Comparing reuse practices in two large software-producing companies

Veronika Bauer[a,*], Antonio Vetrò[a,b]

[a] *Institut für Informatik, Technische Universität München, Germany*
[b] *Centro Nexa su Internet & Società, Politecnico di Torino, Italy*

Abstract

Context: Reuse can improve productivity and maintainability in software development. Research has proposed a wide range of methods and techniques. Are these successfully adopted in practice?

Objective: We propose a preliminary answer by integrating two in-depth empirical studies on software reuse at two large software-producing companies.

Method: We compare and interpret the study results with a focus on reuse practices, effects, and context.

Results: Both companies perform pragmatic reuse of code produced within the company, not leveraging other available artefacts. Reusable entities are retrieved from a central repository, if present. Otherwise, direct communication with trusted colleagues is crucial for access.

Reuse processes remain implicit and reflect the development style. In a homogeneous infrastructure-supported context, participants strongly agreed on higher development pace and less maintenance effort as reuse benefits. In a heterogeneous context with fragmented infrastructure, these benefits did not materialize.

Neither case reports statistically significant evidence of negative side effects of reuse nor inhibitors. In both cases, a lack of reuse led to duplicate implementations.

Conclusion: Technological advances have improved the way reuse concepts can be applied in practice. Homogeneity in development process and tool support seem necessary preconditions. Developing and adopting adequate reuse strategies in heterogeneous contexts remains challenging.

Keywords: Software reuse, survey research, technology transfer, empirical, software engineering

*Corresponding author
Email address: `bauerv@in.tum.de` (Veronika Bauer)

1. Introduction

Software reuse, "the use of existing engineering knowledge and artefacts to build new software systems" [23], is considered a key element to reach the goal of delivering high quality software in time and on budget over the entire life cycle of software development, maintenance, and evolution [47, 51, 66, 85].

Starting from the late 1960'ies [58], decades of research efforts have been spent on analyses, methods, tools, and empirical investigations targeting to support practitioners in executing reuse tasks [45, 63, 24].

Many conceptual approaches have been proposed, addressing aspects ranging from the creation and organization of reusable artefacts, e.g. Object-Oriented techniques and programming languages, Components-off-the-shelf (COTS) approaches [7], reuse repositories [43, 35], or software product lines (SPLs) [6], to organizational support and templates e.g. the "experience factory" [4], the "reuse curator model" [22], or the REBOOT approach [43, 82].

Visions were that reuse would soon be effected on an abstract level by means of techniques such as automated code generation or Very High Level Languages [47]. In addition, planned and strategic reuse programs were advocated as most beneficial, once the high initial efforts were completed. Whilst the reuse community had addressed many technical aspects of reuse (in the form of domain engineering frameworks, programming language concepts, reuse libraries, architectures, and generative methods), research on sustaining reuse initiatives and organizational aspects were still to be tackled [24].

However, due to technological limitations, many of these proposals could at the time not be transferred into practice in a feasible way. For instance, the reuse repositories at the core of several approaches proved to be tedious to set up and maintain, involving significant manual intervention [39].

"Component-based software reuse faces an inherent dilemma: in order for the approach to be useful, the repository must contain enough components to support developers, but when many examples are available, finding and choosing appropriate ones becomes troublesome." [35] This led to a significant research effort in classification and retrieval techniques, that, however, could not overcome the challenge of providing practitioners with an effective way to build, populate, and maintain a dedicated reuse repository that could be adapted to meet changing organizational needs [35].

In addition, organizational challenges soon became apparent: Early-on, researchers vocalize one of the most challenging aspect of planned reuse in practice, namely the separation of those who are investing into reuse and those who profit from this investment, which requires a global focus of management transcending the scope of single projects: *"The traditional unit of analysis and control for software managers is the software project, and subsequently the resulting application system. [..] Yet there is a range of insights that can only be attained through the monitoring and management of the software inventory at the level of the entire firm. [..] Reuse, by its nature, is an activity that spans multiple projects and application systems enterprise-wide. To manage such reuse requires monitoring the firm's software at the level of the organization or enterprise." [3]*

Technological evolution, especially the creation of the internet technologies, have since enabled the building of advanced infrastructures, providing an unprecedented accessibility to knowledge and potentially reusable artefacts ranging from code snippets over libraries, components, to services [62].

Furthermore, Software Engineering best practices, such as e.g. requirements engineering, software architecture design, automated quality analyses by means of static analyses, code reviews, and continuous integration, support (to different extents) reuse and avoiding redundancies. Also development paradigms based on the new infrastructure possibilities, such as agile methodologies, Inner Source [88], test driven development, etc., have significantly changed the way software can be developed. This has also lead to a drastic change in the way reuse can be approached in terms of publishing, retrieving, and maintaining reusable entities [24, 39, 88]. As a previous scarcity of reusable items has been replaced by a plethora of available options, research has proposed elaborate tool support to facilitate access and selection of suitable reuse candidates, e.g., by means of code [77, 72, 14, 32, 59] and model recommenders [31]. At the same time, empirical research has started to quantify the benefits of reuse [65]. In addition, the Open Source community has embraced the potential of reuse [84], e.g., in terms of software libraries [33]. Consequentially, the opportunities, challenges, and risks of employing third-party reusables have been studied [11, 73, 74, 15, 61]. In this context, the following questions arise:

How do software producing companies nowadays approach reuse? Which reuse approaches do they choose for development, maintenance, and evolution? Do they proceed in a planned and strategic manner on an abstract level? Is reuse a problem solved in practice?

Current industry encounters provide a variety of insights [10, 9, 8]: Reuse in general is playing a key role in everyday software development, maintenance, and evolution. Furthermore, in some highly specialized domains, Software Product Lines (SPLs) [71] have been successfully adopted, lead to commercial success, and support a high level of reuse [92]. However, literature suggests that for a large number of software producing companies, finding, adopting, and sustaining an adequate reuse strategy remains a challenge [67, 57, 8]. In addition, the means to effect reuse are often limited due to lack of tool support or organizational limitations [21].

To be able to support practitioners in experiencing successful reuse, we need to deepen our understanding of the current state of reuse in practice. In addition, we need to integrate existing evidence and investigate if reported issues of previous studies (e.g., [23, 76, 30, 67, 65, 49, 33, 21, 87]) are still current in today's professional software development[1].

[1]For the selection of studies, we started out from well-known sources, such as textbooks (e.g. [43]) and journal summary papers on reuse adoption and results in practice (e.g. [65]) as well as reports on research projects (e.g. the REBOOT [82] and ESPRIT [30] projects) on industrial accounts on reuse and manually followed the references, as well as searching the citing papers. In addition, we searched the last 10 years of proceedings of the main venue ICSR for current publications related to reuse in practice (e.g. [49, 33]). Additionally, we included

The goal of this paper is to take a first step towards this investigation. To this end, we systematically integrate insights from two recent in-depth industrial case studies on reuse practices. We compare their results and relate them to previous work with the aim to identify open issues potentially relevant for a wider range of companies.

Specifically, we compare observed reuse practices, effects and influence factors in two large and diverse software producing companies (denoted as G and U in the following[2]). We collected information from a total of 138 professional software developers by means of an extensive on-line questionnaire (108 respondents) and interviews (30 participants, 35 hours of interviews). The study at G preceded the study at U and was designed as an exploratory study on the state of software reuse in practice. Both studies confirm the evidence reported in the body of reuse literature regarding the prevalence of reuse being limited to source code.

In this study we present the result of our integration, detailing thoroughly on study design, methodology, company contexts, and the respective findings.

Besides adding value in terms of reliability[3], our findings highlight discriminant factors in the approaches with respect to contexts and related benefits as well as insights on the pre-conditions for successful reuse. When translated to recommendations to practice, such findings lay the foundation for devising successful reuse strategies.

Outline Section 2 provides a brief overview on current studies on software reuse in practice. Section 3 outlines the study goal and derives our research questions. Section 4 then presents in detail the two studies that we integrate according to the methodology described in Section 5. Section 6 reports our results which are subsequently discussed and related to previous research in Section 7. Section 8 details the limitations of this work before Section 9 concludes the paper and proposes future work.

2. Reuse in practice

The alleged benefits of software reuse have not failed to attract the attention of industry, where the need for reduction of redundancies, as well as cost reduction and quality improvements is perceived. In addition, the vision of fostering innovation and market penetration due to shorter production cycles

known papers from different venues that contribute to the question (e.g. [21]). Lastly, we performed an unstructured search on Google scholar to find articles related to software reuse and adoption and practice (e.g. [91]).

[2]Whilst the identity of G is disclosed in [9], company U preferred to remain anonymous.

[3]In addition to our contribution specific to the software reuse community (and to software engineering at large), we would also like to stress the fact that, from an epistemological point of view, the importance of confirmatory studies (and even negative results i.e., lack of effect) has been largely recognised in science (e.g., see [89]), and is becoming more relevant also in the field of empirical software engineering (see, e.g., the special issue on the Empirical Software Engineering Journal on Negative Results).

promised obvious strategic business advantages. Benefits have been reported from successful adoptions: lower cost and faster development [83, 54, 91, 9], higher quality [83], standardized architecture [76, 83], and risk reduction [91] by resorting to known artefacts. Whilst these reports are encouraging, they are outnumbered by reports of unsolved challenges (see Section 2.2).

Literature and practice suggest that adoption of a suitable reuse strategy is challenging: Software reuse takes place in a multi-faceted environment and, thus, incorporates aspects ranging from technical to organizational at different levels of abstraction [23, 54, 8].

Congruence of business goals, context factors, and processes need to be established for reuse to provide the desired economical effects [57]. Options for reuse approaches range from loose to tight reuse [67] that involve different levels of organizational commitment.

In addition, the selection of adopted reuse approaches differs among the various domains of software development: reuse practices in embedded and non-embedded software development differ with component-based approaches and product lines prevailing in embedded systems development, whilst in non-embedded contexts ad-hoc reuse is most frequent [91].

In accordance with our research questions (see Section 3), the remainder of this Section gives an overview of reported success factors, enablers, challenges, and hindrances for reuse adoption. It, furthermore, details briefly on reuse approaches adopted in practice.

2.1. Success factors and enablers

Several works have proposed processes for, or reported on, adoption attempts of structured reuse in practice and deduced respective success and failure factors. In particular, they identified technical as well as non-technical aspects [54, 23, 76] that significantly impact the chances of success in conducting reuse. The following paragraphs give a brief overview on a selection of recent studies.

2.1.1. Technical success factors and enablers

Technical infrastructure: The presence of an adequate technical infrastructure is reported as a key reuse enabler [83, 54] or even prerequisite [87] to conduct software reuse. Infrastructure ranges from repositories for code and the respective documentation, to support for development, quality assurance, configuration management, and deployment. Particularly, the ease of access to reusable entities provided by tool support facilitates reuse [84, 21].

2.1.2. Organizational success factors and enablers

Incentives: In an Open Source context, the *personal conviction of the benefits* provided by reuse incited developers to rely on reuse [84].

Knowledge: *personal networks* as well as *exposure to a variety of projects* reportedly enable Open Source developers to reuse more code as these factors helped them to discover and access the respective reusables [84]. In a closed source environment, *experience of developers* is reported as success factor in some studies (e.g. [54]), whilst it did not impact reuse success in others (e.g. [83]).

Management: Various sources suggest that sustained management commitment is a key enabler for any advanced reuse program [67, 54, 88]. In particular, this management commitment is needed to drive the *modification of non-reuse processes* as well as to create the *awareness of human factors*, e.g. changes in responsibility [88] and adaptation to new processes [67], that impact organizational change (and address them accordingly).

Process: In several studies, the relevance of a systematic reuse process is reported as success factor [54, 76]. Tailoring of non-reuse processes is reported as enabler [67].

Organizations structure: For medium size and large companies, institutionalized reuse by means of a dedicated workforce is reported as success factor [54, 71].

Artefacts: Reuse of artefacts other than code, as well as reuse of already existing artefacts are reported as enabling successful reuse [54, 76].

Quality assurance: High quality of reusable artefacts is a key reuse success factor as it establishes trust with the users [88]. To achieve this, adequate methods should be adopted (e.g. quality models [54] or code reviews [87]).

2.2. Challenges and Inhibitors

Besides technical challenges, a substantial number of organizational and human factors have been identified as potential inhibitors to a successful application of advanced reuse practices [67]. Technical factors include creation, retrieval, modification, and maintenance of reusables. However, these topics are strongly linked to human factors, such as cognitive effort [47], program understanding, and motivation, as well as organizational factors, such as business strategy, management commitment, and company culture [86].

2.2.1. Technical challenges and inhibitors

Creation of reusables: The creation of reusables can be challenging due to several factors. First, determining what reusables should be built by design is non-trivial. It requires a detailed understanding of the envisioned application context to reduce friction when integrating reusables [28][4]. Second, providers must strike a critical balance: on the one hand, a reusable should encapsulate a specific functionality in order to be coherent, understandable, and clearly fit a defined task. On the other hand, it should be as generic as possible to allow being reused in numerous different contexts with little adaptation effort [64].

Technical incompatibility is a strong inhibitor to reuse, denoting problems of interoperability due to incompatible platforms, paradigms, and technologies [80,

[4]According to Greenfield et al. [28], this lack of knowledge about the final context is an enormous challenge when providing reusables that are consumed in an ad-hoc way.

26, 36]. Technical incompatibilities can decrease or annihilate the possibilities of extracting or combining existing parts [46].

Storage and retrieval of reusables: Companies aiming for internal reuse repositories are still facing the challenge of populating and classifying them, which requires a considerable upfront investment and often proves infeasible [39, 8].

Also in the context of Open Source software, challenges in retrieval of reusable entities as one core inhibitor to successful and widespread adoption of reuse in practice [64, 26, 52, 36]. Whilst originally the challenge of retrieval lay in locating and accessing catalogues of reusable entities [64], it is nowadays the number of potential reusable entities that challenges developers aiming to reuse [77, 32].

Technical Infrastructure: On a technical level, the development infrastructure used by a company can significantly impact the way reuse can be approached [87, 8]: the absence of a supporting infrastructure de-facto renders structured reuse impossible, as it hinders developers to access and retrieve reusables in a coordinated and controlled way[21]. Furthermore, advanced infrastructures can mitigate the risk of errors introduced into reusables [9].

2.2.2. Organizational challenges and inhibitors

Many of the technical challenges mentioned above are challenging on the conceptual level. However, they tend to be exacerbated due to the organizational context that embeds them. The following organizational hindrances are particularly prominent in literature:

Organizations structure: the quality of inter-unit relationships has a significant impact on a successful outcome of reuse adoption. Competition, overlapping or unclear responsibilities, priority conflicts, and lack of coordination of reuse activities diminish the likelihood of success of a reuse program [55].

Inertia: product-centric organizations tend to promote a focused view on development. Managers and developers are usually assessed based on the success of their isolated projects, incentivising local optimization that counteract reuse on a company-wide scale [28, 22, 55].

Knowledge: Adoption of advanced reuse is a global topic that requires a clear positioning of the organization [81] and research into current methods and techniques for reuse (which tends to be neglected [55]).

Measurement: Introducing central reuse requires significant resources and collaboration across different organizational units. Without measurement and adequate compensation, this might lead to unwillingness to cooperate [55].

Management: Introducing the required governance strategies for creation, maintenance, and decommission of reusable items can be challenging in the face of heterogeneous preferences and process weaknesses [28]. Adjusting the context, thus, causes additional overhead that tends to be underestimated in the initial planning [55, 67], endangering reuse success.

Economic: Investing into the reusability of software or supporting infrastructure imposes non-negligible costs onto projects and requires firm and long-term support from management to resolve restrictive resource constraints [55, 40, 76].

Disincentives: One of the strongest disincentives is lack of quality of the entities provided for reuse [55]. This inhibitor needs to be overcome by means of transparent quality assurance and clear governance lining out assumptions and guarantees that hold for entities designed for reuse. In addition, the criteria that are applied to assess developers and managers have an impact on their motivation to engage into reuse [22, 87].

2.3. Reuse approaches in practice

The following paragraphs briefly introduce reuse approaches that have reportedly been adopted in practice.

Clone-and-Own is the most frequent realization of pragmatic reuse and denotes a reuse approach that relies on copying and, potentially, modifying of (proven) solutions for the purpose of effective development. It is also known as: *code scavenging* [47], *ad hoc reuse* [19], *opportunistic reuse* [78], and *copy-and-paste* (or *cut-and-paste* or *copy-and-modify*) reuse [50], *pragmatic reuse* [36]. As it has virtually no preconditions on the organizational context, it is applied widely in industry [21]. Depending on the given industrial context, this practice can serve as a disciplined tool [17, 42, 21]. On other occasions, clone-and-own is the only feasible reuse mechanism at disposal due to, e.g., organizational restrictions or absence of supporting technology. However, they incur the risk of inducing errors as well as significantly increasing maintenance efforts [41]. On the conceptual level, the task of finding working code examples among the vast amount of available source code can be a time-consuming challenge [44].

Inner Source

As empirical studies have shown, Open Source projects heavily build on code reuse on the basis of libraries, reaching reuse rates between 30 and 90% [84, 33]. Open Source development relies on transparency, self-selection of tasks, asynchronous communication, and quality assurance. *Inner Source*[5] attempts to transfer this reuse-inducing[6] development style from the Open Source community to industry [88][7]. The key benefits of *Inner Source* lie in the full access of developers to the seed project's source code and the shared responsibility for reusable assets. This transparency and availability serve as a key enabler for reuse. Literature reports instances of successful *Inner Source* e.g. at Hewlett-Packard, Alcatel-Lucent, Philips Healthcare, IBM, and SAP. Pointers to the respective material and a summary of the studies are provided in [87].

Component-based reuse aims to build software systems out of interchangeable components [75], potentially provided by third-party vendors [37], enabled by

[5] Research on the topic dates back only to 2002 and has not yet been very extensive. However, in recent years, the topic has attracted more attention from the research community. [87]

[6] One of the key goals of *Inner Source* is reuse. However, it comprises more than just mechanisms for reuse. In addition, its principles and development practices, as well as the advanced tool support, clearly enable code-based reuse.

[7] Literature knows the phenomenon also as *Progressive Open Source* and *Controlled Source* [20], *Corporate Source* [27], *Corporate Open Source* [29], and *Internal Open Source* [27].

separation of implementation and interfaces, with a possibility of extension via well-defined extension points [37].

Adoptions of *component-based* reuse have been reported [49], particularly in the domains of embedded software development [68]; however, most of the proposed methods and techniques designed to support the approach are currently lacking validation of benefits and accounts of application in practice [38, 2, 90]. *Software Product Lines (SPLs)* aim to *"reduce the overall engineering effort required to produce a collection of similar systems by capitalizing on the commonality among the systems and by formally managing the variation among the systems"* [48]. Implemented successfully, SPLs reportedly lead to very high reuse rates and enable rapid creation and delivery of new product variants [71]. However, adopting a product line approach demands significant maturity of an organization's process and development capabilities. Commercially successful product line implementations are showcased in the *Product Line Hall of Fame*[8].

