Data Science Techniques for Modelling Execution Tracing Quality

Tamas Galli

Faculty of Computing, Engineering and Media

May 2022

A thesis submitted in fulfilment of the University's requirements for the Degree of Doctor of Philosophy.

DE MONTFORT UNIVERSITY
LEICESTER

Declaration

To the best of my knowledge I confirm that the work in this thesis is my original work undertaken for the degree of PhD in the Faculty of CEM, De Montfort University. I confirm that no material of this thesis has been submitted for any other degree or qualification at any other university. I also declare that parts of this thesis have been submitted for publications and conferences.

Abstract

This research presents how to handle a research problem when the research variables are still unknown, and no quantitative study is possible; how to identify the research variables, to be able to perform a quantitative research, how to collect data by means of the research variables identified, and how to carry out modelling with the considerations of the specificities of the problem domain. In addition, validation is also encompassed in the scope of modelling in the current study. Thus, the work presented in this thesis comprises the typical stages a complex data science problem requires, including qualitative and quantitative research, data collection, modelling of vagueness and uncertainty, and the leverage of artificial intelligence to gain such insights, which are impossible with traditional methods.

The problem domain of the research conducted encompasses software product quality modelling, and assessment, with particular focus on execution tracing quality. The terms execution tracing quality and logging are used interchangeably throughout the thesis.

The research methods and mathematical tools used allow considering uncertainty and vagueness inherently associated with the quality measurement and assessment process through which reality can be approximated more appropriately in comparison to plain statistical modelling techniques. Furthermore, the modelling approach offers direct insights into the problem domain by the application of linguistic rules, which is an additional advantage.

The thesis reports (1) an in-depth investigation of all the identified software product quality models, (2) a unified summary of the identified software product quality models with their terminologies and concepts, (3) the identification of the variables influencing execution tracing quality, (4) the quality model constructed to describe execution tracing quality, and (5) the link of the constructed quality model to the quality model of the ISO/IEC 25010 standard, with the possibility of tailoring to specific project needs.

Further work, outside the frames of this PhD thesis, would also be useful as presented in the study: (1) to define application-project profiles to assist tailoring the quality model for execution tracing to specific application and project domains, and (2) to approximate the present quality model for execution tracing, within defined bounds, by simpler mathematical approaches.

In conclusion, the research contributes to (1) supporting the daily work of software professionals, who need to analyse execution traces; (2) raising awareness that execution tracing quality has a huge impact on software development, software maintenance and on the professionals involved in the different stages of the software development life-cycle; (3) providing a framework in which the present endeavours for log improvements can be placed, and (4) suggesting an extension of the ISO/IEC 25010 standard by linking the constructed quality model to that. In addition, in the scope of the qualitative research methodology, the current PhD thesis contributes to the knowledge of research methods with determining a saturation point in the course of the data collection process.

International Journals:

The below publications appear in the References list of the present thesis as entries [68], [70], [69], [65], [64], [63], respectively.

1. PhD:

   (a) T. Galli, F. Chiclana, and F. Siewe. Genetic Algorithm-Based Fuzzy Inference System for Describing Execution Tracing Quality. Mathematics, 9(21), MDPI 2021. ISSN 2227-7390. doi: `https://doi.org/10.3390/math9212822`. URL `https://www.mdpi.com/2227-7390/9/21/2822`

   (b) T. Galli, F. Chiclana, and F. Siewe. On the use of quality models to address distinct quality views. Applied System Innovation, 4(3), MDPI 2021. ISSN 2571-5577. doi: `https://doi.org/10.3390/asi4030041`. URL `https://www.mdpi.com/2571-5577/4/3/41`.

   (c) T. Galli, F. Chiclana, and F. Siewe. Quality properties of execution tracing, an empirical study. Applied System Innovation, 4(1), MDPI 2021. ISSN 2571-5577. doi: `https://doi.org/10.3390/asi4010020`. URL `https://www.mdpi.com/2571-5577/4/1/20`.

   (d) T. Galli, F. Chiclana, and F. Siewe. Software product quality models, developments, trends and evaluation. SN Computer Science, 1(3), Springer 2020. ISSN 2661-8907. doi: `https://doi.org/10.1007/s42979-020-00140-z`. URL `https://link.springer.com/article/10.1007%2Fs42979-020-00140-z`

2. MPhil:

   (a) T. Galli, F. Chiclana, J. Carter, and H. Janicke. Towards introducing execution tracing to software product quality frameworks. Acta Polytechnica Hungarica, 11(3), Obuda University and IEEE Hungary 2014. ISSN 1785-8860. doi: `https://doi.org/10.12700/APH.11.03.2014.03.1`. URL `http://acta.uni-obuda.hu/Galli_Chiclana_Carter_Janicke_49.pdf`

   (b) T. Galli, F. Chiclana, J. Carter, and H. Janicke. Modelling execution tracing quality by type-1 fuzzy logic. Acta Polytechnica Hungarica, 8(10), Obuda University and IEEE Hungary 2013. ISSN 1785-8860. doi: `https://doi.org/10.12700/APH.10.08.2013.8.3`. URL `http://acta.uni-obuda.hu/Galli_Chiclana_Carter_Janicke_46.pdf`

Book Chapters:

The below publication appears in the References list of the present thesis as entry [66].

1. T. Galli, F. Chiclana, and F. Siewe. Performance of execution tracing with aspect-oriented and conventional approaches. In V. Gupta and C. Gupta, editors, Research and Evidence in Software Engineering, From Empirical Studies to Open Source Artifacts, pages 1–40. Auerbach Publications, Routledge, Taylor and Francis Group, 2021. ISBN 9780367358525.

Acknowledgements

This research comprises many years of careful work. I gratefully thank both of my supervisors: Prof. Francisco Chiclana and Dr. Francois Siewe for being open, outstanding, and supportive mentors. They both provided me with friendly and professional guidance in the course of the PhD studies, which I sincerely appreciate and for which I am grateful. I would also like to thank Professor Eerke Boiten for the encouraging, and precise work on the annual reviews, as well as examining my research, and providing feedback on my work and progress each year. Furthermore, I would like to thank the members of the Doctoral College for their supportive attitude.

I also gratefully acknowledge the participants, who voluntarily contributed to the data collection process, to the model evaluation, adjustments and tuning. They all provided invaluable assistance in the course of the research. In addition, I also thank the managers, who helpfully consented to recruit participants for the research and encouraged the voluntary participation of their companies, departments, and groups. Moreover, I also thank my direct managers and immediate colleagues, who carried the workload during my regular unpaid holidays. As a sign of my appreciation, I would like to list the first names of all participants, colleagues, and managers, who were directly involved in this process. The names are listed alphabetically. If the same name is listed more than once, it means more than one individual. Thanks go to all of them: Abraham, Adam, Agoston, Akos, Albert, Andras, Anja, Aron, Artur, Balazs, Bela, Bence, Bence, Botond, Dennis, Dorka, Edina, Erika, Erzsebet, Eva, Gabor, Gabor, Gabor, Gabor, Gabor, Gabriella, Gergely, Gergely, Gergely, Geza, Gyula, Hajnalka, Ildiko, Janos, Judit, Klara, Kornel, Laszlo, Laszlo, Laszlo, Laszlo, Mario, Mark, Marta, Mihaly, Monika, Nora, Norbert, Norbert, Norbert, Oliver, Ors, Pal, Peter, Peter, Peter, Peter, Peter, Reka, Robert, Robert, Robert, Robert, Robert, Ruben, Sara, Szabolcs, Szabolcs, Szilvia, Tamas, Tamas, Tamas, Tamas, Tamas, Timea, Tunde, Viktoria, Vilmos, Zita, Zsolt, Zsuzsanna.

I am also grateful to my friends, brothers, and sisters, who prayed for me regularly while I was conducting the research: Sister Mary FMM., Rev. Paul and Kati, P. Nemeshegyi Peter SJ., Rev. Philip and Agi, Teofil, Petra, Agnes, and Edit. Special thanks go to Ulrike Nater who proofread some chapters of this thesis in spite of the short notice.

Furthermore, I would like to gratefully thank my family members for all the love and care they showed towards me. I would especially like to thank my mother, Maria, who always followed my journey on the road of the PhD studies with love and appreciation, my step-father Misi, my granny Irma, my step-grandfather Bandi bacsi, who showed lively interest towards my studies, and my brother, Richard, who encouraged me while I approached the end of the PhD research.

Although I write it at the end, but I mean it first of all: I would like to give thanks for the encompassing love to the compassionate God of the Holy Trinity, who gave everything described above including strength, creativity, opportunities, and who guided my steps on the way of the research journey: “Praise the LORD, my soul!” Psalm 103,1

Contents

List of Figures

List of Tables

Definition of Terms

AI Artificial intelligence

ANFIS Adaptive Network-based Fuzzy Inference System

Execution tracing A mechanism or tool to dump the data about the program state, and the path of execution for offline analysis, mainly for software developers; frequently used as synonym for logging in the literature.

GA Genetic algorithm

Logging A mechanism or tool to dump the data about the program state, and the path of execution for offline analysis, mainly for system administrators; frequently used as synonym for execution tracing in the literature.

MAE Mean absolute error

Quality attribute A low-level quality property in ISO/IEC 25000 standard family, in contrast to the ISO/IEC 14598 standard where the term attribute is used to refer to the high-level quality properties of the ISO/IEC 9126 family.

Quality characteristic A high-level quality property that is located at the top of the hierarchy in the ISO/IEC 9126 standard family; in the terminology of ISO/IEC 14598 standard it is called attribute.

Quality measure The association of quality measure elements, and a measurement function to compute with; it is used in the ISO/IEC 25000 standard family with similar meaning as the metric in the ISO/IEC 9126 standard family.

Quality measure element A measurable property of quality; A predefined list of quality measure elements is published in the ISO/IEC 25021 standard.

Quality metric The definition of the measurement method of quality properties including the definition of the measurement scale; quality metrics are assigned to the low-level quality attributes of the ISO/IEC 9126 standard family.

RMSE Root-mean-square error

Software product quality framework A quality model that aims to describe each known aspect of software product quality with the goal of completeness; it is a complete software product quality model.

Software product quality model A model that describes either each known aspect of the software product quality or only a part of it; it is not necessarily a complete model.

Software product quality model class A set of related software product quality models with the same concepts.

Software product quality model family The term is a synonym for software product quality model class.

SPQM Software Product Quality Model

Chapter 1

Introduction

This chapter outlines the research problem, the motivation and the research methods implemented, as well as a summary of the research contributions to the existent knowledge.

1.1 Background

Execution tracing is a tool to follow the thread of execution and state changes in software applications to assist software developers and software maintainers while analysing software faults offline [62, 65, 69]. In a strict sense, execution tracing is meant for software developers and software logging, which has a similar purpose, rather for system administrators or security professionals, who deal with wide ranges of applications and also perform auditing activities [65]. Nevertheless, the literature uses both terms in an interchangeable manner, and this convention is observed throughout the thesis.

The increasing complexity, distributed nature and growing size of software applications make localising errors in the source code more challenging. The quality of execution tracing, including where log statements are added to the source code, influences the time spent on error analysis drastically [195]. Recent research focuses on (1) where to insert log statements in the source code [43, 61, 157, 158, 250, 252, 263, 264], (2) what data to log to provide sufficient information for the error analysis [87, 251, 252, 263], and (3) how to perform logging [33, 252, 263]. Tool support for identifying the key places in the source code for inserting log statements and for attaining optimal performance with regard to the data needs is available to some extent [43, 87, 250, 251, 252, 263, 264].

Nevertheless, many different publications reveal that no industrial standard and very limited or no guidelines are available to support software professionals to implement execution tracing in an application [32, 33, 87, 157, 162].

In addition, debuggers do not offer adequate solution for the error analysis if (1) issues in deployed applications related to concurrency, or (2) performance need to undergo investigation, or (3) actions of distributed systems need to be followed throughout component and host boundaries, or (4) reproduction of errors in real-time embedded systems can cause damages [74, 143, 195, 231, 237].

Profilers can serve as an alternative for collecting, analysing and presenting the required data; however, such software requires installation on the target system, which implicates security clearance of third party products and possibly also network and security reconfiguration to provide access for the software professionals to the profiler [55, 88, 182]. This is not always a viable option. In addition, collecting extensive sets of data from the target system has also performance cost [66] beyond the licencing cost of the profilers [55, 88].

Karahasanovic and Thomas found the understanding of the program logic as the primary difficulty while maintaining object-oriented applications [121]. Execution tracing and its quality contribute to understanding the program logic to a large extent [62, 195]. Thus, execution tracing impacts on the analysability of software systems, through which it directly impacts on the development and maintenance costs [62, 63, 66].

Preliminary analysis showed that execution tracing quality and its quality properties are not defined and adequately dealt with in the software product quality frameworks [62, 64]. Therefore, further analysis, based on broader inputs, was necessary to verify these statements, which was done in the scope of the present study.

Unless a very limited set of software product quality is measured [70], both the software product quality measurement and the assessment process require human intervention, which means that further problems may manifest regarding the uncertainty stemming from human evaluation and human judgements. Thus, if human intervention is required in the quality measurement process, then uncertainty may be introduced. Conventional modelling approaches endeavour to exclude uncertainty as a disturbing factor but the reality can be approximated better if the inherent uncertainty is not ruled out but the quality model considers and handles it [68].

In conclusion, execution tracing is an important tool to analyse software faults. Furthermore, to address the above identified gaps, there is a need to model execution tracing quality with regard to the uncertainty inherently present in the software quality modelling process.

The conducted research presents how to handle a research problem when the research variables, i.e. the quality properties of execution tracing, are still unknown, and no quantitative study is possible; how to identify the research variables to be able to perform a quantitative research, how to collect data by means of the identified research variables, and how to carry out modelling with the considerations of the specificities of the problem domain. Thus, the work presented in this thesis comprises the typical stages a complex data science problem requires, including qualitative and quantitative research, data collection and design, modelling of vagueness and uncertainty, and the leverage of artificial intelligence to gain such insights, which are impossible with traditional methods. Moreover, it is shown how the developed quality model for execution tracing can be linked to a software product quality framework to consider its effects while measuring and assessing the software product as a whole.

Throughout the thesis a distinction is made between software process quality models and software product quality models. The latter consider the quality properties of software products, while the former deal with the properties of the processes, which result in software products [106, 112].

1.2 Related Works

On the one hand, Hegeman [91] provides a review on the software product quality models published up to 2011, and an analysis of the correlation between SQALE indices [152] and the perceived quality. On the other hand, Ferenc et al. investigate the software product quality models up to 2014 [54]. However, (1) neither Hegeman nor Ferenc et al. identified and analysed the sources in the scope of a systematic literature review with precise rules [130], (2) nor they measure the relevance of the identified quality models in the scientific and industrial communities, and (3) achievements have obviously been absent since 2014. Moreover, both research reports also introduce 'partial' quality models which are able to handle only a specific part of software product quality like a high-level quality property, such as maintainability [54, 91].

Different machine learning methods have already been used, in the quality modelling domain, to discover unknown patterns, predict and classify errors, handle uncertainty and missing information [6, 24, 29, 125, 144, 145, 164, 170, 172, 181, 186, 193, 203, 221, 234]. However, the listed studies do not involve modelling execution tracing quality. The only research identified to test different modelling techniques, in the scope of a pilot study, for execution tracing quality was conducted by Galli et al. in [62, 63, 64].

Galli et al. review six software product quality frameworks and investigate the extension facilities they offer with regard to execution tracing quality [62, 64]. They present a pilot study to test four different modelling approaches to construct a quality model for execution tracing, and they conclude that the Takagi-Sugeno-Kang inference with overlapping Gaussian membership functions produce the best results in the context of the study [62, 63]. The mentioned review was not a systematic literature review [130] to identify the software product quality frameworks, while the relevance of the software product quality models introduced was not determined [62, 64]. In addition, the pilot quality model for execution tracing was not constructed on data gained from a defined study population with an adequate sample because its goal was merely to test the modelling performance of different techniques [62, 63]. The defined fuzzy rules in these studies describe the experiences of solely one individual expert, and the fuzzy rule extraction process was not based on machine learning [62, 63]. The current PhD research builds on the findings and experiences of the pilot study documented in [62, 63, 64] but it is a new, distinct research project with novel contributions to the body of knowledge as introduced in section 1.5.

1.3 Research Questions

The research was conducted to answer the following questions:

RQ1: What software product quality models aim to assess all defined characteristics of software product quality? This research question is handled in chapter 2.

RQ2: Do the identified software product quality models adequately handle execution tracing quality? This research question is examined in chapter 2.

RQ3: Which quality manifestations can the identified software product quality frameworks address? This research question is answered in chapter 3.

RQ4: What quality properties, i.e. input variables, determine execution tracing quality? This research question is introduced and answered in chapter 4.

RQ5: How can execution tracing quality be modelled with regard to uncertainty inherently present in the quality measurement process? This research question is elaborated and answered in chapter 5.

RQ6: To which software product quality framework should the quality model of execution tracing be linked, and how, to consider its effect in the overall software product quality? This question is answered in chapter 6.

The research questions require different methodological approaches, which are documented in the corresponding chapters indicated in the listing above. RQ1 was necessary to identify the existing software product quality frameworks, while RQ2 is required to decide whether execution tracing quality is appropriately dealt with by any of the identified software product quality frameworks. RQ3 was implied by RQ1 to understand how the different quality models handle the abstract notion of software product quality and to what extent they are able to capture it. RQ4 is a prerequisite for RQ5 to implement quantitative modelling. Furthermore, RQ6 requires input in the form of answers for RQ1, RQ2 and RQ3; moreover, the results of RQ4 and RQ5 also affect it.

1.4 Research Methodology

The research conducted consists of three distinct but related activities: (1) analysing the literature in a manner that reduces the potential bias, (2) identifying the quality properties of execution tracing, and (3) constructing the quality model for execution tracing. These different activities influence each other, as illustrated in figure 1.1, and their outputs connect by linking the constructed quality model for execution tracing to a relevant software product quality framework that can accommodate it. The research includes both qualitative and quantitative approaches. As human experience is directly considered and collected in the course of the study, it was necessary to apply for ethical approval to follow good scientific practise and to comply with the regulations of the De Montfort University (ethical approval code: 1718/453, date of approval: 31 May 2018).

1.4.1 Systematic Literature Review

Identifying and analysing the relevant sources in the literature is a challenge and a necessary step in each research project. The bias of the source selection and the information extracted from the sources influences the foundations of the study. Kitchenham et al. presented and adapted the method of systematic literature review for the computing domain to address these issues [129, 130, 133].

The author followed the rules of the systematic literature review to minimise the potential bias while identifying and analysing the existing software product quality frameworks. The review is reported in chapter 2. Kitchenham et al. [130] propose to use a quality score for each individual publication investigated. In addition to the individual quality scores, a scoring scheme was developed using four relevance indicators: (1) relevance score, (2) quality score average, (3) publication range and (4) the average of the 12-month Google Relative Search Index [76], to determine the importance of the software product quality models the publications can be linked to.

Software product quality frameworks usually offer the possibility of tailoring to specific project needs. When the result of tailoring a software product quality framework counts as a new quality model, or how the scoring scheme considers whether a software product quality framework was published in a mature state at once or in parts with smaller increments are defined in chapter 2.

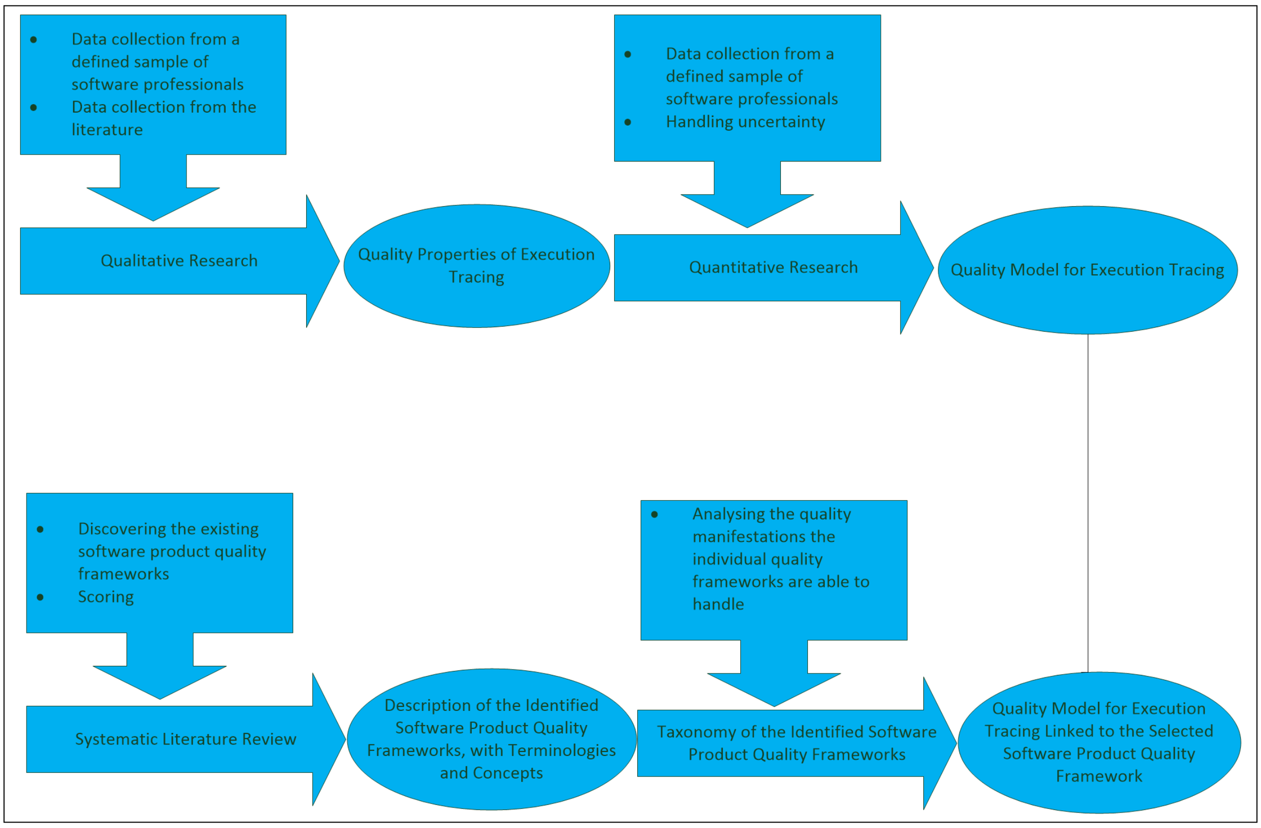
Data collection from a defined sample of software professionals
Data collection from the literature
Qualitative Research
Quality Properties of Execution Tracing
Data collection from a defined sample of software professionals
Handling uncertainty
Quantitative Research
Quality Model for Execution Tracing
Discovering the existing software product quality frameworks
Scoring
Systematic Literature Review
Description of the Identified Software Product Quality Frameworks, with Terminologies and Concepts
Analysing the quality manifestations the individual quality frameworks are able to handle
Taxonomy of the Identified Software Product Quality Frameworks
Quality Model for Execution Tracing Linked to the Selected Software Product Quality Framework

Figure 1.1: Simplified Schematic Depiction of the Research Methodology

Finally, each software product quality framework identified in the scope of the systematic literature review is reported with its terminology, concepts, and relevance indicators computed in the scope of the study. Furthermore, it is also investigated whether execution tracing quality is dealt with by the identified software product quality frameworks.

1.4.2 Taxonomy of Software Product Quality Models

The identified software product quality frameworks differ significantly in handling the distinct manifestations of software product quality, which has a considerable impact on the quality measurement and assessment results. Chapter 3 reports the analysis performed and the taxonomy laid down; moreover, it highlights the application and practical consequences of the established taxonomy.

1.4.3 Identifying the Quality Properties of Execution Tracing

Conducting quantitative research, such as modelling, requires research variables. This stage of the study was carried out to identify the research variables, i.e. the quality properties of execution tracing. The extraction of these variables from the collected data corpus and the data collection process is reported in chapter 4.

This part of the research relies on two different inputs: (1) the literature and (2) the experiences of software professionals. The literature search was implemented systematically, the search strings and the returned results from the different document databases are reported in appendix B.4.1. In addition, the collected text of the relevant publications was investigated, and the explicit or implicit wishes or articulated requirements towards execution tracing are reported in appendix B.4.2. After transcribing the implicit requirements or wishes to explicit ones, the collected list of quality property candidates underwent data coding [210] and the quality properties based on the literature were formed.

The data collection was repeated with different focus groups of software professionals. Each focus group produced a weighted list of ideas for the potential quality properties based on the experiments of the participants. The data collection was stopped when the saturation point was approximated with the list of all ideas collected. This process is also explained in chapter 4. The data were analysed and coded [210], which resulted in the quality properties based on the experience of the software professionals.

Finally, the quality properties stemming from two different sources: (1) the literature and (2) the experiences of software professionals were checked for deviations. After resolving the identified deviations, the process led to the definition of the ultimate quality properties for execution tracing.

The collected data, and the outputs of data collection process are documented in appendix B. As part of the ethical approval mentioned above, informed consent was obtained from the software professionals, who participated. Each participant had freedom to withdraw from the study at any time.

1.4.4 Constructing the Quality Model for Execution Tracing

Formalising the input variables, i.e. the quality properties, of execution tracing opened the possibility for quantitative investigations. The experiences of software professionals were collected through an online questionnaire [67] in the form of rating eight fictive use cases, rating real projects and assigning quality values to the combinations of extreme input variable values. The ratio-scale data underwent exploratory data analysis and outlier detection. After removing the outliers, the collected data set was randomly split into a training and a checking set. The model construction took place in an adaptive manner: (1) a fuzzy model [255, 258] was constructed by means of genetic algorithms [51], and (2) a model was developed by the ANFIS approach [116]. The outcomes of the two different approaches were compared to verify the results. The constructed quality model was validated and tuned.

The identification of the study population, the sampling methods, the sample size were handled in chapter 5. In addition, the model construction by genetic algorithms, including the defined upper bounds for the number of linguistic rules [256, 257], was also introduced in the same chapter, accompanied by the model verification and validation stages.

The outcomes of the adaptive model construction, comprising of the error indicators: mean absolute error, root-mean-square error, minimum error and maximum error are included in appendix C. Again, the respondents of the online questionnaire [67] provided informed consent for the voluntary participation with the freedom to withdraw from the study.

1.5 Research Contributions

This section summarises the obtained research outcomes with references to the chapters of the thesis that report the specific contributions.

1. Identification of existing software product quality frameworks with the terminology and concepts of each model. This is considered a scientific contribution obtained from the systematic literature review with strict rules [130], documented in chapter 2, that allows understanding each software product quality framework and the quality measurement and assessment results.

2. Thorough analysis revealed that execution tracing quality was not appropriately handled in the identified software product quality frameworks, which has not changed with the developments in the field during the recent years.

3. In addition to the unified description of the identified software product quality frameworks, relevance indicators were developed and computed for each software product quality model in chapter 2.

4. A possible taxonomy of the identified software product quality frameworks, with regard to the quality manifestations they are able to capture and describe, is provided in chapter 3.

5. Furthermore, to highlight the practical consequences of the quality model taxonomy, guidelines are provided for (1) selecting a software product quality model for a specific problem, and (2) assessing a software product quality model to determine the extent of the quality manifestations it is able to handle. The latter can assist software professionals to avoid pitfalls caused by "good-quality software" statements based on software product quality models, which are capable to capture only a very limited extent of software product quality. This is explained in detail in chapter 3.

6. Identifying the quality properties of execution tracing, based on a defined study population, is introduced in chapter 4, which is a requirement for quantitative modelling and a necessity to form practical guidelines for software professionals.

7. While collecting data in the focus groups for the identification of the quality properties of execution tracing, a simple but novel approach was used to estimate whether the saturation point was reached in the course of the data collection. This approach is also documented in chapter 4.

8. Modelling execution tracing quality with the consideration of uncertainty implicated by (1) the experiences of many different software professionals from a defined study population, and (2) by the software product quality measurement process is reported in chapter 5.

9. Extracting linguistic rules by machine learning to construct a fuzzy model and to gain insight into the problem domain in a human-understandable manner is documented in chapter 5.

10. Linking the quality model for execution tracing to a software product quality framework, relevant for both the industrial and scientific communities that allows considering execution tracing quality while the software product is measured and assessed as a whole, is reported in chapter 6.

1.6 Structure of the Thesis

Chapter 2 describes the systematic literature review [130] of the research, introduces the scoring scheme with the developed relevance indicators, highlights the relevance of each identified software product quality framework, and lays down each quality model with terminologies and concepts in a unified manner. In addition, a brief summary is provided regarding how some of the concepts introduced assist with handling abstraction in fields other than software product quality modelling.

Chapter 3 introduces a taxonomy of the identified software product quality frameworks to describe which manifestations of software product quality the different models can handle; moreover, it illustrates the application of the established software product quality framework taxonomy in the practice.

Chapter 4 describes how the quality properties of execution tracing are defined based on the literature and on the data collected from software professionals. Furthermore, this chapter also includes an investigation whether the quality properties extracted from the two different data sets are in accordance with each other.

Chapter 5 presents a quantitative modelling approach using AI, based on fuzzy logic [255], genetic algorithms [51], and ANFIS [116] to construct a quality model for execution tracing. The online questionnaire used for data collection and the collected data are available in [67].

Chapter 6 reports on how the constructed quality model for execution tracing can be linked to the quality model with the highest relevance score and with the largest scope of known quality manifestations.

Chapter 7 closes the thesis with conclusions and potential future work plans.

1.7 Summary

This chapter provided a brief summary on the background of the research; illustrated the gaps to address; introduced the research questions; reported the outline of the applied research methods and the research contributions. Furthermore, it also gave an account of how the defined research variables are used in the course of the quantitative investigation.

Chapter 2

Review of the Relevant Literature

A significant part of the content of this chapter appeared in the author's articles published in SN Computer Science (Springer Nature – 2020, reference [65]) and Applied System Innovation (MDPI – 2021, reference [69]).

The review was performed as a systematic literature review as per Kitchenham [130], which is considered a research sub-project with own research questions and goals. The chapter contains the unified descriptions of the identified 23 software product quality model families with their terminologies and concepts.

This chapter focuses on software product quality models published since 2000, which aim for completeness to handle each know aspect of software product quality; moreover, the relevance of each model is measured by introducing indicators with regard to the scientific and industrial communities. The identified 23 software product quality model families differ significantly in terms of publication intensity, publication range, quality score average, relevance score and the 12-month average of the Google Relative Search Index [76].

The experiences accumulated on the field of software product quality modelling motivated researchers to transfer the concepts to other areas where abstract entities need to be compared or assessed including the quality of higher educational teaching and business processes [82, 223, 224], which is also briefly highlighted.

Systematic literature reviews encompass the following major stages [129, 130]: (1) identifying the problem, (2) developing a protocol for the research, (3) defining the research questions, (4) developing the keywords for the search, (5) defining the search strategy and identifying the document databases, (6) defining the inclusion and exclusion criteria, (7) defining the data to be extracted from the documents identified, (8) establishing the evaluation criteria for the documents, (9) searching the document databases, (10) recording the data, and (11) synthesizing, reporting the results.

2.1 Problem Identification

Since the 1970's, software product quality models have evolved in their abilities to capture and describe the abstract notion of software quality [64, 65]. Many of these models only deal with a specific part of software quality, which makes them ineligible to assess the quality of software products as a whole [65]. The existent literature has not thoroughly examined and listed all the available models aiming to describe the known set of the whole software product quality 1.2.

2.2 Research Protocol of the Systematic Literature Review

The research protocol explains the preformed steps of the implemented systematic literature review.

2.2.1 Research Questions of the Review

The research problem can be covered with the questions:

RQ1: What software product quality models aim to assess all defined characteristics of software product quality?

RQ2: Do the identified software product quality models adequately handle execution tracing quality?

2.2.2 Keyword Development and Search Strategy

In the scope of the keyword development the particularities of the topic area were considered to form search terms that are specific enough to limit the search results on the given field. The terminology of the known models were analysed to form the search strings.

The ISO/IEC models and their derivatives match with the term “software product quality model” as this is the terminology of the standard. In contrast, the terminology of the SQALE model is different and less specific; thus, a broader search string would have been required to identify all the publications automatically. The broader search terms however return such high number of publications that processing each of them would have caused a serious impediment. For this reasons, in accordance with the proposal made by Kitchenham and Brereton [129], automatic and manual searches were integrated. The references of the automatically identified publications pointing at unidentified software product quality models were followed and those publications were also analysed. Depending on the information revealed by the publications gained from the references, new conventional literature review searches were performed, focused on the particular area of interest shown by the referenced publications. The search strings for these manual searches were not recorded and the publications included this way in the scope of the analysis were marked with the origin: “manual search” in Appendix A.2.

Search Strings

The search strings developed consider the most important synonyms on the field:

1. For ISO/IEC specific models: “Software Product Quality Model” OR “Software Product Quality Framework”

2. For filtering the topic area in general: (“quality model” OR “quality framework”) AND assessment AND software AND analysis AND (measure OR measurement)
3. For the field of SQALE: manual search integrated with automatic author search

Identifying the Research Databases

For the software engineering research field, Kitchenham and Brereton recommend IEEE and ACM digital libraries due to their good coverage of important journals and conferences; moreover, at least two general purpose digital libraries from SCOPUS, El Compendix and Web of Science [129]. The university library of the De Montfort University recommends the following digital libraries for searching Computer Science literature[1]: ACM, IEEE, EBSCO Academic Search Premier, Science Direct, Web of Science. Consolidating the two recommendations, the following digital libraries were selected:

1. ACM Digital Library
2. IEEE
3. EBSCO Academic Search Premier
4. SCOPUS
5. Science Direct
6. Web of Science

In all the databases the search was conducted on the area of Computer Science for the period 2000-2017 on the metadata fields including title, keywords and abstract. In addition, the search was repeated in the ACM and IEEE databases for the period 2018-2022. For reproducibility purposes, the search strings are documented in appendix A.1.

2.2.3 Inclusion and Exclusion Criteria

All documents returned by the automatic search are included in the review unless the below exclusion criteria apply. In addition, quality frameworks with the concepts of existing software product quality models on a field unrelated to computing are introduced in the section 2.3.4.

Exclusion Criteria for Documents Identified by the Automatic Search

1. The publication is not a software product quality model (e.g. it is a *software process quality model*),
2. The model introduced does not attempt to deal with the whole set of quality properties but focusses on a specific subset of them (e.g. models for maintainability or performance),

[1] `http://libguides.library.dmu.ac.uk/c.php?g=51890&p=335386`, [accessed: 30.05.2017]

3. The publication is a comparative study about software product quality models but does not introduce a new software product quality model or the adaptation of an existing model.
4. The publication reports the progress about creating a new software product quality model but the model is not defined at the time of the publication. In contrast, publications defining the enhancements of already available software product quality models are included.
5. The publication highlights only some principles of existing software product quality models without extending or without tailoring a concrete model to a specific application domain.

Exclusion and Inclusion Criteria for the Complementing Manual Searches

The automatic searches were complemented by manual searches as mentioned in section 2.2.2. This process resulted in the identification of software product quality models. The exclusion criteria of the automatic search were enforced. However, if a manually identified publication prior to year 2000 introduced a software product quality model from which a new model was derived after 2000, then it was included in the list of software product quality models identified with a zero relevance score value.

2.2.4 Extracted Information

The following data were extracted: The names of the quality models identified, the model descriptions and information to determine whether the quality model is a new model, or an adaptation of a known model.

It was not always simple to consider the boundaries of quality models with regard to published adjustments and tailoring. While doing the classification of the publications, every reasonable effort was made to correctly decide whether the given publication introduced a new quality model or enhancements of an existent model. If the adjustment or tailoring to a specific context of use articulated completely new concepts in comparison to the existing quality model it was built on, then the publication was classified as the introduction of a new software product quality model. Otherwise, the publication was classified as adapting an existing software product quality model.

2.2.5 Evaluation Criteria for the Documents

The description of each individual software product quality model differs considerably. Quality models exist with high number of publications, analysis and demonstration of use such as SQALE [152], the ISO/IEC 9126 [106], and the ISO/IEC 25010 [112] standard families. However, other quality models come with concise descriptions and a brief demonstration of use. In addition, a third group of quality model definitions also appear in the identified sources with incomplete descriptions or with the lack of metric definitions. Thus, the quality models introduced differ significantly in the amount and the depth of introduction they present.

For assessing the documents returned by the queries, the recommendations in [129, 130] were considered. In this manner for each individual publication we constructed a quality score that shows: (1) clarity of presentation and (2) actuality of the publication. Both scoring criteria are presented below in detail. The individual quality score value of each publication is defined as the product of these two factors.

Presentation Clarity

Scale: ratio

Range: [0;5]

Meaning of the score value: The value indicates how clearly, and completely the publication introduces the software product quality model with regard to concepts and details.

Score value 5: In terms of the software product quality models, the model is presented clearly in appropriate depth including the outline of the concepts of the model, all defined quality properties, sub-properties, possibly metrics, measures and any other defined characteristics necessary to use. If the metrics or measures are not available in the given publication, they must be available elsewhere and the given publication and the documents that publish the measures and metrics must form a consistent unit.

Score value 0: In terms of the software product quality models, (a) the presentation of the model is unclear, (b) the concepts of the model are not outlined, (c) the references to related models are inaccurate, or (d) the model's quality properties, sub-properties metrics, measures or necessary characteristics are not defined nor published elsewhere in an available manner.

If a model is published fully-fledged with its concepts, quality properties, metrics as a consistent unit, with accurate references to other models which influenced its development, then the maximal score value is assigned. If the model undergoes further adjustments and these developments are published, then the adjustments are considered as increments that do not define a complete model. Consequently, the publications with the increments depending on the depth of information they present usually earn a score less than the maximum.

Actuality of the model

Scale: ratio

Range: [0;5]

Meaning of the score value: The value indicates how up-to-date the publication is.

Score value 5: The publication is up-to-date.

Score value 0: The publication is not up-to-date. If the publication has a zero score value, then its quality model is not relevant for the current investigation; however, its concepts can be of importance.

Table 2.1: Scoring Criteria on Actuality, Source: [65]

Publication date	Maximum score value that can be given
2014 or after	5
]2014; 2011]	4
]2011; 2007]	3
]2007; 2004]	2
]2004; 2000]	1
1999 or before	0

Computation of the Quality Score for an Individual Publication

The quality score is computed for each publication as the product of its (1) clarity score and (2) actuality score values. The metrics and the guides of the software product quality models if published separately are not counted towards the relevance scores unless they introduce the concepts of the model not published elsewhere. SQUALE publishes two of its model concepts (1) concept of practices [16] and (2) concept of quality defect resolution strategy [100] in detail as part of two separate reports, which mainly define metrics and constitute guides. Consequently, these two publications have been considered for the above reason while computing the scores.

2.2.6 Relevance Indicators

In the scope of the review four different relevance indicators have been created: (1) quality score average, (2) relevance score, (3) publication range, and (4) the 12-month average of the Google Relative Search Index [76] each of which is explained below.

The identified publications define, tailor or adapt software product quality models. Consequently, each publication can be assigned to a software product quality model. This way, clusters of related publications can be constructed and labelled with the name of the software product quality model on which the publication is based. Such clusters are named software product quality model classes or software product quality model families, as a synonym, depending on the context of use throughout the thesis. If a publication appears with a new software product quality model, then it establishes a new model class for which the name of the new software product quality model is used.

Quality score for an individual publication was defined in section 2.2.5. Computing the quality score average within the software product quality model classes illustrates how clear and how up-to-date the given software product quality model class is, based on the publications identified.

The relevance scores of the software product quality model classes are computed by summing the quality score values of the individual publications in the cluster. The relevance score designates how vivid the research interest is in connection with a given software product quality model family.

The publication range carries important information showing the period in which the identified publications regarding the given quality model class appeared. It helps to elicit whether there is a constant interest related to a given software product quality model or only a limited resource, such as a research grant, was used without raising interest in the scientific and industrial communities. The relevance score value and the publication range need to be interpreted together.

In addition, the publications stem from both the academic and industrial research domains and involve companies such as Air France, Siemens, IBM, Samsung, Qualixo and Mitre. Consequently, the relevance score value mirrors, to some extent, the industrial interest of the quality model classes; however, industrial research interest does not necessarily mean practical everyday use cases. Thus, an additional indicator is introduced: the average of the Google Relative Search Index during a 12-month period. Google Relative Search Index shows on a continuous scale of [0; 100] how popular the given search strings are in comparison to each other. The names of the quality model classes were applied as search string. In some of the cases, the quality model class names possess different connotations exceeding the quality domain. Such cases are designated as “n.a.” values in the ranking table of the identified models.

Kitchenham et al. [129] propose the use of quality indicators for each publication and the computation of quality scores with regard to several criteria, including more researchers to compute average scores to enhance the reliability of the quality assessment. However, they solely utilise these values for the quality assessment of the individual publications and through that for the quality assessment of the review process. Thus, they apply the quality scores in a similar way to our individual quality scores to assess each publication. To the best knowledge of the author, no publication applies the aggregation of these scores according to particular clusters among the research data. This simple technique can be used to express existing differences among the research subjects in a quantitative manner.

By means of this novelty, the relevance of the different software product quality model classes, from the point of view of the scientific and industrial communities, is described since it shows publication intensity and, through that, research intensity. Some of the quality models were defined in one step and further enhancements were published afterwards. In contrast, other quality models were published in smaller increments and not defined fully in one step. This latter approach inherently results in more publications, which was considered while assessing the individual publications as defined in section 2.2.5.

2.2.7 Recording

The following data about the publications identified were recorded:

1. The citation of the publication;
2. Content of the publication for analysis;
3. Reasons for exclusion, if any.

2.3 Outcomes of the Review

2.3.1 Relevance of the Identified Software Product Quality Models

In the scope of the literature review, all software product quality models returned by the documented searches were considered according to the defined inclusion and exclusion criteria. Software process quality models and cost models were excluded, as well as, product quality models that do not aim to handle the whole set of software product quality. Each software product quality framework has twofold purpose: (1) to provide means to assesses the quality of a concrete software product as a whole and (2) define quality targets for a given software product [99]. Process quality, product quality and cost models serve different purposes. Kläse et al. published a classification scheme in [134], in which they classify twenty models including process quality, product quality and cost models to provide assistance with quality model selection for a specific purpose, organisation or project.

Table 2.2 lists all the identified quality model classes, with relevance score, average quality score, publication range, and the 12-month average of Google Relative Search Index, sorted by relevance scores to highlight the ones in the focus of vivid research interest in the academic and industrial communities. Moreover, all the model classes identified by the automatic search are listed separately in table 2.3.

Table 2.2: Ranking of the Quality Model Classes by Relevance Scores Including Manual and Automatic Searches, Source: [65]

Ranking	Model Class	Relevance Score	Quality Score Average	Publication Range After 2000	Google Relative Search Index, Average for 12 Months
1	ISO25010 [44, 58, 97, 98, 112, 167, 167, 192, 219, 219, 235]	130	16.25	[2011; 2018]	30.02
2	ISO9126 [10, 36, 94, 106, 120, 160, 164, 196, 240]	120	13.33	[2000; 2017]	53.06
3	SQALE [91, 150, 151, 152, 153, 154, 155, 156]	107	13.38	[2009; 2016]	18.33
4	Quamoco [73, 242, 243, 244]	90	22.5	[2012; 2015]	0
5	EMISQ [141, 200, 201]	38	12.67	[2008; 2011]	0
6	SQUALE [16, 100, 146, 183]	36	9	[2012; 2015]	n.a.
7	ADEQUATE [93, 124]	18	9	[2005; 2009]	n.a.
8	COQUALMO [26, 169]	15	7.5	[2008; 2008]	0.21
=9	FURPS [49, 79, 80]	10	3.33	[2005; 2005]	20.56
=9	SQAE and ISO9126 combination [38]	10	10	[2004; 2004]	0
=9	Ulan et al. [235]	10	10	[2018; 2018]	n.a.
10	Kim and Lee [128]	9	9	[2009; 2009]	n.a.
11	GEQUAMO [71]	5	5	[2003; 2003]	0
12	McCall et al. [20, 179]	1	0.5	[2002; 2002]	n.a.
=13	2D Model [262]	0	0	n.a.	n.a.
=13	Boehm et al.[27]	0	0	n.a.	n.a.
=13	Dromey [45]	0	0	n.a.	n.a.
=13	GQM [239]	0	0	n.a.	40.73
=13	IEEE Metrics Framework Reaffirmed in 2009[99]	0	0	n.a.	0
=13	Metrics Framework for Mobile Apps [59]	0	0	n.a.	0
=13	SATC [96]	0	0	n.a.	n.a.
=13	SQAE [173]	0	0	n.a.	n.a.
=13	SQUID [132]	0	0	n.a.	n.a.

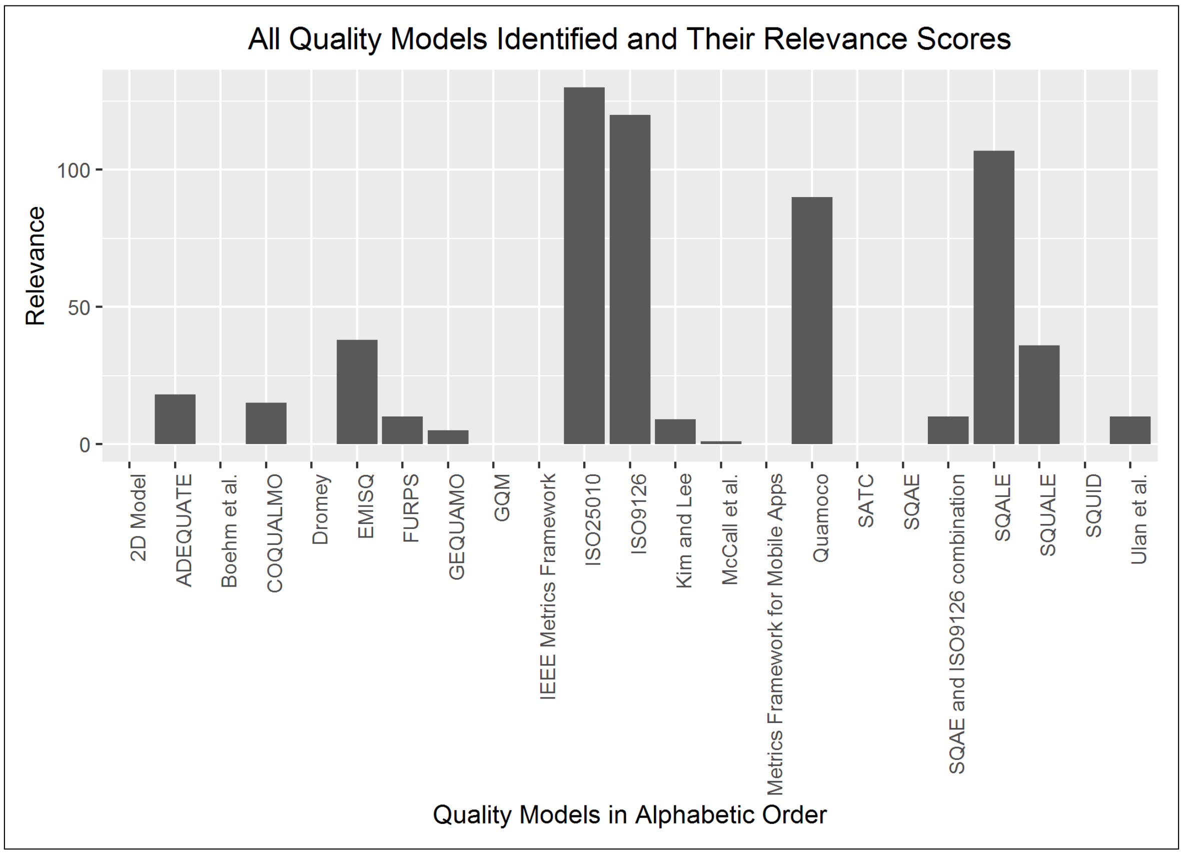

Figure 2.1: Relevance Scores of the Quality Model Families Including Manual and Automatic Searches, Source: [65]

Table 2.3: Relevance Scores of the Quality Model Families Excluding Manual Search, Source: [65]

No	Ranking	Model Class	Relevance
1	1	ISO25010	95
2	2	ISO9126	75
3	3	Quamoco	45
4	4	EMISQ	20
5	5	Ulan et al.	10
6	6	SQALE	9
7	7	McCall et al.	1
8	=8	Metrics Framework for Mobile Apps	0
9	=8	2D Model	0

This gives an illustration to which extent the quality models could be identified by the automatic search and to which extent the manual search contributed to the final list. Plain automatic search was not efficient enough to identify 14 quality model families. The quality models are also depicted with their relevance scores in figures 2.1 and 2.2, respectively.

