An Exploratory Study on Reuse at Google

Veronika Bauer, Jonas Eckhardt, Benedikt Hauptmann
Technische Universität München, Germany

Manuel Klimek
Google, Germany

ABSTRACT

Software reuse is a challenging and multifaceted topic. Significant research effort has been spent to address technical and organizational aspects. However, adoption of proposed practices and novel approaches often proceeds slowly. Additionally, little is known on how reuse is currently effected in practice and which solutions have proven useful.

This paper aims to shed light on the matter by studying the current practice of reuse at Google. We conduct an exploratory study with a total of 49 participants of which 39 answered our online questionnaire and 10 participated in our 1h interviews. We assess reuse practices, success factors and challenges and collect ideas for improvement.

We distill our findings to provide practitioners with examples of scalable reuse practices and detail on prerequisites required to implement/tailor a similar reuse approach. Furthermore, we point out open issues to support researchers and practitioners alike to align their efforts for developing solutions.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable software

General Terms

Human Factors, Measurement, Management

Keywords

Reuse, Practice, Industrial, Case study, Empirical, Software

1. INTRODUCTION

Software reuse, often defined as "the use of existing engineering knowledge and artifacts to build new software systems" [7], is a challenging and multifaceted topic and an active area of research since the late 1960'ies [19]. Significant effort has been spent to address technical issues [14, 20, 8] as well as organizational aspects and barriers to adoption [2, 6, 26]. Benefits [9, 17] as well as failures [10, 28] of reuse adoption in practice have been reported.

However, technological advances have introduced new possibilities and challenges to effect reuse. For instance, enabled by the Internet, code reuse by means of publicly available libraries has become common practice [8, 11]. This change introduces challenges in development and maintenance: firstly, the amount of available functionality is vast and steadily increasing [11]. Retrieving desired functionality for reuse involves considerable cognitive effort [24] and is challenging without tool support. Research has addressed this issue by proposing tools such as code search [1] or recommendation systems to support developers with reuse [25]. However, anecdotal evidence from industrial research projects indicates that these current research efforts have not yet significantly impacted the software development industry. Secondly, systems built on external libraries face novel challenges in maintenance, entailed by their dependencies and the, usually, ad-hoc way libraries are integrated [4, 23, 15]. However, we as research community have little knowledge on how reuse is currently effected in practice.

Problem Information on how reuse is currently effected in practice is scarce. As a result, practitioners aiming to improve reuse in their organizations face difficulties in selecting reuse practices that have been applied successfully. Furthermore, researchers receive little feedback on their work from industry.

Contribution We study the current practice of reuse at Google by means of an exploratory study consisting of an online questionnaire with 39 participants and interviews with 10 participants. We report on the state of practice of reuse, comprising success factors, challenges and ideas for improvement. Based on our results, we provide a list of considerations for implementing reuse, prerequisites and open issues.

Outline Section 2 presents related work, Section 3 describes our study and Section 4 summarizes our results. In Sections 5 we discuss our findings and detail on threats to validity in Section 6. Section 7 lists our considerations for practitioners. Section 8 concludes the paper.

2. RELATED WORK

There is a number of quantitative empirical studies on reuse in practice, summarized by [21], that focus on specific aspects of reuse. Since we aim to capture the big picture of reuse in a running software development environment, we do not detail on these studies. The following paragraphs give

SERIPs '14, June 1, 2014, Hyderabad, India

an overview of qualitative industrial studies related to our research goal.

Studies on reuse adoption Joos [12] reports on the experience of introducing a systematic reuse process at Motorola in the 1990s. She describes the encountered issues during adoption as well as the solutions (management support, education of engineers, suitable incentives, tool support) leading to success. Our findings confirm the mentioned solutions as success factors.

Fichman and Kemerer [6] examine the extent to which the introduction of a formal and systematic reuse program has succeeded in one large organization. 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 study built on semi-structured interviews with 15 teams. Based on their findings, the authors propose strategies for implementing incentive-compatible systematic reuse programs including effective management of the cost of reuse failure and introducing a reuse curator model. Our study confirms the ad-hoc stance on reuse. However, we find a working strategy realizing significant inter-project reuse, enabled by a solid infrastructure. Our results confirm conflicts between project goals and reuse efforts.

Sherif and Vinze [26] report on barriers to reuse adoption. They identify four stakeholders involved in reuse: the reuse expert, asset creator, asset utilizer, and IT manager. In 32 structured interviews they identify barriers on organizational and individual levels. They conclude that individual declination towards reuse was caused mainly by the organizational stance on reuse adoption. Our study identifies similar stakeholders and confirms a potential influence of organizational culture on individuals' reuse behavior.

Lynex and Layzell [18] assess the management and organizational issues raised by the introduction of reuse programs in industry. They collect inhibitors to adoption gained from experiences from reuse projects reported in the literature, provide reasoning for causes and present possible solutions. Our study design is based on aspects of reuse from the literature. However, we analyze inhibitors and mitigations from software development at a company.

Studies on reuse practices Slyngstad et al. [27] conduct an empirical survey with 16 developers of the IT department of a Norwegian oil company and answer five questions on the perceived benefits of reuse (lower cost, faster development, higher quality, standardized architecture), reuse facilitators (existence of component information repositories, education and experience do not matter), potential rework caused by reuse (no connection found), the information available on reusable components (documentation could be improved) and the degree of trust of developers concerning the quality specification of components (medium to low). The authors presented the participants with a questionnaire followed by 10–15 minutes of interview. The goal of our study is to explore a complete picture of the state of the practice in a large company focusing on software development. We therefore collected information on a wider range of topics.