3. Study goal and research questions

We compare software reuse implementations in two large companies: we look for commonalities and differences in the way the companies perform reuse, on the problems and benefits experienced and the factors which tend to inhibit or enable it. Formally, we define the study goal according to [5]:

> The goal of the study is to *characterise* reuse practices
> for the purpose of *comparing them*
> with respect to *their realisation and effects*
> from the viewpoint of *software professionals*
> in the context of two large *software producing companies.*

Effects refer to positive (*benefits*) and negative (*difficulties*) consequences of the presence or absence of the effected reuse approach. From our goal, we derive two main research questions:

RQ1: Which reuse practices are applied? We assess which (and to what extent) reuse practices and reuse activities are conducted and how they are supported by tools and infrastructures. From our point of view, *reuse practices* refer to how reuse is organised and implemented in a given company context. Consequently, the term encompasses several aspects: the entities that are reused (namely knowledge and artefacts), the process that is followed to create, obtain, and reuse them, as well as the way in which they are reused on the technical level.

RQ2: What are the effects of the respective approaches? We investigate which problems and challenges occur in practice and capture the perceived

[8] http://www.splc.net/fame.html

benefits of and the success factors for reuse. We are especially interested to see whether our findings confirm the consensus of the literature, and whether the two companies report different aspects.

4. Case descriptions

This section introduces the two cases that we subject to comparison in the present study. The two companies under study are named G and U. We detail their context and outline the respective case study designs previously conducted and their main results. Table 1 summarises the context details of both companies[9]. In addition, we provide detail on the material on which we base the comparison in the present paper, and report the challenges related to data integration.

Both studies featured an extensive on-line questionnaire, containing mainly closed-questions, and a number of semi-structured 1-2 hours interviews always conducted by two researchers (one conducting the interview, one scribe). The interview guide and both original questionnaires are included for reference in the Appendix, Sections 10.1 and 10.4, respectively[10].

We conducted each study within a period of 3-4 months during September 2012 to February 2015. The study at G preceded the study at U. Figure 1 shows the relationship between the studies and this current contribution. Experience from the first study at G lead us to first conduct the interviews in case U in order to focus the questionnaire to the most relevant parts for U.

4.1. Case description G

Company G is a multinational corporation, specialised on Internet-related services and products. The company structure supports flat hierarchies and multi-project assignments for engineers. Development follows a homogeneous process with advanced tool support centred around collective code ownership and agile practices [70]. Developers at G work on multiple projects at the same time, they are organised in small teams, and develop software with several programming languages (mainly C++, Java, Python, and JavaScript). Reuse is mandated for a small set of utility functionalities; however, reusing existing code in an adequate way is considered best practice and fostered by the development style and organizational incentives. The reuse goals for the company are faster development of new features, lower maintenance costs and consistency.

Study demographics We interviewed 10 engineers and collected 39 responses to a 45-minute on-line questionnaire. The participants originated from more

[9] Please note that factors in the categories *development context* and *reuse characteristics* reflect tendencies and the state of the companies at the moment of the studies. Both companies continuously strive to improve their craft, so this table does not necessarily reflect their situation at the time of reading.

[10] Due to confidentiality of the data, we are unable to share interview transcripts and questionnaire response data.

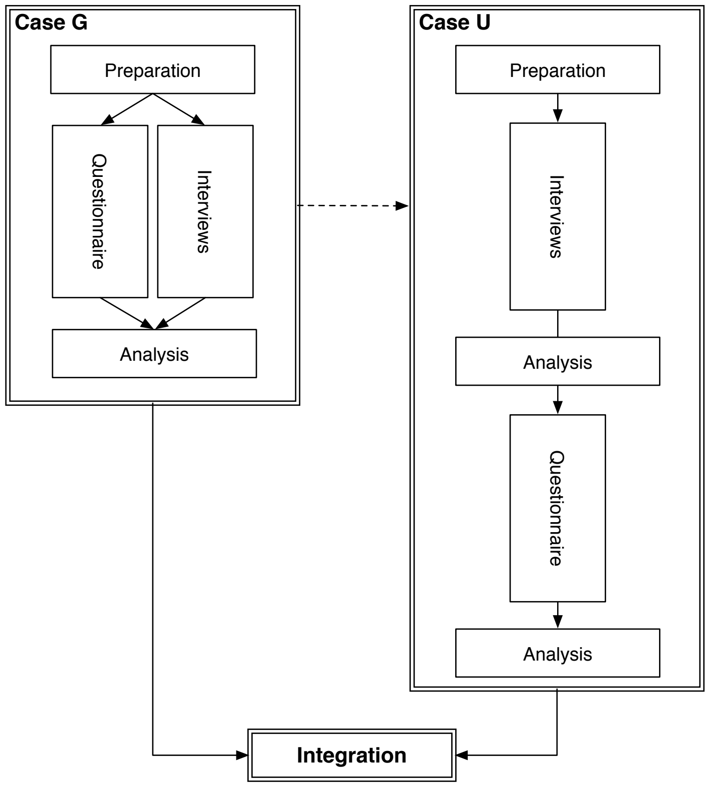

Figure 1: Study setup: case G, case U, and case integration.

than 25 different teams distributed worldwide and held varying organisational roles (developers, maintainers, managers, as well as any combination of the three roles). Their experience ranged from <1 to 20+ years in their current role (time at the company: <1 to 10+ years). By means of qualitative data analysis, we extracted the context of reuse, involving roles, responsibilities, and reuse practices, i.e. reused artefacts and reuse realizations. We collected current issues, success factors, and ideas for improvement.

Some of the results of the study at G have been published in [9].

4.2. Case description U

U is a national software producing company providing technical information services and business information products to their clients. The company was founded in the 1960s, emerged as a service provider and gradually moved to providing stand-alone software products and services. Currently, U counts around 6000 employees. The company structure is hierarchical, structured along market segments. The products have historically grown over decades and contain a broad mix of technologies. Software development is very heterogeneous across departments and teams, ranging from waterfall processes to tailored Scrum approaches. Also the level of development tool support, testing practices, and code ownership is highly diverse. As a result, products are integrated on a binary level. Developers usually work on specialized topics of a single product and tend to be responsible for the respective subsystems (Subsystem code ownership, see [70]). Reuse is mandated for an internal utility platform providing domain-independent functionality to products. The company's reuse goals are: consistent extension of the .NET framework, consistent integration of existing products, lower maintenance costs.

Study demographics

We study the current practice of reuse at U by means of an exploratory study consisting of an interview phase with 20 participants, followed by questionnaire phase with 69 respondents. We report on the state of practice of reuse, comprising success factors, challenges and ideas for improvement.

We drew interview participants from each of Us product and support development departments and all levels of the hierarchy. The participants worked at U between 15 and more than 30 years. Even though the company is mainly based in one area, the teams are distributed. For the data collection and analysis, we proceeded as for case G.

Questionnaire participants were invited by a newsletter and a post on a company news portal. Respondents came from 10 of the 13 departments. 44% worked at U for at most 10 years, 20% for 11-20 years, and 36% for more than 20 years. 15% reported their role as manager. The respondents job focus was mainly on development (78%), and architecture (13%). Respondents at U usually work within one product area and are organised in product departments over several hierarchical units. They are developing software most frequently in C# and SQL. In addition, they use Java and C++.

Parts of case U have been published in [8].

4.3. Original case study designs

In preparation of the first case study, we conducted a literature review to derive the original concepts for interview guides and questionnaire. To meet our research objective, we opted for a combination of interviews and questionnaire as they compensate each other's weaknesses: interviews provided us with a highly detailed account on reuse practices, highlighting particularities of the company context, as well as raising new ideas and concerns. However, they were expensive and time-intensive to conduct for both parties. Therefore, we chose to complement the interviews with an on-line questionnaire that was designed to capture responses from a wide range of participants.

Before rolling out the study in either of the cases, we piloted and revised the interview guide and questionnaires to remove ambiguities, increase understandability, and, in case of the questionnaires, to ensure technical performance.

Semi-structured interviews We conducted semi-structured interviews with developers, maintainers, and managers at G and U. In case G, our interview guideline was based on a pre-study with a scope similar to the questionnaire. In

U, we added company specific aspects to the interview guideline and iteratively refined it during the course of the study to accommodate new aspects impacting U's reuse practices [11].

In both cases the aim of the interviews was to obtain detailed insights into reuse application in different development teams and projects, as well as its implications regarding non-technical aspects such as company culture and interpersonal skills. We, therefore, selected participants from different departments of both companies. Each interview lasted between one and 2 hours and was conducted by two researchers, one leading the conversation with the participant while the other created the transcript and asked clarification questions.

Online questionnaire To gain a comprehensive overview of reuse at the companies, we developed an on-line questionnaire for each case[12].

For each reuse aspect, several multiple choice questions were asked. Furthermore, we invited the participants to contribute additional information in the form of free text. We asked the participants to provide their main job focus, their level of experience in their current role, the time spent working at the company and the type of project they were working on. Taking part in the questionnaire took approximately 30–40 minutes. Participation was optional.

4.4. Data collection & analysis procedures

After performing the interviews, we processed the transcripts by applying techniques from grounded theory, which support inductive content analysis. To extract the important information, we coded the transcripts[13] twice: first, we went through a phase of initial coding [16] to separate the transcripts into statements, assign them with codes, and triage them to focus on the ones relevant to reuse in practice. Based on the relevant codes, we build up emergent categories. In another, focused, round of coding [16], we pruned the categories to the most significant ones and created relationships between them. The coding process

[11] We found the following differences to case G: a strong cultural heterogeneity of the different development units, an organizational structure that skewed resources to the products, diverging development and quality assurance approaches, processes, and toolsets, different product release and rollout processes, restriction in access to code across development units, a significant amount of highly relevant legacy systems with maintenance cycles of 20+ years, heterogeneity of the employees' skill sets and experience.

[12] When conducting the first study, we noticed technical limitations of the platform and room for improvement in the resolution of our scales. After conducting the first study, we improved the questionnaire for the next by migrating to a professional platform and adjusting the resolution of the scales: As described in the methodology section (Section 5), in case G we used a simple questionnaire setup with multiple choice options for selection. We, therefore, could not measure tendencies of opinions or attitudes, as it would have been possible using, e.g., Likert scales. For this reason, we changed the scales on most questions during the preparation of case U. In addition, we incorporated results, such as success factors or challenges, from the first study. In particular, we aimed to test the success factors of case G (see Table 7, question SFB4) for their suitability as improvements in case U. For the two questionnaires, refer to the Appendix, Section 10.4. In addition, we accounted for the different company contexts and philosophies, as mentioned above.

[13] Coding means "categorising segments of data with a short name that simultaneously summarises and accounts for each piece of data" [16].

resulted in clusters of categories connected with each other, containing the relevant statements. In the case of U, we set out with a collection of potential codes obtained from the study at G. However, we adapted and pruned the collection to accommodate new information related to the new organizational context[14].

4.5. Methodological differences

Although the two investigations presented above shared goals, research methods and research questions, their implementation was not identical and, thus, necessitated methodological fine-tuning to perform the comparison. The following paragraphs highlight the differences of the cases and their consequences.

Timing. The investigation at G preceded the one at U and the comparison of the two studies was not planned at that time. When designing case U, we built partially on results of case G, including, e.g., items that were relevant as additional answer options in some questions (see, e.g., SFB3 in Table 3).

Study design. In case G, interviews and questionnaire phases were run simultaneously: The questionnaire sourced information on a large scale, whilst the interviews delivered a high level of detail to interpret the questionnaire. In case U, the interviews were conducted first. They delivered a very detailed view of the reuse practices and served as a filter to tailor the questionnaire to U's context. As a result, some questions do not have a direct counterpart in both studies (see, e.g., FAR3 in Table 3).

Scales. The scales used in the questionnaires differ: in case G questions could be answered only by multiple choice and free text. In case U, a different approach was used: The responses were given either on a four- or five-point Likert scale or as free text. For each question in U, we report the question code, followed by a tuple (L<scale level 4 or 5>, <semantics of lowest bound of scale>, <semantics of highest bound of scale>). Example: FAR1 (L4, never, always). Last, the wording of the questions is not always identical[15] (see e.g. question CHR1 in Table 3).

Diffusion. In G, 600 participants were randomly selected from a database of employees and invited via personalized email. In U, due to legal restrictions, we were unable to invite participants directly. Instead, we published a link in a company internal newsletter of the development departments and an entry in a company internal developer news portal. In addition, we invited our company contact and interview partners to spread the word.

[14]For instance, the roles and responsibilities present at the companies differed noticeably due to different hierarchy structures (flat at G vs. hierarchical at U) and development approaches. Consequently, also the notions of teams, products and projects required a mapping between the cases (the notion of *projects* in G best corresponded to *products* in the context of U) with G providing a more fluid communication and contribution structure compared to U that followed a strict hierarchical structure.

[15]Note that the questions in case U were originally expressed in German and translated for this paper.

Table 1: Characterization of the participating companies

	Company U	Company G [9]
Company settings		
Established	1960s	1990s
Overall staff*	~6000	~40000
Software staff*	~1000	>2000
Software production*	client product portfolio	online product portfolio
Application domain*	business information systems	online services
Type of software*	business	business
Organisation of development units	hierarchical, strong separation in departments	flat hierarchies, peer-driven, interconnected
Scope	national	international
Development context		
External requirements for release cycles	yes	no
Development style	heterogeneous	homogeneous
Code ownership	strong	collective
Code reviews	rarely	mandated
Development infrastructure	local	central
Source code repositories	several local ones	one central
Product assembly	binary integration	continuous source integration
Developer focus	dedicated aspects of single products	multiple aspects of multiple projects
Staff experience of sample*	high	medium
Reuse characteristics		
Reuse approach*	ad-hoc (loose*) in transition to structured (tight*)	tool-supported ad-hoc (loose*)
Current reuse scope	department	company
Global requirements engineering for reuse	limited, grass-root	limited, tool-based
Global incentives for reuse	no	yes
Co-ordination of reuse	within department	on-demand
Co-ordination overhead for reuse	significant	low
Reuse consumer**	within department	all
Reuse producer**	within department	all
Pool of available artefacts for reuse	limited	significant
Dedicated personnel for reuse	yes for basic, domain independent functionality	yes for basic, domain independent functionality
Reuse tool support	low	advanced
Accessibility of reusable artefacts	mixed	good
Formal reuse assessment	no	no
Motivation for reuse	high	high
Satisfaction with current reuse benefits	mixed	positive
Study data		
Total number participants	89	49
Participant average time in company	11-20 years	1-3 years

* adapted from [67], ** adapted from [51].

4.6. Selected material for comparison

Tables 2 and 3 contain the questions we used for the comparison between the two cases. They were selected since they had matching counter-parts in both studies. The response options are reported in Section 6 (see Tables 6 and 7). To interpret and discuss the findings (Section 7), we draw on parts of the interview data. Due to the differences reported above (and verifiable also in the Appendix, Section 10.4, and on Tables 2 and 3), we rely both on reported guidelines to qualitatively compare case studies [18] and quantitative methods to analyse and compare the surveys' questions [1]: we report our analysis methodology in the following section.

Table 2: Questions selected for comparison for RQ1

Question ID	Question text U	Question text G
Extent of code reuse (ECR)		
ECR2	What type of functionality do you reuse and which organisational unit provides it? — L5, no use, always use	What type of functionality do you reuse?
ECR3	What is the scope of functionality that you reuse? — L5, no use, always use	What is the scope of the reused artefacts?
Finding artefacts (FAR)		
FAR1	How often do you use the following ways to find reusable artefacts? — L4, never, always	What are your top-three ways to search for reusable artefacts?
Reused artefacts (RAF)		
RAF1	How often do you reuse the following artefacts? — L4, never, always	Which are the top-three types of artefacts you reuse?
RAF2	What are your sources for obtaining reusable artefacts? — L4, never, always	What are your standard sources for reusable artefacts?
Technical realization of reuse (TRR)		
TRR1	How often do you use the following techniques to realize reuse? — L4, does not apply, strongly applies	Which of the following possibilities of reuse do you employ most? Please indicate the top three.
TRR2	Which granularity do the reused entities have? — L4, does not apply, strongly applies	What granularities do the reused entities typically have?

5. Analysis Methodology

From the analysis methodology perspective, the different scales are the most relevant issue for the comparison. We address it by applying an aggregation on the Likert scales of U and a scale conversion procedure to answers from survey G to make them fully comparable with those of survey U. Subsequently, we apply regular hypothesis testing. To explain why data conversion was needed we will refer as example to the question on $RAF1$ from Table 2.

On survey U, the question $RAF1$ was formulated on the following way: *How often do you reuse the following artefacts?* Participants could select on a 4 points Likert scale the frequency of usage of that item (values: never, occasionally, regularly, always). We aggregate answers on points 3 and 4 and label them *Category H: regular or high usage*. Similarly, we aggregate answers on points 1 and 2 and label them as *Category L: irrelevant or low usage*. Table 5 in the Appendix provides the aggregations used for the other scales types.

On survey G, question $RAF1$ was formulated differently: *Which are the top-three types of artefacts you reuse?* For such type of questions participants could select up to three items, for others they did not have any limit. However, this was not enforced by the software: as a consequence, some participants could exceeded the number of available choices and selected up to four items, however most participants selected only one or two options[16]. Thus we believe it is

[16]This applies to all questions affected by this issue, i.e.: $FAR1, RAF1, TRR1$ from Table 2,

Table 3: Questions selected for comparison for RQ2

Question ID	Question text U	Question text G
Challenges, effects, and context factors of reuse (CHR)		
CHR1	How often do the following aspects negatively impact reuse in your team? — L4, never, always	In your opinion, are there difficulties disrupting the reuse process in your team?
CHR2	How often do potential disadvantages of reuse occur in your project? —L4, never, always	In your opinion, are there problems caused by reuse in your project?
CHR3	How often do the following problems occur due to lack of reuse in your project? —L4, never, always	In your opinion, are there problems caused by the absence of reuse in your project?
Success factors and benefits (SFB)		
SFB1	How often are the potential benefits of reuse realized in your project? —L4, never, always	Which benefits of reuse do you experience in your project?
SFB3	How important are the following factors to increase the benefits from reuse? — L4, unimportant, very important	In your opinion, what would be the three most important actions to make reuse beneficial in your company?
SFB4	In your opinion, which factors contribute to successful reuse projects in your company? — free text	In your opinion, what are the three most important key factors to make reuse beneficial in your company?
Reuse in everyday development practice (RED)		
RED1	How much do you agree with the following statement* on reuse on your daily work? — L4, does not apply, strongly applies	not present in G, taken from Reuse failure modes, Frakes and Fox [23].
RED4	How much do the following statements* regarding the implementation of reuse apply to your organizational unit? — L4, does not apply, strongly applies	not present in G, taken from the organizational part of reuse maturity models, e.g. [25].
Finding artefacts (FAR)		
FAR3	How much do the following statements* apply regarding the accessibility and modifiability of company internal source code? — L4, does not apply, strongly applies	Success factor derived from G [9].