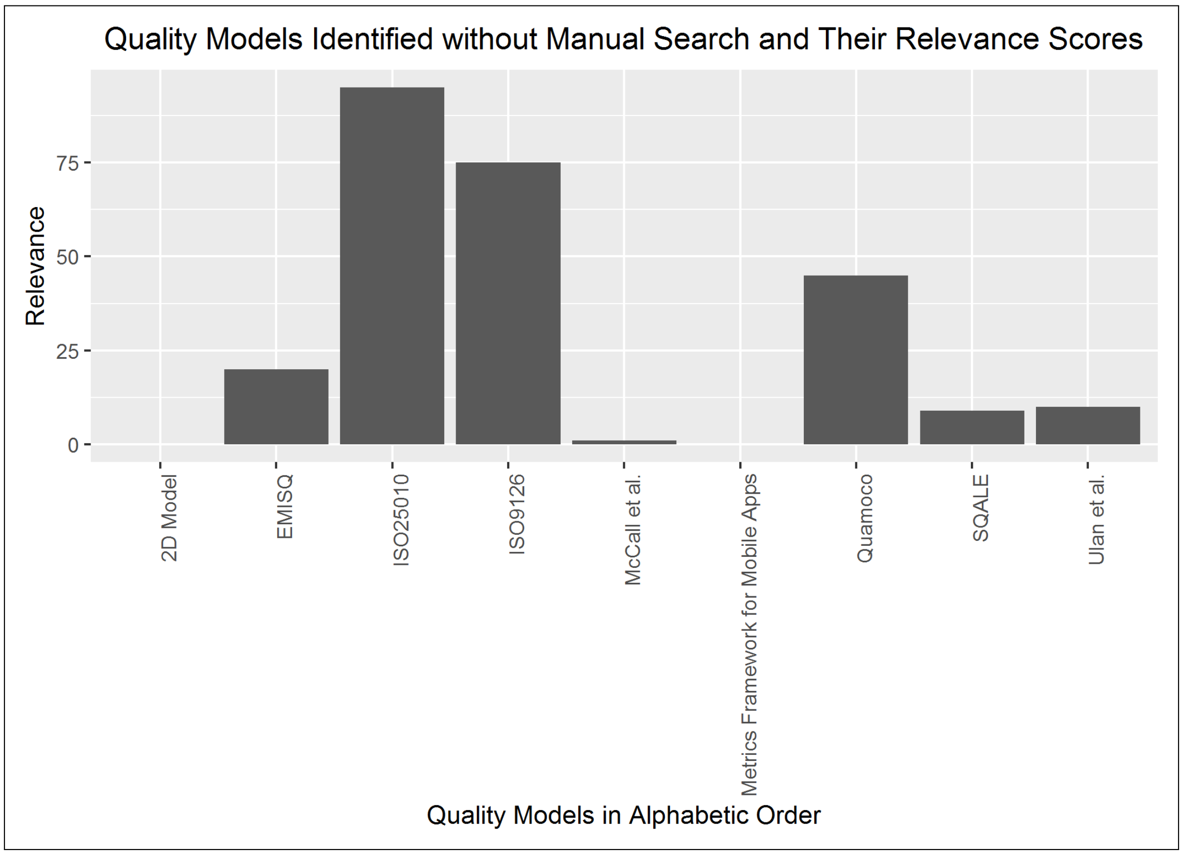

Figure 2.2: Relevance Scores of the Quality Model Families Excluding Manual Search, Source: [65]

The four relevance indicators, defined in section 2.2.6, assist to interpret how far the quality model family is accepted and cultivated by the scientific and industrial communities. The relevance score value encompasses publications and model definitions also from the industrial field such as EMISQ [141], SQUALE [183], FURPS [79, 80], SQAE and ISO9126 combination [38], Quality Model of Kim and Lee [128], which involve research from companies such as Air France, Siemens, Qualixo, IBM, MITRE, and Samsung, but the industrial popularity of the quality model families and their practical, everyday use cases are better approximated by the Google Relative Search Index. The relevance score value is in accordance with the Google Relative Search Index apart from two items in the ranking: FURPS [80] and GQM [239], which means that these two quality model families are far more widespread and presumably possess more applications than it could have been assumed based solely on the publications associated with them.

Google Relative Search Index indicates more activity for ISO/IEC 9126 [106] than for ISO/IEC 25010 [112], which shows that the practical use cases of ISO/IEC 9126 probably still exceed the applications of its successor standard ISO/IEC 25010 despite the amount of research associated with ISO/IEC 25010. The SQALE model [152] is ranked the third according to the relevance scores of the publications. This result is also in accordance with the Google Relative Search Index if the two outliers are ignored: FURPS [80] and GQM [239]. In addition, if SonarQube [222], a popular implementation of the SQALE model [152], is involved in the Google Relative Search Index, then it suppresses all other search indexes nearly to zero, which means that the most activity in the domain seems to be associated with the widespread implementation of the SQALE model. In addition, Quamoco [242] and EMISQ [141] quality model families comprise research but exhibit no search activities, which probably indicates less acceptance in everyday industrial settings. The quality model families marked with “n.a.” values in the Google Relative Search Index field indicate that the model name is also associated with other connotations; therefore this indicator cannot be used as measurement for the quality model families. Furthermore, the relevance score of the Quamoco quality model family [73, 242, 243, 244] is relatively high although its publication range, 2012-2015, is small, which indicates that this quality model does not show up in the focus of continuous research in a wider community.

As the introduced software product quality models indicate, the endeavour to support the quality assessment by tools and possibly automate it can be observed in the case of all the major quality models. By segmenting quality into internal, external, and quality in use views, only the internal view can be described and assessed by static code analysis. By definition [106, 112], the external view of quality deals with the software in execution and describes how the product relates to its environment, while the quality in use view describes the perception of the end user, which cannot fully be automated at present. Kim and Lee solved to automate the quality assessment based on a model derived from the ISO/IEC 9126 software product quality framework but they restricted the model to the internal quality view [128]. The inherent presence of quality views and how they affect the software product quality measurement and assessment are explained in chapter 3.

Letouzey and Coq created an individual model, SQALE, with focus on the development lifecycle of the software product and on the integration of the quality measurement and assessment into this lifecycle [152]. The implementation of the SQALE model is widespread due to its extensive tool support including SonarQube [180, 214, 222, 226, 260]. Nevertheless, SQALE is able to handle the internal quality view by means of static code analysis, which is a very strong restriction in comparison to the whole set of software product quality. Letouzey and Ilkiewicz anticipates to extend the SQALE model to include architecture analysis in the future [156].

Other quality models such as EMISQ, ADEQUATE require manual intervention to assess quality [93, 124, 141, 200, 201]. In addition, the SQUALE model applies metrics that can be computed automatically and metrics that require manual intervention to measure the external manifestation of quality [146, 183]. Quality models that measure and assess all known manifestations of software product quality are inherently not able to offer full automation potential at present. Tool support for the identified software product quality models is summarised in table 2.4.

Table 2.4: Tool Support of the Identified Software Product Quality Model Classes, Source: [65]

No.	Software Product Quality Model Classes, Names in Alphabetic Order	Tool Support
1	2D Model [262]	Unknown
2	ADEQUATE [93, 124]	Unknown
3	Boehm et al. [27]	Unknown
4	COQUALMO [26, 169]	Unknown
5	Dromey [45]	Unknown
6	EMISQ [141, 200, 201]	Unknown
7	FURPS [49, 79, 80]	Unknown
8	GEQUAMO [71]	In part
9	GQM [239]	Unknown
10	IEEE Metrics Framework Reaffirmed in 2009[99]	No
11	ISO25010 [44, 58, 97, 98, 112, 167, 192, 219, 235]	Unknown
12	ISO9126 [10, 36, 94, 106, 120, 160, 164, 196, 240]	Unknown
13	Kim and Lee [128]	In part
14	McCall et al. [20, 179]	Unknown
15	Metrics Framework for Mobile Apps [59]	Unknown
16	Quamoco [73, 242, 243, 244]	Yes
17	SATC [96]	Unknown
18	SQAE [173]	Unknown
19	SQAE and ISO9126 combination [38]	Unknown
20	SQALE [91, 150, 151, 152, 153, 154, 155, 156]	Yes
21	SQUALE [16, 100, 146, 183]	Yes
22	SQUID [132]	Experimental Toolset
23	Ulan et al. [235]	Unknown

2.3.2 Execution Tracing Quality in the Identified Software Product Quality Models

While conducting the analysis of the identified publications, maintainability and its sub-property analysability were also considered to answer research question RQ2. As table 2.5 shows, none of the identified 23 software product quality model classes handle the quality of execution tracing or logging quality in an adequate manner. The SQUALE model [16] is the only one that addresses the quality of execution tracing explicitly to assess whether log messages on three severity levels are traced: (1) errors, (2) warnings and (3) infos; however, having severity levels is only one possible aspect of execution tracing quality. In addition, the ISO/IEC 25010 standard family [112, 113] defines quality measure elements (QMEs) which use the word "log" in their naming but their meanings deviate from execution tracing or software logging. Furthermore, the quality model of ISO/IEC 25010 standard family does not consider execution tracing but defines measures which show similarities or overlap with execution tracing quality to some extent even if they are different [113]: (1) user audit trail completeness (ID: SAc-1-G), (2) system log retention (ID: SAc-2-S), (3) system log completeness (ID: MAn-1-G), (4) diagnosis function effectiveness (ID: MAn-2-S), (5) diagnosis function sufficiency (ID: MAn-3-S). Moreover, the predecessor standard ISO/IEC 9126 encompasses metrics which differ from execution tracing quality but which are related to it to some degree [107, 108]: (1) activity recording, (2) readiness of diagnostic functions, (3) audit trail capability, (4) diagnostic function support, (5) failure analysis capability, (6) failure analysis efficiency, and (7) status monitoring capability. The defined metrics and measures are vague and difficult or impossible to compute in industrial applications. More information on the listed metrics, measures and QMEs is provided in appendix B.1.

Table 2.5: Quality of Execution Tracing or Logging in the Identified Software Product Quality Model Classes, Source: [65]

No.	Software Product Quality Model Classes, Names in Alphabetic Order	Execution Tracing or Logging Quality
1	2D Model [262]	No
2	ADEQUATE [93, 124]	No
3	Boehm et al. [27]	No
4	COQUALMO [26, 169]	No
5	Dromey [45]	No
6	EMISQ [141, 200, 201]	No
7	FURPS [49, 79, 80]	No
8	GEQUAMO [71]	No
9	GQM [239]	No
10	IEEE Metrics Framework Reaffirmed in 2009[99]	No
11	ISO25010 [44, 58, 97, 98, 112, 167, 192, 219, 235]	No but related measures are defined.
12	ISO9126 [10, 36, 94, 106, 120, 160, 164, 196, 240]	No but related metrics are defined.
13	Kim and Lee [128]	No
14	McCall et al. [20, 179]	No
15	Metrics Framework for Mobile Apps [59]	No
16	Quamoco [73, 242, 243, 244]	No
17	SATC [96]	No
18	SQAE [173]	No
19	SQAE and ISO9126 combination [38]	No
20	SQALE [91, 150, 151, 152, 153, 154, 155, 156]	No
21	SQUALE [16, 100, 146, 183]	Yes, in part.
22	SQUID [132]	No
23	Ulan et al. [235]	No

2.3.3 Software Product Quality Models with Terminologies and Concepts

This section reports the extracted and synthesized data. Each software product quality model family identified with a relevance score value greater than zero is explained with its terminology and concepts, while the software product quality models yielding a relevance score value zero are introduced in a more concise way.

Distinct software product quality model families use homonyms, i.e. the same names for different things, as discussed in chapter 3.In addition, synonyms can also be found among the terminologies. Therefore, the unified summaries of the terminologies and concepts of the identified software product quality models contribute to the clarification of the problem domain significantly, which carries scientific value.

Software quality is an abstract concept. The first software product quality models, including the popular ISO/IEC 9126 and ISO/IEC 25010 standards, apply a hierarchical approach to deal with complexity, i.e. these models decompose abstract entities in a hierarchic manner as far as they become tractable units, and then metrics are assigned to those decomposed units to assess each part in a quantitative manner. Indeed, the metrics or the scores of the decomposed units are aggregated to obtain a higher-level score in the hierarchy. Latter frameworks, including Quamoco, create complex meta-models that define the relationship of the internal entities of the quality model. Burgues et al. in [28] define a framework to (1) compare and analyse existing quality model approaches, (2) define new models or (3) adapt the existing models to new concepts. A distinction is made between two types of entities in Burgues et al.'s framework: (1) generic model to describe the fundamental concepts of quality assessment, and (2) reference model to establish a particular application of the generic model in a specific domain. In addition, they also demonstrate the use of the framework to create a quality reference model for evaluating external libraries, which can be regarded as a specific case of Commercial-of-the-Shelf (COTS) software selection.

Software product quality in the 1970's was not described by an aggregated individual indicator but by different high-level quality properties [27, 179]. This trait of the software product quality models remained unchanged during the years with the exception of Ulan et al.'s model [235]. The reason for developing quality models in this manner is explained by the nature of the different high-level quality properties, which sometimes possess conflicting goals. Execution performance and maintainability, which usually emerge in all software product quality models in some form, are worth mentioning as examples of these high-level quality properties. If the maintainability quality property shows adequate or good values, then the software product possesses a sophisticated execution tracing mechanism, which can log the internal state changes and the routes of the threads of execution within the software. The more accurate information one has in the log files, the easier it is to identify and locate a potential error in the software product. On the other hand, logging all the necessary data about the threads of execution and the internal state changes in the software requires effort during the execution, which deteriorates the performance. To possess a better execution performance, it would be desirable to deactivate execution tracing. Thus, the quality properties execution performance and maintainability have contradictory quality targets. If a quality model applies hierarchic decomposition to deal with abstraction, then the subordinate quality properties might also have such contradictory quality targets.

ISO/IEC 25010 Software Product Quality Framework

Publications identified: [44, 58, 97, 98, 112, 167, 167, 192, 219, 219]

Score value: 130

Time range of the publications identified: [2011; 2018]

The ISO/IEC 25010 standard issued in 2011 silghtly revises the quality model of the ISO/IEC 9126 standard; moreover, the ISO/IEC 25000 standard family supersedes the ISO/IEC 9126 and ISO/IEC 14598 standard families [112]. The ISO/IEC 25012 introduces a new model for data quality to support the compliance with the data requirements, including legislation [111]. Consequently, the new ISO/IEC 25000 standard family defines three different quality models [111, 112]: (1) quality in use model for defining requirements and assessing user satisfaction, (2) product quality model for defining and assessing quality of software products, and (3) a data quality model for defining and assessing data quality.

The ISO 25000 Certification Portal gives a concise explanation of the different divisions in the ISO/IEC 25000 standard family [115]:

ISO/IEC 25020: "Measurement reference model and guide: Presents introductory explanation and a reference model that is common to quality measure elements, measures of software product quality and quality in use. Also provides guidance to users for selecting or developing, and applying measures."

ISO/IEC 25021: "Quality measure elements: Defines a set of recommended base and derived measures, which are intended to be used during the whole software development life cycle. The document describes a set of measures that can be used as an input for the software product quality or software quality in use measurement."

ISO/IEC 25022: "Measurement of quality in use: Describes a set of measures and provides guidance for measuring quality in use."

ISO/IEC 25023: "Measurement of system and software product quality: Describes a set of measures and provides guidance for measuring system and software product quality."

ISO/IEC 25024: "Measurement of data quality: Defines quality measures for quantitatively measuring data quality in terms of characteristics defined in ISO/IEC 25012."

Terminology

The ISO/IEC 25010 standard kept the internal and external quality property categories but introduced small changes in the terminology of the ISO/IEC 9126 standard. The differences are listed below and definitions are provided for the new entities:

Quality Measure Element (QME): QME instantiates a measurable property of quality defined in ISO/IEC 25021 [102].

Quality Measure (QM): QM contains one or more quality measure elements and a measurement function to compute with. It is similar to the term metric in the ISO/IEC 9126-1:2001 standard [106]. An initial list of quality measures was taken over from ISO/IEC 9126-2:2003 [107], ISO/IEC 9126-3:2003 [108], and ISO/IEC 9126-4:2004 [109].

Other entities in the ISO/IEC 25010 standard [112] provide the terminology of the ISO/IEC 9126 standard [106] as laid out in section 2.3.3.

Concepts

ISO/IEC 25010:2011 [112] defines a new software product quality model, which merges the internal and external models of the ISO/IEC 9126 standard family [106], in addition to revising the quality in use model of the ISO/IEC 9126 standard. However, ISO/IEC 25010:2011 [112] keeps the concepts laid down by the previous ISO/IEC 9126-1:2001 standard [106]. In the course of revising ISO/IEC 9126 [106], naming of the quality characteristics and sub-characteristics were elaborated; moreover, ISO/IEC 25010 introduced new characteristics, sub-characteristics and some further amendments listed below[112]:

1. The scope of quality models was extended to include computer systems as an actor, which implicates that the quality in use model can be interpreted from the perspective of a computer system.

2. New characteristics have been introduced:

   Quality in use model:

   (a) Characteristic context coverage was added with the sub-characteristics: (1) completeness and (2) flexibility.

   Product quality model:

   (a) Characteristic security was added with the sub-characteristics: (1) confidentiality, (2) integrity, (3) non-repudiation, (4) accountability and (5) authenticity. Security was a sub-characteristic of the characteristic functionality in the ISO/IEC 9126 standard [106].
   (b) Characteristic compatibility was added with the sub-characteristics: (1) interoperability and (2) co-existence.
   (c) New sub-characteristics were introduced in the model: (1) functional completeness, (2) capacity, (3) user error protection, (4) accessibility, (5) availability, (6) modularity, and (7) reusability

3. The internal and external quality models have been merged into one product quality model but the view of internal and external quality remained and is reflected by the internal/external quality characteristics and internal/external measures as depicted in Figure C.2 Quality in [112].

The ISO/IEC 25010 standard supports the following activities [112]:

1. Identifying requirements of the software.
2. Validating the requirements identified.
3. Identifying objectives of the system design.
4. Identifying objectives of the system test.
5. Defining the quality control criteria for the quality assurance.
6. Defining the acceptance of criteria for the software product.
7. Defining quality measures linked to quality attributes in support of the above activities.

The principles of the quality models of the ISO/IEC 25010 standard [112] are the same as the principles of the ISO/IEC 9126 standard [106]; moreover, the quality characteristics, sub-characteristics and quality attributes show only minimal deviations, listed above. Consequently, the ISO/IEC 25010 standard [112] is a linear development of the ISO/IEC 9126 standard [106].

The definitions of the two characteristics that extend the ISO/IEC 9126 product quality model are [112]:

Security: “The degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization.”

Compatibility: “The degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions, while sharing the same hardware or software environment.”

The definition of the other quality characteristics are in section 2.3.3, devoted to the ISO/IEC 9126 quality model.

The definition of the product quality model comprises the following characteristics and sub-characteristics [112].

1. Functional suitability: (a) Functional completeness; (b) Functional correctness; (c) Functional appropriateness.
2. Performance efficiency: (a) Time-behaviour; (b) Resource utilisation capacity.
3. Compatibility: (a) Co-existence; (b) Interoperability.
4. Usability: (a) Appropriateness; (b) Recognisability; (c) Learnability; (d) Operability; (e) User error protection; (f) User interface aesthetics; (g) Accessibility.
5. Reliability: (a) Maturity; (b) Availability; (c) Fault tolerance; (d) Recoverability.
6. Security: (a) Confidentiality; (b) Integrity; (c) Non-repudiation; (d) Accountability; (e) Authenticity.
7. Maintainability: (a) Modularity; (b) Reusability; (c) Analysability; (d) Modifiability; (e) Testability.
8. Portability: (a) Adaptability; (b) Installability; (c) Replaceability.

The quality in use model did not possess sub-characteristics in the ISO/IEC 9126 standard. For this reason all the characteristics, including their definition and sub-characteristics, are listed below [112]:

Effectiveness: "Accuracy and completeness with which users achieve the specified goals."

Efficiency: "Resources expended in relation to the accuracy and completeness with which users achieve goals."

Satistfaction: "The degree to which the users' needs are satisfied when a product or system is used in a specific context of use."

- **Usefulness:** "The degree to which a user is satisfied with the perceived achievement of pragmatic goals, including the results of use and the consequences of use."
- **Trust:** "The degree to which users or stakeholders have confidence that a product or system will behave as intended."
- **Pleasure:** "The degree to which users obtain pleasure from fulfilling their personal needs."
- **Comfort:** "The degree to which the user is satisfied with the physical comfort."

Freedom from Risk: "The degree to which a product or system mitigates the potential risk to economic status, human life, health or the environment."

- **Economic risk mitigation:** "The degree to which a product or system mitigates the potential risk to financial status, efficient operation, commercial property, reputation or other resources in the intended context of use."
- **Health and safety risk mitigation:** "The degree to which a product or system mitigates the potential risk to people in the intended context of use."

Environmental risk mitigation: "The degree to which a product or system mitigates the potential risk to the property or environment in the intended context of use."

Context coverage: "The degree to which a product or system can be used with effectiveness, efficiency, freedom from risk and satisfaction in both specified context of use and in contexts beyond those explicitly identified."

Context completeness: "The degree to which a product or system can be used with effectiveness, efficiency, freedom from risk and satisfaction in the specified context of use."

Flexibility: "The degree to which a product or system can be used with effectiveness, efficiency, freedom from risk and satisfaction in contexts beyond those initially specified in the requirements."

Detailed list of the predefined quality measures, quality measure elements, and quality measurement reference model are published in [101, 102, 103, 113]. The software product quality model and quality in use model of the ISO/IEC 25010 standard are independent of programming languages and programming paradigms. Like its predecessor ISO/IEC 9126 standard, ISO/IEC 25010 allows extensions and changes in the quality model; moreover, the standard motivates the users to tailor the quality models to the needs of the specific projects. Further information on tailoring the ISO/IEC 25010 [112] quality model can be found in [63, 64].

ISO/IEC 9126 Software Product Quality Framework

Publications identified: [10, 36, 94, 106, 120, 160, 164, 196, 240]

Score value: 120

Time range of the publications identified: [2001[2]; 2017]

The quality framework of the ISO/IEC 9126 standard family constitutes a hierarchic model for the definition and assessment of software product quality. The hierarchy consists of three levels. The quality properties at the highest level are called quality characteristics, the subordinate quality properties at the second level of the hierarchy are called quality sub-characteristics and the quality properties at the third level of the hierarchy are named quality attributes. The quality metrics are associated with the quality attributes. The relationship in the hierarchy is $1-n$, i.e. one quality characteristic can have many quality sub-characteristics, one quality sub-characteristic can have many quality attributes. A quality metric measures exactly one quality attribute; and a quality attribute can only have one metric assigned. In this sense a quality metric represents a measurable property of an attribute. This association represents a link between an abstract entity and a measurable property. In addition, each entity may belong to only one element of the hierarchy above. The reality, however, does not always reflect this mapping, one quality attribute could also be associated to more quality sub-characteristic as Appendix A describes in [106].

The metric definition of ISO/IEC 9126-2 [107] and ISO/IEC 9126-3 [108] standards lists the ISO/IEC 9126 [106] predefined metrics. From the point of view of the ISO/IEC 9126 software product quality model, these metrics are not coupled to the sub-characteristics but to the quality attributes which are linked to the quality sub-characteristics [106]; however, the metric definition tables do not this in an explicit manner.

[2]The first version of ISO/IEC 9126 appeared in 1991 but the range of the search included publications from 2000 only

The standard delineates three different views of the quality: (1) external, (2) internal, and (3) quality in use view. These three different views define three models, which show predictive validity with each other, i.e. internal quality predicts the external quality and external quality predicts the quality in use. Figures 2.3 and 2.4 depict these models.

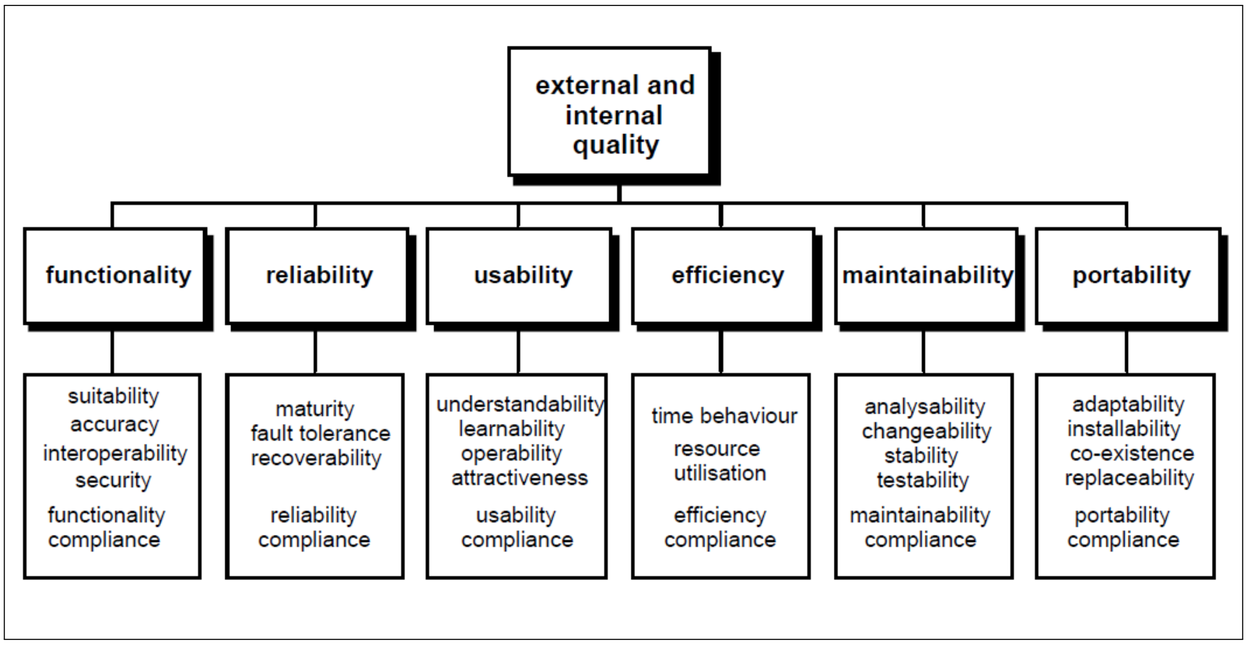

Figure 2.3: ISO/IEC 9126 External and Internal Quality Model, Source: [106]

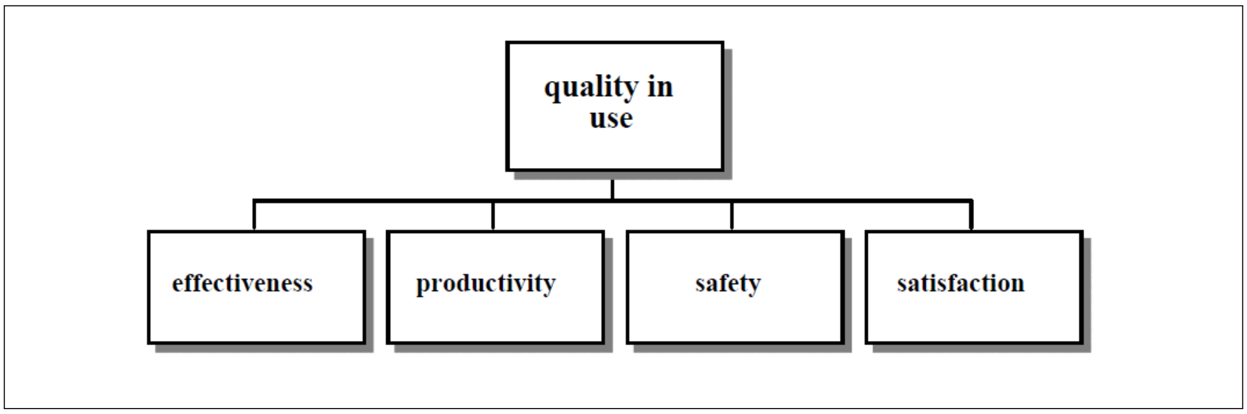

Figure 2.4: ISO/IEC 9126 Quality in Use Model, Source: [106]

Terminology

Quality Characteristics: High-level quality properties located at the top of the hierarchy. In the terminology of ISO/IEC 14598 standard family, they are named attributes.

Quality sub-characteristics: Quality properties that are located at the second level of the hierarchy; sub-characteristics are assigned to a higher level characteristic.

Quality Attributes: Quality properties that are located at the third level of the hierarchy. Quality attributes are always assigned to a higher-level sub-characteristic.

Quality metrics: Definition of the measurement method and measurement scale of quality properties. Quality metrics are assigned to quality attributes.

Internal quality metrics: Metrics whose inputs are formed by the intrinsic properties of the software product.

External quality metrics: Metrics that cannot be measured directly but derived when the software product operates.

Quality of use: The quality perceived by the user.

The first version of the standard issued in 1991 was superseded ten years later by the version ISO/IEC 9126-1:2001 [106]. The description of software evaluation was moved from the second version to the standard ISO/IEC 14598 [105] which introduced some inconsistencies in the definition of quality attribute and quality characteristic. The standard ISO/IEC 25010:2011 [112] revised the quality models described by the ISO/IEC 9126 [106] standard family and endeavours to unify also the definition of terms [112].

Concepts

The ISO/IEC 9126-1:2001 standard defines three basic views of the quality: (1) internal view, (2) external view, and (3) the view of the user as mentioned above. Internal view of the quality is manifested by the quality measured by the internal quality metrics. This reflects the quality of the source code and documentation. The internal quality metrics offer the possibility of static code analysis, which can be automated and integrated in the development pipeline. In addition, internal metrics support the quality assessment even if the software product is not operational. The external view of the quality is constituted by the external quality metrics. It shows how the product relates to its environment for which executing and testing the software product is mandatory. The external quality metrics cannot be measured based on static artefacts as source code or documentation. The user's view of the quality is illustrated by the quality in use reflected by the quality in use metrics. Predefined internal, external and quality in use metrics of the framework were published in [107, 108, 109] in 2003 and 2004. The model allows extensions.

Internal and external metrics need to be in cause-effect relationship or they need to correlate with each other to satisfy the expected predictive validity condition, i.e. from the values of the internal metrics conclusions can be drawn for the values of the external metrics and through that for the external quality of the software.

The software product quality model introduces six high-level characteristics: (1) functionality, (2) reliability, (3) usability, (4) efficiency, (5) maintainability, and (6) portability. These characteristics have an internal and external variant to form an internal and external model. The standard defines the six high-level characteristics in the following manner [106]:

Functionality: "capability of the software product to provide functions meeting stated and implied needs when the software is used under specified conditions."

Reliability: "capability of the software product to maintain a specified level of performance when used under specified conditions."

Usability: "capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions."

Efficiency: "capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions."

Maintainability: "capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications."

Portability: "capability of the software product to be transferred from one environment to another. The environment may include organisational, hardware or software environment."

The quality in use model has four high-level characteristics without sub-characteristics: (1) effectiveness, (2) productivity, (3) safety, and (4) satisfaction. The external quality model needs to demonstrate predictive validity for the quality in use model, i.e. the quality indicated by the external quality model anticipates the quality measured by the quality in use model. The standard defines the four high-level characteristics in the following manner [106]:

Effectiveness: "capability of the software product to enable users to achieve specified goals with accuracy and completeness in a specified context of use."

Productivity: "capability of the software product to enable users to expend appropriate amounts of resources in relation to the effectiveness achieved in a specified context of use. Relevant resources can include time to complete the task, the user's effort, materials or the financial cost of usage."

Security: "capability of the software product to achieve acceptable levels of risk of harm to people, business, software, property or the environment in a specified context of use."

Satisfaction: "capability of the software product to satisfy users in a specified context of use. Satisfaction is the user's response to interaction with the product, and includes attitudes towards use of the product."

The ISO/IEC 9126 standard [106] does not only allow extensions in the quality model but it even emphasizes them by motivating the users to tailor the quality models to the needs of the specific projects. Galli et al. published an in-depth analysis of the ISO/IEC 9126 and ISO/IEC 25010 software product quality models in addition to the investigation of the extension possibilities of the frameworks to incorporate execution tracing quality in the ISO standards [63, 64]. The software product quality model and quality in use model of the ISO/IEC 9126 standard are independent of programming languages and programming paradigms.

SQALE Quality Model

Publications identified: [91, 150, 151, 152, 153, 154, 155, 156]

Score value: 107

Time range of the publications identified: [2009; 2016]

SQALE is an acronym for Software Quality Assessment Based on Lifecycle Expectation [152]. SQALE was motivated by the shortcomings of the ISO/IEC 9126 [106] and Bohm's model [27], which do not give precise instructions how to compute the aggregated quality metrics for the identified quality properties [155]; moreover, the philosophy of Dromey's quality model [45] also formed its initial concepts. The model was created by Letouzey and Coq and first published in 2009 [154], followed by two more publications in 2010 [150, 155]. SQALE comprises two models: (1) a quality model and (2) an analysis model. The main goals of SQALE are: (1) to identify deficiencies in the source code of an application and (2) to associate these deficiencies with an estimated effort needed for their correction, and (3) to set priorities for the correction of the deficiencies identified. SQALE achieves these goals with a precise model that became popular in the software development community for its easy-to-automate features, i.e. many tools support its integration in the development pipeline [180, 214, 222, 226, 260]. Nevertheless, SQALE is able to handle only the internal quality view, based on the static analysis of the source code. Letouzey and Ilkiewicz express as a future possibility to extend the model to include architecture analysis [156].

Terminology

Technical Debt: The term was first used by Ward Cunningham in 1992 to express: (1) "neglecting design is like borrowing money", (2) "refactoring is like paying off the technical debt", (3) "developing slower because this debt exists is like paying an interest on the loan", furthermore, (4) "all time spent on code that was not developed in the right manner is like paying an interest on the technical debt" [151]. Technical debt and its impact on the development and maintenance costs is depicted in figure 2.5

Remediation Costs: The cost to correct the technical debt. It can be expressed in monetary or in time units.

SQALE Index or SQALE Quality Index or SQI: A SQALE index represents the technical debt.

Rating: It is an ordinal scale from A to E where the ratio of the technical debt and the development costs is categorised. A is the best rating, E is the worst rating.

SQALE Pyramid: It is used to depict the distribution of the technical debt over the quality properties of the quality model [151]. In addition, the pyramid defines the remediation priorities.

SQALE Indicators: SQALE names three indicators: (1) SQALE rating, (2) SQALE indices and (3) the Kiviat diagram to illustrate the SQALE index values of each selected quality property [152].

Concepts

The goal set by the SQALE approach is to estimate and manage the technical debt. Letouzey and Ilkiewicz define the actions below as a minimum to be able to manage the technical debt [156]:

1. Establishing a list to describe the bad coding habits that contribute to technical debt.
2. Defining an estimation model that transforms the identified non-compliances into technical debt.
3. Setting the target for the acceptable level of technical debt in the specific project.
4. Assessing the technical debt frequently enough to be able to respond to the changes quickly.
5. Analysing the technical debt identified to understand its potential impact and to make informed decisions.

6. If the technical debt exceeds the target defined, then fixing non-compliances is necessary to return to the acceptable range.
7. Analysing technical debt in correlation with other information including the quality perceived by the end user.
8. Establishing processes and introducing tools in the development pipeline to support proactive technical debt management through automation.

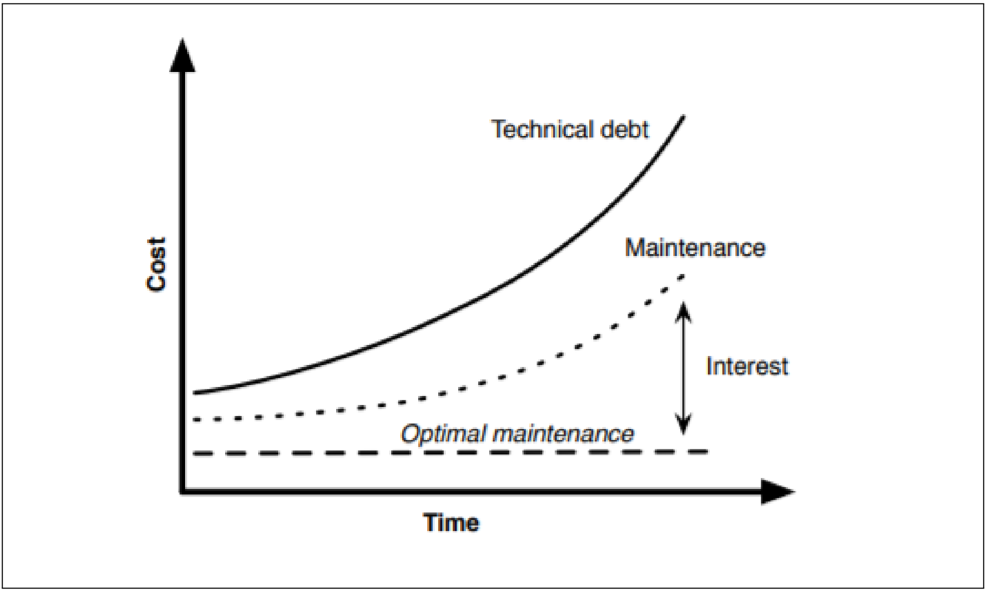

Figure 2.5: Unresolved Technical Debt and Its Interest, Source: [187]

Consequently, the estimation of the technical debt is implemented by the following steps in SQALE[156]:

1. The organisation or specific project needs to create a list of non-functional requirements that define the right code. These requirements need to be verifiable and tangible. Any deviation from the defined list creates a technical debt called non-compliance in the wording of SQALE.
2. Using this list of non-functional requirements, SQALE expects the organisation or specific project to define a technical debt estimation model. This implicates associating a remediation function with each requirement to transform the non-compliances into technical debts.
3. The source code is examined by analysis tools, which use the remediation functions to compute the remediation costs for each non-compliance. The remediation costs of all non-compliances are summed up to calculate the technical debt.

Quality Model: The SQALE quality model is a hierarchic model that organises the non-functional requirements, and expectations towards the software product in nine high-level groups. These groups are named quality characteristics following the terminology of the ISO/IEC 9126 and ISO/IEC 25010 standards. Each characteristic is further decomposed into sub-characteristics and the sub-characteristics are associated with one or more source-code-level requirements. The majority of the quality characteristics stem from ISO/IEC 25010 [112] as stated in the V1.1 definition document [152] but with slightly different characteristic names. Hegeman provides a more detailed explanation on the naming regarding SQALE version 0.8 [91]. In addition to the mentioned concepts of the ISO standards [106, 112], SQALE introduces an order of priority of the characteristics with regard to the software life-cycle, which means that a non-compliance related to a quality characteristic has an impact on all other quality characteristics with lower priority. We list here the nine high-level characteristics[3] named in the definition document V1.1 in the order of their priorities [152]: (1) testability, (2) reliability, (3) changeability, (4) efficiency, (5) usability, (6) security, (7) maintainability, (8) portability and (9) reusability. Consequently, a non-compliance related to the testability characteristic impacts all the other characteristics in the life-cycle of the software.

Analysis Model: Each non-compliance of the quality requirements defined by the quality model causes a technical debt that implicates costs. The analysis model describes how these costs are computed. The cost can be expressed in time with regard to the time of the necessary correction or in money if an hourly rate for the correction time is assumed. Thus, each quality requirement violation needs to be associated with a (1) remediation function showing the costs to locate and correct the deviation; and a (2) non-remediation function to show the cost of the present non-compliances including (a) cost of additional maintenance resources, and (b) additional non-compliance related resources such as CPU or memory. The non-remediation function estimates all the potential cost that the product owner or product manager can claim for accepting the requirement violations, i.e. the poor quality. If the compensation amounts less than the cost of the non-remediation estimated, then the product owner or product manager should not accept the product [156]. However, it is difficult to estimate the financial consequences of a quality requirement violation. Consequently, as an alternative, the non-compliances can be ranked into classes of severity and impact: blocker, critical, major, minor. Costs can be assigned to each category in a symbolic manner to indicate their relative importance [156]. In addition, since the SQALE model can only capture the internal quality view as explained in section 2.3, it is not able to assess the quality and its financial impacts on the software product as a whole including runtime behaviour and the end user's perception.

The analysis model consists of the (1) normalisation of the metrics associated with the source-code-level requirements, and (2) the aggregation of the metrics including two hierarchies: (a) hierarchy of the metrics, and (b) hierarchy of the source-code-level constructs from components to classes [155]. Letouzey and Coq in [155] conclude that the fusion of metrics for a higher level in the hierarchy via weighted averages does not comply with measurement theory's representation condition [152], and they introduce use cases with masking effects, compensation effects, and threshold effects for a variety of fusion operators to illustrate how the underlying values in the hierarchy are hiden [155]. To satisfy the representation condition, SQALE uses a common scale: the technical debt; moreover, it aggregates metrics from a lower level in the hierarchy to a higher level with the sum operation. The most cost-efficient error resolution strategy can be achieved by starting to resolve the non-compliances with the lowest technical debt and with the highest business impact i.e. with the highest non-remediation value [155].

SQALE Indices: SQALE indices manifest costs. They can be computed for any level of the source code hierarchy and for any quality characteristic [155]. The most important index is the technical debt while the other indices, business impact index and consolidated index, support the analysis of the technical debt[152].

SQALE defines the following quality characteristics, named characteristic indices [152]: 1. SQALE Testability Index: STI; 2. SQALE Reliability Index: SRI; 3. SQALE Changeability Index: SCI; 4. SQALE Efficiency Index: SEI; 5. SQALE Usability Index: SUI; 6. SQALE Security Index: SSI; 7. SQALE Maintainability Index:

[3]ISO/IEC 25010 defines eight high-level quality characteristics.

SMI; 8. SQALE Portability Index: SPI; 9. SQALE Reusability Index: SRuI.

SQALE Index Densities: SQALE indices divided by the number of lines of code gives the index density [152]. Index densities computed for source code modules show where the technical debt accumulates. Nevertheless, very-small-sized modules containing low amount of technical debt can produce high index densities, which requires care during the interpretation of the results [152].

SQALE Rating: SQALE rating [152] is an indicator on an ordinal scale from A to E that shows the ratio of the remediation and development costs as illustrated in Table 2.6. The average development costs of 1000 lines of code is estimated at 100 hours [152].

Table 2.6: SQALE Qality Rating Grid Definition, Source: [152]

Rating	Up to
A	1%
B	2%
C	4%
D	8%
E	infinite

SQALE method is easy to automate, it is precise and sensitive to measure the results of code refactorings or to compare the work of different teams [155]. Furthermore, the concepts are independent of programming paradigms and programming languages; however, the concrete, defined non-functional requirements for the source code and the non-compliance detection on the source-code-level depend on the programming paradigm and programming language in which the software product is implemented.

Quamoco Quality Model

Publications identified: [73, 242, 243, 244]

Score value: 90

Time range of the publications identified: [2012; 2015]

Quamoco software product quality model and assessment approach was published by Wagner et al. in 2012 [242]. The main goal of the new model creation can be described as constructing a widely applicable model with the view of quality as a whole, in addition to being able to easily provide quality assessments in concrete settings [242]. This motivation was supported by the shortcomings of software product quality frameworks which either provide abstract quality properties to cover each aspects of software product quality but formulate no or limited assessment possibilities, or implement concrete quality assessment methods for a specific domain or for a quality property or sub-property without the whole view of software product quality [242]. It is necessary to integrate both aspects because the first approach is difficult to apply in concrete quality assessment setting while the second misses the overall view of quality and focuses only on a part of it [242]. The Quamoco approach comprises a meta-model and a base model to implement the meta-model; moreover, it leans on the high-level quality characteristics of the ISO/IEC 25010 standard [112]. The base model supports tailoring, it can be subject to further refinements in a specific project or domain.

Terminology

Wagner et al. define the specific terms of the Quamoco quality model in [242, 243], which are concisely described here (see figure 2.6):

Entity: A thing or a part of a thing that is important from the point of view of quality.

Property: An attribute of an entity.

Factor: General quality property that can be decomposed in a hierarchic manner in sub-factors. Factor has two specific types: (1) the quality aspect and (2) the product factor.

Quality Aspect: A quality aspect defines an abstract quality property, which views the complete software product as its entity. It is similar to the high-level quality characteristic of the ISO/IEC 9126 and ISO/IEC 25010 quality models. Quality aspects can be decomposed in a hierarchic manner in sub-aspects.

Product Factor: A product factor is a measurable quality property of the software product. The product factors as being factors can be decomposed in a hierarchic manner in sub-factors.

Impact: The product factors influence the quality aspects. An impact is the description of such an influence.

Measure: A measure describes how a product factor needs to be measured. A product factor can have multiple measures.

Instrument: An instrument is a concrete implementation of a measure.

Utility: A utility describes the relative satisfaction of stakeholders with regard to a specific factor. For each measurable factor a utility function can be defined to map the factor's value to the utility value.

Evaluation: An evaluation can be assigned to a product factor to aggregate the results collected by the instruments.

Quality Meta-Model: The meta-model describes the relationship of the quality model elements.

Quality Base Model: It is a concrete implementation of the meta-model with a selection of factors and their related elements.

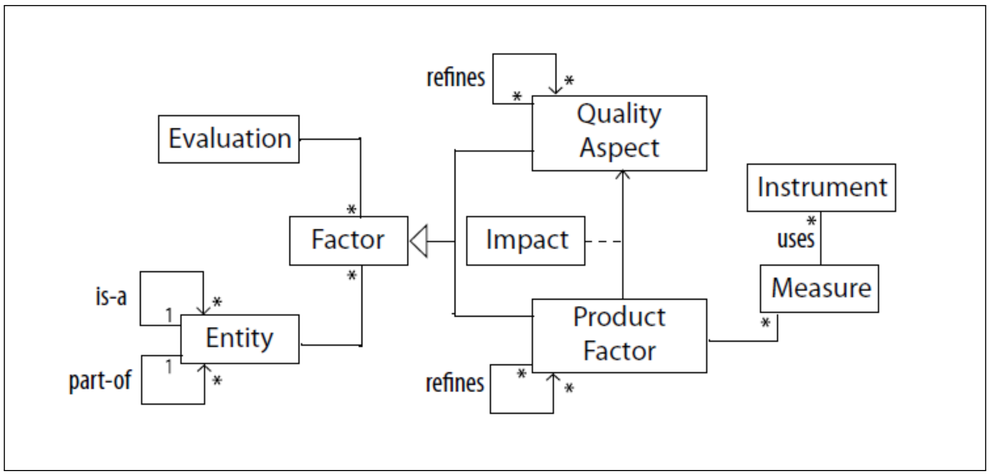

Figure 2.6: Quamoco Quality Meta-Model Showing the Relationship of the Model Elements, Source: [242]

Concepts

The base model is a concrete implementation of the meta-model and it contains [242]: (1) 112 entities, (2) 286 factors, from which (2a) 221 factors have evaluations assigned and (2b) 202 factors have defined impacts on other factors: altogether 492 impacts, (3) 526 measures associated to 542 instruments, from which (3a) 8 manual instruments and (3b) 536 tool-based instruments, which are implemented by static code analysers. The base model can be adapted to different problem domains with modularisation to avoid creating a single model including all model elements for all technologies [242]. A root module is available with general quality aspects, product factors and measures. The root module can be extended by other predefined modules as the module for object-orientation, Java or C# [242, 244]. These programming paradigm and language-specific modules comprise the quality elements for a particular problem domain. Furthermore, the quality model allows adaptations [136, 244].

The model provides tool support [42]: (1) to tailor the quality model to the specific needs and (2) a quality assessment engine that prepares files to report. The tool provided is based on the Eclipse Framework and it integrates the measurement results of static code analysers including PMD, FindBugs, Gendarme, and FxCop into the quality model.

The root model comprises the following high-level factors [204]: 1. Compatibility; 2. Functional suitability; 3. Maintainability; 4. Performance efficiency; 5. Portability; 6. Reliability; 7. Security; 8. Usability.

As a design decision, the quality characteristics of ISO/IEC 25010 standard [112] were selected to constitute the quality aspects of the Quamoco quality model. The associations between the product factors, instruments and the quality aspects are shown in table 2.7. In addition, it is possible to tailor the model to specific project needs with the selection of quality aspects, product factors and measures [136]. The separation of measures and instruments makes possible to integrate the automatic and manual assessments [242].

Table 2.7: Quamoco Base Model Elements Relating to the ISO/IEC 25010 Characteristics, Source: [244]

	Product Factors	Tool-Based Instruments	Manual Instruments
Maintainability	146	403	8
Functional Suitability	87	271	1
Reliability	69	218	0
Performance Efficiency	59	165	1
Security	17	52	0
Portability	11	20	0
Compatibility	0	0	0
Usability	0	0	0

The Quamoco approach defines the following requirements for quality assessment [242]: (1) it must be interpretable for decision makers, (2) it must be able to handle incomplete information, and (3) it must allow for contradictory quality aspects.

The model implements the following steps for quality assessment [242]: (1) all necessary measurement data are associated with measures, (2) measures are normalised to construct general scales that make possible to compare different software, (3) utility functions are defined to construct utility values by means of the measures, and (4) the utility values are aggregated with weighted averages in the hierarchy. The process is illustrated in figure 2.7. The weights applied in the model were derived from the results of a survey research conducted with mixed teams of industrial and academic experts [242]. The weights determine the relative importance of the elements in the hierarchy.