Land et al. [16] present an industrial survey on reuse with software components. Based on 63 valid responses to a questionnaire, the authors conclude that many potential benefits of component-based reuse are not yet realized. However, they also report on successful implementations of this reuse approach. Our study did not focus on a specific reuse approach, but aimed to gain an overview on existent practices.

Studies on reuse success factors and hindrances

Frakes and Fox [7] present a reuse failure mode model, derived from a questionnaire on (code) reuse answered by 113 people from 29 organizations working in different domains. The authors mention four dimensions (managerial, economic, legal, technical) that need to be addressed when implementing systematic reuse. The authors derive seven failure modes (no attempt to reuse, part does not exist, part not available, part not found, part not understood, part not valid, part not integrable). According to the authors, the most frequent failure modes are "no attempt", "not integrable", "not understood", while "not found" or "not available" are the least important. Our results cover most failure modes. However, the ranking differs, with "not found" as the hardest problem.

Morisio et al. [22] report on success factors for adopting or running company-wide reuse programs. They collected evidence from 24 reuse projects in European companies varying in size, business domain and culture. The authors conclude that success of reuse projects depend on management commitment, awareness of human factors and modification of non-reuse processes according to the specific context of the company. Our study confirms that these factors remain important after the adoption of reuse. It is a challenge for management to follow up on the implementation of these factors to prevent a decay of the reuse culture.

3. STUDY SETUP

This section describes context and goal as well as methodology and data collection procedures of our study.

3.1 Context & Subjects

We performed our study at Google, an multinational corporation, specialised on Internet-related services and products. The company is distributed over the globe with 70 offices in more than 40 countries and with more than 40,000 employees[1]. Google is known for its ambitions to continuously improve its craft in terms of quality and efficiency[2]. Therefore, we consider Google a suitable candidate to evaluate reuse in practice.

Our participants at Google work on multiple projects, are organized in small teams, and develop software with several programming languages (mainly C++, Java, Python, and JavaScript). They were drawn from more than 25 different teams, held varying organizational roles (Tables 1 and 2), and had an experience between less than one year and 20+ years in their current role. Their time spent at Google ranged from less than one year to 10+ years.

3.2 Study Goal & Research Questions

The goal of this study is to analyze *reuse practices* for the purpose of *characterization and understanding* with respect to their *effectiveness* from the viewpoint of *software devel-*

[1]Source: Google 2013 Financial Tables, 2013 Q2, Google Stand-Alone Headcount, `http://investor.google.com/financial/tables.html`, accessed 2013/10/09

[2]For details of the software development process at Google see `http://google-engtools.blogspot.de` and `http://goo.gl/4Kl2na`

Table 1: Number of questionnaire participants grouped by their organizational roles

Technical Lead	Developer	Maintainer	Manager
2	21	2	1
	6		
1			
6			

Table 2: Number of interview participants grouped by their organizational roles

Technical Lead	Developer	Maintainer	Manager
	2		1
7			

opment professionals in the context of *an industrial software development company.*[3]

We break down the study goal to the following subgoals: *understanding current reuse practices* and *identifying challenges and opportunities occurring in practice.* We address our subgoals by the following research questions.

3.2.1 Understanding current reuse practices

This category gives insights in the current state of reuse practices at Google.

RQ1: Which roles are involved in reuse practices? We investigate which roles are involved in reuse, as well as their relationships, responsibilities, and motivation for performing reuse.

RQ2: What reuse practices are applied and how often are they used? We assess which (and to what extent) reuse practices and reuse activities are effected and how they are supported by tools and infrastructure.

RQ3: Which measures are taken to assess the adequacy of reuse? We analyze which measures exist to assess adequacy of reuse as well as the strategies for reuse improvement and how they are applied.

3.2.2 Identifying challenges and opportunities occurring in practice

This category identifies problems, challenges, and opportunities that are relevant in practice. The results of these research questions indicate practical solutions and open issues.

RQ4: What are problems, challenges, success factors? We investigate which problems and challenges occur in practice and elaborate the benefits of and the success factors for reuse.

RQ5: What do engineers consider as potential improvement? We collect ideas for improvements of reuse practices from the viewpoint of the engineers. These might not (yet) be realizable or feasible but provide a ranking of the research directions in terms of relevance.

3.3 Methodology

Our sources of research evidence are the two complementary parts of survey research: interviews and questionnaires. In an informal pre-study, we collected a range of aspects, such as *reuse management*, *legal aspects*, *sources and types of reusable artifacts*, *reuse strategies*, *reuse adequacy* or *extent of reuse*, from the literature. These aspects serve as the basis for the questionnaire and the interview guideline.

[3]Following the GQM goal template [3].

Online Questionnaire To gain a comprehensive overview of reuse at Google, we developed an online questionnaire [4]. For each reuse aspect, several multiple choice questions were asked. Furthermore, we invited the participants to contribute additional information. At the end of the questionnaire, we asked the participants to provide their level of experience in their current role, the time spent working at Google and the type of project they were working on. The questionnaire contained 43 questions. Taking part in the questionnaire took approximately 20–30 minutes. Participation was optional.

We randomly selected 600 candidates for participation, based on an internal directory of all software engineers at Google and sent the invitations via email. 39 of them took part in the questionnaire. Our sample included engineers from Google offices around the world, covering all types of projects and technologies.

The responses of the online survey were analyzed with descriptive statistics and visualizations and complement the results obtained in the interviews.

Semi-Structured Interviews We conducted semi-structured interviews with Google engineers. Our interview guideline is based on our pre-study and the scope is similar to the questionnaire. However, 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 10 participants with different responsibilities with respect to reuse from 9 different development teams. Each interview lasted one hour and was conducted by two researchers, one leading the conversation with the participant while the other created the transcript and asked clarification questions.

