An Empirical Analysis of the Costs of Clone- and Platform-Oriented Software Reuse

Jacob Krüger
University of Toronto, Canada
Otto-von-Guericke University Magdeburg, Germany
jkrueger@ovgu.de

Thorsten Berger
Chalmers | University of Gothenburg, Sweden
bergert@chalmers.se

ABSTRACT
Software reuse lowers development costs and improves the quality of software systems. Two strategies are common: clone & own (copying and adapting a system) and platform-oriented reuse (building a configurable platform). The former is readily available, flexible, and initially cheap, but does not scale with the frequency of reuse, imposing high maintenance costs. The latter scales, but imposes high upfront investments for building the platform, and reduces flexibility. As such, each strategy has distinctive advantages and disadvantages, imposing different development activities and software architectures. Deciding for one strategy is a core decision with long-term impact on an organization’s software development. Unfortunately, the strategies’ costs are not well-understood—not surprisingly, given the lack of systematically elicited empirical data, which is difficult to collect. We present an empirical study of the development activities, costs, cost factors, and benefits associated with either reuse strategy. For this purpose, we combine quantitative and qualitative data that we triangulated from 26 interviews at a large organization and a systematic literature review covering 57 publications. Our study both confirms and refutes common hypotheses on software reuse. For instance, we confirm that developing for platform-oriented reuse is more expensive, but simultaneously reduces reuse costs; and that platform-orientation results in higher code quality compared to clone & own. Surprisingly, refuting common hypotheses, we find that change propagation can be more expensive in a platform, that platforms can facilitate the advancement into innovative markets, and that there is no strict distinction of clone & own and platform-oriented reuse in practice.

CCS CONCEPTS
• Software and its engineering → Reusability; Risk management; Software design tradeoffs.

KEYWORDS
economics, software reuse, empirical study, clone & own, software product line, platform engineering

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA
© 2020 Copyright held by the owner/authors(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7843-1/20/11...
https://doi.org/10.1145/3368089.3409684

1 INTRODUCTION
Software reuse is one of the most important practices to save development costs, increase the quality, and reduce the time-to-market of software systems [62, 100]. The idea is to avoid re-implementing existing functionalities (a.k.a., features [1, 8, 91]), but instead to reuse them for new systems (a.k.a., variants), which otherwise need to be developed completely anew. Various reuse strategies exist [1, 62], typically classified into copy & paste, clone & own, and platform-orientation. Copy & paste reuses small code snippets to solve similar or identical problems within the same or another system. In contrast, the other two strategies reuse a whole system or its parts (e.g., components, files, packages, or modules) to engineer a new, but similar variant that is customized to individual stakeholder requirements, such as hardware, features, and runtime environments, or non-functional aspects, such as cost, performance, regulations, or energy consumption. We focus on reuse in the large and, therefore, the latter two strategies.

In the clone & own strategy (a.k.a., ad hoc, opportunistic, or scavenging reuse; or cloning in the large), developers create a copy of an existing system and adapt that copy to new requirements [1, 31, 99]. As a cheap and readily available strategy, organizations typically start with clone & own, which is well-supported by version-control systems, such as Git (branching and merging), and by software-hosting platforms, such as GitHub (forking and pull requests) [39, 71, 99]. However, with an increasing number of cloned variants, maintenance easily becomes a challenge, for instance, when developers need to propagate changes, fix bugs, or features. Developers easily lose their overview understanding of the variants and are challenged by substantial maintenance overheads [7, 10, 14, 90].

In the platform-oriented strategy, developers implement a single code base from which they can derive individual variants, typically using methods and tools known from software product-line engineering (SPL) [1, 24, 29, 104]. A platform employs variability mechanisms—techniques to implement variation points (e.g., preprocessors, components, or runtime parameters), typically controlled by features. The features abstractly represent the variability and are typically organized in a feature model [29, 56, 87]; a tree-like structure of features and their constraints. Using, for instance, configurator tools or dedicated build systems, developers derive individual variants (i.e., reuse software) by enabling or disabling
features. This strategy is usually advocated for systems with many variants, including highly configurable systems, such as the Linux kernel, which boasts over 15,000 configuration options today, allowing it to run on a range of hardware environments, from small embedded devices to large super-computer clusters. Unfortunately, adopting a platform requires large initial investments into architecting and creating the platform, adopting SPL/E methods and tools (e.g., variability mechanisms, feature models, configurators), as well as training and (re-)organizing development teams, before benefits can be achieved, such as substantially reduced time-to-market through the automated derivation of new variants as well as reduced maintenance efforts [60, 66, 67, 95, 104]. In case of migrating from clone & own, an organization needs to re-engineer software not intended for reuse into reusable features, impacting not only the software architecture, but also the business strategy, organizational structure, and development process [79].

Over the last decades, numerous studies focused on software reuse [4, 5, 11, 16, 19–25, 33–37, 42, 50, 66, 72, 81–84, 97, 101, 103, 104], many of which also describe the migration from clone & own towards a software platform as the most common way to adopt platforms in practice [9, 31, 65]. Such studies indicate that the maintenance burden of clone & own becomes the main problem leading organizations to adopt a platform strategy. Still, to the best of our knowledge, most studies discuss the advantages and disadvantages of either strategy based on personal experiences and educated guesses. The fundamental question remains: What costs are associated with either reuse strategy, and under what circumstances is which strategy preferable? Answering this question based on empirical data provides a means for organizations to reason more reliably on several practical decisions—for instance, whether to adopt a platform for a new or existing set of variants. Our long-term goal is to establish qualitative criteria, measurement techniques, and decision models to support the decision making of organizations developing many variants of a software system, as well as to enhance their confidence in their decisions.

We present a study of the activities (e.g., bug fixing), costs (i.e., money or developers’ efforts spent [102]), cost factors (i.e., project properties impacting costs), and benefits (i.e., savings) that are associated with the development process of new variants using either strategy—clone & own and platform-oriented reuse. Specifically, we conducted (1) an interview study with 28 practitioners from a large organization (i.e., Axis) that employs both strategies, and (2) a systematic literature review (SLR) [58, 59]. The SLR complements our interview data with an analysis of the insights and data provided in case studies and experience reports. By triangulating from both sources, we provide consolidated quantitative data, distributions (e.g., costs in per cent), and qualitative arguments on development costs, cost factors, and benefits. We combined the two sources to address the problem that costs in software engineering are hard to quantify and assign to specific activities [43, 53, 55, 102]. In fact, few studies report quantitative data, and often only on a subset of the relevant costs. Since expert estimations are a widely used, reliable estimation method [13, 43, 52, 85], perform comparable to or better than other cost estimation methods (e.g., cost models) [52, 54, 85], and can be improved based on analogies to historical data [13, 53, 54], our two sources provide a reasonable basis for eliciting data. Based on that data, we also discuss the evidence we find for confirming or refuting common hypotheses about software reuse. Our study is of the rare breed that addresses this problem by providing evidence based on empirical data reported for over 100 organizations. As a consequence, our study provides novel insights on processes and costs, as well as a synthesis of existing studies to confirm or refute common hypotheses. Both of these aspects are novel and important, seeing that Axis spent a significant amount of time and effort supporting our study. Moreover, other companies (e.g., Grundfos) initiated discussions with us regarding this study, provided positive feedback indicating that the data is reasonable, and have used our results to analyze their own processes, adding evidence regarding the importance of our study.

