# Limiting Technical Debt with Maintainability Assurance – An Industry Survey on Used Techniques and Differences with Service- and Microservice-Based Systems

Justus Bogner
University of Applied Sciences Reutlingen, Germany
University of Stuttgart, Germany
justus.bogner@reutlingen-university.de

Jonas Fritzsch
University of Stuttgart, Germany
University of Applied Sciences Reutlingen, Germany
jonas.fritzsch@informatik.uni-stuttgart.de

Stefan Wagner
University of Stuttgart, Germany
stefan.wagner@informatik.uni-stuttgart.de

Alfred Zimmermann
University of Applied Sciences Reutlingen, Germany
alfred.zimmermann@reutlingen-university.de

## ABSTRACT

Maintainability assurance techniques are used to control this quality attribute and limit the accumulation of potentially unknown technical debt. Since the industry state of practice and especially the handling of Service- and Microservice-Based Systems in this regard are not well covered in scientific literature, we created a survey to gather evidence for a) used processes, tools, and metrics in the industry, b) maintainability-related treatment of systems based on service-orientation, and c) influences on developer satisfaction w.r.t. maintainability. 60 software professionals responded to our online questionnaire. The results indicate that using explicit and systematic techniques has benefits for maintainability. The more sophisticated the applied methods the more satisfied participants were with the maintainability of their software while no link to a hindrance in productivity could be established. Other important findings were the absence of architecture-level evolvability control mechanisms as well as a significant neglect of service-oriented particularities for quality assurance. The results suggest that industry has to improve its quality control in these regards to avoid problems with long-living service-based software systems.

## CCS CONCEPTS

• **Software and its engineering** → *Software creation and management*; *Software post-development issues*; **Software evolution**; **Maintaining software**;

## KEYWORDS

Maintainability, software quality control, survey, industry, service-based systems, microservice-based systems

**ACM Reference Format:**
Justus Bogner, Jonas Fritzsch, Stefan Wagner, and Alfred Zimmermann. 2018. Limiting Technical Debt with Maintainability Assurance – An Industry Survey on Used Techniques and Differences with Service- and Microservice-Based Systems. In *TechDebt '18: TechDebt '18: International Conference on Technical Debt , May 27–28, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3194164.3194166

## 1 INTRODUCTION

While it can be a conscious and deliberate decision, introducing technical debt during software development is often unintentional and only discovered later on in a lot of cases [2]. A large part of this unintentional technical debt is linked with *maintainability*, the degree of effectiveness and efficiency with which a software system can be modified to correct, improve, extend, or adapt it [10, 22]. Especially the adaptive and extending part of maintainability (also referred to as *evolvability* [3]) can be easily neglected. To avoid accumulation of technical debt and to control maintainability as a quality attribute during the design, development, and maintenance phase, software developing companies employ numerous techniques (i.e. processes, tools, and metrics), which we refer to as *maintainability assurance* or *maintainability control* in this paper. While a lot of publications covered the state of the art of maintainability assurance in scientific literature (e.g. [4, 6, 13, 15, 20, 21]), very few studies are concerned with what software developing organizations in the industry actually apply in this area, i.e. the state of practice w.r.t. maintainability control. Since this state of practice could be significantly different from what scientific publications suggest, it is important to get an idea of what industry is doing in this regard. This knowledge can be used to highlight gaps and deficiencies in the industry practice, to improve existing techniques, or to design new and more suited ones.

*Service-Based Systems* (SBSs) promised to bring a number of structural benefits to software maintainability, which has been supported by several studies (e.g. [16, 18]). In recent years, an agile, light-weight, and DevOps-focused service-oriented variant called *Microservice-Based Systems* ($\mu$SBSs) gained popularity and is now trying to renew this promise for a second time. While service orientation can bring several benefits, the different levels of abstraction make it also important to revise and adapt design, development, and quality control mechanisms, as for example reported by Voelz and

Goeb [23]. Especially in the context of microservices, technological heterogeneity and decentralization of control can have negative impacts on the maintainability of a composed application, if not treated appropriately.

We therefore created a survey for software professionals to gain insights into their notion of maintainability assurance, more specifically a) their used processes, tools, and metrics, b) potentially different treatment of SBSs and $\mu$SBSs w.r.t. maintainability control, and c) influences on their satisfaction w.r.t. their maintainability-related actions. Note that while we are aware of existing differences between Service-Oriented Architecture (SOA) and Microservices [5], we focused on their similarities in this paper and explicitly chose the term "Service-Based Systems", which is not as connected to enterprise-wide governance, standardization, and centralization as SOA. In the following sections, we present related work done in this area, the scope and design of our survey, the concrete results, as well as a more in-depth discussion and threats to validity. We conclude with a summary and outlook on follow-up research.

## 2 RELATED WORK

While we could not identify publications with a mostly similar focus (the state of practice w.r.t. maintainability assurance), there are several works surveying related topics in the industry. In [8], Gorla et al. analyzed answers of 112 Information System (IS) project managers about system reliability, maintainability, ease of use, usefulness, and relevance to identify impact factors of different categories (individual, organizational, and technological). Although the overall association model did not find any significant link to maintainability, there was one significant association between the factor *responsiveness of IS department* and maintainability. Moreover, the responses confirmed the obvious: frequent changes over a long period of time often decrease maintainability. As a general conclusion, they observed a higher importance of organizational factors in comparison to technical factors for software quality in IS projects.

Poort et al. surveyed 39 architects of a Dutch IT services company and conducted two workshops to align participants with vocabulary and to collect additional qualitative data [19]. Their goal was to identify the importance of non-functional requirements in SW development projects as well as the method for dealing with them and their relation to project success. They reported that availability was seen as the most important quality attribute (QA) on average. Modifiability (as a part of maintainability) however turned out to be the only QA where a significant correlation existed between perceived criticality and project success, i.e. if modifiability was seen as highly business critical in a project, this project tended to be less successful. The authors suggested that modifiability was poorly understood, especially by customers. Moreover, it would get too little attention, because it was harder to quantify, had no immediate effect on end-users, and generally became much more important after the project's end and the start of maintenance.