Gleirscher et al. in [73] investigated the application of the Quamoco quality model in small- and medium-sized software enterprises because such companies have strong resource constraints to avoid the expensive bug fixes or compensations that requires high efficiency in the quality assurance. Five projects at five different companies were investigated as a consequence of which the following findings were revealed: the integration of the Quamoco quality model in present projects requires little effort, usually less than 1 person-hour per project, (2) defects could be identified in the production code by means of the Quamoco model, and (3) the participating companies found the results helpful. Gleirscher et al. also indicated that the most effective way to assess quality is to combine several quality assessment methods: static analysis techniques can efficiently contribute to dynamic techniques because their application requires little manual effort.

The Quamoco software product quality model is not programming language independent due to the instruments and specific product factors applied in its complex meta-model but the concepts of the model, including also the meta model, the base model, and adaptation options, are independent of programming paradigms and programming languages.

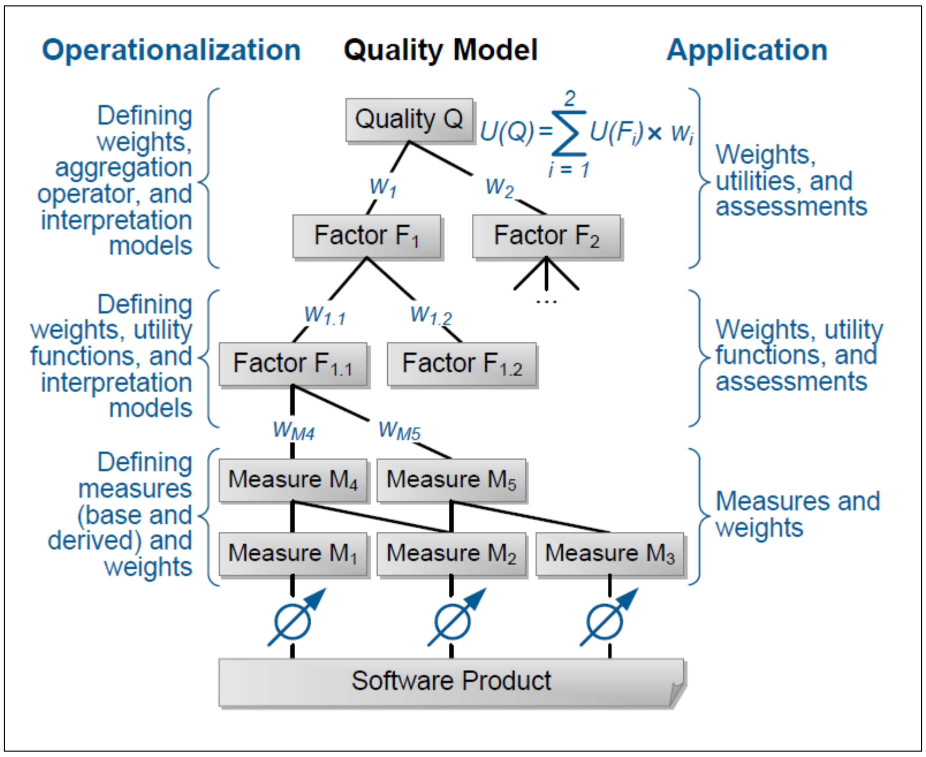
Operationalization
Quality Model
Application
Defining weights, aggregation operator, and interpretation models
Quality Q
$U(Q)=\sum_{i=1}^{2} U(F_i) \times w_i$
Weights, utilities, and assessments
w_1
w_2
Factor F_1
Factor F_2
...
Defining weights, utility functions, and interpretation models
$w_{1.1}$
$w_{1.2}$
Factor $F_{1.1}$
Factor $F_{1.2}$
Weights, utility functions, and assessments
w_{M4}
w_{M5}
Defining measures (base and derived) and weights
Measure M_4
Measure M_5
Measure M_1
Measure M_2
Measure M_3
Measures and weights
Software Product

Figure 2.7: Quamoco Quality Assessment Approach, Source: [242]

EMISQ Quality Model

Publications identified: [141, 200, 201]

Score value: 38

Time range of the publications identified: [2008; 2011]

The acronym EMISQ stands for Evaluation Method for Internal Software Quality, which means that the model solely focuses on the internal view of software quality [200]. The model creation was motivated by (1) the error-prone metric and rule violation computations of static code analysers, and (2) the consideration of the application context and the context of the code artefacts, i.e. no general rules can be applied for each application and for each code module of an application [200]. Therefore, the metrics collected by static code analysers are reviewed by experts and they judge whether a metric value really means a quality issue in the given context. EMISQ is a hierarchic quality model based on the ISO/IEC 9126 standard [106, 200].

Terminology

Quality Attribute: A high-level quality property [200] equivalent to the quality characteristic in the terminology of ISO/IEC 9126 standard.

Quality Sub-attribute: The high-level quality properties are decomposed into sub-properties, which are called sub-attributes [200]. A quality sub-attribute is equivalent to the quality sub-characteristic in the terminology of ISO/IEC 9126 standard [106].

Concepts

EMISQ mainly uses the concepts of the ISO/IEC 9126 quality model [106], in addition to principles laid down for quality assessment in the ISO/IEC 14598 standard [105] [200]. EMISQ, unlike the ISO/IEC 9126, does not provide a third layer for the quality hierarchy but links its metrics directly to the sub-attributes [200].

EMISQ quality model defines the following five attributes and sub-attributes:

Security: (1) integrity, (2) confidentiality, and (3) availability.

Efficiency: (1) time behaviour, (2) resource utilisation, and (3) efficiency conformance.

Reliability: (1) fault tolerance, (2) robustness, (3) runtime stability, (4) correctness, and (5) soundness.

Maintainability: (1) changeability, (2) testability, (3) maintainability conformance, (4) configurability, (5) simplicity, (6) structuredness, (7) readability, (8) documentation, and (9) craftsmanship.

Portability: (1) adaptability, (2) installability, and (3) portability conformance.

The attributes and sub-attributes are rated on an ordinal scale $\{ok, critical, very-critical\}$ [200]. The value of a sub-attribute is computed as the median of the rating of each metric and rule value assigned to the given sub-attribute; moreover, the value of an attribute is computed as the median value of its sub-attributes, after performing a conversion from ordinal scale to interval scale, but an expert can override this computation with regard to the context of application [200]. Each sub-attribute is also assigned an ordinal value of importance on the scale $\{low, medium, high\}$ to draw attention to the relevant items and deviations. An estimated effort is assigned to each sub-attribute gaining an assessment value not *ok* to achieve the *ok*-level. The assigned effort value is also expressed on an ordinal scale.

CQMM, which stands for Code Quality Monitoring Method, was published by Plösch et al. in 2010 [201]. Its application was presented and evaluated on 25 projects by Kothapalli et al. [141]. CQMM [201] describes how to integrate the EMISQ software product quality model in the software development lifecycle; moreover, it illustrates the use of the EMISQ model. The phases of integrating EMISQ in the software development lifecycle are shown in figure 2.8. The process has three major phases: (1) setup and tailor, (2) adjust and control, and (3) measure and enhance [201]. The EMISQ quality model is programming language independent and independent of programming paradigms.

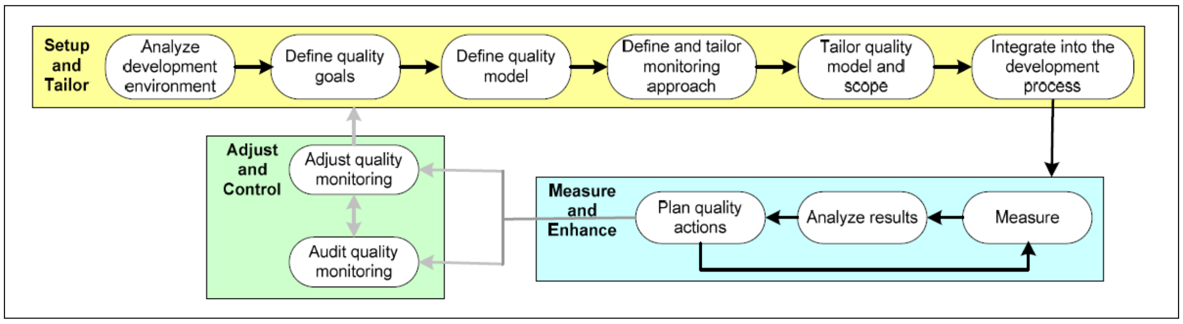
Setup and Tailor
Analyze development environment
Define quality goals
Define quality model
Define and tailor monitoring approach
Tailor quality model and scope
Integrate into the development process
Adjust and Control
Adjust quality monitoring
Audit quality monitoring
Measure and Enhance
Plan quality actions
Analyze results
Measure

Figure 2.8: CQMM, Phases of EMISQ Integration in the Software Development Lifecycle, Source: [201]

SQUALE Quality Model

Publications identified: [16, 100, 146, 183]

Score value: 36

Time range of the publications identified: [2012; 2015]

The hierarchic quality model, SQUALE, was designed and implemented by Qualixo and Air France in 2006, validated by Air France - KLM, PSA Peugeot-Citroen, placed in the public domain in 2008 and published in [146, 183]. SQUALE's motivation stems from the ISO/IEC 9126 standard [106], which makes possible to assess a high-level quality characteristic, although it is difficult to define targets for increasing the quality of a selected characteristic or sub-characteristic in its scope. SQUALE is also influenced by the factors-criteria-metrics model of McCall, Richards, Walter [179, 183]. SQUALE introduces a new level, practices, into the three-level hierarchy of the ISO/IEC 9126 software product quality model [106] (containing characteristics, sub-characteristics, and quality attributes).

In this sense, SQUALE introduces an intermediary level between sub-characteristics and attributes to form groups, which provide technical guidelines to follow, in order to increase a selected quality value of a sub-tree in the hierarchy[146, 183].

It is important to remark, that the metrics of the ISO/IEC 9126 software product quality model[106, 107, 108] are not coupled to the sub-characteristics but to the quality attributes, which are coupled to the quality sub-characteristics. ISO/IEC 9126 metrics [106, 107, 108] represent a measurable property of an attribute. This association represents a link between an abstract entity and a measurable property.

Terminology

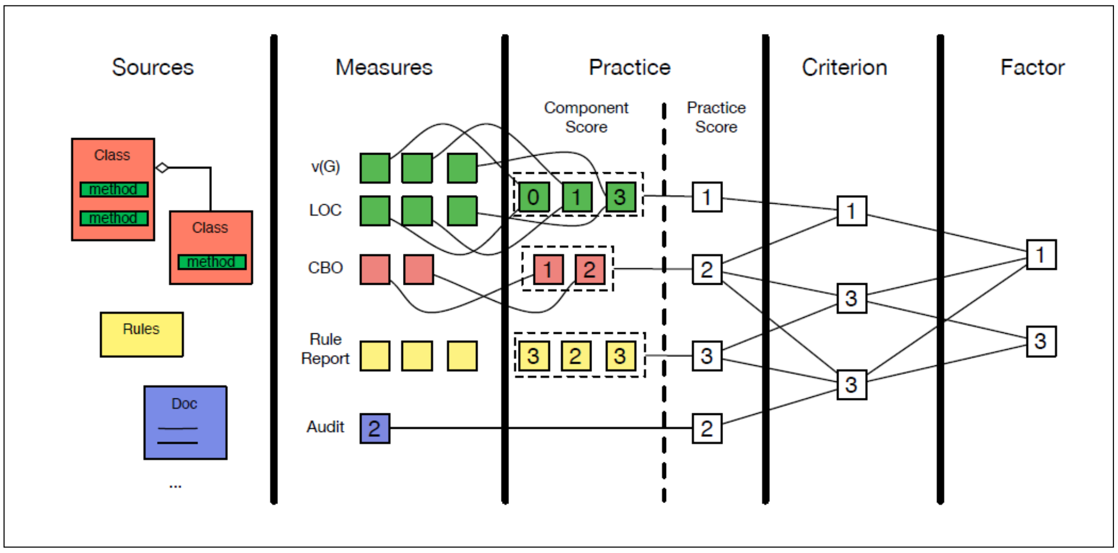

Figure 2.9: SQUALE Model Structure, Source: [16]

Factor: A factor is a high-level quality property. It is the equivalent of the quality characteristic in the terminology of ISO/IEC 9126 [106].

Criterion: A criterion is a quality sub-property, which is located on the second level of the hierarchy in the model.

Metric or measure: A metric or measure is a piece of information extracted form the project data such as lines of code, cyclomatic complexity [16]. Metrics have two types: (1) automatically computable metrics and (2) manually computable metrics.

Practice: A practice can be interpreted as a rule for the developers to follow to achieve optimum software quality. “A practice combines and weights different measures to assess the fulfillment of technical principles.” [16] Consequently, a practice can have one or more metrics associated.

Concepts

SQUALE is a quality model that organises the quality properties hierarchically with the levels: (1) metric, (2) practice, (3) criterion, and (4) factor [146, 183]. Metrics are associated with practices. Metrics can be grouped according to different point of views [16]: (1) metrics that can be computed automatically by tools form the source code, and (2) metrics that can only be computed by human intervention i.e. manually. In addition, group (1) can be decomposed into three further subgroups: (1a) internal metrics of the source code, (1b) metrics from the analysis of rule checkers such as naming conventions or syntactic rules, and (1c) dynamic metrics collected during the test runs[16]. In contrast to the ISO/IEC 9126 standard [106], one element in the quality hierarchy may also belong to more than one element on the next, upper level as shown in figure 2.9.

SQUALE defines six factors inspired by the ISO/IEC 9126 standard family [146, 183]:

Functional capacity: Functional capacity represents to what extent the functionalities offered by the software product satisfies the needs of the user.

Architecture: Architecture describes the technical architecture quality of the software product.

Maintainability: Maintainability represents the capability to correct errors in the software product.

Capacity to evolve: Capacity to evolve describes the capability to extend the software product.

Capacity to re-use: Capacity to re-use describes the capability to re-use the code artefacts of the software product.

Reliability: Reliability describes the stability of the software product.

Metric values can have an arbitrary range but the practices, criteria, and factors have a normalised range $[0;3]$. The computed practice values can be continuous or discrete. The value zero represents the failure to meet the quality objectives, while the value three represents the achievement of the quality objectives [16]. This score value also implicates an easy-to-interpret marking system illustrated by the manual practice functional specification: functional specification can be described with a mark zero if the functional specification does not exist, with a mark three if it exists and is consistent with the requirements, with a mark two if the functional specification exist but it is inconsistent with the requirements [16]. Along the hierarchy from bottom up, the values are aggregated with weighted averages.

Practices are assigned to each code construct including methods, classes, and packages. An individual score is computed and assigned to each practice, which is named practice mark. A global practice mark is computed based on the individual practice marks. The computation of the global practice mark can be performed in three different manners with weighted averages[16]:

Hard weighting: Hard weighting represents low acceptance for low individual practice marks and this kind of weighting deteriorates the global mark strongly to the range $[0;1]$ even if few poor individual practice marks appear in the set to be aggregated.

Medium weighting: Medium weighting represents medium acceptance for low individual practice marks and this kind of weighting deteriorates the global mark to the range $[0;1]$ only if on average poor individual practice marks appear in the set to be aggregated.

Soft weighting: Soft weighting represents high tolerance for low individual practice marks and this kind of weighting deteriorates the global mark to the range $[0;1]$ only if many, poor individual practice marks appear in the set to be aggregated.

Predefined practices of the SQUALE model are published in [16]. The strategy to resolve the quality defects and to estimate the effort for the defect resolution are defined in [100]. The SQUALE software product quality model is not programming language independent due to the language specific metrics and practices but the concepts are independent of programming paradigms and programming languages.

ADEQUATE Quality Model

Publications identified: [93, 124]

Score value: 18

Time range of the publications identified: [2005; 2009]

Khaddaj and Horgan defined an adaptable software product quality model in 2005, which they name ADEQUATE [124]. The motivation behind the model creation was (1) the easy adaptation to the local project needs, and (2) the possibility of comparison of different projects from the point of view of software quality. The model is a hybrid model as it uses product and process quality specific properties [124].

Terminology

Key Personnel: Those individuals who are deemed the most important from the point of view of the project.

Essential View: Each individual from the key personnel has a view on the software quality stemming from the expected use of the product. These views are named as essential views.

Key Quality Factors (KQF): Seven predefined quality factors that are important for all projects. They make also the cross-project quality comparison possible.

Locally Defined Factors (LDF): Quality factors that are important only for a particular project to satisfy the essential views on quality. The model definition does not provide a predefined set of LDFs but transfers the identification and definition of LDFs in the scope of each project depending on the essential views.

Concepts

The definition document does not decompose the quality factors into sub-factors nor specifies metrics. Nevertheless, marking principles are laid down for each key quality factor on an interval scale $[1;5]$ [93]. The published key quality factor definition contains the following factors [93]:

Maintainability: Ability of the software product to be modified.

Usability: Ability of the software product to be used for the selected purpose.

Cost or Benefit: Ability of the software product to satisfy its cost or benefit specification.

Security: Ability of the software product to be defended against unauthorized use.

Reliability: Ability of the software product to reproduce its function over a period of time.

Timeliness: Ability of the software product to meet its delivery deadline.

Correctness: Ability of the software product to meet its functional requirements.

The application of the ADEQUATE model includes the following steps [93, 124]: (1) identification of the key personnel, (2) identification of the essential views and quality factors to mark, (3) resolving the conflicts between the quality targets of the essential views, also with regard to the relationships among the criteria, (4) establishing the expected values for the quality targets, (5) marking the achieved quality targets on a scale from one to five, and (6) analysing the results.

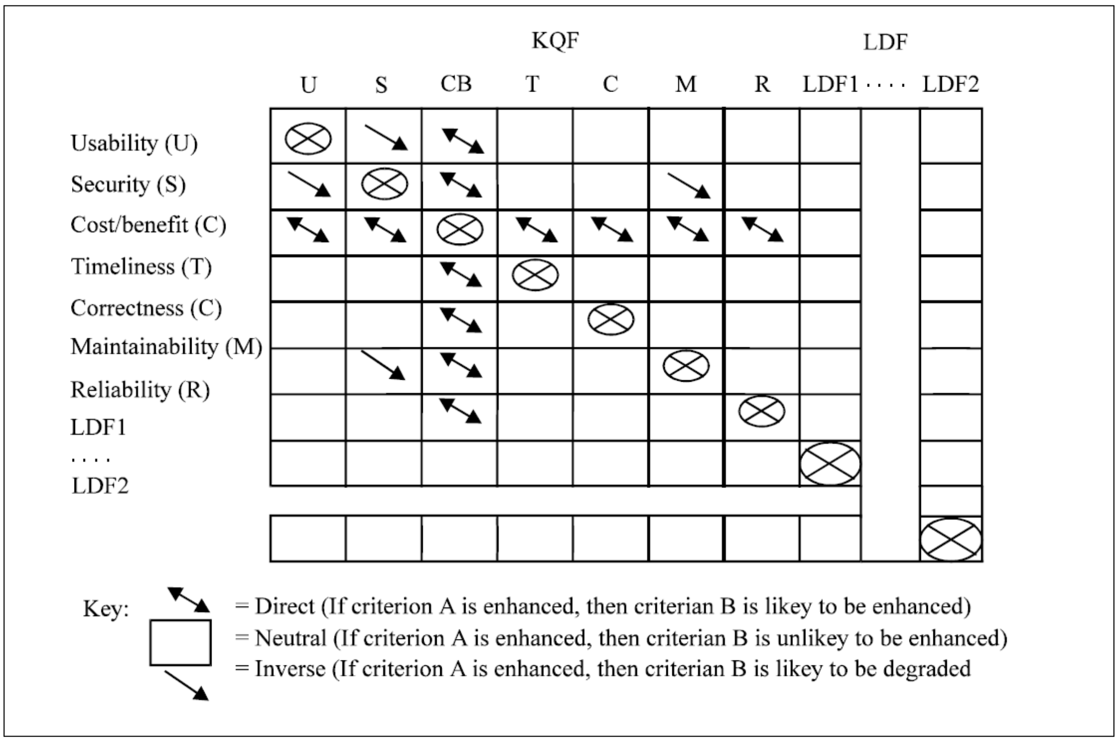
KQF
LDF
U
S
CB
T
C
M
R
LDF1 · · · ·
LDF2
Usability (U)
Security (S)
Cost/benefit (C)
Timeliness (T)
Correctness (C)
Maintainability (M)
Reliability (R)
LDF1
· · · ·
LDF2
Key:
= Direct (If criterion A is enhanced, then criterian B is likey to be enhanced)
= Neutral (If criterion A is enhanced, then criterian B is unlikey to be enhanced)
= Inverse (If criterion A is enhanced, then criterian B is likey to be degraded

Figure 2.10: ADEQUATE Key Quality Factors Relationships, Source: [124]

Figure 2.10 depicts the relationships among the different KQFs with their positive and negative correlations. KQFs and LDFs can be summed up to compute the overall quality factor score. These overall quality score is computed for the achieved and also for the defined quality targets. The ratio of the two score values shows to what extent the quality requirements are met. If a factor is over-engineered, i.e. has a better value than required, then only the required value is taken for the computation so that the high value would not compensate any low value in the measurement.

The use of the model is illustrated on a real-life project [93]. The ADEQUATE quality model is programming language independent and independent of programming paradigms.

COQUALMO Quality Model

Publications identified: [26, 169]

Score value: 15

Time range of the publications identified: [2008; 2008]

Terminology
COQUALMO applies two sub-models to describe defect removal and defect introduction into the software product. The sub-models are named (1) defect removal sub-model and (2) defect introduction sub-model.

Concepts
COQUALMO, which stands for Constructive Quality Model, is a cost model extended with the capability to

deal with software product quality properties [26]. The view of the model is formed on the fact that software product quality is difficult to improve without having an impact on the cost or/and the schedule. Thus, the model also considers process quality related properties. Furthermore, the model is an extension of the COCOMO II cost model, which was meant to estimate cost, schedule and effort for software projects, developed by Bhoem [25]. COQUALMO sets the target to predict the number of residual defects per thousands of lines of source code or function points [91].

COQUALMO describes how the defects are introduced in software products and removed from them [26]. Consequently, it encompasses two sub-models: (1) defect removal sub-model and (2) defect introduction sub-model. The model considers the sources of defect introduction: (1) defects from requirements, (2) defects from design, (3) defects from the source code. The defects are eliminated in each phase of the software product lifecycle; the defects remaining in the software product are named residual software defects [26].

The defects are grouped into the following categories [169]:

1. Requirements: (1) correctness, (2) completeness, (3) consistency, and (4) ambiguity/testability.
2. Design/Code: (1) interface, (2) timing, (3) class-object-function, (4) method-logic-algorithm, (5) data values, and (6) checking.

Both defect removal and defect introduction sub-models can be integrated with the COCOMO II model [26]. The model needs calibration, which is based on the opinions of experts.

FURPS+ Quality Model

Publications identified: [49, 79, 80]

Score value: 10

Time range of the publications identified: [2005; 2005]

The FURPS quality model, originally developed by Grady and Caswell in 1987, was designed to host functional and non-functional requirements [80]. FURPS was extended by Grady in 1992 to consider design and implementation requirements [79]. The latter variant is named FURPS+.

Terminology
Instead of the term quality property, FURPS and FURPS+ quality models use the term requirement, which refers to the view that quality is interpreted as the degree of meeting the specified requirements.

Concepts
FURPS classifies the requirements into the categories; the acronym comes from the first letter of the category names [49, 80]:

Functionality: Requirements that encompass the features of the product.

Usability: Requirements that specify aesthetics and consistency on the user interface.

Reliability: Requirements with regard to reliability including availability, accuracy of system calculations, and the ability of failure recovery.

Performance: Requirements with regard to performance including throughput, response time, recovery time, start-up time, and shutdown time.

Supportability: Requirements with regard to the ability of the software to be supported including testability, adaptability, maintainability, compatibility, configurability, installability, scalability, and localizability.

FURPS+ extended the previous definition with the following entities [49, 79]:

Design requirement or design constraint: Requirements that specify the software design including database design.

Implementation requirement: Requirements with regard to the construction of the software, implementation languages, and resource constraints.

Interface requirement: Requirements with regard to interactions with external systems including formats for data exchange.

Physical requirement: Requirements with regard to the hardware used to host the software.

As mentioned above, the FURPS+ model assist with collecting the architectural requirements. Eeles points out in [49] that gathering architectural requirements is a complex activity for three reasons: (1) use cases of the software product are more visible for end users than the architectural requirements; (2) stakeholders are less familiar with the architectural requirements than with the problem-domain-specific requirements, and (3) systematic approaches to collect architectural requirements are less widespread than those for collecting problem-domain-specific requirements.

Consequently, the ability to ask the right questions from the stakeholders to elicit the requirements and the ability to understand the relationships between the requirements both help to understand the system the stakeholders really wish [49]. Eeles proposes concrete activities to collect architectural requirements [50]; furthermore, he also created a questionnaire to assist with these activities [47]. The FURPS+ quality model is programming language independent and independent of programming paradigms.

SQAE and ISO9126 Model Combination

Publications identified: [38]

Score value: 10

Time range of the publications identified: [2004; 2004]

Côté et al. introduce a software product quality model that merges the SQAE [173] model for software risk assessment and the software product quality model of the ISO/IEC 9126 standard [38, 106]. Both models are introduced in this section. The main motivation is formed by the endeavour to move the concepts of SQAE in the context of an internationally recognised standard while keeping the ability to assess and manage risks [38].

Terminology

The SQAE-ISO/IEC9126 model uses the terminology of both SQAE [173] and ISO/IEC9126 [106] models.

Concepts

SQAE [173] measures the internal software product quality in contrast to ISO/IEC 9126 [106], which measures internal, external, and quality in use manifestations of the quality. Côté et al. define an abstraction layer between the two different models to perform the translation from the one domain into the other; moreover, the authors also extend the SQAE model definition to support external quality [38]. The translation between the two models are performed on the level of the quality factors of the SQAE model and the quality sub-characteristics of the ISO/IEC 9126 [106] model, as shown in table 2.8.

Table 2.8: SQAE Factors - ISO/IEC9126 sub-characteristics Mapping, Source: [38], H: Strong Correlation, L: Weak Correlation

		Consistency	Independence	Modularity	Documentation	Self-descriptiveness	Anomaly Control	Design Simplicity
Functionality	Suitability	L						
	Accuracy	H						
	Interoperability		H					
	Security							L
	Functional Compliance	L						
Reliability	Maturity						H	
	Fault Tolerance						H	
	Recoverability						H	
	Reliability Compliance	L					H	
Usability	Understandability	L				L		
	Learnability				H			
	Operability					H		
	Attractiveness							
	Usability Compliance	L						
Efficiency	Time behaviour							
	Resource utilization							
	Efficiency Compliance	L						
Maintainability	Analysability	H			H	H		H
	Changeability		H	H				H
	Stability						H	L
	Testability			H				H
	Maintainability Compliance	L						
Portability	Adaptability		H					
	Installability		L					
	Coexistence				L			
	Replaceability				L			
	Portability Compliance	L						

The extension of SQAE encompasses the following new SQAE quality area and quality factor definitions [38]:

New quality area, efficiency: This quality area covers the measurements with regard to the efficiency of the software.

New quality factor, run-time efficiency: This quality factor contains the measurements with regard to the run-time efficiency of the software to be evaluated.

New quality factor, interface efficiency: This quality factor contains the measurements with regard to the efficiency of the interfaces of the software to be evaluated.

Côté et al. delineate the possibility of extension to include the quality in use aspect in the SQAE model by introducing a new quality area: quality in use, which would be composed of four quality factors equivalent to the quality sub-characteristics hosted by the quality in use model in the ISO/IEC 9126 standard [38, 106]. The SQAE-ISO9126 quality model integration approach is programming language independent and independent of programming paradigms.

Ulan et al.'s Software Product Quality Model

Publications identified: [235]

Score value: 10

Time range of the publications identified: [2018; 2018]

Starting from the ISO/IEC 25010 [112] and from McCall et al.'s quality framework [179], Ulan et al. lay down the foundation for a new software product quality model [235]. The new model assumes that the same quality metrics can contribute to different subcharaceristics, which in reality results in a directed graph rather than in a tree-like structure. Moreover, Ulan et al. apply probability distributions to aggregate the data into a final quality score value. The technique presented can be utilised for any factor-criteria-metric quality model in case the quality scores need to be aggregated into one indicator value. The authors offer different visualisations to highlight the relationships among the data. The model presented shows one high-level quality property: quality, which possesses eight subordinate quality properties i.e. criteria: (1) cloning issues, (2) anti-patterns, (3) file complexity, (4) hierarchy complexity, (5) language issues, (6) referential complexity, (7) text complexity, and (8) validity issues, to which 31 quality metrics are assigned. The publication delineates the mathematical background, the computations and the visualisation possibilities but the definition of the quality model, including the definition of criteria and metrics, is deficient.

Terminology

Ulan et al. do not apply a specific terminology regarding their quality model.

Concepts

The novelty of the model is depicted by the complex mathematical computations to aggregate the results of quality metrics to higher-level quality properties, and in the visualisation techniques applied. Ulan et al.'s quality model aggregates the subordinate quality properties into one final quality indicator. Other quality models targeted at the complete set of software quality properties endeavour to avoid this ambition as high-level quality properties can have conflicting goals for the software product.

Kim and Lee's Software Product Quality Model

Publications identified: [128]

Score value: 9

Time range of the publications identified: [2009; 2009]

Kim and Lee derived a model from the software product quality model of the ISO/IEC 9126 standard [128]. The authors tailored ISO/IEC 9126 to the particular needs of consumer electronic products with the focus on automatic measurement of internal quality. However, the conceptual changes applied exceeds the extent of tailoring. The hierarchic nature of ISO/IEC 9126 was changed, through which Kim and Lee created a new software product quality model and introduced a simple mechanism to automatically assess internal software product quality.

Terminology
Kim and Lee's software product quality model follows the terminology of the ISO/IEC 9126 standard [106, 128].

Concepts
Kim and Lee determined the relative importance of the high-level quality characteristics of the ISO/IEC 9126 standard with regard to their project and selected the relevant quality characteristics in the specific domain: (1) reliability, (2) maintainability, and (3) portability. As quality characteristics and their sub-characteristics cannot be measured directly, Kim and Lee identified metrics that express the quality of the three selected quality characteristics and that are possible to measure automatically by static code analysis. The selected metrics were assigned to the quality characteristics. Thus, the quality characteristics form categories in this model, not hierarchies. In addition, Kim and Lee defined actions for the issues identified by the static code analysis. After carrying out the defined actions and repeating the evaluation, it could be verified to what extent the changes performed in the source code improved quality.

Kim and Lee's approach can be summarised by the following steps to be used for a general software project: (1) determining the relative importance of the ISO/IEC 9126 high-level quality characteristics for the project, (2) selecting the relevant quality characteristics for measurement, (3) collecting metrics that are possible to measure automatically by static code analysis and that have an impact on the selected quality characteristics, (4) performing the automatic measurement, (5) evaluating the measurement results for the selected quality characteristics, (7) defining actions to improve the deficiencies, (8) carrying out the actions defined, (9) repeating the measurement and evaluation, and (10) determining the quality improvement. The steps from 7 to 10 can be repeated until the desired level of quality is reached. Kim and Lee's quality model is not programming language independent due to the metrics used but the concepts of the model are independent of programming paradigms.

GEQUAMO Quality Model

Publications identified: [71]

Score value: 5

Time range of the publications identified: [2003; 2003]

Georgiadou proposed a new generic quality model, GEQUAMO, to express the views on quality of different stakeholders involved in the software production [71]. Each view can be described with a different model based on the same principles. GEQUAMO encompasses also process specific quality properties.

Terminology
Georgiadou uses the term quality attribute for the quality properties. Quality attributes can be decomposed further in a hierarchic manner with an arbitrary depth. In such cases the term quality sub-attribute is applied to designate a subordinate quality property.

Concepts
Georgiadou defines three views for the stakeholders involved in software projects based on the ISO/IEC 12207 standard [71, 110]: (1) users, (2) sponsors and (3) developers. Furthermore, she creates a quality model for each view. The models constructed possess four of the six high-level quality characteristics of the ISO/IEC 9126 standard [106] as quality attributes: (1) functionality, (2) reliability, (3) usability, and (4) maintainability; moreover, productivity is included to express process related subattributes as costs and delivery on schedule in the sponsor's view [71]. The model definition states that the management should reconcile the conflicts between the quality models stemming from different views of the stakeholders to the benefit of the company [71]. Georgiadou developed a specific diagram technique to depict the expected and the measured quality to efficiently highlight the deviations [71]. The GEQUAMO quality model is programming language independent and independent of programming paradigms.

McCall et al.'s Software Product Quality Model

Publications identified: [20, 179]

Score value: 1

Time range of the publications identified: [2002; 2002]

McCall et al. define software product quality as a combination of three distinct activities: (1) product operation, (2) product revision, and (3) product transition [179]. The main objective of the model is to describe quality in a quantitative manner to assist acquisition managers with software products related to air force applications [179].

Terminology

Quality Factor: Quality factors are high-level quality properties of the software product.

Quality Criteria: Criteria are properties of the software product or software development process by which the factors can be assessed and defined.

Quality Metric: A quality metric is a measurable property of the software product.

Concepts

Quality factors describe the overall set of quality with regard to a software product. The quality factors are linked to the quality criteria. A quality criterion may have sub-criteria in a hierarchical manner. In addition, one criterion may be associated with more quality factors. The criteria or sub-criteria are coupled with metrics to make the measurement possible.

McCall et al. define the following factors [179]:

Correctness: The degree to which a software product satisfies its specifications.

Reliability: The degree to which a software product can be expected to perform its intended function with required precision.

Efficiency: The amount of computing resources and code required by a software product to perform an operation.

Integrity: The degree to which access to software product or its data by unauthorized persons can be controlled.

Usability: Effort required to learn, operate, prepare the input, and interpret the output of a software product.

Maintainability: Effort required to locate and correct an error in an operational software product.

Testability: Effort required to test a software product to insure it performs its intended operation.

Flexibility: Effort required to modify an operational software product.

Portability: Effort required to transfer a software product from one hardware configuration or software system environment to another.

Reusability: The degree to which a software product or its components can be used in other applications.

Interoperability: The degree required to couple one system with another.

The defined factors are coupled with the life-cycle phases in the following manner [179]: (1) product operation: (a) correctness, (b) reliability, (c) efficiency, (d) integrity, (e) usability; (2) product revision: (a) maintainability, (b) testability, (c) flexibility; (3) transition: (a) portability, (b) reusability, (c) interoperability.

Some quality factors show correlations with each other. Beside the positive correlations, negative ones also exist. In such cases compromise needs to be found to resolve the conflict in the given context [179]. Galli et al. gave a more detailed description in [62, 64] where they also put McCall et al.'s quality model in the context of other early software product quality frameworks.

The concepts of McCall et al.'s quality model is programming language independent and independent of programming paradigms. Nevertheless, it uses a metric (simplicity metric SI.2 Use of Structured language) that shows connotations with the structured programming paradigm. However, the definition of the metric leaves open the possibility to interpret it as the existence of programming language statements as: if-then-else statements for conditional branching, and loop construction statements [179].

Software Product Quality Models with Zero Relevance Score

The models introduced in this section received zero relevance score value during the evaluation process either due to their publication date or due to the incomplete quality model definition they provide. Nevertheless, these frameworks articulate important concepts, through which they significantly contribute to software product quality modelling. The depth of introduction of the individual software product quality models in the referenced publications differ significantly.

Boehm, Brown, and Lipow's Model. The first complete software product quality model aiming to deal with all aspects of software quality was developed by Boehm, Brown, and Lipow in 1976 [27]. Boehm et al. defined a set of quality properties, called quality characteristics, which can be decomposed further in quality sub-characteristics [27]. In their investigation, they came to the conclusion that a quality model should be developed with more high-level quality characteristics because establishing a single overall characteristic for software product quality as a whole would implicate more difficulties than benefits as many of the major individual quality characteristics are conflicting [27]. They defined eleven high-level characteristics [27]: (1) understandability, (2) completeness, (3) conciseness, (4) portability, (5) consistency, (6) maintainability, (7) testability, (8) usability, (9) reliability, (10) structuredness, and (11) efficiency. Boehm et al. determined the development of later software product quality models with their work. Galli et al. gave a more detailed analysis in [62, 64] where the model is also put in context of other early software product quality frameworks.

Dromey's Model. Dromey states that a software product does not directly exhibit quality properties but product properties which contribute to the quality in a positive or negative manner [45]. He defines a quality model of non-hierarchic nature with the focus on (1) the product properties, called quality-carrying properties, and on (2) the relationships between product and quality properties [45]. If all quality-carrying properties, associated with a programming language construct or with any other entity related to the software, are satisfied, then that programming language construct or entity will cause no quality defect in the software. In addition, if any of the quality-carrying properties of a programming language construct or of any other entity related to the software are violated, then each violation will cause a quality defect in the software [45]. The latter concept is also reflected in the SQALE quality model [152]. Galli et al. gave a more detailed description in [62, 64] where they also investigated the extension possibilities of Dromey's model.

GQM Approach. Goal-question-metric approach to manage and assess quality was defined by Basili and Weiss, extended by Rombach, and developed further by Solingen and Berghout [239]. The method is flexible and able to capture both product and process related quality. It possesses two basic principles: (1) measurement is not based on a predefined set of metrics but goals, (2) goals and metrics need to be defined or selected for an organisation or project [239]. The approach influenced the definition of recent quality models and standards [99].

IEEE Metrics Framework. The IEEE Standard for Software Quality Metrics Methodology, designated also as IEEE Metrics Framework, was first published in 1998 but it was reaffirmed in 2009 [99]. Nevertheless, IEEE Metrics Framework does not present a concrete software product quality model but explains the principles on which a hierarchic software product quality model with two levels in the hierarchy can be constructed for any project in the context of the specific application. The framework names high level quality properties "factors" and quality sub-properties "subfactors". The metrics are assigned to the subfactors. No predefined metric definition is available in the current version of the standard.

Mobile Application Framework. Franke and Weise express that creating a software product quality framework for developing mobile applications is important as there are significant differences in comparison to the quality assessment of desktop application [59]. They lay out some concepts and practices for such a framework in [59] but no software product quality definition is included.

SQAE Quality Model. The SQAE quality model [173] scored zero due to its date of publication. Nevertheless, its concepts were transferred to the context of the ISO/IEC 9126 quality model [106] in 2004 [38], which resulted in a new quality framework as introduced in section 2.3.3. SQAE, acronym of Software Quality Assessment Exercise, defines a hierarchic product quality model to measure and assess the risks associated with software projects. The software product quality model possesses three levels: (1) quality areas, (2) quality factors, and (3) quality attributes. [173]. SQAE is based on the concepts of the quality model of Boehm et al. [27], McCall et al. [179], and Dromey [45, 173].

SQAE implements many-to-many relationships between the quality areas and the quality factors, but one-to-many relationships between the quality factors and the quality attributes. A quality attribute describes a measurable representation of quality, with which quality metrics or rigid scoring criteria for manual evaluation are associated [38]. Consequently, attributes can be computed in part automatically, in part semi-automatically and in part manually.

SQAE possesses four quality areas at the top of the hierarchy [173]: (1) maintainability, (2) evolvability, (3) portability, and (4) descriptiveness. In addition, it hosts seven core quality factors on the second level of the hierarchy [173]: (1) consistency, (2) independence, (3) modularity, (4) documentation, (5) self-descriptiveness, (6) anomaly control, (7) design simplicity. The values of the quality factors contribute to the quality areas to a defined extent as shown in figure 2.11. Quality attributes are linked to the quality factors. The method is independent of programming languages and programming paradigms.

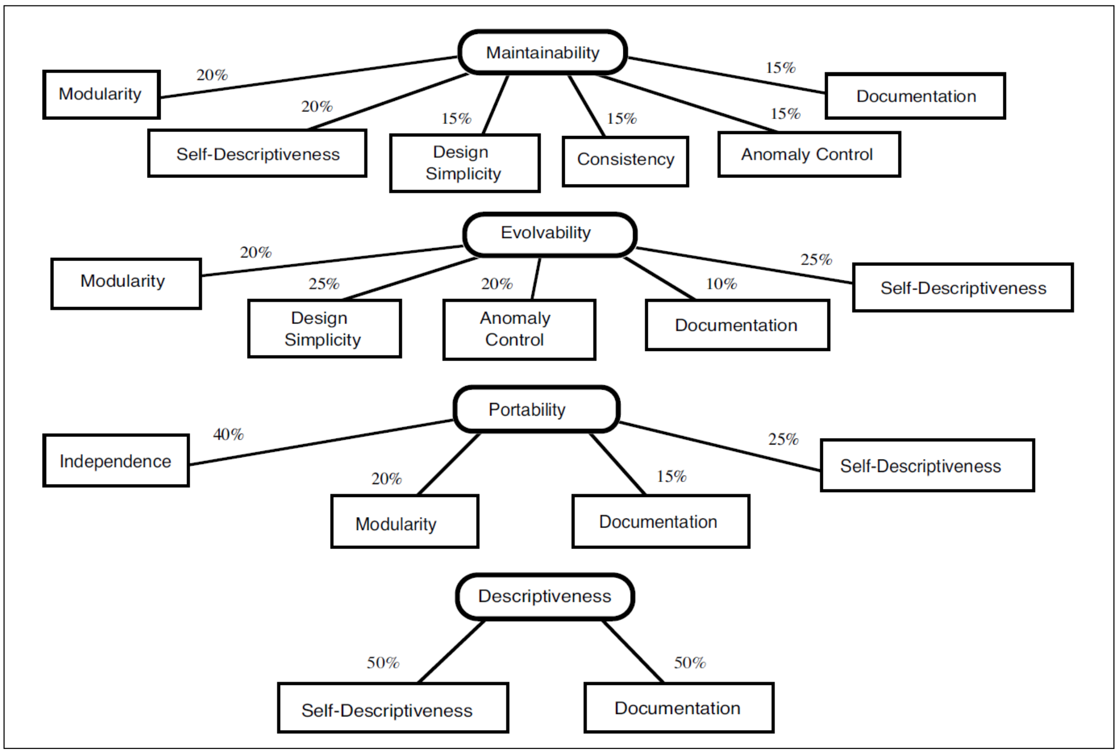

Figure 2.11: Software Quality Assessment Exercise (SQAE), Source: [173]

STAC Model. Hyatt and Rosenberg define the STAC quality model, which can deal with product and process specific quality properties [96]. The model has four high-level quality properties called goals: (1) requirements quality, (2) product quality, (3) implementation effectivity, and (4) testing effcetivity [96]. A set of attributes, which can be measured by defined metrics, are linked to the high-level quality properties [96]. The model also influenced the development of the GEQUAMO quality model [71].

SQUID Model. The SQUID (Software QUality In Development) quality model [132], developed by Kitchenham et al., possesses two types of elements: (1) a structure model that defines the model's elements and their interactions, and (2) a content model that describes the entities used. SQUID is a composite software product quality model in the sense that it includes concepts from McCall et al.'s quality model [179] and the first version of ISO/IEC 9126 quality model [104] issued in 1991 and adapts them [132]. The base principles of SQUID can be expressed as follows [132]: (1) software quality requirements cannot be defined independently of a concrete software product since quality requirements arise from the operational properties expected, (2) it is possible to define a new or use an existing quality model, (3) the target quality requirement values have to be quantitative and identified by measures, and (4) the abstract quality model can be regarded as a check-list for assessing the complete set of quality requirements. The approach for using a quality meta-model to introduce the elements of the model and a separate quality model for assessing the software product quality returns also in the Quamoco quality model [243].

Zhang et al.'s 2D Model. Zhang et al. express the assumption that quality properties in software product quality models constructed by the factor-criteria-metric decomposition approach show cause-effect relationships on the criteria and factor level [262]. Thus, criteria that belong to different factors may have impact on each other; moreover, the factors with a decomposition hierarchy on their own may also have impact on each other

[262]. Therefore, they introduce a second dimension to describe those cause-effect relationships and lay down the concepts in [262].

2.3.4 Software Product Quality Models Beyond Software Quality

The hierarchic decomposition process of software product quality applied in the scope of the ISO/IEC 9126 [106] and its successor ISO/IEC 25010 standards [112] to handle the abstract notion of quality exceeded the boundaries of software product quality modelling and software technology. The quality frameworks [18, 82, 84, 148, 218, 223, 224] which transferred the principles of the software product quality models to a domain different form software technology are presented in this section.

Bansiya and Davis publish the QMOOD model, which is software technology related even if it is not a software product quality model [18]. The main motivation was to create a quality model for assessing the outcomes of the early phases in the development lifecycle such as analysis and design [18]. The model adapts the principles of the ISO/IEC 9126 standard family [106] and Dromey's model [45]; moreover, it establishes own metrics to measure the object-oriented analysis and design quality [18]. Furthermore, QMOOD offers tool support for automatic evaluation for static analysis of the source code [18].

Leveraging the guidelines and structures laid down by the ISO/IEC standards, Sproge and Cevere delineate how the quality of study programmes of a Higher Educational Institution can be dealt with by means of the ISO/IEC 9126 model [223, 224]. Study programmes and software as products share several similar properties from the point of view of quality since both are abstract products [224]. The model they defined possesses five high-level quality characteristics and twenty-one sub-characteristics. The high-level quality characteristics: (1) functionality, (2) usability, (3) efficiency, (4) maintainability, and (5) portability. The Faculty of Information Technology at the Latvia University of Agriculture started to use the model from the academic year 2009/2010. The use of the model was extended to all study programmes at the university from the academic year 2011/2012. The evaluation covered 485 courses of 55 study programmes by 2012.

Guceglioglu and Demirors implement the quality model of the ISO/IEC 9126 standard [106] in the context of business processes [82]. They found analogy between software products and business processes because both have inputs, outputs, logical structure and operations. Starting form this analogy, they defined four high-level quality characteristics: (1) functionality, (2) reliability, (3) usability, and (4) maintainability. In addition, the model contains a level of sub-characteristics that are linked to the quality metrics of the business processes. By means of this new approach, it became feasible to assess business processes of an organisation and to receive early feedback about them.

Gurbuz, Guceglioglu and Demirors define a quality framework, based on the quality model of the ISO/IEC 9126 standard [106], to assess the processes of human resource management. Furthermore, Lepmets, Ras and Renault create a quality model, derived from the ISO/IEC 25010 standard [112], to assess service quality [148].

The above quality models stem from the ISO/IEC standard families [106, 112] even if they do not address software product quality at all.

2.4 Recent Trends

The outcomes of the systematic literature review were published in [65]. The review, including the data evaluation and data processing, took nearly two years. The automatic searches in appendix A.1 were repeated in the two largest computer science archives: (1) IEEE and (2) ACM for this period, not to miss any software product quality model that appeared while conducting the review. All of the returned publications were incorporated in the research report published in [65].

As [65] appeared in 2020, the past two years have also been investigated while closing the PhD manuscript, and the RQ1a and RQ1b queries in appendix A.1 were carried out in the (1) IEEE and (2) ACM archives. The returned documents are listed in table 2.9 and table 2.10.

Table 2.9: Repeated Searches Results

	IEEE	ACM	Date
RQ1a	1	1	01.01.2020-31.12.2022
RQ1b	2	7	01.01.2020-31.12.2022

The abstracts of the publications identified, in the scope of the repeated search, underwent analysis to decide which software product quality model is affected. The results are depicted in table 2.10.

Table 2.10: Affected Software Product Quality Models

Document	Software Product Quality Model Affected
[86]	ISO25010
[254]	QMOOD, not a software product quality model. See 2.3.4 for details.
[19]	Model driven engineering, model transformations, not a software product quality model.
[21]	ISO25010
[163]	Location of log statements in the source code, not a software product quality model.
[207]	Log parsing, not a software product quality model.
[161]	Location of log statements in the source code, not a software product quality model.
[34]	Security related, not a software product quality model.
[140]	Maintainability related, not a software product quality model.
[149]	Performance related, not a software product quality model.
[165]	Root cause analysis, not a software product quality model.

The identified publications by the repeated RQ1a and RQ1b queries show that the rank of relevance of the software product quality models in table 2.2 did not change. In fact, the recent publications increase the relevance of the ISO/IEC 25010 quality model [112], which stands at the first position in the ranking.

2.5 Validity

This sections reports the identified threats to the validity of the literature review and the efforts made for their mitigation.

The search to identify the sources was performed with two approaches as proposed by Kitchenham et al. in [129]: (1) automatic search in the scientific document databases with defined and recorded search strings, and (2) manual search to complement the automatic search. All the identified, relevant documents went through a defined scoring process. The documents unrelated to software product quality models were excluded with providing a reason in each case of exclusion.

Validity can be questioned if publications with the definition, application, tailoring or research of software product quality models are missed during the search. The author endeavoured to minimise this risk by carrying out the automatic searches in six computer science relevant scientific document databases including IEEE and ACM; moreover, the references to software product quality models cited in the identified publications of the automatic search were followed manually and examined.

In addition, the data extraction, the scoring of the publications, and the assignment to the quality model families also pose a threat to validity. This threat was minimised by an internal review and by independent peers who reviewed the research report published in [65].

2.6 Summary

The main goal of this chapter was to identify the existing software product quality frameworks to answer RQ1, and to investigate whether these frameworks appropriately handle execution tracing quality in order to answer RQ2.