In summary, we contribute:

- Empirical data on the costs and cost factors associated with the activities of clone & own and platform-oriented software reuse.
- A discussion of evidence we find for confirming or refuting common hypotheses on software reuse.
- A replication package with our interview guide and the data we analyzed as an open-source dataset.

Our results show several differences in the activities and costs of clone & own and platform-oriented software reuse that have not been clearly described in the literature, yet. For instance, our results confirm the common hypotheses that a platform improves quality and reduces development costs even more than clone & own. Still, our results also indicate that clone & own is more advantageous than usually considered in research. Moreover, we identified findings that contradict each other (i.e., are inconclusive) and refute common hypotheses, such as cheaper change propagation in a platform. So, our results help to reason about the costs associated with development activities and compare these between the two strategies, supporting organizations in their decision making. Finally, our data can help researchers and tool vendors to focus their efforts on developing new techniques tackling the most important activities, namely coordination, gaining knowledge about the reusable system, solving ripple effects due to dependencies, and adopting mixed processes for clone & own and platform-oriented reuse.

The remainder of this paper is structured as follows. In Section 2, we describe the methodology we employed for our interviews and our SLR, with Section 2.4 including the related work of this paper. Within Section 3, we report and discuss the results we obtained for each of our research objectives. Finally, we discuss the threats to validity in Section 4 and conclude in Section 5.

## 2 METHODOLOGY

In this section, we describe our conceptual framework, research objectives, and methods for eliciting data from our two sources.

### 2.1 Conceptual Framework

We provide an overview of our conceptual framework in Figure 1, relating the concepts we are concerned with to each other. A project is the scope in which an organization develops a new software variant. Based on a set of requirements, the organization defines the reuse strategy it employs during the project: clone & own or a platform. The reuse strategy determines a concrete development process.
that ends with the delivery of the specified variant. A development process comprises a number of activities, such as scoping the variant or bug fixing, performed by the organization’s developers. Each activity involves costs, for which we consider monetary spendings as well as developers’ effort (e.g., person hours) [102]. Finally, cost factors (a.k.a., cost drivers) represent any project property that impacts the costs of implementing the variant. These factors may be related to, for example, the reuse strategy, specific activities, and developers—and can impact each other (e.g., a higher modularity of a platform may require fewer design adaptations for a new variant).

2.2 Research Objectives

We address three research objectives (RO) in our study:

**RO1** Identify the process and its respective activities that practitioners employ to reuse software for new variants.

During our collaborations with industrial practitioners [8, 10, 70, 87], we experienced that the idea of using either pure clone & own or a full-fledged platform rarely applies. These experiences motivated this objective of identifying the processes and activities of software reuse in practice.

**RO2** Identify the costs associated with the activities.

Based on our data, we aimed to identify the impact of either reuse strategy on the costs of the activities identified. This information can help to decide for a reuse strategy and to determine activities that are more challenging to perform.

**RO3** Analyze the cost factors that impact either reuse strategy:

Finally, we aimed to understand the cost factors impacting the reuse strategies, which helps to identify those that are important to consider for cost estimations and can potentially be altered (e.g., involving more developers) to reduce costs.

So, we analyze cost factors also based on economical benefits (i.e., reduced costs) that we identified.

Next, we describe how we elicited data to address these objectives.

2.3 Collecting Data with Interviews

We conducted interviews with practitioners to collect data regarding our research objectives. In Table 1, we summarize our interviews, whose identifiers (column ID) we use as references.

**Interviewees.** Our interview study relied on a close collaboration with a system architect and a product manager (i.e., interviews I₈ and I₉) at **Axis Communications AB** who have an overview of the organization’s development practices and an interest in our study. Axis develops network equipment, including network cameras, print servers, camera servers, scanner servers, and storage servers. Together with our contacts, we sampled 26 interviewees—who we identified as knowledgeable experts during our discussions—with various roles (e.g., software engineers, technical leads, firmware architects) to obtain a broad perspective on their reuse practices. The majority of our interviewees is involved in the development of the organization’s large portfolio of network cameras. Our interviewees reported between three to over 20 years of experiences in one or both reuse strategies—considering solely their current position, while several reported to also have experiences from previous organizations. We usually interviewed one interviewee at a time, but twice we involved two interviewees in one interview (i.e., I₇, I₁₂), and regularly discussed our objectives with both of our main contact persons (i.e., I₈, I₉) at Axis at the same time.

**Interview Design.** We structured our 26 interviews into three different phases: exploration, process definition, and cost assessment. During the first two phases, we conducted unstructured and semi-structured interviews without fixed guidelines, but instead had open discussions, taking notes in parallel. For the third phase, we used our insights to construct a guide for semi-structured interviews. During this phase, we recorded all but one interview (personal preference of the interviewee) and transcribed them later. In addition, we took notes during each interview. For all interviews, we allowed our interviewees to look up actual data if they needed or wanted to, enriching their knowledge with some concrete numbers, for instance, on variant sizes, development histories, or team compositions. However, in most cases our interviewees relied on their knowledge and experience.

