software reuse benefits (29 primary studies) were investigated more than costs (seven primary studies). The most investigated benefits in our sample are better quality (20 out of 30 primary studies) and improved productivity (20 out of 30 primary studies). More than three studies reported software reuse benefits in cost-saving, better maintainability, and faster time-to-market. The rest of the benefits are mentioned once or twice. As for the reported software reuse costs, only reduced quality and reduced maintainability were reported three or two times. The rest of the costs are reported only once. The contradicting findings, e.g., better quality and reduced quality, are further explained in metric level in RQ3 (see Section 6.2).

We divided the benefits into two types of improvements: product-related benefits and process and team related benefits. Product-related benefits include better quality, better maintainability, increased performance, and prolonged lifecycle of reused product (game) and standardized architecture. Process and team related benefits include improved productivity, cost-saving, faster time-to-market, and improved learning.

We also found that quantitative studies and experience reports reported the most benefits — better quality, improved productivity, cost-saving, better maintainability, and faster time-to-market. All reported software reuse costs were reported in quantitative studies, except for the increased learning curve. The results show that the stand-out software reuse benefits are better quality and improved productivity.

We mapped some characteristics — reuse approach, reuse type, project size, reuse purpose, reuse method, and measurement level, with the identified software reuse impacts (costs and benefits) to see which characteristics are more likely to benefit companies. The relation between the identified software reuse impacts and reuse approach, reuse type, and reuse method stands out among the other characteristics. Fig. 4 depicts the relation between the observed software reuse benefits and costs, and investigated software reuse approaches. The primary studies investigated SPL and reuse from repository the most (see Table 5). The mapping results between reuse approaches and software reuse benefits indicate that SPL covers more (seven out of nine) benefits than the other reuse approaches. Better quality has been observed in all identified reuse approaches. Improved productivity has been observed in all reuse approaches except engine reuse. Considering the number of primary studies and the coverage of benefits investigated in reuse approaches, SPL and reuse from repository are more likely to benefit companies. Within the limited primary studies on software reuse costs, the results showed that SPL is associated with some costs.

Fig. 5 shows the mapping between the observed software reuse benefits and costs, and reuse types and methods. For reuse types, the results show that in our sample, most benefits were observed when both verbatim and modified reuse were applied. In the cases of using a single reuse type, more primary studies investigated verbatim reuse (nine primary studies) than modified reuse (four primary studies). By looking at those primary studies that only adopt modified or verbatim reuse, the results show that verbatim reuse may result in more benefits to companies, considering the number of primary studies and the coverage of benefits investigated in verbatim/modified reuse. These findings were also observed in primary studies that explicitly investigated the costs and benefits among different reuse types: Thomas et al. (QN3 [12]), and

---

4 Game B reuses components from Game A. Such reuse helped prolong the lifecycle of Game A.

5 The relations between the identified software reuse impacts (costs and benefits) and project size, reuse purpose, and measurement level are available at https://zenodo.org/records/10649264.
Selby (QN5 [23]) found that verbatim reuse has a more positive impact than modified reuse on quality, maintainability, and productivity.

As for the reuse methods, most primary studies have adopted systematic reuse. All reported reuse methods have benefits in better quality and improved productivity. The results show that systematic reuse has more reuse benefits compared to controlled and ad hoc reuse. Similar findings were identified in three primary studies that explicitly investigated the costs and benefits among different reuse methods: Morad and Kuflik (ER6 [14]) and Tomer et al. (ER8 [10]) found systematic reuse had better productivity in all investigated code assets except one. Beyer et al. (ER17 [9]) found systematic reuse resulted in better quality, productivity, and performance when compared to ad hoc reuse.

RQ2 Key findings: Software reuse benefits (29 primary studies) have been investigated more than the costs (seven primary studies) in our sample. The following are the key findings about the identified software reuse costs and benefits.

- Better quality (20 primary studies) and improved productivity (20 primary studies) were the most frequently investigated software reuse benefits in our sample.
- Reduced quality (three primary studies) was the most frequently investigated software reuse costs in our sample. The reasons for software reuse having both positive and negative quality impacts are further explained in RQ3 (see Section 6.2).
- Our results showed that SPL (seven benefits and 14 primary studies) is more likely to benefit companies.
- Our results showed that systematic reuse (seven benefits and 20 primary studies) is more likely to benefit companies.
- Our results showed that verbatim reuse (five benefits and nine primary studies) is more likely to benefit companies.
- Better quality and improved productivity are the software reuse benefits generally observed in all identified reuse approaches.

6. Metrics for measuring software reuse costs and benefits (RQ3)

This section presents the metrics that the companies used to measure software reuse costs and benefits. We cluster the metrics according to the identified software reuse costs and benefits from RQ2. We present the metrics, their definitions, and results in an aggregated form.\footnote{The detailed metrics definitions and results of each primary study are available at https://zenodo.org/records/10649264.}

6.1. Metrics for software reuse benefits

This sub-section presents the metrics used to measure the identified software reuse benefits.

6.1.1. Metrics for better quality

Twenty primary studies identified better quality as a benefit of software reuse. Moreover, 19 of these primary studies used 12 unique metrics to measure quality benefits while practicing software reuse (see details in Table 6). The most used metrics relate to code problems, such as defects and bugs. Some metrics relate to code structure, developers’ subjective rating on debugging and maintaining software, and customer complaints. Defect density is the most frequently used metric. We noticed that primary studies used different terms when describing defect density, such as fault density and error density. We refer to them as defect density in the rest of the paragraph. Some primary studies measured defect density and found software reuse helped reduce the defects (ER2 [7], QN9 [13], QN13 [42], QN26 [46]) and some found higher reuse rate and verbatim reuse resulted in fewer defects (QN3 [12], QN5 [23], QN7 [41], QN12 [15], QN25 [45]). In addition, Gupta et al. (QN21 [43]) found that software reuse helps decrease the defect density of the reused software. Three primary studies counted the number of defects/bugs to investigate better quality in practicing software reuse. Deniz and Bilgen (QN15 [21]) counted the number of defects/bugs to investigate better quality in practicing software reuse. Deniz and Bilgen (QN15 [21]), Quilty and Cinneide (ER16 [5]), and Otsuka et al. (ER24 [6]) used the defect counts to evaluate the quality between reused and non-reused projects or products. However, they did not normalize the number of defects with the project size, which makes...
the result less comparable. Multiple primary studies used complexity and coupling metrics. The results showed that software reuse helps in reducing the code complexity (QN10 [4], QN15 [21], ER17 [9]) and improving the code coupling (QN11 [3], QN15 [21]). The rest of the quality metrics are only used by a single primary study (see details in Table 6).