Yamashita and Moonen conducted a survey with 85 professional software developers about indicators of bad code quality (*code smells*) [24]. They wanted to assess the level of knowledge about code smells, their perceived criticality and the usefulness of code smell related tooling. One third of developers had never heard of code smells and the majority of respondents was only moderately concerned with them. Those who were concerned stated product evolvability, end-product quality, and developer productivity as reasons. Most mentioned code smells were *Duplicated Code*, *Large Class*, and *Long Method*. A majority of respondents wished for better tool support regarding code smells, specifically during software evolution.

In [7], Caracciolo et al. reported about qualitative interviews with 14 software professionals from 6 different organizations followed by a survey with 34 participants. The authors' goal was to identify the usage and definition of software architecture quality requirements and how they are specified and validated. For maintainability, they identified 4 quality requirements: meta-annotations, code quality, dependencies, and naming conventions. Participants most often described quality requirements in one or more text documents, mostly via standard templates enriched with diagrams, but usually without formal specification. If validation was performed, it was mostly done manually on a prototype or via code reviews, seldom automatically. Tool support seemed to be lacking. As an example, they reported that less than 40% used automated tools for code quality control.

To gain insights about the industry practice of refactoring, Leppanen et al. interviewed 12 senior software architects or senior developers in 9 Finnish companies [14]. Participants mostly reported time pressure during past project stages as the primary reason for refactoring later on ("knowingly taking shortcuts"). Furthermore, people would learn new skills and their expertise and familiarity with the domain would grow. Some also named unclear requirements in the beginning as a reason. The authors reported an intuitive knowledge of developers that time spent refactoring paid itself back in the future. However, most had no measurements in place to quantify the need to refactor and the impact of refactoring. Oftentimes, refactoring was done in combination with new features, because otherwise it would be hard to justify to external customers.

There were also some publications with a focus on Service-Based Systems (SBSs). In [23], Voelz and Goeb asked 42 quality experts and executives about their experience with quality control while developing a very large service-oriented software system. Regarding maintainability, participants highlighted the importance of standardization as well as carefully designing service interfaces to prevent client ripple effects of future changes. Moreover, the authors analyzed available quality design documents for the project and compared them with similar documents for non-SBS projects. Their findings showed a large reduction in the number of concrete quality requirements for the SBS project. Only 15 new requirements were introduced. For maintainability, there was a reduction to less than 50% with only 3 new requirements. The authors suggested trust in the positive influence of service orientation (e.g. modularity or reusability) as well as the usage of new frameworks and tools as possible reasons. However, they also reported an increase in requirements for the separately treated category *testing*.

Lastly, Ameller et al. surveyed 56 software professionals about the importance and handling of quality attributes when designing SBSs [1]. Most participants saw QAs and functional requirements as equally important. According to the results, *dependability* was seen as the most important QA for SBSs. Interestingly, maintainability or modifiability were not explicitly mentioned. Similarly

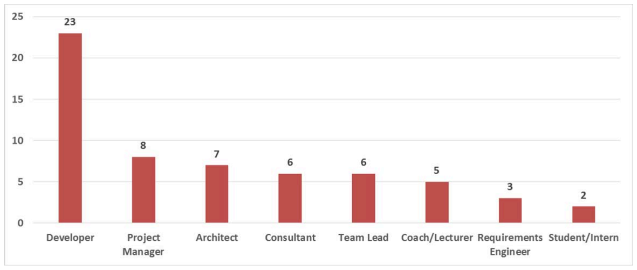

**Figure 1: The roles of the 60 participants**

to [23], the authors assumed as a reason for this that the general architecture of service orientation already comes with a solid base level of maintainability and flexibility.

## 3 SCOPE AND SURVEY DESIGN

A quantitative survey is a method with the research objective to gather information about knowledge, attributes, or behavior of a population, most often via the instrument of a questionnaire [12]. More precisely, we wanted to identify notions and treatment of maintainability control within the target population of software professionals (i.e. what does the industry do to prevent accumulation of unknown maintainability-related technical debt?), as well as potential differences to SBSs and $\mu$SBSs. Furthermore, we wanted to analyze potential correlations and links between applied quality control actions and positive views of maintainability by participants. The second goal is in line with an *exploratory survey*, which investigates existing relations between factors and conditions within a population. [11, 12].

This leads to the following research questions for our survey:

**RQ1:** What processes, tools, and metrics are used in the industry to measure and control maintainability?

**RQ2:** Is the usage of maintainability control different for Service- or Microservice-Based Systems?

**RQ3:** What correlations exist between applied maintainability controls and participants' overall view of maintainability in their project?

To guarantee appropriate scientific rigor, survey repeatability, as well as minimal subjective bias, we followed the seven-stage survey process described in [11]. Information gathering was done via a web-based questionnaire that was first evaluated in a pilot survey. The refined questionnaire was then hosted publicly available for a period of approximately two and a half months. It was distributed via industry contacts, mailing lists, and social media. Participation was voluntary and anonymous. Only completed responses were included and further sanity checks were performed to ensure high quality. Participants were asked to focus on one of their software products or projects for some questions to make it easier for them to give specific answers. The set of questions as well as the results can be found online.[1]