### Table 1: Overview of the interviews we conducted.

<table>
<thead>
<tr>
<th>ID</th>
<th>Ph.</th>
<th>h</th>
<th>Subjects</th>
<th>Strategy</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>I₁</td>
<td>EXP</td>
<td>~0.5</td>
<td>System architect</td>
<td>P</td>
<td>Qual.</td>
</tr>
<tr>
<td>I₂</td>
<td>EXP</td>
<td>~0.5</td>
<td>Software engineer</td>
<td>C&amp;O</td>
<td>Qual.</td>
</tr>
<tr>
<td>I₃</td>
<td>EXP</td>
<td>~0.5</td>
<td>Release engineer</td>
<td>P</td>
<td>Qual.</td>
</tr>
<tr>
<td>I₄</td>
<td>EXP</td>
<td>~0.5</td>
<td>Technical lead</td>
<td>C&amp;O→P</td>
<td>Qual.</td>
</tr>
<tr>
<td>I₅</td>
<td>EXP</td>
<td>~0.5</td>
<td>Technical lead</td>
<td>C&amp;O→P</td>
<td>Qual.</td>
</tr>
<tr>
<td>I₆</td>
<td>EXP</td>
<td>~0.5</td>
<td>Project manager</td>
<td>C&amp;O→P</td>
<td>Qual.</td>
</tr>
<tr>
<td>I₇</td>
<td>EXP</td>
<td>~0.5</td>
<td>2 Software engineers</td>
<td>C&amp;O→P</td>
<td>Qual.</td>
</tr>
<tr>
<td>I₈</td>
<td>PD</td>
<td>&gt;3</td>
<td>Firmware architect</td>
<td>C&amp;O→P</td>
<td>Qual.</td>
</tr>
<tr>
<td>I₉</td>
<td>PD</td>
<td>&gt;3</td>
<td>Software engineer</td>
<td>P</td>
<td>Qual.</td>
</tr>
<tr>
<td>I₁₀</td>
<td>PD</td>
<td>~1</td>
<td>System architect</td>
<td>P</td>
<td>Qual.</td>
</tr>
<tr>
<td>I₁¹</td>
<td>PD</td>
<td>~1</td>
<td>Software engineer</td>
<td>C&amp;O→P</td>
<td>Qual.</td>
</tr>
<tr>
<td>I₁²</td>
<td>PD</td>
<td>~1</td>
<td>2 Software engineers</td>
<td>C&amp;O→P</td>
<td>Qual.</td>
</tr>
<tr>
<td>I₁₃</td>
<td>CA</td>
<td>~1</td>
<td>Firmware developer</td>
<td>C&amp;O→P</td>
<td>Qual. &amp; Quan.</td>
</tr>
<tr>
<td>I₁₄</td>
<td>CA</td>
<td>~1</td>
<td>Software developer</td>
<td>C&amp;O</td>
<td>Qual. &amp; Quan.</td>
</tr>
<tr>
<td>I₁₅</td>
<td>CA</td>
<td>~1</td>
<td>Technical lead</td>
<td>C&amp;O→P</td>
<td>Qual. &amp; Quan.</td>
</tr>
<tr>
<td>I₁₆</td>
<td>CA</td>
<td>~1</td>
<td>Technical lead</td>
<td>C&amp;O→P</td>
<td>Qual. &amp; Quan.</td>
</tr>
<tr>
<td>I₁₇</td>
<td>CA</td>
<td>~1</td>
<td>Software developer</td>
<td>C&amp;O→P</td>
<td>Qual. &amp; Quan.</td>
</tr>
<tr>
<td>I₁₈</td>
<td>CA</td>
<td>~1</td>
<td>Technical lead</td>
<td>P</td>
<td>Qual. &amp; Quan.</td>
</tr>
<tr>
<td>I₁₉</td>
<td>CA</td>
<td>~1</td>
<td>System architect</td>
<td>C&amp;O</td>
<td>Qual. &amp; Quan.</td>
</tr>
<tr>
<td>I₂₀</td>
<td>CA</td>
<td>~1</td>
<td>Technical lead</td>
<td>C&amp;O→P</td>
<td>Qual. &amp; Quan.</td>
</tr>
<tr>
<td>I₂₁</td>
<td>CA</td>
<td>~1</td>
<td>Software developer</td>
<td>P</td>
<td>Qual. &amp; Quan.</td>
</tr>
<tr>
<td>I₂₂</td>
<td>CA</td>
<td>~1</td>
<td>System architect</td>
<td>C&amp;O</td>
<td>Qual. &amp; Quan.</td>
</tr>
<tr>
<td>I₂₃</td>
<td>CA</td>
<td>~1</td>
<td>Software developer</td>
<td>C&amp;O→P</td>
<td>Qual. &amp; Quan.</td>
</tr>
<tr>
<td>I₂₄</td>
<td>CA</td>
<td>~1</td>
<td>Software developer</td>
<td>C&amp;O→P</td>
<td>Qual. &amp; Quan.</td>
</tr>
<tr>
<td>I₂₅</td>
<td>CA</td>
<td>~1</td>
<td>Firmware architect</td>
<td>C&amp;O→P</td>
<td>Qual.</td>
</tr>
<tr>
<td>I₂₆</td>
<td>CA</td>
<td>~1</td>
<td>Software architect</td>
<td>C&amp;O→P</td>
<td>Qual.</td>
</tr>
</tbody>
</table>

**Phase:** EXPloration: Process Definition; Cost Assessment

C&O: Clone & own; C&O→P: Clone & own migration to platform; C&O→P: Clone & own and platform; P: Platform
We started with seven exploratory interviews that took around half an hour each. During this initial phase, we were interested in understanding the general strategies and policies of reuse at Axis. With our two main contacts, we scoped the remaining two phases of our interview study, also considering previous works and our experiences from collaborating with other organizations.

In the process definition phase, we conducted five semi-structured interviews. Our goal was to refine our previous insights to construct the reuse processes that are employed at Axis, and to identify cost factors. Finally, we agreed on a consolidated reuse process (i.e., integrating clone & own and platform-oriented reuse) and a set of 10 activities together with our contacts, which is why these interviews took far longer than the intended one hour.

In the last phase, we aimed to quantitatively and qualitatively assess the costs and cost factors associated with the activities we identified. We conducted a semi-structured interview guide, which we used for 14 interviews. In each interview, we asked our interviewees to assess the costs of developing a new variant based on their experiences, and to distribute these costs among the activities we identified. Moreover, we asked them to assess the cost factors we collected on a seven-point Likert scale, considering what impact adjusting them would have on costs (i.e., strongly reduces to strongly increases). To this end, we were concerned with any of the two reuse strategies the interviewee worked with, and asked especially those who had experiences with both strategies to explain their perceived differences between both.

Resulting Data. In the exploration and process definition phases, we obtained qualitative data (i.e., natural language descriptions) of the reuse processes employed at the organization and the related cost factors (cf. Section 3.1). During the cost assessment phase, we also obtained quantitative data, namely a dataset for each interviewee and their employed reuse strategies—comprising, for each reuse strategy employed, estimates (in percent) of the cost distribution among the 10 activities (cf. Section 3.2), assessments (Likert scale) of the cost factors’ impact (cf. Section 3.3), or both. We could not obtain all data we asked for:

- One interviewee (I23) joined running projects and, therefore, was not confident in estimating the costs of activities.
- One interviewee (I24) was not confident in assessing the costs of the first two activities (SV and DR in Table 3), so we miss these values once for each, clone & own and platform-oriented reuse.
- Two interviewees (I25, I26) could not provide quantitative data, as they are not involved in variant development, but platform maintenance—for which they reported highly valuable insights. Overall, we received eight and seven quantitative datasets for clone & own and platform-oriented reuse, respectively. Considering cost factors, we obtained one more dataset for each strategy (due to I23). We miss three more values, as interviewees did not feel confident to assess the impact of a cost factor (i.e., team size once for each strategy and the number of teams once for clone & own).

### 2.4 Collecting Data from the Literature

Our SLR focused on qualitatively analyzing the publications identified, without computing or reporting statistics about the publications, which is typically part of SLRs [58, 59]. In Table 2, we show an overview of all publications we selected.