* The statements are reported in the Appendix Table 7 next to the respective question code.

reasonable to apply the following conversion procedure to make the answers of survey G comparable to those of survey U:

- We compute for each item the frequency of selection assuming that such a selection is equivalent to *Category H regular or high usage*: in fact participants are asked to select the *top three used artefacts*.
- Accordingly, we assume that when the item is not selected, this is equivalent to *Category L irrelevant or low usage* : we are confident that this is a quite straightforward assumption, because enforcement on the three options was not applied and some participants in fact exceed that number of selected options.

With such conversion the data structure is identical to that of survey U, where for each item a contingency table is assigned. Therefore, we apply the

and $SFB3$ and $SFB4$ from Table 3. For the latter two, categories H and L are not about high or low frequency, but concern high or low relevance.

χ^2 test [1] on each of the resulting contingency tables to check whether the proportions in H and L differ significantly (with $\alpha = 0.05$). If the test is rejected (i.e. $pvalue \leq 0.05$) then we assign usage of item i to the category H or L, depending on which has the highest number of answers. When interpreting the findings of our analysis, we relate the statistical analysis to the findings of the interviews.

6. Study Results

Figures 5 to 6 summarize the study results for RQ1, Figures 7 to 8 for RQ2. Figure 3 explains how to read the graphical representation. For each item, the statistical significance of the answer tendency according to the χ^2 test (low or high relevance or likelihood) is represented. Rectangles denote statistical significance for the respective item, full circles denote an inconclusive answer (due to an even distribution or a non-significant skew), empty circles denote missing data. The underlying statistical data is available in the Appendix, Tables 6 and 7.

Figure 2 represents synthetically the main findings:

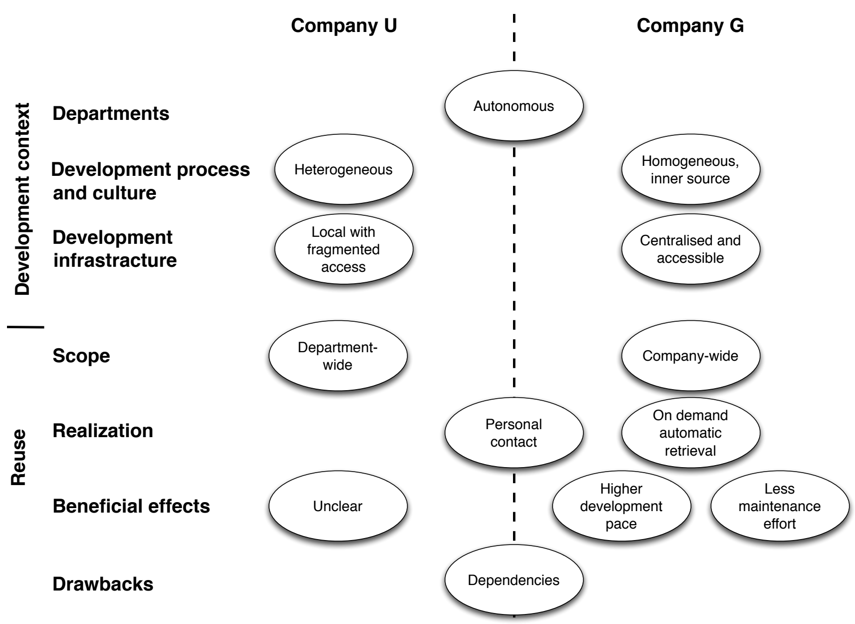

Figure 2: Summary of results, according to authors' data analysis and interpretation.

We found that reuse in both companies focused mainly on one artefact type, i.e. source code, thus not leveraging further reuse possibilities proposed by state of the art techniques. In the presence of an elaborate development tool-support and a quality-gated central repository, this infrastructure is more relevant for

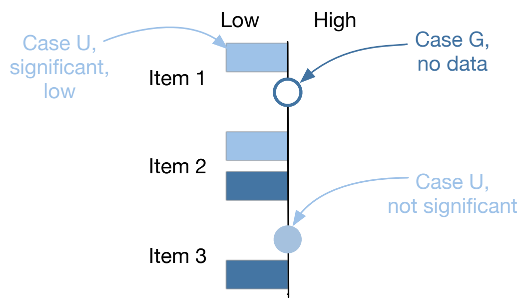

Figure 3: Example of the graphical representation of the results.

access and retrieval than personal contact (and can be seen as an instance of a successful reuse repository implementation); without this infrastructure, personal recommendations and contacts are important for pointers and access to potential solutions.

We found clear benefits (in terms of development speed and less maintenance efforts) when software reuse was effected in a homogeneous, ad-hoc, tool-supported way, and at a comparatively high level of granularity. In contrast, benefits did not materialise when reuse was effected in a heterogeneous way, and tool support was mostly absent. In both cases, these characteristics reflected the development process and culture of the company.

Finally, we can report some improvements in terms of reuse implementation due to technical advances, as well as one instance of inner source practices that enabled reuse. However, many of the organizational challenges remain and need to be addressed in order to establish reuse approaches that are beneficial to companies.

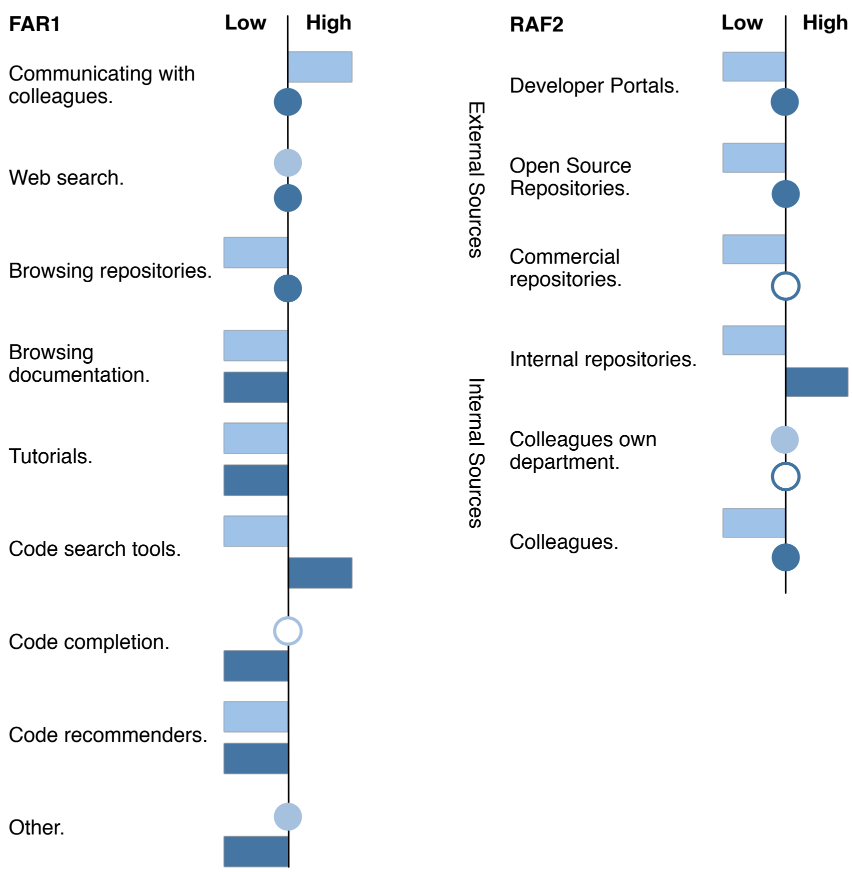

Figure 4: RQ1 - Sources of reusable entities and way of access, questions FAR1 and RAF2.

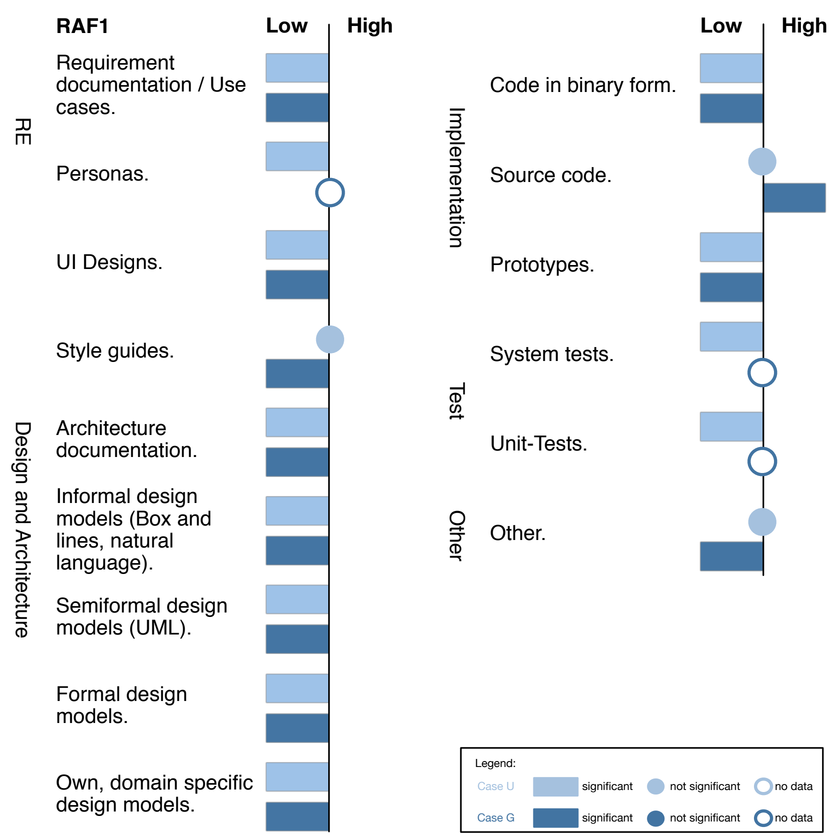

Figure 5: RQ1 - Reused entities, question RAF1.

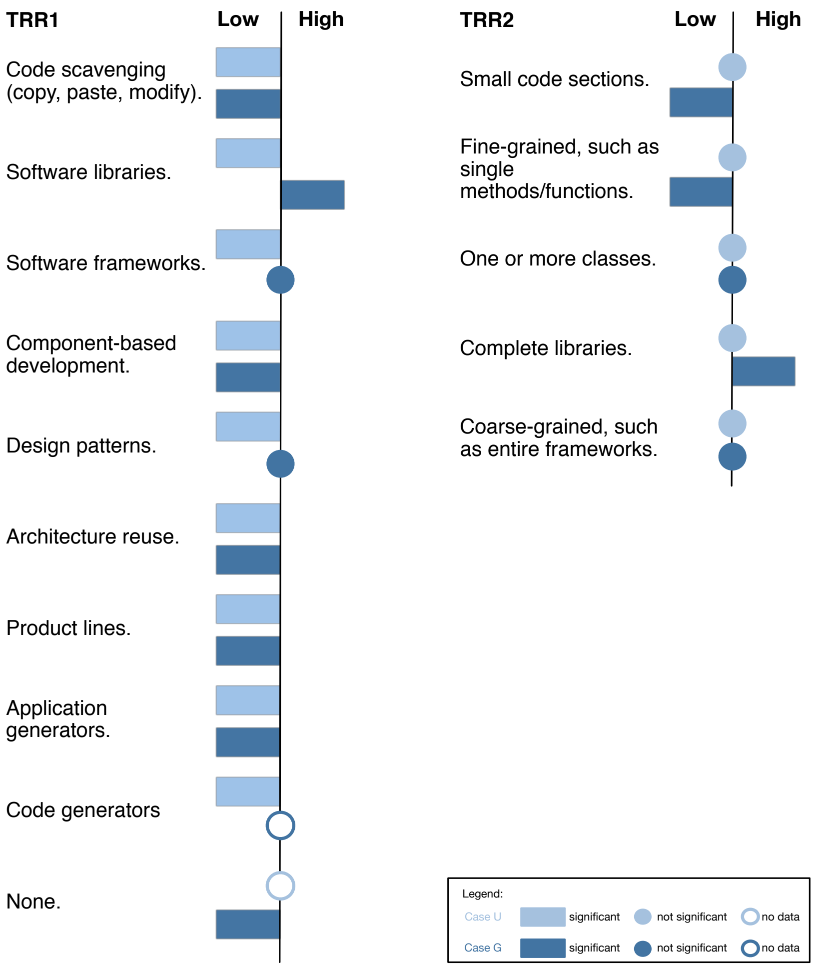

Figure 6: RQ1 - Technical realization of reuse, questions TRR1 and TRR2.

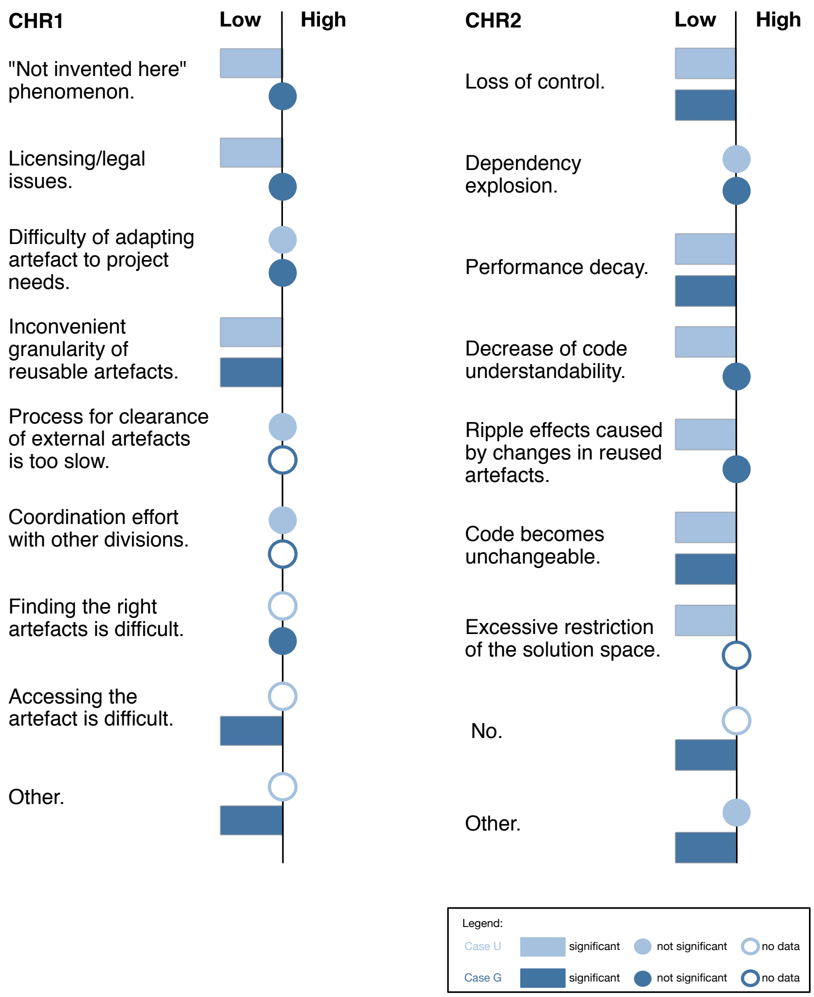

Figure 7: RQ2 - Inhibitors to reuse and issues due to reuse, questions CHR1 and CHR2.

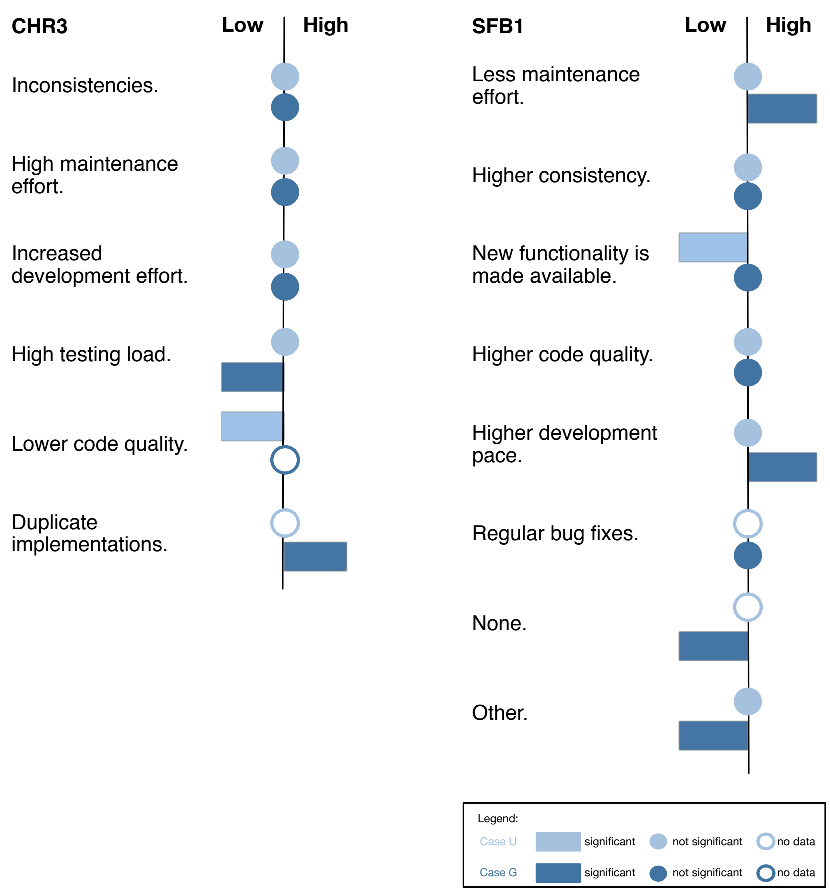

Figure 8: RQ2 - Issues of absence of reuse and benefits of reuse, questions CHR3 and SFB1.

7. Discussion and relation to state of the art

In this section, we discuss and interpret the findings of the data comparison for each research question, providing explanations thanks to the material of the interviews. In addition, we relate the results to the positions found in the literature[17].

We structure the paragraphs as follows: *Comparison agreement* (i.e. aspects in common between the two companies, according to the methodology applied to analyse the results), *comparison differences* (i.e. diverging aspects in the two companies, according to the methodology applied to analyse the results), *interview data* (if appropriate), *interpretation* of the findings, *related literature* in support or that contrast the findings. The *id's* reported with the questionnaire items refer to their label in the respective data table given in the context of each RQ.

7.1. RQ1 — Comparing reuse practices

The survey questions related to this research question are reported in Table 2. See Table 6 for the responses, as well as the comparison and statistical values.

Reused artefacts (RAF1, Figure 5)

Comparison agreement: According to question RAF1 (L4, never, always), we observe that in both cases the majority of potentially reusable artefacts are *not* reused. In particular, no artefacts from the Requirement Engineering, Design and Architecture, and Test and Deployment phases are reused.

Comparison differences: In case G, *source code* (id=25) is the only response of statistical high relevance. In contrast, in case U no artefact is reused frequently with statistical significance (however the responses indicate that about half of respondents reuse *source code* (id=25) and *style guides* (id=34)).

