Towards a Theory-Driven Metric: A Case Study Measuring Maintainability

Florian Demel
Hochschule der Medien Stuttgart
Germany
mail@floriandemel.de

Professor Dr. Tobias Jordine
Hochschule der Medien Stuttgart
Germany
jordine@hdm-stuttgart.de

December 11, 2024

Abstract

Maintenance refers to the work performed by software developers after the initial release of a software system. It is a significant part of the software lifecycle and accounts for more than 50 % of the total cost of a software system. To reduce maintenance costs, this paper introduces a theory-driven approach to measuring software maintainability: A new maintainability metric that has been developed based on a review of scientific literature and a case study including five experts from a German engineering enterprise. The foundation of the metric includes organizational, architectural, social, and technical factors of maintainability. A case study demonstrates the practical application of the metric on a full stack software system of the enterprise.

Keywords: software maintenance, software maintainability, software measurement, software product quality, software design metrics

1 Introduction

Modern software development is a dynamic and complex process, which happens over the entire lifecycle of a software system. While initial development remains significant, maintenance is increasingly important. Maintenance costs range from 50 % to 90 % of the total cost of a software system [2]. Given its significance, the ease and efficiency of maintenance, the so called maintainability, is key to a project's success. However, a common understanding of maintainability and its measurement is still missing.

This paper presents a theory-driven approach to measuring maintainability. A new metric has been developed based on a review of scientific literature and a case study on a full stack software system from a large German mechanical engineering enterprise. The research involved five experts, software developers and architects.

The metric allows the quantitative evaluation of a software system using ordinal scales to measure multiple maintainability criteria— theoretical factors developed in the research. The five most relevant criteria of the metric are:

- **Architectural Integrity**: Alignment of architecture and system requirements; preservation of system knowledge; ongoing evolutionary adaptation.
- **Team**: Autonomous development teams; effective collaboration between team members.
- **Domain Split**: Little dependencies between teams and domains; manageable cognitive load; management of system components by single teams; optimal team size of 5-9 members.
- **Experts**: Availability of experts with system knowledge; low expert turnover rate; knowledge sharing and collaboration; measures to retain system knowledge.
- **Documentation**: Well-structured documentation; readable code; documented system architecture.

Based on these criteria, the metric was used exploratively on the enterprise's software system. Overall, it showed a promising approach to practically measure maintainability that could be used for further research.

2 Key Results

In the research process, multiple artifacts were developed. They are the results of a comprehensive discussion of maintainability and standalone artifacts that can be used in further research. The key results are:

1. The **Criteria Catalog**, a collection of all factors of maintainability discovered in the research.

2. The **Maintainability Metric**, a metric developed from the most relevant criteria of the catalog, that provides rating scales allowing for a numeric evaluation of maintainability.

3. The **Case Study**, an example of a practical application of the metric, evaluating the maintainability of the enterprise's full stack software system.

2.1 Criteria Catalog

The criteria catalog includes maintainability factors identified from the literature and the five experts involved. It consists of 13 criteria that collectively describe maintainability. *Table 1* provides a brief overview of the catalog, introducing the baseline of the maintainability metric.

Table 1: Criteria Catalog

No.	Name	Definition
1	Code Complexity	Code complexity depends on the problem's complexity reflected in the code and the simplicity of the code that enhances understanding and development. It is measured with static code complexity metrics [1]. High complexity is a sign of low maintainability [12].
2	Technical Complexity	The choice of technologies can positively or negatively impact system complexity based on their compatibility with the systems architectural requirements.
3	System Size	The complexity increases with the number of system components.
4	System Modularity	Modularity is determined by dependencies, coupling, cohesion, and interfaces between system components - system level modularity. A good modular system is easier to maintain.
5	Code Modularity	Focuses on dependencies, coupling, cohesion, and interfaces within a system component - code level modularity.
6	Architectural Integrity	An architecture, that is aligned with system requirements and evolutionary developed by experts familiar with the system, is easier to maintain.
7	Technological Sustainability	Using future-proof technologies allows a longer maintenance period of the system without deprecation.
8	Analyzability	The ability to analyze and test a system enables maintenance work.