<table>
<thead>
<tr>
<th>S</th>
<th>Ref</th>
<th>Venue</th>
<th>RM</th>
<th>Organizations (Subjects)</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>[47]</td>
<td>CMPSAC’90</td>
<td>MCS</td>
<td>5</td>
<td>C&amp;A→P</td>
</tr>
<tr>
<td>K</td>
<td>[16]</td>
<td>AL’92</td>
<td>ER</td>
<td>IBM</td>
<td>P</td>
</tr>
<tr>
<td>[78]</td>
<td>IEEE Soft’94</td>
<td>MCS</td>
<td>HP</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>[44]</td>
<td>IEEE Soft’95</td>
<td>ER</td>
<td>Matra Cap Systems</td>
<td>C&amp;A</td>
</tr>
<tr>
<td>V</td>
<td>[30]</td>
<td>ARE’S98</td>
<td>ER</td>
<td>ABB</td>
<td>P</td>
</tr>
<tr>
<td>[93]</td>
<td>JSS’98</td>
<td>IS</td>
<td>83</td>
<td>109</td>
<td>P</td>
</tr>
<tr>
<td>V</td>
<td>[75]</td>
<td>SPLC’00</td>
<td>ER</td>
<td>LG</td>
<td>C&amp;A→P</td>
</tr>
<tr>
<td>R</td>
<td>[22]</td>
<td>Tech. Rep’01</td>
<td>ER</td>
<td>US NRO</td>
<td>P</td>
</tr>
<tr>
<td>V</td>
<td>[36]</td>
<td>JSS’01</td>
<td>QE</td>
<td>4 (4)</td>
<td>C&amp;A</td>
</tr>
<tr>
<td>R</td>
<td>[37]</td>
<td>Tech. Rep’01</td>
<td>ER</td>
<td>Market Maker</td>
<td>P</td>
</tr>
<tr>
<td>R</td>
<td>[27]</td>
<td>Tech. Rep’02</td>
<td>ER</td>
<td>DoD-NWUC</td>
<td>P</td>
</tr>
<tr>
<td>V</td>
<td>[18]</td>
<td>PFE’05</td>
<td>ER</td>
<td>Salon</td>
<td>P</td>
</tr>
<tr>
<td>[31]</td>
<td>ICSE’03</td>
<td>ER</td>
<td>Alcatel</td>
<td>C&amp;A→P</td>
<td></td>
</tr>
<tr>
<td>[34]</td>
<td>SPE’05</td>
<td>CS</td>
<td>Deutsche Bank</td>
<td>C&amp;A→P</td>
<td></td>
</tr>
<tr>
<td>[101]</td>
<td>APS’04</td>
<td>CS</td>
<td>Dialect Solutions</td>
<td>C&amp;A→P</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>[91]</td>
<td>Book’05</td>
<td>ERs</td>
<td>HP, Lucent, Siemens</td>
<td>C&amp;A→P, P</td>
</tr>
<tr>
<td>KR</td>
<td>[44]</td>
<td>OOPSLA’06</td>
<td>ER</td>
<td>Engenio</td>
<td>C&amp;A→P</td>
</tr>
<tr>
<td>[61]</td>
<td>SPLC’06</td>
<td>CS</td>
<td>Testo AG</td>
<td>C&amp;A→P</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>[98]</td>
<td>ISESE’06</td>
<td>IS</td>
<td>Statoil ASA (16)</td>
<td>P</td>
</tr>
<tr>
<td>V</td>
<td>[49]</td>
<td>SPLC’07</td>
<td>ER</td>
<td>OTs</td>
<td>C&amp;A→P</td>
</tr>
<tr>
<td>R</td>
<td>[31]</td>
<td>SPLC’07</td>
<td>ER</td>
<td>Danfoss</td>
<td>P</td>
</tr>
<tr>
<td>K</td>
<td>[104]</td>
<td>Book’07</td>
<td>Eras</td>
<td>AKVAAsmart, Bosch, DNV, Market Maker</td>
<td>P</td>
</tr>
</tbody>
</table>

**Source:** Knowledge; Resources; Manual search; Validation with related work
**Research Method:** Case Study; Experience Report; Interview Study; Multi-Case Study; Survey; Quasi-Experiment

**Search Strategy:** Initially, we experimented with different search strings for conducting an automated search. Unfortunately, established search engines (e.g., ACM Digital Library, Google Scholar) provided too many (irrelevant) results in unreliable orders, since the economics of software reuse (i.e., costs and savings) are mentioned as motivation in many publications, but rarely investigated.
in detail. Due to such issues of selecting suitable search strings and the many additional problems of automated searches threatening the reliability and a potential replication of our study [2, 59, 69, 96], we relied on a manual search that encompassed five sources. All 38 publications we identified through the first three sources served as a starting set for a snowballing search [107], and we validated our results against related work (fifth source).

**Source 1: Knowledge.** First, we analyzed publications that we knew and that we deemed potentially relevant for our SLR. After we applied our inclusion criteria (explained shortly), we selected 15 publications. These publications are marked with a K in Table 2.

**Source 2: Resources.** Clone & own and platform-oriented reuse are core research topics in the field of SPLE. Building on our experience in SPLE, we selected four resources that collect real-world case studies and experience reports on adopting software platforms, usually originating from clone & own: (1) BigLever case-study reports [2] (2) SPLC Hall of Fame; ³ (3) SEI technical reports on software product lines,⁴ and (4) ESPLA catalog [82]. In Table 2, all 24 publications from these sources are marked with an R.

**Source 3: Manual Search.** Using a manual search, we aimed to identify the most recent publications related to our research objectives. We analyzed the last three completed editions (as of July 2019) of relevant conferences (research and industry tracks) and journals through DBLP, namely:

- 2016-18 ESE, ESEC/FSE, ICSE, IST, JSS, IEEE Software, SPE, SPLC, TOSEM, and TSE.
- 2017-19 ICSE, ICSR, and VaMoS.

During this manual search, we found seven publications, marked with an M in Table 2.

**Source 4: Backwards Snowballing.** Using the 38 selected publications, we employed backwards snowballing to identify further publications concerned with our research objectives. We did not limit the snowballing to a specific number of iterations, but continued with any newly identified publication. Still, not all publications were accessible, so we could not check a minority of publications and excluded them from our search. As a result, we identified another 12 relevant publications through backwards snowballing.

**Source 5: Validation Against Related Work.** In our SLR, we also identified SLRs, mapping studies, and surveys that are related to ours. We used these to validate the completeness of our SLR, checking the publications included in the related work against our own sample. We knew the books of Pollet et al. [91] and van der Linden et al. [104], both describing the fundamentals of SPLE as well as providing 11 and 15 industrial case studies as examples, respectively. However, as these have often been published before, we first collected the original publications. Similarly, the paper of Northrop [88] and the book chapter of Krueger and Clemens [64] are introductions to SPLE providing nine and four case studies, respectively, that we already included. Barros-Justo et al. [3] report an SLR assessing what benefits of software reuse have been transferred to industry. Similarly, Bombonatti et al. [15] performed a mapping study to identify how non-functional requirements are supported by software reuse and how they influence each other. In their SLR, Mohagheghi and Conradi [84] are concerned with reuse in general and investigate the economic benefits in the context of industrial case studies. Finally, the Software Engineering Institute recently published a catalog of SPLE publications against which we compared our sample [26].

Our study is complementary to these publications. While we used all of them to validate the completeness of our own SLR, none provides empirical evidence (especially not quantitative data) on the costs associated with clone & own and platform-oriented software reuse. From the related work, we included the seven publications that are marked with a V in Table 2. We only included data directly from the related work if we could not access an original publication, if there was no original publication, or if the related work provided updated data compared to the referenced publications. Note that we employed snowballing also on these new publications.

**Publication Selection.** We selected publications for our study that fulfill the following inclusion criteria (IC):

IC1: Written in English.

IC2: Describes empirical findings on costs of software reuse.

IC3: Concerned with clone & own, a platform, both, or the migration between the two strategies.

IC4: Reports experiences regarding costs, not only estimations.

In addition, we employed two exclusion criteria (EC):

EC1: Not possible to identify whether software was reused based on clone & own or a platform.

EC2: Only cites cost effects reported in previous publications, but does not provide new data.

For EC1, we classified each publication based on keywords, for example, components and the C preprocessor represent the platform strategy, while copying of complete systems or modules represents clone & own (not just copy & paste).

**Quality Assessment.** We analyzed the costs reported in each publication, which were often described as an outcome, insight, or byproduct, for example, in experience reports. So, the data we were interested in is often only reported to show the efficiency of employing a reuse strategy, driven by industrial experiences, not systematically elicited, and sometimes not peer reviewed (e.g., technical reports). We decided to not perform a quality assessment, as the publications have different goals and research methods—challenging a comparison; and the assessment would not add benefits to our analysis—intentionally building on practical experiences [58, 59].

**Data Extraction.** For each publication, we extracted bibliographic data, namely the authors, title, publication venue, and publication year. We extracted further data that was relevant for addressing our research objectives (cf. Table 2): the reuse strategies employed, the research method, the organizations and/or subjects (e.g., for interviews), mentioned cost factors, qualitative insights, and quantitative data. More precisely, we identified concrete statements regarding the costs, benefits, and problems of either reuse strategy, as well as quantified data that was backed up by experiences or measurements. We collected the data in a semi-structured document (provided in our dataset) and used identifiers to trace the data back to its source. To analyze our data, we used an open-card sorting-like method [110] to connect findings on the same reuse strategy and cost factors by identifying synonyms.