Interview data: The interview data reflects the findings of the questionnaire: Source code is the clear reusable entity in G and also mentioned frequently as reusable in U.

Interpretation: These findings indicate that much of the potential for reuse is unleveraged in the two companies. Literature proposes reuse on a much wider scale from models to requirements (see, e.g., [47, 31, 53]). Reasons for this might be that artefacts of earlier development stages are not available, accessible, or considered useful.

Related literature: When comparing these findings to the ones reported by [67], we see that this focus tends to be typical for companies with a loose reuse approach. In companies aiming for a more advanced reuse approach, despite lacking the prerequisites in terms of process and tool support, ad-hoc code reuse is used as best effort to create, e.g., SPLs [21]. Our findings on U supports this observation. Also a more recent on-line survey [91] confirms the presence

[17] We relate our findings to the following work: [9, 11, 12, 13, 21, 22, 25, 28, 31, 33, 34, 39, 40, 41, 43, 47, 51, 53, 54, 56, 60, 67, 69, 71, 72, 77, 79, 81, 82, 83, 84, 88, 91].

of ad-hoc code reuse. Furthermore, it reports moderate reuse of requirements, which, however, we can not confirm in our two cases.

Extent of code reuse (ECR2 and ECR3)

Comparison agreement: The results for questions ECR2 and ECR3 (L5, no use, always use) show that *domain-independent general purpose functionality* (id=9) are highly relevant with statistical significance in both companies.

Comparison differences: In case G, *product-specific functionality* (id=11) are excluded as extent of code reuse. In contrast, in case U, although not in a significant way, *domain-specific* (id=10) and *product specific functionality* are mentioned by more then half of respondents, with all types of functionality highly reused (from id=1 to id=8).

Interview data: For case G, interviews confirm a core of basic functionality, on which systems are built. For case U, interviews report of a reuse approach that is arranged along multiple layers of general purpose functionality, but also domain specific reusable entities. Considerable leeway is given to single departments with respect to their local design decisions.

Interpretation: In U, reuse of more specialised functionality might indicate an opportunity for a more structured tight reuse approach, e.g., in the form of a product line.

Related literature: In literature, reuse of utility functionality is well covered, especially in the form of Open Source libraries and frameworks [39, 33]. In commercial scenarios, product line and component-based approaches suggest a similar behavior [71]. Work on *Inner Source*, furthermore, suggests that well defined domain specific functionality can be a suitable and valuable entity for reuse [88].

Finding and accessing reusables (FAR1, Figure 4)

Comparison agreement: In both cases, the results for questions FAR1 (L4, never, always) show that a number of retrieval options are currently considered irrelevant with statistical significance: *code recommenders* (id=16), *browsing documentation* (id=17), and *tutorials* (id=18). Web search (id=12) is reported in both cases by about half respondents, yet not significant.

Comparison differences: For case U, *communicating with colleagues* is the most important (and the only significant) way to find reusable artefacts. In contrast, within G *code search tools* are in this position, while considered irrelevant in company U.

Interview data: The interviews in G confirm the high usage of the code search tools, but also stress the communication (synchronously and asynchronously) and trust among engineers. In U, interviews as well as one of the answers to FAR3 (L4, does not apply, strongly applies), indicate that code available in U can not readily be searched and accessed across departments. Therefore, retrieval and accessibility are limited by lack of technical infrastructure.

Furthermore, in both interview rounds, the concepts of *reuse producers* and *reuse consumers* emerged: in case G, due to the development process and infrastructure, all developers assume both roles, drawing on, as well as feeding into, a global pool of reusables; in U, on the contrary, the producer role is limited locally, because a dedicated group of developers takes care of the common platform, whereas the remaining developers provide reusable code either within their departments or not at all.

Interpretation: These findings highlight the importance of three enabling factors of reuse: trust between colleagues, automated support for artefact retrieval, and technical accessibility of artefacts. In the absence of infrastructure, personal communication is crucial with the disadvantages of being slow and costly. Communication is still important in the presence of infrastructure; however, it is more concerned with the goal of clarification. The key access point shifts to the tool support asynchronously available to each developer.

Despite the reliance on tools, reactive support systems are not yet widely used to improve reuse.

Related literature: The mentioned enabling factors are, amongst others, considered preconditions for the so-called *Inner Source* approach [88], as well as the development of SPLs [71]. Technical support, such as *code recommenders* (id=16) are one example of research that should contribute to these three aspects [77] and might, in principle, target the right goals; however, these tools are not yet used widely (only one respondent per case declared to use them). This could be an indication that from a research perspective, more work needs to be done to adapt research work to the reality of practitioners, especially in terms of usability and scalability [69, 72].

Sources of artefacts (RAF2, Figure 4)

On the item level, we observe no agreement for question RAF2 (L4, never, always). However, there is a tendency in both cases towards company-internal artefact sources.

Comparison differences: The only statistically significant source in case G is that of *internal repositories* (id=37). In U, all but one source (*colleagues from own department*, id=39), are of low usage.

Interview data: In G, interviews further stress the importance of the central repository for reuse success.

In U, interview data indicates that, in the absence of technical access, one of the main sources of reusable artefacts might be the code that developers have previously written themselves. Also, the fact that developers stay in their department for many years and acquire specialist knowledge might impact their willingness to rely on and trust the work of others.

Participants from both companies mentioned the business and legal risks of reuse across company borders. Licensing was mentioned as significant inhibitor to reusing Open Source code, and the reliability and robustness of an external commercial software provider as high risk to reusing proprietary reusables.

Interpretation: In the case of G, the internal infrastructure and repository seem to provide adequate support for reuse across the company. In U, this kind of reuse is hampered by a combination of specialist knowledge and organizational and technical separation.

In both cases, internal sources preferred over external ones due to the potential risks entailed to the latter.

Related literature: Whilst the web is considered a huge repository in literature [39], this is scarcely reflected in the context of both companies: legal restrictions, security policies, and domain specificity prevent a significant exchange of reusable artefacts across company boundaries. Access to reusable entities is, thus, mediated by personal networks of developers. In addition, we can confirm that the potential risks imposed by dependencies on third parties [11] impact the decision of companies with respect to third-party reuse.

Technical reuse realization (TRR1 and TRR2, Figure 6)

Comparison agreement: For question TRR1 (L4, does not apply, strongly applies), in both cases *code scavenging* (id=42), *component-based development* (id=45), *architecture reuse* (id=47), *product lines* (id=48), and *application generators* (id=49) are *not* considered relevant.

Comparison differences: In case U, all offered ways to technically realize reuse are marked as not relevant with statistical significance. In case G, realising reuse by means of *software libraries* (id=43) is of statistically significant high relevance.

Interview data: In case G, the interview data is largely consistent with the survey, although some instances of code scavenging were mentioned. For U, in contrast to the survey, the interview data suggests application of code scavenging (id=42), as well as some libraries (id=43) and framework-based reuse (id=44) (see also question TRR2).

IP and RL: see TRR2

Comparison agreement: Question TRR2 (L4, never, always) addresses the granularity of the reused artefacts. There is no common findings between the cases.

Comparison differences: Complete libraries (id = 55) are of high relevance in case G. In contrast, reuse in case U takes place on all levels of granularity, however, none of the corresponding answer is statistically more relevant than the other.

Interview data: -

Interpretation: Generally, G realizes reuse homogeneously on a higher abstraction level than U, where reuse is effected in a very heterogeneous way. Furthermore, the selection for U indicates a conflict to the results of the interviews and the responses of TRR1: reusing small code entities (snippets and single classes) suggests the presence of code scavenging. However, it is possible that the respondents do not have a strong preference for any of the reuse methods or do generally not reuse code as much (see RAF1).

Related literature: The realisation of reuse reported in case U aligns with the findings of Fichman and Kemerer [22]: in a study with 15 software developing teams, they found that reuse was prevalent on an informal, local, scope but neglected on an inter-project, systematic, level. The authors identified as root cause to the failure an incentive conflict with respect to team priorities such as completing a project on time and on budget. The case of U, furthermore, also confirms findings by Dubinski et al. [21]: in the absence of supporting technical infrastructure and processes, developers resort to primitive reuse mechanisms to model complex reuse approaches.

Work on Inner Source highlights the pivotal role of technical infrastructure for reuse, especially when effected in a loose and ad-hoc way [88]. Case G confirms these findings: reuse is conducted in an informal and ad-hoc way. However, supported by an advanced infrastructure (that required significant resources and management commitment, confirming findings of, e.g., [67, 54]), a viable company-wide development process, and suitable organizational incentives, reuse takes place in a large scale and across project boundaries.

Summary of RQ1

Case G exhibits a homogeneous approach to reuse, progressing in an inner source style that allows for ad-hoc and opportunity driven development. Case U exhibits a very heterogeneous approach in which many different styles co-exist. From this perspective, both companies reflect their development processes in the way they effect reuse.

Both cases focus on code reuse (G in the form of libraries, U with no predominant granularity). This entails that the large potential present in additional development artefacts remains unleveraged. The frequent reuse of general purpose functionality is confirmed.

Automated and tool-supported access to and retrieval of reusables is considered as key factor for effective reuse. In G, the impact of the infrastructure clearly shows in the finding and retrieving actions of reuse. In U, their absence restricts reuse to a local scope.

In both cases, the sources of reusables are mainly within the companies. The reported reasons for this were business risks in terms of security or robustness of the provider, as well as licensing aspects.

7.2. RQ2 — Comparing effects and context factors

The questions relevant for this RQ are reported in Table 3. The responses, the comparison, and the statistical values are reported in Table 7.

Inhibitors (CHR1, Figure 7)

Comparison agreement: Question CHR1 (L4, never, always) reports disrupting factors to the reuse process. There is no statistically significant inhibitor of high relevance. Respondents in both cases agree that *Inconvenient granularity of reusable artefact* (id=60) does impact them.

Comparison differences: The *"not invented here" phenomenon* (id=57) is reported little and is rated as irrelevant in case U. Questions FAR3 and RED1[18] (both: L4, does not apply, strongly applies) indicate that difficulties in retrieving and accessing artefacts significantly disrupt the reuse process in U.

Interview data: At G, participants report the perceived ease of creating needed functionality from scratch over understanding existing solutions as inhibitor. Also, they mention the considerable (cognitive) effort involved in selecting suitable candidates from a plethora of potential results.

[18] Due to differences in the questionnaires, some of the items present in CHR1 for case G are contained in FAR3 and RED1 in case U. Therefore, we include them in this paragraph.

In U, the interviews disclose a further inhibitors to reuse: organizational and technical separation of departments, as well as the absence of a thorough global reuse strategy that takes into account different life cycle characteristics of system parts. This leads to the creation of unsuitable artefacts[19].

Interpretation and related literature: At G, the reported negative connotation with the required cognitive effort for selection and adaptation could be seen as a more subtle instance of the "not invented here" (NIH) syndrome [28]. At U, the difficulties in access across project boundaries are one of the main inhibitors to reuse, aligning with the observations of [21]. The lack of availability of reusables provided by other parties could, potentially, explain the perceived absence of NIH.

Difficulties due to reuse (CHR2, Figure 7)

Comparison agreement: Question CHR2 (L4, never, always) reports on difficulties encountered *due to* reuse. None of the presented options was of high frequency with statistical significance, nor did the respondents highlight significant other issues. The only difficulty in common in both cases, but not in a statistically significant way, is that of *dependency explosion* (id=67).

Comparison differences: In case G around one third of the respondents reported *ripple effects caused by changes in reused artefacts* (id=70) and a *decrease in code understandability* (id=69) as negative consequences of reuse (no statistical significance).

Interview data: In G, the complexity of the technical dependency structure was considered a challenge, however mitigated by the infrastructure and offset by the experienced gains. In U, participants mentioned a variety of harmful dependencies that they linked to negative aspects of reuse: technical ones (e.g. unstable interfaces, versioning dependencies), as well as organizational ones (e.g. diverging release cycles, delays due to coordination efforts).

Interpretation: Difficulties around reuse arise on several levels ranging from technical to organizational. In the context of organizational heterogeneity, non-technical dependencies impose a variety of challenges that inhibit beneficial reuse.

Related literature: Previous work suggests that additional rework due to reuse might not be a significant overhead [83]. Also, organizational heterogeneity is known as a challenge in the context of establishing development practices exceeding the scope of separate organizational entities [88]. Our findings support both of these suggestions.

Difficulties due to lack of reuse (CHR3, Figure 8)

Comparison agreement: Question CHR3 (L4, never, always) reports the negative consequences due to the *lack of* reuse. Both cases report regular occurrences[20] of *inconsistencies* (id=75), *high maintenance effort* (id=76), and

[19] Unsuitable, e.g., on the conceptual level by incompatible abstractions and decompositions that increase the effort of reusing artefacts, but also on the business level, causing significant investments in low-return artefacts and eroding management trust.

[20] Around 50% of the participants report these occurrences; however, they are not statisti-

increased development effort (id=77).

Comparison differences: The only factor of significant relevance in case G is the occurrence of *duplicate implementations* (id=80). In U, no factor is of statistical significance.

Interview data: The item *duplicate implementations* is missing in case U; however, the interviews indicate multiple instances of this issue. In both cases, unwanted redundancies were not yet tracked systematically (or tracked at all).

Interpretation: Both companies to some degree incur the typical drawbacks associated with lack of reuse. However, the only aspect of significance is the one of duplicate implementations (which, arguably, entails some of the other drawbacks).

Related literature: Research has been addressing discovering and tracking redundancies in the form of code clones [79, 60, 41] and re-implementations [56, 12, 13]. At this point, several industrial tools exist that support structural (as opposed to semantic) detection approaches on an industrially viable scale [34]. Therefore, this issue might be mitigated within a reasonable timeframe.

Benefits (SBF1, Figure 8)

Comparison agreement: For question SBF1 (L4, never, always), there is no statistically significant agreement.

Comparison differences: Only case G reports statistically significant high occurrences of the benefits *higher development pace* (id=104) and *less maintenance effort* (id=100).

Interview data: Participants in case G consider their reuse realisation as beneficial, i.e. fulfilling the goals behind their reuse approach. In case U, participants indicate that the aimed-for benefits of the given reuse strategy have not yet materialized as expected.

Related literature & Interpretation: Generally, improved code quality is one of the benefits associated with reuse [51, 65]. We could not directly confirm this in our studies: In case G, participants already considered the produced artefacts as high quality and, thus, would not attribute this characteristic to reuse in particular. Instead, they considered their code quality as one of the main enablers of reuse. On the other hand, for G, we can confirm the gain in development speed [83, 54] and the decrease in maintenance effort [71].

In case U, the heterogeneity of development did not allow a clear assessment of the quality of the reused code. With respect to development speed and maintenance effort, our data provided no clear results.

Success factors (SBF4)

Comparison agreement: Question SBF4 was multiple choice in case G and free text in case U. Therefore, we can not provide a comparison based on our statistic test, but instead report the findings for each case separately.

Comparison differences: Respondents in case G report two statistically significant relevant success factors: *the high quality of artefacts* (id=132) and *sup-*

cally significant.

porting infrastructure and tools (id=134)[21]. More than 50% of the respondents also mentioned *adequate abstractions* (id=129) as success factor. The remaining options (*direct communication culture (id=130), suitable incentives (id=131), well-defined process for reuse (id=133), stricter rules for dependency management (id=135), homogeneous development culture (id=136)*) for success factors were reported as significant and low relevance[22].

In the case U, the free text responses reflected a negative tendency. However, we added the success factors from case G as potential improvements for case U (question SFB3 — L4, unimportant, important). All but two of these factors were reported as significant and high relevance by the respondents of case U.

Interview data: The interviews in G indicate that the accessibility of artefacts as well as the "safety net" and immediate feedback provided by an extensive tool support increase the inclination to build on existing solutions. In addition, automation is seen as the only feasible way to draw reusable artefacts from a large code base. Last, the benefits were tangible to developers. This further motivated them to reuse during development. In U, the interview participants stress the negative effects of the heterogeneous development culture. As a result, they saw the need for one or more reuse champions that lobby a homogeneous development and reuse strategy across departments.

Interpretation: We consider this a noteworthy finding, as it indicates that developers are more willing to trust existing artefacts if they can thoroughly inspect and validate them, and they have faith in the process (and quality assurance) by which those artefacts were created. In addition, the potential improvements at U indicate the need for changes in the reuse and development processes.

Related literature: Parts of these findings coincide with literature: Kruger [47] considers abstraction the "essential feature of any reuse technique" and stresses the importance of an "integration framework" for reuse. Joss [40] reports management support, education of engineers, suitable incentives, tool support[23] as relevant success factors for introducing structured reuse. Several studies (e.g. [67, 81]) suggest that, besides conceptual difficulties, the adoption of a reuse approach is significantly driven or inhibited by the organisational commitment towards the adoption process. Whilst in case G the most significant reported

[21] Since the difference between these two answers is only one response, we consider the second item also as highly relevant.

[22] Note that the respective question in case G asks for the top 3 success factors, but this was not enforced by the software. As a result, most participants selected only one or two options, whilst others exceeded the number and selected up to four items. Since for this question none of the other options approaches even moderate frequency, the ranking seems comparable to the frequencies in U.

[23] It might be noteworthy to consider the differences in "tool support" w.r.t. the drastic advances of the technologies and paradigms used for programming and reuse. In a component repository, as e.g. proposed by the REBOOT approach in the 1990s (see [43, 82]), this relates to a basic protocol for repository and configuration management, whilst in today's setting, this refers to advanced code search and recommendation systems, central build and testing infrastructure etc. (see e.g. [9])

success factors were of technical nature (indirectly enabled by the organization), the results also align with [84], reporting the belief in benefits as motivator and success factor for reuse. In case U, the most important improvements included the mentioned organisational aspects. In terms of the definitions of reuse maturity and "good" reuse stated in the literature (see [25] for an exemplary reuse maturity model), case G challenges conventional academic assumptions and supports reports of successful reuse through Inner Source (see [87]): Whilst reuse is ingrained in the organisation, thorough planning and formal reuse assessment are not. However, due to their trust in their code and engineering quality, as well as their elaborate development support infrastructure, developers at G implemented a beneficial version of opportunistic ad-hoc reuse that matches exactly the company goals.

Summary of RQ2

For this research question, we found no statistically significant inhibitors or negative effects. However, technical incompatibilities and organizational heterogeneity as well as dependencies were identified as factors challenging beneficial reuse. Furthermore, participants in G report the challenge of identifying the right reusables from a large number of candidates and adapting them to meet current needs.

In terms of difficulties encountered due to lack of reuse, both cases agree on occurrences of duplicate implementations.

In the homogeneous and tool-supported context of G, a significant increase in development speed and a significant decrease of maintenance efforts are reported as benefits of reuse. In U, these effects have not been observed.

In G, the quality of the reusables and the supporting infrastructure are seen as clear success factors. Participants in U considered a tight network of personal connections across departments and reuse champions as crucial preconditions to successful reuse adoption.

