Towards a Practical Maintainability Quality Model for Service-and Microservice-based Systems

Conference Paper · September 2017

DOI: 10.1145/3129790.3129816

3 authors, including:

Justus Bogner
Universität Stuttgart
42 PUBLICATIONS **209** CITATIONS
SEE PROFILE

Alfred Zimmermann
Hochschule Reutlingen
122 PUBLICATIONS **685** CITATIONS
SEE PROFILE

Towards a Practical Maintainability Quality Model for Service- and Microservice-based Systems

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

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

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

ABSTRACT

Although current literature mentions a lot of different metrics related to the maintainability of Service-based Systems (SBSs), there is no comprehensive quality model (QM) with automatic evaluation and practical focus. To fill this gap, we propose a Maintainability Model for Services (MM4S), a layered maintainability QM consisting of Service Properties (SPs) related with automatically collectable Service Metrics (SMs). This research artifact created within an ongoing Design Science Research (DSR) project is the first version ready for detailed evaluation and critical feedback. The goal of MM4S is to serve as a simple and practical tool for basic maintainability estimation and control in the context of SBSs and their specialization Microservice-based Systems (μSBSs).

CCS CONCEPTS

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

KEYWORDS

Maintainability, Quality Model, Metrics, Service-based Systems, SOA, Microservices

ACM Reference format:
Justus Bogner, Stefan Wagner, and Alfred Zimmermann. 2017. Towards a Practical Maintainability Quality Model for Service- and Microservice-based Systems. In *Proceedings of ECSA '17, Canterbury, United Kingdom, September 11–15, 2017,* 4 pages.
https://doi.org/10.1145/3129790.3129816

1 INTRODUCTION

Quickly and efficiently changing software systems is critical for today's organizations. The relevant quality attribute is known as **maintainability**: the degree of effectiveness and efficiency with which a software system can be modified to correct, improve, extend, or adapt it [13, 22]. On top of the positive influence of object orientation (OO), **Service-based Systems (SBSs)** became a very important form of maintainable software [19]. Most recently, service orientation and DevOps culminated in a variation called *Microservices*. A **Microservice-based System (μSBS)** is a specialization of an SBS that consists of very fine-grained services and focuses on decentralization of control, lightweight communication, and technological heterogeneity [17, 30]. μSBSs are usually developed with strong DevOps utilization. Organizations working on such systems will greatly benefit from an explicit understanding of maintainability. This can be supported by a *Quality Model* (QM) with metrics to evaluate and control this quality attribute. Especially in a Microservices context, having a reasonable number of measurements with fast feedback becomes important to ensure high release frequency. While there is a large number of service-specific metrics present in the literature [3], selection is still hard for practitioners. Moreover, characteristics of μSBSs make it difficult to rely on existing object-oriented QMs and impractical to employ very large and complex QMs. To address this problem, we present the following research question that guided our Design Science Research (DSR) project:

RQ: What is a feasible maintainability QM for SBSs and μSBSs with a focus on automatic measurements, simplicity, and practical applicability?

DSR was conceptualized by Hevner et al. as a research paradigm for producing innovative artifacts to solve an existing practical problem [10]. In our case, the business relevant problem of organizations that design, develop, or maintain long-living and frequently changing SBSs or μSBSs is the need for a systematic understanding of maintainability as well as metrics to automatically quantify its degrees. The developed artifact to address this is a maintainability QM.

In [14], Johannesson et al. present a 5-step process to guide DSR projects. In our case (see Fig. 1), the *problem explication* stage consisted of action research in an industry environment combined with a literature review. These results were used in the next stage to *define requirements* in an industry focus group. The research artifact – a maintainability QM – was created in phase 3 (*design and development*). Since this short paper presents research in progress, both the *demonstration* and *evaluation* stage have not been completed yet. It is planned to bring the created artifact back into the environment via case studies, additional action research, and a survey about the

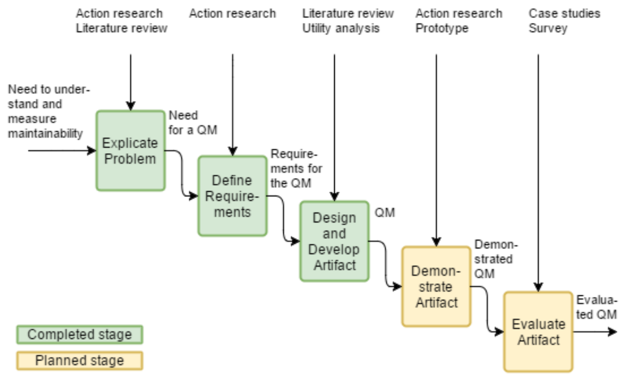

Figure 1: Concrete DSR process based on [14]