**Resulting Data.** We identified the sample of 57 publications we show in Table 2. As expected, few publications (10) are concerned with the costs or benefits of clone & own [48, 73]. In particular, only four publications [36, 44, 47, 89] report five instances of quantified

---

³https://splc.net/fame.html

⁴https://www.sei.cmu.edu/publications/technical-papers
3 RESULTS AND DISCUSSION

In this section, we analyze and discuss the results for each of our research objectives individually. Within the discussion, we highlight how our data relates to common hypotheses and assumptions (e.g., by Knauher et al. [60]) on clone & own and platform-oriented reuse. We denote refutations as $\bigcirc$, confirmations as $\bigtriangleup$, and inconclusive insights as $\bigtriangledown$.

Note that we use median values from our elicited data (interviews and SLR) to provide intuitive examples for the costs and savings that have been reported for both reuse strategies. However, these have to be considered with care, since actual costs depend heavily on an organization’s and project’s properties. We can see this particularly in the large ranges our data spans. However, as our data aligns overall, we consider these examples to provide a good intuition regarding what can be achieved with either reuse strategy.

3.1 RO1: Development Process

We categorized the activities (cf. Table 3) our interviewees reported in the process-definition phase to design and refine the process we show in Figure 2. This abstracted process represents the general development process for new variants at Axis, integrating clone & own and platform-oriented reuse.

**Results.** A major difference to the reuse processes assumed in research is that neither pure clone & own nor pure platform-oriented reuse is employed, but variations of both to varying extents. Usually, Axis scopes features according to new customer requirements and derives a variant (or even the full platform) that is similar to these requirements to separate it from the platform, which is maintained and updated in parallel. On the separated variant, the responsible developers implement the customer requests until the variant can be released. At that point, the developers and platform engineers have to make a decision: On the one hand, they can keep the variant outside of the platform, which would result in what they refer to as a *long-living clone* (i.e., clone & own) that may never be re-integrated into the platform, due to its divergence. On the other hand, and what is mostly done at Axis, the developers can immediately integrate the changes of the variant back into the platform to incorporate new features, wherefore the variant was only a *short-living clone* (i.e., platform-oriented reuse). So, the major difference between clone & own and platform-oriented reuse at Axis is that the maintenance of long-living clones is done by the development team, while short-living clones are maintained by the platform maintainers. Despite the variations in their development process, both reuse strategies essentially comprise the 10 activities we show in Table 3. We use the identifiers to refer to these activities throughout the remaining paper.

**Discussion.** The process we identified is similar to what we experienced [68, 70, 71] and what researchers recently reported for other organizations [51, 105] and open-source communities [99]. However, researchers usually still consider the two reuse strategies as completely separated: developers can either apply clone & own or a platform $\bigcirc$. In contrast, our analysis and such recent publications indicate that a combination of different strategies is regularly used in practice. So, it is problematic to apply findings that are reported, for example, on migrating from clone & own towards a full-fledged software product line, to industry. This finding supports the argument that organizations with a set of similar variants (should) strive towards a platform to some extent (e.g., they may employ clone & own, but support it with automated change propagation [90] or management frameworks [94]).

As a result, the question may not be whether to adopt clone & own or a platform, but for what variant should which strategy be employed? For instance, at Axis, short-living clones are rapidly integrated, and thus represent platform-based reuse. They are quite often separated and re-integrated before major changes on the platform happen. Axis usually uses short-living clones to implement well-defined features for established variants. In contrast, some long-living clones are re-integrated only after years or potentially not at all. During our interviews, we found that clone & own is mostly used to advance independently to new markets or test completely new features. However, a problem of this strategy is that new features or variants may be highly valuable and shall be re-integrated into the platform, which becomes far more expensive, due to the long period of co-evolution $\bigtriangleup$.

**RO2: Development Process and Activities**

We identified a process (cf. Figure 2) for variant development that integrates clone & own and platform-oriented reuse, comprising 10 activities (cf. Table 3). While both strategies are employed in parallel, they can be differentiated for individual variants.

---

**Table 3: Activities we elicited during our interviews.**

<table>
<thead>
<tr>
<th>ID</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SV</td>
<td>Scoping the Variant according to customer requirements.</td>
</tr>
<tr>
<td>DR</td>
<td>Defining the Requirements to specify what must be implemented.</td>
</tr>
<tr>
<td>FEV</td>
<td>Finding an Existing Variant that is most similar to the requirements.</td>
</tr>
<tr>
<td>DF</td>
<td>Designing the Features that implement the requirements.</td>
</tr>
<tr>
<td>PI</td>
<td>Planning the Implementation of features.</td>
</tr>
<tr>
<td>IF</td>
<td>Implementing the Features of the variant.</td>
</tr>
<tr>
<td>QA</td>
<td>Quality Assuring the implemented variant.</td>
</tr>
<tr>
<td>BF</td>
<td>Bug Fixing the variant.</td>
</tr>
<tr>
<td>PBF</td>
<td>Propagating Bug Fixes to other variants and/or the platform.</td>
</tr>
<tr>
<td>CD</td>
<td>Coordinating the Development between developers and teams.</td>
</tr>
</tbody>
</table>

---

Figure 2: Axis’ development process with blue abbreviations relating to Table 3 (CD affects all activities).
On a final note, we have consolidated our insights and experiences regarding the development processes of clone & own and platform orientation to derive promote-pl [70], a round-trip engineering process model for reusing software.

3.2 RO$_2$: Costs of Activities

Next, we describe the costs that are associated with clone & own and platform-oriented reuse. Our analysis for this research objective builds on qualitative insights and relative, quantitative data, for three reasons: First, it is problematic to identify precise data on software costs, which we mitigated by using qualitative insights. Second, absolute numbers are not representative, as they may be completely out of order for different organizations (e.g., start ups versus large organizations). To improve our quantification, we combined the results of our interviews (cf. Figure 3) with those from our SLR (cf. Figure 4). Finally, we avoid repetitions and can clarify connections and discrepancies between the results.

Results. We asked our interviewees during the cost-assessment phase to elicit on the distribution of costs among the development activities for a concrete short-living and/or long-living clone based on their experiences. In Figure 3, we display the resulting distributions. As not all interviewees ended up with a sum of 100 % (min 68 %, max 133 %, avg 99.6 %), we normalized their responses to consider the total costs that went into developing a variant. To avoid faulty data, we verified the normalized data and whether we forgot any activity with each interviewee. However, only some of them stated the integration as additional costs after delivering a variant. To avoid faulty data, we verified the normalized data and whether we forgot any activity with each interviewee. However, only some of them stated the integration as additional costs after delivering a variant, and we asked them to exclude this activity from their assessment.

In our SLR, we identified numerous qualitative insights on the benefits of employing a platform. We also extracted quantitative data on the effects of platform-oriented reuse on activities, measurable benefits, and the total costs in the form of relative values, which we display in Figure 4. The numbers below each entry correspond to the number of quantitative values, and the dots represent medians. We only show results for which we found at least three values, and did not consider the amount (i.e., in LOC) of reuse itself. So, we display the effect of platform-oriented reuse on the three activities (left side): feature development (FD), quality assurance (QA), and variant development (VD). In the middle, we show the quantified benefits of platform-oriented reuse, namely on identified bugs (IB), staffing (S), and time-to-market (TM). To the right, we show the total cost savings that have been reported for platform-oriented reuse. For simplicity, we summarize values for migrations (comparing to clone & own) and general values (comparing to standalone systems). However, we found that the reported costs and benefits did not differ much for these two scenarios, which is why this summarization does not threaten our analysis.