8. Threats to Validity

When studying development practices in specific companies, it is very challenging to generalise results even to similar companies. Our study is not immune to this threat to the external validity, however we provided a detailed contextualisation of the two companies, which should serve as a framework for further studies to compare findings with ours. In the long run, such a contextualisation framework should help to provide a sensible generalisability of results. Regarding the internal threats, we identify three main issues:

Self-selection bias: The participation in our studies was optional and might have led to a biased selection of participants: in case G, most participants of interviews and questionnaire displayed a favorable attitude towards reuse, so it is possible that only engineers considering reuse as beneficial volunteered to take part. In case U, on the contrary, a significant number of participants vented their disappointment with the current state of reuse. In addition, the departments that, during the interviews, appeared least concerned with improvements did not participate at all in the questionnaire. To mitigate this threat, we attempted to select our interview participants in both cases from as many departments and as many different positions on reuse as possible.

Selection of participants: For each study, the participants of the interviews were sampled by convenience through personal contact in the respective company. This might have introduced a bias in the results. To mitigate this threat, we sampled the interview participants from different organizational units and different roles. We could not control the selection of the questionnaire participants. In case G, the respondents matched the expected distribution of departments. In case U, several departments did not contribute.

Heterogeneity of sample: The sample of the participants of the two studies differs in terms of the time spent at the respective company. This could potentially influence the knowledge of reuse practices and, thus, bias the responses. This sampling difference follows from two characteristics in which the studied companies differ: age of the company (G $<$ 20 years, U $>$ 50 years) and turnover of staff on projects (at G, developers moved between projects frequently, whilst at U staying with the same products for more than 10 years was not uncommon). However, we believe that this does not affect our findings: first, at G, reuse practices are homogeneous and streamlined with the development process. Newcomers are trained extensively to adhere to the given development practices (including the reuse practices). Therefore, we are confident that our participants at G were fully familiar with and aware of the reuse practices at their company.

Limitations of research methods: Our interpretation of the answers from the questionnaire at G rely on the assumption that non-selected items in multiple choice questions are considered equivalent to rate those options as irrelevant or of low usage. This assumption might not be completely true for questions in which participants were asked to select up to three items. However, the exact number of selected items was not software-enforced. As a result, respondents typically selected between one and three items and sometimes exceeded the number and selected four items or more. At U, the questionnaire design prevented this complication.

Subjectivity in responses: When designing the questionnaires, we aimed to capture the responses by means of precise measures. However, as we frequently asked participants about their experiences (e.g. on the perceived maintenance effort without reuse) and their agreement, we could not assume that they were equipped with the necessary tooling to provide objective measurements as responses. As a result, we resorted to more abstract, yet subjective, answer options (e.g. high, medium, low). Whilst these can only provide a tendency, this is a typical procedure for this kind of study (see, e.g., [54]) and, nevertheless, captures the perceived benefits/drawbacks of reuse in the respective cases. We, therefore, consider this aspect a minor threat to the validity of our conclusions.

Study design: As mentioned in Section 4, the study setup differs between the two cases: in case G, interviews were conducted during the same phase as the questionnaire. In case U, the interviews preceded the questionnaire. In this way, we could focus the content of the questionnaire and reduce it in size. We consider the change in design a minor threat to validity of our conclusions, as we retained the questions needed for the comparison.

Timeliness of second study: Despite our best efforts, the studies could not be conducted in a more narrow timeframe. However, we consider the impact of this delay as minor for the following reasons: The company studied in the first case is still developing in the same way (Inner Source), focusing on code reuse and trying to compensate drawbacks of the approach by more elaborate tool support. Since the company is stable and continuous in their approach, the data is still accurate. Therefore, we assume that the comparison is valid from this perspective.

9. Conclusion

In this study, we reported on the comparison of two in-depth case studies on software reuse in industrial practice, integrating data from 138 professional developers of two companies, G and U.

The comparison highlights that reuse in practice occurs pragmatically in different flavours, however, mostly limited to source code. This largely confirms the findings of previous research. The technological potential offered by state of the art infrastructure has been partially embraced, rendering operational once infeasible approaches, such as repositories as source for reusable entities.

Successful reuse is tightly coupled to the company goals and ingrained in the development culture, also in terms of management and tool support. Perceived business success of reuse seems more determined by coherence between culture and approach than by the structuredness of the adopted approach.

In the homogeneous and coherent reuse setting, clear benefits for development and maintenance are reported. These benefits did not materialize in the heterogeneous setting.

Establishing any kind of systematic reuse in heterogeneous company and development contexts poses significant challenges and requires structured decision support. Further work is needed to support companies in heterogeneous contexts to identify and install the required preconditions of suitable reuse approaches.

Acknowledgements

The authors would like to thank all participants and supporters from the companies for their support. Furthermore, thanks go to Maximilian Junker and the anonymous reviewers for their valuable feedback on earlier versions of this work. Parts of this work were funded by the Federal Ministry of Education and Research, Germany (BMBF).

[1] A. Agresti. *An introduction to categorical data analysis.* Wiley, New York, 1996.

[2] C. Ayala, Ø. Hauge, R. Conradi, X. Franch, and J. Li. Selection of third party software in off-the-shelf-based software development—an interview study with industrial practitioners. *Journal of Systems and Software*, 84(4):620–637, 2011.

[3] R. D. Banker, R. J. Kauffman, and D. Zweig. Repository evaluation of software reuse. *Software Engineering, IEEE Transactions on*, 19(4):379–389, 1993.

[4] V. Basili, G. Caldiera, and H. Rombach. The experience factory. *Encyclopedia of software engineering*, 1994.

[5] V. R. Basili, G. Caldiera, and H. D. Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley, 1994.

[6] L. Bass, P. Clements, S. Cohen, L. Northrop, and J. Withey. Product line practice workshop report. Technical report, Software Engineering Institute, 1997.

[7] D. Batory and S. O'Malley. The design and implementation of hierarchical software systems with reusable components. *ACM Trans. Softw. Eng. Methodol.*, 1(4):355–398, Oct. 1992.

[8] V. Bauer. Challenges of structured reuse adoption — Lessons learned. In *Profes 2015*, 2015.

[9] V. Bauer, J. Eckhardt, B. Hauptmann, and M. Klimek. An Exploratory Study on Reuse at Google. In *SER&IP's 2014*.

[10] V. Bauer and L. Heinemann. Understanding API Usage to Support Informed Decision Making in Software Maintenance. In *CSMR 2012*, 2012.

[11] V. Bauer, L. Heinemann, and F. Deissenboeck. A Structured Approach to Assess Third-Party Library Usage. In *ICSM'12*, 2012.

[12] V. Bauer, T. Voelke, and E. Juergens. A novel approach to detect unintentional re-implementations. In *ICSME'14*, 2014.

[13] V. Bauer, T. Völke, and S. Eder. Combining clone detection and latent semantic indexing to detect reimplementations. In *under review at SANER 2016*, 2016.

[14] M. Bruch, M. Monperrus, and M. Mezini. Learning from examples to improve code completion systems. In *ESEC/SIGSOFT FSE*, 2009.

[15] J. Businge, A. Serebrenik, and M. G. J. van den Brand. Eclipse API Usage: The Good and The Bad. In *Sixth International Workshop on Software Quality and Maintainability*, 2012.

[16] K. Charmaz. *Constructing grounded theory: A practical guide through qualitative analysis*. Pine Forge Press, 2006.

[17] J. R. Cordy. Comprehending Reality - Practical Barriers to Industrial Adoption of Software Maintenance Automation. In *Proceedings of the IEEE 11th International Workshop on Program Comprehension*, 2003.

[18] D. Cruzes, T. Dybå, P. Runeson, and M. Höst. Case studies synthesis: a thematic, cross-case, and narrative synthesis worked example. *Empirical Software Engineering*, pages 1–32, 2014.

[19] R. P. Díaz. Status report: Software reusability. *IEEE Software*, 10(3):61–66, 1993.

[20] J. Dinkelacker, P. Garg, D. Nelson, and R. Miller. Progressive Open Source. In *ICSE' 02*, 2002.

[21] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki. An exploratory study of cloning in industrial software product lines. In *CSMR 2013*, 2013.

[22] R. G. Fichman and C. F. Kemerer. Incentive compatibility and systematic software reuse. *The Journal of Systems and Software*, 57:45–60, 2001.

[23] W. B. Frakes and C. J. Fox. Quality improvement using a software reuse failure modes model. *Software Engineering, IEEE Transactions on*, 22(4):274–279, 1996.

[24] W. B. Frakes and K. Kang. Software reuse research: Status and future. In *IEEE Transactions on Software Engineering*, volume 31, pages 529–536, 2005.

[25] V. C. Garcia, D. Lucrédio, A. Alvaro, E. S. De Almeida, R. P. de Mattos Fortes, and S. R. de Lemos Meira. Towards a maturity model for a reuse incremental adoption. In *SBCARS*, pages 61–74. Citeseer, 2007.

[26] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch or why it's hard to build systems out of existing parts. In *Proceedings of the 17th International Conference on Software Engineering*, ICSE '95, pages 179–185, New York, NY, USA, 1995. ACM.

[27] R. Goldman and R. P. Gabriel. *Innovation happens elsewhere: Open source as business strategy.* Morgan Kaufmann, 2005.

[28] J. Greenfield, K. Short, S. Cook, and S. Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools.* Wiley, 2004.

[29] V. K. Gurbani, A. Garvert, and J. D. Herbsleb. A case study of a corporate open source development model. In *Proceedings of the 28th international conference on Software engineering*, pages 472–481. ACM, 2006.

[30] S. Hallsteinsen and M. Paci. *Experiences in software evolution and reuse: twelve real world projects*, volume 1. Springer Science & Business Media, 1997.

[31] L. Heinemann. Facilitating reuse in model-based development with context-dependent model element recommendations. In *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on*, pages 16–20, June 2012.

[32] L. Heinemann, V. Bauer, M. Herrmannsdoerfer, and B. Hummel. Identifier-based context-dependent api method recommendation. In *CSMR'12*, 2012.

[33] L. Heinemann, F. Deissenboeck, M. Gleirscher, B. Hummel, and M. Irlbeck. On the Extent and Nature of Software Reuse in Open Source Java Projects. In *ICSR'11*, 2011.

[34] L. Heinemann, B. Hummel, and D. Steidl. Teamscale: Software quality control in real-time. In *Proceedings of the 36th ACM/IEEE International Conference on Software Engineering (ICSE'14)*, 2014.

[35] S. Henninger. An evolutionary approach to constructing effective software reuse repositories. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 6(2):111–140, 1997.

[36] R. Holmes and R. J. Walker. Systematizing Pragmatic Reuse Tasks. *ACM Trans. Softw. Eng. Methodol.*, 2012.

[37] J. Hopkins. Component primer. *Communications of the ACM*, 43(10):27–30, 2000.

[38] D. Hristov, O. Hummel, M. Huq, and W. Janjic. Structuring software reusability metrics for component-based software development. In *Proceedings of Int. Conference on Software Engineering Advances (ICSEA)*, 2012.

[39] O. Hummel and C. Atkinson. Using the web as a reuse repository. In *Reuse of Off-the-Shelf Components*, pages 298–311. Springer, 2006.

[40] R. Joos. Software reuse at motorola. In *IEEE Software*, volume 11, 1994.

[41] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In *Proceedings of the 31st International Conference on Software Engineering*, pages 485–495. IEEE Computer Society, 2009.

[42] C. J. Kapser and M. W. Godfrey. "Cloning considered harmful" considered harmful: patterns of cloning in software. *Empirical Software Engineering*, 2008.

[43] E.-A. Karlsson. *Software reuse: a holistic approach.* John Wiley & Sons, Inc., 1995.

[44] I. Keivanloo, J. Rilling, and Y. Zou. Spotting working code examples. In *Proceedings of the 36th International Conference on Software Engineering*, pages 664–675. ACM, 2014.

[45] Y. Kim and E. A. Stohr. Software reuse: Issues and research directions. In *Twenty-Fifth Hawaii International Conference on System Sciences*, 1992.

[46] H. Koziolek, T. Goldschmidt, T. de Gooijer, D. Domis, S. Sehestedt, T. Gamer, and M. Aleksy. Assessing software product line potential: An exploratory industrial case study. *Empirical Software Engineering*, 2015.

[47] C. Krueger. Software reuse. *ACM Computing Surveys*, 24(2):131–183, 1992.

[48] C. W. Krueger. Software product line reuse in practice. In *Application-Specific Systems and Software Engineering Technology, 2000. Proceedings. 3rd IEEE Symposium on*, pages 117–118. IEEE, 2000.

[49] R. Land, D. Sundmark, F. Lueders, I. Krasteva, and A. Causevic. Reuse with software components - a survey of industral state of practice. In *ICSR'09*, 2009.

[50] B. M. Lange and T. G. Moher. Some strategies of reuse in an object-oriented programming environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '89, pages 69–73, New York, NY, USA, 1989. ACM.

[51] W. Lim. Effects of reuse on quality, productivity, and economics. *IEEE Software*, 11(5):23–30, 2002.

[52] J. Llorens, J. Fuentes, R. Prieto-Diaz, and H. Astudillo. Incremental Software Reuse. *ICSR*, 2006.

[53] M. Luckey, A. Baumann, D. Méndez, and S. Wagner. Reusing security requirements using an extended quality model. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, pages 1–7. ACM, 2010.

[54] D. Lucrédio, K. dos Santos Brito, A. Alvaro, V. C. Garcia, E. S. de Almeida, R. P. de Mattos Fortes, and S. L. Meira. Software reuse: The brazilian industry scenario. *Journal of Systems and Software*, 81(6):996 – 1013, 2008. Agile Product Line Engineering.

[55] A. Lynex and P. J. Layzell. Organisational considerations for software reuse. *Annals of Software Engineering*, 5:105–124, 1998.

[56] A. Marcus and J. I. Maletic. Identification of high-level concept clones in source code. In *ASE'01*.

[57] Marcus A. Rothenberger and Kevin J. Dooley and Uday R. Kulkarni and Nader Nada. Strategies for Software Reuse: A Principal Component Analysis of Reuse Practices. In *IEEE Transactions on Software Engineering*, 2003.

[58] M. McILROY. Mass produced soptware components. In *NATO Software Engineering Conference Report*, 1968.

[59] C. McMillan, N. Hariri, D. Poshyvanyk, J. Cleland-Huang, and B. Mobasher. Recommending source code for use in rapid software prototypes. In *ICSE*, 2012.

[60] T. Mende, F. Beckwermert, R. Koschke, and G. Meier. Supporting the grow-and-prune model in software product lines evolution using clone detection. In *CSMR 2008*, 2008.

[61] Y. Mileva, V. Dallmeier, and A. Zeller. Mining API Popularity. In *Testing - Practice and Research Techniques*, volume 6303 of *Lecture Notes in Computer Science*, pages 173–180. Springer, 2010.

[62] A. Mili, R. Mili, and R. T. Mittermeir. A survey of software reuse libraries. *Annals of Software Engineering*, 5:349–414, 1998.

[63] H. Mili, F. Mili, and A. Mili. Reusing software: Issues and research directions. In *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, volume 21, 1995.

[64] R. T. Mittermeir and W. Rossak. Software bases and software archives: alternatives to support software reuse. In *Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow*, pages 21–28. IEEE Computer Society Press, 1987.

[65] P. Mohagheghi and R. Conradi. Quality, productivity and economic benefits of software reuse: a review of industrial studies. *Empirical Software Engineering*, (12):471–516, 2007.

[66] P. Mohagheghi and R. Conradi. An empirical investigation of software reuse benefits in a large telecom product. *ACM Trans. Softw. Eng. Methodol.*, 17(3):13:1–13:31, June 2008.

[67] M. Morisio, M. Ezran, and C. Tully. Success and failure factors in software reuse. *Software Engineering, IEEE Transactions on*, 28(4):340–357, 2002.

[68] M. Morisio, C. B. Seaman, V. R. Basili, A. T. Parra, S. E. Kraft, and S. E. Condon. Cots-based software development: Processes and open issues. *Journal of Systems and Software*, 61(3):189–199, 2002.

[69] E. Murphy-Hill and G. C. Murphy. Recommendation Delivery. In *Recommendation Systems in Software Engineering*, chapter 9, pages 223–242. Springer, 2014.

[70] I. Nordberg, M.E. Managing code ownership. *Software, IEEE*, 20(2):26–33, Mar 2003.

[71] K. Pohl, G. Böckle, and F. J. van der Linden. *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[72] S. Proksch, V. Bauer, and G. C. Murphy. How to build a recommendation system for software engineering. In *Software Engineering*, pages 1–42. Springer, 2015.

[73] S. Raemaekers, A. van Deursen, and J. Visser. An analysis of dependence on third-party libraries in open source and proprietary systems. In *CSMR'12*, 2012.

[74] S. Raemaekers, A. van Deursen, and J. Visser. Measuring software library stability through historical version analysis. In *SQM*, 2012.

[75] H. Rehesaar. Capability assessment for introducing component reuse. In *ICSR'11 - Top Productivity through Software Reuse*, pages 87–101. Springer, 2011.

[76] D. C. Rine. Success factors for software reuse that are applicable across domains and businesses. In *Proceedings of the 1997 ACM symposium on Applied computing*, pages 182–186. ACM, 1997.

[77] M. Robillard, R. Walker, and T. Zimmermann. Recommendation systems for software engineering. *Software, IEEE*, 27(4):80–86, 2010.

[78] M. B. Rosson and J. M. Carroll. The reuse of uses in smalltalk programming. *ACM Transactions on Computer-Human Interaction*, 3:219–253, 1996.

[79] C. K. Roy and J. R. Cordy. A survey on software clone detection research. *SCHOOL OF COMPUTING TR 2007-541, QUEEN'S UNIVERSITY*, 115, 2007.

[80] M. Shaw. Architectural issues in software reuse: It's not just the functionality, it's the packaging. In *ACM SIGSOFT Software Engineering Notes*, volume 20, pages 3–6. ACM, 1995.

[81] K. Sherif and A. Vinze. Barriers to adoption of software reuse. a qualitative study. *Information and Management*, 41:159–175, 2003.

[82] G. Sindre, R. Conradi, and E. Karlsson. The reboot approach to software reuse. *Journal of Systems and Software*, 30(3):201–212, 1995.

[83] O. P. N. Slyngstad, A. Gupta, R. Conradi, P. Mohagheghi, H. Rønneberg, and E. Landre. An empirical study of developers views on software reuse in statoil asa. In *ACM/IEEE international symposium on Empirical software engineering*, 2006.

[84] M. Sojer and J. Henkel. Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments. *Journal of the Association for Information Systems*, 2010.

[85] W. Spoelstra, M. Iacob, and M. van Sinderen. Software reuse in agile development organizations: a conceptual management tool. In *SAC 2011*, 2011.

[86] T. A. Standish. An essay on software reuse. *Software Engineering, IEEE Transactions on*, (5):494–497, 1984.

[87] K.-J. Stol, P. Avgeriou, M. A. Babar, Y. Lucas, and B. Fitzgerald. Key Factors for Adopting Inner Source. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2014.