"MUST" criteria
- Generally applicable to SBSs
- Related to a selected maintainability SP
- Automatically collectable
- Each SP needs at least one metric

"SHOULD" criteria with weights
- Specifically designed for SBSs (0.2)
- Holistic representation of the SP (0.2)
- Practical applicability to μSBSs (0.15)
- Degree of evaluation (0.15)
- Number of appearances in the literature (0.1)
- Design-time relevance (0.1)
- Applicable to a single service (0.1)

Table 1: Criteria for metric selection

perceived usefulness for practitioners. With these inputs, we plan to modify and improve the artifact further.

2 RELATED WORK

The first QMs with metrics that received attention were designed for OO software, e.g. the QMOOD model [2], which was adapted to different scenarios and paved the way for successors. Since then, industry standards regarding software quality have been created, e.g. ISO/IEC 9126 [12] and its replacement ISO/IEC 25010 [13]. However, these failed to provide simple applicability by omitting concrete metrics. In [9], Heitlager et al. tried to address this with a more practical QM with general metrics derived from industry experience. Similarly, other QMs with proclaimed practical focus were created, e.g. Columbus QM [1], SQUALE [16], or QUAMOCO [28]. Especially QUAMOCO moved away from having a single numeric value for maintainability and focused on the impact of system properties on maintenance activities. In [6], Ferenc et al. provided a thorough overview of the intricacies of these "modern" QMs and their different advantages.

There are also some approaches specific for SBSs. Shim et al. adopted the QMOOD structure to create a SOA QM with concrete metrics at the lowest level [25]. However, their model covers more than maintainability (e.g. effectiveness). Senivongse et al. built on this QM and created a version that focused solely on maintainability [24]. Unfortunately, they introduced even more metrics of semantic nature that can not be gathered automatically. Lastly, Goeb et al. used QUAMOCO to design a domain-specific SOA variation [8]. While their approach seems promising, their model appears to be too complex for some use cases and some of their metrics are of semantic nature.

3 MAINTAINABILITY MODEL FOR SERVICES (MM4S)

The presented approaches in chapter 2 make it obvious that QMs with practical applicability are in high demand, yet there seems to be little consensus on the details of these models despite hierarchical layering and concrete metrics. Moreover, in the field of SBSs and μSBSs there is currently no comprehensive maintainability QM that focuses on simplicity and purely automatic measurements. We therefore propose a Maintainability Model for Services (MM4S).

An intra-company focus group with developers combined with a literature review yielded the following first requirements for the desired quality model. It should a) be specifically tailored to SBSs, b) be also applicable to μSBSs, c) be focused on practical applicability w.r.t. scope and complexity, d) provide concrete metrics usable for automatic evaluation with fast feedback, and e) rely on existing and proven approaches and metrics.

Based on these general requirements, we chose a simple hierarchical structure. Maintainability is the top-level quality attribute in the first layer. The second layer consists of *Service Properties* (SPs) that represent some maintainability-related characteristic inherent to a service or the complete system. In the first version, 5 Service Properties (SPs) were identified via the literature review and the focus group discussion, namely coupling, cohesion, granularity, complexity, and code maturation. The last level consists of *Service Metrics* (SMs) that are related to one SP. To limit issues with subjective bias, the metric selection from existing literature like [3] or [18] was guided by a weighted scoring method with a set of *MUST* and *SHOULD* criteria listed in Table 1 [27]. These criteria were derived from the initial requirements and subsequently discussed and prioritized in the focus group. If a candidate metric satisfied all *MUST* criteria, it was assigned a score between 0 and 1 for each *SHOULD* criterion based on how well it fulfilled this criterion. These values were then multiplied with the respective criterion weight and summed up for the final score. Lastly, the candidate metrics with the highest scores were brought back into the focus group for voting, which resulted in the selection of 11 metrics for the first version of MM4S.

A visual representation of the QM can be seen in Fig. 2. While 4 of the 11 metrics (CB, CR, CC, TC) were not specifically designed for SBSs, we see them as a valuable foundation. They are included in publications with practical focus, e.g. [9] or [18], which is exactly what the QM is aiming for. The following section presents the details of MM4S.

Coupling: The degree or indication of the strength of interdependencies and interconnections of a service with other services.