Discussion. We structure the following discussion according to activities that are related (e.g., to set up development). To this end, we refer to the abbreviations we define in Table 3 and the ones in the previous paragraph.

SV, DR, FEV. One interviewee stated on selecting a variant (FEV) for development from their platform:

"That’s something that you understand way in the beginning when you get the requirement[s] for the project. You understand now, it’s a derivative of this one, which will be very obvious [...]"

As we can see in Figure 3, this statement aligns to all activities relating to setting up variant development: The costs for defining requirements (DR) and finding a corresponding variant (FEV) are lower and more narrow compared to clone & own, while the initial scoping (SV) is similar for both. This matches with qualitative findings from our SLR, suggesting that platforms can actually improve developers’ knowledge about variations [16, 24, 27, 34], that such knowledge is elementary for clone & own [6, 30, 32, 34], and that particularly scoping and selecting an existing variant poses problems in clone & own [31, 34].

DF, PI, IF, FD, VD. Clone & own can considerably reduce the costs for developing variants [7, 47, 57, 106]. For instance, Henry and Faller [44] report a cost reduction of 35 %. However, in line with common hypotheses, most data from our SLR indicates that platform-oriented reuse reduces these costs even further [7, 11, 17, 18, 23–25, 27, 92, 97, 98, 103]. In Figure 4, we can see that the
nine studies reporting quantitative data suggest median savings of around 67% for developing a platform variant (VD).

Despite the benefits for developing new variants, we could also confirm that developing features for reuse (FD) is typically more expensive than implementing them for single use [11, 78, 103, 103]. Five studies indicate a median increase of roughly 20%. We can confirm this trend based on our interviews: In Figure 3, we can see that the design (DF) and the implementation (IF) of features are considered to be more expensive for platform-oriented reuse—especially with drastic outliers towards high costs. This supports the argument that platforms will only pay off if features are reused several times. Arguably, we can best summarize this insight with an interview quote on designing features for platform-oriented reuse, highlighting the need to design features in alignment with the platform architecture:

“For short-living clones, we have to design [...] it to be able to be used by others. A long-lived clone, with that, we can ignore that.”

We found five studies suggesting that organizations can develop a larger number of features more efficiently with a platform, but they do not provide costs for single features [12, 33, 45, 101]. For instance, Fogdal et al. [35] report that Danfoss could develop over 2,500 features a year instead of under 300 before adopting a platform. The origin of this benefit is unclear, but considering that feature development is more expensive, it should be caused by other cost factors (e.g., higher quality, more reuse, staffing).

BF, PBF, QA. The research community usually argues that propagating bug fixes (PBF) between variants is one of the major drawbacks of clone & own [31, 34, 72]. Indeed, most studies report that bug fixing is a challenging aspect of clone & own [12, 30, 35, 45, 72, 76, 76], particularly as variants co-evolve, which is why fixes must not only be propagated, but also adapted to the new variant. In contrast, platforms are challenging to test in their entirety [7, 14, 97], but are argued to not require change propagation [33, 101].

Interestingly, our data contradicts this assumption: We can see in Figure 3 that our interviewees consider bug fixing (BF) more expensive in clone & own, but the propagation seems more problematic for a platform. One of our interviewees explained this situation as follows:

“Propagate bug fixes, of course, is longer for the short-living clones because we would actually have to do it. For long-living clones, we don’t do it at all.”

This indicates that bug propagation may be an important issue, but it requires a re-evaluation by the research community. In particular, it seems necessary to consider that change propagation between clones may not be intended, and thus poses no problem—while change propagation in platforms suffers from feature dependencies (cf. Section 3.3). Still, our SLR (median: -60%) and interview results confirm that a platform can drastically reduce the costs for quality assurance (QA) software.

CD. Coordinating the development is a core activity for which we found contradicting reports, considering that researchers usually assume that clone & own provides independence, whereas platforms enforce clearly defined responsibilities and roles. Mostly, coordination and responsibilities (e.g., who owns a feature or variant) are mentioned when problems appear in clone & own [7, 34] or platform-oriented reuse [7, 30, 32, 104]. We identified only one study of Jepsen et al. [51] in which the authors report that a platform facilitates coordination. Our interview data supports this ambiguity: In Figure 3, we can see that coordination is considered as similarly expensive during development for both reuse strategies at a median of 5% for clone & own and 7% for a platform.

Integration. Considering the integration (or migration) of cloned variants into a platform, we received eight cost estimations from our interviewees (four for either strategy), which align to the results of our SLR. Not surprisingly, developers require considerably less time to fully re-integrate a short-living clone into a platform compared to integrating a long-living clone. In particular, the costs heavily depend on the co-evolution of a separated variant compared to the platform, which we also identified as a major problem in our SLR [12, 30, 32, 45, 57, 72], and the extent of the variant’s delta compared to the platform. Again, we can best summarize the problems and costs of re-integrating variants with an interview quote, clearly highlighting the preference of the interviewee towards platform-oriented reuse:

“I think a lot of time is wasted on the long-living clones, because, if you wait one-and-a-half years until you merge, everything [has] changed, maybe. The new Linux kernel, a new version of something else, and then suddenly, your branch is just not working anymore. The longer you wait, the more pain it is. I think you waste a lot of time with those. It’s always better to be up-to-date with master.”

This statement also describes causes for the costs of integrating long-living clones, such as understanding the co-evolution, updating old features, as well as fixing outdated dependencies and bugs.

### RO2: Costs of Activities

Our findings support the argument that the success of reuse depends heavily on platform-orientation [80, 93], with our SLR indicating overall median savings of 52%. The data shows:
- Setting up development is cheaper with a platform.
- Developing features for a platform is more expensive (+20%), but will pay off due to decreased variant-development costs (-67%), outperforming clone & own (-35%).
- A platform increases the quality of the derived software, reducing the costs for quality assurance (-60%).
- Longer co-evolution between variants (and the platform) increases integration costs.
- Surprisingly, change propagation is more costly for a platform.
- Coordination costs are similar for both strategies.

### 3.3 RO3: Cost Factors and Benefits

Finally, we investigated the cost factors impacting both reuse strategies, and their relations. Again, we combined the results from our SLR (cf. Figure 4) with those from our interviews.

**Results.** In Figure 5, we show the responses on the Likert-scale ratings for cost factors that we elicited during our interviews. A positive rating indicates that an increase of the factor (e.g., more reuse for a new variant) is considered to have a positive impact on the costs for a new variant (i.e., reduces costs). In contrast, negative ratings (e.g., a larger delta for the new variant) indicate that our interviewees experienced that an increase of that factor increases costs. We display the assessment separately for clone & own and

ESEC/FSE ’20, November 8–13, 2020, Virtual Event, USA Jacob Krüger and Thorsten Berger

439
platform-oriented reuse. For an easier comparison, we show the average values of both strategies in the middle of each cost factor. **Discussion. Reuse & Delta.** The first two cost factors we analyzed in our interviews are the amount of reusable code and the delta of newly required code for a variant. For both strategies, more reuse reduces costs, while a larger delta increases costs. Interestingly, the impact of reuse is similar for both strategies, with one particular outlier for clone & own indicating a negative impact:

"Basically, for us it would be more of an effort to remove things, stuff we don’t need compared to just having it there."