3.4 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[5] twice: first, we went through a phase of initial coding [5] 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 to group them. In another, focused, round of coding [5], we pruned the categories to the most significant ones and created relationships between them. The coding process resulted in clusters of categories connected with each other, containing the relevant statements.

4. STUDY RESULTS

To answer the research questions, we extract our information from the questionnaire and the interview data. We refer to outcomes of software development activities as artefacts. Artefacts provided or available for reuse are called reusables.

Reuse activities: During coding, we identified a set of activities related to reuse and use them to structure the results of RQs 2 and 4. The activities are: publish - reasons to create and publish reusables, find - where and how

[4]Link to the questionnaire: `http://goo.gl/R3fZPS`.

[5]Coding means "categorising segments of data with a short name that simultaneously summarises and accounts for each piece of data". [5]

to find reusables, understand - means to properly understand reusables, select - selecting the right reusable, adapt - adapting the reusable, integrate - technically integrating the reusable. These activities encompass and extend the ones proposed by Karlsson [13].

RQ1: Which roles are involved in reuse practices?

From the interviews, we identified the following organizational and processual roles involved with reuse at Google.

Organizational roles.

Engineer Engineers are responsible for software development and maintenance at Google. Their motivation for reuse depends on the direct benefit they can obtain from it. Benefit means, for example, faster completion of features, visibility and impact of work, expressed for instance by a high amount of users for a reusable. Senior Engineers are asked to share their experience and advise (e.g. in library design discussions). Engineers are directly involved into reuse: they can introduce libraries, invest time in reusability, propose features to libraries, and share their own artifacts.

Manager, Technical Lead As far as reuse is concerned, the managing roles have several responsibilities: firstly, they can decide to push or suppress reuse as they see fit. Secondly, they serve as coordinating role within and between teams to ensure that reuse options are identified on a larger level, e.g. on a feature granularity. In this way, reuse serves as a means to achieve consistency over the range of products. Thirdly, managing roles are responsible to mandate certain reuse decisions on different scales from teams to product areas. Lastly, the managing roles are also responsible to ensure legal compliance of reuse.

Team A development team at Google usually consists of about 5 to 6 engineers, working on focussed tasks. They are coordinated by technical and product managers. The motivation to reuse is strongly dependent on the individuals in the team.

Processual roles.

Producer Every engineer takes the role of a producer internally as (nearly) all code is available to other internal projects, (about 41% of the participants share artifacts also externally, i.e. Open Source). Typically, the producers or current implementers of a piece of code have ownership over it and are contacted by engineers desiring a change. Ownership is lived as responsibility for the artefact, allowing for others to effect changes.

Apart from the default producer, there are designated producers, which provide central libraries and components, oftentimes taking the role of reuse champions, motivating and pushing reuse strategies at different scales. Their responsibilities and motivation differ as follows: producers serve as the gatekeeper to the library or component, ensuring that only new and valuable functionality is adopted. Furthermore, they are responsible to upgrade all consumers if they effect changes in the artefact. In order to ensure adoption of a reusable by the consumers, producers attempt to perceive common needs as early as possible, e.g. from usage patterns or legacy code. Furthermore, producers are lobbying the use of core libraries and components to improve the quality in the code base (since "introducing new stuff causes problems").

Consumer As a consumer, the engineer is responsible to integrate the reusable. If integration requires changes to someone else's code, the consumer is responsible for contacting the code owner to find a solution. If the integration breaks other people's code, the engineer is responsible to either fix the problem or to roll back the change within a fixed tolerance window.

Reviewer Tightly integrated with the software development cycle at Google is the role of the reviewer, which is again taken by each engineer. Every piece of code submitted to the central code repository is subject to a review cycle. It is the reviewer's responsibility to assess the quality of a proposed solution and to suggest better options, if available. In the context of reuse, this implies proposing reusables and assessing the adequacy of a reuse action.

Legal The legal department is the main point of reference with respect to licensing issues, induced by third-party artifact reuse. Also, sharing artifacts with the Open Source community needs to be confirmed.

Third-party Third-party describes all entities external to Google, whose artifacts are reused by Google in its software development. Most of the time, these are Open Source projects providing libraries for specific problems.

RQ2: What reuse practices are applied and how often are they used?

We follow a top-down structure to report the results of this research question: We first report processual results, then results concerning the individual reuse activities (*Publish, Find, Understand, Select, Adapt, Integrate*), and finally results that concern the reusable itself.

Reuse practices and processes.

In our study, we found that, apart from a set of regulations regarding third party reuse, at Google their is no centrally controlled mandate for organizing reuse. Reuse processes and strategies can be initiated by engineers as well as managers. According to the questionnaire, reuse is generally more ad-hoc [6] (47%) than strategical[7] (37%). Comparing different types of development goals, reuse is more strategical for product development (24%) than for prototype development (13%) or tool development (13%). The interviews revealed that it is up to the engineers and managers to decide how much effort to invest into reuse. The scope of effort ranges from individuals investing their 20%-time[8] for reuse, to entire teams dedicating 30–40% of their time to it. Mostly, reuse takes place on demand: teams focus on completing features, and refactor for reusability when the need for reuse occurs. Furthermore, except for important artefacts, reuse knowledge is often in peoples' heads. The exception to this general philosophy are the teams providing the utility libraries for the entire company. These *core libraries* and components are reused by the entire company.

[6]From the questionnaire: "Ad-hoc reuse means that developers are allowed to reuse any available artefact which seems suitable for the tasks at hand."