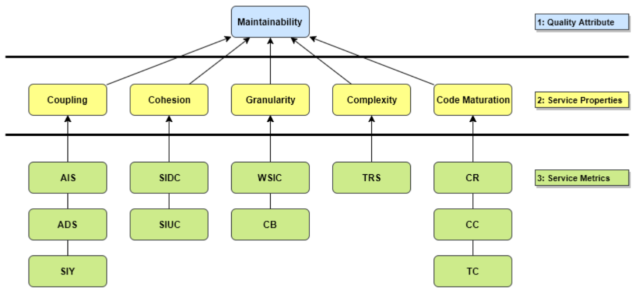

Figure 2: Maintainability Model for Services (MM4S)

An SBS consisting of loosely coupled services that form only a small number of couplings can be easier maintained.

Absolute Importance of the Service (AIS): The number of clients that invoke at least one operation of a service's interface [23]. With a focus on network traffic, Rud et al. exclude clients that reside on the same node, which we do not recommend, because it is not very important for coupling in terms of maintainability, especially in the age of cloud computing.

Absolute Dependence of the Service (ADS): The number of other services that service S depends on, i.e. the number of services from which S invokes at least one operation [23].

Services Interdependence in the System (SIY): The number of service pairs that are bi-directionally dependent on each other [23], i.e. pairs $\langle S_1, S_2 \rangle$ where service S_1 calls S_2 while service S_2 may also call S_1. Such interdependent pairs should be avoided. The ideal value for SIY is therefore 0. As opposed to most other presented metrics, it is a system-level metric.

Cohesion: The extent to which the operations of a service contribute to one and only one task or functionality. A high degree of cohesion positively impacts maintenance activities.

Service Interface Data Cohesion (SIDC): The cohesion of a given service S with respect to the similarity of parameter data types of the operations of its interface [20]. If all operations use the same parameter data types, S is highly cohesive with $\text{SIDC}(S) = 1$. To calculate $\text{SIDC}(S)$, the number of operations with common data types is divided by the total number of discrete data types present in the interface. Values close to 1 are favorable for the maintainability.

Service Interface Usage Cohesion (SIUC): The cohesion of a given service S based on the invocation behavior of clients using operations from its interface [20]. S is considered highly cohesive, if every operation is used by every consumer calling S. $\text{SIUC}(S)$ is therefore defined as the ratio between the summed up numbers of used operations per client and the product of number of clients and number of operations of S. Like SIDC, it produces values between 0 and 1.

Granularity: The size and degree of decomposition of an SBS, which is the aggregate of its children, i.e. its services. A large SBS with a too fine-grained or coarse-grained structure is harder to maintain than a small one with appropriate granularity (provided all other things are similar). Due to the high degree of technological heterogeneity in SBSs and μSBSs, the traditionally used metric *Lines of Code (LOC)* is of limited value here.

Weighted Service Interface Count (WSIC): $\text{WSIC}(S)$ is the number of exposed interface operations of service S [11]. Operations can be weighted based on the number or granularity of parameters with default weight 1. Lower WSIC values are generally more favorable for the maintainability of a service.

Component Balance (CB): A system-level metric to evaluate the appropriateness of granularity, i.e. if the number and size uniformity of components (in this case services) are in a favorable range for maintainability [4]. CB is the product of two other metrics, namely *System Breakdown* (SB, producing a value between 0 and 1 based on the number of top-level components) and *Component Size Uniformity* (CSU, producing a value between 0 and 1 based on the Gini coefficient of component volumes). Bouwers et al. apply LOC to measure the volume of a component. However, as pointed out above this is problematic in a very heterogeneous system. Alternatively, WSIC could be used. The metric has to be initialized with both the optimum (CB = 1) and upper bound worst (CB = 0) number of top-level components. Bouwers et al. use heuristics based on a repository of existing software systems to do that.

Complexity: The amount and variety of internal work carried out by an SBS as well as the degree of interaction between its services necessary to achieve this. High complexity has a negative influence on maintainability.

Total Response for Service (TRS): An adaptation of the OO metric Response for Class (RFC) [5] to a service context [21]. Response for Operation (RFO) for an operation O is defined as the number of sequences of other operations and local methods that can be executed in response to an incoming request for O. This includes other services as well. Based on that, $\text{TRS}(S)$ is the sum of all RFO values for all operations of the interface of service S.

Code Maturation: The degree of technical proficiency and consistency of the code base of an SBS. High code maturation indicates a strong professionalism of the developing organization and makes it easier to maintain the SBS.

Comment Ratio (CR): The ratio indicating how well the source code is commented to aid understandability [7]. It is defined as the ratio between the number of comment lines and the total number of lines. A variation would be the ratio between comment lines and non-commented lines. Sophisticated approaches are also able to distinguish between useful and superfluous comments [26].