Continued on next page

[1] Static code complexity metrics: Halstead-Metric [5], Cyclomatic-Complexity [11]

9	External System Dependencies	Dependencies to external components are causing higher efforts than internal dependencies and required knowledge within the system.
10	Team	The team is a organizational unit solving problems and maintaining the system. A self-responsible and collaborative team is more effective.
11	Domain Split	Dividing the system into domains and teams affects development and therefor maintenance work; consider team dependencies, competencies, application division, and team size.
12	Experts	Experts hold critical domain knowledge, technical knowledge, and interpersonal knowledge. They are required for sustainable development and maintenance.
13	Documentation	Well-structured, compact documentation that includes architecture supports developers in understanding and maintaining the system; code is part of this documentation.

As the criteria are the result of a discussion between the expert's opinions and the results of a literature review, there is no single source for each criterion. Some criteria are based on the literature, some on experts' opinions, some are combinations of both. Nevertheless, the following theoretical basics introduce the approaches and theories upon which the catalog has been built.

Software Evolution and the Staged Model

The understanding of maintainability in this research aligns with the *Staged Model*, a software lifecycle model introduced by Bennett [2]. It is a counter approach to the famous *Waterfall Model* of the IEEE from the 1960s [7] and is based on Lehman's Laws of Software Evolution [8] from the 1970s.

Considering the work of Lehman and Bennett, **Maintenance** includes all changes to a software system after its initial deployment. These changes can be bug fixes, adaptations to new requirements, or servicing work. In the Staged Model, maintenance occurs in the stages two (Evolution) to five (Closedown). Maintenance is a continuous process in the lifecycle, performed by developers. **Maintainability** describes how effortful the maintenance of a system is. A system has good maintainability if changes can be made effectively, easily, and quickly. Maintainability is poor if changes are effortful, complex, or cannot be made.

Besides the definition of maintenance and maintainability, Bennett describes *Architectural Integrity* (Criterion 6) as a key factor for maintainability. As long as the architectural integrity of a system is preserved, it can be maintained and stays in the Evolution stage of the Staged Model. Once the architectural integrity is lost, developers cannot maintain the system effectively anymore. The

system will then enter the later stages of the Staged Model (Servicing, Phase-Out, Closedown), become a legacy system, and be more effortful to maintain. The key reason for the architectural integrity to get lost, is the lack of preserved knowledge about the architecture and the system. *Documentation* and *Experts* (Criteria 13 and 12) are key factors for preserving knowledge and maintain architectural integrity.

Software Quality and Modularity

Referencing maintainability, the ISO/IEC 25010 standard [4] describes it as a software quality characteristic of software systems with the sub-characteristics: *Modularity*, *Reusability*, *Analysability*, *Modifiability*, and *Testability*.

While modularity is the first sub-characteristic of maintainability, it serves as the theoretical foundation for all other sub-characteristics. It depends on low coupling, high cohesion [13] and well-defined interfaces, which enable the concept of information hiding [1]. While modularity is described with technical architecture in mind, it is influenced by the organization, as Conway's Law implies [1].

The research found that modularity can be divided into two parts: *System Modularity* (Criterion 4) and *Code Modularity* (Criterion 5). System modularity describes the modularity between system components, while code modularity describes the modularity within a system component.

The only sub-characteristic of ISO/IEC 25010 included in the criteria catalog after discussion is *Analyzability* (Criterion 8). It not only depends on well-designed modularity and low system complexity, as Herrmann describes [6]. The experts added observability tooling and the setup of the tools as another factor, prioritizing it over the other sub-characteristics.

Concept of Complexity

Besides software quality and the Staged Model, the concept of complexity is central to maintainability. Malhotra and Chug define complexity as [9]: "Complexity is anything related to the structure of a software system that makes it hard to understand and modify the system." Additionally, the book *A Philosophy of Software Design* describes complexity as the limiting factor in software development. Complexity can be measured as the sum of the complexity of the parts of a system, weighted by the time developers spend on those parts [9]. As maintenance is the process of changing a system, it is limited by the system's complexity. The more complex a system is, the lower its maintainability.

Technical Complexity (Criterion 2) is one factor of complexity, understood with this definition. As the experts implied, the sum of a system's complexity increases if it uses technologies requiring significant effort to run properly. If these technologies do not decrease complexity elsewhere in the system, the system's maintainability decreases. If the technology reduces system complexity more than it increases it, it is a good choice for the system.