[7]From the questionnaire: "Strategical reuse implies that reuse is driven by specific organizational goals. Usually guidelines or policies describe which reuse is adequate for a given situation."

[8]At the time of the study, Google encouraged their employees to spend about 20 percent of their time experimenting with their own ideas.

Therefore, they are carefully designed by experienced engineers, and the scope of their functionality is guarded by the respective team. New functionality can be proposed by everyone but is only adopted if it provides significant value to the company and is not yet contained in similar form in the library. In particular, one core team uses the concept of an incubator, a dedicated place where engineers can place potential reusables. Over the time, their usage is monitored and if the reusable obtains a sufficient usage, the team will invest in improving its reusability and, over different maturity stages, incorporate it into the library.

As far as third party libraries and frameworks are concerned, there is no central coordination for their selection. However, if multiple options for large libraries exist, the core teams decide on one option to ensure homogenous use. The policies for introducing a third party library are as follows: the engineer introducing a library is responsible to ensure that no library for the same functionality is imported yet and to ensure with the legal department that the license is compatible. If these prerequisites are met, the engineer is responsible for integrating the library in a dedicated part of the code base, ensuring it is ready to use and keeping it updated. Third party library usage is governed by several rules, e.g. the libraries are only allowed to be integrated by linking; copy and paste from third party code is forbidden. Third-party reusables are frequently used in projects: more than half of the participants in the questionnaire have introduced at least one or two, with 10% having introduced more than five[9].

Reuse Activities.

Publish

The interviews indicate that the decision to create and provide a reusable is mainly value-driven. We found two main reasons why a reusable should be created: either there is a common need for a specific functionality or the functionality itself provides substantial value (even if it is not commonly needed). However, there is no explicit process for creating and publishing reusables; the decision whether to create a reusable is up to the developer/team. Usage patterns in legacy code can provide indicators for common needs and serve as basis for creating reusables.

Within Google, most of the code is available for everyone via one central repository: 68% of the participants of the questionnaire stated that all artefacts are available for other projects and 41% stated that only some artefacts are available for other projects. To promote reusables, we found several ways: there are dedicated mailing lists, internal web pages, Google+, newsletters and user guides. Table 3 shows the top three ways of sharing artefacts: via a common repository (97%), via packaged libraries (34%), and via tutorials (31%). These numbers support the above findings.

Find The questionnaire indicates that the most common source for finding reusables are internal repositories (87%), closely followed by colleagues (38%). Furthermore, resources on the web are used to find third party libraries and ideas. External build automation tools were not used.

[9]Introducing here could mean "importing to third party base at Google" as well as "starting to use a library that was imported by another engineer".

[10]Multiple responses allowed.

Table 3: Which are your top-three ways of sharing artefacts?[10]

Answer	#Answers	Percentage
Common repository	31	97%
Packaged libraries	11	34%
Tutorials	10	31%
Blogs	6	19%
Email	3	9%
I do not share artefacts	1	3%
Other	1	3%
None of the above	0	0%

Furthermore, the questionnaire suggests that the most frequently used way to retrieve reusables is the internal code search engine, which provides (read) access to the complete code within the Google main repository (see Table 4). Code search is followed closely by "communicating with colleagues". The interviews detailed on the communication between the engineers: the communication culture is very direct, so asking for advice on what to reuse is happening regularly. Furthermore, the reviewers will suggest reuse if they recognize reimplementations or suboptimal solutions in the submitted code. Lastly, user mailing lists are available for all bigger libraries and are used for questions on reusables. Focussed support for finding reusables, such as code recommenders or code completion are not widely used for finding reusables.

Table 4: Which are your preferred ways to find reusables? Please indicate the top three.[10]

Answer	#Answers	Percentage
Code Search	30	77%
Communicating with colleagues	25	64%
Web search	19	49%
Browsing repositories	16	41%
Browsing documentation	9	23%
Other	3	8%
Code completion	2	5%
Code recommenders	1	3%
Tutorials	1	3%

Understand The interviews indicate that code search provides code snippets, which are used to understand reusables. As most participants of the questionnaire find reusables by code search, this provides a means to properly understand reusables by usage examples. The questionnaire indicates that all sorts of code documentation and tutorials are among the most often used to properly understand a reusable (see Table 5): reviewing interface documentation (72%), searching example usages in blogs and tutorials (64%), reviewing implementations (64%), and reading guidelines (51%).

Table 5: What do you do to properly understand and adequately select reusable artefacts?

Answer	#Answers	Percentage
I review interface documentation	28	72%
I look for example usages on blogs and tutorials	25	64%
I review implementations	25	64%
I read guidelines	20	51%
I explore third-party products	11	28%
Other	4	10%
I participate in trainings for third-party technologies/artefacts	2	5%
Nothing	0	0%

Select The interviews as well as the questionnaire only give some hints for this activity. The interviews indicate that there is a difference between reusables originating from an internal or an external source; Internal reusables are usually preferred, since it is more difficult to reuse external code due to processual and legal issues. Usually, external libraries or code are only used if there is confidence in the library/code and only for specialized tasks. Depending on the project type, the selection criteria for external reuse may vary: there are some projects where the footprint needs to be as low as possible, and there are some projects where run-time performance is most important. In general, one important selection criterion is that the documentation of the reusable needs to be good. This is also supported by the questionnaire (see Table 5), as the 4 top-most answers concern documentation.

Adapt When deciding whether to integrate a reusable, engineers assess the effort and the possibilities of integration. If major changes need to be effected on the reusable, the engineer is required to discuss them with the respective owner before copying or modifying it.