Clone Coverage (CC): The amount of duplicated code, usually expressed via the ratio between duplicated lines and the total number of lines [15]. Code clones make it harder to analyze and change a service.

Test Coverage (TC): A percentage value indicating how well the source code has been covered by automatic tests [29]. Although a lot of different test coverage metrics exist, the most frequently used ones are line coverage (based on the number of lines) and condition coverage (based on the number of control flow branches). Tools like SonarQube often use a combination of both. The TC metric is especially important in a Microservices context.

4 SUMMARY AND CONCLUSION

While there are a lot of metrics for SBSs as well as several quality models in the literature, there is no dominant maintainability QM for SBSs with a focus on fast and automatic evaluation, simplicity, and practical applicability. As a proposed answer to this research question, we presented MM4S, a layered QM with *Service Metrics* related to *Service Properties*. This version of MM4S is the first artifact draft created by combining the results of an industry focus group (action research) and a literature review. One important caveat with our approach is concrete tool support. While the chosen metrics can all be gathered automatically in theory, there needs to be tooling for the comfortable integration in existing development processes. With μSBSs, this becomes much more difficult due to the large degree of heterogeneity and number of services, which needs to be addressed appropriately. Since this paper documents research in progress, our QM has not been fully evaluated yet. However, true to the nature of DSR projects, we plan to bring our research artifact back into the environment, where the idea for its conception emerged from. Further action research, quality control case studies on prototypes as well as concrete industry projects, and finally a developer survey about estimated usefulness are planned to evaluate and enhance MM4S. Moreover, we are very interested in feedback from the research community.

ACKNOWLEDGMENTS

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).

REFERENCES

[1] Tibor Bakota, Peter Hegedus, Peter Kortvelyesi, Rudolf Ferenc, and Tibor Gyimothy. 2011. A probabilistic software quality model. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 243–252. https://doi.org/10.1109/ICSM.2011.6080791

[2] J. Bansiya and C.G. Davis. 2002. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering* 28, 1 (2002), 4–17. https://doi.org/10.1109/32.979986 arXiv:arXiv:1011.1669v3

[3] Justus Bogner, Stefan Wagner, and Alfred Zimmermann. 2017. Automatically Measuring the Maintainability of Service- and Microservice-based Systems - a Literature Review. In *2017 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, IWSM-MENSURA 2017*. ACM, Gothenburg, Sweden.

[4] Eric Bouwers, Jose Pedro Correia, Arie van Deursen, and Joost Visser. 2011. Quantifying the Analyzability of Software Architectures. In *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*. IEEE, 83–92. https://doi.org/10.1109/WICSA.2011.20

[5] S.R. Chidamber and C.F. Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 20, 6 (jun 1994), 476–493. https://doi.org/10.1109/32.295895 arXiv:1011.1669

[6] Rudolf Ferenc, Péter Hegedűs, and Tibor Gyimóthy. 2014. Software Product Quality Models. In *Evolving Software Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 65–100. https://doi.org/10.1007/978-3-642-45398-4_3

[7] Beat Fluri, Michael Wursch, and Harald C. Gall. 2007. Do Code and Comments Co-Evolve? On the Relation between Source Code and Comment Changes. In *14th Working Conference on Reverse Engineering (WCRE 2007)*. IEEE, 70–79. https://doi.org/10.1109/WCRE.2007.21

[8] Andreas Goeb and Klaus Lochmann. 2011. A software quality model for SOA. In *Proceedings of the 8th international workshop on Software quality - WoSQ '11*. ACM Press, New York, New York, USA, 18. https://doi.org/10.1145/2024587.2024593

[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] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. 2004. Design Science in Information Systems Research. *MIS Quarterly* 28, 1 (2004), 75–105. https://doi.org/10.2307/25148625 arXiv:http://dl.acm.org/citation.cfm?id=2017212.2017217

[11] Mamoun Hirzalla, Jane Cleland-Huang, and Ali Arsanjani. 2009. A Metrics Suite for Evaluating Flexibility and Complexity in Service Oriented Architectures. In *Service-Oriented Computing – ICSOC 2008 Workshops: ICSOC 2008 International Workshops, Sydney, Australia, December 1st, 2008, Revised Selected Papers*, Bernd J. Krämer, Kwei-Jay Lin, and Priya Narasimhan (Eds.). Lecture Notes in Computer Science, Vol. 4749. Springer Berlin Heidelberg, Berlin, Heidelberg, 41–52. https://doi.org/10.1007/978-3-642-01247-1_5

[12] International Organization For Standardization. 2001. ISO/IEC 9126:2001. (2001), 25 pages. https://doi.org/10.1002/(SICI)1099-1670(199603)2:1<35::AID-SPIP29>3.0.CO;2-3

[13] International Organization For Standardization. 2011. ISO/IEC 25010:2011. (2011), 25 pages.