[1]https://github.com/xjreb/research-maintainability-survey

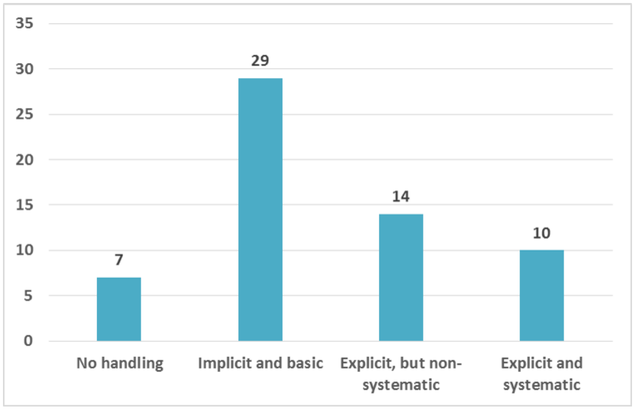

**Figure 2: The 4 levels of sophistication of applied maintainability handling**

## 4 RESULTS

In total, 127 participants with a context of software development responded to our online survey. After filtering out incomplete responses, 60 valid responses were left. The vast majority of these 60 participants were from Software & IT Services companies (70%) followed by automotive companies (10%). Moreover, a lot of participants (32 of 60) worked at large enterprises (≥10.000 employees). Team sizes were generally smaller than 20 people (a combined ~83%) with teams of 6 to 10 being the most frequent size (~38%). There were a bit more participants in technical roles (software developer, software architect, software consultant) than in non-technical ones (36 compared to 24) with the developers being the largest single group (~38%, see Fig. 1). 55% of participants had 5 or more years of experience in the industry while people with 3-5 years of experience were the largest group (22 of 60 participants). To evaluate the experience with regards to SBSs and $\mu$SBSs, participants were also asked about used technologies. ~72% reported to have used some form of agile methodology while exactly half of the participants (30) stated having used DevOps tools or principles. Concerning service orientation, nearly three quarters (~73%) had used Restful HTTP services, ~57% had used SOAP web services while ~38% had even worked with message-based service communication. Besides participant information questions, this population sample was asked questions about maintainability assurance in three categories.

### 4.1 Controlling Maintainability (RQ1)

To assess how maintainability is handled on a general level in their projects, we asked a question concerning the level of sophistication of applied methods. The levels ranged from not addressing it at all, implicit and basic handling (e.g. basic refactoring, some team-wide standardization, etc.), explicit yet non-structured handling (e.g. tool support, metrics, quality model, etc.), and lastly the highest level of using a systematic process (i.e. applying explicit techniques in a structured, communicated, and recurring way). A combined 60% of participants report that they do not address maintainability at all (~12%) or only in very basic fashion (~48%). Only ~17% (10 of 60) use a systematic process (confer Fig. 2).

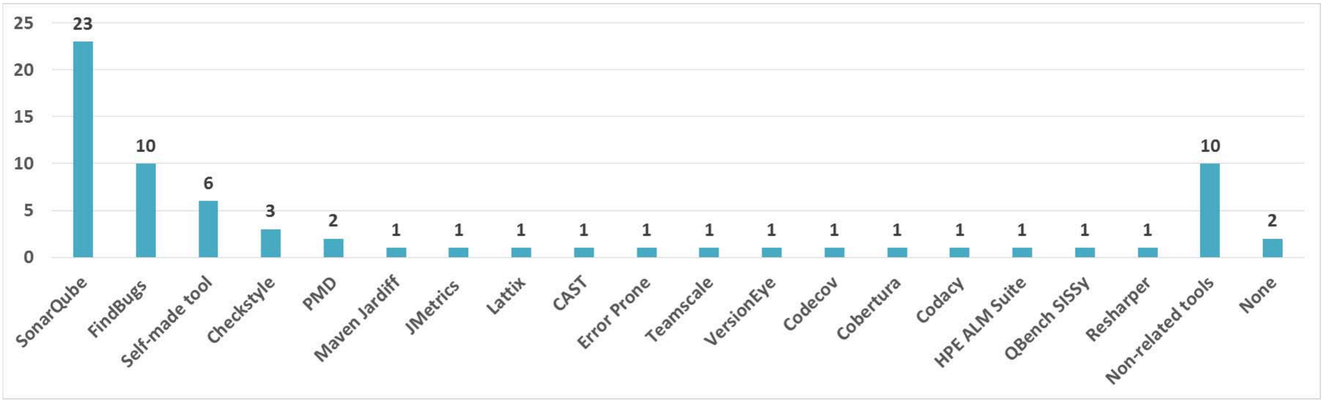

**Figure 3: Used tools to support maintainability control**

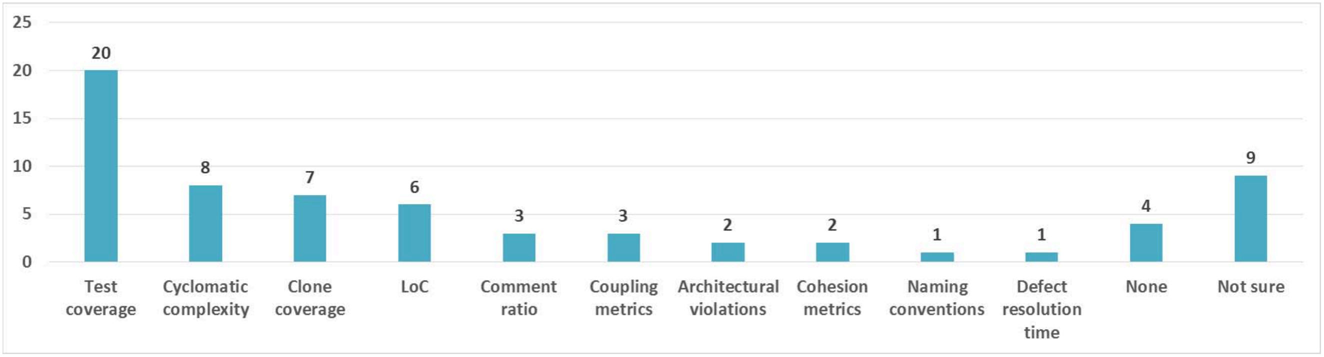

**Figure 4: Used metrics to quantify maintainability**

When asked about the ISO/IEC 25010 standard, the majority of the participants did not hear of or was not familiar with it and its notion of maintainability (more than 85%). This could be seen as an indication that this standard is either not known in industry or not deemed useful enough by team leads and architects to introduce their developers to it.