6.1.2. Metrics for improved development productivity

Twenty primary studies identified improved productivity as a benefit of software reuse. Moreover, 18 of these primary studies used eight unique metrics to measure productivity gains while practicing software reuse (see details in Table 7). Most productivity metrics relate to development effort, while some of them relate to effort in requirements (QN15 [21]), function points (QN4 [2]), design (QN5 [23]), or testing (ER17 [9]). We found four kinds of productivity metrics related to development effort: the size of the assets per development effort (size per effort), the development effort per the size of the assets (effort per size), the total development effort, and the total size of the developed assets. The most used productivity metrics are related to size per effort. Depending on whether to include the size of reused assets in the total size of the assets, productivity (size per effort) can be further divided into apparent/gross productivity (total size of the written code and reused code per development effort) and net/actual productivity (size of the written code per development effort). Ten primary studies measured apparent/gross productivity, and three measured net/actual productivity. By using apparent/gross productivity, the primary studies found (1) software reuse increases productivity (ER2 [7], QN10 [4], QN15 [21], ER19 [8], ER22 [50], QN26 [46]), and (2) the higher reuse rate, the more increased in productivity (QN7 [41], QN12 [15], QN25 [45], QN28 [47]). Morisio et al. (QN4 [2]) and Deniz and Bilgen (QN15 [21]) used actual productivity and found that software reuse helps improve productivity. Baldassarre et al. (QN10 [4]) investigated actual productivity in different periods of reuse approaches and found systematic reuse helps increase productivity in the early phase when developing for reuse. When both the non-reuse and reuse processes mature, their actual productivity tends to range within a defined interval. Three primary studies used metrics of effort per size to measure productivity. The results showed that software reuse helps increase productivity (ER18 [49]) and the more reuse, the better productivity (QN5 [23], QN7 [41]). In addition to the normalized metrics, Selby (QN5 [23]) also used non-normalized metrics (development effort and code size) to measure productivity and found that verbatim reuse has the least development effort and smallest code size. Morad and Kuflik (ER6 [14]) and Tomer et al. (ER8 [10]) measured the development effort of using different reuse methods for the same assets and found systematic reuse required the least development effort. Three primary studies (QN11 [3], ER16 [5], ER17 [9]) investigated productivity only using a non-normalized metric — development effort and found software reuse helps reduce development effort.

6.1.3. Metrics for better maintainability

Six primary studies identified better maintainability as a benefit of software reuse. They used six unique metrics to measure better maintainability when practicing software reuse (see details in Table 8). Maintainability is not limited to code changes but their required effort. The six metrics can be categorized into maintenance effort (QN3 [12], QN5 [23], ER16 [5]), modification rate (QN5 [23], QN9 [13], QN13 [42]) and size of the reduced code (QN11 [3]). Only metrics — fault correction effort and code modification rate, were used by more than one primary study. The results of fault correction effort showed projects following software reuse have less effort spent on bug fixing (ER16 [5]) and verbatim reuse has the lowest effort in fault correction (QN5 [23]). Mohagheghi et al. (QN9 [13]), Mohagheghi and Conradi (QN13 [42]), and Selby et al. (QN5 [23]) found reused components are less changed. Selby et al. (QN5 [23]) found that verbatim reuse has the lowest fault isolation effort, change correction effort, change density, and average number of changes per module. Zhang and Jarzabek (QN11 [3]) found that reuse helps reduce the number of codes compared to the original code size. Thomas et al. (QN3 [12]) found that verbatim reused and slightly modified reused modules had less rework effort than newly created ones.
are calculated is not clear. However, how the costs were not clearly defined. In another case, Quilty and Cinneide calculated by multiplying the overall development effort by the salary rate of the developers. In contrast, the management and support related costs were not clearly defined. In another case, Quilty and Cinneide (ER16 [5]), and Otsuka et al. (ER24 [6]) found that the costs of developing reusable assets will be paid off after several times of implementing reusable assets can help prolong the lifespan of previous games. Mihale-Wilson (QN29 [48]) used grid analysis and found that software reuse helps faster time-to-market (QN11 [3]) and reduce memory usage (QN11 [3], ER17 [9]). Morisio et al. (QN4 [2]) investigated the developers' learning in development with and without reuse conditions, and they found that software reuse increases the developers' learning. Mihale-Wilson (QN29 [48]) used grid analysis and found that reuse from previous games can help prolong the lifespan of previous games.

### 6.1.4. Metrics for cost-saving

Eight primary studies identified cost-saving as a benefit of software reuse. Moreover, seven of these primary studies used six unique metrics to measure cost-saving when practicing software reuse (see details in Table 9). Lim (ER2 [7]), Ha et al. (QN12 [15]), Sun et al. (QN25 [45]), and Powell and Brown (ER27 [51]) monetized the cost-saving, and they found reuse helps save cost. Morad and Kuflik (ER6 [14]), Quilty and Cinneide (ER16 [5]), and Otsuka et al. (ER24 [6]) found that the costs of implementing reusable assets will be paid off after several times of implementations.

However, compared to other extracted quality or productivity benefits related metrics, these monetized cost-saving metrics lack a clear definition. For example, Ha et al. (QN12 [15]) monetized the total software reuse costs using software reuse development costs and management and support costs. The software reuse development costs were calculated by multiplying the overall development effort by the salary rate of the developers. In contrast, the management and support related costs were not clearly defined. In another case, Quilty and Cinneide (ER16 [5]) looked for the breakeven point when the reuse benefits have paid off the costs of developing reusable assets. However, how the costs are calculated is not clear.

### 6.1.5. Metrics for faster time-to-market

Four primary studies identified faster time-to-market as a benefit of software reuse. Moreover, these four primary studies used a unique metric to measure faster time-to-market when practicing software reuse (see details in Table 10). All four primary studies tracked how long it took to develop an asset from beginning to end. Furthermore, the results show that software reuse helps faster time-to-delivery (ER2 [7], ER19 [8], QNL20 [53], ER24 [6]).