[14] Paul Johannesson and Erik Perjons. 2014. *An Introduction to Design Science*. Springer International Publishing, Cham. 197 pages. https://doi.org/10.1007/978-3-319-10632-8

[15] Rainer Koschke. 2007. Survey of Research on Software Clones. In *Duplication, Redundancy, and Similarity in Software (Dagstuhl Seminar Proceedings)*, Rainer Koschke, Ettore Merlo, and Andrew Walenstein (Eds.). Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, Dagstuhl, Germany, 24.

[16] Karine Mordal-Manet, Francoise Balmas, Simon Denier, Stephane Ducasse, Harald Wertz, Jannik Laval, Fabrice Bellingard, and Philippe Vaillergues. 2009. The squale model - A practice-based industrial quality model. In *2009 IEEE International Conference on Software Maintenance*. IEEE, 531–534. https://doi.org/10.1109/ICSM.2009.5306381

[17] Sam Newman. 2015. *Building Microservices: Designing Fine-Grained Systems* (1st ed.). O'Reilly Media. 280 pages.

[18] 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

[19] M.P. Papazoglou. 2003. Service-oriented computing: concepts, characteristics and directions. In *Proceedings of the 7th International Conference on Properties and Applications of Dielectric Materials (Cat. No.03CH37417)*. IEEE Comput. Soc, 3–12. https://doi.org/10.1109/WISE.2003.1254461

[20] Mikhail Perepletchikov, Caspar Ryan, and Keith Frampton. 2007. Cohesion Metrics for Predicting Maintainability of Service-Oriented Software. In *Seventh International Conference on Quality Software (QSIC 2007)*. IEEE, 328–335. https://doi.org/10.1109/QSIC.2007.4385516

[21] Mikhail Perepletchikov, Caspar Ryan, Keith Frampton, and Zahir Tari. 2007. Coupling Metrics for Predicting Maintainability in Service-Oriented Designs. In *2007 Australian Software Engineering Conference (ASWEC'07)*. IEEE, 329–340. https://doi.org/10.1109/ASWEC.2007.17

[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] Dmytro Rud, Andreas Schmietendorf, and Reiner R. Dumke. 2006. Product Metrics for Service-Oriented Infrastructures. In *IWSM/MetriKon*.

[24] Twittie Senivongse and Assawin Puapolthep. 2015. A Maintainability Assessment Model for Service-Oriented Systems. In *Proceedings of the World Congress on Engineering and Computer Science*, Vol. I. San Francisco, CA, USA.

[25] Bingu Shim, Siho Choue, Suntae Kim, and Sooyong Park. 2008. A Design Quality Model for Service-Oriented Architecture. In *2008 15th Asia-Pacific Software Engineering Conference*. IEEE, 403–410. https://doi.org/10.1109/APSEC.2008.32

[26] Daniela Steidl, Benjamin Hummel, and Elmar Juergens. 2013. Quality analysis of source code comments. In *2013 21st International Conference on Program Comprehension (ICPC)*. IEEE, 83–92. https://doi.org/10.1109/ICPC.2013.6613836

[27] Evangelos Triantaphyllou. 2000. *Multi-criteria Decision Making Methods: A Comparative Study*. Applied Optimization, Vol. 44. Springer US, Boston, MA. https://doi.org/10.1007/978-1-4757-3157-6

[28] Stefan Wagner, Andreas Goeb, Lars Heinemann, Michael Kläs, Constanza Lampasona, Klaus Lochmann, Alois Mayr, Reinhold Plösch, Andreas Seidl, Jonathan Streit, and Adam Trendowicz. 2015. Operationalised product quality models and assessment: The Quamoco approach. *Information and Software Technology* 62 (jun 2015), 101–123. https://doi.org/10.1016/j.infsof.2015.02.009

[29] Hong Zhu, Patrick a. V. Hall, and John H. R. May. 1997. Software unit test coverage and adequacy. *Comput. Surveys* 29, 4 (dec 1997), 366–427. https://doi.org/10.1145/267580.267590

[30] Olaf Zimmermann. 2016. Microservices tenets. *Computer Science - Research and Development* (nov 2016), 1–10. https://doi.org/10.1007/s00450-016-0337-0