Integrate Usage of software libraries are the most frequent reuse mechanisms employed at Google, followed by frameworks, design patterns, and code scavenging. Component-based development does occur, but significantly less (see Table 6). Usage of software libraries occurs by linking/calling and includes-linking to the Google-internal code base.

Table 6: Which of the following possibilities of reuse do you employ most? Please indicate the top three.

Answer	#Answers	Percentage
Software libraries	32	89%
Software frameworks	19	53%
Design patterns	13	36%
Code scavenging (copy, paste, modify)	12	33%
Component-based development	8	22%
Architecture reuse	5	14%
Product lines	1	3%
Application generators	1	3
None	0	0%
Other	0	0%

Reusables.

The data from the questionnaire implies that in the software development at Google, a variety of development artefacts is being reused (see Table 7) on different levels of granularity, however with a strong preference of libraries (see Table 9) and a focus on general utility. There is also a significant amount of reuse of domain-specific functionality (see Table 8) and on a small granularity, such as classes.

The interviews provided detailed insight on the nature of reusables: the majority of the reused artefacts are provided internally. A lot of them are provided by the different projects and available, yet normally not curated, for reuse. Furthermore, there is a *core* of utility and infrastructure libraries that are reused by entire Google. These libraries are well documented, maintained by dedicated teams and provide among others basic infrastructure for the programming languages used at Google.Open Source libraries also belong to the set of reusables. Their contributions are also curated for maintainability.

Apart from code and other development artefacts, we found that reuse of knowledge/ideas is prevalent. It occurs if problems need to be solved again, but the existing solution is not reusable due to fundamentally differing programming paradigms, for example. In these cases, engineers rely on "tested knowledge" gained from experience or their co-workers.

Furthermore, tools, training documents and examples are reused across the company.

Table 7: Which are the top-three types of artefacts you reuse?

Answer	#Answers	Percentage
Source code	37	97%
Code in binary form	12	32%
Style guides	11	29%
UI Designs	10	26%
Requirement docs. / Use cases	5	13%
Architecture documentation	5	13%
Prototypes	2	5%
Informal design models	2	5%
Own, domain specific design models	2	5%
Semiformal design models (UML)	0	0%
Formal design models	0	0%
Other	0	0%

Table 8: What is the scope of the reused artefacts?

Answer	#Answers	Percentage
Domain-independent general functionality	27	77%
Domain-specific functionality	18	51%
Product-specific functionality	9	26%
Other	0	0%

Table 9: What granularity do the reused entities typically have?

Answer	#Answers	Percentage
complete libraries	31	84%
one or more classes	18	49%
coarse-grained, e.g. entire frameworks	13	35%
fine-grained, e.g. single methods/functions	11	30%
small code sections	8	22%
Other	0	0%

RQ3: Which measures are taken to assess the adequacy of reuse?

According to our questionnaire, the majority of engineers attempt to implement sustainable design decisions (76%) to ensure reusability of their artifacts. This applies especially to the "core" and infrastructure libraries. To improve the code quality and avoid duplicate solutions built by engineers, the library teams inspect legacy code for unusual usage patterns. These serve as sources for requirements of new reusables.

83% of our participants effect unit tests and 71% effect system tests to ensure a certain level of quality of their artifacts. We also found individual solutions to mitigate negative effects of reuse "hacks", such as cloning: one engineer ensures traceablility by adding the original location in the internal directory to copied code to enable traceability in cases of bugs and need for clarification, despite the presence of a system wide clone detection.

Overall, reuse adequacy is not monitored following a structured process. However, to determine the impact of a reusable, the library teams measure the adoption of reusables by counting the number of users. Effected reuse manifests itself by calls to the reusable, which is treated as the current metric for successful reuse. Currently, some engineers are working

on a more elaborated reuse metric. Moreover, to handle the problem of dependency explosion and limit unwanted reuse, build visibility rules and access rules were introduced.

RQ4: What are problems, challenges, success factors?

In the questionnaire and the interviews, we asked the engineers about challenges and benefits they experienced with reuse, as well as factors making reuse beneficial to them.

Issues and Challenges The factor considered most *disruptive to the reuse process* was difficulties in finding artifacts (56%). This ranged before the difficulty of adapting the artefact to project needs (53%) and licensing issues (44%). The "not invented here" phenomenon is listed (34%) on the fourth place. Accessing the artifacts was hardly considered an issue (6%). 13% of the engineers did not experience difficulties disrupting the reuse process within their teams.

Dependency explosion was considered the most severe *issue attributed to reuse* (52%). 39% considered the ripple effects caused by changes in reused artifacts as problematic, while 35% linked reuse to a decrease of code understandability. Loss of control was mentioned by 29%, while 26% did not experienced issues caused by reuse.

73% of the engineers agreed that the *absence of reuse* in their projects led to duplicate implementations. 64% attributed increased development effort to insufficient reuse. Inconsistencies (48%) and high maintenance effort (45%) were also considered as negative consequences.

The interviews revealed more details on issues with respect to the reuse activities, management and philosophy.

Publish One significant challenge for library providers is to create the "right" reusables. To this end, they need to identify common needs before engineers start to create own solutions. The availability of all code for reuse poses a challenge, as also unmaintainable solutions might be reused. Structuring reusables is a difficult challenge when publishing them. The engineers need to find a suitable level of abstraction for the reusable and classify it accordingly so that others can identify it as reuse candidate. Some teams provide infrastructure to address this issue; however, the solution is not widely known yet.

Find The cost of searching for reusables, composed of the time needed to look up and assess candidates as well as the probability to find nothing or to not be able to integrate, is still high. As a result, engineers create their own solutions.

Understand If understanding whether a reuseable fulfills the current need is too difficult, engineers will create their own solutions. This happens especially when reusables are too abstract and thus the engineers cannot understand them anymore.