[88] K.-J. Stol and B. Fitzgerald. Inner Source – Adopting Open Source Development Practices in Organizations: A Tutorial. In *IEEE Software*, 2015.

[89] J. W. Tukey. We need both exploratory and confirmatory. *The American Statistician*, 34(1):23–25, 1980.

[90] T. Vale, I. Crnkovic, E. S. de Almeida, P. A. da Mota Silveira Neto, Y. C. Cavalcanti, and S. R. de Lemos Meira. Twenty-eight years of component-based software engineering. *Journal of Systems and Software*, 111:128 – 148, 2016.

[91] J. Varnell-Sarjeant, A. A. Andrews, and A. Stefik. Comparing reuse strategies: An empirical evaluation of developer views. In *COMPSACW 2014*, pages 498–503. IEEE, 2014.

[92] D. M. Weiss, P. Clements, K. Kang, and C. Krueger. Software product line hall of fame. In *Software Product Line Conference, 2006 10th International*, pages 237–237, Aug 2006.

10. Appendix

10.1. Interview guide

Table 4 presents an overview of the interview topics as well as sample questions from the interview guide.

10.2. Scale aggregations

Table 5 provides the details of how the aggregation of the Likert scales was computed for each question. The column *Question ID* refers to the ID of the questions. *Scale U* encodes the type of the Likert scale for the respective question in the questionnaire of U. L4 encodes a four point scale; L5 encodes a five point scale. Together with the scale code, we report the two boundary values of each scale. The column *aggregation* shows how the aggregation of the respective scale was computed: P1 to P4 (or P5, where applicable) encode the possible options that could be selected. Their sum represents the aggregation of the number of participants that selected the respective options. For instance, for the question FAR1, we aggregated the number of participants that selected option P1 or P2 in code Low (L) and the number of participants that selected option P3 or P4 in code High (H). Next to the aggregation, we report the semantic value of the aggregated value.

10.3. Result of comparison RQ 1 and RQ 2

Tables 6 and 7 report the results of the comparison for RQs 1 and 2. They contain the following information:

Question ID refers to the code of the question in the questionnaire (for the respective questions see Tables 2 and 3).

Response options lists the possible answers.

Answ low/high represent the count of participants that selected the low/high end of the Likert-scale (case D) or marked a selection (case G) for the respective item, according to the analysis methodology described in Section 5. This data is reported to facilitate replication of the analysis by third parties.

Pval chi.square reports the p-value of the χ^2 test.

Verdict expresses if the vote for the item tended to the low or the high category. Blank fields in the tables represent information that was missing in the respective questionnaire.

Table 4: Interview guide: topics and sample questions for cases G and U

Interview topics
Economic, social, conceptual, and technical aspects of reuse
What are current goals of reuse? Where do you see potential?
Are there current issues? If yes, which?
Is there need for support? If yes, which? (Tools, processes, SE practices, ...)
Reuse assessment
When do you consider reuse as successful? Generally (fulfilling company goals)? From your specific perspective?
How do you assess the success of reuse, the fulfilment of reuse goals?
How do you decide on how reuse should be done?
How do you proceed to implement the selected way of reuse? Steps? Support?
In which phases do you expect/target reuse benefits? In which form should benefits occur?
Which business goals do you aim to support with (internal/external) reuse?
What are the current product goals and requirements?
In which way is reuse currently effected?
What kind of reuse do you aim for?
What are preconditions/requirements/challenges on the process/conceptual/technical/organisational/communication levels?
Experiences with current reuse
What works currently in terms of reuse? Why?
What did not work wrt. reuse so far? Why?
What were the biggest mistakes committed wrt. reuse?
How can these mistakes be mitigated/corrected?
Planning and conflicts of interest
Reuse as a source of conflict within company, local vs. global optimization
How is reuse planning done locally (department/team/group) and globally (company-wide)?
Balance of resources between products and basis?
What is harder: providing or maintaining reusable entities?
What do you consider essential for a professional stance wrt. reuse?
How do you address reuse and evolution of entities?
How important is tool support? For which parts of the process?
Product line adoption
Starting points, goals, trigger for decision
Process, issues and challenges: on which level? How addressed?
Successful? How could success be validated?
Which methods/strategies were effective? What would you do differently next time?
Product line evolution
Challenges? Key points? Success criteria and factors?
What (in terms of content) should be in the platform? In initial phase? In further evolution?
How do you proceed to coordinate the different stakeholders? Requirements engineering? Sources of requirements for platform? Process? How are requirements persisted? How is a decision reached?
Reference architectures: relevant? present? in which shape? Are deviations acceptable?
What needs to happen to achieve the acceptance and use of an internal framework? How is knowledge transferred?
How do you proceed with a common platform? What about governance, guarantees, compensations? Resources?
Development context
How important is homogeneity of process, tool infrastructure, quality assurance, goals?
Reuse from external sources
How important is the provenance of reused code? Security? Certification? Accountability, liability? Type of usage?
How do you procure/inspect/maintain (clone-and-own/external/central)? Who is responsible, has an overview, assesses external entities and their usage? Are there rules/limitations for the use of external entities?
Improvements
What is your most important wish for improvement wrt. reuse?

Table 5: Scale aggregations for questionnaire U. *Question ID* refers to the respective question, *Scale U* relates the type of the scale (*L4* for a 4 point Likert scale, *L5* for a 5 point Likert scale) and reports the extreme values of the given scale. *Aggregation* illustrates how, for the given scale, the values were aggregated in the categories *Low* and *High*. *P<n>* denotes the number of responses at the given point of the Likert scale.

Question ID	**Scale U**	**Aggregation**	
		Low	**High**
Extent of code reuse (ECR)			
ECR2	L5, no use, always use	P1: no use	P2+P3+P4+P5: use
ECR3	L5, no use, always use	P1: no use	P2+P3+P4+P5: use
Finding artefacts (FAR)			
FAR1	L4, never, always	P1+P2: irrelevant, low usage	P3+P4: regular, high usage
Reused artefacts (RAF)			
RAF1	L4, never, always	P1+P2: irrelevant, low usage	P3+P4: regular, high usage
RAF2	L4, never, always	P1+P2: low usage	P3+P4: high usage
Technical realization of reuse (TRR)			
TRR1	L4, does not apply, strongly applies	P1+P2: irrelevant, low usage	P3+P4: regular, high usage
TRR2	L4, does not apply, strongly applies	P1+P2: irrelevant, low frequency	P3+P4: regular, high frequency
Challenges, effects, and context factors of reuse (CHR)			
CHR1	L4, never, always	P1+P2: never, occasionally	P3+P4: regularly, always
CHR2	L4, never, always	P1+P2: never, occasionally	P3+P4: regularly, always
CHR3	L4, never, always	P1+P2: never, occasionally	P3+P4: regularly, always
Success factors and benefits (SFB)			
SFB1	L4, never, always	P1+P2: never, occasionally	P3+P4: regularly, always
SFB3	L4, unimportant, important	P1+P2: unimportant, slightly important	P3+P4: important, very important
SFB4	free text	—	
Reuse in everyday development practice (RED)			
RED1	L4, does not apply, strongly applies	P1+P2: does not apply, applies slightly	P3+P4: applies, strongly applies
RED4	L4, does not apply, strongly applies	P1+P2: does not apply, applies slightly	P3+P4: applies, strongly applies
Finding artefacts (FAR)			
FAR3	L4, does not apply, strongly applies	P1+P2: does not apply, applies slightly	P3+P4: applies, strongly applies

Table 6: Responses and values relevant for RQ1

	Question ID	Response options	U answ low	U answ high	U pval chi.square	U verdict	G answ low	G answ high	G pval chi.square	G verdict
1	ECR2	Serialization (e.g. XML).	14	38	<0.001	HIGH	17	19	0.74	-
2	ECR2	Networking.	19	35	0.03	HIGH	19	17	0.74	-
3	ECR2	Persistency.	16	37	<0.001	HIGH	19	17	0.74	-
4	ECR2	Visualization/GUI.	11	44	<0.001	HIGH	24	12	0.05	LOW
5	ECR2	Architecture (e.g. rich client, plugin).	16	38	<0.001	HIGH	21	15	0.32	-
6	ECR2	Algorithms	18	38	0.01	HIGH	NA	NA	NA	NA
7	ECR2	User Interfaces	17	38	0.01	HIGH	NA	NA	NA	NA
8	ECR2	General utility.	NA	NA	NA	NA	12	24	0.05	HIGH
9	ECR3	Domain-independent general functionality.	9	46	<0.001	HIGH	8	27	<0.001	HIGH
10	ECR3	Domain-specific functionality.	24	33	0.23	-	17	18	0.87	-
11	ECR3	Product-specific functionality.	23	33	0.18	-	26	9	<0.001	LOW
12	FAR1	Web search.	27	32	0.52	-	20	19	0.87	-
13	FAR1	Browsing repositories.	46	12	<0.001	LOW	23	16	0.26	-
14	FAR1	Communicating with colleagues.	17	43	<0.001	HIGH	14	25	0.08	-
15	FAR1	Code search tools.	52	6	<0.001	LOW	9	30	<0.001	HIGH
16	FAR1	Code recommenders.	58	1	<0.001	LOW	38	1	<0.001	LOW
17	FAR1	Browsing documentation.	50	11	<0.001	LOW	30	9	<0.001	LOW
18	FAR1	Tutorials.	50	8	<0.001	LOW	38	1	<0.001	LOW
19	FAR1	Other	2	0	0.16	-	36	3	<0.001	LOW
20	FAR1	Code completion.	NA	NA	NA	NA	37	2	<0.001	LOW
21	RAF1	System tests	44	14	<0.001	LOW	NA	NA	NA	NA
22	RAF1	Unit-Tests	41	18	<0.001	LOW	NA	NA	NA	NA
23	RAF1	Personas	50	7	<0.001	LOW	NA	NA	NA	NA
24	RAF1	Code in binary form	36	18	0.01	LOW	26	12	0.02	LOW
25	RAF1	Source code	30	29	0.90	-	1	37	<0.001	HIGH
26	RAF1	Informal design models (Box and lines, natural language)	42	12	<0.001	LOW	36	2	<0.001	LOW
27	RAF1	Semiformal design models (UML)	52	4	<0.001	LOW	38	0	<0.001	LOW
28	RAF1	Formal design models	45	9	<0.001	LOW	38	0	<0.001	LOW
29	RAF1	Own, domain specific design models	44	13	<0.001	LOW	36	2	<0.001	LOW
30	RAF1	Requirement documentation / Use cases	44	16	<0.001	LOW	33	5	<0.001	LOW
31	RAF1	Architecture documentation	45	11	<0.001	LOW	33	5	<0.001	LOW
32	RAF1	Prototypes	51	8	<0.001	LOW	36	2	<0.001	LOW
33	RAF1	UI Designs	38	21	0.03	LOW	28	10	<0.001	LOW
34	RAF1	Style guides	28	30	0.79	-	27	11	0.01	LOW
35	RAF1	Other	2	0	0.16	-	38	0	<0.001	LOW
36	RAF2	Developer Portals.	39	22	0.03	LOW	25	14	0.08	-
37	RAF2	Internal repositories.	43	17	<0.001	LOW	5	34	<0.001	HIGH
38	RAF2	Commercial repositories	51	7	<0.001	LOW	NA	NA	NA	NA
39	RAF2	Colleagues own department.	26	36	0.20	-	NA	NA	NA	NA
40	RAF2	Colleagues.	42	19	<0.001	LOW	24	15	0.15	-
41	RAF2	Open Source Repositories.	50	11	<0.001	LOW	25	14	0.08	-
42	TRR1	Code scavenging (copy, paste, modify).	47	10	<0.001	LOW	24	12	0.05	LOW
43	TRR1	Software libraries.	37	21	0.04	LOW	4	32	<0.001	HIGH
44	TRR1	Software frameworks.	36	19	0.02	LOW	17	19	0.74	-
45	TRR1	Component-based development.	38	18	0.01	LOW	28	8	<0.001	LOW
46	TRR1	Design patterns.	48	9	<0.001	LOW	23	13	0.10	-
47	TRR1	Architecture reuse.	47	9	<0.001	LOW	31	5	<0.001	LOW
48	TRR1	Product lines.	51	3	<0.001	LOW	35	1	<0.001	LOW
49	TRR1	Application generators.	51	3	<0.001	LOW	35	1	<0.001	LOW
50	TRR1	Code generators	50	6	<0.001	LOW	NA	NA	NA	NA
51	TRR1	None.	NA	NA	NA	NA	36	0	<0.001	LOW
52	TRR2	small code sections.	28	27	0.89	-	29	8	<0.001	LOW
53	TRR2	fine-grained, such as single methods/functions.	24	29	0.49	-	26	11	0.01	LOW
54	TRR2	one or more classes.	27	26	0.89	-	19	18	0.87	-
55	TRR2	complete libraries.	27	28	0.89	-	6	31	<0.001	HIGH
56	TRR2	coarse-grained, such as entire frameworks.	34	21	0.08	-	24	13	0.07	-

Table 7: This table contains the responses relevant for RQ2

	Question ID	Response options	U answ low	U answ high	U pval chi.square	U verdict	G answ low	G answ high	G pval chi.square	G verdict
57	CHR1	"Not invented here" phenomenon.	39	18	0.01	LOW	21	11	0.08	-
58	CHR1	Licensing/legal issues.	49	12	<0.001	LOW	18	14	0.48	-
59	CHR1	Difficulty of adapting artefact to project needs.	33	29	0.61	-	15	17	0.72	-
60	CHR1	Inconvenient granularity of reusable artefacts.	41	21	0.01	LOW	25	7	<0.001	LOW
61	CHR1	Process for clearance of external artefacts is too slow.	33	25	0.29	-	*Interview data*			
62	CHR1	Coordination effort with other departments.	30	32	0.80	-	*Interview data*			
63	CHR1	Other.	3	6	0.32	-	27	5	<0.001	LOW
64	CHR1	Finding the right artefacts is difficult.	*See FAR3 and RED1.*				14	18	0.48	-
65	CHR1	Accessing the artefact is difficult.	*See FAR3 and RED1.*				30	2	<0.001	LOW
66	CHR2	Loss of control.	46	14	<0.001	LOW	22	9	0.02	LOW
67	CHR2	Dependency explosion.	35	27	0.31	-	15	16	0.86	-
68	CHR2	Performance decay.	43	18	<0.001	LOW	27	4	<0.001	LOW
69	CHR2	Decrease of code understandability.	50	10	<0.001	LOW	20	11	0.11	-
70	CHR2	Ripple effects caused by changes in reused artefacts.	44	17	<0.001	LOW	19	12	0.21	-
71	CHR2	Code becomes unchangeable.	44	17	<0.001	LOW	28	3	<0.001	LOW
72	CHR2	Excessive restriction of the solution space.	47	13	<0.001	LOW	*Interview data*			
73	CHR2	No.	NA	NA	NA	NA	23	8	0.01	LOW
74	CHR2	Other.	3	2	0.66	-	30	1	<0.001	LOW
75	CHR3	Inconsistencies.	32	29	0.70	-	17	16	0.86	-
76	CHR3	High maintenance effort.	30	31	0.90	-	18	15	0.60	-
77	CHR3	Increased development effort.	34	27	0.37	-	12	21	0.12	-
78	CHR3	High testing load.	29	32	0.70	-	23	10	0.02	LOW
79	CHR3	Lower code quality.	43	16	<0.001	LOW				
80	CHR3	Duplicate implementations.	*Interview data*				9	24	0.01	HIGH
81	FAR3	I can readily read the source code available within the company.	45	14	<0.001	LOW	*Interview data*			
82	FAR3	I can effect required changes independently.	44	14	<0.001	LOW	*Interview data*			
83	FAR3	The integration of existing code requires little effort from my side.	42	14	<0.001	LOW	*Interview data*			
84	FAR3	The original developer of reused code is responsible for maintaining it.	14	44	<0.001	HIGH	*Interview data*			
85	RED1	The quality of artefacts is acceptable for reuse.	34	28	0.45	-	*Interview data*			
86	RED1	Reusable assets are classified in a comprehensive way.	55	6	<0.001	LOW	*Interview data*			
87	RED1	In everyday work I attempt to reuse artefacts.	5	58	<0.001	HIGH	*Interview data*			
88	RED1	When I want to reuse an artefact, it already exists in the company.	48	14	<0.001	LOW	*Interview data*			
89	RED1	Reusable assets are accessible with acceptable effort.	44	19	<0.001	LOW	*Interview data*			
90	RED1	Existing artefacts are found easily.	54	9	<0.001	LOW	*Interview data*			
91	RED1	Existing artefacts are understandable.	39	22	0.03	LOW	*Interview data*			
92	RED1	Existing artefacts match the required functionality.	42	21	0.01	LOW	*Interview data*			
93	RED1	Functionally matching artefacts can be integrated with ease.	42	21	0.01	LOW	*Interview data*			
94	RED4	Reuse is the responsibility of individual developers.	11	51	<0.001	HIGH	*Interview data*			
95	RED4	Planning for reuse happens in a grassroots fashion.	20	42	0.01	HIGH	*Interview data*			
96	RED4	Reuse is initiated and coordinated by a small group of people.	34	28	0.45	-	*Interview data*			
97	RED4	Providing reusable assets is a shared initiative.	46	18	<0.001	LOW	*Interview data*			
98	RED4	Dedicated personal is assigned to provide reusable assets.	41	23	0.02	LOW	*Interview data*			
99	RED4	Development and provision of reusable artefacts is coordinated across departments and departments.	51	13	<0.001	LOW	*Interview data*			

Table 7: This table contains the responses relevant for RQ2 — continued from previous page

	Question ID	Response options	U answ low	U answ high	U pval chi.square	U verdict	G answ low	G answ high	G pval chi.square	G verdict
100	SFB1	Less maintenance effort.	33	26	0.36	-	10	22	0.03	HIGH
101	SFB1	Higher consistency.	31	29	0.80	-	20	12	0.16	-
102	SFB1	New functionality is made available.	38	22	0.04	LOW	19	13	0.29	-
103	SFB1	Higher code quality.	37	23	0.07	-	17	15	0.72	-
104	SFB1	Higher development pace.	36	24	0.12	-	3	29	<0.001	HIGH
105	SFB1	Regular bug fixes.	NA	NA	NA	NA	21	11	0.08	-
106	SFB1	None.	NA	NA	NA	NA	32	0	<0.001	LOW
107	SFB1	Other.	2	2	1	-	32	0	<0.001	LOW
108	SFB3	Suitable abstractions.	14	46	<0.001	HIGH	*see SBF4.*			
109	SFB3	Direct communication culture.	9	51	<0.001	HIGH	*see SBF4.*			
110	SFB3	Suitable incentives.	19	41	0.01	HIGH	*see SBF4.*			
111	SFB3	Higher quality of artefacts.	5	57	<0.001	HIGH	*see SBF4.*			
112	SFB3	Well-defined process for reuse.	14	47	<0.001	HIGH	*see SBF4.*			
113	SFB3	Supporting infrastructure and tools.	9	53	<0.001	HIGH	*see SBF4.*			
114	SFB3	Stricter rules for dependency management.	16	43	<0.001	HIGH	23	6	<0.001	LOW
115	SFB3	Homogeneous development culture.	17	44	<0.001	HIGH	*see SBF4.*			
116	SFB3	None of the above.	NA	NA	NA	NA	27	2	<0.001	LOW
117	SFB3	Other.	2	4	0.41	-	25	4	<0.001	LOW
118	SFB3	Clear strategic decisions for interface support.	7	54	0	HIGH	21	8	0.02	LOW
119	SFB3	Introduce maturity levels for reused artefacts.	22	40	0.02	HIGH	24	5	<0.001	LOW
120	SFB3	Bundle code more coherently in terms of functionality, e.g. into dedicated libraries.	21	40	0.02	HIGH	17	12	0.35	-
121	SFB3	Split libraries to provide more specific functionality.	25	35	0.2	-	18	11	0.19	-
122	SFB3	Merge libraries to ease the discovery of already implemented functionality.	NA	NA	NA	NA	23	6	<0.001	LOW
123	SFB3	List available artefacts in a "marketplace" to ease the discovery of useful functionality.	9	54	0	HIGH	16	13	0.58	-
124	SFB3	Announce the release of new artefacts.	8	55	0	HIGH	25	4	<0.001	LOW
125	SFB3	Developers could broadcast requests for specific functionality.	10	50	0	HIGH	28	1	<0.001	LOW
126	SFB3	Existing code could be consolidated and prepared for reuse.	16	46	0	HIGH	23	6	<0.001	LOW
127	SFB3	Structured and company-wide requirements engineering.	14	49	0	HIGH	*Interview data*			
128	SFB3	Focus on usefulness for the respective customer.	22	39	0.03	HIGH	NA	NA	NA	NA
129	SFB4	Adequate abstractions.	*Free text*				13	18	0.37	-
130	SFB4	Direct communication culture.	*Free text*				25	6	<0.001	LOW
131	SFB4	Suitable incentives.	*Free text*				29	2	<0.001	LOW
132	SFB4	High quality of artefacts.	*Free text*				10	21	0.05	HIGH
133	SFB4	Well defined reuse process.	*Free text*				25	6	<0.001	LOW
134	SFB4	Supporting infrastructure and tools.	*Free text*				11	20	0.11	-
135	SFB4	Dependency management.	*Free text*				21	10	0.05	LOW
136	SFB4	Homogeneous development culture.	*Free text*				21	10	0.05	LOW
137	SFB4	Other.	*Free text*				29	2	<0.001	LOW

10.4. Questionnaires

This part of the appendix presents the two original questionnaires as distributed to the participants of the studies at G and U. The questionnaire at G was rolled out in English, the one at U in German. The questions used for the comparison are translated to English in the paper and the respective data tables.