The free-text question for maintainability tool support was optional and answered by 40 of 60 participants. With a share of over one third, the de-facto standard reported by participants seems to be SonarQube (23 of 60) followed by FindBugs (10) and custom implementations (6). After that, the field gets extremely wide-spread with 14 different tools being mentioned only once (see Fig. 3). A lot of reported tools support only one programming language (e.g. 5 for Java) or focus on a very specific code quality subset (e.g. test coverage). Interestingly, 10 tools were mentioned that are not or only indirectly linked with maintainability, e.g. tools for project management, version control, building, continuous integration, or security analysis. Still, the vast majority of mentioned tools are static source code analyzers or full-blown software quality suites. Although some of the suites can support quality control on an architectural level, none of the participants explicitly mentioned the usage of these capabilities, e.g. architecture conformance checking. Lastly, only 7 of 60 participants (~12%) mentioned the usage of commercial tools.

Similar to the used tools, the free-text question about the most important used maintainability metrics was again optional and answered by 36 of 60 participants (see Fig. 4). Interestingly, 9 people explicitly reported this time that they did not know which metrics were used and 4 that they used no metrics at all (as opposed to 2 people that explicitly stated no tool support). Test coverage is the undisputed metric champion used by one third (20 of 60 participants). After that, cyclomatic complexity (8 of 60) and clone coverage (7 of 60) are followed by code volume metrics based on Lines of Code (6 of 60), e.g. class size or method size. Only 3 people mentioned metrics not related to source code, namely the defect resolution time and the number of architectural violations. While the nature of coupling and cohesion metrics (mentioned by a total of 5) was not defined more clearly and could theoretically be on an architectural level, the majority of answers is still in line with the source code focus of the tool support question, which shows a worrying neglect for architecture-level maintainability/evolvability control. Moreover, the sensibility of using cyclomatic complexity and metrics based on LoC has been a very much discussed topic in recent years. Several publications argue that their values are at best dubious and in the worst case completely useless, e.g. [9] or [17].