The existing models are not of equal relevance however sophisticated the quality model definition is. All of this made necessary to conduct a systematic literature review (1) to discover which quality models exist that aim to deal with the whole set of the quality properties of software products, (2) to define and implement a mechanism to measure the relevance of the published software product quality models, and (3) to examine the role of execution tracing quality in the identified software product quality frameworks.

A novel approach was implemented, which can be applied in systematic literature review processes [130] (1) to score each included document in the review, and (2) to compute scores for the cluster of related documents to construct an indicator of relevance for the given research subjects. The designed scoring standard was consistently applied in the course of the research; moreover, individual quality scores, average of the individual quality scores, relevance scores, publication ranges and the 12-month average of the Google Relative Search Index [76] were also considered, which highlight relevance of the identified software product quality model families from different angles. The individual quality score considers the clarity and the actuality of each identified publication; moreover, it also considers that quality models can be published as fully defined models in one step and also in smaller increments.

The quality models of the ISO/IEC frameworks [106, 112] and the SQALE model [152] stand out as the ones with the most vivid research interest. The documents published while this study was being conducted also corroborated this trend. The popularity of the ISO/IEC models [106, 112] resides in their simplicity, in their ability of hierarchic decomposition to cope with abstraction, and in their potential offered for adaptation to specific project needs. The popularity of the SQALE model [152] is warranted by its quick and simple integration in the software development pipeline and by the automation potential it provides; however, it only considers the internal view of quality, which is a significant restriction as explained in chapter 3. The SQALE model [152] implementations, including SonarQube, are widespread [180, 214, 222, 226, 260]. As Kim and Lee [128] introduced, automating the derivatives of the ISO/IEC models [106, 112] is also possible. However, only the internal quality view of the framework can be assessed by static code analysis in each case.

SQALE [152] is the only model analysed in the scope of the systematic literature review which explicitly endeavours to satisfy the representation condition of measurement theory [155]. The representation condition asserts that properties of real world entities measured are mapped to numeric representations in such a way that the numeric representations are equivalent to the reality [152]. Most quality models leverage weighted averages to aggregate quality property values or metrics. Letouzey and Coq point out in [155] that using weighted averages in hierarchies to aggregate the metrics violates the representation condition; moreover, they illustrate the problem by means of use cases with (1) masking effects, (2) compensation effects, and (3) threshold effects for different aggregation operations and show how these effects hide the underlying values in the hierarchy [155]. Consequently, SQALE [152] applies a common scale for measuring each quality property and only the sum operation is used to avoid the mentioned anomalies.

The simplicity of the ISO/IEC quality models [106, 112] and its ability to tackle abstraction provided a starting point for researchers who transferred the same concepts to different application domains, including quality modelling in Higher Education teaching and business processes [82, 84, 148, 223, 224].

Wagner et al. express the following requirements for a quality model to be able to assess quality in an appropriate manner [242]: (1) it must be interpretable for decision makers, (2) it must be able to handle incomplete information, and (3) it must allow for contradictory quality aspects. These requirements set out the way for the application of fuzzy logic in this domain. Fuzzy logic makes possible to consider contradictory quality targets, incomplete information; and it makes possible to describe the quality model with linguistic rules, which eases interpretation [255, 257, 258, 259].

Finally, the literature review verified that existing software product quality models do not adequately assess the quality of execution tracing, however, it would be required [63, 64, 65, 69].

Chapter 3

Quality Model Taxonomy and Its Application

A significant part of the content of this chapter appeared in the author's article published in Applied System Innovation (MDPI – 2021, reference [70]).

The systematic literature review introduced in chapter 2 provided deep insights into the identified software product quality model families, in the terminology they use and in the concepts they implement. The literature review discovered that the identified, distinct software product quality model families are able to address the manifestations of quality differently; therefore, it is necessary to analyse and to consider this ability to avoid misleading statements and misunderstandings while explaining or comparing the results of software product quality measurements and assessments. The quality manifestations used for the quality measurement always need to be presented, otherwise the quality assessment results cannot be interpreted in an appropriate manner.

This chapter introduces the study of the quality manifestations the software product quality models are able to address; moreover, each identified software product quality model family is classified according to the quality manifestations they can capture (1) to assist with the interpretation of the quality measurement and assessment results, and (2) to provide guidelines while selecting a software product quality model for a specific project. Thus, the reported results in the chapter answer RQ3 in section 1.3.

3.1 Quality Manifestations as Quality Views

Following the terminology of the ISO/IEC 9126 standard and its successor ISO/IEC 25010 [106, 112], the quality manifestations the software product quality models are able to capture are called quality views throughout the thesis. Developing a unified terminology is necessary since some software product quality frameworks, including ADEQUATE [93, 124], FURPS+ [49, 79, 80], and GEQUAMO [71], reflect the opinions of different stakeholders, directly or indirectly, as quality views in the model definition documents.

The ISO/IEC 9126 and ISO/IEC 25010 standards distinguish three distinct quality views [106, 112, 132]:

Internal quality view: The set of quality properties of the source code and documentation that are deemed to influence the behaviour and use of the software.

External quality view: The set of quality properties that determine how the software product behaves while it operates. These properties are usually measured when the software product is operational and is being tested.

Quality-in-use view: The set of quality properties that determine how the user perceives software product quality, i.e. to what extent the objectives the software is used for can be achieved.

The software product quality assessed by the three different views needs to have predictive power towards each other, called predictive validity [106, 112]. Thus, the measured internal quality should predict the external quality, and the measured external quality should predict the quality-in-use [106, 112]. Nevertheless, this validity does not necessarily hold since quality metrics and measures associated with the quality properties do not always have such a predictive power [132]. Software product quality model families explicitly or implicitly define which manifestations of quality they are able to address.

3.2 Intrinsic Presence of Quality Views

Software product quality models measure quality properties of the software products, which belong to three basic categories even if the model definition documents do not mention quality views or quality manifestations explicitly: source code and documentation related properties; dynamic properties, which can be measured or observed while the software operates; and properties based on the feedback of the stakeholders while the software is used. These three categories correspond directly to the three quality views defined by the ISO/IEC standards as provided above. The intrinsic presence of the quality views in software product quality modelling motivates the need to classify the quality models based on the quality views they are able to handle.

3.2.1 Methods

In the scope of the taxonomy, qualitative research was applied to analyse the definition documents and the tailored quality models to specific project needs. The publications forming the input of the classification were identified by the systematic literature review presented in chapter 2. As the internal, external quality views and quality-in-use view are not explicitly defined in all the model descriptions, the quality property specifications and model concepts were investigated to obtain an insight as to whether a given quality model is able to address external or quality-in-use manifestations.

Quality models able to provide guidelines to investigate the source code or documentation, without exhibiting any quality property assessing the software's runtime behaviour or the perception of the end user, are designated as quality models with internal quality view only (denoted "I" in Table 3.1). Quality models that contain quality properties to assess the software's runtime behaviour are designated with the external quality view (denoted "E" in Table 3.1). In addition, if a quality model also exhibits quality properties to reflect the end user's assessments in any form, then the quality-in-use view is also assigned (denoted "U" in Table 3.1).

If a given quality model does not provide quality property specifications, but it contains concepts that allow defining or dealing with existing quality property specifications to extend the model, then these concepts were analysed to decide which quality views the quality model is able handle. Consequently, quality models lacking explicit information about the quality views were classified based on the author's interpretation of the published model definitions with regard to concepts, and quality properties. Cases where alternative interpretations may exist are indicated with brackets in Table 3.1 and with additional explanations in section 3.2.2. Quality models that do not provide enough information to perform a reasonable classification are classified “Undefined” in Table 3.1. There are also quality assessment approaches that allow the definition of a new quality model or the use of a different quality model from the predefined one. The quality views applied in such cases depend on the implemented quality models (denoted “D” in Table 3.1). The flow chart of the taxonomy process is depicted in Figure 3.1.

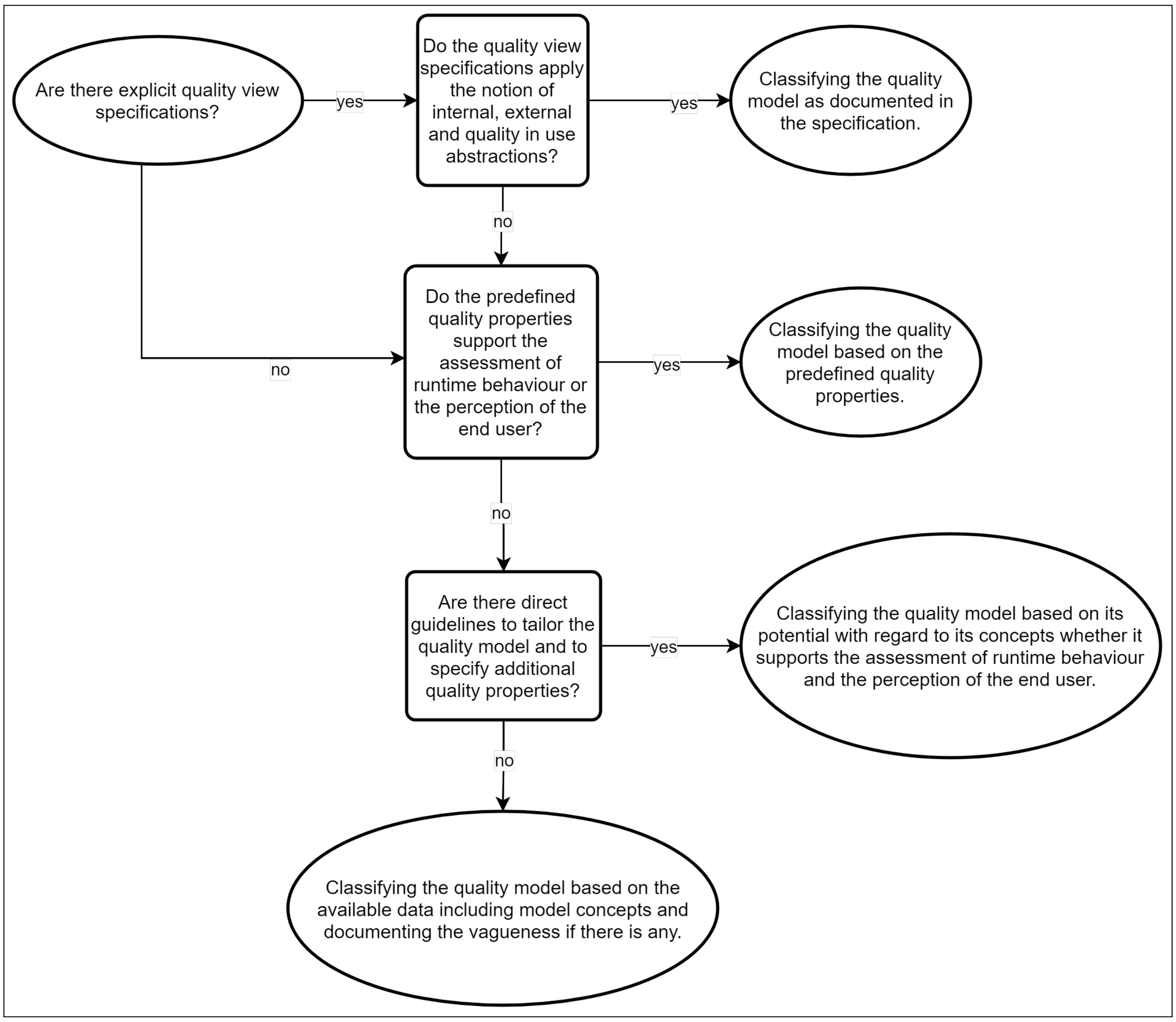
Are there explicit quality view specifications?
yes
Do the quality view specifications apply the notion of internal, external and quality in use abstractions?
yes
Classifying the quality model as documented in the specification.
no
no
Do the predefined quality properties support the assessment of runtime behaviour or the perception of the end user?
yes
Classifying the quality model based on the predefined quality properties.
no
Are there direct guidelines to tailor the quality model and to specify additional quality properties?
yes
Classifying the quality model based on its potential with regard to its concepts whether it supports the assessment of runtime behaviour and the perception of the end user.
no
Classifying the quality model based on the available data including model concepts and documenting the vagueness if there is any.

Figure 3.1: The Flow Chart of the Quality Model Classification, Source: [70]

3.2.2 Taxonomy

This section presents the taxonomy of the software product quality model families from the point of view of quality manifestations they can capture. An important aspect of quality models is their ability to measure the properties of software products; therefore, software product quality models are involved in the study; however, some quality models also exhibit properties to assess the process by which the software products come into existence. Such models are also marked in Table 3.1 to provide support for the selection of a potential quality model.

Quamoco, ADEQUATE, FURPS+, GEQUAMO, McCall et al.'s model, and Boehm et al.'s model are classified as quality models with internal, external and quality-in-use views in Table 3.1: the Quamoco quality model encompasses an entity "utility" that describes the relative satisfaction of stakeholders with regard to a specific "factor" in the model's terminology; ADEQUATE determines the key personnel and their views. FURPS+ includes systematic approaches to collect architectural requirements involving the stakeholders to consider their opinions; GEQUAMO encompasses quality views for users, sponsors and developers; McCall et al.'s model contains usability and testability factors; Boehm et al.'s model reflects quality characteristics for usability and human engineering, which could make it possible to consider the end user's view of quality.

SQUALE, COQUALMO and STAC models are classified as quality models with internal and external quality views in Table 3.1: SQUALE possesses different metrics and practices related to distinct test levels; COQUALMO comprises a defect introduction and removal sub-model, which assumes the software is operational; the STAC model defines metrics related to testing.

Ulan et al.'s model, SQALE, EMISQ, Kim and Lee's model, Dromey's model, and SQAE are classified in Table 3.1 as quality models with internal quality view only: Ulan et al.'s model permits the use of any goal-question-metric approach; its novelty lies in the complex mathematical computations used in a clustering approach, although the model published demonstrated internal quality properties only without direct guidelines for extension; SQALE also contains the testability quality property divided into two further quality properties, Unit Testing Testability and Integration-level Testability, that are based on the results of code analysers.

GQM and SQUID are rather general quality assessment approaches with no predefined quality models. These approaches assume the creation of a software product quality model or the use of an existing one, named content model in SQUID terminology, which is suitable to achieve the goals set in the given context. Consequently, the quality views they consider depend on the defined metrics and goals. In the published case study with the model definition [132], SQUID also encompasses metrics that can only be collected when the software is operational and metrics on usability that allow the expression of end users' quality perception. For these reasons, they are classified in Table 3.1 as quality assessment approaches with quality views that depend on the quality model used.

In addition to the quality views applied, further information is presented in the columns: "Predefined Quality Properties or Metrics Available" to describe whether the specified model defines quality properties with associated computation methods; "Research Interest" to indicate whether there are more publications related to the quality model and one in the last six years; "Widespread Use Cases" to provide the judgement based on the specified model's 12-month average GRSI indicator from Table 2.2; "Also Process Related Properties" to show whether the specified model is a hybrid model with quality properties that describe process quality.

Table 3.1: Taxonomy: Software Product Quality Models and Their Quality Views, Source: [70]

ID	Relevance Rank	Name	Quality Views Considered	Predefined Quality Properties or Metrics Available	Research Interest	Widespread Use Cases	Also Process Related Properties
1	1	ISO25010 [44, 58, 97, 98, 112, 167, 167, 192, 219, 219]	I, E, U	Yes	Yes	Yes	No
2	2	ISO9126 [10, 36, 94, 106, 120, 160, 164, 196, 240]	I, E, U	Yes	Yes	Yes	No
3	3	SQALE [91, 150, 151, 152, 153, 154, 155, 156]	I	Yes	Yes	Yes	No
4	4	Quamoco [73, 242, 243, 244]	I, E, U	Yes	No	No	No
5	5	EMISQ [141, 200, 201]	I	Yes	No	No	No
6	6	SQUALE [16, 100, 146, 183]	I, E	Yes	No	No	Yes
7	7	ADEQUATE [93, 124]	I, E, U	Yes	No	No	Yes
8	8	COQUALMO [26, 169]	I, E	Yes	No	No	Yes
9	9	FURPS [49, 79, 80]	I, E, (U)	Yes	No	Yes	Yes
10	9	SQAE and ISO9126 combination [38]	I, E	Yes	No	No	No
11	9	Ulan et al. [235]	I	Yes	No	No	No
12	10	Kim and Lee [128]	I	Yes	No	No	No
13	11	GEQUAMO [71]	I, E, U	Yes	No	No	Yes
14	12	McCall et al. [20, 179]	I, E, (U)	Yes	No	No	Yes
15	13	2D Model [262]	Undefined	No	No	No	Undefined
16	13	Boehm et al. [27]	I, E, (U)	Yes	No	No	No
17	13	Dromey [45]	I	Yes	No	No	No
18	13	GQM [239]	D	No	No	Yes	D
19	13	IEEE Metrics Framework Reaffirmed in 2009[99]	Undefined	No	No	No	Undefined
20	13	Metrics Framework for Mobile Apps [59]	Undefined	Yes	No	No	Undefined
21	13	SATC [96]	I, E	Yes	No	No	Yes
22	13	SQAE [173]	I	Yes	No	No	No
23	13	SQUID [132]	D	D	No	No	D

From Table 3.1, it is clear that the identified software product quality model families apply different quality views to capture the abstract notion of quality. The capabilities of the listed software product quality models deviate significantly; many of them are unable to assess the external quality view and the quality-in-use view. The provided taxonomy makes it possible to interpret the quality assessment results. Based on quality assessments with internal quality view only, it is not possible to make valid statements regarding software product quality as a whole. This is because such models lack the capability to capture how the software behaves while it is used and how the end user perceives its quality. Quality assessment results always need to be interpreted with the quality model's capabilities, including quality views, which is especially important for project negotiations. In addition, Table 3.1 also contains relevant information to support the selection of a quality model for a specific project. In such cases, not only the model's quality views play an important role but the following criteria can also influence the decision: (1) the amount of associated research studies on the model, i.e., the research interest, (2) the model's public spread, and (3) whether the model has process related quality properties.

3.3 Application of the Established Quality Model Taxonomy

This section presents two different case studies about two distinct, realistic but fictional projects to highlight the application of the defined quality model taxonomy. In addition, tailoring the quality model of the ISO/IEC 25010 standard [112] to specific project needs is also briefly illustrated in a simplified manner. Furthermore, the quality measurement and the assessment process are also demonstrated by means of the tailored quality model.

The first case study shows the process how to classify a given software product quality model. This use case is important for being able to formulate valid statements with regard to the assessment results coming from a given software product quality model.

The second case study demonstrates how to select a software product quality model based on the taxonomy of Table 3.1; and how the selected quality model can be tailored to a specific project, if the requirements are given. This use case is essential if the quality measurement or assessment policy need to be defined or changed.

3.3.1 Case Study 1: Assessing a Given Software Product Quality Model

Background: In the scope of this case study a fictional but realistic project is investigated, which is already set up and running. SonarQube [222] is applied for assessing software product quality, which is based on the SQALE model [152]. The SQALE model [152] has different tool implementations [180, 214, 222, 226, 260]. The model's metrics, called indices and rating in SQALE terminology, provide objective measures to compare the quality of the same project at different points in time or the quality of different projects [151, 152, 153, 155]. The estimation of the so-called technical debts, remediation and non-remediation costs form also important indicators. When a software project is transferred to an external provider for maintenance or for further development such indicators may directly influence the financial negotiations. Therefore, special attention is devoted to the SQALE model, which explains, besides its widespread use, its selection for demonstration. The SQALE-specific terminology and the concepts of the quality model are summarised in section 2.3.3.

Activities: Assuming the project team does not know the relationship between SonarQube [222] and SQALE [152], they start with the documentation of the SonarQube [222] tool used, which reveals that the tool applies the SQALE quality model [152]. The project team needs to figure out the quality views used by this underlying model. The possible quality views of the SQALE model mean the theoretical maximum of scope of quality the SonarQube tool [222] can consider and assess. Thus, after successfully identifying the quality model applied by the tool, the project team locates the quality model in Table 3.1 and learns that it can assess the internal quality view only, which is a fraction of the whole set of software product quality. Consequently, no valid statements can be made about the software product quality as a whole based on SonarQube's [222] results.

Optional Activities: Optionally, to verify the above finding; and to elicit how far the quality assessment results of SonarQube [222] reflect software product quality as a whole, the project team performs a literature review and they identify Hegeman's study in [91]. Hegeman measured an $r = 0.5$ Pearson correlation between the SQALE indices and the perceived quality [91], which yields a 25% coefficient of determination ($r^2 = 0.25$). Thus, according to the measurement, the SQALE results can explain only the 25% of quality perceived by the user. However, Hegeman performed his measurement on a small sample containing 9 software projects, and 11 experts with 22 responses. Nevertheless, Hegeman's findings support the outcome of the classification based on Table 3.1.

Conclusions: The SQALE model and its implementations including SonarQube [222] are unable to handle the external and quality in use views, which means that no valid statements, based on SQALE, can be made regarding the software's direct runtime behaviour and the quality the end user perceives. Consequently, SQALE and its implementations are useful to measure internal quality in an objective manner but statements about software product quality as a whole based solely on SQALE assessments are invalid. Besides the SQALE assessment, the outcomes of the different testing stages need to be considered and the end user's feedback on the software needs to be taken into account before forming any conclusion about the software product quality as a whole. For the above reasons, the statement made in [156], that non-remediation costs estimated by SQALE can be claimed, may not be considered as valid in a general sense but solely for the internal quality view as any other statement based on SQALE indices.

3.3.2 Case Study 2: Selecting a Quality Model when the Requirements are Given

Background: This case study illustrates how a software product quality model can be selected if the requirements of a particular project are given. While collecting the architectural requirements, Eeles's checklist [49] can be of assistance as the customer is usually not accustomed to answer questions necessary to define the architectural needs. The stakeholders formulate the following quality guidelines: (1) source code quality must be assessed, (2) the quality of the operational software must be assessed, and (3) the perception of the end user about the software must also be assessed and reported after each defined milestone in the project's life-cycle.

Activity 1: The project team investigates the quality guidelines formulated by the stakeholders, which require the application of a software product quality model with the three distinct quality views: internal, external and quality in use. The quality guidelines do not explicitly mention process related activities; therefore, the quality model does not need to be a hybrid model to assess the development process. The project team locates the quality models in Table 3.1 and finds that the following models qualify for this criterion: ISO/IEC 25010 [112], ISO/IEC 9126 [106], Quamoco [73, 242, 243, 244], ADEQUATE [93, 124], GEQUAMO [71], GQM [239], SQUID [132].

Activity 2: The project team narrows down the list of eligible quality models. They prefer applying a quality model with constant research interest and widespread use cases; moreover, they do not want to create a new quality model, which rules out GQM [239] and SQUID [132]. Consequently, they could select ISO/IEC 25010 [112] and ISO/IEC 9126 [106] as shown by the columns "Research Interest" and "Widespread Use Cases" in Table 3.1; nevertheless, ISO/IEC 9126 [106] is superseded by its successor standard ISO/IEC 25010 [112]. For these reasons, they choose to apply the ISO/IEC 25010 [112] quality model.

Activity 3: The project team starts to outline how to assess software product quality by means of the selected quality model. Since internal quality means the quality of source code in the given context, static code analysers can be used to automate this assessment. Nevertheless, the project team would need to set up the analysers individually and interpret their results with regard to the selected quality model: ISO/IEC 25010 [112]. Kim and Lee present a similar study in [128] with the predecessor standard ISO/IEC 9126 [106]. At this point, the project team challenges its previous decision about the quality model selection and comes up with the idea to use a different quality model to measure the internal quality, which offers tool support and full automation, while they intend to use ISO/IEC 25010 for measuring and assessing external quality and quality in use. Based on Table 3.1 they choose SQALE, which is widespread and implemented by many tools including SonarQube [222]. In addition, the high-level quality properties of SQALE [152] were derived from the ISO/IEC 25010 [112] and ISO/IEC 9126 [106] standard families as discussed in [152]. The project team easily sets up SonarQube [222] in the development pipeline to measure and assess the internal quality.

Activity 4: The project team confronts the extensive definition of the quality model in the ISO/IEC 25010 [112] standard. They find out, however, that the standard encourages tailoring the quality model to the particular needs of the specific project, which is also a reasonable solution to decrease the costs of the assessment and select the most relevant quality properties to measure. The project team organises a meeting with the stakeholders and they discuss which of the eight high-level quality properties of the ISO/IEC 25010 [112] quality model are the most important ones in the given context. The selection of the most important high-level quality properties can be carried out by constant sum scaling [171], i.e. each stakeholder distributes the same amount of scores for the quality properties according to the importance in her/his opinion, and the properties are selected based on their accumulated score values; alternatively, they can use Analytic Hierarchy Process [209] to produce a ranking. After the meeting, based on the decisions of the stakeholders, the following high-level quality properties[4] are selected: (1) for the external quality view: (a) performance efficiency, and (b) reliability; (2) for quality in use: (a) satisfaction.

Activity 5: Based on the previously selected high-level quality properties, the project team chooses subordinate quality properties[5] to be measured while, using the flexibility offered by the ISO/IEC 25010 standard, they define new quality metrics[6] and associate them with the chosen subordinate quality properties. They define a measurement scale, a measurement method, and the attributes to measure[7] for each metric, called measure in the terminology of ISO/IEC 25010 [112], in Table 3.2. They have the metric definitions approved by the stakeholders. Using the formulated metrics, the quality assessment takes place after each defined milestone in the project's life-cycle. The scale of each attribute, i.e. quality measure element, is ratio scale in statistical sense; consequently, each quality attribute possesses a non-arbitrary zero point which expresses the complete absence of the attribute. In addition, the defined quality measures produce a value in the [0; 1] range, where the value 1 indicates the best score that can be achieved while the value 0 indicates the worst score that can be achieved.

Table 3.2: Case Study 2: Tailoring the ISO/IEC 25010 Quality Model to the Needs of the Specific Project

View[8]	Quality Characteristic	Purpose	Quality Sub-characteristic	Quality Measure	Measurement Method	Computation Formula
E	Performance efficiency	How performantly the software behaves	Time-behaviour	Response time measure	Determine the most important business operations in the system or at external interfaces and measure the response time for them. Compute the measure for each business operation determined as defined by the formula.	$\frac{1}{1+\frac{time_{response}}{time_{maxallowed}}}$
E	Reliability	How reliably the software can operate	Availability	Crash measure [9]	Count the number of crashes and the number of "freezes" (when the software is available but it does not respond) in a given time frame, then compute the measure as defined by the formula.	$\frac{1}{1+count_{crash}+count_{freeze}}$
			Fault tolerance	Manual intervention measure	Count the number of manual interventions, which are necessary to maintain the operational state of the software in a given time frame, then compute the measure as defined by the formula.	$\frac{1}{1+count_{manualintervention}}$
U	Satisfaction	How satisfied the end user is when the software is used	Usefulness	Usefulness goal measure	The end users assess the software how easily they can achieve their goals by the use of the software. The user assessment results in a mark in the range [0; 10]. The higher the value is, the higher the user's satisfaction is. Compute the measure as defined by the formula.	$\frac{\sum_{n=1}^{count_{users}} assessment_n}{10*count_{users}}$

Activity 6: The project team defines the acceptable metric ranges for each metric in Table 3.2 with regard to the context of the project and have them approved by the stakeholders. The following acceptable ranges are defined after the approval: (1) response time measure over 0.55, (2) crash measure over 0.5, (3) manual intervention measure over 0.5, and (4) usefulness goal measure over 0.7.

[4]High-level quality properties are called quality characteristics in the terminology of the ISO/IEC 25010 standard [112].
[5]Subordinate quality properties are called quality sub-characteristics in the terminology of the ISO/IEC 25010 standard [112].
[6]Quality metrics are called quality measures in the terminology of the ISO/IEC 25010 standard [112].
[7]Attributes to measure are called quality measure elements in the terminology of the ISO/IEC 25010 standard [112].
[8]I: Internal quality view, E: External quality view, U: Quality in-use view
[9]If the software starts up quickly and automatically, then the up-time ratio does not appropriately mirror availability.

In the course of the quality measurement the quality measure elements, i.e. the inputs of the quality measures, are determined and the formulas in Table 3.2 are used for computation. The project team measures the following values: (a) the external interface mean response time: 20 msec while its maximally allowed response time amounts 30 msec, (b) 2 crashes in the time frame of the system test, (c) 1 manual intervention during the system test so that the software's operational state could be maintained, and (d) three end users rate the software's usefulness to achieve their work goals with the scores: 6, 9, and 7. These quality measure elements result in the following quality measures: (1) response time measure: 0.6, (2) crash measure: 0.33, (3) manual intervention measure: 0.5, and (4) usefulness goal measure: 0.73. Consequently, not all of the quality measures achieve the defined quality target which results that the software cannot be released without improvements.

The quality measurement takes place regularly but at latest after each defined milestone in the project's lifecycle. The project team assesses whether the measured values fall in the acceptable ranges. Defining a fine-granular ordinal scale is also possible for the assessment, with many ranges in the acceptable and non-acceptable domain and with a transient in between. The measurement and assessment of the source code's quality takes place automatically after each check-in in the version control system.

Conclusions: Assuming a fictional but realistic project with the above settings, it is illustrated how the taxonomy in Table 3.1 can be used to select appropriate software product quality models for the project's needs. In addition, it has been demonstrated how the selected software product quality model can be tailored to the specific needs and how it can be used for quality measurement and assessment. Furthermore, to satisfy the automation endeavours, a different quality model was also introduced to measure the internal quality in a realistic scenario. Human intervention is necessary in the measurement of the external and the quality in use views in contrast to the internal quality.

3.4 Summary

This chapter presented the taxonomy of the 23 identified software product quality frameworks based on the quality manifestations, called quality views [106, 112], they are able to handle. Measuring software product quality and its assessment can involve quality models of a wide spectrum, ranging from simple hierarchic decomposition techniques to complex meta-models to deal with the abstract notion of software product quality [62, 64, 65, 70].

Nevertheless, the available software product quality frameworks and their concrete implementations significantly deviate in terms of their ability to capture the quality views. The differences in this ability of the identified software product quality models are important for making the right decision whether a particular quality model is suitable for a specific task or whether a statement made on the basis of a specific software product quality model holds for the software quality of the product as a whole. In addition, examining the applied quality views while comparing the quality measurement results of different quality models is inevitable.

By means of the two case studies based on fictional but realistic project settings in section 3.3, it is simply demonstrated how the taxonomy in Table 3.1 can be used in practice. The first case study illustrated the investigation of a given quality model with the outcome that the assessment results based on the selected model may only refer to the internal quality view, which implies that no conclusions may be drawn regarding the whole software product quality based solely on the given quality model's assessments. The second case study demonstrated how to select a software product quality model if the requirements of a given project are known; moreover, it was briefly highlighted how to tailor the selected quality model to specific project needs, how to define metrics to address the external and quality-in-use views, and how to perform the quality measurement and assessment.

The software product quality models with full-automation potential can assess only the internal view of quality, which can be captured by static code analysers. The software product quality models capable of handling internal, external and quality-in-use views offer partial automation features so that the results of the measurement of the external and quality-in-use views could be considered. This is not an impediment but a broader range of quality manifestations to consider for approximating the reality better while measuring and assessing quality. In contrast, if such models are integrated into the development pipeline with the expectation of full automation, then their capabilities are restricted to the internal quality view only.

Chapter 4

Identifying the Quality Properties of Execution Tracing

A significant part of the content of this chapter appeared in the author's article published in Applied System Innovation (MDPI – 2021, reference [69]). It reports the quality properties of execution tracing, their identification, and answers RQ4 in section 1.3.

4.1 Research Context and Related Works

Recent research in the field of logging improvement explicitly states the need for overall guidelines for preparing execution tracing; moreover, it confirms the absence of such directives [32, 33, 87, 157, 162]. Chen and Jiang analyse the reasons for log code changes and articulate six antipatterns for logging but do not form overall guidelines for its quality assessments [32, 33].

The issues addressed by the majority of the publications in the problem domain can be classified in three groups: (1) where to insert log statements in the source code [43, 61, 157, 158, 250, 252, 263, 264], (2) what to log to provide sufficient information for failure analysis [87, 251, 252, 263], and (3) how to log [33, 252, 263]. Tool support for selecting the appropriate places in the source code for inserting log statements consistently or for achieving optimal performance with respect to the amount of information logged and the speed of execution usually involve more than one of the issues mentioned above [43, 87, 250, 251, 252, 263, 264].

Li et al. in [158] investigate where to insert log statements in the code what the benefits and costs are; moreover, they list the tools for supporting the automatic insertion of log messages. Zhu et al. in [264] construct a machine learning based tool that processes changes between the versions in source code repositories and make suggestions on where to insert log statements. Chen and Jiang in [31] and Yuan et al. in [253] investigate whether error reports with logs are resolved more quickly than those without logs. In addition, Chen and Jiang in [33] define the below five different anti-patterns in the logging code and develop a tool to detect them:

Nullable object: While constructing the log message, an object that should deliver data to the log message is also null, which causes NullPointerException. Resolution: Checking the objects contributing to the log messages.

Explicit cast: An explicit cast can throw exception during the runtime conversion. Resolution: Removing the explicit cast.

Wrong severity level in the log message: Wrong severity level can cause serious overhead in the log data or lack of information in the necessary analysis. Resolution: Using the appropriate severity level.

Logging code smells: Retrieving the necessary information might result in long logging code with chained method calls like *a.b.c.d.getSmg()* . This causes maintenance overhead and deteriorates understanding. Resolution: Using a local variable or a method to provide the necessary information to the logging code.

Malformed output: The output cannot be read by humans. Resolution: Formatting it appropriately.

The Apache Common Best Practices document in [11] describes points to consider for creating consistent severity levels in the logging code statements but the description does not contain overall guidelines for logging. Galli et al. focus on the performance characteristic of execution tracing in [66], they examine the performance impacts of conventional and aspect-oriented approaches with regard to different different Java-version environments. Zeng et al. in [261] investigated logging practices in 1444 open-source Android applications and conclude that mobile applications deviate from desktop and server applications. They explicitly state that the usefulness of logging depends on the quality of logs and support is needed for logging decisions with respect to (1) severity level, (2) logging code location, (3) text in the log message, and (4) when to update the log statements [261].

Hassani et al. in [87] analysed log related issues in Jira reports and in the corresponding code changes as missing or outdated information can have considerable impact causing extra overhead with the analysis carried out by developers or wrong decisions made by system operators. They also identified the below root causes of the log-related issues, some of which are trivial to correct:

Inappropriate log messages: Log message is incorrectly formed including missing or incorrect variables. Majority of the log-related issues belong to this category.

Missing logging statements: Not enough information is logged. Each Jira ticket belongs to this category where further logging was added to help with the problem analysis.

Inappropriate severity level: An important message is logged with lower severity level or a less important message with a higher severity level.

Log library configuration issues: Different logging APIs require different configuration files. Issues related to the configuration files or their existence belong to this category.

Runtime issues: If logging statements produce a runtime failure including NullPointerException.

Overwhelming logs: Issues with redundant and useless log messages fall into this category, which make the log output noisy and difficult to read.

Log library changes: Changes required due to changes in the logging API.

Kabinna et al. in [119] conducted a study to examine whether the logging statements remain unchanged in the source code, which is important for the tools to process the logs; they also created a classification model to predict whether a log statement in the source code is likely to change. They found (1) the experience of software developers, who created the log statements, has an impact on determining whether it will change; (2) log statements introduced by the owner of the file are less likely to change in the future; (3) log statements in files with lower log density are more likely to change than log statements in files with higher log density. Yuan et al. [252] and Zhao et al. [263] confirmed that log messages are frequently the only tools to analyse system failures in the field in a postmortem way. Yuan et al. in [252] studied the efficiency of logging practices in five large software systems, with randomly selected 250 failure reports, and the corresponding source code. They found that postmortem analysis caused problems in 57% of the cases based on the provided log data. In addition, Yuan et al. discovered frequent, repetitive patterns among the logging failures, which motivated tool support [251, 252]. Their study with 20 programmers showed that improved logging resulted in reducing the failure diagnosis time by 60.7% [252] while 39% of the error manifestations were not logged at all. Yuan et al. formulate the below recommendations regarding what to log:

1. Log detected errors regardless the existence of code to handle it.
2. Log errors of system calls.
3. Log CPU, memory usage and stack frames for signals SIGSEGV, SIGTERM.
4. Log switch-case default branches.
5. Log exceptions including input validation and resource leaks.
6. Log the branches of unusual input or environment effects that might not be covered by testing.

Zhao et al. in [263] implemented tool support for placing log statements in the code without human intervention and without the need of domain knowledge. Their tool collects execution path information by means of a logging library and evaluates the frequencies of different execution paths to compute the optimal places for log statements [263]. In [43], Ding et al. design and implement a cost-aware logging system, which also considers the performance degradation caused by logging, to decide which useful log messages to keep and which less useful ones to discard. Kabbina et al. in [118] analysed log library migration projects at Apache Software Foundation (ASF) based on Jira ticket reports and git commit history. They found that (1) 33 of 223 ASF projects underwent log library migration, (2) log library migration is not a trivial task as the median number of days to complete the migration for the 33 projects was 26 days, (3) flexibility (57%) and performance (37%) are the main drivers for the log library migrations performed, (4) log library migration is error-prone: more than 70% of the projects had post-migration bugs, on average 2 issues per project including missing dependencies, configuration, interactions between old and new libraries, and (5) the performance improvement achieved by the log library migration is usually negligible. Shang et al. examines the possible ways for operators and administrators (1) to elicit and understand the meaning, and impact of the log lines and (2) to identify a solution for resolving them [216]; while in [217] Shang et al. found that log-related metrics can also be effective predictors for post-release defects. They found that log-related metrics and post-release defects are as strongly correlated as the pre-release defects and the post-release defects, which is known to be the strongest predictor at present [217]. The reason for this high correlation can be explained by the attitude of the software developers, who tend to add more logs to the code about which they have concerns [217]. Consequently, decreasing the number of log statements in a source file does not imply an improvement in the source code quality.

In conclusion, several publications state that no industrial standard and very limited or no guidelines are available to support software professionals to create logging in an application [32, 33, 87, 157, 162]. Therefore, it is necessary to identify the quality properties of execution tracing, which forms a foundation to define overall guidelines and to define a quality model to make quality assessments possible in the problem domain.

4.2 Research Protocol

The research to identify the quality properties of execution tracing comprises the following steps: (1) analysing the accumulated experiences of software professionals related to the influencing factors for execution tracing quality, (2) examining the literature with regard to influencing characteristics for execution tracing quality, (3) establishing a consensus between the two different approaches, and (4) validating the results with regard to the works related to the present study.

4.3 Research Methods

The applied methodology aims to elicit all possible properties that influence execution tracing quality. To achieve this goal, data from three different sources were investigated: (1) direct experiences of software professionals as described in section 4.3.1; (2) publications on execution tracing, identified by defined search queries in different scientific document databases and by reference searches as introduced in section 4.3.2; and (3) the works related to this study as presented in section 4.1, which were discovered by traditional literature review. A separate section 4.6 is devoted to the validity of the research.

After the data collection, the data corpus was coded [210], and the variables, i.e., the quality properties of execution tracing, defined. The data coding was performed separately (1) from the data collected from the experiences of software professionals, and (2) from the data collected from the identified publications. The outcomes were compared, deviations resolved and the variables defined were also validated using the publications related to this study in section 4.1. Figure 4.1 illustrates the above process, which results in defining the quality properties of execution tracing. Each stage of the research is explained in the subsequent sections below.

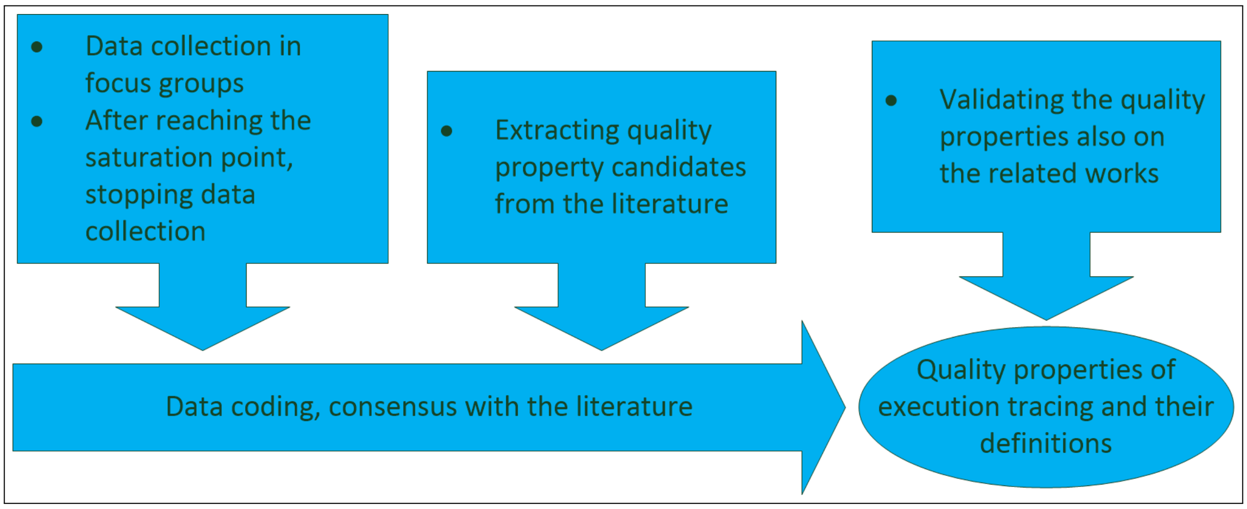
• Data collection in focus groups
• After reaching the saturation point, stopping data collection
• Extracting quality property candidates from the literature
• Validating the quality properties also on the related works
Data coding, consensus with the literature
Quality properties of execution tracing and their definitions

Figure 4.1: Identification of the Quality Properties of Execution Tracing, Source: [69]

Quality properties of execution tracing can be extracted from experiences of software professionals (software architects, software developers, software maintainers, software testers, and system administrators). Thus, determining these quality properties requires research that is empiric and qualitative in nature as human experiences need to be collected and processed. In addition, analysing the identified literature also involves qualitative research. The empiric study comprised the following steps:

1. Sampling the study population to construct focus groups;
2. Collecting ideas and experiences of software professionals related to execution tracing quality in the focus groups;
3. Analysing and coding the collected data to form the quality properties on which execution tracing quality depends, i.e., defining research variables;
4. Comparing the identified quality properties based on the focus groups with those extracted from the literature and resolving potential contradictions. Checking the results on the related studies introduced in section 4.1.

4.3.1 Data Collection from Software Professionals

The data collection was carried out in focus groups by means of brainstorming [114, 188] as explained below.

Sampling Method

Non-random sampling was selected to choose individuals from the study population as it offers extremely useful techniques for collecting data on phenomena known only to a limited extent [142]. The study population was defined by the international software companies located in Hungary, the employee count of which exceeds 1.000 and the main activity of which belongs to the IT service delivery sector (TEAOR code 620x at the Hungarian Central Statistical Office). Large international software houses possess internationally defined style guides, and development principles; consequently, the geographical localisation imposes no significant risk to the study from the point of view of the accumulated experiences of the software professionals involved in the research.

Data Collection in the Focus Groups and Data Coding

Experiences of the sampled individuals were collected in moderated focus groups in the following manner:

1. Ideation phase: The following question was introduced to each group "What properties influence execution tracing/logging quality?" and the participants had a short time to think over the question individually in the group, in silence.
2. Idea collection phase: The participants could freely express their ideas in a couple of words, which were recorded. No idea collected was allowed to be exposed to critic in the group at this stage.
3. Evaluation phase after no more ideas appeared:

(a) The same items in the collected list were looked for and these were merged into one;

(b) Each group member received the same amount of score which had to be distributed among the collected items, i.e. constant sum scaling was applied [171], to show which ideas represented the most important entities in the group's opinion. The higher score value was assigned to an item, the higher its importance was.

4. After concluding a focus group, the moderator summed the scores assigned by the participants to each of the collected items in the given group. Thus, each focus group produced a raw list of ideas, i.e., quality property candidates, with a score of importance, which represented the view of the group.

The technique used for collecting the ideas in the above manner is known as brainstorming, which was developed by Osborn and later sophisticated by Clark to create, collect and express ideas on a topic [188]. The main principle of the method consists of [114]: (1) group members must have the possibility to express ideas while criticism is ruled out, and (2) the ideas recorded can be developed and combined by other group members; this way the presented ideas generate new ones. Before and after the idea generation phase, an additional phase must take place, which includes thinking over the starting question and evaluating the collected ideas. Brainstorming was combined with constant sum scaling [171] in the present study during the evaluation phase for measuring the importance of each collected item in the opinion of the group.

Data coding is applied to gain structured information from unstructured data; moreover, it aids to explore or verify relationships between entities in the data corpus [210]. Brainstorming produces semi-structured data. As the adapted process with importance score assignment shows, a list of quality property candidates for execution tracing were formed, with a score value assigned to each quality property candidate. The preparation for data coding started in the focus groups when the participants looked for the identical items in the produced list and while the focus group members assigned scores of importance to each item. After having a scored list of quality property candidates from the focus groups, the produced data were compared to all the data produced by all the previous focus groups and the same or very similar items were matched (1) to determine the stop condition for conducting further focus groups, and (2) to progress with the data coding. The activities included:

1. Determining how many quality property candidates a focus group identified;
2. Determining how many new quality property candidates a focus group identified, which the previous focus groups have not yet discovered;
3. Determining how much score of importance to the new quality property candidates was assigned;
4. Normalising the number of items identified and their scores of importance to be able to compare these values as the number of the participants slightly deviated in the different focus groups;
5. When the score of importance of the newly identified items in a focus group approximated zero, i.e., a saturation point was reached, the data collection was concluded.

By comparing the list of quality property candidates collected in each focus group, a common list was created where the same or very similar items, produced by the different focus groups, were matched and their score values added after the above preparation steps for further data coding, which included:

1. Each item in this list was labelled with a matching word possibly from the merged description of the item;
2. Similar labels were associated while keeping the separation with the distant labels. Thus, the items were assigned to clusters.

The clusters created by associating the similar labels in the above manner form the quality properties of execution tracing based on the focus groups. The quality properties, i.e., the variables on which execution tracing quality depends, were compared to the variables extracted from the literature.

4.3.2 Data Collection from the Literature and Data Coding

The data collection with regard to the literature included the following steps:

1. Publications dealing with execution tracing or software logging or a related area were identified. The following logical query was executed in the scientific document databases: ("execution tracing" OR logging) AND quality AND maintainability AND software AND (model OR framework). The concrete queries performed in each scientific document database are recorded in appendix B.4.1. In addition, the quality property candidates extracted from the publications are listed in appendix B.4.
2. The identified publications were scrutinised to find out whether they contain any quality property candidate for execution tracing
   (a) explicitly, by naming it directly;
   (b) implicitly, by a longer description or by proposed actions to avoid negative consequences;
3. The implicit quality property candidates were transcribed to explicit ones as shown in appendix B.4.2.
4. Data coding was performed to construct the quality properties of execution tracing based on the literature.

The searched scientific document databases covered: (1) ACM Digital Library, (2) IEEE, (3) EBSCO Academic Search Premier, (4) Scopus, (5) Science Direct, and (6) Web of Science.

Inclusion criteria: Every identified publication is included unless any of the below exclusion criteria apply.

1. If the publication identified is not related to execution tracing or auditing, then it is excluded.
2. If the publication is not a primary source, then it is excluded. Books are not considered as primary source.

4.3.3 Consensus on the Results

Quality properties of execution tracing were identified separately from the focus groups and from the literature. Thus, after performing the data coding process, two sets of quality properties were defined, which underwent further analysis to investigate agreements and potential contradictions. In conclusion, consensus was established between the two sources to arrive at the final list of quality properties of execution tracing.

4.3.4 Validation

The defined quality properties of execution tracing were investigated in the context of the works related to this study to approve them.

4.4 Analysis of the Collected Data

This section presents the implementation of the data collection and the data coding from (1) the focus groups, and (2) from the literature. While highlighting the results from the literature, the consensus with the focus groups is also reported.

4.4.1 Data Collection in the Focus Groups and Data Analysis

The data collection was conducted in seven focus groups which involved 49 participants, software architects, software developers, software maintainers, software testers and system administrators, who are intensively affected by the use of execution tracing in the course of their work. During the evaluation phase in a given focus group, each participant received 20 scores to be distributed among the identified quality property candidates. After concluding a focus group, the collected items were compared with the results of the previous focus groups, and the number and score values of the newly identified items were computed.

When the normalised importance score value of the identified new quality property candidates dropped below a threshold score 2 (10% of the individual score to distribute) in the last two focus groups, then the importance score of the identified new items approximated zero. This indicates that the saturation point was reached. The new items identified were not important in the sight of the focus group; and no further focus groups were organised. The distribution of the identified quality property candidates, the identified new quality property candidates, the assigned importance scores, the number of participants and the normalised importance score per focus group are highlighted in Table 4.1.