### 6.1.6. Metrics for other benefits

Table 11 presents four unique metrics used to verify the software reuse benefits in increased performance, improved learning, and prolonged lifecycle of reused product (game). Zhang and Jarzabek (QN11 [3]) and Beyer at al. (ER17 [9]) investigated the performance using metrics — running time and memory usage. They found software reuse can help faster the game processing time (QN11 [3] and reduce memory usage (QN11 [3], ER17 [9]). Morisio et al. (QN4 [2]) investigated the developers’ learning in development with and without reuse conditions, and they found that software reuse increases the developers’ learning. Mihale-Wilson (QN29 [48]) used grid analysis and found that reuse from previous games can help prolong the lifespan of previous games.

### Table 6
Better quality metrics, definitions and results.

<table>
<thead>
<tr>
<th>Metrics (number of primary studies used)</th>
<th>Definition from primary studies</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defect/Fault/Error density: the number of defects/errors divided by the software size (10)</td>
<td>- ER2 (1994), QN7 (2001), QN9 (2004), QN12 (2012), QN21 (2009), QN25 (2015): the number of defects/errors/ trouble reports per (thousand) non-commented source line of code. - QN3 (19997), QN5 (2005), QN13 (2008), QN26 (2016): Errors/faults/trouble reports/defects (thousand) source line of code.</td>
<td>- ER2, QN3, QN9, QN13, QN26 reported that the defect densities of software that were developed with reuse approaches have decreased. - QN21 reported that the reusable software had a defect reduction after it had been reused. - QN5 reported that verbatim reuse had the fewest defect density. - QN7 reported that the more external reuse, the fewer defect densities. QN12, QN25 reported the higher reuse rate, the fewer defect densities.</td>
</tr>
<tr>
<td>Number of defects/bugs (4)</td>
<td>- QN15 (2014), ER16 (2011), QN24 (2011): the number of defects/bugs. - ER16 (2011): the number of bugs at different bug severity levels.</td>
<td>- QN15 reported a decrease in defect counts in reusable components after they have been reused in several other products. - QN16 and QN24 reported a decrease in defect counts in software that was developed with reuse approaches. - ER16 reported software that was developed with reuse approaches had less number of bugs at all bug severity level.</td>
</tr>
<tr>
<td>Coupling (2)</td>
<td>- QN11 (2005), QN15 (2014): Coupling Between Objects (CBO)</td>
<td>- QN11 and QN15 report that software developed with reuse approaches had lower code coupling.</td>
</tr>
<tr>
<td>Customer complaint density (1)</td>
<td>- QN1 (2001): The number of customer complaints per line of code.</td>
<td>- QN1 reported that software reuse helped decrease the customer complaint density.</td>
</tr>
<tr>
<td>Rate of error slippage</td>
<td>- QN3 (1997): Percentage of errors that escape unit test.</td>
<td>- QN3 reported that slightly modified reused components have significantly lower error slippage rates.</td>
</tr>
<tr>
<td>Index of quality of programming (1)</td>
<td>- QN4 (2002): Development effort (excluding rework effort) per total development effort (development effort + rework effort).</td>
<td>- QN4 reported that development with reuse was less error-prone than traditional development.</td>
</tr>
<tr>
<td>Average faults per module (1)</td>
<td>- QN5 (2005): Average (and standard deviations) for total faults in different types of reuse modules.</td>
<td>- QN5 reported that verbatim reused modules had fewer faults compared to modules that were newly developed, majorly revised, or slightly revised.</td>
</tr>
<tr>
<td>Number of Delta (1)</td>
<td>- QN7 (2001): Delta represents a change to the software, such as an enhancement or a repair, which are sometimes used to estimate error rates.</td>
<td>- QN7 reported that the higher the External Reuse Frequency or External Reuse Level, the lower the Delta.</td>
</tr>
<tr>
<td>Quality rating (1)</td>
<td>- QN7 (2001): Quality rating is a subjective ordinal measure from developers' view regarding debugging and maintaining software. The scale ranges from 1-worst to 10-best.</td>
<td>- QN7 reported that the higher the External Reuse Frequency or External Reuse Level, the higher the rating in quality.</td>
</tr>
<tr>
<td>Number of field defects reported by customers (1)</td>
<td>- ER17 (2008): The aggregated number of field defects reported by customers.</td>
<td>- ER17 reported that customers reported fewer defects when following systematic reuse compared to ad hoc reuse.</td>
</tr>
<tr>
<td>Fault rate (1)</td>
<td>- ER19 (1995): Number of major errors to total recorded faults at the time of delivery.</td>
<td>- ER19 reported software reuse helps improve the fault rate of systems at the time of delivery.</td>
</tr>
</tbody>
</table>
6.2. Metrics for Software Reuse Costs

We identified seven primary studies that reported five software reuse costs. Table 12 presents all the costs and their corresponding metrics. Reduced quality was measured according to the defect density per defect type/defect severity/impact attributes (QN21 [43], complexity (QN11 [3]), error source, and error type (QN3 [12])). Both Thomas et al. (QN3 [12]) and Gupta et al. (QN21 [43]) investigated the characteristics of defect types in different reusable assets. In addition, Zhang and Jarzabek (QN11 [3]) found that complexity increased slightly due to reuse. Maintainability drops due to the higher percentage of errors that require more than one day to fix (QN3 [12]), code complexity, and structuredness (QN23 [44]). Deniz and Bilgen (QN15 [21]) used the number of cycles, time delay, and CPU usage to measure performance and found that software reuse can result in performance decay. Frakes and Succi (QN7 [41]) measured the productivity in two cases, using two metrics — size per effort and effort per size. The results showed a negative relationship between software reuse and productivity. Kolb et al. (QN5 [23]) reported that the error rate per size of code was higher in the modules with software reuse than in those without. Another case showed a negative relationship between productivity and the number of requirements per person-hour. QN7 reported that one case showed a positive relationship between productivity and External Reuse Level, and another case showed a marginally positive relationship between productivity and External Reuse Frequency. ER2 and QN10, QN15, ER19, QN26, QN28, and ER22 reported that productivity increased after applying software reuse. QN12 and QN25 reported a positive relationship between productivity and reuse rate.