System Size (Criterion 3) can also be explained with complexity calculation. The more parts a system has, the more complex it is, as long as the new components are worked on and have a complexity above zero.

Overall, Ousterhout describes simplicity as the key to maintainability [12]. If a system is simple, it is easy to understand and change; it is not complex. This explains the importance of simple code in *Code Complexity* (Criterion 1).

Complexity Metrics

As complexity is a central part of maintainability, the research includes metrics that measure it. According to Malhotra and Chug, the three most commonly used metrics for maintainability are complexity metrics: Halstead Metrics, Cyclomatic Complexity, and C&K Metrics [9]. For this research, the exact numbers or formulas of the metrics are less important than the concepts behind the measurements.

Code Complexity (Criterion 1) is based on Halstead Metrics and Cyclomatic Complexity, both measuring code complexity. Halstead Metrics measure code complexity based on the number of operators and operands [5]. Cyclomatic Complexity measures code complexity based on the number of paths through the code [11]. Both indicate code complexity, not the maintainability of the entire system.

C&K Metrics measure complexity based on code-level modularity measurements. For example, one metric measures the number of methods two classes call between each other (Coupling between Object Classes). Understanding this metric is part of *Code Modularity* (Criterion 5) [3].

Teams, Domains, and Team Topologies

The organization of the development team was introduced by the experts as a key factor for maintainability. The book *Team Topologies* by Skelton and Pais describes how the design of teams, domains, and applications influences the effectiveness of development teams [14]. It is based on Conway's Law, which has already been mentioned, and describes that the way a software system is structured should match how the development teams communicate. Team Topologies mentions that organizational modularity, similar to system modularity, relies on low coupling, high cohesion, and well-designed interfaces. These guidelines must be considered when designing an organizational structure on the team level [14]. Besides organizational-level modularity, cognitive load—the amount of competencies a developer must handle— is a factor to consider when designing teams and domains.

This impacts how teams are organized, how they communicate and collaborate, which is reflected in the criterion *Team* (Criterion 10). Additionally, it covers how competencies and responsibilities are divided between teams and domains; in short, this is called *Domain Split* (Criterion 11).

2.2 Maintainability Metric

Following the introduction of the theoretical foundations of the research, table 1 provides an overview of the maintainability metric. The metric was developed based on the most relevant criteria from the criteria catalog.

Each criterion consists of multiple aspects that describe the factors of maintainability in detail and introduce a rating ranging from 0 (low maintainability) to 1 (high maintainability). The score from 0 to 1 can be interpreted as a percentage, where 0 is 0 % and 1 is 100 % maintainability. Its practical application will be described in the case study. Table 2 provides an overview:

Table 2: Maintainability Metric

Criterion	Aspect	Description
Criterion 1: Architectural Integrity		
1.1	Architectural Fit	The architecture aligns with system requirements. Rated from 0 (not aligned) to 1 (fully aligned).
1.2	Preservation of knowledge	Knowledge about the architecture is preserved in the organization. Rated from 0 (lost) to 1 (fully preserved).
1.3	Evolutionary Adaptation	Regular adaptation of the architecture to new requirements. Rated from 0 (none) to 1 (continuous).
Criterion 2: Team		
2.1	Team Autonomy	Teams operate autonomously with minimal communication required. Rated from 0 (none) to 1 (full).
2.2	Team Collaboration	Teams members are collaborating effectively. Rated from 0 (poor) to 1 (excellent).
Criterion 3: Domain Split		
3.1	Team Dependencies	Few dependencies between teams enhance efficiency. Rated from 0 (many) to 1 (few).
3.2	Cognitive Load	Teams are not overloaded with responsibilities. Rated from 0 (high) to 1 (low).
3.3	Microservice Ownership	Each microservice is managed by a single team. Rated from 0 (multiple teams) to 1 (single team).
3.4	Team Size	Optimal team size is 5-9 members. Rated from 0 (outside range) to 1 (within range).
Criterion 4: Experts		
4.1	Knowledge Availability	Experts with necessary knowledge are available. Rated from 0 (none) to 1 (sufficient).
4.2	Expert Turnover	Low turnover ensures knowledge retention. Rated from 0 (high) to 1 (low).
4.3	Knowledge Sharing	Active knowledge sharing prevents silos. Rated from 0 (poor) to 1 (excellent).