Table 4.1: Number of Items and Distribution of Importance Score in the Focus Groups, Source: [69]

Group	Number of Identified Items	Number of Identified New Items	Assigned Score Value to New Items	Number of Participants	Normalized Scrore Value of New Items for One Participant
1.	15	15	160	8	20
2.	14	9	76	5	15.2
3.	19	12	62	7	8.86
4.	8	1	0	8	0
5.	11	4	30	8	3.75
6.	6	0	0	6	0
7.	7	1	7	7	1

The collected execution tracing quality property candidates were analysed and the same items identified by various focus groups were matched; moreover, their score values were summed up as described in section 4.3. Consequently, a list of 42 different items with cumulated score values was produced. The data coding process had further three stages: (1) each item was labelled with a short term possibly contained in the item description, (2) similar labels were grouped in clusters, and (3) the clusters created with their scores of importance were analysed to merge the similar ones. The last stage was repeated in three cycles. The clusters, based on the output of the focus groups, constitute the quality properties of execution tracing. The list of the execution quality property candidates with the importance scores, the labels assigned and the clusters, in which the items were grouped at the end of data coding cycle two, is added to Table B.8 in Appendix B.3; while, the merged item descriptions by the clusters are presented in Table B.9 in Appendix B.3. The quality properties of execution tracing based on the output of the focus groups are summarised in Table B.10 in Appendix B.3 after concluding data coding cycle three.

In summary, the research with the focus groups delivered the below variables, on which execution tracing quality depends, with their corresponding importance scores. The participants expect the quality property *accuracy and consistency* to be the most important, while the quality property *security* is the least important one, which is in agreement with the goal of execution tracing: localise errors in the application.

Accuracy and Consistency, Importance score: 521 This variable defines the quality of the execution tracing output with regard to its content but not how the output appears and how legible it is. It shows how accurate and consistent the output of execution tracing is, which includes whether the trace output contains (1) the appropriate details in a concise, non-redundant manner, (2) the origin of the trace entry can definitely be identified in the source files, (3) date and time (including the fractals of the second), (4) the host, (5) the process, (6) the thread id are traced for each entry, (7) exceptions are traced once with full stack trace, (8) all important information is traced, (9) the build date and (10) the software version are traced, (11) environment variables are traced, (12) higher level actions, business processes can definitely be identified based on the trace entries, (13) the flow control is identifiable over component boundaries, (14) the transactions can definitely be identified and followed over component boundaries, (15) the sequence of the trace entries are kept in chronological order, (16) there are different severity levels, (17) the severity levels are consistently used across the application, (18) the same error is traced with the same message across the application in a unified manner, including error codes with defined messages.

Legibility, Importance score: 232 This variable defines the quality of the appearance of the execution tracing output but not the content itself. It shows how legible and user-friendly the output of execution tracing is, which includes whether (1) the trace output is well-structured, (2) the trace is appropriately segmented in the case of multi-threaded and and multi-component applications, (3) the necessary variable values are traced not just object or function references, (4) the formatting is appropriate, (5) the trace output is easily searchable in a performant manner, (6) the trace output is legible in text format possibly without an external tool, (7) the language of the trace output is unified (e.g., English).

Design and Implementation, Importance score: 187 This variable defines the quality of the design and implementation. It shows how well the execution tracing mechanism is designed and implemented, which includes whether (1) the trace output is accessible and processable even in the case of a critical system error, (2) the trace output is centrally available, (3) the trace mechanism is designed and not ad hoc also with regard to the data to be traced, (4) the trace size is manageable, (5) if the trace is written to several files whether these files can be joined, (6) the tracing mechanism is error-free and reliable, (7) configuration is easy and user-friendly, (8) event-driven tracing is possible, i.e., where and when needed, more events are automatically traced, where and when not needed, then less events, (9) the performance of the trace is appropriate with regard to the storage, (10) the structure of the trace output is the same at the different components or applications in the same application domain, (11) the trace analysis can be automated, (12) standard trace implementations are used.

Security, Importance score: 40 This variable defines the quality of security with regard to the execution tracing mechanism and its output. It shows how secure the execution tracing mechanism is, covering the trace output, including whether (1) the trace mechanism is trustworthy from audit point of view in the case of auditing, (2) data tampering can be ruled out, (3) sensitive information is appropriately traced and its accessibility is regulated, (4) personal information is protected including the compliance with the GDPR regulations.

4.4.2 Data Collection from the Literature and Data Analysis

As presented in section 4.3.2, a literature review was performed to discover publications which formulate requirements or express wishes in an explicit or implicit form with regard to execution tracing. In the course of the query design and test for the different scientific document databases, the less specific queries, from the point of view of the investigation, returned unmanageably large data sets with many false positives while specific queries reduced the returned documents drastically. For this reason, a relatively specific logical query was selected, as shown in appendix B.4.1: ("execution tracing" OR logging) AND quality AND maintainability AND software AND (model OR framework). In addition, the automatic search was complemented by reference search and traditional key-word search to enhance the number of publications in the result set.

The search identified 58 publications in the scientific databases with high number of duplicates. The result set reduced to 39 publications without duplicates, which included 15 sources that contained relevant data for execution tracing. In addition, no research was identified that would have been explicitly dealing with the elicitation of the quality properties of execution tracing.

Recording and Extracting Data

The data extracted from the publications underwent data coding to identify the quality properties of execution tracing based on the literature.

Data Coding from the Literature

Each publication with execution tracing related data was scrutinised; furthermore, the data with regard to any potential quality property of execution tracing were extracted and recorded. Distinction was made in the recording between explicitly and implicitly defined quality properties, i.e., if a quality property directly appeared in the text, e.g., "accurate", then it is an explicitly defined property; however, if solely its negative appeared as something to avoid or a wish was articulated in longer text to achieve, then it was considered as an implicitly defined property. In this latter case, the text was recorded and transcribed to a short explicit term which was equivalent to the implicitly described quality property. The final transcribed list of properties were further processed, the similar items were grouped in one cluster, while the dissimilar ones were kept separately as illustrated in section 4.4.1. The transcription of the identified publications is presented in table B.11 in appendix B.4.2 while clustering is presented in appendix B.4.3.

In summary, the clusters provide four quality properties based on the literature:

Design and Implementation: This variable shows how well the execution tracing mechanism is designed and implemented. The relevant quality property candidates from the literature for this cluster are listed in table B.12 in appendix B.4.3.

Accuracy: This variable shows how accurate the output of execution tracing is with regard to its content. The relevant quality property candidates from the literature for this cluster are listed in table B.13 in appendix B.4.3.

Performance: This variable shows how performant the implementation of the execution tracing mechanism is. The relevant quality property candidates from the literature for this cluster are listed in table B.14 in appendix B.4.3.

Legibility and Interpretation: This variable shows how legible the output of execution tracing is and how well it can be interpreted. Thus, this variable illustrates the quality of the appearance of the content of the trace output. The relevant quality property candidates from the literature for this cluster are listed in table B.15 in appendix B.4.3.

Consensus with the Focus Group Results

The literature identified by the search, documented above, did not include any empiric research regarding the discovery of the variables on which execution tracing quality depends. Nevertheless, 15 publications could be identified which involve in execution tracing or in a related area such as monitoring or forensic log data analysis. After extracting and data coding the relevant information from these publication with regard to execution tracing, an agreement was observed between the variables obtained from the literature and the variables obtained from the focus groups. The variables (1) *accuracy*, (2) *legibility*, (3) *design and implementation* appear both in the focus groups and in the literature; however, the variable *performance* is established from the literature only. On the other hand, security forms a variable on its own based on the output of the focus groups. Nevertheless, security also appeared in the literature with less emphasis in relation to cloud infrastructures [184, 232, 233]. For this reason it was data coded as part of the variable *design and implementation*. In summary, the defined variables based on the literature show no contradictions with the variables based on the focus groups. The slight deviations between the two sources may be due to the lack of empiric research in the identified literature on the field. It is also worth mentioning that some of the analysed publications did not involve in execution tracing but in areas such as cloud data centre monitoring or log forensic analysis.

4.5 Definition of the Quality Properties of Execution Tracing

While defining the final variables, the number of items in a given cluster, the importance score and consistency of the identified clusters were considered. In addition, a final cleaning of the data clusters was done. The final variable definitions are provided below with a scale definition which supports possible measurements along these dimensions. Ratio scale was selected because it supports all kinds of mathematical operations in contrast to interval and ordinal scale types [142, 171]. It is important to stress that the below four variables do not form a quality model, yet, but they offer a foundation on which a quality model can be built. The scale range is [0, 100] in the case of each variable. The zero value expresses the complete absence of the property while the maximum value expresses the complete presence.

Variable: ***Accuracy***

Range: [0, 100]

Scale type: Ratio

Definition: This variable defines the quality of the execution tracing output with regard to its content, to what extent it trustworthily and consistently helps to identify the cause, the date, the time, and the location of the issue in the source code but not how legible the trace output is.

Depending on the context, it might include whether (1) the trace output contains the appropriate details in a concise, non-redundant manner, (2) the origin of the trace entry can definitely be identified in the source files, (3) date and time with appropriate granularity, in the case of several nodes with clocks synchronised, are traced for each entry, (4) the host, (5) the process, (6) the thread id are traced for each entry, (7) exceptions are traced once with full stack trace, (8) all important information is traced, (9) the build date and (10) the software version are traced, (11) environment variables are traced, (12) higher level actions, business processes can definitely be identified based on the trace entries, (13) the flow control is identifiable over component boundaries, (14) the transactions can definitely be identified and followed over component boundaries, (15) the sequence of the trace entries are kept in chronological order, (16) there are different severity levels, (17) the severity levels are consistently used across the application, (18) the same error is traced with the same message across the application in a unified manner, including error codes, (19) data sources and data destinations can definitely be identified, (20) metadata of the file system including time for modified, and accessed attributes are traced, (21) metadata of backups and recoveries are traced, (22) the necessary variable values are traced not just object or function references.

Variable: ***Legibility***

Range: [0, 100]

Scale type: Ratio

Definition: This variable defines the quality of the appearance of the execution tracing output but not the content itself. It shows how legible and user-friendly the output of execution tracing is.

Depending on the context, it might include whether (1) the trace output is well-structured, (2) the trace is appropriately segmented in the case of multi-threaded and multi-component applications, (3) the formatting is appropriate, (4) the trace output is searchable with ease in a performant manner, (5) the trace output is legible in text format without conversion, (6) the language of the trace output is unified (e.g., English).

Variable: ***Design and Implementation of the Trace Mechanism***

Range: [0, 100]

Scale type: Ratio

Definition: This variable defines the quality of the design and implementation of the execution tracing mechanism with regard to how reliable, stable and sophisticated it is, to what extent and how easy it can be configured, activated and deactivated, whether sophisticated mechanisms for measuring performance are implemented to reduce the impact on the application and the interference with the actions of execution.

Depending on the context, it might include whether (1) the trace output is accessible and processable even in the case of a critical system error, (2) the trace output is centrally available, (3) the trace mechanism is designed with regard to the data to be traced, (4) the trace size is manageable, (5) if the trace is written to several files whether these files can be joined, (6) the tracing mechanism is error-free and reliable, (7) the trace configuration is easy, user-friendly and configuration changes do not implicate a restart, (8) event-driven tracing is possible, i.e., where and when needed, more events are automatically traced, where and when not needed, then less events, (9) the performance of the trace is appropriate with regard to the storage, (10) the structure of the trace output is the same at the different components and applications in the same application domain, (11) the trace mechanism offers automation feature such as filtering duplicate entries, (12) standard trace implementations are used (e.g., log4j), (13) the trace mechanism can deal with increasing data volume, (14) the trace mechanism is capable of dealing with parallel execution, (15) the trace mechanism is capable of measuring performance (a) at different levels of the architecture (b) with different granularities and (c) not only the response time but also the quality of data returned can be investigated based on the trace, (16) the trace mechanism offers the possibility to make queries on timestamps, (17) the trace mechanism is capable of making a prediction about the system's availability based on the traces, (18) the trace mechanism is capable of generating usage statistics, (19) the trace mechanism is capable of measuring quality of service metrics for SLAs, (20) the trace code insertion can be done automatically in the application, (21) the trace mechanism is capable of replaying and simulating sequence of actions based on the traces, (22) the trace data are available as the events occur not only after the trace session is finished, (23) the trace data are available also through APIs, (24) the output format of the trace can be changed including DB and text file formats, (25) the trace code can be located in separate modules and not intermingled into the application code, (26) the trace mechanism is capable of correlating actions on different nodes in the case of distributed application, (27) the tracing mechanism is capable of keeping its perturbations on a possible minimum level with regard to the application, (28) the CPU usage can be traced, (29) the memory usage can be traced, (30) the trace mechanism is capable of tracing the following properties in the case of real-time applications: (a) task switches, which task, when, (b) interrupts, (c) tick rate, (d) CPU usage, (e) memory usage, (f) network utilisation, (g) states of the real-time kernel, which tasks are waiting to execute (waiting queue), which tasks are running, which tasks are blocked.

Variable: *Security*

Range: [0, 100]

Scale type: Ratio

Definition: This variable defines how secure the execution tracing mechanism and its outputs are, i.e. it shows to what extent the execution tracing mechanism and its outputs are exposed to vulnerabilities and how likely it is that sensitive information leaks out through them.

Depending on the context, it might include whether (1) the trace mechanism is trustworthy from audit point of view in the case of auditing, (2) data tampering and tracing fake actions can be ruled out, (3) sensitive information is appropriately traced and its accessibility is appropriately regulated, (4) personal information is protected including the compliance with the GDPR regulations.

4.6 Validation

All the investigations and results of the related works, summarised in section 4.1, can be placed in a four dimensional space determined by the variables defined above: (1) accuracy, (2) legibility, (3) design and implementation, and (4) security. The majority of the studies listed in section 4.1 deal with three dimensions at most.

None of the related studies provided information that would have required a change in the defined dimensions. Consequently, the related works confirm the quality properties of execution tracing defined in this study.

4.6.1 Validity Indicators Considered

Content, construct, internal, and external validity are considered. In addition, the usual indicators for validity in qualitative research: credibility, transferability, dependability, confirmability are also examined [142].

Content validity [211] in the context of the study refers to the extent to which the universe of all possible items were considered while the possible variable candidates were being collected. While identifying the potential variable candidates that influence execution tracing quality, both (1) the academic literature, and (2) the personal experiences of software professionals were considered. Search strings in the scientific document databases were recorded and the queries were complemented with reference searches. Explicit and implicit variable candidates were extracted and the transcribed as illustrated in section 4.4.2. In addition, the collected variable candidates from the focus groups of software professionals were recorded and the data collection process was ceased when the number of the newly identified items dropped below a defined threshold value. After constructing the variables from the collected data, they were verified based on data of the related works on logging as reported in section 4.1.

Construct validity shows the extent to which the findings reflect the content of the constructs in the phenomenon investigated [211]. Quality assessment examines to which extent concrete quality requirements are satisfied. This activity is performed along defined variables, i.e. quality properties, in all the identified quality models in [65]. Moreover, variables are necessary to define to be able to capture a manageable part of quality. Consequently, variables from the universe of the identified potential quality property candidates were defined. The variables were formed (1) from the data extracted from the publications, and (2) from the data collected from the sample of software professionals. Both sets of data were processed and data coded separately. The data coding yielded the same results with minimal deviations. The final variables were defined with the data from both sources and also validated on the findings of the related works.

Internal validity ensures that the study conducted represents the truth with regard the phenomenon investigated [197]. The data collection process was extended to all reasonably possible items to ensure content validity. The data coding process to form the variables was performed by the author, checked by an internal review and by independent peers who reviewed the research report published in [69]. In addition, construct validity was also considered in the scope of the study.

External validity refers to generalizability of the research. The variable definitions derive from data from two different sources: (1) the academic literature identified as documented in section 4.3.2, and (2) empiric research with focus groups at large international software houses with employees located in Hungary as introduced in section 4.3.1. The results were also validated on the related works in section 4.1. The data from the academic literature identified for data collection and the literature identified with related works stem from different parts of the world and corroborate the findings based on the focus groups. Thus, the geographical localisation of the focus group research does not impose a limitation to the present study.

Credibility refers to the judgement whether the research manages to express the feelings, and opinions of the participants [142]. This indicator is satisfied with the data collection in the moderated focus groups as the collected quality property candidates were examined in the course of the evaluation phase by the whole group and a score value was assigned to each item that represents its importance within the group. Moreover, the data collection phase in each group was moderated so that no critic or long monologues could be expressed during this phase, which would have retarded or suppressed certain opinions.

Transferability refers to the extent to which the achievements of the research can be transferred to other contexts or can be generalized [142]. In the scope of the study being conducted, transferability refers to whether the same quality properties could be established as the quality properties of execution tracing in general, regardless of geographical localisation or software companies involved in the research. The variables were defined from data based on two different sources: (1)identified international publications and (2) empiric research with focus groups located in Hungary. In addition, the defined variables were validated on the related works which also stem from international context. Moreover, the sample for the empiric research with the focus groups contains companies with premises spanning over the boundaries of Hungary and having international coding guidelines. Consequently, the geographical localisation of the focus groups does not impose a restriction to the research with regard to the studied phenomenon.

Dependability describes whether the same results would be achieved if the research process was applied more times [142]. The research process was extensively documented; moreover, the validity indicators as introduced above were also satisfied. The data collection with the focus groups was ceased only when a saturation point was reached in identifying the collected items.

Confirmability shows whether the achievements can be confirmed by others [142]. Two expert software professionals with many years of experience in the software industry were asked whether the defined variables in their opinion are able to cover all necessary aspects of execution tracing quality. The experts affirmed the results.

4.7 Summary

No framework exists that would offer a basis to describe execution tracing quality in a quantitative manner. Recent publications state the necessity of creating guidelines to assist developers with logging implementations [32, 33, 87, 157, 162], some of which attempt to formalise concrete guidelines for specific parts of logging but not for the whole. Moreover, many of the previous studies solely perform an analysis of source code changes in version control systems. Consequently, they can only address the manifestation of internal quality as they consider source code artefacts. Nevertheless, addressing quality requires the examination of logging while it is in operation and the evaluation of human experience is inevitable to decide how the running implementation relates to its environment, how the users, in this case the software developers, software maintainers, software testers and system administrators, perceive execution tracing quality as a whole instead of focusing only on its individual properties such as performance or informativeness.

Defining execution tracing quality properties is the first step towards creating a quality model for execution tracing. The current research fills this gap by identifying and defining the variables, i.e., the quality properties, on the basis of which the quality of execution tracing can be judged. The present study analyses the experiences of software professionals in focus groups at multinational companies, and also scrutinises the academic literature to elicit the mentioned quality properties. Moreover, in the domain of qualitative research, the present study also contributes to knowledge with the combination of methods while computing the saturation point for determining the number of the necessary focus groups to be held. Furthermore, to pay special attention to validity, content, construct, internal and external validity were considered in addition to the indicators of qualitative research: credibility, transferability, dependability, and confirmability.

The quality properties synthesized from the focus groups and the variables synthesized from the literature were compared to resolve contradictions, and to address deviations. The final variables, on which execution tracing quality depends, were established by reaching a consensus between the two sources: (1) the empiric research with the focus groups and (2) the data collected from the literature. In addition, the final variables were also validated on the data extracted from the related works. The literature identified by the automatic and manual searches has roots in several different countries; thus, the geographical localisation of the focus groups imposes no risk to the validity of the study.

The variables identified in the present study can be defined as ratio, interval or ordinal scale variables with guidelines for the values they can take. The author selected ratio scale because it allows performing the most mathematical operations, which is suitable for modelling. On the other hand, the definition of a ratio scale type determines that the zero value expresses the complete absence of the property measured while the maximum value expresses the full presence of the property measured.

The variables defined in the study make possible to address internal, external and quality-in-use aspects of software product quality with regard to execution tracing. In conclusion, the definition of concrete guidelines for logging with regard to these variables would make possible to articulate rules that enable the comparison of defined quality targets and the actual state.

Chapter 5

Modelling Execution Tracing Quality

A significant part of the content of this chapter appeared in the author's article published in Mathematics (MDPI – 2021, reference [68]). It reports how execution tracing quality can be modelled and it answers RQ5 in section 1.3.

Execution tracing is a tool used in the course of software development and software maintenance to identify the internal routes of execution and state changes while the software operates. Its quality has a high influence on the time of the analysis devoted to locate software faults as explained in the previous chapters. Nevertheless, execution tracing quality has not been described by a quality model yet, which is a limitation when measuring software product quality. In addition, such a model needs to consider uncertainty as the underlying factors involve human analysis and assessment. The goal of this chapter is to address both issues and to fill the gap by defining a quality model for execution tracing, which makes possible to place each previous individual endeavour for logging improvement and to see their relations to each other and to the quality as a whole. Furthermore, modelling execution tracing quality facilitates to articulate general guidelines or requirements to achieve good quality. In addition, it supports the measurement and assessment of logging quality when comparing the same project at different milestones of the development and maintenance or when comparing different projects.

The data collection was conducted on a defined study population with the inclusion of software professionals to consider their accumulated experiences; moreover, the data were processed by genetic algorithms [51] to identify the linguistic rules of a fuzzy inference system [255, 257, 259].

The present study preferred fuzzy logic [255, 257, 258] to evidential reasoning [245, 249] due to its efficient modelling language of linguistic rules and its capability to build adaptive systems. Linguistic rules are if-then rules, formed with linguistic variables modelled as fuzzy sets [257, 258]. Since fuzzy sets make possible the description of a partial membership to a given set, the antecedents of the linguistic rules do not require a full match with the given conditions. In such cases, the consequent part is also partially applied according to the degrees of memberships. The effects of the different consequent parts of the linguistic rules are combined by the inference mechanism Takagi-Sugeno-Kang [208] in the present study. The basic steps to define a fuzzy inference system can be summarised as follows: (1) determining the input variables and their domain partitions, i.e. the regions to describe with linguistic labels to be represented with fuzzy membership functions, (2) determining the output variables and their domain partitions (3) determining the type of inference, (3) defining the membership functions for the inputs and for the output, (4) defining the linguistic rules, (5) testing the system. The parameters of the membership functions and the number of the linguistic rules can be tuned adaptively [116, 117, 175, 176].

5.1 Research Context and Related Works

Studies related to modelling execution tracing quality lie on two different fields: (1) logging quality, which forms the problem domain, and (2) quality modelling with adaptive methods including the consideration of uncertainty, which aims at developing tools to deal with the problem domain from the point of view of modelling. The former has already been introduced in section 4.1, while the latter is summarised below.

As human experiences need to be considered while modelling execution tracing quality, artificial intelligence (AI) methods were used in the present study. AI and machine learning have been applied in the field of software quality modelling, including software defect prediction, bug report classification, uncertainty modelling, and related multi-criteria decision making problems as reported in this section. Nevertheless, none of the listed studies focused on logging quality.

Lai et al. in [145] lay down key performance metrics for computer network quality and predict the customer satisfaction by means of recurrent neural networks. Software defect prediction became an intensively researched field with different machine learning methods, including AI, to forecast errors or system breakdowns [193]. Pradhan et al. use machine learning to predict software defects in large systems [203]. Madera and Tomon apply machine learning to identify the source code artefacts that are endangered by software defects [170]. Software defect prediction is highly influenced by the amount of data available to train the machine learning models including neural networks, SVM, KNN, K-Means Clustering, Naive Bayes, decision trees, logistic and linear regression models; and their combinations with ensemble learning [125]. Khan et al. use transfer learning to utilise the data of different projects and to overcome the data availability barrier [125]. Blas predicts the quality of the software by means of simulation and modelling based on the architecture defined in [24]. Xing et al. predict software quality based on complexity-related quality metrics by means of SVM [247]. Lafi et al. apply classification for bug reports to reduce the maintenance efforts [144]; their method assigns an action to the fault report with the labels: (1) faults to repair, (2) new functionality desired, (3) existing functionality needs change, and (4) the software needs adaptation to new environments.

Ubayashi et al. in [234] give an account of the uncertainty developers face in software development. Their publication reveals the potential for the use of fuzzy logic. Singh et al. determine four quality metrics [221]: (1) separation of concerns, (2) coupling, (3) cohesion, and (4) size, to describe the reliability of aspect-oriented software systems while they use fuzzy modelling. The linguistic rules were constructed on basis of the opinions of experts. Li et al. in [160] define a quality model, using concepts from the ISO/IEC 9126 standard, for software products involved in digitalising antiquarian resources with the use of the fuzzy logic and apply fuzzy mathematics to evaluate the membership of the defined quality properties labelled with words: *excellent, good, general, unqualified*. Liang and Lien in [164] extend the quality model of the ISO/IEC 9126 standard to cover enterprise resource planning (ERP) software related properties; moreover, they describe the selection of the optimal ERP software as a multi-criteria decision making problem and use fuzzy analytic hierarchy process to produce a ranking with regard to the criteria considered [164]. In addition, Aggarwal et al., Nerurkar et al., Canfora et al. and Mittal and Bhatia apply fuzzy modelling to consider qualitative and quantitative data for assessing reusability and maintainability of software components in [6, 29, 181, 186]. Galli et al. in [62, 63] test the performance of the different fuzzy inference methods with different membership functions in the context of modelling execution tracing quality and conclude that (1) Takagi-Sugeno-Kang inference with overlapping Gaussian membership functions achieved the best performance, and (2) manual linguistic rule creation is error-prone, contradictions can easily be introduced into the rule-base. For these reasons the study concludes that Takagi-Sugeno-Kang inference with overlapping Gaussian-shaped membership functions should be used to describe execution tracing quality with an adaptive fuzzy system, where the linguistic rules are constructed automatically from the collected data. Malhotra and Lata outlines the machine learning approaches used for software maintainability in [172].

5.2 Research Protocol

The research included the following steps: (1) investigating the current state of the research in the field; (2) determining the goal of the study; (3) designing the data collection including (a) the identification of the study population, and (b) the determination of the sampling method; (4) collecting the data by an online questionnaire [67]; (5) performing exploratory data analysis, and preprocessing the data; (6) constructing the model by the extraction of the linguistic rules of a fuzzy inference system by means of genetic algorithms from the collected data; (7) verifying the constructed model's performance through a different machine learning approach, ANFIS [116, 117]; (8) carrying out pre-validation and model adjustments in a mini-focus group; (9) validating the constructed model by international experts through another online questionnaire and tuning the model based on their feedbacks; and (10) writing the research report.

5.3 Research Methods

This section reports the implemented research methods as illustrated in figure 5.1: data collection, modelling, pre-validation and validation stages. Since the nature of the study is empirical, experiences of software professionals within the IT domain are collected and processed.

Uncertainty and vagueness are inherently present in the software product quality measurement and assessment process as explained in the preceding sections and in section 5.1. A previous pilot study, reported in [63], tested how the available approaches modelled the uncertainty and vagueness in the context of execution tracing quality. The findings of the pilot study [63] were used in the current research; therefore, an adaptive Takagi-Sugeno-Kang fuzzy system with overlapping, Gaussian membership functions for the inputs has been selected for the model construction.

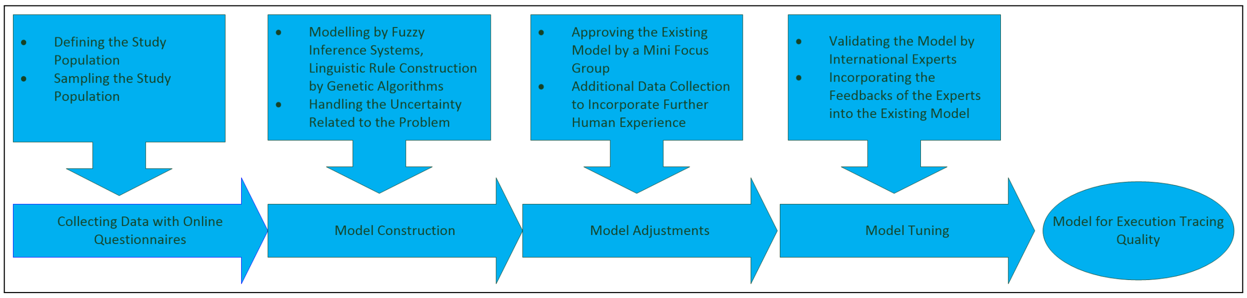
Defining the Study Population
Sampling the Study Population
Modelling by Fuzzy Inference Systems, Linguistic Rule Construction by Genetic Algorithms
Handling the Uncertainty Related to the Problem
Approving the Existing Model by a Mini Focus Group
Additional Data Collection to Incorporate Further Human Experience
Validating the Model by International Experts
Incorporating the Feedbacks of the Experts into the Existing Model
Collecting Data with Online Questionnaires
Model Construction
Model Adjustments
Model Tuning
Model for Execution Tracing Quality

Figure 5.1: Process of the Quantitative Research Conducted, Source: [68]

5.3.1 Data Collection

The study population was defined as an approachable and geographically localised group of software professionals in Hungary including software developers, software maintainers, testers and system administrators, from software companies of the competitive sector, the main activity of which belongs to the IT service delivery domain (TEAOR ID 620x at the Hungarian Central Statistical Office), and with a minimum employee count 1.000. Many companies may have the TEAOR ID 620x or its sub-IDs in addition to a different main TEAOR ID; however, only those ones whose main activity belongs to the TEAOR ID 620x or to one of its sub-IDs were included in the study population. Such software houses are usually international software companies with coding guidelines and company culture that overarches countries including premises also in Hungary. For this reason, the geographical localisation does not impose an impediment for the generalisability of the study. However, extra measures were taken, as described at the validation stage, to mitigate this risk.

The size of the study population is estimated to be $n = 12.107$[10] full-time employees. Multistage random sampling [60, 171] was applied to achieve the desired target, i.e. the companies of the study population were sampled and then the employees of the selected companies. Random sampling possesses several advantages over stratified sampling [60] with the major advantage in the research context being that the different strata in the study population do not need to be known in advance to achieve a trustworthy sample.

Information and consent letters were sent to the management of the chosen companies, and in the case of approval, employees were also selected. Before their involvement in the study, each selected employee was informed about the research aims and objectives as well as the research requirement of participation: their informed and voluntary consent.

The data were collected through an online questionnaire [67], which was tested by two software professionals before sending it out. Furthermore, ω coefficient and Cronbach's α[11] were computed for the collected data. The online questionnaire [67] collected the input variable values for (1) accuracy, (2) legibility, (3) design and implementation, (4) security, and the corresponding values for the output variable, i.e. for execution tracing quality to produce the data for supervised machine learning. The online survey contains three parts.

Part 1 of the questionnaire introduced eight use cases with textual description of each use case to provide background information and sample log data. The demonstrated use cases showed different execution tracing mechanisms and trace outputs ranging from poor quality to good quality, where the respondents had to assign

[10]Data originate from August 2017.

[11]Both ω coefficient and Cronbach's α are measurement reliability indicators whose values the closer are to 1 the more reliable measurement is indicated [56].

a score value in the range of $[0;100]$ to each use case for the inputs: (1) accuracy, (2) legibility, (3) design and implementation, and (4) security; and for the output: execution tracing quality. Necessary variable definitions were provided for the respondents. Altogether 40 questions were contained in this part of the survey.

Part 2 of the questionnaire collected data about the quality of logging in two real projects to which the respondents were assigned in the past. In addition to the input variables and the output variable given above, the software professionals had to answer a question about the type of the application: (1) server application, (2) desktop application, (3) web UI, (4) mobile application, or (5) embedded application. Altogether 12 questions were contained in this part. No other specifics of the given projects were collected.

Part 3 of the questionnaire introduced extreme input variable value combinations for (1) accuracy, (2) legibility, (3) design and implementation, and (4) security, which might not be frequent in real-life but which need also to be considered for the model construction. The respondents had to assign an output variable value, i.e. execution tracing quality, to each, listed combination of the inputs. This part of the survey contained 23 questions altogether.

Two further questions were also included in the questionnaire about the professional background and about the opinion of the respondents whether the type of the application, (1) server application, (2) desktop application, (3) web UI, (4) mobile application, or (5) embedded application, has an influence on the responses to the questions in the survey.

5.3.2 Data Processing, Exploratory Data Analysis and Modelling

After concluding the data collection stage, the data were cleaned, i.e. checked for valid responses, and the distribution of the data was analysed, outliers were removed, and the confidence intervals computed. The variable with the worst, i.e. largest, confidence interval was considered; moreover, with regard to the confidence interval and to the statistical reliability of the sample, the model validation was designed. The collected data were reshaped to possess the format necessary for machine learning and randomly split up into a training and a checking set. The training set was used to fit the model to the data, while the checking set was used to test the training process and to avoid overfitting. Constructing the quality model was accomplished by an adaptive fuzzy system, in which genetic algorithms provided the learning capability [176]. The performance of the model constructed was verified by a different machine learning approach, ANFIS [116, 117].

5.3.3 Model Adjustment and Pre-Validation

The created fuzzy systems with different number of linguistic rules were reviewed by a mini focus group containing three software professionals with many years of experience in industrial and academic settings. The optimal model was selected, adjusted and checked for validity.

5.3.4 Model Validation

The data for model validation were collected using a second online questionnaire [67]. The responses stemmed from selected international experts located in Austria, Germany, and Hungary (countries listed in alphabetic order). The questionnaire contained the linguistic variables, and the model as a set of linguistic rules; moreover, the impacts of the model's pairwise inputs in the output were also depicted on charts. An online meeting, if deemed necessary, was offered to the experts for checking the model, its fit and the model's responses to the changes in the inputs. The model was tuned according to the responses of the experts. This way the risk of geographical localisation in the course of the data collection was mitigated.

5.4 Analysis of the Collected Data

5.4.1 Data Collection Implemented

The employee count of the software companies that agreed to participate in the study amounted to approx. 6.000 people. Thus, the first stage of sampling covered nearly 50% of the study population. The goal was to achieve a completely random sample from the companies involved, possibly selecting individuals randomly from the register and then contacting them; however, this was considered unfeasible later, in terms of effort involved. Thus, all the employees of selected areas at the selected companies received an email notification with information and with the link to the online questionnaire to fill in. Overall, 41 software professionals filled in the survey, which had three parts with 77 questions in total.

The reliability indicators, ω coefficient and Cronbach's α were computed for each separate part of the questionnaire after removing the rows with NA values, and removing the outliers as discussed in section 5.4.2. Both ω coefficient and its alternative Cronbach's α are widespread in psychometric and economic research to estimate the reliability of measurements [56, 236]. α values 0.6 and above are acceptable, values above 0.8 are very good but values above 0.95 might indicate the measurement of more than one independent variable [236].

Table 5.1: Reliability Indicators of the Online Questionnaire [67], Source: [68]

	Use Cases	Real Projects	Extreme Input Values
α	0.98	0.92	0.83
ω_{total}	0.98	0.94	0.87

The extreme high reliability indicator values for the use cases might have been caused by the repeating questions, which are valid in the context of the present research as each use case represents a different setting.

5.4.2 Data Processing and Exploratory Data Analysis

The responses to the questions were depicted on box plots to illustrate their dispersion and the outliers. Questions in part 2 of the survey constituted an exception from the outlier detection as each respondent could think of a different project in the past, which describe valid input-output data pairs from the point of view of the data collection, but they do not necessarily come from the same probability distribution.

The box plots depict the Q1 and Q3 quartiles. The median is shown by the middle bold line in each box while the outliers are highlighted with the dots over the whiskers. The distance of the whiskers (W) from the Q1 and

Q3 quartiles is computed by 1.5 times the interquartile range. If the exact W value computed is not present in the data set, then the whiskers are placed to the closest value towards the median.

The chart in figure 5.2 illustrates the dispersion of the assigned quality scores, i.e. the output values, of the defined use cases. An upwards going trend in the median value can be observed, which mirrors the description of the use cases and the quality of the log snippets the respondents rated. The box plots of the input variable values are depicted in Appendix C.1 in charts C.1, C.2, C.3, and C.4. In addition, figure 5.3 highlights the Q1, Q3 and median values of the quality scores assigned to the combination of the extreme input values.

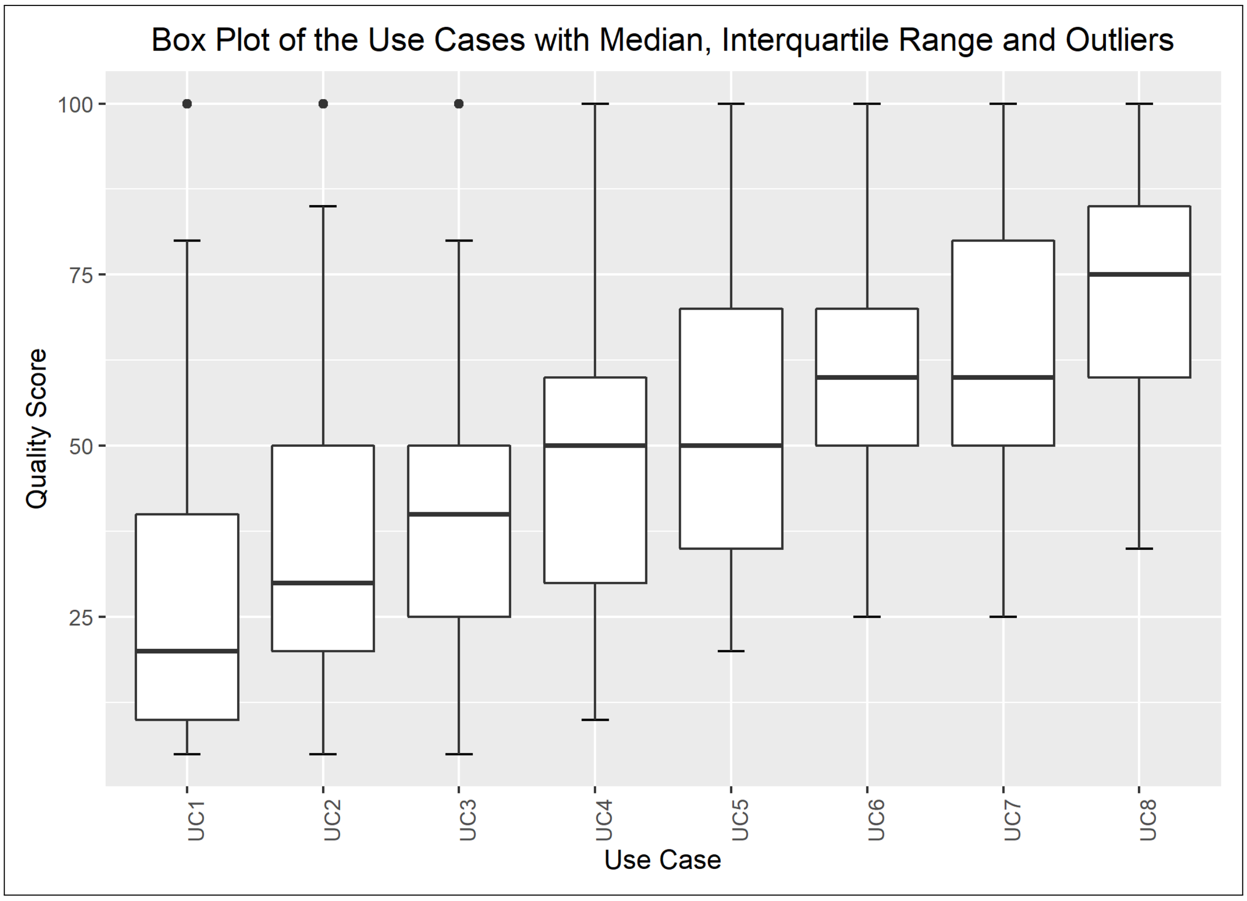

Figure 5.2: Quality Scores Assigned to the Use Cases, with Q1, Q3 and Median Values, Source: [68]

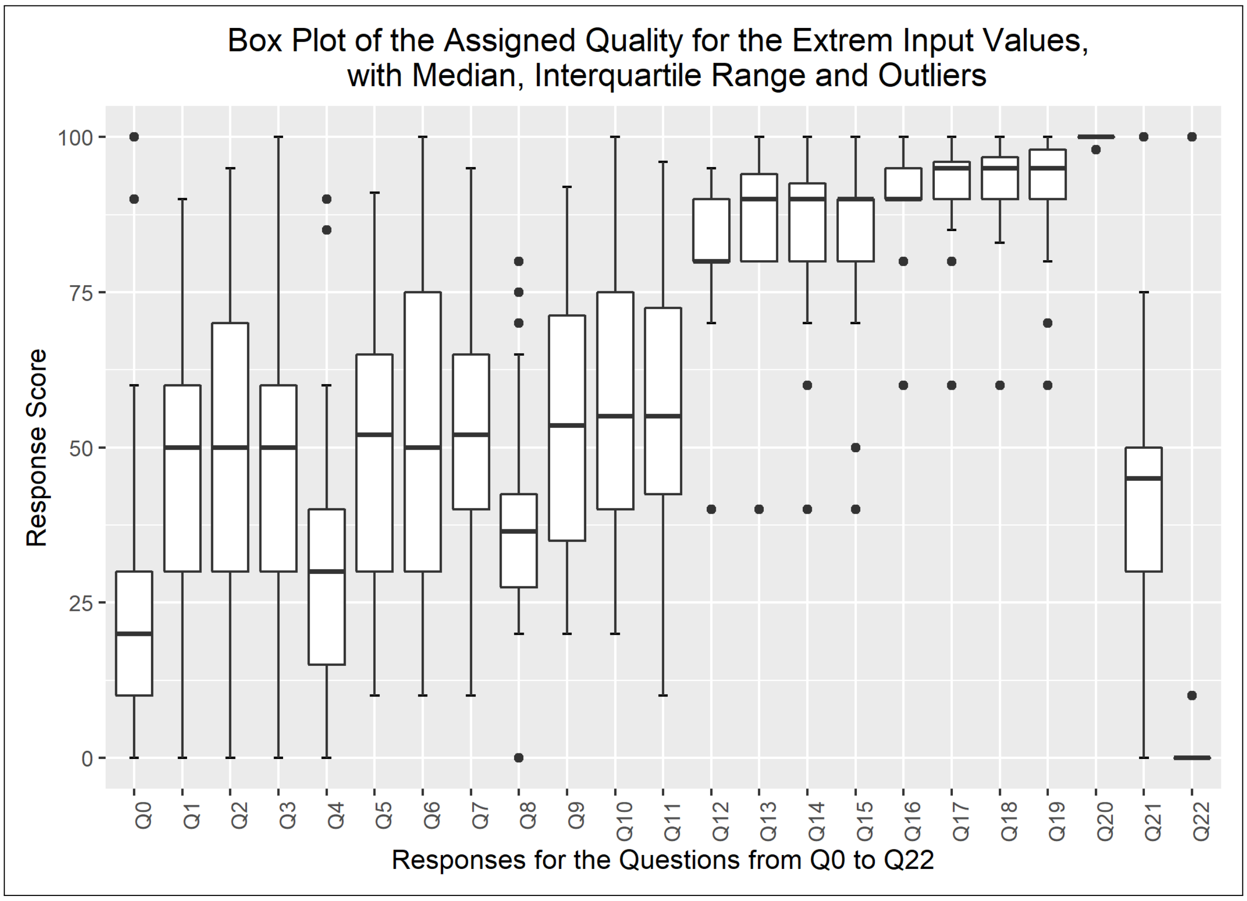

Figure 5.3: Quality Scores Assigned to the Extreme Input Values, with Q1, Q3 and Median Values, Source: [68]

The data show asymmetric dispersion and skewness regarding the majority of the collected variables. Nevertheless, quantile-quantile plots were computed to check how the collected data approximate the theoretical normal distribution. The charts, available in Appendix C.1.2, show the 0.95% confidence intervals and the Sharpio-Wilk Normality Tests. Before computing the quantile-quantile plots, the outliers and the rows with NA values were removed. The use case data are illustrated in figures C.5, C.6, C.7, C.8, C.9, C.10, C.11, C.12. The following variables do not approximate the theoretical normal distribution $p_{Sharpio-Wilk} < 0.05$: Use Case 1: all variables; Use Case 2: Accuracy, Security; Use Case 3: Accuracy; Use Case 4: all variables approximate the theoretical normal distribution; Use Case 5: Design and Implementation; Use Case 6: Design and Implementation, Quality; Use Case 7: Accuracy, Quality; Use Case 8: Accuracy, Design and Implementation. The collected data about real projects are not normally distributed as shown in figure C.13 but they may represent different projects. The following variables do not approximate the theoretical normal distribution $p_{Sharpio-Wilk} < 0.05$ regarding the data assigned to the extreme input variable values: Q2, Q4, Q8, Q10, Q12, Q13, Q14, Q15, Q16, Q17, Q18, Q19, Q21 as illustrated in charts C.14, C.15, C.16, C.17. The variables Q20 and Q22 were excluded from the normality test as they only possessed identical values.

As many of the variables collected were not normally distributed, the p=90% confidence interval was computed by the non-parametric Wilcoxon Test [72]. To highlight the uncertainty related to all questions in a meaningful way, the range rather than the lower and upper values of the confidence intervals are reported below.

The ranges of confidence intervals for the input and output variable values without the outliers, related to the described eight use cases, possess the following characteristics: min value: 9.99, max value: 24.99, median:

12.49. The 24.99 range value emerged for the input variable Security at use case 1 and use case 2 while the ranges of the other variables were close to the median.

The assigned execution tracing quality, i.e. the output variable, to the extreme input variable value combinations showed lower dispersion than the responses to the use cases. The most definite answers arrived when all the four input variables had the maximum value 100 at question Q20 and when they all had the value 0 at question Q22. In these cases, 100 was assigned at Q20 and 0 was assigned at Q22 by all respondents not counting the outliers. The ranges of the confidence intervals show the following characteristics after removing the outliers: min value: 1, max value: 15, median: 10.

In conclusion, the collected data after removing the outliers show 12.5 median value uncertainty related to the constructed use cases and 10 median value uncertainty related to the extreme input variable value combinations on a scale $[0;100]$ when p=90%. Thus, the uncertainty on the scale $[0;100]$ at this specified probability value lies between 10 % and 12.5 % with regard to the medians of the confidence intervals. An approximately 10% uncertainty while rating software quality on a ratio scale is an acceptable value; however, the small sample size (N=41) makes a stronger validation necessary as usual.

5.4.3 Modelling by Means of Fuzzy Logic with Rule Extraction by Genetic Algorithms

The data collection and exploratory data analysis resulted in a tabular representation of the data where the questions of the survey form the columns; however, another data format is necessary for machine learning containing pairs of inputs and output. To achieve this, the data were transformed into a matrix representation where the columns represent (a) the four input variables: (1) accuracy, (2) legibility, (3) design and implementation, and (4) security; and (b) the corresponding output: execution tracing quality. The rows of the matrix represent input and output pairs. Where an outlier was marked in the rows of the matrix, the complete row was removed. After reshaping the data this way and removing the outliers, 1185 pairs of the inputs and the corresponding output were obtained, which was considered suitable for machine learning. The data set was randomly split up in 70% and 30% parts for training and checking as described in section 5.3.

Modelling was performed in Matlab R2021a environment, where a fuzzy inference system was created with four inputs and one output, according to the data set above. The input membership functions and their partitions are shown in figure 5.4. The output membership functions were defined as five constants, i.e. zero-order membership functions were used in the output at the values: $\{0, 25, 50, 75, 100\}$ for the singleton fuzzy sets $\{$very poor, poor, medium, good, very good$\}$. The fuzzy inference system was constructed with the inference type Takagi-Sugeno-Kang with no initial linguistic rules.

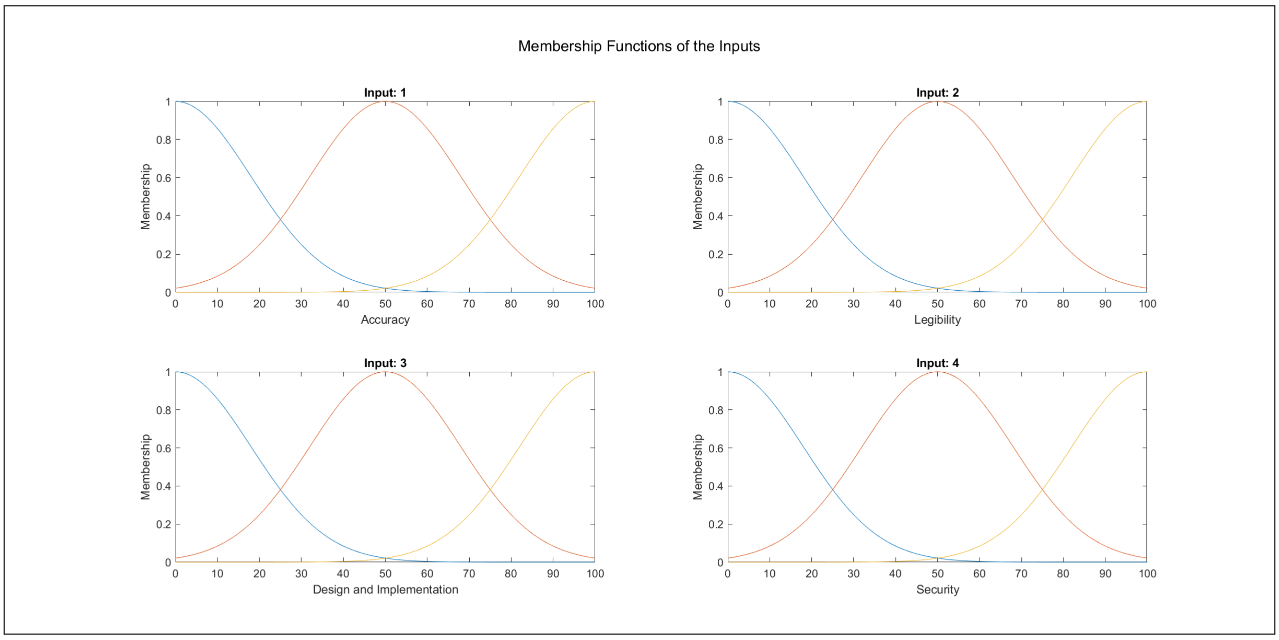

Figure 5.4: Membership Functions of the Inputs, Source: [68]

In the scope of machine learning, the system was adapting to minimise the root mean squared error on the input-output data pairs, while the learning was carried out with genetic algorithms [51], and the number of possible linguistic rules were limited to {1, 3, 5, 8, 12, 16, 20, 40, 80, 160} per runs. Genetic algorithms have a non-deterministic nature, i.e. more runs of the same algorithm with the same parameters in the same problem domain may achieve different but similar results. Therefore, each run with the same parameters and with the same upper bound settings for the linguistic rules were executed five times and the results were recorded. In addition, the whole run-suite was executed six times.