Table 12

<table>
<thead>
<tr>
<th>Metrics (number of primary studies used)</th>
<th>Definition from primary studies</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apparent/Gross productivity: total size of project per total development effort (10)</td>
<td>- ER2 (1994), QN7 (2001), QN12 (2012), QN15 (2014), QN25 (2014): (thousand) non-comment line of code per engineering month/person-day/person-hour. - QN10 (2005), QN26 (2016): Line of code per person-hour/person-day. - QN15 (2014): number of requirements per person-hour. - ER19 (1995): average line of code per person-day. - QN26 (2016): total number of Web pages per person-day. - QN28 (1999): The productivity is calculated in a combination of the productivity of new code, the productivity of reused code, the quality of the reuse decision, and the relative value of reuse to the company. - ER22 (2014): number of development models per year.</td>
<td>- QN7 reported that one case showed a positive relationship between productivity and External Reuse Level, and another case showed a marginally positive relationship between productivity and External Reuse Frequency. - ER2, QN10, QN15, ER19, QN26, QN28, and ER22 reported that productivity increased after applying software reuse. - QN12 and QN25 reported a positive relationship between productivity and reuse rate.</td>
</tr>
<tr>
<td>Development effort (6)</td>
<td>- QN5 (2005), ER6 (2005), ER8 (2004), QN11 (2005), ER17 (2008), ER16 (2011): Total development time is (tenth of) person hours/person-day/person/month.</td>
<td>- QN7 reported that newly developed and majorly revised modules had the most development effort, slightly revised modules had the second most, and those completely reused had the least. - ER6 and ER8 reported that systematic reuse had the least development effort compared to development from scratch and other reuse methods. - ER6, QN11, ER17, and ER16 reported that software reuse helped reduce the development effort.</td>
</tr>
<tr>
<td>Development effort per size of project (3)</td>
<td>- QN5 (2005): tenths of hours per line of code. - QN7 (2001): person-day per number of modules.</td>
<td>- QN5 reported that newly developed modules had the most effort per source line, majorly revised and slightly revised modules had the second most, and completely reused modules had the least. - QN7 reported that one case showed a negative relationship between effort/module and External Reuse Frequency and a marginally negative relationship between effort/module and External Reuse Level. Another case showed a negative relationship between External Reuse Frequency and effort/module. - QN18 reported that software reuse helped reduce the time and effort spent on each new product.</td>
</tr>
<tr>
<td>Net/Actual productivity (3)</td>
<td>- QN4 (2002): the net size of the developed application (object-oriented function points)/hours - QN10 (2005): written line of code per person-hour - QN15 (2014): number of new requirements per man-hour.</td>
<td>- QN4 reported that applications developed with reuse provide higher net productivity than the ones developed without reuse. - QN10 reported that reuse-oriented development helped increase the actual productivity in the early phase when developing for reuse. - QN15 reported that productivity improved when developers were familiar with the development of the product line.</td>
</tr>
<tr>
<td>Size of the code (1)</td>
<td>- QN5 (2005): line of code.</td>
<td>- QN5 reported that verbatim reuse has the smallest size of code compared to reusing majorly revised modules, newly developed modules, and slightly revised modules.</td>
</tr>
<tr>
<td>Actual/Estimated development effort (1)</td>
<td>- ER22 (2014): Actual development effort: total man-hours (month or year) spent in developing the core assets and the number of models with software reuse. Estimated development effort: estimated total man-hours (month or year) spent in developing the core assets and the number of models without software reuse.</td>
<td>- ER22 reported that the actual development effort with software reuse was less than the estimated development effort without software reuse.</td>
</tr>
<tr>
<td>Percentage of development effort spent on design (1)</td>
<td>- QN5 (2005): The effort spent during module design activities divided by total module development effort.</td>
<td>- QN5 reported that newly developed modules had a higher percentage of effort spent in design than verbatim reused, slightly revised, and majorly revised modules.</td>
</tr>
<tr>
<td>Testing effort (1)</td>
<td>- ER17 (2008): total testing time (person-month).</td>
<td>- ER17 reported that software reuse helped save testing time.</td>
</tr>
</tbody>
</table>
of the operations/parameters/members/methods. The same goes for
maintainability, that the overall effort used to fix or isolate the errors
is reduced in reusable assets (QN3 [12], QN5 [23], ER16 [5]), but the
proportion of errors requiring a longer fix is more in reusable assets
(QN3 [12]). Ten primary studies (ER2 [7], QN7 [41], QN12 [15],
QN15 [21], QN25 [45], ER19 [8], QN10 [4], QN26 [46], QN28 [47])
reported that software reuse increased apparent/gross productivity.
This result is expected since software reuse reduces the effort of writing
the code from scratch. Three primary studies (QN4 [2], QN10 [4],
QN15 [21]) also reported that following a reuse approach allows devel-
opers to develop new assets faster than those without reuse. However,
Frakes anducci (QN7 [41]) observed a contradictory finding when
comparing productivity with External Reuse Frequency (total number
of references to reused external components/total number of references
to reused both internal and external components) and External Reuse
Level (number of external reusable assets that are used more than
the maximum number of uses/total number of external and internal
assets). Further investigation is needed for productivity, External Reuse
Frequency, and External Reuse Level. Zhang and Jarzabek (QN11 [3])
and Beyer et al. (ER17 [9]) reported that software reuse benefits
performance since it takes less space (ROM), while deniz and Bilgen
(QN15 [21]) found that software reuse causes lower performance since
it takes longer to process (CPU usage, number of cycles, time delays).
Slyngstad et al. (QL14 [16]) claimed that software reuse would benefit
standardized architecture. However, when Kolb et al. (QN23 [44])
measured the conformance in architecture, the results showed that
the conformance decreased. Further investigation into software reuse’s
impact on architecture is needed.

RQ3 Key findings: The key findings about the measurements for
software reuse costs and benefits from the included 30 primary studies
are as follows.

- Better quality and improved productivity are the most evaluated
software reuse benefits in our sample.
7. Strength of the evidence on software reuse costs and benefits (RQ4)

This section presents the strength of reported evidence regarding study types, software reuse costs and benefits, and quality assessment results.