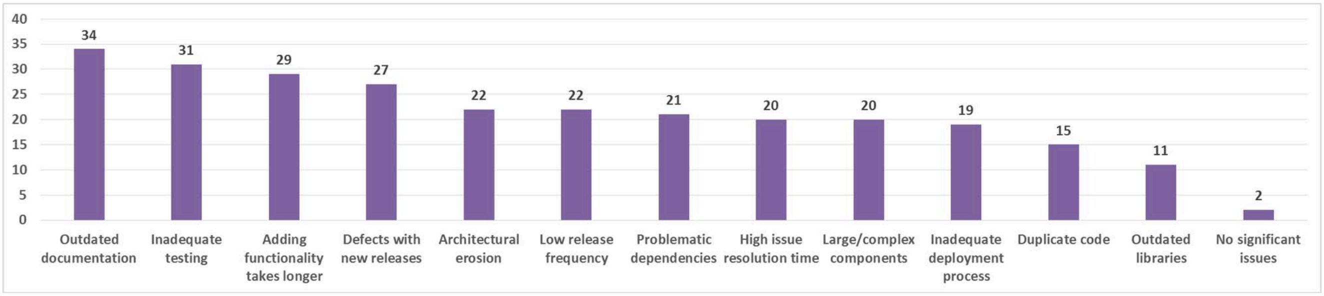

**Figure 5: Reported symptoms of low maintainability**

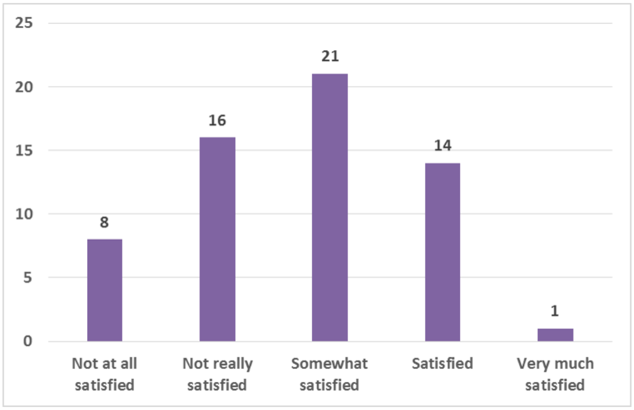

**Figure 6: The participants’ degree of satisfaction w.r.t. maintainability**

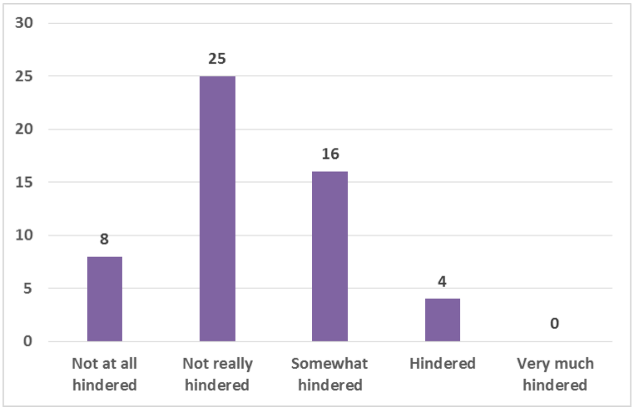

**Figure 8: Self-reported impact on productivity**

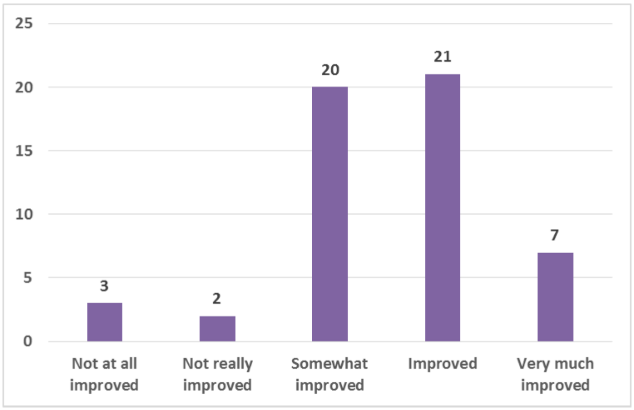

**Figure 7: Self-reported impact on software quality**

## 4.2 Maintainability Satisfaction

The next category addressed the general satisfaction with maintainability as well as the subjective attitude towards applied control actions. In general, 40% of participants were not satisfied with the degree of maintainability of their software, while 35% were somewhat satisfied. Only one fourth of participants (15 of 60) were actually content with the degree of maintainability (confer Fig. 6).

This is further strengthened by the fact that *nearly every participant* mentioned some symptoms of low maintainability with their project (see Fig. 5). 80% of participants (48 of 60) reported 3 or more symptoms, ∼47% even reported 5 or more symptoms. Only ∼3% (2 of 60 participants) said they had no significant issues. The most frequently encountered ones were *missing or outdated documentation* (∼56%) and *inadequate testing* (∼51%). These were followed by 4 symptoms mainly associated with evolvability, namely *more time needed to add new functionality* (∼48%), *high number of defects with new releases* (∼45%), *architectural erosion or complexity* (∼36%), and *low release frequency* (∼36%). Unsurprisingly, there is a medium negative correlation with very high significance the number of reported symptoms and the reported level of satisfaction w.r.t. maintainability (Spearman’s rho: -0.446, p-value: 0.00035).

Participants reported a mixed subjective feeling for the effectiveness and efficiency of the applied methods. ∼53% believed that their maintainability related actions do increase the quality of their software significantly. ∼38% were somewhat satisfied with the effectiveness, while ∼9% believed that quality is not or not really increased by their actions. So a little bit less than half of participants saw no or only little benefits from their maintainability control (Fig. 7). Concerning the efficiency, around 62% thought that these actions do not or not really hinder their productivity in some form. ∼30% felt somewhat hindered and only ∼8% (4 of 60) reported that they truly experience a decrease in productivity (Fig. 8).

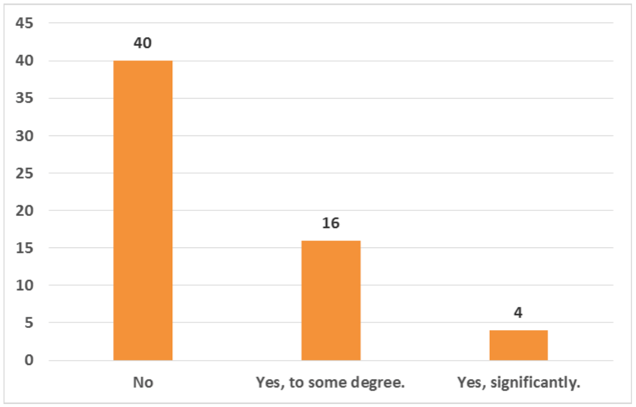

**Figure 9: Are SBSs/$\mu$SBSs treated differently?**

In accordance with that, 3 out of 4 participants thought they should still do more for maintainability (8 of 60 even think "a lot more" is necessary). 15% wanted to keep their efforts the same while 10% wanted to do less. Unsurprisingly, there was a highly significant correlation between changing the current maintainability-related effort and the number of reported symptoms of low maintainability (Spearman's rho: 0.428, p-value: 0.00064), i.e. the more symptoms participants reported the more likely they were to increase current maintainability-related efforts. Interestingly however, there is no correlation with having a technical role in this entire category.

## 4.3 Treatment of SBSs and $\mu$SBSs (RQ2)

The final category was used to investigate whether SBSs and $\mu$SBSs received special treatment by participants or if the same maintainability assurance methods were applied as for systems without service orientation. Furthermore the specifics of potential differences or respectively the reason for not treating them differently were asked. ~67% do not treat maintainability differently for these systems (see Fig 9). ~26% apply a somewhat different strategy and only ~7% (4 of 60) treat these systems in a significantly different way. The differences reported by the 20 participants are mostly in the area of the applied process (55%). Examples are a different testing process (additional integration and behavior-driven testing), the usage of reference architectures, standardization of interfaces, agile methodologies, or a different deployment process. 5 participants mention different tool support (e.g. due to the increased technological heterogeneity) and only 3 report a different metric usage to respect the additional abstraction of services. Interestingly, 6 participants mention relaxed maintainability controls because of trust in the high base level of maintainability gained through service orientation (e.g. flexibility, scalability, less dependency problems, separation of concerns, etc.), which seems to be in line with the findings in [23] and [1].

When the 40 participants that did nothing differently in a service-oriented context were asked for their rationale, over half of them did not know and had never thought about it while an additional ~17% wanted to change something, but didn't exactly know what to do. ~25% regarded their standard approach for maintainability as sufficient. This seems to indicate that a significant amount of software developing companies is not really aware of existing service-oriented approaches and metrics for maintainability or sees no need for special treatment despite their vastly different nature (see Fig. 11).