Change propagation and scalability issues have been well investigated for clone & own, in contrast to this problem of removing unwanted features. Finally, it is interesting that a larger delta is considered to have less impact on platform-oriented reuse, which somewhat contradicts our findings that developing new features is more expensive on a platform (cf. Section 3.2).

**Developers & Staffing.** A factor that we identified in our SLR and interviews is the number of developers involved in development. In Figure 4, we can see that ten publications report a decrease in the staff (S) needed to develop a new variant from a platform with a median reduction of 75% [4, 22, 24, 28, 34, 35, 63, 76, 77, 91]. This supports the established hypothesis that the same number of developers can develop a larger number of variants when using a maintained platform instead of clone & own. At the organization, we experienced a situation similar to another report [61]:

"We had fewer products and fewer developers in the company, the platform was in a horrible state, so you [couldn’t] really use it to release. [It] got more stable, but also the products that were using the platform increased exponentially. Instead of having 10 products on a lousy platform, you have a hundred products on a good platform."

In both cases, the organization’s potential for growth was large enough to require more staff to address additional customer demands. Still, for developing a specific variant, our interviewees generally consider having more developers (cf. Figure 5) as beneficial. The outliers represent cases in which teams get too large and coordination becomes an issue.

**Knowledge.** The knowledge software developers have about a system is essential for their performance. For developing a set of variants, additional challenges arise, as the developers have to also comprehend variability mechanisms (platforms) or what variants exist (clone & own). Missing knowledge seems to be a primary problem of clone & own and may motivate switching towards a platform. After such a migration, developers must only remember a smaller, structured code base, which may explain this benefit. Still, knowledge is also key to establishing a platform and missing knowledge is a major problem raised in two publications [7, 98]. Our interviewees also highly value knowledge as the factor with the most impact for both reuse strategies (cf. Figure 5). For this reason, they also apply specific policies:

"[We are] growing, so we try to have teams with experienced people together with new people."

Overall, our results show that knowledge is a primary factor that heavily impacts the costs of software reuse. Unfortunately, we are not aware of a hypothesis or research that is concerned with understanding the impact and differences of knowledge on the two reuse strategies or migrations between them.

**Teams & Hand Overs.** Related to the previous cost factors, our interviewees indicated that coordination is heavily impacted by clearly defining the size and roles of teams as well as the required hand overs between them:

"I think that the more people you have, it becomes a lot of coordination, and also responsibilities [are] not as clear. If you are three people, it’s hard to hide."

In Figure 5, we can see that the number of teams with defined responsibilities is considered to have a similar positive impact on costs for both reuse strategies. However, the necessary hand overs between teams (e.g., moving a variant from development to quality assurance) is perceived as slightly negative. So, there seem to be no larger differences between both reuse strategies for coordinating in and between teams, aligning to our previous insights.

**Bugs & Quality.** Software reuse is considered to improve the quality of software, and thus reduce the number of bugs. For clone & own, we found three publications that support this argument [6, 36, 106] and three that argue that quality is a problem [12, 45, 48]. Still, for platform-oriented reuse, numerous publications argued that it improves the quality compared to other reuse strategies [24, 30, 33, 38, 51, 61, 76–78, 92, 97, 98, 101, 103, 104], including clone & own. Surprisingly, while the quality of a platform is known to be important for successful reuse, we identified only one paper that stated this as a costly challenge for platform-oriented reuse [61].

We can support both insights with our interview data. For instance, one of our interviewees stated that the organization pushed strongly against clone & own to avoid quality and compatibility issues originating from long-living clones:

"I guess we tried to kill them off because it is a hassle to maintain. [...] If it’s not tested every day, if it’s not daily rebuilt and checked, [...] something is rotting in the code, it’s not being compatible anymore with the platform."

![Figure 5: Ratings of cost factors elicited in interviews.](image-url)
We can further underpin this with our results for RO2, indicating that quality assurance is less costly with platform-oriented reuse. Despite these benefits, several interviewees also stated that the path towards this improvement was challenging. The platform was initially of low quality and, therefore, not trusted. As our interviewees’ assessment (cf. Figure 5) indicates, they consider the quality of the reused system as one of the cost factors with most impact, and it is even more important for platform-oriented reuse ◊.

The improved quality of a platform is used as an argument that fewer bugs must be fixed, further reducing costs. Indeed, as we can see in Figure 4, several studies report that considerably fewer bugs (median: -70 %) are identified in platforms [4, 22–24, 35, 78, 89, 92] ◊. For clone & own, we also found two studies suggesting that this strategy can reduce bugs by 35 % [44] to 66.7 % [89] ◊. Our interviewees assessed that a change in the number of bugs would have roughly the same effect on development costs for both strategies, as we can see in Figure 5.

Modularity/Dependencies. One benefit that we already mentioned is the independence of clone & own [30–32, 101, 106]. This independence allows developers to test variants, implement innovative features or reply faster to customer requests. For example, the organization employs clone & own especially to innovate:

“If it’s a new business, we don’t want it [in the platform] because we don’t want to maintain it.”

This is regularly considered to be the main reason to still employ clone & own. In contrast, two publications report that platforms allow to move easier to new markets [24, 61], which is quite surprising. Kolb et al. [61] state that only the platform allowed them to develop highly complex variants that emerged to new markets. We received an identical response from one of our interviewees:

“I would not be able to have such a complex product if I would not be able to reuse.”

This opposes common hypotheses in research, and we need to better understand what enables organizations to adapt to new and changing markets faster ◊: independence or a reliable platform?

The independence of clone & own is also considered to free developers of dependencies, but the results of our SLR are quite contradicting in this regard ◊. For instance, Bowen [16] argues that platforms can help resolve dependencies between clones that are reported as a problem in several publications [6, 7, 50, 106]. This opposes the hypothesis that platforms can pose dependency problems, for which we found little evidence [7, 72]. The main problem may be best explained with the analysis of open-source platforms by Bogart et al. [14], aligning to a statement of one interviewee:

“Since we’re not part of the platform […] they can sometimes break things they think […] no one is using […]. Then, we found out they broke something that we actually use.”

Missing knowledge, community policies or unintended side effects can easily result in misbehaving or missing features, breaking some variants. However, this does not only influence platforms, but also clone & own, where clones are derived from the platform, but intentionally co-evolved. To investigate this issue further, we asked our interviewees about the importance of having a modular structure in the cloned variants or platform. They consider modularity to have a positive impact on costs with almost identical impact for clone & own and platform-oriented reuse ◊.

Time to Market. Faster time to market is considered to be a main benefit of software reuse and particularly platforms. Four publications confirm faster time to market for clone & own [6, 32, 48], for example, of around 30 % [89] ◊. Still, platform-orientation can drastically outperform clone & own [4, 11, 24, 27, 33, 38, 49, 50, 61, 76–78, 89, 97, 103, 104], as new variants can be delivered instantly, if all required features have been implemented (median: 63 % ◊).