7.1. Study type and quality assessment

We assessed the quality of the primary studies using the modified quality assessment checklist (as described in Section 3.4). We used nine out of 11 criteria from our quality assessment checklist to evaluate the primary studies’ quality. We removed the first two criteria since they were used to screen out irrelevant papers. In addition, we categorized the primary studies into three study types: experience reports (ER), quantitative studies (QN), and qualitative studies (QL). To account for the variations in the study types, we customized the quality assessment criteria for each study type.

- ER: We removed the research design and reflexivity criteria for experience reports since experience reports are often written by practitioners who share their experiences or observations in a particular context and may not always follow the same formal research design and methodology required for a research study. In addition, the reflexivity criterion (relationship between researchers and participants) may not be necessary for experience reports since practitioners wrote them. Therefore, the maximum score for the experience report is seven.
- QL: We removed the criteria of the control group for qualitative studies since qualitative studies only gather practitioners’ perceptions about a specific project in a particular context, which does not require the control group. Therefore, the maximum score for the qualitative study is eight.
- QN: All criteria are applicable for quantitative studies, and the maximum score for quantitative study is nine.

We categorize the selected primary studies into three levels of quality.

- High-quality studies: the total score of the study is greater than 80% of the maximum scores (ER > 5.6, QL > 6.4, QN > 7.2).
- Moderate-quality studies: the total score of the study is less or equal to 80% of the maximum scores but greater than 50% of the maximum scores (3.5 < ER ≤ 5.6, 4.0 < QL ≤ 6.4, 4.5 < QN ≤ 7.2).
- Low-quality studies: the total score of the study is less or equal to 50% of the maximum scores (ER ≤ 3.5, QL ≤ 4.0, QN ≤ 4.5).

The primary studies have four high-quality primary studies, 20 moderate-quality primary studies, and six low-quality primary studies° (see Table 13). All experience reports fall under moderate quality due to a lack of rigorous data collection and analysis methods reporting.

7.2. Relating quality assessment results with software reuse results

We also mapped the quality assessment results with the study findings — software reuse costs and benefits. Fig. 6 depicts the primary studies’ quality of the identified software reuse costs and benefits. All four high-quality primary studies reported better quality as one of the software reuse benefits. Furthermore, the overall quality of the primary studies that reported better quality benefits is higher than those that reported improved productivity benefits. Most of the primary studies that reported better quality (14 out of 20) and improved productivity (15 out of 20) benefits are of moderate quality. Primary studies that reported cost-saving and faster time-to-market benefits are of moderate quality, except ER27. Primary studies reported better maintainability

° The detailed score of each primary study is available at https://zenodo.org/records/10649264.

Table 11
Other benefits, metrics, definitions and results.

<table>
<thead>
<tr>
<th>Other benefits</th>
<th>Metrics (number of primary studies used)</th>
<th>Definition from primary studies</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increased performance</td>
<td>Running time and memory usage (1)</td>
<td>- QN11 (2005): The authors measured the running time with the memory monitor and profiler turned on in the Wireless Toolkit.</td>
<td>- QN11 reported that the memory decreased, and the running time got faster after software reuse.</td>
</tr>
<tr>
<td></td>
<td>Memory usage (1)</td>
<td>- ER17 (2006): The usage of ROM (i.e., the size of the binary program code).</td>
<td>- ER17 reported that development with systematic reuse had a lower usage percentage than development with ad hoc reuse.</td>
</tr>
<tr>
<td>Improved learning</td>
<td>Proxy of programmer’s learning (1)</td>
<td>- QN4 (2002): The cumulated size of developed software (Object Oriented Function Points) up to the total number of developed applications.</td>
<td>- QN4 reported that the learning rate in development with reuse is more than the learning rate in development without reuse.</td>
</tr>
<tr>
<td>Prolonged lifecycle of reused product (game)</td>
<td>Grid Analysis (daily activity levels in the Game per 1 square km grid) (1)</td>
<td>QN29 (2021): Average Actions per grid and day. Pre-new game release, Average Actions per grid and day, Post-new game release, Average number of old game GIP per grid, Average number of new game GIP per grid.</td>
<td>QN29 reported that developing a new game with reusable components from the old game results in an increase in the number of active players in the old game.</td>
</tr>
</tbody>
</table>
The results show that high-quality studies are quantitative and only investigate systematic reuse. The high-quality studies are diverse in reuse approaches (see Table 13) and investigated verbatim or verbatim and modified reuse types with software reuse benefits in better quality, better maintainability, and improved productivity.

The moderate-quality studies investigated all identified reuse approaches and various reuse types and reuse methods. These studies have identified all software reuse benefits except increased performance. Two moderate-quality studies mentioned software reuse costs in reduced quality, reduced maintainability, and reduced productivity when performing verbatim reuse and ad hoc or systematic reuse. Most of the software reuse costs were identified through low-quality studies. The low-quality studies often fail to report one or more than one reuse-specific characteristic.