10.4.1. Questionnaire for study at G

The questionnaire has been redacted to remove sensitive information and comply with the disclosure policy of the company.

Assessing reuse in the software industry

Welcome to our survey on reuse in the software industry!

Reuse of development artefacts has become an important aspect in large-scale software development. Many studies have assessed different aspects of reuse in open-source projects. Data on reuse in industrial context, however, is scarce.

With your help, we aim to collect information about a wide range of aspects of reuse in industrial software development. We want to better understand current strategies and issues related to reuse in the industrial context. Based on these results, we aim to focus our research on aspects of reuse relevant in industry.

The questions address topics in company-wide reuse management, as well as project-specific details. If you currently work on more than one project, please answer the questions for the project you are mainly working on.

The survey is anonymous. The participating researchers are under NDA. Any information drawn from this study is subject to clearance by Google before publication.

Should you have any questions regarding this survey, do not hesitate to contact us via mail:
Contact Technical University Munich: bauerv@in.tum.de
Contact Google: klimek@google.com

Thank you for participating in our survey!

Weiter »

Assessing reuse in the software industry

Some facts about you

Please indicate your current role.

- [ ] Engineer (development).
- [ ] Engineer (maintenance).
- [ ] Technical lead.
- [ ] Product manager.
- [ ] Technical program manager.

Which product area are you currently working in?

How many years of experience do you have in your current role?
Including equivalent roles at other companies.

- [ ] < 1 year.
- [ ] 1 year to 4 years.
- [ ] 5 years to 10 years.
- [ ] 11 years to 15 years.
- [ ] 15 years to 20 years.
- [ ] > 20 years.

For how many years have you been working for your current company?

- [ ] < 1 year.
- [ ] 1 year to 3 years.
- [ ] 4 years to 6 years.
- [ ] 6 years to 10 years.
- [ ] > 10 years.

Please indicate your [redacted]

[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
○	○	○	○	○	○	○	○	○	○

[redacted]

- ○ [redacted]
- ○ [redacted]
- ○ [redacted]

On average, how long have you been on your project(s)?

- ( ) < 1 year.
- ( ) 1 year to 3 years.
- ( ) 4 years to 10 years.
- ( ) > 10 years.

If you currently develop software, which of the following programming languages are you using on a regular basis?

- [ ] Java
- [ ] C/C++
- [ ] C#
- [ ] Python
- [ ] Ruby
- [ ] Php
- [ ] Scala
- [ ] JavaScript
- [ ] Go
- [ ] Haskell
- [ ] Sonstiges: ______

Please indicate the country you are currently working in.

______

« Zurück   Weiter »

Assessing reuse in the software industry

Legal aspects of reuse

Which of the following statements do you believe reflects best legal aspects of reuse in your projects?

- ○ Reuse of third-party artefacts always has to be approved.
- ○ Legal aspects of reuse are very carefully monitored.
- ○ Legal aspects of reuse are considered.
- ○ Legal aspects of reuse are ignored.
- ○ Legal aspects of reuse are not relevant, because we reuse internal artefacts only.
- ○ None of the above.
- ○ I don't know.

Who do you believe to be in charge to ensure compliance with legal regulations of reuse?

- ☐ The project manager.
- ☐ The developer.
- ☐ There is a dedicated person/department surveilling compliance.
- ☐ I don't know.
- ☐ Sonstiges: ________

« Zurück | Weiter »

[redacted]

Assessing reuse in the software industry

Reuse strategies and artefacts

Reuse strategies

For which types of projects do you employ strategical reuse?
Strategical reuse implies that reuse is driven by specific organizational goals. Usually guidelines or policies describe which reuse is adequate for a given situation.

- [ ] We generally follow strategical reuse.
- [ ] For prototype development.
- [ ] For product development.
- [ ] For internal tool development.
- [ ] We do not follow a specific strategy for reuse.

For which types of projects do you employ ad-hoc reuse?
Ad-hoc reuse means that developers are allowed to reuse any available artefact which seems suitable for the task at hand.

- [ ] We generally follow ad-hoc reuse.
- [ ] For prototype development.
- [ ] For product development.
- [ ] For internal tool development.
- [ ] We do not follow a specific strategy for reuse.

Reused artefacts

Which artefacts are you encouraged to reuse?

- [ ] Binaries.
- [ ] Source code.
- [ ] Informal design models (Box and lines, natural language).
- [ ] Standardized semiformal design models (e.g. UML).
- [ ] Formal design models.
- [ ] Own, domain specific design models.

- ☐ Requirement documentation / Use cases.
- ☐ Architecture documentation.
- ☐ Prototypes.
- ☐ UI Designs.
- ☐ Style guides.
- ☐ Sonstiges: ________

Which are the top-three types of artefacts you reuse?

- ☐ Code in binary form
- ☐ Source code
- ☐ Informal design models (Box and lines, natural language)
- ☐ Semiformal design models (UML)
- ☐ Formal design models
- ☐ Own, domain specific design models
- ☐ Requirement documentation / Use cases
- ☐ Architecture documentation
- ☐ Prototypes
- ☐ UI Designs
- ☐ Style guides
- ☐ Sonstiges: ________

« Zurück | Weiter »

Assessing reuse in the software industry

Sources for reusable artefacts

What are your top-three ways to search for reusable artefacts?

- [ ] Web search.
- [ ] Browsing repositories.
- [ ] Communicating with colleagues.
- [ ] Code search tools.
- [ ] Code recommenders.
- [ ] Code completion.
- [ ] Browsing documentation.
- [ ] Tutorials.
- [ ] Sonstiges: ______

What are your standard sources for reusable artefacts?

- [ ] Sourceforge.
- [ ] Maven.
- [ ] GitHub.
- [ ] Stackoverflow.
- [ ] Internal repositories.
- [ ] Colleagues.
- [ ] Sonstiges: ______

What do you do to properly understand and adequately select reusable artefacts?

- [ ] I read guidelines.
- [ ] I review interface documentation.
- [ ] I review implementations.
- [ ] I participate in trainings for third-party technologies/artefacts.
- [ ] I explore third-party products.
- [ ] I look for example usages on blogs and tutorials.
- [ ] Nothing.
- [ ] Sonstiges: ______

« Zurück | Weiter »

Powered by Google Docs

Assessing reuse in the software industry

Extent of code reuse

How many different third-party libraries and frameworks have you introduced your project?
Third-party refers to artefact producers outside your company.

- ◯ None.
- ◯ 1-2
- ◯ 3-4
- ◯ more than 5.

The last time you wanted to migrate away from an artefact, how difficult was it?

- ◯ 1 - minor impact, linking with a new library was be sufficient.
- ◯ 2
- ◯ 3
- ◯ 4 - high impact, migrating away was (nearly) impossible.
- ◯ 0 - have never done it.

The last time you reused an artefact, how complex was it?

- ◯ 1 - very simple, e.g. logging functionality.
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5 - very complex, e.g. mathematical algorithm library.

What type of functionality do you reuse?

- ☐ Serialization (e.g. XML).
- ☐ Networking.
- ☐ Persistency.
- ☐ Visualization/GUI.
- ☐ Architecture (e.g. rich client, plugin).
- ☐ General utility.
- ☐ None.
- ☐ Sonstiges: ________

What is the scope of the reused artefacts?

- ☐ Domain-independent general functionality.
- ☐ Domain-specific functionality.

- [ ] Product-specific functionality.
- [ ] Sonstiges: ________

Which of the following possibilities of reuse do you employ most? Please indicate the top three.

- [ ] Code scavenging (copy, paste, modify).
- [ ] Software libraries.
- [ ] Software frameworks.
- [ ] Component-based development.
- [ ] Design patterns.
- [ ] Architecture reuse.
- [ ] Product lines.
- [ ] Application generators.
- [ ] None.
- [ ] Sonstiges: ________

What granularities do the reused entities typically have?

- [ ] fine-grained, such as single methods/functions.
- [ ] small code sections.
- [ ] one or more classes.
- [ ] complete libraries.
- [ ] coarse-grained, such as entire frameworks.
- [ ] Sonstiges: ________

« Zurück | Weiter »

10.4.2. Questionnaire for study at U

The questionnaire has been redacted to remove sensitive information and comply with the disclosure policy of the company.

Druckversion

Fragebogen

1 Willkommen

Herzlich Willkommen zu unserer Umfrage zur "Wiederverwendung in der Software-Entwicklung"!

Mit diesem Fragebogen möchten wir die Informationen aus bereits in [redacted] geführten Interviews mit Ihrer Hilfe auf eine breitere Basis stellen.

Unser Ziel ist es, ein besseres Verständnis für die aktuellen Strategien und Probleme in punkto Wiederverwendung in [redacted] zu erhalten. Die Informationen dienen somit dazu, ein besseres Bild über die vorhandenen Prozesse zu bekommen und diese zu verbessern.

Unsere Fragen adressieren sowohl Themen des unternehmensweiten Wiederverwendungsmanagements, als auch abteilungsspezifische Details.

Die Umfrage dauert ca. 30 Minuten und ist anonym. Ergebnisse werden nur in aggregierter Form erhoben. Alle Informationen, die mittels dieser Studie gewonnen werden, unterliegen vor Veröffentlichung einer Überprüfung und Freigabe durch [redacted]

Bei Fragen zu dieser Umfrage kontaktieren Sie uns gerne per Email:
Kontakt Technische Universität München: Veronika Bauer [redacted]
Kontakt [redacted]

Vielen Dank für Ihre Teilnahme!

2 Persönliche Daten

Persönliche Angaben

In diesem Abschnitt des Fragebogens erheben wir einige Daten zu Ihrer Person und Ihrer Rolle in [redacted]

In welcher [redacted] bteilung arbeiten Sie aktuell?

[redacted]

Bitte wählen Sie Ihren fachlichen Schwerpunkt aus:

Entwicklung
Architektur
UX-Design
Management

Bitte wählen Sie Ihre Rolle aus:

○ Mitarbeiter

○ Führungskraft

Wieviele Jahre Berufserfahrung haben Sie in Ihrer aktuellen Tätigkeit?

< 5 Jahre	5 - 10 Jahre	11 - 15 Jahre	16 - 20 Jahre	21 - 25 Jahre	> 25 Jahre.
○	○	○	○	○	○

Seit wievielen Jahren arbeiten Sie für [redacted]

< 5 Jahre	5 - 10 Jahre	11 - 15 Jahre	16 - 20 Jahre	21 - 25 Jahre	> 25 Jahre.
○	○	○	○	○	○

Falls Sie zurzeit Software entwickeln, welche der folgenden Programmiersprachen benutzen Sie regelmäßig?

	Nie	Gelegentlich	Regelmäßig	Ständig
Java	○	○	○	○
C	○	○	○	○
C#	○	○	○	○
Php	○	○	○	○
JavaScript	○	○	○	○
C++	○	○	○	○
SQL	○	○	○	○
Sonstiges	○	○	○	○

3 Wiederverwendung im Berufsalltag

Im Folgenden wird der Begriff Artefakt für Ergebnisse des Entwicklungsprozesses benutzt, z B. Quellcode, Anforderungsdokumente oder Dokumentation.

Wie sehr treffen folgende Aussagen auf Ihre Erfahrung mit Wiederverwendung im Alltag zu?

	Trifft nicht zu.			Trifft stark zu.
Im Alltag versuche ich Wiederverwendung zu betreiben.	○	○	○	○
Wenn ich etwas wiederverwenden möchte, existiert das benötigte Artefakt schon in [redacted]	○	○	○	○
In [redacted] vorhandene Artefakte kann ich mit vertretbarem Aufwand für die Nutzung in meinem Projekt bekommen.	○	○	○	○
Es ist einfach, vorhandene Artefakte zu finden.	○	○	○	○
Die vorhandenen Artefakte sind gut zu verstehen.	○	○	○	○
Die vorhandenen Artefakte erfüllen die von mir gesuchte Funktionalität.	○	○	○	○
Die funktional passenden Artefakte können einfach integriert werden.	○	○	○	○
Die Qualität der Artefakte ist gut genug für die Wiederverwendung.	○	○	○	○
Die vorhandenen und wiederverwendbaren Artefakte sind verständlich kategorisiert.	○	○	○	○

Aus welchen Gründen betreiben Sie Wiederverwendung?

Welche Ziele werden Ihnen bezüglich Wiederverwendung vorgegeben?

Wie stark treffen folgende Aussagen auf die Umsetzung der Wiederverwendung in Ihrer Organisationseinheit zu?

Wiederverwendung involviert viele organisatorische Ebenen. Mit dieser Frage möchten wir ausloten, auf welchen Ebenen das Thema in [redacted] eine Rolle spielt.

	Trifft nicht zu.			Trifft stark zu.
Wiederverwendung wird individuell von den Entwicklern betrieben.	○	○	○	○
Die Koordination von Wiederverwendung findet auf inoffiziellem Weg statt.	○	○	○	○
Ein kleinerer Personenkreis initiiert und koordiniert Wiederverwendung.	○	○	○	○
Die Bereitstellung von wiederverwendbaren Artefakten ist eine gemeinschaftliche Initiative.	○	○	○	○
Es gibt designierte Rollen, die für die Herstellung von wiederverwendbaren Artefakten zuständig sind.	○	○	○	○
Die Entwicklung und Bereitstellung wiederverwendbarer Artefakte wird abteilungsübergreifend abgestimmt.	○	○	○	○

4 Wiederverwendungsstrategien

Wiederverwendungsstrategien

Die folgenden Aussagen betreffen die allgemeinen strategischen Aspekte von Wiederverwendung. Bitte geben Sie Ihren Zustimmungsgrad an.

	Trifft nicht zu.			Trifft stark zu.
Die Möglichkeit, Wiederverwendung auszunutzen, beeinflusst unsere Geschäftsentscheidungen.	○	○	○	○
Wir nutzen strukturierte Anforderungserhebung, um Möglichkeiten zur Wiederverwendung zu identifizieren.	○	○	○	○
Wir sammeln gezielt bereits existierende Artefakte für Wiederverwendung.	○	○	○	○
Wiederverwendung findet auf eine geplante und strukturierte Weise statt.	○	○	○	○
Die Angemessenheit von Wiederverwendung wird am unmittelbaren Nutzen gemessen.	○	○	○	○
Mein Management fördert Wiederverwendung.	○	○	○	○
Wir haben einen strukturierten Entwicklungsprozess, der Wiederverwendung explizit mit einschließt.	○	○	○	○
Wir entwickeln Produktfamilien.	○	○	○	○
Software ist unser wichtigstes Kapital.	○	○	○	○
Wir halten unsere Wiederverwendungsstrategie strikt ein.	○	○	○	○
Wir zielen darauf ab, Lücken in unseren wiederverwendbaren Artefakten zu füllen.	○	○	○	○

Die folgenden Aussagen betreffen die organisatorische Umsetzung von Wiederverwendung in Ihrer Abteilung. Bitte geben Sie Ihren Zustimmungsgrad an.

	Trifft nicht zu.			Trifft stark zu.
Wir stellen wiederverwendbare Artefakte innerhalb unserer Abteilung zur Verfügung.	○	○	○	○
Wir stellen firmenweite Basisfunktionalität zur Verfügung.	○	○	○	○
Wir verfügen über eine einheitliche und tragfähige Infrastruktur, die Wiederverwendung unterstützt.	○	○	○	○
Unsere Produkte spiegeln unsere Organisationsstruktur wider.	○	○	○	○
Unsere Produkte spiegeln die Art, wie wir Wiederverwendung betreiben, wider.	○	○	○	○
Wir messen den Nutzen, der durch Wiederverwendung entsteht.	○	○	○	○
Wir erheben den Aufwand, der in die Wiederverwendung fließt.	○	○	○	○
Alle wiederverwendbaren Artefakte haben einen Verantwortlichen, der sich um Wartung und Support kümmert.	○	○	○	○
In unserer Abteilung nehmen wir bereits existierende Artefakte und passen sie für uns an.	○	○	○	○
In unserer Abteilung tendieren wir dazu, unsere eigenen Versionen von existierenden Artefakten herzustellen.	○	○	○	○

5 Herausforderungen und potenzielle Risiken der Wiederverwendung

Herausforderungen und potenzielle Risiken der Wiederverwendung

Wie häufig beeinträchtigen folgende Aspekte den Wiederverwendungsprozess in Ihrem Team?