Adapt Challenges occurring during adaption concern the usability of library interfaces, as well as the overhead required to adapt a reusable. Low usability might cause users to employ libraries in unintended ways. The overhead for adaption is currently underestimated, especially for copy-paste reuse.

Integrate Incompatibilities in programming paradigms currently truncate reuse possibilities at the level of ideas. Another issue is bad modularization of libraries, which increases the size of the product binaries.

Management Reuse is challenging for management at different levels: despite management support being important, reuse cannot be simply induced by a manager. Especially under time pressure, reuse will not be as beneficial as intended. Operationally, the management of dependencies remains a challenge, as changes run the risk of breaking multiple projects. This is a challenge, as the implementation of reuse must not block anyone in accomplishing their work.

Philosophy A challenge with respect to company philosophy is to strike the right balance with respect to reuse: on the one hand, a lot of people need to be motivated and educated for reuse. On the other hand, excessive reuse should be prevented.

Success factors and benefits According to our questionnaire, the main benefits of reuse experienced by the engineers were: higher development pace (91%), less maintenance effort (69%), and higher code quality (47%). Furthermore, they attributed the availability of new functionality (41%), higher consistency(38%) and regular bug fixes (34%) as beneficial to reuse.

The questionnaire indicates that high quality of reusable artifacts is the most important success factor of reuse at Google (68%). It is followed closely by supporting infrastructure and tools (65%) and adequate abstractions (58%). Homogeneous development culture, as well as dependency management are also considered important (each 32%). Surprisingly, the direct communication culture (19%) or the presence of suitable incentives (6%) were considered as significantly less important. In the interviews, in contrast, both aspects were pointed out by several engineers.

In the interviews, we found an overall sensitivity to code quality with the engineers. This expresses itself through a variety of applied constructive and analytical quality assessment methods, such as "serious review cycles", patterns and guidelines, maturity levels for some components, as well as a solid testing infrastructure. These measures are are partially mandated and employed independently of reuse. However, their presence is seen as precondition of beneficial reuse as they provide confidence in the quality of the reusables as well as a "safety net" for effecting changes.

Engineers stressed that the supporting infrastructure, especially the code search platform, as well as the continuous integration approach and the communication culture were success factors. They particularly enable finding, understanding and integration of reusables. Furthermore, the selection of adequate reuse mechanism as well as a suitable abstraction level are important. Engineers agreed that success factors for reusables, besides overall good quality, were an intuitive usability as well as high stability.

In terms of management, the low organizational overhead for engineers to initiate reuse encourages its adoption. In contrast, the library providers named thorough planning, involving senior engineers, and a strict error handling to ensure high quality of the reusables as one of their main success factors.

With respect to the company culture, the engineers saw the networking and communication attitude as a success factor. In particular, they mentioned the 20%-time, read access to the codebase, open criticism culture, "dog fooding"[11], and extensive profiling of products to continuously improve performance.

[11]"Dog fooding" refers to using own products in one's daily work, thus finding problems immediately.

RQ5: What do engineers consider as potential improvement?

In the interviews and the questionnaire, we found wishes for improvement in three categories: culture, technical support, and methods.

Culture Despite reuse being an established tool in development, engineers wish for more reuse spirit with managers and fellow engineers. Furthermore, they wish for a dual development strategy, interleaving feature production phases with phases focussing on code quality.

Technical support Most of the wishes for potential improvement of technical infrastructure/support address the three reuse activities *Publish, Find,* and *Adapt*:

Publish Participants in the interview wished for a tool that automatically makes code reusable. Furthermore, they wished for a shared place where common utility functions can be published. The questionnaire indicates that the architecture of reusables or libraries should be improved to meet reuse needs: 41% of the participants of the questionnaire answered to the question "In your opinion, what would be the three most important actions to make reuse beneficial in your company" that reusables need to be bundled more coherently in terms of functionality and 38% that libraries should be split to provide more specific functionality.

Find Most wishes expressed in the interviews concern finding the *right* reusable: They wish for natural language queries for code search, a better keywording mechanism, a better discovery tool, and a pattern and components catalogue that contains reusables. Furthermore, an oracle, that looks at a piece of code and tells whether it already exists is wished for. The questionnaire indicates that discovery is still an issue, since 45% answered to the question "In your opinion, what would be the three most important actions to make reuse beneficial in your company" that available artefacts shall be listed in a *marketplace* to ease the discoverability of useful functions, and 21% answered that libraries should be merged to ease the discoverability of already implemented functionality.

Adapt In the interviews, the participants said that it should be easier to change code (more than just refactoring) and that protocols should be provided for the usage of functions.

Furthermore, some wishes expressed in the interviews concern the assessment of reuse: Engineers wished for a method to determine how many times specific methods are used in different projects. The questionnaire further indicates that structured rules for dependency management (21%), clear strategic decisions for interface support (28%), and maturity levels for reused artefacts (17%) would make reuse more beneficial.

Method The interviews further indicate that different levels of abstraction is still an issue as the participants wished for homogeneous abstractions and a programming language with an ideal abstraction mechanism.

Remarks

Overall, we noticed a decisively reuse-friendly atmosphere during our interviews. The participating engineers considered it a significant and beneficial part of their development practice. This impression was backed up by a closing question from the questionnaire: 62% of the participants stated that the current state of reuse was "just right". 21% opinionated that there "should be more of it to leverage the full potential". None of the participants wished the effort spent into reuse to be decreased. 18% opted for "other".