This way, six times five models were created for each rule limit in the set {1, 3, 5, 8, 12, 16, 20, 40, 80, 160}. In addition, the bests of the five models were collected per each rule limit in each of the six run suite. The prediction of the models with the different number of maximal rules were analysed with regard to the mean absolute error (MAE), root mean squared error (RMSE), minimal deviation and maximal deviation from the desired target in the course of each run. The runs with the best performance, which come from run suite 2, are depicted in figure 5.5 and the corresponding data are illustrated in table 5.2 with regard to the training data set; moreover, the same runs are also shown in figure 5.6 with regard to the checking data set and the corresponding data are highlighted in table 5.3. The RMSE indicators on the training data and on the checking data are shown in Appendix C.2 in figure C.18 and in figure C.19 with the corresponding data in table C.1 and C.2. The machine learning process was ceased when the new generations produced by the genetic algorithm reached a saturation point with regard to the mean RMSE value of the individuals in the population.

The charts with the data the created models predict as outputs and the original, collected outputs are shown for each rule limit in the set {1, 3, 5, 8, 12, 16, 20, 40, 80, 160} in Appendix C.3 with the best maximum-error and the best RMSE-error model candidates for the checking data in figures C.20, C.21, C.24, C.25, C.28, C.29, C.32, C.33, C.36, C.37, C.40, C.41, C.44, C.45, C.48, C.49, C.52, C.53, C.56, C.57, and for the corresponding training data in figures C.22, C.23, C.26, C.27, C.30, C.31, C.34, C.35, C.38, C.39, C.42, C.43, C.46, C.47, C.50, C.51, C.54, C.55, C.58, C.59. The original data have been sorted to ensure that the deviations from the desired targets are easy to notice. Each rule set extracted designates a distinct model.

When the maximal number of linguistic rules allowed were 80 or 160 rules, then the number of rules learnt by the systems was usually below 80 linguistic rules. The data in figures 5.5, and 5.6, furthermore, the data in Appendix C.2 in figures C.18, and C.19 indicate that increasing the number of linguistic rules results in a better accuracy as far as the number of rules approximate 12. Exceeding this value does not imply better performance. Consequently, the extracted rule set with 12 rules became the target of further processing. The extracted rule sets of the selected run suite 2 are listed in Appendix C.3 with their evaluation charts on the checking data set.

Table 5.2: Best Maximal Errors on the Training Data, Source: [68]

	Limit: 1 Rule	Limit: 3 Rules	Limit: 5 Rules	Limit: 8 Rules	Limit: 12 Rules	Limit: 16 Rules	Limit: 20 Rules	Limit: 40 Rules	Limit: 80 Rules	Limit: 160 Rules
RMSE	30.59485	19.56332	18.47743	16.21253	17.859	15.67844	15.83771	15.45328	15.69313	18.56351
MAE	26.16265	15.88731	15.17334	11.79144	14.31864	11.31696	11.97357	11.01256	11.41314	15.22481
Min. Error	0	7.11E-15	2.12E-05	0.000731	2.43E-06	0.025736	0.022271	0.004223	0.039392	0.153897
Max. Error	50	74.97749	50.51683	55.28113	50.51682	50.522	51.50747	50.01046	49.81493	52.96985

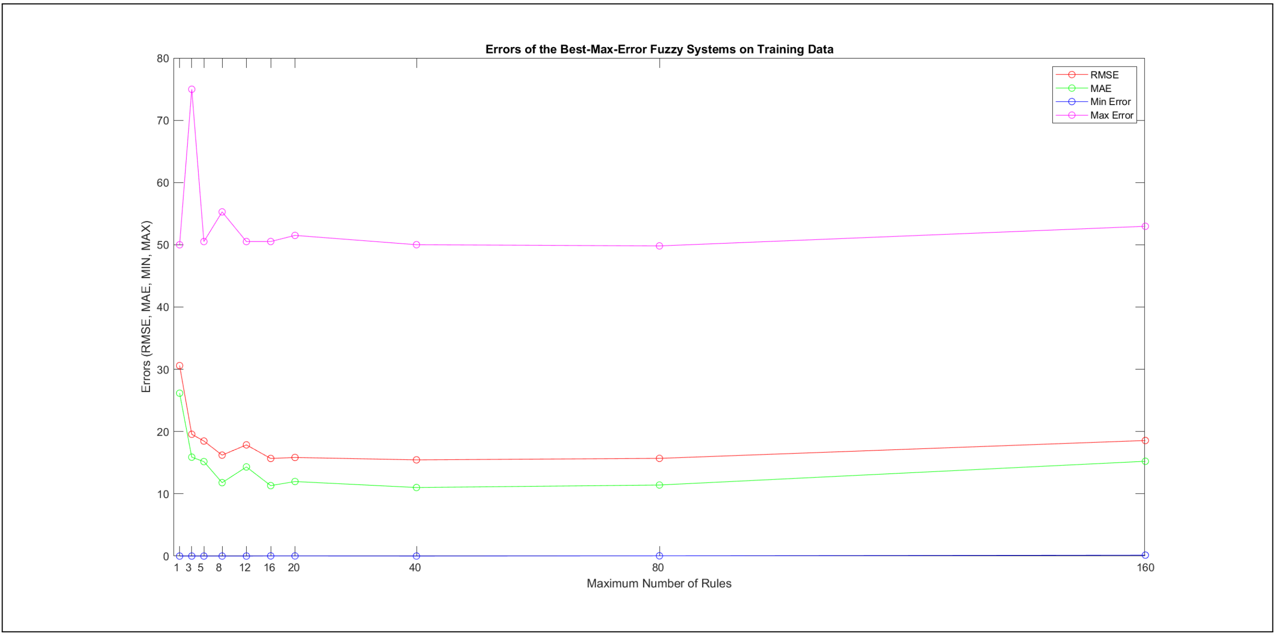

Figure 5.5: Best Maximal Errors on the Training Data, Source: [68]

Table 5.3: Best Maximal Errors on the Checking Data, Source: [68]

	Limit: 1 Rule	Limit: 3 Rules	Limit: 5 Rules	Limit: 8 Rules	Limit: 12 Rules	Limit: 16 Rules	Limit: 20 Rules	Limit: 40 Rules	Limit: 80 Rules	Limit: 160 Rules
RMSE	30.23458	19.79506	17.11367	16.42312	15.58643	15.5542	15.39485	15.32827	15.32682	16.35666
MAE	25.92113	15.74546	12.96622	11.99584	11.08251	11.26821	11.27761	11.13463	10.90988	12.2782
Min. Error	0	7.11E-15	9.93E-06	0.010229	0.014746	0.000246	0.011135	0.000583	0.007301	0.01093
Max. Error	50	74.97749	50.00047	50.5166	51.03274	50.77524	51.05456	51.48755	50.05721	50.25765

5.4.4 Model Verification by Means of an Adaptive Network-based Fuzzy Inference System

As introduced in section 5.4.3, the quality model for execution tracing was prepared using machine learning through genetic algorithms which adapted a fuzzy inference system to the collected data set, i.e. the model was trained on the training data, tested on both the training and on the checking data, while the input and output membership functions were kept unchanged and the linguistic rules were being identified. To test the model created at the end of the machine learning stage, a different machine learning approach was applied and the error indicators were compared to the ones reported with genetic learning.

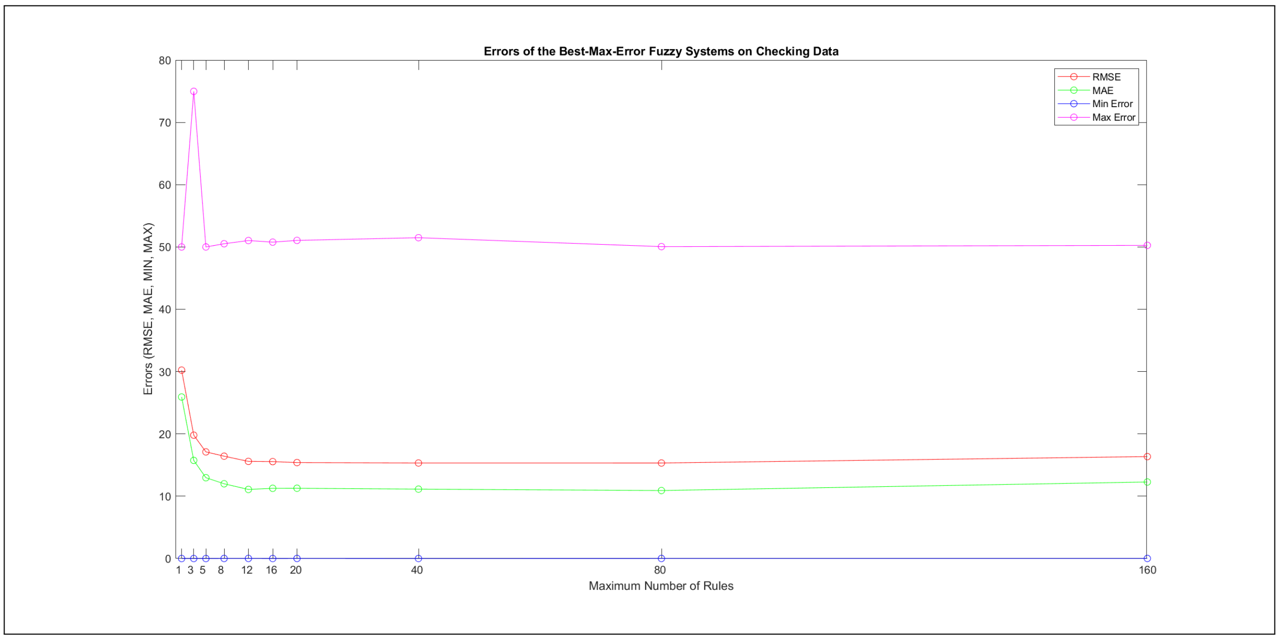

Figure 5.6: Best Maximal Errors on the Checking Data, Source: [68]

A new model was created by ANFIS [116, 117], i.e. the linguistic rules of the fuzzy inference system were specified with grid-partitioning at the start and each linguistic rule was associated with a distinct output membership function, while the same initial input membership functions were assigned as in the case of the previous model trained by genetic algorithms. ANFIS adjusted the parameters of the input and output membership functions. Grid partitioning, selected due to the homogeneous coverage of the whole problem domain, resulted in as many linguistic rules as the possible variations of the input partitions are, i.e. four inputs with three partitions ($3^4 = 81$). Consequently, 81 output membership functions were required. Thus, the efficient human interpretation of the rule set to understand the problem domain, after training the model, is not feasible in contrast to the model created in section 5.4.3.

Furthermore, ANFIS training was implemented in two different ways: with (1) backpropagation and (2) hybrid approaches. For each learning approach, a separate model has been trained with one of the initial step sizes from the set: $\{0.1, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5\}$. The best-performing models with regard to the maximal errors and RMSE errors were selected in the course of 100 learning epochs. Table 5.4 presents the best error indicators with regard to the initial step sizes. The adaptation of the initial step sizes during the learning process can be observed in figures C.77, C.79, C.81 and C.83.

The change of the RMSE error, for the best four models in Table 5.4, as a function of learning epochs on the training and on the checking data are depicted in Appendix C.6 in figures C.76, C.78, C.80 and C.82. In addition, the evaluation charts of the four best models, in Table 5.4, with the ANFIS approach are illustrated in Appendix C.6 in figures C.72, C.73, C.74, C.75.

The evaluation charts and the best error indicators in Table 5.4 show only minor differences in comparison to those achieved with genetic learning in section 5.4.3. Thus, the results of section 5.4.3 are verified.

Table 5.4: Error Indicators of the Best-Performing Models Trained by ANFIS Approach, Source: [68]

	Training: Backpropagation		Training: Hybrid	
	Best Max. Error	Best RMSE Error	Best Max. Error	Best RMSE Error
RMSE	15.04308982	14.92059348	16.35542215	16.31984729
MAE	10.91966752	10.68467212	10.9482579	10.66809624
Min. Error	0.046226241	0.042525692	0.049740427	0.012379938
Max. Error	50.61656014	51.15983193	66.11773115	96.19663932
Initial Step Size	2.5	3	3.5	5

5.5 Model Tuning and Validation

Model validation was performed in three distinct steps: (1) adjustments and pre-validation in a mini focus group, (2) validation by international experts and tuning the model based on their feedbacks, and (3) face validity carried out by the author. All three stages of the validation are reported below.

5.5.1 Adjustments and Pre-Validation

The rule set with the 12-upper-bound value of the linguistic rules became, for the above reasons, the starting point of the investigation where human experience was directly incorporated into the results gained by computational intelligence. A mini focus group with three software professionals including a software maintainer, a software tester and a system administrator, whose job roles include intensive log analysis, with many years of experience in industry and in academia, investigated the established rules, their plausibility and the evaluation charts C.36, C.37. In addition, the pairwise effect of the inputs on the output was also analysed as shown in appendix C.4 in figures C.60, C.61, C.62, C.63, C.64, and C.65.

The aim of the discussion in the mini focus group was to incorporate direct human experiences in the rule base while investigating the existing linguistic rules by (1) leaving the correct and plausible linguistic rules in the rule set, (2) removing or changing the incorrect ones and (3) adding missing ones. Thus, the adjustments also implement a pre-validation by the three participants of the mini focus group.

Furthermore, the adjustments did not attempt to improve the error indicators: RMSE, MAE, maximal error, and minimal error on the training and checking data sets because the genetic algorithm used had already approximated the optimal fit to the data sets with the extracted rules.

The following problems were identified: The initial evaluation charts in figures C.36, C.37, C.38, and C.39 show an accumulation of the predicted values in the middle range of the output variable. Moreover, (P0) no linguistic rules existed, which would have explicitly defined the output value {good} from the possible outcome set {very good, good, medium, poor, very poor}. In addition, (P1) if all input values lie in the domain {good}, then the output value should lie in the domain {very good}; (P2) the input variables legibility and security, while they decrease in the {poor} - {poor} domain, they should not contribute an increase to the output in figure C.64; (P3) along all values of the input variable security while the input variable accuracy increases in the {good} domain, the output in figure C.64 should not fall back; (P4) while moving from the {poor} and {medium} values of the input variable design and implementation towards the domain {good} and increasing the input variable accuracy from {medium} to the {good} domain, then the output should not decrease in figure C.61; (P5) while increasing the input variable accuracy along all values of the input variable legibility, the output in figure C.60 in the {good} domain of accuracy should not produce a decreasing effect; and (P6) while increasing the input variable legibility from the {medium} domain and keeping the variable accuracy in the {poor} domain, the output in figure C.60 should not decrease.

The rules R1, R2, R4, R6, R7, R10 were not modified. The rules R3, R5, R9, R11, R12 were changed as per the feedback from the mini focus group, while new rules R13 to R19 were added to resolve the issues P0 to P6 listed above.

R13. resolves P1, i.e. if all input values possess {good} values, then the output value should belong to the domain {very good};

R14. resolves P2 as illustrated in figure C.70

R15. resolves P3 and P4 as shown in figures C.70 and C.67

R16.-17. resolve P5 and P6 as depicted in figure C.66

R18.-19. resolves P0, i.e. explicit linguistic rules were created for the output value {good}

While identifying the necessary changes and applying the adjustments, the fuzzy model was tested with the combination of the vector [0, 50, 100], i.e. each input received a value from the vector and the output of the model was analysed. Furthermore, the response of the model was also investigated while three of the four inputs of the model were fixed and one input underwent a continues increase and decrease. The final set of linguistic rules sent for validation is summarised below:

R1. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)

R2. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is good) and (Security is poor), then (Quality is medium)

R3. If (Accuracy is poor) and (Legibility is poor) and (Security is medium), then (Quality is very poor)

R4. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is medium)

R5. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is poor), then (Quality is medium)

R6. If (Accuracy is poor) and (Legibility is good) and (DesignAndImplementation is good) and (Security is good), then (Quality is poor)

R7. If (Legibility is poor) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is poor)

R8. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is very poor)

R9. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is good) and (Security is medium), then (Quality is medium)

R10. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is poor)

R11. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is poor)

R12. If (Accuracy is medium) and (Legibility is poor) and (Security is medium), then (Quality is medium)

R13. If (Accuracy is good) and (Legibility is good) and (DesignAndImplementation is good) and (Security is good), then (Quality is very good)

R14. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is very poor)

R15. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is not good), then (Quality is medium)

R16. If (Accuracy is good) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is medium)

R17. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is poor)

R18. If (Accuracy is good) and (Legibility is good) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is good)

R19. If (Accuracy is good) and (Legibility is good) and (DesignAndImplementation is good) and (Security is medium), then (Quality is good)

5.5.2 Validation by International Experts

The final step of modelling was done within the validation stage, which was implemented (1) by sending out a second questionnaire [67] to selected international experts located in Austria, Germany, Hungary, and (2) considering their feedbacks the created model was tuned.

The online questionnaire contained the input variable definitions, the input membership function, their partitions, a brief description of the technique used for modelling, and the linguistic rules adjusted by the mini focus group as introduced in section 5.5.1. Furthermore, the survey ensured the possibility for giving a textual feedback. It also contained the description of the effects of the pairwise inputs on the output as shown in Appendix C.5 in figures C.66, C.67, C.68, C.69, C.70, and C.71. Online model demonstration was offered to the experts, which was accepted by one expert.

Based on the feedback from the experts rule R19 was changed as shown below:

R19. If (Accuracy is good) and (Legibility is good) and (DesignAndImplementation is good) and (Security is medium), then (Quality is very good)

The experts did not propose or requested any further change in the constructed model. In addition, two insights from two different experts is summarised to highlight the nature of the quality property security and its possible impacts on execution tracing quality: (1) increasing security has a negative impact on execution tracing quality as leaving out sensitive or GDPR-related data, including login names, deteriorates analysability; (2) security requirements are domain-specific, security is a must for medical software, and without a certain base-level quality for this property the whole application is inadequate. These opinions illustrate that, even with regard to one individual quality property, contrasting requirements may exist in connection with the output and a possible optimum needs to be found. This also supports the use of fuzzy logic in the problem domain which can adequately describe contradicting information.

Table 5.5: Error Indicators of the Created Model After the Different Modelling Stages, Source: [68]

	Base 12-rule Model		Model After Adjustments		Model After Validation	
Error Type	Checking Data	Training Data	Checking Data	Training Data	Checking Data	Training Data
RMSE	15.5864	15.5653	19.5772	19.9272	23.5540	23.3813
MAE	11.0825	11.3225	14.1003	14.0510	16.3166	16.1725
Min. Error	0.0147	0.0033	0.0055	0.0015	0.0055	0.0015
Max. Error	51.0327	55.7013	73.9973	76.0869	97.4981	97.4981

Table 5.5 provides the different errors of the created model after each modelling stage. The error indicators were measured on the training and checking data sets and rounded to four decimal places. The measurements indicate an error increase as per the adjustments and tuning of the model, which can be explained: (1) an approximated optimal fit of the linguistic rules was obtained by the machine learning, (2) new information was added to the model from the mini focus group, and (3) the opinions of experts were considered and incorporated into the created model to tune it.

5.5.3 Validation by the Author

The author investigated the outcome of the modelling stage after incorporating the feedbacks of the international experts. The formalised linguistic rules were found to be reasonable and plausible; moreover, the pairwise impacts of the inputs on the output were also analysed and illustrated in figures 5.7, 5.8, 5.9, 5.10, 5.11, and 5.12.

Accuracy - Legibility in figure 5.7: When accuracy lies in the {poor} domain, the increase of legibility produces a slight increase which quickly saturates. The behaviour looks similar in the {medium} range of accuracy but the saturation occurs at a higher value, while the increase of legibility in the range of {good} accuracy values produces a strong increase in the output. In the {good} ranges of legibility, the described relationship contributes an increasing effect to the output as accuracy increases. On the other hand, in the {poor} and {medium} ranges of legibility, the increase of accuracy produces a far less increase in the output than in the {good} range of legibility; moreover, the increasing effect saturates at a lower value.

The chart demonstrates that accuracy has more impact on the output in the {medium} - {good} accuracy domain if legibility lies in the {good}domain.

Accuracy - Design and Implementation in figure 5.8: In the {poor} domain of design and implementation, the increase of accuracy can only produce a low increase with a quick saturation in the output, while this saturation occurs in the {medium} domain of design and implementation at a higher value. Furthermore, in the {good} range of design and implementation, the increase of accuracy contributes an increase to the output, with a horizontal segment in the {medium} domain of accuracy. When accuracy possesses {poor} ranges and design and implementation increases, the variable combination contributes a slight

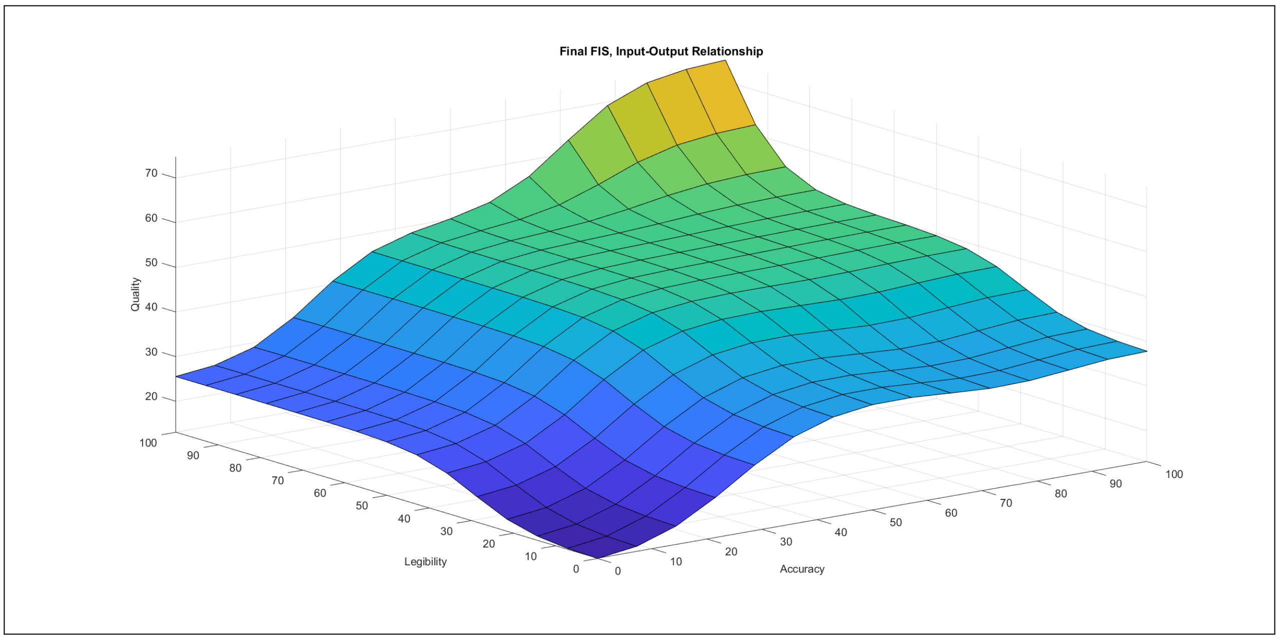

Figure 5.7: Effect of the Input Accuracy and Legibility on Execution Tracing Quality After Tuning the Rules, Source: [68]

increase to the output, with a horizontal segment at a low value while the contribution increases in the {good} domain of design and implementation. The {medium} domain accuracy values with increasing design and implementation values contribute an increase with a saturation to the output, while the same increase in the {good} ranges of accuracy produce an increase with a horizontal segment at {medium} design and implementation values.

The chart demonstrates that both variable have approximately the same effect on the output.

Accuracy - Security in figure 5.9: In the {poor} range of accuracy, the increase of security can only contribute a small increase with a very quick saturation to the output in the relation of these two variables. In the {medium} range of accuracy this contribution is bigger but the saturation is quick and occurs at a higher value than in the {poor} domain. At {good} values of accuracy, security only has an influence on the output in the {good} domain of security, where it contributes a slight rise. When security lies in the {poor} range, the increase of accuracy produces an increasing effect to the output with a late saturation in the {good} accuracy domain. At {medium} security values, the increase of accuracy produces the same increasing effect with a quicker saturation at the same value as in the {poor} domain of security. In the {good} security domain, the increase of accuracy produces a similar effect to the {medium} security domain but it also contributes an increase to the output in the range of {good} accuracy values.

The chart illustrates that the effect of accuracy is far stronger than the effect of security, which is similar to the case with the variables security and design and implementation.

Legibility - Design and Implementation in figure 5.10: In the {poor} range of legibility, the increase of design and implementation causes and increasing effect in the output with a short, nearly horizontal segment in the {medium} design and implementation domain. In the {medium} domain of legibility, rising design and implementation contributes an increase to the output, which achieves its maximum at a higher value and at lower design and implementation inputs than in the {poor} legibility domain. Increasing design and implementation in the {good} legibility domain contributes a strong rise to the output. In the {poor} design and implementation domain, the increase of legibility contributes a small rise to the

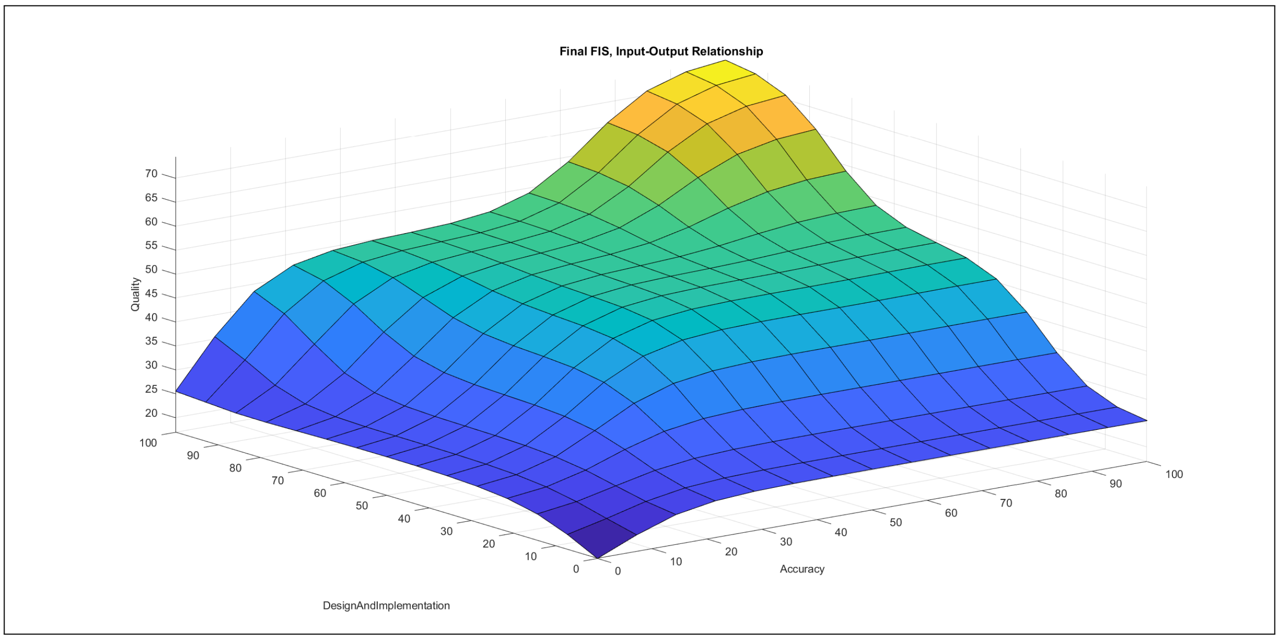

Figure 5.8: Effect of the Input Accuracy and Design and Implementation on Execution Tracing Quality After Tuning the Rules, Source: [68]

output with a saturation at a low value. The domain with {medium} design and implementation values produces a similar effect for increasing legibility but the starting point of the contribution to the output is a higher value and the saturation occurs also at a higher value than in the {poor} domain. The increase of legibility in the {good} range of design and implementation produces a contribution to the output only in the {good} domain of legibility, which is an increase.

The chart depicts that the effect of design and implementation is stronger than the effect of legibility.

Legibility - Security in figure 5.11: In the {poor} range of legibility, the rise of security contributes a strong increasing effect to the output in the {poor} security domain but this effect saturates at the beginning of the {medium} security domain. A slightly increasing contribution to the output can be observed in the {medium} legibility domain if security increases but the initial contribution without the changes is high. In the {good} legibility domain, the increase of security only in the {good} domain has an effect on the output, which is increasing, in the relation of these two variables. In the {poor} domain of security, the increase of legibility produces a strongly increasing contribution to the output in the {poor} legibility domain. In the {medium} security domain, the rise of legibility contributes a slight increase to the output. The contribution has a high initial value. In the {good} security domain, the contribution to the output starts from a high initial value while increasing legibility but the rise is mild with a horizontal segment in the range of {medium} legibility values.

Practically these variables take effect only in the {poor} - {poor} domain but then the effect triggered is strong.

Design and Implementation - Security in figure 5.12: In the {poor} range of design and implementation, the increase of security produces a slightly increasing effect in the output but this effect saturates at low security values. In the {medium} domain of design and implementation, the rise of security has an effect only in the {poor} security domain, which is increasing with regard to the output. In the {good} range of design and implementation, the increase of security contributes no effect to the output in the relation of these two variables. In the {poor} domain of security, the increase of design and implementation

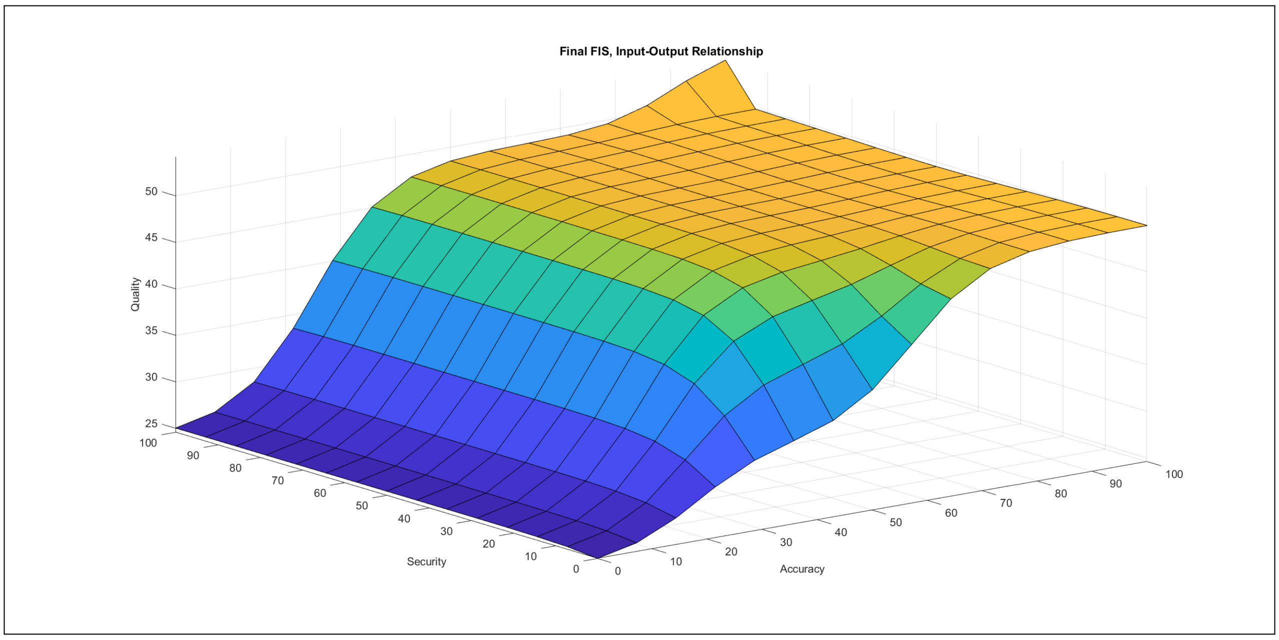

Figure 5.9: Effect of the Input Accuracy and Security on Execution Tracing Quality After Tuning the Rules, Source: [68]

produces an increasing effect in the output. This effect is stronger in the {medium} and {good} ranges of security if design and implementation rises but then the contribution to the output saturates at lower design and implementation values.

The chart is very similar to the chart of accuracy - security, i.e. the variable design and implementation shows a far stronger effect towards the output than the variable security. In the {good} range of design and implementation, the increase of security contributes no effect to the output in the relation of these two variables, which does not mean that the increase of security, in the relation to all inputs, contributes no effect to the output.

The above comparisons describe three dimensions, two inputs and one output. All input variable combinations are depicted to highlight their effects on the output. Consequently, the described effects refer to the output with regard to the presented input combinations. All the input combinations above, depicted on the three-dimensional charts and described in the listing, have a reasonable effect on the output, which is in accordance with the experience of the author.

5.5.4 Validation Aspects Considered

This section is devoted to the different validity aspects of the model construction study. In complex modelling studies, validity is a scale with different degrees that describes the distinct validity aspects of the conducted research. Internal, external, content and construct validities are investigated and documented below. Every reasonable effort was made in the context of the study to satisfy the documented validity characteristics.

Internal validity establishes that the research conducted represents the truth with regard to the phenomenon examined [197]. The conducted study is based on the (1) representativeness of the sample drawn from the defined study population, on the (2) human experiences incorporated in the rule base, and on its (3) internal consistency. The online questionnaire was tested before sending it out; moreover, the reliability indicators ω_{total}

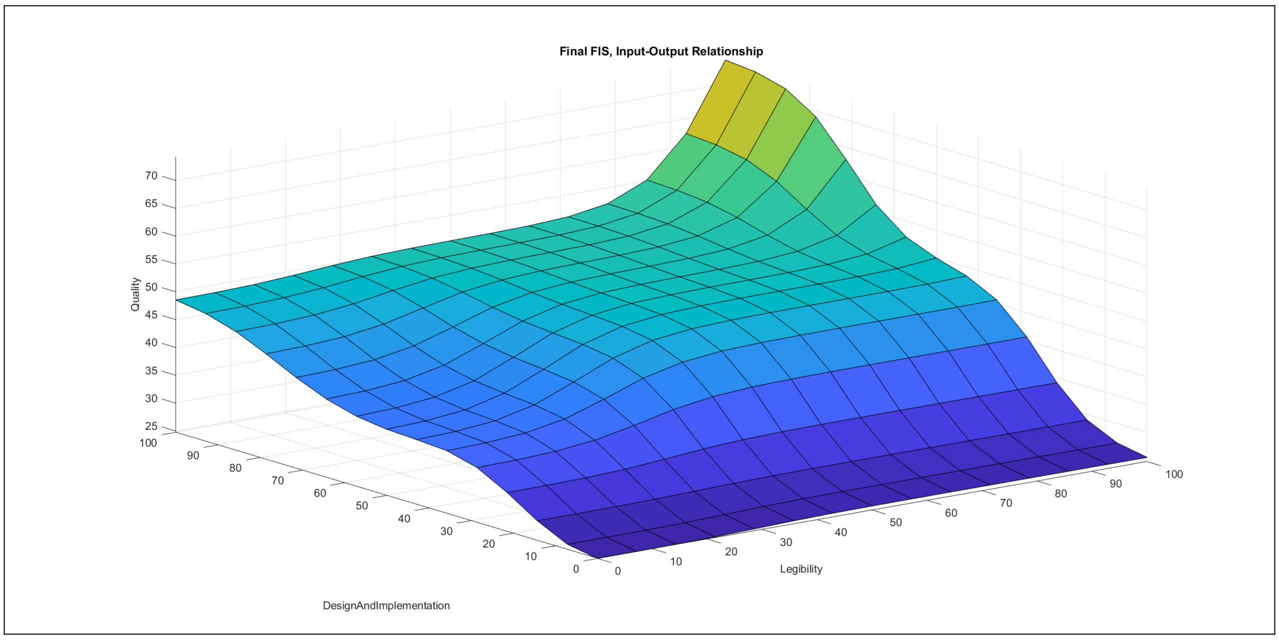

Figure 5.10: Effect of the Input Legibility and Design and Implementation on Execution Tracing Quality After Tuning the Rules, Source: [68]

and Cronbach's α were computed for the collected data and they show high-reliability values as presented in section 5.4.2. The sample size (N=41) is low; however, the median value of the ranges of the p=90% confidence intervals are acceptable with regard to the problem domain as discussed in section 5.4.2. Nevertheless, 1185 inputs-output data pairs were collected to accomplish machine learning; furthermore, additional data collection was carried out in the mini focus group while investigating the performance of the created model as described in section 5.5.1. The internal consistency of the study was achieved (a) in the scope of the machine learning by training the model on the training data set, which contained 70% of the data randomly selected from the whole data set, and checking the performance of the created models on both the training and on the checking data; (b) the linguistic rules generated in the scope of machine learning underwent adjusting in the mini focus group; (c) finally, the created model was investigated by experts and tuned based on their feedbacks as explained in section 5.5.2. In addition, machine learning was implemented in two different manners and the errors of the two approaches were compared as reported in section 5.4.4; furthermore, the non-deterministic algorithms were executed several times and the models with best performance runs were selected as described in sections 5.4.3 and 5.4.4.

External validity establishes the generalisability of the findings identified in the study [197]. This kind of validity was satisfied by including a broad range of software professionals in the study: software architects, software developers, software maintainers, software testers and system administrators. On the other hand, the study population was defined with the localisation to Hungary; however, international companies with employee counts over 1.000 were involved in the study with international coding habits and guidelines overarching geographical boundaries. In addition, the model created underwent final tuning based on the opinions of international experts located in Austria, Germany, and Hungary.

Content validity [211] in the context of the study refers to the extent to which the universe of all possible opinions of software professionals were considered. A higher sample size could have helped to better satisfy this kind of validity but the p=90% confidence intervals and the variance of the data presented in section 4 show acceptable ranges for the context of the research. Nevertheless, the error indicators in Table 5.5 increased with

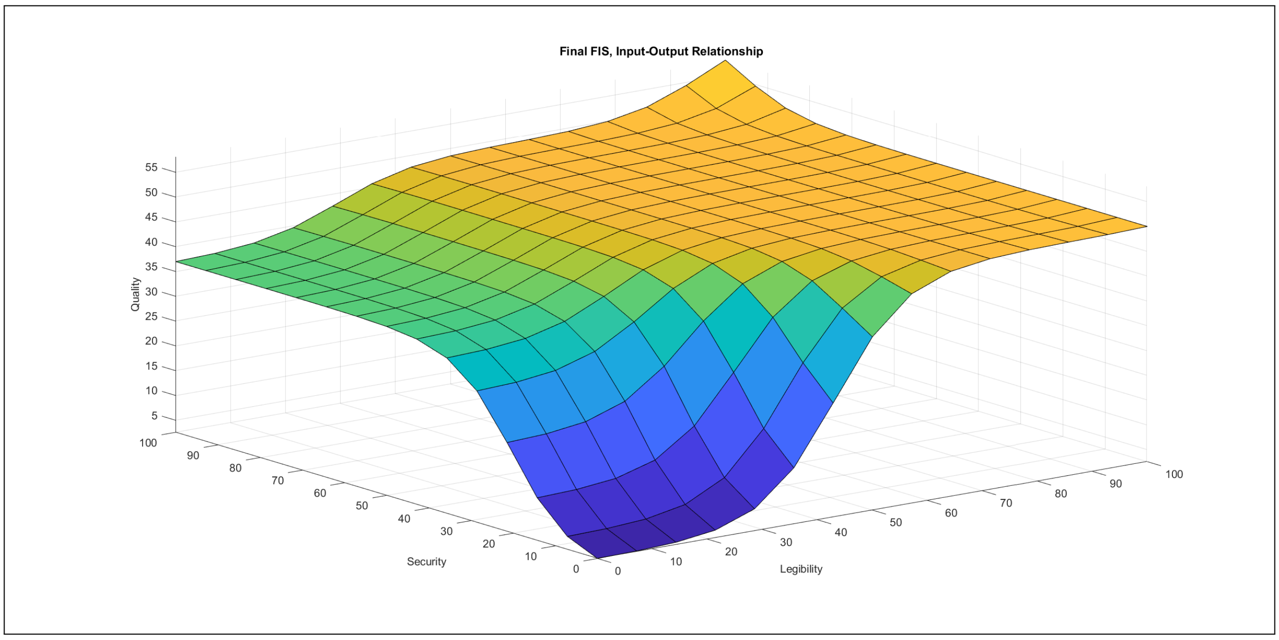

Figure 5.11: Effect of the Input Legibility and Security on Execution Tracing Quality After Tuning the Rules, Source: [68]

regard to the collected data set when direct, valid human experiences were incorporated in the rule base in the scope of the mini focus group and in the course of tuning the model based on the feedback of the international experts. This increase could probably have been diminished or eliminated if the sample size had been higher.

Construct validity represents the extent to which the findings reflect the content of the constructs in the phenomenon investigated [211], i.e. whether the model created really reflects execution tracing quality, including the input variables, and their impact on the output. On the one hand, the content of the constructs with regard to the input variables was elicited while defining the quality properties of execution tracing as introduced in chapter 4. On the other hand, the experiences collected from the software professionals describe the content of the phenomenon with regard to the effect of the input variables on the output: (1) the p=90% confidence intervals show acceptable ranges for the context of the research, (2) an adjustment of the created model was done by a mini focus group in the scope of an adjustment and pre-validation stage, and (3) international experts also validated the model. In addition, the method applied for modelling can capture and describe uncertainty; therefore, the deviations among the opinions collected from the software professionals did not cause a problem.

In summary, internal validity, external validity, content validity and construct validity of the research conducted are satisfactory. Nevertheless, increasing the sample size of the data collected for machine learning could have improved the above validity indicators. However, the low sample size does not cause a serious impediment in the present study as the technique used for modelling, fuzzy logic, allows the direct incorporation of human experiences accumulated in the problem domain, by which further data were incorporated in the model during the mini focus group discussion and in the course of tuning the model based on the opinions of international experts.

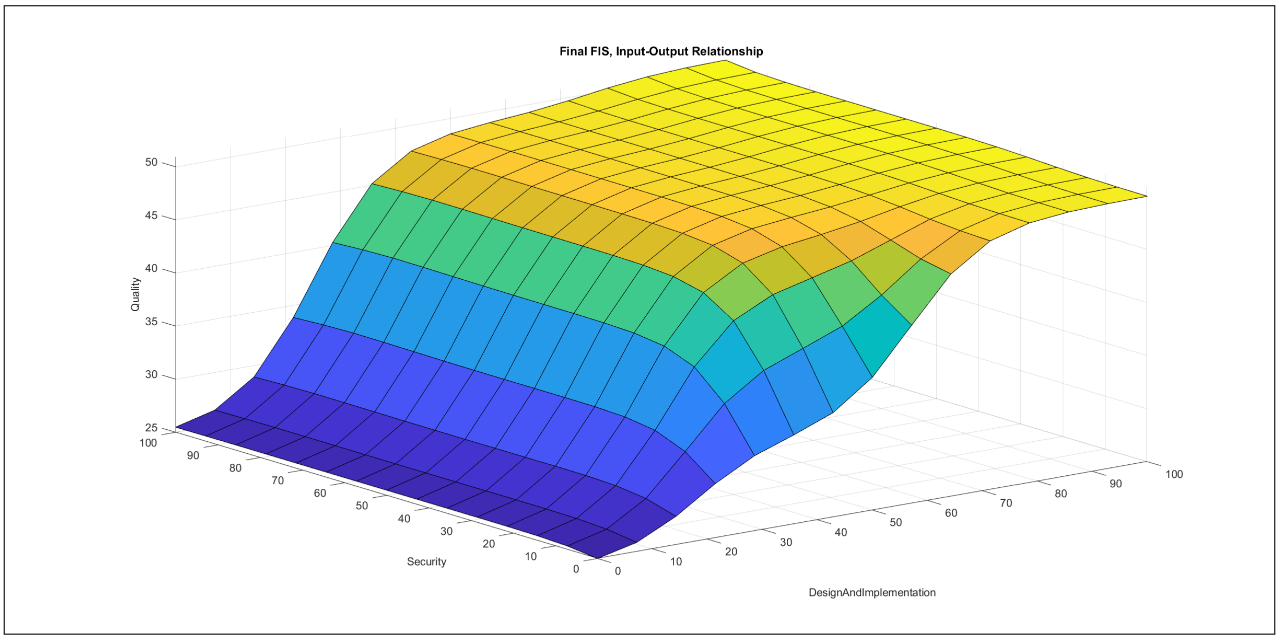

Figure 5.12: Effect of the Input Design and Implementation and Security on Execution Tracing Quality After Tuning the Rules, Source: [68]

5.6 Summary

Previous studies introduced focus on individual aspects of logging such as what to log, how to log and where to embed log statements in the source code of the applications. Overall guidelines on how to construct logging do not exist but they are desirable as stated in [32, 33, 87, 157, 162]. In addition, measuring logging quality quantitatively, setting quality targets for it, comparing log quality of different project or the same project at different points in time were not possible. For these reasons, it was necessary to create a quality model for execution tracing, which was done in the scope of the present research.

Uncertainty and vagueness are inherently present in the software product quality measurement and assessment process. In addition, the dispersion of the collected data in comparison to the range of the possible answers the respondents could give was high, which indicates vagueness associated with the rating process. Consequently, the application of fuzzy logic was an ideal candidate to capture and describe the phenomenon; on the other hand, more detailed guidelines can be elaborated how the respondents should judge the variables in question to decrease the uncertainty.

The application of fuzzy logic in the context of the present research shows the following main advantages: (1) uncertainty related to execution tracing quality and its inputs can be captured and described, (2) linguistic rules, which describe the problem domain in an human-understandable manner, are available, (3) human experience can directly be incorporated in the rule base to tune the model, (4) adaptive methods exist to extract the linguistic rules from a data set containing the experiences of software professionals. The properties of fuzzy systems mentioned in points 2 and 3 outperform those of the neural networks (NN). NNs neither produce a rule base in a human-understandable manner nor make possible the direct incorporation of human experiences in the system. In addition, the linguistic rules can also be regarded as guidelines for the software professionals how to implement a good-quality execution tracing mechanism, as they explicitly formulate which input variable combinations result in a {good, very good} output, and they also describe the poor results in linguistic terms; however, the rules also impact on each other's output.

In the scope of a pilot study [63] it was tested which approach best captures the uncertainty and vagueness in the context of execution tracing quality. The findings of the pilot study were used in the current research; therefore, an adaptive fuzzy system with overlapping Gaussian-shaped membership functions at the inputs, with Takagi-Sugeno-Kang inference mechanism [208] was selected for modelling. In contrast to the pilot study, the present research is based (1) on the elicited quality properties of execution tracing as shown in chapter 4, which formulate the input variables of the created quality model, (2) on the experiences of many software professionals from a well-defined study population, (3) on the linguistic rules extracted from the collected data by machine learning, (4) on the incorporation of direct human experiences into the model, created by computational intelligence, from software professional with many years of experience in the academia and in the industry, and (5) on the feedback from international experts, who also validated the created model.

While establishing the knowledge base of the fuzzy system in the form of linguistic rules, it is easy to introduce contradictions based on the different opinions of individual experts in the problem domain. To avoid this issue, the linguistic rules have been identified in an adaptive manner by fixing the input membership functions, the output membership functions and letting the system find the optimum root mean squared error (RMSE) by fitting the rules to the collected data by genetic algorithms. The same training procedure was repeated in a manner that all possible variations of the input partitions were combined into linguistic rules with separate output membership functions. In this second case, the linguistic rules were fixed and the adaptive process looked for the minimum RMSE while changing the parameters of the membership functions by adaptive-network-based fuzzy inference system approach (ANFIS). Both the GA and the ANFIS approach achieved the same accuracy, i.e. they supported the outcome of each other with regard to the data. The adaptive process was implemented as supervised learning, i.e. the optimal parameters of the membership functions and the best matching rules were identified in the scope of the learning process where input and output data pairs were provided for the system. After achieving the desired accuracy and performance, the trained system became available for carrying out the specified tasks. In the context of the present research, the output of the GA approach was used for further processing as it defined the knowledge base in the form of linguistic rules with upper bound, which has been further tuned and validated by experts.