<table>
<thead>
<tr>
<th>RO2: Cost Factors &amp; Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regarding cost factors, our data shows:</td>
</tr>
<tr>
<td>• Reusing more code reduces, while more new code increases costs, with platform-orientation being affected more positively than clone &amp; own for both.</td>
</tr>
<tr>
<td>• The number of developers developing a variant is a challenging factor to assess correctly for both reuse strategies, but platforms allow to develop more features and variants with the same staff.</td>
</tr>
<tr>
<td>• Independent of the reuse strategy, knowledge about the system is an essential cost factor.</td>
</tr>
<tr>
<td>• As coordination is challenging, having teams with defined roles is beneficial for both reuse strategies. Still, our results are slightly negative on the impact of hand overs between teams.</td>
</tr>
<tr>
<td>• The quality of a platform is essential for its success and results in fewer bugs compared to individual systems (-70 %). Still, both reuse strategies benefit similarly from quality.</td>
</tr>
<tr>
<td>• Independence favors clone &amp; own, but can also be achieved with a platform. Surprisingly, both strategies are vulnerable to dependencies and rippling effects, and thus benefit from modularity.</td>
</tr>
<tr>
<td>• Clone &amp; own can reduce the time to market (-30 %), but an established platform can lead to a far larger reduction (-63 %).</td>
</tr>
</tbody>
</table>

4 THREATS TO VALIDITY

We now discuss threats to the construct, internal, external, and conclusion validity, following established guidelines [58, 108].

Construct Validity. We conducted 26 interviews with practitioners who did not use the terminology that has been established in research. To mitigate this threat, we used our exploratory interviews to understand the terms and practices established at Axis. Together with our two contacts, who are familiar with the research terminology, we clarified and unified terms. We used the terms established in the organization to conduct our interviews. Moreover, at least one of the authors was present during all interviews to allow the interviewees to ask about any unclear constructs, and each interview started with an introduction into the purpose and terms of the study.

We extracted data from 57 publications that used different terminology, based on the research focus, domain, and authors. To tackle this issue, we carefully read each publication and extracted keywords to categorize them into one of the two reuse strategies, and to assign our data to the correct costs and cost factors. We aimed to mitigate any threat of wrongly assigning publications or data by using an open-card sorting-like method to unify synonyms. Finally, we aligned this terminology to the one we established for the organization to triangulate the data from both sources.

Internal Validity. It is challenging to precisely assign costs to specific activities and cost factors in software engineering, due to, for instance, the tangling of activities (e.g., bug fixing and bug propagation) or cost factors (e.g., developers’ knowledge and team
composition). Moreover, the costs of developing and particularly reusing software are hard to assign to a specific system, as just copying software has close to zero costs, while the costs of developing features may be distributed among all variants using it. So, costs in software engineering are far more challenging to quantify than in manufacturing, where costs can usually be assigned to a specific product. For those reasons, and because most organizations do not track their costs on a detailed level (e.g., assigning costs to complete projects), quantified data is hard to obtain. We addressed this problem by adopting our research method, triangulating from interviews with experienced software developers and an SLR.

To ensure that we could derive valid results, we used qualitative and quantitative data from two different sources. We conducted the SLR particularly to avoid threats that may be caused by conducting interviews only at one organization. So, we aimed to improve the internal validity of our results by incorporating the experiences of skilled engineers with established research findings. Also, we aimed to improve the completeness of our SLR, verifying our sample against related work. We discussed regularly with our contacts to verify that our data and results were sensible. Finally, we discussed our findings with three practitioners from another large organization (i.e., Grundfos) that employs platform-oriented reuse to perform a sanity check based on their experiences. Nonetheless, a potential threat to the internal validity remains that we may have misinterpreted or collected false data that biases our results.

**External Validity.** An interview study at a single organization is limited in its external validity. Still, many software platforms exhibit similar characteristics, whether they originate from open-source projects or industry [46]. Also, many industrial organizations employ similar reuse practices considering clone & own and platform-oriented reuse [10, 31]. So, while we cannot fully overcome this threat, our results are still important for organizations to understand the costs of their reuse strategies.

We also conducted our SLR to improve the external validity of our study, mitigating the limitations of conducting interviews at a single organization. The SLR includes publications from various domains, levels of maturity, programming languages, countries, and over 100 organizations. So, we argue that the results of our SLR improve the external validity of our study. As we found similar results regarding reuse practices, costs, and cost factors in our interviews, we argue that these are also reliable.

**Conclusion Validity.** We employed interviews and an SLR to mitigate many of the aforementioned threats, and argue that this methodology was appropriate to obtain reliable insights regarding the costs of software reuse. For this reason, we argue that our findings result in an extensive body of knowledge that provides an intuition and guidance for organizations aiming to establish a platform, and researchers to focus their research efforts. Due to confidentially reasons, we cannot provide all of our data, for example, on the interview results and the transcripts. Still, we described our methodology in detail and provide a dataset with all artifacts that allow other researchers to replicate our study.

## 5 Conclusion

In this paper, we reported the combined results of an interview study conducted at a large organization and an SLR. We described the development process and costs associated with the two main reuse strategies for developing variants: clone & own and platform-orientation. Our analysis is the first to provide systematically elicited and substantial empirical data on these reuse strategies. As a result, we found 18 pieces of evidence that confirm common hypotheses on software reuse. The results suggest that platform-orientation is preferable over clone & own, reducing overall costs, improving quality, and allowing for more variants. Still, we also identified seven inconclusive pieces of evidence, and three pieces of evidence that refute established hypotheses on software reuse:

- Platform-orientation and clone & own are not strictly separated practices, which is often assumed in the literature.
- Bug propagation is usually considered to be the main problem of clone & own, but our findings indicate that it can be even more challenging and expensive for a platform.
- Clone & own is usually argued to facilitate innovation compared to a platform, but our findings suggest that platforms can actually be essential to successfully advance into innovative markets.

Overall, besides providing the first substantial, evidence-based body-of-knowledge on costs of the two reuse strategies, we obtained important insights that are relevant for practitioners to consider in their decisions, and for researchers to scope their work. For future research, we are especially interested in detailed investigations of our inconclusive and refuting pieces of evidence, and in deriving a decision model for reuse strategies based on empirical evidence. To this end, we also plan to conduct additional studies with more organizations and other setups (e.g., surveys) to verify our results.

## Acknowledgments

This work has been supported by the German Research Foundation (LE 3382/2-1, SA 465/49-3), the Swedish Research Council Vetenskapsrådet (257822902), the Swedish Software Center, an IFI fellowship of the German Academic Exchange Service, and the Wallenberg Academy.

We thank Umut Tezduyar Lindskog and Fredrik Hugosson for the support, discussions, and feedback. We thank all our interviewees at Axis AB for their helpful insights. Finally, we thank the anonymous reviewers of the research and artifact tracks for valuable feedback that helped us fix smaller bugs and improve the paper.

## References


Feature? A Qualitative Study of Features in Industrial Software Product Lines. In SFLC.


[38] Ronny Koh, Isabel John, Jens Knodel, Dirk Muthig, Uwe Hauß, and Gerald Meier. 2006. Experiences with Product Line Development of Embedded Systems at Testo AG. In SPLC.


An Empirical Analysis of the Costs of Clone- and Platform-Oriented Software Reuse

ESEC/FSE ’20, November 8–13, 2020, Virtual Event, USA


Parastoo Mohagheghi and Reidar Conradi. 2007. Quality, Productivity and Economics of Applying Software Reuse in Statoll ASA. In ISESE.


Byoung Choi. 2000. Domain-Oriented Engineering of Elevator Control Software. In SPLC.


Shohei Sugawara, Yuko Kang, and Bokyung Kim. 2000. Domain-Oriented Engineering of Elevator Control Software. In SPLC.