5. DISCUSSION

Our study found that a considerable share of reuse in the considered environment is ad-hoc and opportunistic, guided by few strict principles. The major motivational factor for reuse is the short-term reduction in development effort for new features. Due to the development infrastructure and collective code ownership paradigm, all code is, in principle, reusable by everyone. In essence, much of the reuse occurs "along the way", during the feature-driven evolution of the code base. If code parts evolve that can be used in multiple places, commonly used abstractions are extracted. An exception to this general tendency can be observed for the company's *core libraries*, which provide generic, product-independent functionality for which a common need among the developer community is known to exist or anticipated for the future. The library providers are responsible to manage the evolution of libraries according to consumers' needs, keeping the libraries' structure coherent and providing a certain quality level. This is to a large extent supported by both the shared code base and the open communication culture.

A noteworthy finding of the study is that there is no explicit common notion for what constitutes adequate reuse. Besides legal constraints on the reuse of external artifacts, few commonly mandated rules or guidelines for developers exist. The existing rules are either imposed from the reuse champions, or originate from developers addressing a reuse issue. Apart from these rules, the judgment about the adequacy of reuse, involving factors like the maturity of the reused artifacts and the degree of entangledness between the own code and the reused artifacts, lies in the responsibility of individual teams or developers.

The study shows that despite of decades of research in software reuse, fundamental challenges to integrate reuse seamlessly into the developer workflow remain. The company uses a shared code base and provides a powerful code search environment that allows searching in the entire code base. Nevertheless, the major obstacles to reusing artifacts mentioned by the developers were the identification of suitable reusables and the adaptation of reusables to specific needs. The most challenging issues involved in the creation of reusable artifacts were the a priori identification of commonly needed functionality and the structuring and publishing of reusable artifacts.

Despite the identified challenges, a considerable amount of reuse does occur at Google. The main factors for successful reuse are the high quality of the reusable artifacts, induced by comprehensive code reviews, as well as the development infrastructure (in particular the shared code base and code search) and the open communication culture among the developers. Reviews of code changes identify missed reuse opportunities, allowing the developer to rework the code accordingly, thus increasing the extent of reuse.

On the other hand, the study found that while the open development culture and shared code base at Google undoubtedly fosters reuse of code, it also poses a significant risk: Immature code may be reused inadvertently, resulting in quality and maintenance problems. This risk is mitigated by the obligatory code reviews and build visibility rules.

As most prevalent open issues for effective reuse the study identified the improvement of tool support for creating and

finding reusable artifacts. The participants suggested tooling for an automated publishing of code as a reusable artifact. Ideas for the improvement of search capabilities included natural language queries for code, a component catalogue and a tool that can identify functionally similar code given a code snippet.

6. THREATS TO VALIDITY

Internal Validity

Self-selection bias: Most participants displayed a favorable attitude towards reuse. Since participation in our study was optional, it is possible that only engineers considering reuse as beneficial volunteered to take part. The tendency of the answers seems to confirm this bias.

Selection of participants: The participants of the interviews were sampled by convenience through personal contact in just one company. This might have introduced a bias. To mitigate, we sampled the participants from different teams and different roles.

The participants of the questionnaire were sampled in an automated way. Therefore, we could not avoid including absent engineers in our sample. No overlap occurred between interview and questionnaire participants. In this way, we obtained the viewpoint of 49 different people within the organization.

External Validity

Sample of participants: We sampled all our study participants from Google. We are aware that this greatly impacts the generalizability of our results. However, since we aim to report on applied and scalable reuse practices, we still consider our results as valid.

Although the response rate to our questionnaire was low, we consider the answers as valuable since the set of participants reflected the distribution of teams and products within Google.

Construct Validity

Limitations of research methods: To compensate the limitations of our research methods, we employed multiple methods to collect the data, namely interviews and questionnaires.

Interpretation of the interviews: To ensure the correct selection and categorization of the statements, the interviews were always conducted by two researchers to ensure the correct understanding of the information. The coding and triaging of the data was always performed in discussion by three researchers.

7. CONSIDERATIONS FOR PRACTITIONERS

Our results indicate that certain reuse practices can be applied beneficially at a large scale. This section distills these practices and includes some remarks on the prerequisites to implement them. Furthermore, it highlights open issues.

Considerations.

Quality matters. Our results provide the following insights to practitioners: Most importantly, the *quality of reusables* is crucial for motivating developers to reuse them. Alongside this concern follows the *quality of the documentation* available for the reusables. This enables developers to quickly establish whether the reusable meets their needs. A prerequisite for achieving the desired high quality is a quality-conscious mindset with engineers and managers, which translates into investments in constructive and analytical quality measures, e.g. reviews, continuous quality assessment and thorough testing.

Invest in infrastructure and automation. Effective reuse requires a suitable *supporting infrastructure*. This impacts especially finding, integrating, and publishing reusables. Automated indexing of the code base, a powerful search engine, as well as the possibility to link to reusables provide the basis for efficient reuse.

Control organizational overhead. *Low organizational hurdles* together with *suitable incentives* make it easier for developers to initiate reuse, as it decreases the additional effort for reuse. Nevertheless, a *systematic strategy* and process for the central reusables is important to strike the balance between including new functionality and keeping the reusable maintainable and usable.

Culture matters, too. Reuse requires *trust* between the different parties involved. We found that an *open communication and criticism culture*, supported by reviews and design discussions, help to establish the necessary confidence.

Open issues.