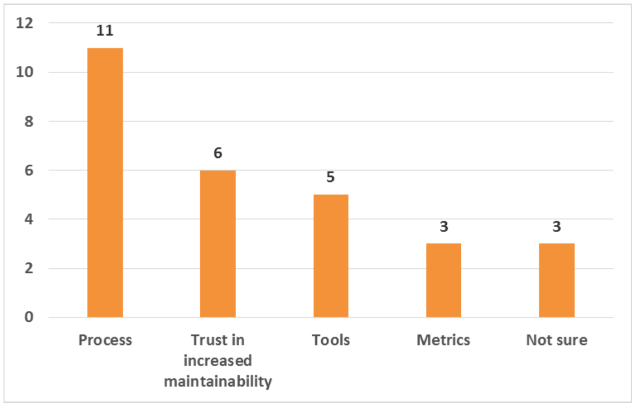

**Figure 10: Categorized differences when treating SBSs/$\mu$SBSs**

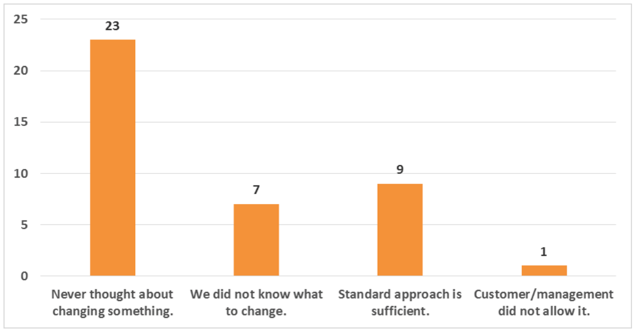

**Figure 11: Reasons for not treating SBSs/$\mu$SBSs differently**

## 5 DISCUSSION (RQ3)

On top of the mere distribution results, we were especially interested in potential statistical links to gain some more insights into positive influences on maintainability. When looking at concrete correlations, we did not find very strong ones, but there are some interesting smaller ones that are also significant when considering the sample size. One focus for correlation is obviously the **level of sophistication w.r.t. addressing maintainability** (values ranging from "not at all" up to using a systematic process). Simply looking at the 10 participants that reported using a systematic process (the highest level of sophistication) revealed that 90% of these believed that this increased software quality at least somewhat and 80% still wanted to invest more effort (with 20% wanting to keep efforts the same). In line with that, of the 40% (24 participants) that are less than somewhat satisfied with the state of maintainability of their software, ~70% address maintainability only implicitly (~50%) or not at all (~20%), which again seems to support the importance of an explicit understanding of maintainability as well as following a more structured approach. However, only 3 of the 10 participants using a systematic process were in technical roles (1 developer, 2 architects). To build on this, when separating participants into two groups (technical and non-technical roles), we found a small but significant negative correlation between having a technical role and the reported level of sophistication (Spearman's rho: -0.273, p-value: 0.035). A possible interpretation could be that non-technical roles are more likely to label their used methods as explicit and systematic, which also would suggest that the actual number of systematic approaches in practice was even lower. However, there is still a small correlation between the general level of sophistication of the applied methods and satisfaction with the overall state of maintainability of the software system (Spearman's rho: 0.259, p-value: 0.046). This supports the general notion of the findings mentioned above, i.e. the more explicit and systematic the applied maintainability controls are, the more likely people are to be satisfied with this QA. Although this effect could in theory be strengthened by a form of cognitive dissonance (investing a lot of effort in something can lead to higher personal affection), it is not very likely. Moreover, we also see a highly significant correlation between the level of sophistication and whether SBSs and $\mu$SBSs are treated with special maintainability assurance mechanisms (Spearman's rho: 0.349, p-value: 0.006). A straight-forward explanation would be that experienced professionals with sophisticated methods are also aware of the necessity to respect the differences introduced by service orientation in this regard.

Another interesting focus for correlations is the degree to which participants believe their maintainability related actions increase **software quality**. There is for instance a small correlation with the degree to which participants feel hindered in productivity by said actions (Spearman's rho: 0.273, p-value: 0.047), i.e. participants that feel they managed to improve software quality are more likely to also feel slightly hindered in productivity by doing so. This suggests that the majority of currently applied methods in this population subset that successfully increase software quality may not be efficient enough. Interestingly however, there is no direct correlation between a hindrance in productivity and the level of sophistication of the applied methods. So the intuitive notion that using a systematic process would have a relation to decreased productivity does not seem to hold in this sample.

Lastly, there is a highly significant correlation between **reported tool usage** and wanting to change the current maintainability-related effort (Spearman's rho: 0.439, p-value: 0.00045), i.e. people that use tools to support maintainability control are more likely to say that they want to invest even more time to increase this QA. A suggested explanation could be that tool support makes it not only easier to identify problems that developers would have otherwise missed, it can also reduce the effort to correct some of them through automation. Interestingly, the same cannot be said for using metrics. While there is a small correlation with investing more effort, its p-value of 0.117 is too large to be seen as significant in this sample. Unsurprisingly however, using metrics is strongly and highly significantly correlated with using tool support (Spearman's rho: 0.667, p-value: 6.04e-09), so there is still an indirect link between the two. A second correlated property to using tools with high significance is treating SBSs and $\mu$SBSs different in comparison to other types of systems (Spearman's rho: 0.372, p-value: 0.0035). Similarly, the same holds true for reporting the use of metrics (Spearman's rho: 0.307, p-value: 0.017). So people that explicitly reported the usage of tools or metrics are more likely to apply a special treatment to systems based on service orientation. This strengthens the already mentioned correlation between the general level of sophistication and treating service-oriented systems differently: The more elaborate the applied processes, tools, and metrics are, the more experienced and skilled the developers usually are with maintainability and therefore, they appropriately treat the different nature of services.