	Nie	Gelegentlich	Regelmäßig	Ständig
"Not invented here" Phänomen.	○	○	○	○
Lizenzierung/gesetzliche Aspekte.	○	○	○	○
Schwierigkeiten, die Artefakte auf die Projektanforderungen anzupassen.	○	○	○	○
Unpassende Granularität der wiederverwendbaren Artefakte.	○	○	○	○
Prozess für Freigabe von externen Artefakten ist zu langsam.	○	○	○	○
Abstimmungsaufwand mit anderen Abteilungen.	○	○	○	○
Sonstige:	○	○	○	○

Wie häufig treten potentielle Nachteile der Wiederverwendung in Ihrem Projekt auf?

	Nie	Gelegentlich	Regelmäßig	Ständig
Kontrollverlust	○	○	○	○
Abhängigkeitsexplosion	○	○	○	○
Performanzeinbußen	○	○	○	○
Verringerung der Veständlichkeit des Codes	○	○	○	○
Folgeänderungen durch Änderungen in den wiederverwendeten Artefakten	○	○	○	○
Code wird unveränderbar	○	○	○	○
Übermäßige Einschränkung des Lösungsraums	○	○	○	○
Sonstige:	○	○	○	○

Wie häufig treten folgende Probleme aus Mangel an Wiederverwendung in Ihrem Projekt auf?

	Nie	Gelegentlich	Regelmäßig	Ständig

Inkonsistenzen	○	○	○	○
Hoher Wartungsaufwand	○	○	○	○
Zunehmender Entwicklungsaufwand (z.B. durch doppelte Implementierungen)	○	○	○	○
Steigender Testaufwand	○	○	○	○
Geringere Code-Qualität	○	○	○	○
Sonstige: [ ]	○	○	○	○

Gibt es Ihrer Meinung nach weitere Schwierigkeiten und Herausforderungen, die bisher nicht genannt sind?

Sie können diese hier kurz beschreiben.

[ ]

6 Erfolgsfaktoren und potenzielle Vorteile der Wiederverwendung

Erfolgsfaktoren und potenzielle Vorteile der Wiederverwendung

Wie häufig realisieren sich die potentiellen Vorteile der Wiederverwendung in Ihrem Projekt?

	Nie	Gelegentlich	Regelmäßig	Ständig
Weniger Wartungsaufwand	○	○	○	○
Höhere Konsistenz	○	○	○	○
Neue Funktionalität verfügbar	○	○	○	○
Höhere Code-Qualität	○	○	○	○
Höheres Entwicklungstempo	○	○	○	○
Sonstiges [ ]	○	○	○	○

Wie wichtig wären folgende Faktoren, um Wiederverwendung in Ihrem Unternehmen nützlicher zu machen?

	1 - Unwichtig	2	3	4 - Sehr wichtig
Code kohärenter zusammenbündeln im Sinne von Funktionalität (z B. in Bibliotheken).	○	○	○	○
Bibliotheken aufteilen, um genauere Funktionalität zu liefern.	○	○	○	○
Listen mit verfügbaren Artefakten in einem "Marketplace", um das Auffinden von bereits implentierten Funktionalität zu vereinfachen.	○	○	○	○
Verfügbarkeit von neuen Artefakten bekanntmachen.	○	○	○	○
Entwickler könnten Nachfragen für bestimmte Funktionalitäten melden.	○	○	○	○
Bereits existierender Code könnte konsolidiert und für die Wiederverwendung aufbereitet werden.	○	○	○	○
Strengere Regeln zum Abhängigkeitsmanagement.	○	○	○	○
Klare Kommunikation von Schnittstellenstrategien (z.B. geplante Stabilität, Dauer der Unterstützung).	○	○	○	○
Wiederverwendbare Artefakte mit einem Reifegrad kennzeichnen.	○	○	○	○
Strukturierte und abteilungsübergreifende Anforderungserhebung.	○	○	○	○
Orientierung am Nutzen für den jeweiligen Kunden.	○	○	○	○
Sonstiges [ ]	○	○	○	○

Wie wichtig sind folgende Faktoren, damit Wiederverwendung in Ihrer Firma nutzbringender wird?

	1 - Unwichtig	2	3	4 - Sehr wichtig
Geeignete Abstraktionen	○	○	○	○
Direkte Kommunikationskultur	○	○	○	○
Passende Anreize	○	○	○	○
Höhe Qualität der Artefakte	○	○	○	○
Klar definierter Prozess zur Wiederverwendung	○	○	○	○
Unterstützende Infrastruktur und Werkzeuge	○	○	○	○
Abhängigkeitsmanagement	○	○	○	○
Homogene Entwicklungskultur	○	○	○	○
Sonstiges [ ]	○	○	○	○

Welche Faktoren tragen Ihrer Meinung zu den erfolgreichen Wiederverwendungsprojekten [redacted] bei?

[ ]

7 Wiederverwendbare Artefakte

Wiederverwendbare Artefakte

Wie häufig verwenden Sie folgende Artefakte wieder?

	Nie	Gelegentlich	Regelmäßig	Ständig
Quellcode in binärer Form	○	○	○	○
Quellcode	○	○	○	○
Informelle Designmodelle (Kästen und Linien, natürliche Sprache)	○	○	○	○
Standarisierte semi-formale Designmodelle (z.B. UML)	○	○	○	○
Formale Designmodelle	○	○	○	○
Eigene, dömanspezifische Designmodelle	○	○	○	○
Anforderungsdokumentation/Anwendungsfälle	○	○	○	○
Architekturdokumentation	○	○	○	○
Prototypen	○	○	○	○
UI Designs	○	○	○	○
Gestaltungsrichtlinien	○	○	○	○
Systemtests	○	○	○	○
Unit-Tests	○	○	○	○

Sonstiges	○	○	○	○
Personas	○	○	○	○

Was sind Ihre Bezugsquellen für wiederverwendbare Artefakte?

	Nie	Gelegentlich	Regelmäßig	Ständig
Entwicklerportale (z B. Stackoverflow)	○	○	○	○
Interne Repositories	○	○	○	○
Kommerzielle Repositories (z B. CodePlex)	○	○	○	○
Kollegen in eigener [redacted]abteilung	○	○	○	○
Kollegen in anderer [redacted] bteilung	○	○	○	○
Open Source Repositories (z.B. GitHub, Sourceforge, ...)	○	○	○	○
Sonstiges:	○	○	○	○

8 Quellen von wiederverwendbaren Artefakten

Auffinden von wiederverwendbaren Artefakten

Wie häufig benutzen Sie folgende Wege, um wiederverwendbare Artefakte aufzufinden?

	Nie	Gelegentlich	Regelmäßig	Ständig
Websuche	○	○	○	○
Suche in lokalen Repositories	○	○	○	○
Kommunikation mit Kollegen	○	○	○	○
Werkzeuge für Code-Suche	○	○	○	○
Code-Recommender	○	○	○	○
Explorieren von Dokumentation	○	○	○	○
Tutorien	○	○	○	○
Sonstiges:	○	○	○	○

Was machen Sie, um wiederverwendbare Artefakte richtig zu verstehen und die angemessenen auszuwählen?

	Nie	Gelegentlich	Regelmäßig	Ständig
Ich lese Leitfäden.	○	○	○	○
Ich lese die Schnittstellendokumentation.	○	○	○	○
Ich lese den Quellcode.	○	○	○	○
Ich nehme an Trainings für Technologien/Artefakte von Drittanbietern teil.	○	○	○	○
Ich experimentiere mit Produkten von Drittanbietern.	○	○	○	○
Ich lese Tutorials.	○	○	○	○
Ich arbeite mit Code-Beispielen.	○	○	○	○
Ich kontaktiere den Ersteller.	○	○	○	○
Ich tausche mich mit Kollegen aus.	○	○	○	○
Ich verwende Unit-Tests.	○	○	○	○
Sonstiges:	○	○	○	○

Wie weit treffen folgende Aussagen bezüglich der Verfügbarkeit und Änderbarkeit des firmeninternen Quellcodes zu?

	Trifft nicht zu.			Trifft stark zu.
Der firmenweit vorhandene Quellcode ist für mich gut einsehbar.	○	○	○	○
Ich kann nötige Änderungen unabhängig durchführen.	○	○	○	○
Die Integration von bestehendem Code erfordert wenig Aufwand von meiner Seite.	○	○	○	○
Die Verantwortung zur Pflege von wiederverwendetem Code liegt bei dem jeweiligen Ersteller.	○	○	○	○

9 Artefakte zur Verfügung stellen

Bereitstellen von Artefakten

Stellen Sie Artefakte (z.B. Code) innerhalb des Unternehmens zur Wiederverwendung zur Verfügung?

- ○ Ja, alles ist für andere Projekte verfügbar.
- ○ Ja, manche Artefakte sind verfügbar.
- ○ Nein, Artefakte werden nicht mit anderen Projekten geteilt.

Aus welchen Gründen stellen Sie Artefakte (nicht) zur Wiederverwendung bereit?

Wie stellen Sie sicher, dass andere Ihre Artefakte richtig wiederverwenden können?

Bitte beschreiben Sie kurz ob Sie Maßnahmen treffen, und wenn ja welche, um andere bei der Wiederverwendung Ihrer Artefakte zu unterstützen.

Welche Art von Funktionalität teilen Sie auf welcher Ebene?

	Stelle ich nicht zur Verfügung.	Stelle ich bereit für abteilungsinterne Basis.	Stelle ich bereit für alle (nicht Basis).	Integriere ich in firmeninterne Basis.
Domänenunabhängige Funktionalität.	○	○	○	○
Domänenspezifische Funktionalität, z.B. Standardlösungen.	○	○	○	○
Projektspezifische Funktionalität, z B. bestimmte Algorithmen, die auf die Projektanforderungen zugeschnitten sind.	○	○	○	○
Sonstiges	○	○	○	○

Über welche Kanäle stellen Sie Ihre Artefakte zur Verfügung?

	Nie	Gelegentlich	Meistens	Immer
Firmenweites Repository	○	○	○	○
Tutorien	○	○	○	○

Blogs	○	○	○	○
Email	○	○	○	○
Entwicklerportal	○	○	○	○
Abteilungsinternes Repository	○	○	○	○
Sonstiges [ ]	○	○	○	○

10 Umfang der Code-Wiederverwendung

Umfang der Code-Wiederverwendung

Wie verhält sich in ihrem Projekt geschätzt der Anteil an <u>firmenintern</u> wiederverwendetem Code zum Anteil des neu entwickelten Codes?

	Keine Angabe	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Anteil Wiederverwendung												

Welche Art von Funktionalität verwenden Sie wieder und woher beziehen Sie diese?

	Verwende ich nicht wieder.	Beziehe ich aus abteilungsinterner Basis.	Beziehe ich aus firmeninterner Quelle (nicht Basis).	Beziehe ich aus firmeninterner Basis.	Beziehe ich von externen Anbietern.
Serialisierung (z.B. XML)	○	○	○	○	○
Netzwerk	○	○	○	○	○
Persistenz	○	○	○	○	○
Visualisierung/GUI	○	○	○	○	○
Architektur (z B. Rich client, Plugin)	○	○	○	○	○
Algorithmen.	○	○	○	○	○
Oberflächen.	○	○	○	○	○
Sonstiges: [ ]	○	○	○	○	○

Welchen Typ Funktionalität verwenden Sie wieder?

	Verwende ich nicht wieder.	Beziehe ich aus abteilungsinterner Basis.	Beziehe ich aus firmeninterner Quelle (nicht Basis).	Beziehe ich aus firmeninterner Basis.	Beziehe ich von externen Anbietern.
Domänenunabhängige Basisfunktionalität, z B. Infrastrukturlösungen.	○	○	○	○	○
Domänenspezifische Basisfunktionalität, z B. Standardlösungen für das Geschäftsfeld.	○	○	○	○	○
Projektspezifische Funktionalität, z B. bestimmte Algorithmen, die auf die Projektanforderungen zugeschnitten sind.	○	○	○	○	○

Haben Sie Bibliotheken oder Frameworks von Drittanbietern in Ihr Projekt eingebracht?

Ja	Nein
○	○

11 Technische Umsetzung der Wiederverwendung

Wie setzen Sie Wiederverwendung technisch um?

Ich baue auf:

	Trifft nicht zu.			Trifft stark zu.
Copy-Paste-Modify.	○	○	○	○
Softwarebibliotheken.	○	○	○	○
Software Frameworks.	○	○	○	○
Komponentenbasierte Entwicklung.	○	○	○	○
Designvorlagen.	○	○	○	○
Architekturvorlagen.	○	○	○	○
Produktlinien.	○	○	○	○
Anwendungsgeneratoren.	○	○	○	○
Code-Generatoren.	○	○	○	○
Sonstiges: [ ]	○	○	○	○

Welche Granularität haben die wiederverwendeten Entitäten typischerweise?

	Nie	Gelegentlich	Regelmäßig	Ständig
Kleine Codeabschnitte.	○	○	○	○
Einzelne Methoden/Funktionen.	○	○	○	○
Eine oder mehrere Klassen.	○	○	○	○
Vollständige Bibliotheken.	○	○	○	○
Ganze Frameworks.	○	○	○	○
Sonstiges [ ]	○	○	○	○

Wie schwierig war das Integrieren der letzten Funktionalität, die Sie wiederverwendet haben?

	1- Sehr einfach	2	3	4	5 - Sehr Komplex
Komplexität technische Integration	○	○	○	○	○

12 Migrieren von wiederverwendeten Artefakten

Wartung von wiederverwendeten Artefakten

Was machen Sie, wenn Sie Fehler in von anderen Projekten-Teams im Unternehmen bereitgestellten Artefakten finden?

	Nie	Gelegentlich	Meistens	Immer
Ich tausche das Artefakt aus.	○	○	○	○
Ich behandle den Fehler in meinem Projekt.	○	○	○	○

Ich sende einen Fehlerbericht.	○	○	○	○
Ich ignoriere den Fehler.	○	○	○	○
Sonstiges [ ]	○	○	○	○
Ich behebe den Fehler an der Quelle.	○	○	○	○

Wann migrieren Sie zu einer neuen Version von einem von anderen Firmen bereitgestellten Artefakt?

	Nie	Gelegentlich	Meistens	Immer
Sobald es verfügbar ist.	○	○	○	○
Mit dem eigenem Releasezyklus.	○	○	○	○
Wenn Teile, die im System benutzt werden, betroffen sind.	○	○	○	○
Wenn es kritische Fehlerbehebungen (Bug fixes) beinhaltet.	○	○	○	○
Wenn es aufgrund von Abhängigkeiten sein muss.	○	○	○	○
Niemals.	○	○	○	○

Wann migrieren Sie zu einer neuen Version von einem von anderen Projekt-Teams im Unternehmen bereitgestellten Artefakt?

	Nie	Gelegentlich	Meistens	Immer
Sobald es verfügbar ist.	○	○	○	○
Mit dem eigenem Releasezyklus.	○	○	○	○
Wenn Teile, die im System benutzt werden, betroffen sind.	○	○	○	○
Wenn es kritische Fehlerbehebungen (Bug fixes) beinhaltet.	○	○	○	○
Wenn es aufgrund von Abhängigkeiten sein muss.	○	○	○	○
Niemals.	○	○	○	○

Wie schwierig war die letzte Migration, die Sie durchgeführt haben?

- ○ 0 - Nie gemacht.
- ○ 1 - Sehr einfach, es waren nur minimale Änderungen nötig.
- ○ 2 - Akzeptabler Aufwand.
- ○ 3 - Aufwändig.
- ○ 4 - Sehr schwer, die Migration war (fast) unmöglich.

13 Persönliche Einschätzung des aktuellen Stands der Wiederverwendung

Persönliche Einschätzung des aktuellen Stands der Wiederverwendung in [redacted]

Wie weit treffen Ihrer Meinung nach folgende Aussagen zu [redacted] Library zu?

	Trifft nicht zu.			Trifft stark zu.
Die [redacted] ist eine sinnvolle Idee für die Entwicklung unserer Produkte.	○	○	○	○
Die Umsetzung de [redacted] entspricht meinen Erwartungen.	○	○	○	○
Die [redacted] bringt mein Projekt aufgrund von Abhängigkeiten in Schwierigkeiten.	○	○	○	○
Der Funktionsumfang der [redacted] ist zu groß.	○	○	○	○
Die [redacted] erleichtert meine Entwicklungsarbeit.	○	○	○	○
Die Qualität der [redacted] erfüllt meine Erwartungen.	○	○	○	○
Ich bin bereit, für die Weiterentwicklung der [redacted] Ressourcen aufzuwenden.	○	○	○	○
Der Nutzen der [redacted] wiegt die investierten Aufwände auf.	○	○	○	○

Welche Wünsche haben Sie bezüglich der [redacted]

Haben Sie Schwierigkeiten mit der [redacted]

Wie erleben Sie die Arbeit der Spezialistenkreise?

(z.B [redacted]

	Trifft nicht zu.			Trifft stark zu.
Diese Gruppen sind ein Schritt zu einer besseren Abstimmung.	○	○	○	○
Für eine wirkliche Einflussnahme sind diese Gruppen mit zu wenig Ressourcen ausgestattet.	○	○	○	○
Diese Gruppen tragen wesentlich zu einem abteilungsübergreifenden Verständnis der Probleme bei.	○	○	○	○
Die Vernetzung, die über diese Gruppen stattfindet, hat einen positiven Einfluss auf meine Arbeit.	○	○	○	○
Die von den Gruppen betrachteten Probleme sind relevant.	○	○	○	○
Die Spezialistenkreise haben einen zu geringen Einfluss.	○	○	○	○
Die Spezialisten sind in unserem Team gut vernetzt.	○	○	○	○
Über die Spezialistenkreise werden wichtige Themen schneller an die richtigen Leute getragen.	○	○	○	○
Die Arbeit der Spezialisten führt zu einer besseren Wiederverwendungsstrategie.	○	○	○	○

Sind Sie insgesamt zufrieden mit dem gegenwärtigen Umfang der Wiederverwendung bei [redacted]

	... wird zu viel Wiederverwendung betrieben.	... wird Wiederverwendung auf angemessene Weise betrieben.	... wird noch nicht genügend Wiederverwendung betrieben.
Abteilungsübergreifend	○	○	○
In meiner eigenen Abteilung	○	○	○

Sie sind nun am Ende des Fragebogens angekommen. Mit dem Weitergehen auf die nächste Seite können Sie Ihre Antworten abschicken.

Falls Sie uns noch Kommentare zum Fragebogen zukommen lassen möchten, können Sie dies hier tun.

Folgende Aspekte hätte ich noch erwartet/Folgende Rückmeldung zum Fragebogen möchte ich Ihnen noch mitteilen:

14 Endseite

Vielen Dank für Ihre Teilnahme!

Ihre Antworten wurden gespeichert.

Mit Fragen oder Rückmeldungen bezüglich dieses Fragebogens können Sie uns gerne per Email a[redacted] kontaktieren.