In our study, the engineers pointed out issues that would further improve reuse: Producers need support to *uncover required functionality* as early as possible, (ideally before developers create their own solutions) to avoid multiple implementation of the same functionality. Despite the presence of a powerful search engine, it is still hard to select the best candidate from a huge list of options. Support for *natural language queries* was seen as one option to alleviate this issue. For frequently used reusables, even minor changes might impact all consumers and entail major rework effort. To reduce the burden of modifying reusables, there should be a feasible and scalable way to automatically *effect and propagate changes*. The effort spent on reuse might not pay off immediately but will amortize after some time. Therefore, there should be *adequate incentives* for developers and managers to do tasks from which they do not benefit directly but are necessary to leverage reuse properly. Reuse entails a trade-off between functionality that is easily accessible and loss of control. If employed in an inadequate way, reuse can become a risk to projects. Therefore, it is important to devise structured approaches, meaningful metrics and tools to *assess the adequacy of reuse*.

8. CONCLUSION & FUTURE WORK

We performed an exploratory study on reuse in practice with the goal to provide practitioners with examples of scalable reuse practices.

To this end, we interviewed 10 engineers at Google and collected the opinion of 39 engineers via a comprehensive online questionnaire. By means of qualitative data analysis, we extracted the context of reuse at Google, involving roles and responsibilities, as well as reuse practices, reused artifacts and reuse mechanisms. We furthermore collected the current issues as well as success factors and ideas for improvement. Reuse at Google is performed in an ad-hoc manner on an inter-project scale with the goal of decreasing development time for features. There is no structured approach to assess the adequacy of reuse. The main unit

of reuse is code, which is integrated in a library style. This is enabled by a capable support infrastructure that turns the largest part of the code base into a searchable reuse repository. Furthermore, pervasive automated testing increases the confidence of modifying reusables and ensures that code incorporating reusables behaves as expected. The quality of the reusables, the supporting infrastructure, as well as the communication culture are seen as clear success factors. The biggest challenges to reuse are finding the right reusables from the vast amount of functionality, adapting the reusables to meet current needs, and licensing issues. For the future, the participants wished for more reuse commitment within the company, as well as better support for finding and adapting suitable reusables, identifying the best candidates for new reusables, as well as assessing the reuse effort.

Based on our results we invite practitioners to invest in the quality of reusable artefacts and in reuse supporting infrastructure. Furthermore, organizational overhead should be kept low. Lastly, a culture of openness and trust seems to support reuse.

Research addresses current open issues of the software industry. However, the low adoption poses questions of usability and scalability. As future work, we propose to collect detailed accounts of research results potentially applicable to the challenges pointed out in our study. Our results reflect the reuse practices at Google. While they provide a detailed account of success factors and challenges, data from further companies is needed to assure generalizability.

9. ACKNOWLEDGMENTS

Many thanks to: all participants and supporters at Google, Alejandra Rodriguez for support with the analyses, Lars Heinemann, Florian Deißenböck, Antonio Vetro‘, and Andreas Vogelsang for their helpful input. Parts of this work were funded by the Federal Ministry of Education and Research, Germany (BMBF). Project code Software Campus (TU München), grant number 01IS12057.

10. REFERENCES

[1] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. Sourcerer: a search engine for open source code supporting structure-based search. In *OOPSLA '06*.

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

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

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

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

[6] R. G. Fichman and C. F. Kemerer. Incentive compatibility and systematic software reuse. *The Journal of Systems and Software*, 2001.

[7] W. Frakes and C. J. Fox. Quality improvement using a software reuse failure modes model. *Software Engineering, IEEE Transactions on*, 1996.

[8] W. Frakes and K. Kang. Software reuse research: Status and future. In *IEEE Transactions on Software Engineering*, 2005.

[9] W. Frakes and G. Succi. An industrial study of reuse, quality, and productivity. In *Journal of Systems and Software*, 2001.

[10] R. L. Glass. Reuse: What's wrong with this picture? *IEEE Software*, 1998.

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

[12] R. Joos. Software reuse at motorola. In *IEEE Software*, 1994.

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

[14] Y. Kim and E. A. Stohr. Software reuse: Issues and research directions. In *HICSS 1992*.

[15] B. Klatt, Z. Durdik, H. Koziolek, K. Krogmann, J. Stammel, and R. Weiss. Identify impacts of evolving third party components on long-living software systems. In *CSMR'12*.

[16] 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.

[17] W. Lim. Effects of reuse on quality, productivity, and economics. *IEEE Software*, 2002.

[18] A. Lynex and P. J. Layzell. Organisational considerations for software reuse. *Annals of Software Engineering*, 1998.

[19] M. McILROY. Mass produced soptware components. In *NATO SE Conference Report*, 1968.

[20] H. Mili, F. Mili, and A. Mili. Reusing software: Issues and research directions. In *IEEE Transactions on Software Engineering*, 1995.

[21] P. Mohagheghi and R. Conradi. Quality, productivity and economic benefits of software reuse: a review of industrial studies. *Empirical Software Engineering*, 2007.

[22] M. Morisio, M. Ezran, and C. Tully. Success and failure factors in software reuse. *Software Engineering, IEEE Transactions on*, 2002.

[23] S. Raemaekers, A. van Deursen, and J. Visser. Exploring Risks in the Usage of Third-Party Libraries. In *SQM 2012*.

[24] M. P. Robillard and R. Deline. A field study of API learning obstacles. *Empirical Software Engineering*, 2011.

[25] M. P. Robillard, R. Walker, and T. Zimmermann. Recommendation systems for software engineering. *Software, IEEE*, 2010.

[26] K. Sherif and A. Vinze. Barriers to adoption of software reuse. a qualitative study. *Information and Management*, 2003.

[27] 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 *ISESE 2006*.

[28] M. Zand, V. Basili, I. Baxter, M. Griss, E.-A. Karlsson, and D. Perry. Reuse r&d: Gap between theory and practice. In *SSR 1999*.