## 6 THREATS TO VALIDITY

We took several precautions to increase the **internal validity** of our results. First, participation was voluntary and anonymous to ensure that answers were as honest as possible. Furthermore, participants were presented with definitions of the key terms to create a common understanding of the main concepts (maintainability, SBSs, etc.). Participants were also asked about their experience with the systems of interest to ensure they had a base level of knowledge (which was confirmed for the majority of them). Incomplete or obvious fake responses were not included into the results evaluation. Lastly, a pilot survey was conducted to fine-tune questions and analyze if the research objectives can really be sufficiently addressed with the results of the questionnaire. With all that being said, we cannot fully exclude the possibility that some participants understood something differently and therefore gave unfitting answers. This limitation can also be seen in combination with the mixture of technical and non-technical roles. As already discussed above, non-technical roles were for example more likely to label their applied maintainability actions as explicit and systematic. Additionally, while we did not find any confounding variables that may blur interpretation, there still remains a small possibility that we missed an important confounder not even present in the answers of our questionnaire.

With regards to **external validity**, we face the same limitations as every quantitative survey: can we generalize our results that were obtained from a limited sample to the complete target population? Due to the very large target population (software professionals world-wide), the discovered distributions in our sample are only of limited generalizability. In comparison to 5 quantitative studies presented in the related work section, only 2 had more than our 60 participants. However, our sample population was also dominated by large Software & IT Services companies with teams of 20 or less and the majority of participants were most certainly from Germany. So we should be careful to give too much attention to the numerical distribution of characteristics. The identified correlations, however, are interesting and valid take-aways, even for such a small sample.

Furthermore, the collected responses to address RQ2 (potentially different maintainability assurance of SBSs/$\mu$SBSs) suggest that a quantitative survey might not be the best instrument to thoroughly investigate such a specialized topic. Very few participants had to provide valuable insights in this area. Even with a larger sample size, the differences and especially the rationale for them are hard to pin down within the constraints of a questionnaire. Qualitative methods like expert interviews or a focus group may be much better ways to assess RQ2, especially for such a recent topic as Microservices. The survey results may provide a valuable starting point for such qualitative follow-up research.

## 7 SUMMARY AND CONCLUSION

Because there is little scientific research on the industry state of practice w.r.t. maintainability control in general and for SBSs and $\mu$SBSs specifically, we created a survey to find out about a) applied processes, tools, and metrics, b) satisfaction with this quality attribute, and c) potentially different treatment of systems based on service orientation. When analyzing the answers of 60 software professionals, we discovered that a combined 60% of our sample do not address maintainability at all or only implicitly at a very basic level. A combined 40% reported explicit handling, but only 10 participants reported using a systematic process. Furthermore, 40% of participants were not satisfied with the state of maintainability of their software. 35% were somewhat satisfied and only 25% reported to be pleased with the maintainability of their system. Lastly, ~67% reported to not treat SBSs and $\mu$SBSs with special maintainability controls. ~26% applied somewhat different controls and only ~7% mentioned significantly different treatment. Since the sample population is limited in size however, further research in this area is necessary to confirm similar distribution in other parts of the population, probably also exclusively with companies that specialize in the development of SBSs and $\mu$SBSs.

Regardless of the sample size though, some interesting points with potential impact for future research can be highlighted. First, there was extremely little reported usage of methods, tools, and metrics that focused specifically on evolvability. While issues related to **architecture-level evolvability** were very present amongst the reported symptoms (e.g. *architectural erosion or complexity* or *more time needed to add new functionality*), very few participants reported actually addressing them. To avoid technical debt in a long-living software system, industry needs to improve their quality control: they need to go beyond source code level metrics and also include an architecture-centric view on software evolution, potentially with scenario-based methods.

A second take-away from this work is the suggested usefulness of using a **systematic process for maintainability control**. While very few reported using such an approach, the vast majority of these 10 participants believed that this increased software quality and even wanted to invest more time for maintainability. People that used tools were also more likely to invest more time. To strengthen this, there was an overall correlation between the level of sophistication w.r.t. maintainability control and the reported satisfaction with this quality attribute: participants that used explicit and systematic approaches were more likely to be satisfied with the state of maintainability of their software. Moreover, no direct link between a hindrance in productivity and the level of sophistication of applied actions could be found.

Lastly, the **neglect of service-oriented particularities for quality control** – especially on the metrics side – combined with the reported trust in the base level maintainability of service orientation could pose a threat for the quality of a long-living SBS. Aspects like service granularity, service cohesion, or service coupling can easily become problematic when not tended to and can significantly hinder the maintenance and evolution of such a system. Luckily, there was a correlation between treating service-oriented systems differently and the level of sophistication of applied actions (on top of using tool support). So there is hope that with a rise in the general proficiency of maintainability control, understanding for a different treatment of SBSs and $\mu$SBSs will grow as well. When looking specifically at the differences with treating Microservices however, there is an even wider gap. Of the participants that reported different treatment, no one mentioned how potential maintainability-related problems with increased technological heterogeneity, decentralization of control, or appropriate service granularity could be addressed. While these topics do not necessarily have to negatively impact maintainability, they still hold the potential to do so, if not executed with some care. Since this is a rather young topic, qualitative interviews with experts in the field of Microservices would most likely be a more effective approach.

## ACKNOWLEDGMENT

This research was partially funded by the Ministry of Science of Baden-Württemberg, Germany, for the Doctoral Program "Services Computing" (http://www.services-computing.de/?lang=en).

We kindly thank Maximilian Jager from the University of Mannheim for his external quality assurance of the used statistical methods.