7.3. Relating quality assessment results with reuse-specific characteristics and software reuse results

We also mapped the quality assessment results with reuse-specific characteristics and the identified software reuse results.
<table>
<thead>
<tr>
<th>Study quality</th>
<th>Reuse approaches</th>
<th>Reuse type</th>
<th>Reuse method</th>
<th>Primary studies</th>
<th>Software reuse benefits</th>
<th>Software reuse costs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>High-quality</strong></td>
<td>SPL</td>
<td>Verbatim</td>
<td>Systematic reuse</td>
<td>QN13</td>
<td>Better quality, better maintainability</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reuse from repository</td>
<td>Verbatim and modified</td>
<td>Systematic reuse</td>
<td>QN10</td>
<td>Better quality, improved productivity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Compositional reuse</td>
<td>Not reported</td>
<td>Systematic reuse</td>
<td>QN1</td>
<td>Better quality</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Framework reuse</td>
<td>Verbatim</td>
<td>Systematic reuse</td>
<td>QN21</td>
<td>Better quality, Reduced quality</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Template reuse</td>
<td>Verbatim and modified</td>
<td>Systematic reuse</td>
<td>QN10</td>
<td>Better quality, improved productivity</td>
<td></td>
</tr>
<tr>
<td><strong>Moderate-quality</strong></td>
<td>SPL</td>
<td>Verbatim</td>
<td>Systematic reuse</td>
<td>QN3, QN9</td>
<td>Better quality, better maintainability</td>
<td>Reduced quality, reduced maintainability</td>
</tr>
<tr>
<td></td>
<td>Reuse from repository</td>
<td>Verbatim and modified</td>
<td>Systematic reuse</td>
<td>ER8, ER19</td>
<td>Better quality, improved productivity, cost-saving</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified</td>
<td>Not reported</td>
<td>QN29</td>
<td>Prolonged lifecycle of reused product (game)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not reported</td>
<td>Systematic reuse</td>
<td>ER18, ER24</td>
<td>Better quality, improved productivity, faster time-to-market</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ad hoc reuse</td>
<td>ER17</td>
<td>Better quality, increased performance, improved productivity</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Not reported</td>
<td>ER16</td>
<td>Better quality, better maintainability, improved productivity, cost-saving</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Component reuse</td>
<td>Verbatim and modified</td>
<td>Not reported</td>
<td>QL14</td>
<td>Better quality, standardized architecture, improved productivity, cost-saving</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Compositional reuse</td>
<td>Verbatim</td>
<td>Ad hoc reuse</td>
<td>QN7</td>
<td>Better quality, improved productivity</td>
<td>Reduced productivity</td>
</tr>
<tr>
<td></td>
<td>Framework reuse</td>
<td>Verbatim and modified</td>
<td>Systematic reuse</td>
<td>QN4</td>
<td>Better quality, improved productivity, improved learning</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Template reuse</td>
<td>Modified</td>
<td>Not reported</td>
<td>QN26</td>
<td>Better quality, improved productivity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Architecture reuse and context-dependent module reuse</td>
<td>Verbatim and modified</td>
<td>Systematic reuse</td>
<td>QN5</td>
<td>Better quality, better maintainability, improved productivity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Not reported</td>
<td>Verbatim and modified</td>
<td>Systematic reuse</td>
<td>ER2</td>
<td>Better quality, improved productivity, cost-saving, faster time-to-market</td>
<td></td>
</tr>
<tr>
<td><strong>Low-quality</strong></td>
<td>SPL</td>
<td>Modified</td>
<td>Not reported</td>
<td>QN11</td>
<td>Better quality, better maintainability, increased productivity, improved productivity</td>
<td>Reduced quality</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Systematic reuse</td>
<td>ER22, QN23</td>
<td>Improved productivity</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Not reported</td>
<td>QN15</td>
<td>Better quality, reduced productivity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Engine reuse</td>
<td>Not reported</td>
<td>Not reported</td>
<td>QN15</td>
<td>Better quality</td>
<td>Performance decay</td>
</tr>
<tr>
<td></td>
<td>Architecture reuse and context-dependent module reuse</td>
<td>Modified</td>
<td>Not reported</td>
<td>QL30</td>
<td>Improved productivity</td>
<td>Increased learning curve</td>
</tr>
<tr>
<td></td>
<td>Not reported</td>
<td>Verbatim and modified</td>
<td>Systematic reuse</td>
<td>ER27</td>
<td>cost-saving</td>
<td></td>
</tr>
</tbody>
</table>
7.4. Validity threats of the primary studies

Validity threats demonstrate how researchers address their study limitations, which also presents the strength of evidence. Only 11 out of 20 primary studies (excluding ten experience reports) - two qualitative and nine quantitative studies, reported validity threats. Seven of these 11 primary studies have discussed the validity threats extensively, while the other four only mentioned some threats without detailed descriptions. We discuss the reported threats to validity in four types as follows.

**Construct validity** is concerned with whether the selected metrics or questionnaire instruments reflect the intended construct. For quantitative studies, researchers discuss construct validity from four perspectives: study design, justification of the selected metrics and data, confounding factors, and mitigating solutions. Seven primary studies discussed construct validity. Morisio et al. (QN4 [2]) justified how the selected metrics measure the target attributes — productivity, quality, and learning. In addition, they also described the problem of measuring developers’ learning and provided mitigate solutions — using the learning metrics used in manufacturing. Gupta et al. (QN21 [43]) justified that the selected data helped answer the research questions, and they triangulated the results with interviews. Goldin and Berry (QN20 [53]) also mentioned interview triangulation to address the understanding of the measurements and their purpose. Deniz and Bilgen (QN15 [21]) discussed the reliability of the data in terms of whether it is measured and collected according to the theoretical constructs. Mihale-Wilson et al. (QN29 [48]) lacked revenue and cost information for the profit impact of reusing components in location-based games. Gupta et al. (QN21 [43]) selected the metrics based on the lesson learned from existing literature. As for qualitative studies, researchers discuss construct validity in terms of the construction of the questionnaire.

**Internal validity** reflects the causal relationship the researchers are investigating is not influenced by other factors. In quantitative studies, the researchers discuss mainly three types of confounding factors: missing, inconsistent, and wrong data; the causality between the input and the responses; and confounding factors to the investigated factor. 11 primary studies reported internal validity. Mohagheghi et al. (QN13 [42]), Mohagheghi et al. (QN9 [13]), and Gupta et al. (QN21 [43]) discussed the missing and incomplete data threats due to the reporting process in the company, and provided mitigation and its justifications. Succi et al. (QNI1 [1]) also mentioned that the data collection relies much on the company’s standard procedure. Morisio et al. (QN4 [2]) and Goldin and Berry (QN20 [53]) have mentioned the causality in design and the assumption causality. Mao et al. (QN26 [46]) referred to the sources could not afford to have two teams with similar experience levels develop the same project with and without following software re-use practices due to limited resources. To address this, the authors asked the same team to develop projects with similar complexity with and without following software re-use practices. Four studies mentioned the confounding factors: programming language, reusable assets have more work from experienced developers or testers, different functionality and constraints between reused and non-reused components (QN9 [13], QN13 [42]), knowledge and skill improvement of developers [46], and enthusiasm of trying the new approach (QN20 [53]). The internal threats in qualitative studies are related to whether participants answered the questionnaire truthfully.

**External validity** refers to what extent the results can be generalized to another context. Ten primary studies have reported external validity or generalizability. All ten primary studies discussed a common limitation: they noted that the results were limited in the investigated company context. Considering the specific context and sampling size of the studied object or participants, five of these ten studies (QL14 [16], QL30 [52], QN2 [4], QN21 [43], QN15 [21]) mentioned the results cannot be generalized to other companies, while the other five (QN1 [1], QN9 [13], QN13 [42], QN29 [48], QN20 [53]) claimed that the results could be applicable for similar projects with similar context.