In addition to the constructed quality model, the study identified the following findings: (1) The inputs accuracy and design and implementation have the most influence on the quality of execution tracing, these quality properties approximately determine the quality with the min-relationship, i.e. the lower value determines the quality of logging; (2) Legibility is nice-to-have. It helps to reduce the effort while localising errors; moreover, it reduces the psychological load on the software development and software maintenance professionals while performing the analysis. The absence of this quality property causes an unnecessary load on the staff and deteriorates analysis performance but it does not block the analysis itself, which might be the case with the quality properties accuracy and design and implementation if they are missing; (3) The quality property security is a feature that does not lie in the primary focus while localising errors although this feature also needs to be satisfied to some extent to avoid leaking sensitive information and to observe legal regulations. Tracing highly sensitive information might block the deployment of the application in certain domains due to legal regulations such as the medical and financial fields. The quality property security has a reverse-relationship with execution tracing quality as far as the amount of information is concerned. On the other hand, tracing sensitive information can also cause additional work to remove if the logs need to be passed on to other teams for further analysis.

The application domains: (1) server applications, (2) desktop applications, (3) web UIs, (4) mobile applications, and (5) embedded applications have different characteristics. These differences might influence the created model, the formalised linguistic rules can have different importance in each of the listed domains, which opens further tuning possibilities with regard to the specificities of these domains.

Chapter 6

Linking the Created Execution Tracing Quality Model to a Software Product Quality Framework

The goal of this chapter is to answer RQ6 in section 1.3, i.e. to link the execution tracing quality model, developed in chapter 5, to an overall quality framework to consider execution tracing with regard to the whole set of software product quality. The existing software product quality frameworks were identified and scored in chapter 2, which revealed that the quality model of the ISO/IEC 25010 standard is the most relevant for the industrial and scientific communities. In addition, the quality model of the ISO/IEC 25010 standard allows complex mathematical computations, which are necessary for the constructed execution tracing quality model. Furthermore, the author investigated the extension possibilities of different quality models in [62, 64]. The analysis identified the quality model of the ISO/IEC 25010 standard as first option for extension.

The extension process includes the following steps:

1. Identifying a node in the hierarchic structure of the ISO/IEC 25010 standard where execution tracing quality can be linked to.
2. Defining the quality measure elements to describe the measurable quality properties of execution tracing.
3. Defining the quality measure, including a measurement function to compute with, to describe execution tracing quality.
4. Optionally, formalising guidelines for tailoring to specific project needs as tailoring is encouraged in the ISO/IEC 25010 standard.

6.1 Identification of the Node for Extension

The hierarchic nature of the ISO/IEC 25010 quality model offers a characteristic: maintainability, which possesses further sub-characteristics, including analysability. As the primary goal of execution tracing is to analyse software defects, this node provides the basis for linking. In addition, the ISO/IEC 25010 standard defines the goal of analysability measures in a way, which further corroborates the possibility of linking [112][12]: “Analysability measures are used to assess the degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or cause of failure or to identify parts to be modified.”.

The predefined analysability measures, shown in Table 6.1 columns A and B, indicate the quality measure elements. In the case of the measure MAn-1-G, it is not possible to determine the correct or approximately correct value of the quality measure elements for a complex software system because the number of the necessary or required logs are unknown. Furthermore, the measures MAn-2-S and MAn-3-S describe the efficiency of the diagnostic functions, which do not characterise the analysability of the software or the prepared log output. This problem is introduced in detail in chapter 4 and in appendix B.1.2. Thus, execution tracing quality can be linked to the sub-characteristic analysability as a quality measure according to the terminology of the ISO/IEC 25010 standard.

Table 6.1: Analysability Measures of the ISO/IEC 25010 Standard, source: [112]

ID	Measure Name	Definition	Measurement Function	A	B
MAn-1-G	System log Compelteness	To what extent does the system record its operations in logs so that they are to be traceable?	X=A/B	Number of logs that are actually recorded in the system	Numer of logs for which audit trails are required during operation
MAn-2-S	Diagnosis function effectiveness	What proportion of diagnosis functions meets the requirements of casual analysis?	X=A/B	Number of diagnostic functions used for casual analysis	Number of diagnostic functions implemented
MAn-3-S	Diagnosis function sufficiency	What proportion of the required diagnosis functions has been implemented?	X=A/B	Number of diagnostic functions implemented	Number of diagnostic functions required

6.2 Definition of the Quality Measure Elements

The inputs of the model for execution tracing quality constitute the quality measure elements (QMEs), which were covered in section 4.5. The current section provides the definition following the format of the ISO/IEC 25000 software product quality model family. The standard identifies the QMEs with consecutive numbers. Adhering to the ISO/IEC 25010 conventions, the defined QMEs are prefixed with “ET” and numbered consecutively; moreover, the original variable names are also provided in brackets:

QME: ***ET-1 (Accuracy)***

Range: [0, 100]; Worst value: 0; Best value: 100

Scale type: Ratio

Definition: The variable defines accuracy of the execution tracing output with regard to its content, to what extent it trustworthily and consistently helps to identify the cause, the date, the time, and the location of the issue in the source code but not how legible the trace output is. Detailed description can be found in section 4.5.

QME input: Human judgement based on the output of execution tracing.

[12]Section 8.8.3 in the ISO/IEC 25010 Standard

QME: ***ET-2 (Legibility)***

Range: [0, 100]; Worst value: 0; Best value: 100

Scale type: Ratio

Definition: The variable defines the legibility of the execution tracing output but not the content itself. It shows how legible and user-friendly the output of execution tracing is. Detailed description can be found in section 4.5.

QME input: Human judgement based on the output of execution tracing.

QME: ***ET-3 (Design and Implementation of the Trace Mechanism)***

Range: [0, 100]; Worst value: 0; Best value: 100

Scale type: Ratio

Definition: The variable defines the quality of the design and implementation of the execution tracing mechanism with regard to how reliable, stable and sophisticated it is, to what extent and how easy it can be configured, activated and deactivated, whether sophisticated mechanisms for measuring performance are implemented, which produce lower impact on the application to reduce interference with the actions of execution. Detailed description can be found in section 4.5.

QME input: Human judgement based on the design, operation, configuration and source code of the execution tracing mechanism.

QME: ***ET-4 (Security)***

Range: [0, 100]; Worst value: 0; Best value: 100

Scale type: Ratio

Definition: The variable defines how secure the execution tracing mechanism and its outputs are, i.e., it shows to what extent the execution tracing mechanism and its outputs are exposed to vulnerabilities and how likely it is that they leak sensitive information. Detailed description can be found in section 4.5.

QME input: Human judgement based on the implementation and output of the execution tracing mechanism.

6.3 Definition of the Quality Measure

The quality model to describe execution tracing quality is defined in chapter 5. It is capable to handle uncertainty related to the quality measurement process. Linking this quality model to the ISO/IEC 25010 standard results in the quality measure *Execution Tracing Quality (ETQ)*, which comprises the whole quality model introduced in chapter 5; moreover, the quality model defines the corresponding quality measurement function to apply for an arbitrary project.

6.4 Guidelines and Illustration for Tailoring

The quality model defined in chapter 5 manifests the measurement function required by the ISO/IEC 25010 standard. However, specificities of the given project might need changes in comparison to a general setting, which basically offers two possibilities:

Option 1: Modifying the fuzzy rule base to consider the particularities of the given project and test the performance of the model, whether in each variable partition the results delivered are sensible as shown in section 5.5. This means repeating the adjustment and validation steps described in chapter 5, which is effort-intensive and requires the knowledge of fuzzy modelling.

Option 2: Aggregating the quality measure elements by weighted averages for the quality measure ETQ, which mirrors the experiences of the stakeholders in the given project. This approach follows the convention of the ISO/IEC 25000 standard family. The weights can be determined by any arbitrary method that reflects the opinions of the stakeholders, including constant sum scaling [171].

To illustrate option 2, the quality measures of the case study project in section 3.3.2 will be extended with the newly defined Execution Tracing Quality measure. The case study introduced a fictional but realistic project for which a software product quality model had to be selected with regard to given requirements. The development team, in discussion with the stakeholders, based on the taxonomy in table 3.1 selected the SQALE model implemented by SonarQube to automatically measure and assess internal quality, and the quality model of the ISO/IEC 25010 standard to measure and assess external and quality in-use quality manifestations. The original example has already demonstrated how the relevant quality characteristics can be selected for the context of use, which included (1) the external quality view: (a) performance efficiency; (b) reliability, and (2) quality in use: (a) satisfaction. In addition, quality measures and the quality measure elements to apply were also defined. Continuing the case study, the following realistic scenario is added: The development team and the stakeholders come to the conclusion, after investigating the error reports and their resolution times, that analysing the software faults and carrying out the corrections in the source code requires more time than expected. Thus, the development team and the stakeholders examine the quality assessment process and find that no quality property is considered for maintainability in the external quality view, which would exactly address the above problem. Consequently, they make the decision to include the maintainability characteristic with its analysability sub-characteristic into the quality measurement and assessment process. Furthermore, they consider the Execution Tracing Quality measure with its quality measure elements. In a further meeting, they identify the importance of the quality measure elements in the context of the project and come to the following findings in terms of the elements' weights: (1) ET-1 (Accuracy): 0.3, ET-2 (Legibility): 0.3, ET-3 (Design and Implementation of the Trace Mechanism): 0.3, ET-4 (Security): 0.1. Thus, the development team, in agreement with the stakeholders, measures and assesses maintainability as well. Table 6.2 demonstrates all measured characteristics, sub-characteristics and quality measures with the corresponding quality measure elements.

Table 6.2: Case Study 2 Continued: Tailoring the ISO/IEC 25010 Quality Model to the Needs of the Specific Project

View[13]	Quality Characteristic	Purpose	Quality Sub-characteristic	Quality Measure	Measurement Method	Computation Formula
E	Performance efficiency	How performantly the software behaves	Time-behaviour	Response time measure	Determine the most important business operations in the system or at external interfaces and measure the response time for them. Compute the measure for each business operation determined as defined by the formula.	$\frac{1}{1+\frac{time_{response}}{time_{maxallowed}}}$
E	Reliability	How reliably the software can operate	Availability	Crash measure [14]	Count the number of crashes and the number of "freezes" (when the software is available but it does not respond) in a given time frame, then compute the measure as defined by the formula.	$\frac{1}{1+count_{crash}+count_{freeze}}$
			Fault tolerance	Manual intervention measure	Count the number of manual interventions, which are necessary to maintain the operational state of the software in a given time frame, then compute the measure as defined by the formula.	$\frac{1}{1+count_{manualintervention}}$
E	Maintainability	How easy the software is to maintain	Analysability	Execution tracing quality measure	Determine the following quality measure elements in the range [0; 10]: (1) Accuracy, (2) Legibility, (3) Design and Implementation, (4) Security. The higher the value is, the higher the quality of the defined quality measure element is. Compute the measure as defined by the formula.	$\frac{0.3Q_A+0.3Q_L+0.3Q_{DAI}+0.1Q_S}{10}$
U	Satisfaction	How satisfied the end user is when the software is used	Usefulness	Usefulness goal measure	The end users assess the software how easily they can achieve their goals by the use of the software. The user assessment results in a mark in the range [0; 10]. The higher the value is, the higher the user's satisfaction is. Compute the measure as defined by the formula.	$\frac{\sum_{n=1}^{count_{users}} assessment_n}{10 * count_{users}}$

6.5 Summary

This chapter answered RQ6 in section 1.3. An individual quality model, which describes execution tracing quality, but ignores any other software product quality properties does not provide assistance when the software product as a whole needs to be measured and assessed. Therefore, linking the quality model for execution tracing quality to an overall software product quality framework is necessary.

The quality model for execution tracing was coupled to the software product quality framework of the ISO/IEC 25010 standard. This software product quality framework possesses the highest relevance in the industrial and scientific community [65]; and allows performing complex mathematical computations. In addition, the ISO/IEC 25010 quality framework can address all views of software product quality to capture all quality manifestations. Thus, the quality model for execution tracing was defined as a quality measure in accordance with the ISO/IEC 25010 standard, and the inputs of the constructed model were defined as quality measure elements.

The ISO/IEC 25010 standard encourages tailoring. Consequently, guidelines were provided to tailor the new quality measure, Execution Tracing Quality (ETQ), to specific project needs. Basically, two different ap-

[13] I: Internal quality view, E: External quality view, U: Quality in-use view

[14] If the software starts up quickly and automatically, then the up-time ratio does not appropriately mirror availability.

proaches can be implemented to tailor the quality measure ETQ: (1) adjusting the fuzzy rule base of the quality model introduced in chapter 5, or (2) creating a new measurement function for the measure ETQ, which utilises weighted averages of the defined quality measure elements to describe the experiences of the stakeholders.

Chapter 7

Conclusion and Future Work

This chapter concludes the thesis; it highlights the main research findings and potential future work plans. From a methodological point of view, the research presented in the PhD thesis illustrates: (1) how a problem, with no previous known research variables, can be framed, (2) how the research variables can be elicited by performing qualitative studies, (3) how the problem can be modelled quantitatively by means of artificial intelligence, (4) how the created model can be coupled to a larger context of the problem domain, and (5) how the created model can be adjusted and tailored to specific context of use.

7.1 Revisiting the Research Questions: Brief Answers

The research yielded the following findings as summarised for each research question below.

RQ1: What software product quality models aim to assess all defined characteristics of software product quality?
Answer: Software product quality frameworks defined, or tailored since the year 2000 were identified with a systematic literature review. In addition, the relevance of each software product quality model family was determined with regard to the industrial and scientific communities. The identified 23 software product quality model families were introduced including their terminology and concepts in a unified manner.

RQ2: Do the identified software product quality models handle execution tracing quality?
Answer: The systematic literature review verified that existing software product quality models do not adequately consider, measure and assess execution tracing quality.

RQ3: Which quality manifestations can the identified software product quality frameworks address?
Answer: The identified 23 software product quality model families were classified and their taxonomy provided. The classification extended the defined quality manifestations of the ISO/IEC 25010 standard, called quality views, to all identified software product quality model families, which includes those that use the term "quality view" as a homonym with different meaning in the own terminology of the given quality model. Moreover, those quality models that do not explicitly define the quality manifestations they can address were also investigated and classified.

RQ4: What quality properties, i.e. input variables, determine execution tracing quality?
Answer: The quality properties that impact on execution tracing quality were identified: (1) accuracy, (2) legibility, (3) design and implementation, and (4) security. The variable identification considered (1) accumulated experiences in the praxis of software professionals, and (2) appeared publications with implicit or explicit statements towards execution tracing quality.

RQ5: How can execution tracing quality be modelled with regard to uncertainty inherently present in the quality measurement process?
Answer: The collected data were analysed by means of the combination of genetic algorithms and fuzzy logic. This way linguistic rules could be extracted to describe the problem domain in a human-understandable manner and to define a fuzzy model. The linguistic rules were adjusted in the scope of a mini-focus group; while the constructed model was tuned using the feedback of international software professionals.

RQ6: To which software product quality framework should the quality model of execution tracing be linked, and how, to consider its effect in the overall software product quality?
Answer: The software product quality framework of the ISO/IEC 25010 standard is the best possible option to link the created execution tracing quality model to. This framework possesses the highest relevance in the industrial and scientific communities; moreover, it allows complex mathematical computations to aggregate inputs from the measured quality properties, which is necessary for the constructed execution tracing quality model. In addition, the ISO/IEC 25010 software product quality framework encourages tailoring to specific project needs and has a defined method to perform such adjustments.

7.2 Findings and Outcomes of the Research

This section summarises the research outcomes and provides references to the chapters with the specific contributions.

1. Identification of existing software product quality frameworks with the terminology and concepts of each model. This is considered a scientific contribution based on the systematic literature review with strict rules, documented in chapter 2.
2. Thorough analysis, in chapter 2, revealed that execution tracing quality was not appropriately handled in the identified software product quality frameworks, which has not changed with the developments in the field during the recent years.
3. In addition to the unified description of the identified software product quality frameworks, relevance indicators were developed and computed for each software product quality model. The relevance indicators, and their computations are documented in chapter 2.
4. A possible taxonomy of the identified software product quality frameworks with regard to the quality manifestations they are able to capture and describe was introduced in chapter 3.
5. Taxonomy based guidelines are provided to (1) select a software product quality model for a specific problem, and (2) assess a software product quality model with respect to the quality manifestations it

is able to handle. The latter can assist software professionals to avoid pitfalls caused by "good-quality software" statements based on software product quality models which capture a very limited extent of software product quality. This is explained in detail in chapter 3.

6. The identification of the quality properties of execution tracing based on a defined study population, which is a requirement for quantitative modelling and a necessity to provide practical guidelines for software professionals, is introduced in chapter 4.
7. While collecting the data to identify the quality properties of execution tracing, a simple but novel approach was used to estimate whether the saturation point was reached in the course of the data collection process. The approach is reported in chapter 4.
8. Modelling execution tracing quality, with the consideration of the uncertainty coming from (1) the experiences of many different software professionals from a defined study population, and (2) the software product quality measurement process, was performed. This is documented in chapter 5.
9. Extracting linguistic rules by machine learning to construct a fuzzy model and to gain insight into the problem domain in a human-understandable manner is documented in chapter 5.
10. Linking the quality model for execution tracing to a software product quality framework, which is relevant for both the industrial and scientific communities, is introduced in chapter 6.

7.3 Future Work Plans

Potential future work plans include (1) model simplification for execution tracing quality, (2) creating distinct models for different application and project profiles; and (3) contributing to automatic quality measurements and assessments on the field of external quality and quality in-use by the application of computational intelligence.

7.3.1 Simplification

The quality model developed for execution tracing applies fuzzy logic, the linguistic rules of which were extracted by genetic algorithms. The model was adjusted and further tuned based on the feedback of software professionals as introduced in chapter 5. In spite of the many advantages the application of fuzzy logic offers in the problem domain, including modelling uncertainty and using linguistic rules that are human understandable, it also implicates complexity. The inherent complexity of the quality model obtained in chapter 5 might be discouraging for deploying it in the quality measurement and assessment process. Therefore, the simplification of the constructed quality model is desired. The quality model for execution tracing can be thought of as a five-dimensional function with four inputs and one output. A data set with input-output pairs produced by the existing fuzzy model can be approximated with different regression approaches:

1. Regression by genetic programming: Genetic programming also makes it possible to constrain the allowed operations in the scope of a regression computation as illustrated in [202]. Consequently, the existing quality model for execution tracing could be approximated within defined bounds by means of

the combination of constants and the four basic operations: additions, subtractions, multiplications, and divisions.

2. Multivariate polynomial regression: The conventional statistical multivariate polynomial regression also offers a potential solution to approximate the existing quality model for execution tracing. The degrees of the polynomials need to be tested with regard to each input to achieve optimal performance.

Both alternative approaches to fuzzy logic imply the loss of the efficient modelling language of linguistic rules, in addition to not being able to handle uncertainty.

7.3.2 Distinct Models of Execution Tracing Quality for Predefined Profiles

Application domains, including (1) server applications, (2) desktop applications, (3) web UIs, (4) mobile applications, and (5) embedded applications, might have an impact on the particularities of modelling execution tracing quality. In addition, software project domains, such as (1) medical, (2) financial, (3) telecommunication, (4) military, might also influence the importance of each individual input of the model for execution tracing quality. Consequently, for the combination of each possible application and software project domain, a distinct profile can be created, and the model for execution tracing quality can be adjusted to that profile. This approach would assist to tailor the existing quality model to predefined scenarios.

7.3.3 Automation

Since the urge for automation on the field of quality assessment is very strong [65], and the external and quality in-use views inherently need human evaluation, the application of computational intelligence to mimic human behaviour and to reduce manual interventions in the measurement and assessment process is a potential future research avenue.

Many research works focus on software defect prediction by using artificial intelligence techniques or modern machine-learning analytics [8, 125, 138, 170, 172, 191, 193, 203] but few concentrate on the assessment of external or quality in-use manifestations [145, 174, 246]. As a potential future goal, the author plans to adapt the constructed model for execution tracing quality to the existing automation approaches to replace, in part or in special situations in full, the human evaluations required by the conventional measurement and assessment of external and quality in-use quality manifestations.

References

[1] 2nd international conference on network computing and information security, ncis 2012. In *2nd International Conference on Network Computing and Information Security, NCIS 2012*. Springer, 2012.

[2] 2014 world congress on computer applications and information systems, WCCAIS 2014. In *2014 World Congress on Computer Applications and Information Systems, WCCAIS 2014*. Institute of Electrical and Electronics Engineers Inc., 2014.

[3] Proceedings - 26th international workshop on software measurement, iwsm 2016 and the 11th international conference on software process and product measurement, mensura 2016. In *Proceedings - 26th International Workshop on Software Measurement, IWSM 2016 and the 11th International Conference on Software Process and Product Measurement, Mensura 2016*. Institute of Electrical and Electronics Engineers Inc., 2017.

[4] Proceedings - 2016 10th international conference on the quality of information and communications technology, quatic 2016. In *Proceedings - 2016 10th International Conference on the Quality of Information and Communications Technology, QUATIC 2016*. Institute of Electrical and Electronics Engineers Inc., 2017.

[5] S. Agarwala, Y. Chen, D. Milojicic, and K. Schwan. QMON: qos- and utility-aware monitoring in enterprise systems. In *IEEE International Conference on Autonomic Computing*, pages 124–133, 2006. doi: 10.1109/ICAC.2006.1662390.

[6] K. K. Aggarwal, Y. Singh, P. Chandra, and M. Puri. Measurement of software maintainability using a fuzzy model. *Journal of Computer Sciences*, pages 541–, 2005.

[7] K. K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra. Investigating effect of design metrics on fault proneness in object-oriented systems. *Journal of Object Technology*, 6(10):127–141, 2007.

[8] M. R. Ahmed, M. A. Ali, N. Ahmed, M. F. B. Zamal, and F. J. M. Shamrat. The impact of software fault prediction in real-world application: An automated approach for software engineering. In *Proceedings of 2020 the 6th International Conference on Computing and Data Engineering*, ICCDE 2020, page 247–251, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450376730. doi: 10.1145/3379247.3379278. URL `https://doi.org/10.1145/3379247.3379278`.

[9] M. Andreolini, M. Colajanni, M. Pietri, and S. Tosi. Real-time adaptive algorithm for resource monitoring. In *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pages 67–74, 2013. ISBN 2165-9605. doi: 10.1109/CNSM.2013.6727811.

[10] A. S. Andreou and M. Tziakouris. A quality framework for developing and evaluating original software components. *Information and Software Technology*, 49(2):122–141, 2007. doi: 10.1016/j.infsof.2006.03.007.

[11] Apache Software Foundation. Apache commons logging, best practices. [Online], [Accessed: 04.09.2020.], 2014. URL `http://commons.apache.org/proper/commons-logging/guide.html#JCL_Best_Practices`.

[12] G. C. Apostol and F. Pop. Mice: Monitoring high-level events in cloud environments. In *2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pages 377–380, May 2016. doi: 10.1109/SACI.2016.7507405.

[13] I. Atterzadeh and S. H. Ow. A novel soft computing model to increase the accuracy of software development cost estimation. In *Proceedings of the 2nd International Conference on Computer and Automation Engineering*, pages 603–607, 2010.

[14] T. Bakota, P. Hegedus, P. Kortvelyesi, R. Ferenc, and T. Gyimothy. A probabilistic software quality model. In *27th IEEE International Conference on Software Maintenance (ICSM)*, 2011. doi: 10.1109/ICSM.2011.6080791.

[15] T. Bakota, P. Hegedus, and G. Ladanyi. A cost model based on software maintainability. In *28th IEEE International Conference on Software Maintenance (ICSM)*, 2012. doi: 10.1109/ICSM.2012.6405288.

[16] F. Balmas, F. Bellingard, S. Denier, S. Ducasse, B. Franchet, J. Laval, K. Mordal-Manet, and P. Vaillergues. Practices in the Squale quality model (squale deliverable 1.3). [Online], October, 2010, [Accessed: 16.11.2017], 2010. URL `http://www.squale.org/quality-models-site/research-deliverables/WP1.3Practices-in-the-Squale-Quality-Modelv2.pdf`.

[17] F. Balmas, A. Bergel, F. Bellingard, S. Denier, S. Ducasse, J. Laval, K. Mordal-Manet, H. Abdeen, and F. Bellingard. Software metric for java and c++ workpackage: 1.1 version: 2. [Online], March, 2010, [Accessed: 16.11.2017], 2010. URL `http://www.squale.org/quality-models-site/research-deliverables/WP1.1Software-metrics-for-Java-and-Cpp-practicesv2.pdf`.

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

[19] F. Basciani, D. D. Ruscio, L. Iovino, and A. Pierantonio. Automated quality assessment of interrelated modeling artifacts. In *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 234–243, Sep. 2021. doi: 10.1109/SEAA53835.2021.00037.

[20] L. Benedicenti, V. W. Wang, and R. Paranjape. A quality assessment model for java code. In *Canadian Conference on Electrical and Computer Engineering*, volume 2, pages 687–690, 2002.

[21] J. Bernardes Boarim and A. R. Cavalcanti da Rocha. Crm systems quality evaluation. In *XX Brazilian Symposium on Software Quality*, SBQS '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450395533. doi: 10.1145/3493244.3493273. URL `https://doi.org/10.1145/3493244.3493273`.

[22] M. Berry and C. S. Johnson. *Software Process and Product Measurement*, chapter Improving the Quality of Information for Software Project Management, pages 1–20. Springer-Verlag, Berlin, Heidelberg,

2008. ISBN 978-3-540-85552-1. doi: 10.1007/978-3-540-85553-81. URL `http://dx.doi.org/10.1007/978-3-540-85553-81`.

[23] M. Berry and C. S. Johnson. *Improving the quality of information for software project management*, volume 4895 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2008.

[24] M. J. Blas. An analysis model to evaluate web applications quality using a discrete-event simulation approach. In *2017 Winter Simulation Conference (WSC)*, pages 4648–4649, 2017. doi: 10.1109/WSC.2017.8248248.

[25] B. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981. ISBN 0-13-822122-7.

[26] B. Boehm and S. Chulani. Modeling software defect introduction and removal – COQUALMO (constructive quality model). Technical report, USC-CSE Technical Report, 1999.

[27] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd International Conference on Software Engineering*, 1976.

[28] X. Burgués, X. Franch, and J. M. Ribó. *A MOF-compliant approach to software quality modeling*, volume 3716 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2005.

[29] G. Canfora, L. Cerulo, and L.Troiano. Can fuzzy mathematics enrich the assessment of software maintainability? *ICEISSAM - Software Audit and Metrics*, 2004.

[30] C. K. Chang and T. hyung Kim. Distributed systems design using function-class decomposition with aspects. In *Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems*, pages 148–153, 2004. ISBN 0-7695-2118-5. doi: 10.1109/ICECCS.1995.479364.

[31] B. Chen and Z. M. (Jack) Jiang. Characterizing logging practices in java-based open source software projects — a replication study in apache software foundation. *Empirical Softw. Engg.*, 22(1):330–374, Feb. 2017. ISSN 1382-3256. doi: 10.1007/s10664-016-9429-5. URL `https://doi.org/10.1007/s10664-016-9429-5`.

[32] B. Chen and Z. M. Jiang. Extracting and studying the logging-code-issue-introducing changes in java-based large-scale open source software systems. *Empirical Softw. Engg.*, 24(4):2285–2322, Aug. 2019. ISSN 1382-3256. doi: 10.1007/s10664-019-09690-0. URL `https://doi.org/10.1007/s10664-019-09690-0`.

[33] B. Chen and Z. M. J. Jiang. Characterizing and detecting anti-patterns in the logging code. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, page 71–81. IEEE Press, 2017. ISBN 9781538638682. doi: 10.1109/ICSE.2017.15. URL `https://doi.org/10.1109/ICSE.2017.15`.

[34] D. D. Chen, W. S. Lim, M. Bakhshalipour, P. B. Gibbons, J. C. Hoe, and B. Parno. Herqules: Securing programs via hardware-enforced message queues. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS 2021, page 773–788, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383172. doi: 10.1145/3445814.3446736. URL `https://doi.org/10.1145/3445814.3446736`.

[35] T. Coq. Verification and validation in the recommended practice for integrated software-dependent systems. In *First International Conference on Advances in System Testing and Validation Lifecycle*, pages 57–61, 2009. doi: 10.1109/VALID.2009.36.

[36] J. Correia and J. Visser. Certification of technical quality of software products. In *International Workshop on Foundations and Techniques for Open Source Software Certification*, pages 35–51, 2008.

[37] J. P. Correia, Y. Kanellopoulos, and J. Visser. A survey-based study of the mapping of system properties to iso/iec 9126 maintainability characteristics. In *2009 IEEE International Conference on Software Maintenance*, pages 61–70, 2009. ISBN 1063-6773. doi: 10.1109/ICSM.2009.5306346.

[38] M.-A. Côté, W. Suryn, R. A. Martin, and C. Y. Laporte. Evolving a corporate software quality assessment exercise: A migration path to ISO/IEC 9126. *Software Quality Professional*, 6(3):4–17, Jun 2004.

[39] M.-A. Côté, W. Suryn, and E. Georgiadou. In search for a widely applicable and accepted software quality model for software quality engineering. *Software Quality Journal*, 15(4):401–416, 2007. doi: 10.1007/s11219-007-9029-0.

[40] A. Davila, K. Melendez, and L. Flores. Establishing software product quality requirements according to international standards. *IEEE Latin America Transactions*, 4(2):100–106, April 2006. doi: 10.1109/TLA.2006.1642457.

[41] F. Deissenboeck, E. Juergens, K. Lochmann, and S. Wagner. Software quality models: Purposes, usage scenarios and requirements. In *2009 ICSE Workshop on Software Quality*, pages 9–14, 2009.

[42] F. Deissenboeck, L. Heinemann, M. Herrmannsdoerfer, K. Lochmann, and S. Wagner. The quamoco tool chain for quality modeling and assessment. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 1007–1009, New York, NY, USA, 2011. ACM. ISBN 9781-450304450. doi: 10.1145/1985793.1985977.

[43] R. Ding, H. Zhou, J.-G. Lou, H. Zhang, Q. Lin, Q. Fu, D. Zhang, and T. Xie. Log2: A cost-aware logging mechanism for performance diagnosis. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 139–150, Santa Clara, CA, July 2015. USENIX Association. ISBN 978-1-931971-225. URL `https://www.usenix.org/conference/atc15/technical-session/presentation/ding`.

[44] F. J. Domínguez-Mayo, M. J. Escalona, M. Mejías, M. Ross, and G. Staples. Quality evaluation for model-driven web engineering methodologies. *Information and Software Technology*, 54(11):1265–1282, 2012. doi: //doi.org/10.1016/j.infsof.2012.06.007.

[45] R. Dromey. A model for software product quality. In *IEEE Transactions on Software Engineering*, volume 21, pages 146–162, 1995.

[46] S. Ducasse, S. Denier, F. Balmas, A. Bergel, J. Laval, K. Mordal-Manet, and F. Bellingard. Visualization of practices and metrics, workpackage: 1.2 version: 1.1. [Online], March, 2010, [Accessed: 16.11.2017], 2010. URL `http://www.squale.org/quality-models-site/research-deliverables/WP1.2Visualization-of-Practices-and-Metricsv1.1.pdf`.

[47] P. Eeles. Appendix c: Sample architectural requirements questionnaire. Online, [Accessed: 20.04.2018], Nov 2004. URL `https://www.ibm.com/developerworks/rational/library/4710.html`.

[48] P. Eeles. Capturing architectural requirements. IBM [Online], [Accessed: 16.11.2018], 2005. URL https://www.ibm.com/developerworks/rational/library/4706-pdf.pdf.

[49] P. Eeles. Capturing architectural requirements. Online, [Accessed: 19.04.2018], Nov 2005. URL https://www.ibm.com/developerworks/rational/library/4706-pdf.pdf.

[50] P. Eeles. Appendix b: Architectural requirements. Online, [Accessed: 20.04.2018], Nov 2005. URL https://www.ibm.com/developerworks/rational/library/4708.html.

[51] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag Berlin, Heidelberg, Germany, 2010. ISBN 978-3-642-07285-7.

[52] M. Ericsson, W. Löwe, T. Olsson, D. Toll, and A. Wingkvist. A study of the effect of data normalization on software and information quality assessment. In *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, volume 2, pages 55–60, 2013. ISBN 1530-1362. doi: 10.1109/APSEC.2013.112.

[53] D. Falessi, M. Sabetzadeh, L. Briand, E. Turella, T. Coq, and R. K. Panesar-Walawege. Planning for safety standards compliance: A model-based tool-supported approach. *IEEE Software*, 29(3):64–70, 2012. doi: 10.1109/MS.2011.116.

[54] R. Ferenc, P. Hegedüs, and T. Gyimóthy. *Software Product Quality Models, Chapter*. In book: Evolving Software Systems. Springer, Berlin, Heidelberg, 2014. doi: 10.1007/978-3-642-45398-43.

[55] A. Flaig, D. Hertl, and F. Krüger. Evaluation of java profiling tools. Universität Stuttgart, Institut für Softwaretechnologie, Abteilung Zuverlässige Softwaresysteme, Online, [Accessed: 22.01.2022], DOI: http://dx.doi.org/10.18419/opus-3222. URL https://elib.uni-stuttgart.de/handle/11682/3239.

[56] D. B. Flora. Your coefficient alpha is probably wrong, but which coefficient omega is right? a tutorial on using r to obtain better reliability estimates. *Advances in Methods and Practices in Psychological Science*, 3(4), 2020. doi: 10.1177/2515245920951747. URL https://doi.org/10.1177/2515245920951747.

[57] O. Foley and M. Helfert. Information quality and accessibility. In *Innovations and Advances in Computer Sciences and Engineering*, pages 477–481, 2010. doi: 10.1007/978-90-481-3658-284.

[58] S. Forouzani, Y. K. Chiam, and S. Forouzani. Method for assessing software quality using source code analysis. In *ACM International Conference Proceeding Series*, pages 166–170. Association for Computing Machinery, 2016. doi: 10.1145/3033288.3033316.

[59] D. Franke and C. Weise. Providing a software quality framework for testing of mobile applications. In *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation, ICST 2011*, pages 431–434, 2011. doi: 10.1109/ICST.2011.18.

[60] D. Freedaman, R. Pisani, and R. Purves. *Statisztika (Translated Title: Statistics)*. Typotex, Budapest, Hungary, 2005.

[61] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie. Where do developers log? an empirical study on logging practices in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, page 24–33, New York, NY, USA, 2014.

Association for Computing Machinery. ISBN 9781450327688. doi: 10.1145/2591062.2591175. URL `https://doi.org/10.1145/2591062.2591175`.

[62] T. Galli. Fuzzy logic based software product quality model for execution tracing. MPhil Thesis, Centre for Computational Intelligence, De Montfort University, Leicester, UK, [Online], 2013, [Accessed: 05.02.2018]. URL `https://www.dora.dmu.ac.uk/bitstream/handle/2086/9736/MPhilThesisTamasGalli2013ExaminedFinal.pdf`.

[63] T. Galli, F. Chiclana, J. Carter, and H. Janicke. Modelling execution tracing quality by type-1 fuzzy logic. *Acta Polytechnica Hungarica*, 8(10):49–67, 2013. doi: 10.12700/APH.10.08.2013.8.3.

[64] T. Galli, F. Chiclana, J. Carter, and H. Janicke. Towards introducing execution tracing to software product quality frameworks. *Acta Polytechnica Hungarica*, 11(3):5–24, 2014. doi: 10.12700/APH.11.03.2014.03.1.

[65] T. Galli, F. Chiclana, and F. Siewe. Software product quality models, developments, trends and evaluation. *SN Computer Science*, 2020. doi: 10.1007/s42979-020-00140-z.

[66] T. Galli, F. Chiclana, and F. Siewe. Performance of execution tracing with aspect-oriented and conventional approaches. In V. Gupta and C. Gupta, editors, *Research and Evidence in Software Engineering, From Empirical Studies to Open Source Artifacts*, pages 1–40. Auerbach Publications, Routledge, Taylor and Francis Group, 2021. ISBN 9780367358525.

[67] T. Galli, F. Chiclana, and F. Siewe. Genetic algorithm-based fuzzy inference system for describing execution tracing quality - collected data [data set]. *Zenodo*, October 2021. doi: 10.5281/zenodo.5552684.

[68] T. Galli, F. Chiclana, and F. Siewe. Genetic algorithm-based fuzzy inference system for describing execution tracing quality. *Mathematics*, 9(21), 2021. ISSN 2227-7390. doi: https://doi.org/10.3390/math9212822. URL `https://www.mdpi.com/2571-5577/4/1/20`.

[69] T. Galli, F. Chiclana, and F. Siewe. Quality properties of execution tracing, an empirical study. *Applied System Innovation*, 4(1), 2021. ISSN 2571-5577. doi: 10.3390/asi4010020. URL `https://www.mdpi.com/2571-5577/4/1/20`.

[70] T. Galli, F. Chiclana, and F. Siewe. On the use of quality models to address distinct quality views. *Applied System Innovation*, 4(3), 2021. ISSN 2571-5577. doi: 10.3390/asi4030041. URL `https://www.mdpi.com/2571-5577/4/3/41`.

[71] E. Georgiadou. GEQUAMO — a generic, multilayered, customisable, software quality model. *Software Quality Journal*, 11(4):313–323, Nov 2003. doi: 1025817312035.

[72] J. D. Gibbons and S. Chakraborti. *Non-Parametric Statistical Inference*. Statistics: Textbooks and Monographs. Marcel Dekker, 270 Madison Avenue, New York, NY 10016, 2003. ISBN 0-8247-4052-1.

[73] M. Gleirscher, D. Golubitskiy, M. Irlbeck, and S. Wagner. Introduction of static quality analysis in small- and medium-sized software enterprises: experiences from technology transfer. *Software Quality Journal*, 22(3):499–542, 2014. doi: 10.1007/s11219-013-9217-z.

[74] P. Godefroid and N. Nagappan. Concurrency at Microsoft – an exploratory survey. [Online], [Accessed: 12.03.2019], 2007. URL `https://patricegodefroid.github.io/publicpsfiles/ec2.pdf`.

[75] J. Gong, J. Lu, and L. Cai. An induction to the development of software quality model standards. In *2016 Third International Conference on Trustworthy Systems and their Applications (TSA)*, pages 117–122, 2016. doi: 10.1109/TSA.2016.28.

[76] Google. Google search trends for the past 12 months, worldwide. [Online], [Accessed: 17.02.2020], 2020. URL `https://trends.google.com/trends/explore`.

[77] R. Grady. *Practical Software Metrics for Project Management and Process Improvement.* Prentice Hall, 1992.

[78] R. Grady and D. Caswell. *Software Metrics: Establishing a Company-Wide Program.* Prentice Hall, 1987. ISBN 0138218447.

[79] R. B. Grady. *Practical Software Metrics for Project Management and Process Improvement.* Prentice Hall, USA, NJ, 1992. ISBN 0-13-720384-5.

[80] R. B. Grady and D. L. Caswell. *Software Metrics: Establishing a Company-wide Program.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987. ISBN 0-13-821844-7.

[81] A. D. Gronewold and M. E. Borsuk. A software tool for translating deterministic model results into probabilistic assessments of water quality standard compliance. *Environmental Modelling and Software*, 24(10):1257–1262, 2009. doi: //doi.org/10.1016/j.envsoft.2009.04.004.

[82] A. S. Guceglioglu and O. Demirors. *A process based model for measuring process quality attributes*, volume 3792 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* 2005. doi: 10.1007/1158601212.

[83] R. Gulezian. Software quality measurement and modeling, maturity, control and improvement. In *Proceedings of Software Engineering Standards Symposium*, pages 52–59, 1995. ISBN 1082-3670. doi: 10.1109/SESS.1995.525951.

[84] O. Gurbuz, A. S. Guceglioglu, and O. Demirors. Application of process quality measurement frameworks for human resource management processes. In *2011 IEEE International Conference on Quality and Reliability*, pages 426–430, Sept 2011. doi: 10.1109/ICQR.2011.6031754.

[85] S. Hamdan and S. Alramouni. A quality framework for software continuous integration. *Procedia Manufacturing*, 3:2019–2025, 2015. doi: 10.1016/j.promfg.2015.07.249.

[86] S. Han, R. Sinha, and A. Lowe. Assessing support for industry standards in reference medical software architectures. In *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, pages 3403–3407, Oct 2020. doi: 10.1109/IECON43393.2020.9255309.

[87] M. Hassani, W. Shang, E. Shihab, and N. Tsantalis. Studying and detecting log-related issues. *Empirical Softw. Eng.*, 23(6):3248–3280, Dec. 2018. ISSN 1382-3256. doi: 10.1007/s10664-018-9603-z. URL `https://doi.org/10.1007/s10664-018-9603-z`.

[88] D. Headquarters. Dynatrace. Online, [Accessed: 22.01.2022]. URL `https://www.dynatrace.com/solutions/`.

[89] P. Hegedus. A probabilistic quality model for c# — an industrial case study. *Acta Cybernetica*, 21(1): 135–147, 2013.

[90] P. Hegedus. Revealing the effect of coding practices on software maintainability. In *29th IEEE International Conference on Software Maintenance (ICSM)*, 2013. doi: 10.1109/ICSM.2013.99.

[91] J. H. Hegeman. On the quality of quality models. MSc Thesis, University Twente, [Online], [Accessed: 16.11.2018], 2011. URL `https://essay.utwente.nl/61040/1/MScJHegeman.pdf`.

[92] I. Heitlager, T. Kuipers, , and J. Visser. A practical model for measuring maintainability. In *Proceedings of the Quality of Information and Communications Technology QUATIC 2007*, page 30–39, 2007.

[93] G. Horgan and S. Khaddaj. Use of an adaptable quality model approach in a production support environment. *The Journal of Systems and Software*, 82(4):730–738, 2009. doi: 10.1016/j.jss.2008.10.009.

[94] W. Hu, T. Loeffler, and J. Wegener. Quality model based on ISO/IEC 9126 for internal quality of matlab/simulink/stateflow models. In *2012 IEEE International Conference on Industrial Technology*, pages 325–330, 2012. doi: 10.1109/ICIT.2012.6209958.

[95] T. M. Hughes. *SAS® Data Analytic Development: Dimensions of Software Quality*, pages 1–606. SAS® Data Analytic Development: Dimensions of Software Quality. wiley, 2016. doi: 10.1002/9781119255680.

[96] L. E. Hyatt and L. H. Rosenberg. A software quality model and metrics for identifying project risks and assessing software quality. In *Proceedings of Product Assurance Symposium and Software Product Assurance Workshop, EAS SP-377, European Space Agency*, 19-21 March 1996.

[97] A. Idri, M. Bachiri, J. L. Fernandez-Aleman, and A. Toval. Experiment design of free pregnancy monitoring mobile personal health records quality evaluation. pages 1–6. IEEE, 2016. doi: 10.1109/HealthCom.2016.7749501.

[98] A. Idri, M. Bachiri, and J. L. Fernández-Alemán. A framework for evaluating the software product quality of pregnancy monitoring mobile personal health records. *Journal of medical systems*, 40(3): 1–17, 2016. doi: 10.1007/s10916-015-0415-z.

[99] IEEE Computer Society. *IEEE Stdandard 1061-1998: IEEE Standard for a Software Quality Metrics Methodology*. IEEE, 1998.

[100] INRIA RMoD, Paris 8, Qualixo. Technical model for remediation (workpackage 2.2), [online], [accessed: 16.11.2017], Sep 30 2010. URL `http://www.squale.org/quality-models-site/research-deliverables/WP2.2Technical-Model-for-Remediationv1.pdf`.

[101] International Organization for Sandardization. ISO/IEC 25020:2007, software engineering - software product quality requirements and evaluation (SQauRE) - measurement reference model and guide, 2007.

[102] International Organization for Sandardization. ISO/IEC 25021:2012, software engineering - software product quality requirements and evaluation (SQauRE) - quality measure elements, 2012.

[103] International Organization for Sandardization. ISO/IEC 25022:2016, software engineering - software product quality requirements and evaluation (SQauRE) - measurement of quality in use, 2016.

[104] International Organization for Standardization. ISO/IEC 9126:1991, software enginnering – product quality, 1991.

[105] International Organization for Standardization. ISO/IEC 14598:1999, information technology – software product evaluation – part 1: General overview, 1999.

[106] International Organization for Standardization. ISO/IEC 9126-1:2001, software engineering – product quality – part 1: Quality model, 2001.

[107] International Organization for Standardization. ISO/IEC TR 9126-2:2003, software engineering – product quality – part 2: External metrics. 2003.

[108] International Organization for Standardization. ISO/IEC TR 9126-3:2003, software engineering – product quality – part 3: Internal metrics. 2003.

[109] International Organization for Standardization. ISO/IEC TR 9126-4:2004, software engineering – product quality – part 4: Quality in use metrics. 2004.

[110] International Organization for Standardization. ISO/IEC 12207:2008, systems and software engineering – software life cycle processes, 2008.

[111] International Organization for Standardization. ISO/IEC 25012:2008, software engineering – software product quality requirements and evaluation (SQuaRE) – data quality model, 2008.

[112] International Organization for Standardization. ISO/IEC 25010:2011, systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – system and software quality models, 2011.

[113] International Organization for Standardization. ISO/IEC 25023:2016, systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – measurement of system and software product quality, 2016.

[114] S. G. Isaksen and J. P. Gaulin. A reexamination of brainstorming research: Implications for research and practice. *Gifted Chiled Quarterly The Official Journal of the National Association for Gifted Children*, 49.(4.), 2005.

[115] ISO25000.com. ISO 25000 Certification Portal. [Online], [Accessed: 20.02.2018.]. URL `http://iso25000.com/index.php/en/iso-25000-standards/52-iso-iec-2502n`.

[116] J.-S. R. Jang. Anfis: Adaptive-network-based fuzzy inference system. In *IEEE Transactions on Systems, Man and Cybernetics*, pages 665–685, 1993.

[117] J.-S. R. Jang, C.-T. Sun, and E. Mizutani. *Neuro-Fuzzy and Soft Computing*. Prentice Hall, 1997.

[118] S. Kabinna, C. Bezemer, W. Shang, and A. E. Hassan. Logging library migrations: A case study for the apache software foundation projects. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 154–164, 2016.

[119] S. Kabinna, C.-P. Bezemer, W. Shang, M. D. Syer, and A. E. Hassan. Examining the stability of logging statements. *Empirical Softw. Engg.*, 23(1):290–333, Feb. 2018. ISSN 1382-3256. doi: 10.1007/s10664-017-9518-0. URL `https://doi.org/10.1007/s10664-017-9518-0`.

[120] Y. Kanellopoulos, C. Tjortjis, I. Heitlager, and J. Visser. Interpretation of source code clusters in terms of the ISO/IEC-9126 maintainability characteristics. In *Proceedings of the European Conference on*

Software Maintenance and Reengineering, CSMR, pages 63–72, 2008. doi: 10.1109/CSMR.2008.4493 301.

[121] A. Karahasanovic and R. Thomas. Difficulties experienced by students in maintaining object-oriented systems: An empirical study. *Proceedings of the 9th Australasian Conference on Computing Education*, pages 81–87, 2007.

[122] F. Karim and H. Thanneer. A classification scheme for evaluating management instrumentation in distributed middleware infrastructure. In *Proceedings of 4th IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services, held in conjunction with 10th IEEE/IFIP Network Operations and Management Symposium*, Vancouver, Canada, April 2006.

[123] F. Karim and H. Thanneer. Automated health-assessment of software components using management instrumentation. In *Proceedings of the 30th Annual International Computer Software and Applications Conference - Volume 02*, pages 177–182, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0769-526551. doi: 10.1109/COMPSAC.2006.117. URL `http://dx.doi.org/10.1109/COMPSAC.2006.117`.

[124] S. Khaddaj and G. Horgan. A proposed adaptable quality model for software quality assurance. *Journal of Computer Science*, 1(4):482–487, Apr 2005. doi: 10.3844/jcssp.2005.482.487.

[125] B. Khan, D. Iqbal, and S. Badshah. Cross-project software fault prediction using data leveraging technique to improve software quality. In *Proceedings of the Evaluation and Assessment in Software Engineering*, EASE '20, page 434–438, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450377317. doi: 10.1145/3383219.3383281. URL `https://doi.org/10.1145/3383219.3383281`.

[126] T. M. Khoshgoftaar and E. B. Allen. Multivariate assessment of complex software systems: a comparative study. In *Engineering of Complex Computer Systems, 1995. Held jointly with 5th CSESAW, 3rd IEEE RTAW and 20th IFAC/IFIP WRTP, Proceedings., First IEEE International Conference on*, pages 389–396, 1995. doi: 10.1109/ICECCS.1995.479364.

[127] B. Khosravifar, J. Bentahar, A. Moazin, and P. Thiran. Analyzing communities of web services using incentives. *Int.J.Web Serv.Res.*, 7(3):30–51, jul 2010. doi: 10.4018/jwsr.2010070102. URL `http://dx.doi.org/10.4018/jwsr.2010070102`.

[128] C. Kim and K. Lee. Software quality model for consumer electronics product. In *Proceedings of the 9th International Conference on Quality Software*, pages 390–395, 2008.

[129] B. Kitchenham and P. Brereton. A systematic review of systematic review process research in software engineering. *Information and Software Technology*, 55:2049–2075, 2013.

[130] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report, EBSE-2007-01, 2007.