## REFERENCES

[1] David Ameller, Matthias Galster, Paris Avgeriou, and Xavier Franch. 2016. A survey on quality attributes in service-based systems. *Software Quality Journal* 24, 2 (jun 2016), 271–299. https://doi.org/10.1007/s11219-015-9268-4

[2] Paris Avgeriou, Philippe Kruchten, Ipek Ozkaya, Carolyn Seaman, and Carolyn Seaman. 2016. Managing Technical Debt in Software Engineering. *Dagstuhl Reports* 6, 4 (2016), 110–138. https://doi.org/10.4230/DagRep.6.4.110

[3] Stephan Bode. 2009. On the Role of Evolvability for Architectural Design. *Science* (2009), 3256–3263. http://www.theoinf.tu-ilmenau.de/$\sim$sbode/publications/bode_OnTheRoleOfEvolvability.pdf

[4] Justus Bogner, Stefan Wagner, and Alfred Zimmermann. 2017. Automatically measuring the maintainability of service- and microservice-based systems. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement on - IWSM Mensura '17*. ACM Press, New York, New York, USA, 107–115. https://doi.org/10.1145/3143434.3143443

[5] Justus Bogner, Alfred Zimmermann, and Stefan Wagner. 2018. Analyzing the Relevance of SOA Patterns for Microservice-Based Systems. In *Proceedings of the 10th Central European Workshop on Services and their Composition (ZEUS'18)*. CEUR-WS.org.

[6] Hongyu Pei Breivold, Ivica Crnkovic, and Magnus Larsson. 2012. A systematic review of software architecture evolution research. *Information and Software Technology* 54, 1 (2012), 16–40. https://doi.org/10.1016/j.infsof.2011.06.002

[7] Andrea Caracciolo, Mircea Filip Lungu, and Oscar Nierstrasz. 2014. How do software architects specify and validate quality requirements? *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8627 LNCS (2014), 374–389. https://doi.org/10.1007/978-3-319-09970-5_32

[8] Narasimhaiah Gorla and Shang-Che Lin. 2010. Determinants of software quality: A survey of information systems project managers. *Information and Software Technology* 52, 6 (jun 2010), 602–610. https://doi.org/10.1016/j.infsof.2009.11.012

[9] Ilja Heitlager, Tobias Kuipers, and Joost Visser. 2007. A Practical Model for Measuring Maintainability. In *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*. IEEE, 30–39. https://doi.org/10.1109/QUATIC.2007.8

[10] International Organization For Standardization. 2011. *ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.* Technical Report. 1–25 pages. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35733

[11] Mark Kasunic. 2005. *Designing an Effective Survey.* Technical Report. Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA. 143 pages. http://www.sei.cmu.edu/reports/05hb004.pdf

[12] Barbara A. Kitchenham and Shari L. Pfleeger. 2008. Personal Opinion Surveys. In *Guide to Advanced Empirical Software Engineering*. Springer London, London, 63–92. https://doi.org/10.1007/978-1-84800-044-5_3

[13] Heiko Koziolek. 2011. Sustainability evaluation of software architectures. *Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of software architectures – QoSA and architecting critical systems – ISARCS - QoSA-ISARCS '11* (2011), 3. https://doi.org/10.1145/2000259.2000263

[14] Marko Leppanen, Simo Makinen, Samuel Lahtinen, Outi Sievi-Korte, Antti-Pekka Tuovinen, and Tomi Mannisto. 2015. Refactoring-a Shot in the Dark? *IEEE Software* 32, 6 (nov 2015), 62–70. https://doi.org/10.1109/MS.2015.132

[15] Sajjad Mahmood, Richard Lai, Yong Soo Kim, Ji Hong Kim, Seok Cheon Park, and Hae Suk Oh. 2005. A survey of component based system quality assurance and assessment. *Information and Software Technology* 47, 10 (jul 2005), 693–707. https://doi.org/10.1016/j.infsof.2005.03.007

[16] Yaser I. Mansour and Suleiman H. Mustafa. 2011. Assessing Internal Software Quality Attributes of the Object-Oriented and Service-Oriented Software Development Paradigms: A Comparative Study. *Journal of Software Engineering and Applications* 04, 04 (2011), 244–252. https://doi.org/10.4236/jsea.2011.44027

[17] Jan-Peter Ostberg and Stefan Wagner. 2014. On Automatically Collectable Metrics for Software Maintainability Evaluation. In *2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*. IEEE, 32–37. https://doi.org/10.1109/IWSM.Mensura.2014.19

[18] Mikhail Perepletchikov, Caspar Ryan, and Keith Frampton. 2005. Comparing the impact of service-oriented and object-oriented paradigms on the structural properties of software. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3762 LNCS (2005), 431–441. https://doi.org/10.1007/11575863_63

[19] Eltjo R. Poort, Nick Martens, Inge Van De Weerd, and Hans Van Vliet. 2012. How architects see non-functional requirements: Beware of modifiability. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7195 LNCS (2012), 37–51. https://doi.org/10.1007/978-3-642-28714-5_4

[20] Arun Rai, Haidong Song, and Marvin Troutt. 1998. Software quality assurance: An analytical survey and research prioritization. *Journal of Systems and Software* 40, 1 (jan 1998), 67–83. https://doi.org/10.1016/S0164-1212(97)00146-5

[21] Mehwish Riaz, Emilia Mendes, and Ewan Tempero. 2009. A systematic review of software maintainability prediction and metrics. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, 367–377. https://doi.org/10.1109/ESEM.2009.5314233

[22] David Rowe, John Leaney, and David Lowe. 1998. Defining Systems Architecture Evolvability - a taxonomy of change. *International Conference and Workshop: Engineering of Computer-Based Systems* December (1998), 45–52. https://doi.org/10.1109/ECBS.1998.10027

[23] Dirk Voelz and Andreas Goeb. 2010. What is Different in Quality Management for SOA?. In *2010 14th IEEE International Enterprise Distributed Object Computing Conference*. IEEE, 47–56. https://doi.org/10.1109/EDOC.2010.27

[24] Aiko Yamashita and Leon Moonen. 2013. Do developers care about code smells? An exploratory survey. In *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 242–251. https://doi.org/10.1109/WCRE.2013.6671299