Continued on next page

4.4	Team Collaboration	Experts work well together. Rated from 0 (poor) to 1 (excellent).
4.5	Knowledge Retention Measures	Measures are in place to retain knowledge. Rated from 0 (none) to 1 (sufficient).
Criterion 5: Documentation		
5.1	Structured Documentation	Documentation is well-structured and concise. Rated from 0 (poor) to 1 (excellent).
5.2	Readable Code	Code is readable and well-commented. Rated from 0 (poor) to 1 (excellent).
5.3	Documented Architecture	Architecture and system structure are fully documented. Rated from 0 (none) to 1 (complete).

2.3 Case Study

The metric provides a rating for each aspect of the criteria, allowing for a quantitative evaluation of each aspect, which together form an evaluation for the entire system. The case study was conducted with the five experts from the enterprise with as a questionnaire to practically apply the metric. The experts rated each aspect with ordinal scales, which are used in social sciences to measure subjective opinions. A typical approach of questionnaire design.

The questions have been designed based on the metric's ratings, with ordinal scales for each aspect of the criteria. For example, a question for aspect 1.1, architectural fit, criterion 1 could be:

Does the current system architecture align with the requirements of the system?

The possible ordinal answers in the questionnaire are:

Strongly Agree >Somewhat Agree >Somewhat Disagree >Strongly Disagree

The ordinal rating of each aspect can be transformed into a number based on a scale from 0 to 1. This scale could look like the one in Table 3.

Score	**Description**
1	Strongly Agree
0.66	Somewhat Agree
0.33	Somewhat Disagree
0	Strongly Disagree

Table 3: Ordinal Rating Scale

This way, each answer of the questionnaire can be transformed into a number, as it is needed for the metric. The questionnaire is designed with questions

similar to the one in the example, but for all aspects of the criteria. So each expert rates each aspect of each criterion with a number between 0 and 1. The mean of the ratings of all experts of all aspects of all criteria is the quantitative evaluation of the maintainability of the software system.

3 Research Procedure

To provide a better understanding how the research delivered the results, the procedure of the research is presented in Figure 1.

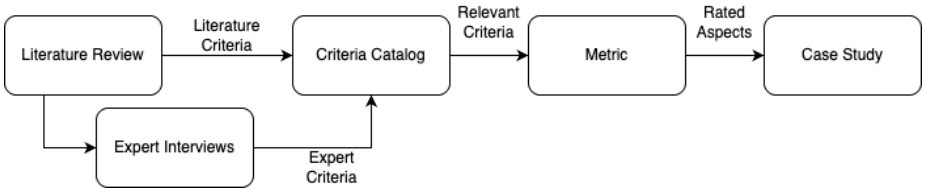

Figure 1: Research Procedure

The research began with a comprehensive *literature review* to identify the theoretical foundation, collecting the literature-criteria. Following this, *expert interviews* were conducted to gain new insights with the experts and create the expert-criteria. The analysis of the interviews has been carried out with a methodical content analysis following the approach of Mayring [10]. The literature- and expert-criteria were then compared and discussed to develop the *criteria catalog*.

After this, the most relevant criteria were selected to create the *metric*. The selection of relevant criteria was based on an assessment of the literature, priorities identified by the experts, and the saturation of the topics in the interviews. To finalize the metric, detailed descriptions of the criteria aspects were provided, and a rating scale was introduced. Finally, the *case study* applied the metric in a real-world scenario to validate its practical applicability.

4 Discussion

Theoretical Foundations

This paper does not allow the presentation of the detailed steps involved in the creation of the maintainability metric or the criteria catalog, as it is a summary of the research. Some of the most valuable results of the research are the discussions that happened to create these artifacts. For everyone interested in the details of the research, the full research paper is available, get in contact with the authors.

Results of the Case Study

As the research is based on five experts from a single enterprise, the results of the case study are not generalizable or representative. They are not the focus of this summary.

Nevertheless, they are a proof of concept that a theory-driven metric like this one can be applied in a real-world scenario and deliver reasonable results. The measured value of the maintainability of the enterprise's system by all five experts was 0.6 with a standard deviation of 0.13, indicating a tendency towards this value—an indicator that the metric is capable of measuring at least a somewhat objective or comparable value of maintainability.