**Conclusion validity** refers to whether the results are statistically significant or reliable. Seven primary studies discussed the conclusion validity. Mohagheghi et al. (QN9 [13]) and Gupta et al. (QN21 [43]) discussed the factors that may influence the results. Mohagheghi et al. (QN9 [13]) acknowledged that different ways of handling user interface in reused and non-reused components could be internal validity in measuring the reuse impact. In addition, they also acknowledged the limitations of the defects coverage in the collected Trouble Reports. Gupta et al. (QN21 [43]) and Mohagheghi et al. (QN9 [13]) mentioned that the developers’ experiences and skill levels would not be a threat since the assets under study are developed within the same development unit. Slynxstad et al. (QL14 [16]), Succi et al. (QNI1 [1]), and Morisio et al. (QN4 [2]) discussed the statistic significance in regards to the population and confirmed the value of the results. Goldin and Berry (QN20 [53]) and Deniz and Bilgen (QN15 [21]) have discussed the study’s replicability when conducted by different researchers or analysts and claimed that the results would be similar.

**RQ4 Key findings:** We customized the quality assessment checklist developed by Dybä and Dingsøyr [32] and assessed the quality of the included 30 primary studies. The key findings about the quality of the included primary studies are as follows.
• Most of the primary studies (20 out of 30) are of moderate quality. Four primary studies are of high quality, and six are of low quality.
• Better quality and improved productivity are reported in all quality studies. All four high-quality studies reported that software reuse benefits quality and one reported that it benefits productivity. The results showed strong evidence behind the claims that software reuse improves quality and productivity. Moreover, the studies’ rigor behind the claim that software reuse improves quality is better than productivity.
• Our validity threats results showed that researchers need to pay more attention to reporting validity threats. We found a relatively low number of included quantitative and qualitative studies (11 out of 20, excluding ten experience reports) reported validity threats, and not all of the 11 primary studies reported validity thoroughly.
• Experience reports often get lower scores in data collection and analysis criteria, which require reporting formal data collection and analysis methods.
• Quantitative studies have lower scores in research design and reflexivity, which require discussing and justifying why the authors decided which research method to use and whether their roles influence sample recruitment and data selection.

8. Discussion

This section discusses the difference between our study findings and other secondary studies and the study implications for researchers and practitioners.

8.1. Comparison with existing secondary studies on software reuse costs and benefits

We mainly discuss our findings with secondary studies from Mohagheghi and Conradi [17] and Barros et al. [20]. In the primary studies, the most frequently adopted reuse approaches are SPL and reuse from repository. The results of Mohagheghi and Conradi [17] showed more studies are reusing libraries or repositories, while Barros et al. [20] identified component-based development and SPL are the most practiced reuse approaches. In addition to the reuse approaches identified by Mohagheghi and Conradi [17] and Barros et al. [20], we found a new reuse approach: engine reuse [21], which refers to reuse an automated process that generates or finds/selects the components with the needed input information. The emerging engine reuse may result from the evolving software engineering technology. With time passing by, more reuse approaches may arise, such as microservices reuse [55] and InnerSource reuse [56]. Regarding reuse types, our results showed that verbatim reuse is applied more than modified reuse, and ten primary studies (43%) used a combination of verbatim and modified reuse. Similar findings were observed in Mohagheghi and Conradi [17], where five studies used verbatim reuse, and six used a combination of verbatim and modified reuse in their included 11 primary studies. As for reuse methods, like Mohagheghi and Conradi [17] and Barros et al. [20], we found that the included industrial cases investigated systematic reuse more than the other reuse methods — 20 reuse cases followed systematic reuse (74%), while four followed ad hoc reuse (15%) and three followed controlled reuse (11%). In addition, the primary studies report 18 domain reuse cases, which is more than infrastructure (five reuse cases) and architecture reuse (one reuse case), aligning with the results of Mohagheghi and Conradi [17].

We also investigated the comparison scheme that the primary studies used to evaluate software reuse costs and benefits. The most common reuse comparisons are three types: comparing the original project developed from scratch and the projects reused the original project [8, 13,43,44,48,50,51,53], comparing the assets that follow or do not follow a reuse approach [1–6], and comparing the assets with different reuse rates [15,21,41,45,47]. These three comparison schemes are similar to the ones Mohagheghi and Conradi [17] extracted — whether development happens with or without systematic reuse and reuse rate. Few studies [12,23] investigated the reuse impact based on modification rate, which is also mentioned by Mohagheghi and Conradi [17]. In addition to what Mohagheghi and Conradi [17] reported, the primary studies also compared the reuse impacts based on the reuse methods (whether following systematic reuse) [9,10,14] and between the reused and new assets in the same project [7,13,42].

The most investigated software reuse benefits in the primary studies are better product quality and improved productivity, which align with what Mohagheghi and Conradi [17] and Barros et al. [20] reported. We found one additional software reuse benefit: prolonged lifecycle of reused product (game) [48]. Like Mohagheghi and Conradi [17] and Barros et al. [20], we also noted that fewer studies reported software reuse costs: two out of 11 primary studies in Mohagheghi and Conradi [17], two out of 49 primary studies in Barros et al. [20] and seven out of 30 primary studies in our systematic literature review. In addition, we also assessed the study quality to understand the strength of the evidence behind the identified software reuse costs and benefits. The results showed strong evidence behind the claims that software reuse improves quality and productivity.

Regarding reuse methods, our results showed that more primary studies investigated systematic reuse and its benefits (19 primary studies, 76%) over controlled reuse (three primary studies, 12%) and ad hoc reuse (three primary studies, 12%), which aligns with the results of Barros et al. [20]. As for reuse approaches, our results showed that all reuse approaches benefit the companies in quality, and SPL and reuse from repository are most likely to benefit companies. Barros et al. [20] found that component-based development, SPL, and model-based development can all benefit product quality, productivity, development cost, and development time, whereas component-based development has more evidence to benefit companies.

The primary studies have used different metrics for measuring software reuse costs and benefits, especially for quality and productivity. In the primary studies, defect density, complexity, number of faults/defects/bugs, and the effort to fix them are the most common metrics used to measure software quality, which is in line with what Mohagheghi and Conradi [17] reported. In addition to these metrics, some primary studies have instead used software coupling [3,21], the developers’ [41] and customers’ [1,9] views on the product quality to measure quality. Like Mohagheghi and Conradi [17], the extracted productivity metrics are mostly used to measure the effort spent developing and designing reusable assets. Moreover, we also found that the primary studies measure the testing effort [9] and the amount of produced assets [23]. The metrics used to measure maintainability are the same as what Mohagheghi and Conradi [17] reported. Compared to Mohagheghi et al. [17], we also reported metrics used to measure software reuse effects other than software quality, productivity, and economics, e.g., learning and architecture conformance.

We extracted six monetized cost-saving metrics from primary studies. However, we found that those monetized cost-saving metrics lack clear definitions (see Section 6). Barros et al. [20] noted similar findings that their included primary studies did not monetize the costs of development time reduction, quality increase, and faster time-to-market. Quantifying the reuse benefits monetarily is important, and it is a more intuitive way for companies in decision-making.

8.2. Study implication for researchers and practitioners

Based on our results, this subsection presents the possible implications for practitioners and researchers.

• Quantifying the reuse benefits monetarily is important, providing a more intuitive way for companies to make decisions. As discussed in Section 8.1, the extracted six monetized cost-saving...
Our review provides practitioners evidence that software reuse benefits and costs are measured.

- Most primary studies are about software reuse benefits, with only a few focusing on costs. There is a need to investigate more into software reuse costs, especially the tradeoff between software reuse costs and benefits (e.g., the tradeoff between quality increases regarding overall defect density and quality decrease regarding the number of severe defects or the defects that take more time to fix).

- Mäkitalo et al. [57] found that opportunistic reuse is getting increasingly popular, especially reusing from open source software. It will be interesting to investigate the differences in reusing external and internal reusable assets.

- Some identified costs and benefits from qualitative studies need to be validated by quantitative data, such as increased learning curve and standardized architecture.

- Researchers should improve their quality of reporting. Researchers need to report all necessary characteristics and validity threats to help understand the context of the investigated case, which further contributes to the generalizability of the study findings. According to the quality assessment results, the researchers can improve the quality of their reports by justifying the reason for their selected methods, improving sampling, and reflecting more on their role implications to the study.

- The evolution of technologies and methods, such as microservices and InnerSource, may impact software reuse differently. Microservices is one type of reusable unit, and it benefits software reuse [55,58]. InnerSource benefits software reuse [56,59], and its shared space is closely related to reuse from repository. However, the studies above only mentioned microservices and InnerSource benefits software reuse. Qualitative and quantitative evidence about how microservices and InnerSource influence software reuse has not yet been investigated. Our systematic literature review did not find studies reporting software reuse costs and benefits from InnerSource and microservice. The reasons for not finding such studies are as follows: (1.) The identified systematic secondary studies focus mainly on traditional software reuse practices, and (2.) Most of the InnerSource and microservice reuse related studies only mentioned that they improve software reuse, but it is unclear which aspects and to what extent they improve software reuse. (3.) Microservices and InnerSource are relatively new approaches. Chen et al. [11] conducted an exploratory case study to address this gap and investigated the impact of microservices and InnerSource on software reuse from the practitioners’ views. However, more software reuse investigations on InnerSource and microservices are needed.

- Our review provides practitioners evidence that software reuse benefits companies, especially in improving software quality and productivity. Our sample showed that development with reuse reported all of the identified benefits with only a few costs. Developing reusable assets has upfront costs [7,14], which will be paid by reusing them to develop other products [5,52]. In addition, Baldassarre et al. [4] found that a systematic way of reuse will improve the productivity in development for reuse compared to conventional development. Systematic reuse is the most likely optimized option compared to other software reuse practices. Reusing as-is - verbatim reuse has more benefits than modified reuse. Modified reuse is needed when the assets cannot be reused as is. However, extensive modification is not recommended due to its higher risk in quality issues. No matter what reuse approaches the companies use, the discoverability of the reusable assets is important. Investigating how companies ensure the discoverability of reusable assets will be an interesting topic.

9. Conclusion and future work

In this study, we followed the guidelines from Kitchenham et al. [28] and conducted a systematic literature review on software reuse characteristics, costs, and benefits. In total, we included 30 primary studies.

Within these 30 primary studies, 29 reported software reuse benefits, and seven reported software reuse costs. We identified nine software reuse benefits and six software reuse costs. The most reported software reuse benefits are better quality and improved productivity. We cannot draw any conclusions on software reuse costs because of their limited number of studies. We identified eight reuse approaches, of which SPL and reuse from repository were the most reported. Almost all reuse approaches resulted in better quality and improved productivity. Verbatim reuse and systematic reuse were used the most and benefited the companies over the other reuse types and methods.

We extracted all the metrics used to evaluate software reuse costs and benefits and found that the primary studies adopted multiple metrics to evaluate quality and productivity. More metrics are needed to validate other identified software reuse costs and benefits. Software reuse impacts on one attribute may vary according to the metrics from different aspects. For example, the primary studies showed that software reuse helped decrease the overall software defect density, but certain defect types and critical defects have increased.

We also examined the quality of the reviewed primary studies using a quality assessment checklist and categorized them into different study types. The quality assessment results showed that most primary studies are of moderate quality. All types of quality studies reported that software reuse benefits quality and productivity, indicating there is strong evidence behind the claims that software reuse benefits product quality and productivity.

We identified some research gaps for future research as follows:

- To investigate what software reuse costs and benefits the microservice and InnerSource can bring to the companies.
- To provide the quantitative evidence for the software reuse benefits that were not measured: standardized architecture and increased learning curve.
- To provide more empirical evidence on the less reported software reuse costs and benefits.
- To investigate how to monetize the software reuse costs.
- To investigate which context information is important to be reported in software reuse cases.

CRediT authorship contribution statement

Xingru Chen: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Validation, Visualization, Writing – original draft, Writing – review & editing, Project administration. Muhammad Usman: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Supervision, Validation, Visualization, Writing – review & editing. Deepika Badampudi: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Supervision, Visualization, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Acknowledgments

This research is supported by the Knowledge Foundation, Sweden through the OSIR project (reference number 20190081) at Blekinge Institute of Technology, Sweden.

References