Results of the Catalog and Metric

Besides the case study, the criteria catalog and the maintainability metric are the results of the qualitative part of research. The expert interviews showed a saturation of the discussed topics, suggesting a representative selection and prioritization of criteria, especially focused on the enterprise's system and full stack software systems in general. These artifacts can serve as a foundation for further research in maintainability.

5 Summary

The introduced maintainability metric provides a comprehensive view of maintainability for full stack systems like the one of the enterprise. It does not only take aspects into account that can be measured on specific code-based characteristics, like the presented complexity metrics do. It includes multiple criteria identified through research in literature and with experts. This is how a maintainability metric should work: it must consider all relevant factors, including organizational, architectural, social, and technical aspects. This is a key finding and proposal of this research. The way these measurements can be done, compared, and generalized should be the focus of further scientific work.

References

[1] Fabian Beck and Stephan Diehl. "On the congruence of modularity and code coupling". In: *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. ESEC/FSE '11 (2011), pp. 354–364. DOI: 10.1145/2025113.2025162. URL: https://doi.org/10.1145/2025113.2025162.

[2] Wilde Bennett Rajlich. "Software Evolution and the Staged Model of the Software Lifecycle". In: *Advances in Computers* 56 (Dec. 2002), pp. 1–54. DOI: 10.1016/S0065-24580280003-1.

[3] S.R. Chidamber and C.F. Kemerer. “A metrics suite for object oriented design”. In: *IEEE Transactions on Software Engineering* 20.6 (1994), pp. 476–493. DOI: 10.1109/32.295895.

[4] John Estdale and Elli Georgiadou. *Applying the ISO/IEC 25010 Quality Models to Software Product: 25th European Conference, EuroSPI 2018, Bilbao, Spain, September 5-7, 2018, Proceedings*. Jan. 2018, pp. 492–503. ISBN: 978-3-319-97924-3. DOI: 10.1007/978-3-319-97925-0_42.

[5] T Hariprasad et al. “Software complexity analysis using halstead metrics”. In: *2017 International Conference on Trends in Electronics and Informatics (ICEI)* (2017), pp. 1109–1113. DOI: 10.1109/ICOEI.2017.8300883. URL: https://ieeexplore.ieee.org/document/8300883.

[6] Andrea Herrmann. “Analysierbarkeit”. In: *Softwaretechnik-Trends Band 43, Heft 4* (2023). ISSN: 0720-8928.

[7] IEEE. *IEEE Standard Glossary of Software Engineering Terminology*. Zugriff: 2024-09-22. 1990. URL: https://ieeexplore.ieee.org/document/159342.

[8] M.M. Lehman. “Programs, life cycles, and laws of software evolution”. In: *Proceedings of the IEEE* 68.9 (1980), pp. 1060–1076. DOI: 10.1109/PROC.1980.11805.

[9] Ruchika Malhotra and Anuradha Chug. “Software Maintainability: Systematic Literature Review and Current Trends”. In: *International Journal of Software Engineering and Knowledge Engineering* 26.08 (2016), pp. 1221–1253. DOI: 10.1142/S0218194016500431. URL: https://doi.org/10.1142/S0218194016500431.

[10] Philipp Mayring. “Qualitative Inhaltsanalyse”. In: (2010), pp. 601–613. DOI: 10.1007/978-3-531-92052-8_42. URL: https://doi.org/10.1007/978-3-531-92052-8_42.

[11] T.J. McCabe. “A Complexity Measure”. In: *IEEE Transactions on Software Engineering* SE-2.4 (1976), pp. 308–320. DOI: 10.1109/TSE.1976.233837.

[12] John Ousterhout. *A Philosophy of Software Design*. Yaknyam Press, 2018. ISBN: 978-1732102200.

[13] Mark Richards and Neal Ford. *Handbuch moderner Softwarearchitektur: Architekturstile, Patterns und Best Practices*. 1. Edition. Seitenzahl von EPUB-Datei möglicherweise ungenau. O’Reilly Verlag, 2020. ISBN: 978-3960091493.

[14] Matthew Skelton and Manuel Pais. *Team Topologies: Organizing Business and Technology Teams for Fast Flow*. IT Revolution Press, 2019. ISBN: 978-1942788812.