[131] B. Kitchenham and S. Pfleeger. Software quality: the elusive target. *IEEE Software*, 13(1):12–21, 1996.

[132] B. Kitchenham, S. Linkman, A. Pasquini, and V. Nanni. The SQUID approach to defining a quality model. *Software Quality Journal*, 6(3):211–233, Sep 1997. doi: 1018516103435.

[133] B. Kitchenham, R. Pretorius, D. Budgen, O. P. Brereton, M. Turner, M. Niazi, and S. Linkman. Systematic literature reviews in software engineering - a tertiary study. *Information and Software Technology*, 52(8):792–805, 2010.

[134] M. Kläs, J. Heidrich, J. Münch, and A. Trendowicz. CQML Scheme: A classification scheme for comprehensive quality model landscapes. In *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*, pages 243–250, Aug 2009. doi: 10.1109/SEAA.2009.88.

[135] M. Kläs, C. Lampasona, S. Nunnenmacher, S. Wagner, M. Herrmannsdörfer, and K. Lochmann. How to evaluate meta-models for software quality? In *Proceedings of the Joined International Conferences IWSM/ MetriKon/ Mensura*, pages 443–462, 2010.

[136] M. Kläs, C. Lampasona, and J. Münch. Adapting software quality models: Practical challenges, approach, and first empirical results. In *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 341–348, Aug 2011. doi: 10.1109/SEAA.2011.62.

[137] E. Knauss and C. E. Boustani. Assessing the quality of software requirements specifications. In *2008 16th IEEE International Requirements Engineering Conference*, pages 341–342, 2008. ISBN 1090-705X. doi: 10.1109/RE.2008.29.

[138] M. Kondo, C.-P. Bezemer, Y. Kamei, A. E. Hassan, and O. Mizuno. The impact of feature reduction techniques on defect prediction models. *Empirical Software Engineering*, 24(4):1925–1963, 2019.

[139] B. M. Konopka, J. C. Nebel, and M. Kotulska. Quality assessment of protein model-structures based on structural and functional similarities. *BMC Bioinformatics*, 13(1), 2012. doi: 10.1186/1471-2105-13-242.

[140] K. Kontogiannis, M. Grigoriou, C. Brealey, and A. Giammaria. Compliance by design: Software analytics and aiops. In *Proceedings of the 31st Annual International Conference on Computer Science and Software Engineering*, CASCON '21, page 294–295, USA, 2021. IBM Corp.

[141] C. Kothapalli, S. G. Ganesh, H. K. Singh, D. V. Radhika, T. Rajaram, K. Ravikanth, S. Gupta, and K. Rao. Continual monitoring of code quality. In *Proceedings of the 4th India Software Engineering Conference 2011, ISEC'11*, pages 175–184, 2011. doi: 10.1145/1953355.1953379.

[142] R. Kumar. *Research Methodology, A Step-by-step Guide for Beginners*. Sage, 2011.

[143] R. Laddad. *AspectJ in Action*. Manning, Second Edition, 2009.

[144] M. Lafi, B. Hawashin, and S. AlZu'bi. Maintenance requests labeling using machine learning classification. In *2020 Seventh International Conference on Software Defined Systems (SDS)*, pages 245–249, 2020. doi: 10.1109/SDS49854.2020.9143895.

[145] Y. C. Lai, C. C. Kao, J. D. Jhan, F. H. Kuo, C. W. Chang, and T. C. Shih. Quality of service measurement and prediction through ai technology. In *2020 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, pages 254–257, 2020. doi: 10.1109/ECICE50847.2020.9302008.

[146] J. Laval, A. Bergel, and S. Ducasse. Assessing the quality of your software with MoQam. Online, [Accessed: 06.03.2018], Oct 17 2008. URL `https://hal.inria.fr/inria-00498482`.

[147] M. Lehman and J. Ramil. Rules and tools for software evolution planning and management. *Annals of Software Engineering, Special Issue on Software Management*, 11(1):15–44, 2001.

[148] M. Lepmets, E. Ras, and A. Renault. A quality measurement framework for IT services. In *Proceedings - 2011 Annual SRII Global Conference, SRII 2011*, pages 767–774, 2011. doi: 10.1109/SRII.2011.84.

[149] L. Lersch, I. Schreter, I. Oukid, and W. Lehner. Enabling low tail latency on multicore key-value stores. *Proc. VLDB Endow.*, 13(7):1091–1104, mar 2020. ISSN 2150-8097. doi: 10.14778/3384345.3384356. URL `https://doi.org/10.14778/3384345.3384356`.

[150] J. Letouzey and T. Coq. The SQALE models for assessing the quality of real time source code. Online, [Accessed: 17.07.2017], September 2010. URL `https://pdfs.semanticscholar.org/4dd3/a72d79eb2f62fe04410106dc9fcc27835ce5.pdf?ga=2.24224186.1861301954.1500303973-1157276278.1497961025`.

[151] J. L. Letouzey. The SQALE method for evaluating technical debt. In *Third International Workshop on Managing Technical Debt (MTD)*, pages 31–36, 2012. doi: 10.1109/MTD.2012.6225997.

[152] J. L. Letouzey. The SQALE method for managing technical debt, definition document v1.1. [Online], [Accessed: 02.08.2017], 03 2016. URL `http://www.sqale.org/wp-content/uploads//08/SQALE-Method-EN-V1-1.pdf`.

[153] J. L. Letouzey. Managing large application portfolio with technical debt related measures. In *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, page 181, 2016. doi: 10.1109/IWSM-Mensura.2016.035.

[154] J. L. Letouzey and T. Coq. The SQALE quality and analysis models for assessing the quality of ada source code. Online, [Accessed: 17.07.2017], September 2009. URL `http://www.adalog.fr/publicat/sqale.pdf`.

[155] J. L. Letouzey and T. Coq. The SQALE analysis model: An analysis model compliant with the representation condition for assessing the quality of software source code. In *2010 Second International Conference on Advances in System Testing and Validation Lifecycle*, pages 43–48, 2010.

[156] J. L. Letouzey and M. Ilkiewicz. Managing technical debt with the SQALE method. *IEEE Software*, 29 (6):44–51, 2012. doi: 10.1109/MS.2012.129.

[157] H. Li, T.-H. P. Chen, W. Shang, and A. E. Hassan. Studying software logging using topic models. *Empirical Softw. Engg.*, 23(5):2655–2694, Oct. 2018. ISSN 1382-3256. doi: 10.1007/s10664-018-9595-8. URL `https://doi.org/10.1007/s10664-018-9595-8`.

[158] H. Li, W. Shang, B. Adams, M. Sayagh, and A. E. Hassan. A qualitative study of the benefits and costs of logging from developers' perspectives. *IEEE Transactions on Software Engineering*, 2020.

[159] J. Li, T. Skramstad, and T. Coq. Interface information management tools for the maritime and oil and gas industry. In *2015 IEEE 39th Annual Computer Software and Applications Conference,*, pages 164–169, 2015. doi: 10.1109/COMPSAC.2015.227.

[160] Y. Li and Z. Man. A fuzzy comprehensive quality evaluation for the digitizing software of ethnic antiquarian resources. In *2008 International Conference on Computer Science and Software Engineering*, volume 5, pages 1271–1274, 2008. doi: 10.1109/CSSE.2008.304.

[161] Z. Li. *Studying and Suggesting Logging Locations in Code Blocks*, page 125–127. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450371223. URL `https://doi.org/10.1145/3377812.3382168`.

[162] Z. Li, T.-H. P. Chen, J. Yang, and W. Shang. Dlfinder: Characterizing and detecting duplicate logging code smells. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, page 152–163. IEEE Press, 2019. doi: 10.1109/ICSE.2019.00032. URL `https://doi.org/10.1109/ICSE.2019.00032`.

[163] Z. Li, T.-H. P. Chen, and W. Shang. *Where Shall We Log? Studying and Suggesting Logging Locations in Code Blocks*, page 361–372. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450367684. URL `https://doi.org/10.1145/3324884.3416636`.

[164] S.-K. Liang and C.-T. Lien. Selecting the optimal ERP software by combining the ISO 9126 standard and fuzzy AHP approach. *Contemporary Management Research*, 3(1):23, Dec 27 2006. doi: 10.7903/cmr.10.

[165] F. Lin, K. Muzumdar, N. P. Laptev, M.-V. Curelea, S. Lee, and S. Sankar. Fast dimensional analysis for root cause investigation in a large-scale service environment. *Proc. ACM Meas. Anal. Comput. Syst.*, 4 (2), jun 2020. doi: 10.1145/3392149. URL `https://doi.org/10.1145/3392149`.

[166] R. Lincke, J. Lundberg, and W. Löwe. Comparing software metrics tools. In *ISSTA'08: Proceedings of the 2008 International Symposium on Software Testing and Analysis 2008*, pages 131–141, 2008. doi: 10.1145/1390630.1390648.

[167] X. Liu, Y. Zhang, X. Yu, and Z. Liu. A software quality quantifying method based on preference and benchmark data. In *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 375–379, June 2018. doi: 10.1109/SNPD.2018.8441145.

[168] Z. Liu, T. Liu, T. Lu, L. Cai, and G. Yang. Agent-based online quality measurement approach in cloud computing environment. In *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, volume 1, pages 686–690, 2010. doi: 10.1109/WI-IAT.2010.213.

[169] R. Madachy and B. Boehm. *Assessing Quality Processes with ODC COQUALMO*, volume 5007 of *Making Globally Distributed Software Development a Success Story*, pages 198–209. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 3540795871. doi: 10.1007/978-3-540-79588-918.

[170] M. Madera and R. Tomoń. A case study on machine learning model for code review expert system in software engineering. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1357–1363, 2017. doi: 10.15439/2017F536.

[171] N. H. Malhotra. *Marketingkutatas (Translated title: Marketing Research)*. Akademia Kiado, 2009.

[172] R. Malhotra and K. Lata. A systematic literature review on empirical studies towards prediction of software maintainability. *Soft Computing*, 24:16655–16677, 2020.

[173] R. A. Martin and L. H. Shafer. Providing a framework for effective software quality assessment—a first step in automating assessments. In *Proceedings of the first annual software engineering and economics conference*, 1996.

[174] S. Martínez-Fernández, A. M. Vollmer, A. Jedlitschka, X. Franch, L. López, P. Ram, P. Rodríguez, S. Aaramaa, A. Bagnato, M. Choraś, and J. Partanen. Continuously assessing and improving software quality with software analytics tools: A case study. *IEEE Access*, 7:68219–68239, 2019. doi: 10.1109/ACCESS.2019.2917403.

[175] MathWorks. Tune fuzzy rules and membership function parameters. [Online], [Accessed: 04.06.2021.]. URL https://de.mathworks.com/help/fuzzy/tune-fuzzy-rules-and-membership-function-parameters.html.

[176] MatWorks. Tunefis options. Online, [Accessed: 25.05.2021], 2019. URL https://de.mathworks.com/help/fuzzy/tunefisoptions.html.

[177] A. Mayr, R. Plösch, and M. Saft. Towards an operational safety standard for software: Modelling iec 61508 part 3. In *Proceedings - 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, ECBS 2011*, pages 97–104, 2011. doi: 10.1109/ECBS.2011.8.

[178] A. Mayr, R. P. osch, and M. Saft. Objective measurement of safety in the context of iec 61508-3. In *Proceedings of the 2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA '13, pages 45–52, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 9780-769550916. doi: 10.1109/SEAA.2013.32. URL http://dx.doi.org/10.1109/SEAA.2013.32.

[179] J. A. McCall, P. K. Richards, and G. F. Walters. Factors in software quality, concept and definitions of software quality. [Online], [Accessed: 06.03.2018], 1977. URL http://www.dtic.mil/dtic/tr/fulltext/u2/a049014.pdf.

[180] Mia Software. Mia Quality. [Online], [Accessed: 16.02.2018], 2017. URL http://www.mia-software.com/produits/mia-quality/.

[181] H. Mittal and P. Bhatia. Software maintainability assessment based on fuzzy logic technique. *ACM SIGSOFT Software Engineering Notes*, Volume 34(3), 2009.

[182] K. Mittal and R. Khan. Performance testing and profiling of web based application in real time. *International Journal of Computer Applications*, 180(46). ISSN 0975-8887.

[183] K. Mordal-Manet, F. Balmas, S. Denier, S. Ducasse, H. Wertz, J. Laval, F. Bellingard, and P. Vaillergues. The Squale model - a practice-based industrial quality model. [Online], [Accessed: 06.03.2018], Nov 25 2009. URL https://hal.inria.fr/inria-00637364.

[184] M. Nagappan, A. Peeler, and M. Vouk. Modeling cloud failure data: A case study of the virtual computing lab. In *Proceedings of the 2Nd International Workshop on Software Engineering for Cloud Computing*, SECLOUD '11, pages 8–14, New York, NY, USA, 2011. ACM. ISBN 9781-450305822. doi: 10.1145/1985500.1985503. URL http://doi.acm.org/10.1145/1985500.1985503.

[185] P. Narman, P. Johnson, and L. Nordstrom. Enterprise architecture: A framework supporting system quality analysis. In *Proceedings of the IEEE International Annual Enterprise Distributed Object Computing Conference EDOC*, page 63–72, 2007.

[186] N. W. Nerurkar, A. Kumar, and P. Shrivastava. Assessment of reusability in aspect-oriented systems using fuzzy logic. *ACM SIGSOFT Software Engineering Notes*, Volume 35(5), 2010.

[187] A. Nugroho, J.Visser, and T. Kuipers. An empirical model of technical debt and interest. Online, [Accessed: 01.09.2017], 2011. URL https://www.sig.eu/files/en/084icsews11mtdfull-p005-nugroho-11.pdf.

[188] U. of Cologne. Methodenpool: Brainstorming. [Online], [Accessed: 16.03.2019]. URL http://methodenpool.uni-koeln.de/brainstorming/framesetbrainstorming.html.

[189] U. of Stuttgart. Quamoco open quality model. Online, [Accessed: 15.12.2018], 2012. URL http://www.quamoco.de/.

[190] U. of Stuttgart. Quamoco tool support for quality modelling and evaluation. Online, [Accessed: 15.12.2018], 2012. URL http://www.quamoco.de/webmodel/home.html.

[191] S. Omri and C. Sinz. Deep learning for software defect prediction: A survey. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ICSEW'20, page 209–214, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379632. doi: 10.1145/3387940.3391463. URL https://doi.org/10.1145/3387940.3391463.

[192] S. Ouhbi, A. Idri, J. L. Fernández-Alemán, A. Toval, and H. Benjelloun. Applying ISO/IEC 25010 on mobile personal health records. In *HEALTHINF 2015 - 8th International Conference on Health Informatics, Proceedings; Part of 8th International Joint Conference on Biomedical Engineering Systems and Technologies, BIOSTEC 2015*, pages 405–412. SciTePress, 2015.

[193] M. B. R. Pandit and N. Varma. A deep introduction to ai based software defect prediction (sdp) and its current challenges. In *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, pages 284–290, 2019. doi: 10.1109/TENCON.2019.8929661.

[194] R. K. Panesar-Walawege, M. Sabetzadeh, L. Briand, and T. T. Coq. Characterizing the chain of evidence for software safety cases: A conceptual model based on the iec 61508 standard. In *2010 Third International Conference on Software Testing, Verification and Validation*, pages 335–344, 2010. doi: 10.1109/ICST.2010.12.

[195] I. Park and R. Buch. Improve debugging and performance tuning with ETW. [Online], [Accessed: 06.03.2020], 2007. URL https://docs.microsoft.com/en-us/archive/msdn-magazine/2007/april/event-tracing-improve-debugging-and-performance-tuning-with-etw.

[196] S. Parthasarathy and S. Sharma. Impact of customization over software quality in ERP projects: an empirical study. *Software Quality Journal*, 25(2):581–598, 2017. doi: 10.1007/s11219-016-9314-x.

[197] C. M. Patino and J. C. Ferreira. Internal and external validity: can you apply research study results to your patients? *J Bras Pneumol.*, 44(3):183, May-June 2018. doi: 10.1590/S1806-37562018000000164.

[198] W. E. Perry. *Quality Assurance for Information Systems: Method, Tools and Techniques*. John Wiley and Sons, 1991.

[199] K. Pitula. *On Requirements Elicitation for Software Projects in Ict for Development*. PhD thesis, 2010. AAINR71106.

[200] R. Plösch, H. Gruber, A. Hentschel, C. Körner, G. Pomberger, S. Schiffer, M. Saft, and S. Storck. The EMISQ method and its tool support-expert-based evaluation of internal software quality. *Innovations in Systems and Software Engineering*, 4(1):3–15, Apr 2008. doi: 10.1007/s11334-007-0039-7.

[201] R. Plösch, H. Gruber, C. Körner, and M. Saft. A method for continuous code quality management using static analysis. In *2010 Seventh International Conference on the Quality of Information and Communications Technology*, pages 370–375, Sept 2010. doi: 10.1109/QUATIC.2010.68.

[202] R. Poli, W. B. Langdon, and N. McPhee. *A Field Guide to Genetic Programming*. lulu.com: http://www.gp-field-guide.org.uk, England, 2008. ISBN 978-1-4092-0073-4.

[203] S. Pradhan, V. Nanniyur, and P. K. Vissapragada. On the defect prediction for large scale software systems – from defect density to machine learning. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, pages 374–381, 2020. doi: 10.1109/QRS51102.2020.00056.

[204] Quamoco Consortium. Quamoco quality model explorer. [Online], [Accessed: 28.02.2018], 2012. URL `http://www.quamoco.de/webmodel/explorer.html`.

[205] A. A. Rahman, S. Sahibuddin, and S. Ibrahim. A unified framework for software engineering process improvement - a taxonomy comparative analysis. In *2011 5th Malaysian Conference in Software Engineering, MySEC 2011*, pages 153–158, 2011. doi: 10.1109/MySEC.2011.6140661.

[206] A. A. Rahman, S. Sahibuddin, and S. Ibrahim. A taxonomy analysis for multi-model process improvement from the context of software engineering processes and services. *International Journal of Digital Content Technology and its Applications*, 6(22):56–65, 2012. doi: 10.4156/jdcta.vol6.issue22.6.

[207] J. Rand and A. Miranskyy. On automatic parsing of log records. In *Proceedings of the 43rd International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '21, page 41–45. IEEE Press, 2021. ISBN 9780738133249. doi: 10.1109/ICSE-NIER52604.2021.00017. URL `https://doi.org/10.1109/ICSE-NIER52604.2021.00017`.

[208] T. Ross. *Fuzzy Logic with Engineering Applications*. Wiley, . 2010. ISBN 978-0-470-74376-8.

[209] T. L. Saaty and J. M. Katz. How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, pages 9–26, 1990.

[210] J. Saldana. *The Coding Manual for Qualitative Researchers*. Sage, 2009.

[211] N. J. Salkind. *Exploring Research*. Pearson, Prentice-Hall, 2009. ISBN 978-0-13-601137-8.

[212] M. Schmid, M. Thoss, T. Termin, and R. Kroeger. A generic application-oriented performance instrumentation for multi-tier environments. In *IEEE, 10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 304–313, May 2007. ISBN 1-4244-0799-0.

[213] M. Schmid, T. Stein, M. Thoss, and R. Kroeger. An eclipse ide extension for pattern-based software instrumentation. In *14th GI/ITG Conference - Measurement, Modelling and Evalutation of Computer and Communication Systems*, pages 1–3, 2008.

[214] Security Reviewer Srl. Security Reviewer. [Online], [Accessed: 16.02.2018], 2017. URL `http://www.securityreviewer.net/`.

[215] A. Seffah, N. Kececi, and M. Donyaee. QUIM: a framework for quantifying usability metrics in software quality models. In *Proceedings Second Asia-Pacific Conference on Quality Software*, pages 311–318, 2001. doi: 10.1109/APAQS.2001.990036.

[216] W. Shang, M. Nagappan, A. E. Hassan, and Z. M. Jiang. Understanding log lines using development knowledge. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 21–30, 2014.

[217] W. Shang, M. Nagappan, and A. E. Hassan. Studying the relationship between logging characteristics and the code quality of platform software. *Empirical Softw. Eng.*, 20(1):1–27, Feb. 2015. ISSN 1382-3256. doi: 10.1007/s10664-013-9274-8. URL `https://doi.org/10.1007/s10664-013-9274-8`.

[218] R. Shatnawi and W. Li. An empirical assessment of refactoring impact on software quality using a hierarchical quality model. *International Journal of Software Engineering and its Applications*, 5(4): 127–150, 2011.

[219] P. Shen, X. Ding, W. Ren, and C. Yang. Research on software quality assurance based on software quality standards and technology management. In *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 385–390, June 2018. doi: 10.1109/SNPD.2018.8441142.

[220] B. R. Shubhamangala, V. Suma, P. A. Reddy, and C. G. Singh. Quality attribute focused multilayer requirement elicitation: Judicious approach to drive business value. In *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2069–2075, 2013. doi: 10.1109/ICACCI.2013.6637500. ID: 10.

[221] P. K. Singh, O. P. Sangwan, A. P. Singh, and A. Pratap. A framework for assessing the software reusability using fuzzy logic approach for aspect oriented software. *IJ Information Technology and Computer Science*, 7(2):12–20, 2015.

[222] SonarSource. SonarQube. [Online], [Accessed: 16.02.2018], 2017. URL `https://www.sonarqube.org`.

[223] S. Sproge. Evaluation of study programme external quality. *Research for Rural Development 2011, Vol 1*, pages 179–185, 2011.

[224] S. Sproge and R. Cevere. Assessment of study programme quality at higher education institution. *Engineering for Rural Development - International Scientific Conference*, 11:663, 2012.

[225] SQUALE. Final version of the spreadsheet tool used to estimate the return of investment of a global quality process [in french], workpackage: 2.3. [Online], September 2010, [Accessed: 16.11.2017], 2010. URL `http://www.squale.org/quality-models-site/research-deliverables/WP2.3ROI-estimationv1.xls`.

[226] Squoring Technologies SAS. Squoring. [Online], [Accessed: 16.02.2018], 2017. URL `https://www.squoring.com/`.

[227] M. Staron, W. Meding, K. Niesel, and A. Abran. A key performance indicator quality model and its industrial evaluation. In *2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, pages 170–179, 2016. doi: 10.1109/IWSM-Mensura.2016.033.

[228] R. M. Szabo and T. M. Khoshgoftaar. An assessment of software quality in a c++ environment. In *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on*, pages 240–249, 1995. ISBN 1071-9458. doi: 10.1109/ISSRE.1995.497663.

[229] B. Tekinerdogan, N. Ali, J. Grundy, I. Mistrik, and R. Soley. *Quality concerns in large-scale and complex software-intensive systems*, pages 1–17. Software Quality Assurance: In Large Scale and Complex Software-intensive Systems. Elsevier Inc., 2015. doi: 10.1016/B978-0-12-802301-3.00001-6.

[230] N. Terada, E. Kawai, and H. Sunahara. Extracting client-side streaming qos information from server logs. In *IEEE Pacific Rim Conference on Communications, Computers and signal Processing*, pages 621–624, 2005. ISBN 1555-5798. doi: 10.1109/PACRIM.2005.1517366.

[231] H. Thane. Monitoring, testing, and debugging of distributed real-time system. PhD Thesis, Royal Institute of Technology, Stockholm, [Online], [Accessed: 24.05.2018], 2002. URL `http://www.mrtc.mdh.se/publications/0242.pdf`.

[232] S. Thorpe, I. Ray, T. Grandison, and A. Barbir. Cloud log forensics metadata analysis. In *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*, pages 194–199, 2012. doi: 10.1109/COMPSACW.2012.44.

[233] D. Tovarnak and T. Pitner. Continuous queries over distributed streams of heterogeneous monitoring data in cloud datacenters. In *Proceedings of the 9th International Conference on Software Engineering and Applications*, Aug 2014.

[234] N. Ubayashi, Y. Kamei, and R. Sato. When and why do software developers face uncertainty? In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, pages 288–299, July 2019. doi: 10.1109/QRS.2019.00045.

[235] M. Ulan, S. Hönel, R. M. Martins, M. Ericsson, W. Löwe, A. Wingkvist, and A. Kerren. Quality models inside out: Interactive visualization of software metrics by means of joint probabilities. In *2018 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 65–75, Sep. 2018. doi: 10.1109/VISSOFT.2018.00015.

[236] G. Ursachi, I. A. Horodnic, and A. Zait. How reliable are measurement scales? external factors with indirect influence on reliability estimators. In *7th International Conference on Globalization and Higher Education in Economics and Business Administration, GEBA 2013*, Procedia Economics and Finance 20, pages 679–686, Romania, 2015. Elsevier.

[237] V. Uzelac, A. Milenkovic, M. Burtscher, and M. Milenkovic. Real-time unobtrusive program execution trace compression using branch predictor events. *CASES 2010 Proceedings of the 2010 international conference on Compilers, Architectures and Synthesis for Embedded Systems, ISBN: 978-1-60558-903-9*, 2010.

[238] R. Vaculin and K. Sycara. Semantic web services monitoring: An OWL-S based approach. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, page 313, 2008. ISBN 1530-1605. doi: 10.1109/HICSS.2008.385.

[239] R. van Solingen and E. Berghout. *The Goal/Question/Metric Method a practical guide for quality improvement of software development*. McGraw Hill Publishing, England, 1999. ISBN 007 709553 7.

[240] A. Vetro, N. Zazworka, C. Seaman, and F. Shull. Using the iso/iec 9126 product quality model to classify defects: A controlled experiment. In *16th International Conference on Evaluation Assessment in Software Engineering (EASE 2012)*, pages 187–196, May 2012. doi: 10.1049/ic.2012.0025.

[241] G. Villasana and R. Castello. An agile software quality framework lacking. In *2014 World Congress on Computer Applications and Information Systems, WCCAIS 2014*. Institute of Electrical and Electronics Engineers Inc., 2014. doi: 10.1109/WCCAIS.2014.6916549.

[242] S. Wagner, K. Lochmann, L. Heinemann, M. K. as, A. Trendowicz, R. Plösch, A. Seidl, A. Goeb, and J. Streit. The quamoco product quality modelling and assessment approach. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 1133–1142, Piscataway, NJ, USA, 2012. IEEE Press. ISBN 9781-467310673.

[243] S. Wagner, K. Lochmann, S. Winter, F. Deissenboeck, E. Juergens, M. Herrmannsdoerfer, L. Heinemann, M. Kläs, A. Trendowicz, J. Heidrich, R. Plösch, A. Goeb, C. Koerner, K. Schoder, J. Streit, and C. Schubert. The quamoco quality meta-model. [Online], October, 2012, [Accessed: 18.11.2017], 2012. URL `https://mediatum.ub.tum.de/attfile/1110600/hd2/incoming/2012-Jul/517198.pdf`.

[244] S. Wagner, A. Goeb, L. Heinemann, M. Kläs, C. Lampasona, K. Lochmann, A. Mayr, R. Plösch, A. Seidl, J. Streit, and A. Trendowicz. Operationalised product quality models and assessment: The quamoco approach. *Information and Software Technology*, 62:101–123, Jun 2015. doi: 10.1016/j.infsof.2015.02.009.

[245] N. Walkinshaw. Using evidential reasoning to make qualified predictions of software quality. In *Proceedings of the 9th International Conference on Predictive Models in Software Engineering*, PROMISE '13, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320160. doi: 10.1145/2499393.2499402. URL `https://doi.org/10.1145/2499393.2499402`.

[246] F. Xing, P. Guo, and M. R. Lyu. A novel method for early software quality prediction based on support vector machine. In *16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05)*. IEEE, 2005.

[247] F. Xing, P. Guo, and M. R. Lyu. A novel method for early software quality prediction based on support vector machine. In *16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05)*, pages 10–pp. IEEE, 2005.

[248] Z. Xu. *Fuzzy Logic Techniques for Software Reliability Engineering*. PhD thesis, 2001. AAI3001234.

[249] J.-B. Yang and D.-L. Xu. On the evidential reasoning algorithm for multiple attribute decision analysis under uncertainty. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 32(3):289–304, 2002. doi: 10.1109/TSMCA.2002.802746.

[250] K. Yao, G. B. de Pádua, W. Shang, C. Sporea, A. Toma, and S. Sajedi. Log4perf: suggesting and updating logging locations for web-based systems' performance monitoring. *Empirical Software Engineering*, 25 (1):488–531, 2020.

[251] D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage. Improving software diagnosability via log enhancement. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, page 3–14, New York, NY, USA, 2011.

Association for Computing Machinery. ISBN 9781450302661. doi: 10.1145/1950365.1950369. URL https://doi.org/10.1145/1950365.1950369.

[252] D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, and S. Savage. Be conservative: Enhancing failure diagnosis with proactive logging. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 293–306, Hollywood, CA, Oct. 2012. USENIX Association. ISBN 978-1-931971-96-6. URL https://www.usenix.org/conference/osdi12/technical-sessions/presentation/yuan.

[253] D. Yuan, S. Park, and Y. Zhou. Characterizing logging practices in open-source software. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, page 102–112. IEEE Press, 2012. ISBN 9781467310673.

[254] D. Yuniasri, T. Badriyah, and U. Sa'adah. A comparative analysis of quality page object and screenplay design pattern on web-based automation testing. In *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, pages 1–5, June 2020. doi: 10.1109/ICECCE49384.2020.9179470.

[255] L. Zadeh. Fuzzy sets. *Information and Control*, pages 338–353, 1965.

[256] L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning—i. *Information sciences*, 8(3):199–249, 1975.

[257] L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning-ii. *Information Sciences*, pages 301–357, 1975.

[258] L. A. Zadeh. Fuzzy logic = computing with words. *IEEE Transactions on Fuzzy Systems*, 4.(2.):103–111, 1996.

[259] L. A. Zadeh. Is there a need for fuzzy logic? Technical report, Annual Meeting of the North American Fuzzy Information Processing Society, 2008.

[260] Zen Program Ltd. NDepend. [Online], [Accessed: 16.02.2018], 2017. URL https://www.ndepend.com/.

[261] Y. Zeng, J. Chen, W. Shang, and T.-H. Chen. Studying the characteristics of logging practices in mobile apps: a case study on f-droid. *Empirical Software Engineering*, 24(6):3394–3434, 2019. doi: 10.1007/s10664-019-09687-9.

[262] L. Zhang, L. Li, and H. Gao. 2-D software quality model and case study in software flexibility research. In *Proceedings of the 2008 International Conference on Computational Intelligence for Modelling Control and Automation*, CIMCA '08, pages 1147–1152, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 9780-769535142. doi: 10.1109/CIMCA.2008.70.

[263] X. Zhao, K. Rodrigues, Y. Luo, M. Stumm, D. Yuan, and Y. Zhou. Log20: Fully automated optimal placement of log printing statements under specified overhead threshold. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 565–581, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350853. doi: 10.1145/3132747.3132778. URL https://doi.org/10.1145/3132747.3132778.

[264] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang. Learning to log: Helping developers make informed logging decisions. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, page 415–425. IEEE Press, 2015. ISBN 9781479919345.

Part I

Appendices

The appendices comprise detailed material about the systematic literature review, the focus groups for identifying the quality properties of execution tracing, the analysis of the academic literature to extract the quality properties of execution tracing, the data coding [210], and the modelling process. Moreover, verbatim content from three of the author's journal publications [65, 68, 69] are also used to support the claims, and conclusions drawn while conducting the research. Nevertheless, to shorten the thesis, the questionnaire and the collected data at the model construction stage reported in [68] were published as a separate data article, which is freely available in [67]. Each appendix provides further information on the dully cited sources.

Appendix A

Systematic Literature Review

The process of the performed systematic literature review [130] the author published in detail in [65]. The tables in this chapter stem from [65] and are necessary to highlight the relevance of the software product quality models identified.

A.1 Exact Queries in the Different Scientific Databases

Processing the publications returned by the queries and extracting the relevant pieces of information required considerable effort and lasted nearly two years. Consequently, the document search was repeated in the two most relevant computer science archives: ACM and IEEE after this period to extend the analysis with the publications which appeared while the research was being conducted. The author documented this process in detail in [65].

A.1.1 RQ1a Query

Logical Query: "Software Product Quality Model" OR "Software Product Quality Framework"

Concrete Implementation in the Specific Databases:

1. ACM

   Search term: "Software Product Quality Model" OR "Software Product Quality Framework"

   Date: 2000-2017

2. EBSCO

   Search term: "Software Product Quality Model" OR "Software Product Quality Framework"

   Searching: Academic Search Premier DB

   Journal type: Peer-reviewed

   Date: 2000-2017

3. IEEE

Search term: "Software Product Quality Model" OR "Software Product Quality Framework"

Date: 2000-2017

4. Science Direct

   Search term: "Software Product Quality Model" OR "Software Product Quality Framework"

   Date: 2000-2017

5. Scopus

   Search term: "Software Product Quality Model" OR "Software Product Quality Framework"

   Metadata: title, keyword, abstract

   Date: 2000-2017

6. Web of Science

   Search term: "Software Product Quality Model" OR "Software Product Quality Framework"

   Date: 2000-2017

   Databases: (1) Science Citation Index Expanded (SCI-EXPANDED) –1970-present, (2) Conference Proceedings Citation Index- Science (CPCI-S) –1990-present

A.1.2 RQ1b Query

Logical Query: ("quality model" OR "quality framework") AND assessment AND software AND analysis AND (measure OR measurement)

Concrete Query Implementations in Specific Document Databases:

1. ACM

   Search term: recordAbstract:(+("execution tracing" logging) +quality +maintainability +software +(model framework))

   Date: 2000-2017

2. EBSCO

   Search term: ("execution tracing" OR logging) AND quality AND maintainability AND software AND (model OR framework)

   Searching: Academic Search Premier DB

   Journal type: Peer-reviewed

   Date: 2000-2017

3. IEEE

   Search term: (("quality model" OR "quality framework") AND assessment AND software AND analysis AND (measure OR measurement))

   Search Type: Abstract search

   Date: 2000-2017

4. Science Direct

Search term: pub-date >1999 and title-abstr-key((“quality model” OR “quality framework”) AND assessment AND software AND analysis AND (measure OR measurement))

Subject: Computer Science

Date: 2000-2017

5. Scopus

Search term: (“quality model” OR “quality framework”) AND assessment AND software AND analysis AND (measure OR measurement)

Metadata: title, keyword, abstract

Date: 2000-2017

6. Web of Science

Search term: TS=((“quality model” OR “quality framework”) AND assessment AND software AND analysis AND (measure OR measurement))

Date: 2000-2017

Databases: (1) Science Citation Index Expanded (SCI-EXPANDED) –1970-present, (2) Conference Proceedings Citation Index- Science (CPCI-S) –1990-present

A.2 Raw List of Software Product Quality Model Publications Identified

The table below, published by the author in [65], contains 125 sources identified in the scope of the systematic literature review. The duplicates are removed. Each publication has been analysed. The decision, whether a publication is included in the detailed analysis or excluded, is also documented with providing the reason for the decision made. The list is alphabetically sorted by the domain descending, and by author and title in ascending order.

List of Abbreviations:

SPQM: Software Product Quality Model

Table A.1: List of Software Product Quality Model Publications Identified, Source: [65]

No.	Domain	Excluded	Reason for Exclusion	Ref.
1	Software Product Quality Model	yes	Not an SPQM	[28]
2	Software Product Quality Model	yes	Language Spanish	[40]
3	Software Product Quality Model	yes	Book, secondary source	[54]
4	Software Product Quality Model			[59]
5	Software Product Quality Model	yes	Process model with concepts from ISO9126	[82]
6	Software Product Quality Model	yes	Process model with concepts from ISO9126	[84]
7	Software Product Quality Model	yes	Not a model	[85]
8	Software Product Quality Model	yes	Book, secondary source	[95]
9	Software Product Quality Model			[98]
10	Software Product Quality Model			[120]
11	Software Product Quality Model	yes	Process model with some concepts taken over from ISO25010	[148]

Table A.1: List of Software Product Quality Model Publications Identified, Source: [65]

No.	Domain	Excluded	Reason for Exclusion	Ref.
12	Software Product Quality Model			[192]
13	Software Product Quality Model			[196]
14	Software Product Quality Model	yes	Process model	[206]
15	Software Product Quality Model	yes	Process model	[205]
16	Software Product Quality Model	yes	Process model with concepts from ISO9126	[223]
17	Software Product Quality Model	yes	Process model with concepts from ISO9126	[224]
18	Software Product Quality Model	yes	Book, secondary source	[229]
19	Software Product Quality Model			[240]
20	Software Product Quality Model	yes	Process model	[241]
21	Software Product Quality Model	yes	It is full conference proceedings volume with the abstract of the single papers' titles listed	[2]
22	Quality Model Assessment	yes	Safety standard not software product quality model	[178]
23	Quality Model Assessment	yes	Safety standard conformance, not software product quality model related	[177]
24	Quality Model Assessment	yes	Model for useability metrics not a full SPQM	[215]
25	Quality Model Assessment	yes	Not a software quality model	[7]
26	Quality Model Assessment	yes	Process model	[220]
27	Quality Model Assessment	yes	Information quality for software project management not software product quality	[23]
28	Quality Model Assessment	yes	Book, secondary source	[22]
29	Quality Model Assessment			[44]
30	Quality Model Assessment	yes	Software requirement specification not software product quality model	[137]
31	Quality Model Assessment			[42]
32	Quality Model Assessment	yes	Information quality frameworks not software product quality models	[57]
33	Quality Model Assessment			[58]
34	Quality Model Assessment			[73]
35	Quality Model Assessment	yes	Not software quality related	[81]
36	Quality Model Assessment	yes	Not a quality model but a description on the evolution of the quality models	[75]
37	Quality Model Assessment			[155]
38	Quality Model Assessment	yes	It is maintainability model not a complete SPQM	[37]
39	Quality Model Assessment	yes	Not computer scienece related	[139]
40	Quality Model Assessment			[141]
41	Quality Model Assessment			[20]
42	Quality Model Assessment			[262]
43	Quality Model Assessment	yes	It is a maintainability model, not a complete SPQM	[166]
44	Quality Model Assessment	yes	Not an SPQM	[52]
45	Quality Model Assessment	yes	Quality model for KPIs not for software product quality	[227]
46	Quality Model Assessment	yes	Not software product quality model related	[199]
47	Quality Model Assessment	yes	publication year: 1995	[83]

Table A.1: List of Software Product Quality Model Publications Identified, Source: [65]

No.	Domain	Excluded	Reason for Exclusion	Ref.
48	Quality Model Assessment	yes	It is not a software product quality model but a classification of the C++ source code modules in three risk groups: high, medium and low based on quality metrics. -publication year 1995	[228]
49	Quality Model Assessment			[242]
50	Quality Model Assessment	yes	Not an SPQM	[218]
51	Quality Model Assessment	yes	Not an SPQM	[126]
52	Quality Model Assessment			[94]
53	Quality Model Assessment	yes	The dissertation could not be obtained but newer research has been identified using similar methods.	[248]
54	Quality Model Assessment			[160]
55	Quality Model Assessment	yes	Not an SPQM	[168]
56	Quality Model Assessment	yes	Full conference proceeding	[1]
57	Quality Model Assessment	yes	Full conference proceeding	[4]
58	Quality Model Assessment	yes	Full conference proceeding	[3]
59	Manual Search			[99]
60	Manual Search			[97]
61	Manual Search			[10]
62	Manual Search	yes	It is a cost model	[13]
63	Manual Search			[100]
64	Manual Search			[26]
65	Manual Search	yes	Not an SPQM	[131]
66	Manual Search			[27]
67	Manual Search	yes	Not a complete SPQM	[14]
68	Manual Search	yes	Not a complete SPQM	[15]
69	Manual Search			[128]
70	Manual Search	yes	Process model	[35]
71	Manual Search			[71]
72	Manual Search	yes	Metric definitions, therefore they are excluded as separate metric definition are also excluded for other SPQMs.	[17]
74	Manual Search	yes	QM classification not a SPQM	[41]
75	Manual Search	yes	Safety standards related	[53]
76	Manual Search			[93]
77	Manual Search			[80]
78	Manual Search	yes	Not a complete SPQM, it describes maintainability only	[89]
79	Manual Search	yes	Not an SPQM	[90]
80	Manual Search			[91]
81	Manual Search			[96]
82	Manual Search	yes	Model for maintainability not a complete SPQM	[92]
83	Manual Search			[112]
84	Manual Search			[106]
85	Manual Search	yes	Not a model description but definition of metrics and measurement.	[103]

Table A.1: List of Software Product Quality Model Publications Identified, Source: [65]

No.	Domain	Excluded	Reason for Exclusion	Ref.
86	Manual Search	yes	Not a model description but definition of metrics and measurement.	[113]
87	Manual Search	yes	Not a model description but definition of metrics and measurement.	[101]
88	Manual Search	yes	Not a model description but definition of metrics and measurement.	[102]
89	Manual Search			[179]
90	Manual Search			[18]
91	Manual Search			[153]
92	Manual Search			[146]
93	Manual Search			[36]
94	Manual Search			[150]
95	Manual Search			[154]
96	Manual Search			[183]
97	Manual Search			[132]
98	Manual Search	yes	Not an SPQM.	[135]
99	Manual Search			[151]
100	Manual Search			[152]
101	Manual Search			[156]
102	Manual Search	yes	Not an SPQM.	[159]
103	Manual Search			[38]
104	Manual Search	yes	Not an SPQM.	[136]
105	Manual Search	yes	Not a model	[147]
106	Manual Search	yes	Comparison of models not a model description.	[39]
107	Manual Search			[173]
108	Manual Search	yes	Not a software product quality model but a model for enterprise architecture models	[185]
109	Manual Search	yes	Requirement traceability based on ISO/IEC 61508 to deal with safety. Not an SPQM.	[194]
110	Manual Search	yes	Book published in 1991	[198]
111	Manual Search			[49]
112	Manual Search			[189]
113	Manual Search	yes	Not a model description but a tool.	[190]
114	Manual Search			[79]
115	Manual Search			[45]
116	Manual Search			[169]
117	Manual Search			[200]
118	Manual Search			[239]
119	Manual Search	yes	Secondary source	[54]
120	Manual Search	yes	Not a model description but a tool.	[46]
121	Manual Search			[124]
123	Manual Search	yes	Not a model description but a tool. Text is in French.	[225]
124	Manual Search			[244]
125	Manual Search			[243]

A.3 Evaluation of the Publications on Software Product Quality Models

The table below, published by the author in [65], lists all of the 56 publications that define software product quality models or show adaptations of the identified models, based on the systematic literature review. Each identified publication is evaluated and a score value for clarity, actuality and relevance is assigned. The computation of the score values is documented in depth in [65]. The list is alphabetically sorted by (1) the search domain descending, (2) by the software product quality model family ascending, and (3) by the assigned relevance score descending.

Abbreviations:

Act.	Actuality
Clar.	Presentation Clarity
Domain	Search Domain
ETP	Execution Tracing Present?
MS	Manual Search
QMA	The results produced by the query RQ1b
Ref.	Reference
Rel.	Relevance
SPQM	The results produced by the query RQ1a

Table A.2: Software Product Quality Model Publications Scored and Sorted by Relevance by Search Domain, Source: [65]

Rel.	Act.	Clar.	Domain	Model Class	ETP	Model	Ref.
25	5	5	SPQM	ISO25010	no	Application of ISO25010	[98]
15	5	3	SPQM	ISO25010	no	Application of ISO25010	[192]
25	5	5	SPQM	ISO9126	no	Application of ISO9126	[196]
20	4	5	SPQM	ISO9126	no	ISO9126-application, research of ISO9126	[240]
15	3	5	SPQM	ISO9126	no	ISO9126 with clsutering and AHP for quality visualisation	[120]
0	4	0	SPQM	Metrics Framework for Mobile Apps	no	metrics framework	[59]
25	5	5	QMA	Quamoco	no	Quamoco	[73]
20	4	5	QMA	EMISQ	no	CQMM (process model for fitting EMISQ in the development process)	[141]

Table A.2: Software Product Quality Model Publications Scored and Sorted by Relevance by Search Domain, Source: [65]

Rel.	Act.	Clar.	Domain	Model Class	ETP	Model	Ref.
20	4	5	QMA	ISO25010	no	MDWE, ISO25010-adaptation, hybrid model	[44]
20	4	5	QMA	Quamoco	no	Quamoco	[242]
12	4	3	QMA	ISO9126	no	ISO9126-implementation in model driven development	[94]
10	5	2	QMA	ISO25010	no	ASQuS , Application ISO-25010	[58]
9	3	3	QMA	2D Model	no	2D Model	[262]
9	3	3	QMA	SQALE	no	SQALE	[155]
3	3	1	QMA	ISO9126	no	Fuzzy, Antiquarian Resource Digit Software, ISO9126	[160]
1	1	1	QMA	McCall et al.	no	QM for Java Agents with Classification	[20]
25	5	5	MS	Quamoco	no	Quamoco	[244]
25	5	5	MS	SQALE	no	SQALE	[152]
20	4	5	MS	ISO25010	yes, in part.	ISO25010	[112]
20	4	5	MS	Quamoco	no	Quamoco	[243]
20	4	5	MS	SQALE	no	SQALE	[91]
15	3	5	MS	COQUALMO	no	Constructive QUALity Model (COQUALMO)	[169]
15	5	3	MS	ISO25010	no	Application of ISO25010	[97]
15	3	5	MS	ISO9126	no	ISO9126 application	[36]
15	3	5	MS	ISO9126	no	ISO9126-application (tailored to commercial off-the-shelf (COTS) software product and software component evaluation)	[10]
15	3	5	MS	SQALE	no	SQALE	[154]
15	3	5	MS	SQUALE	yes, in part.	SQUALE	[16]
15	3	5	MS	SQUALE	no	SQUALE	[100]
12	3	4	MS	ADEQUATE	no	ADEQUATE (prod, proc.)	[93]
12	4	3	MS	SQALE	no	SQALE	[151]
12	4	3	MS	SQALE	no	SQALE	[156]
10	2	5	MS	FURPS		FURPS+	[48]
10	2	5	MS	ISO9126	no	ISO9126-Fuzzy-AHP	[164]
10	2	5	MS	SQAE and ISO9126 combination	no	SQAE-ISO9126	[38]
9	3	3	MS	EMISQ	no	EMISQ	[200]
9	3	3	MS	EMISQ	no	CQMM (process model for fitting EMISQ in the development process)	[201]
9	3	3	MS	Kim and Lee	no	Kim and Lee	[128]
9	3	3	MS	SQALE	no	SQALE	[150]
6	2	3	MS	ADEQUATE	no	ADEQUATE (prod, proc.)	[124]
5	1	5	MS	GEQUAMO	no	GEQUAMO, ISO9126 derivate	[71]
5	1	5	MS	ISO9126	yes, in part.	ISO9126	[106]

Table A.2: Software Product Quality Model Publications Scored and Sorted by Relevance by Search Domain, Source: [65]

Rel.	Act.	Clar.	Domain	Model Class	ETP	Model	Ref.
5	1	5	MS	QMOOD	no	QMOOD	[18]
5	5	1	MS	SQALE	no	SQALE	[153]
3	3	1	MS	SQUALE	no	SQUALE	[183]
3	3	1	MS	SQUALE	no	Qualixo	[146]
0	0	5	MS	Boehm et al.	no	Boehm et al.	[27]
0	0	5	MS	COQUALMO	no	COnstructive QUALity Model (COQUALMO)	[26]
0	0	5	MS	Dromey	no	Dromey	[45]
0	0	5	MS	FURPS		FURPS	[78]
0	0	5	MS	FURPS	no	FURPS+	[77]
0	0	5	MS	GQM	no	Goal Question Metric approach (hybrid model)	[239]
0	3	0	MS	IEEE Metrics Framework	no	IEEE Metrics Framework	[99]
0	0	5	MS	McCall et al.	no	McCall et al.	[179]
0	0	1	MS	SATC	no	SATC, hybrid	[96]
0	0	5	MS	SQAE	no	SQAE	[173]
0	0	5	MS	SQUID	no	SQUID	[132]

Appendix B

Identifying the Quality Properties of Execution Tracing

The author published a significant part of this chapter in [69] to introduce the data collection, the collected data; furthermore, to describe the data coding and quality property construction process. The quality properties are derived from the outcome of the focus groups and from the quality property candidates of the academic literature.

B.1 Similarities to Execution Tracing in the Identified Software Product Quality Models

As [69] states, none of the 23 identified software product quality model families addresses the quality of execution tracing in an adequate manner. In this section, a brief summary is given about the entities in the identified software product quality model families which are similar to or show overlap with execution tracing quality. In addition, homonyms are also included which do not address execution tracing or software logging but use these terms in their definition or naming for different purposes.

B.1.1 SQUALE

Explicitly only the SQUALE model [16] considers the quality of execution tracing but only with regard to the severity levels. SQUALE [16] names and defines a practice: Tracing Standard to assess whether log messages are appropriately traced on three levels: (1) errors, (2) warnings and (3) infos. In the concept and terminology of SQUALE, practices represent an intermediary level in the quality model hierarchy between sub-characteristics and attributes. The practice Tracing Standard does not identify the quality properties on which execution tracing quality depends; moreover, the definition leaves room for many interpretations including what needs to count as error, as warning, and as info level.

B.1.2 ISO/IEC Standard Families

The ISO/IEC 9126 [106] and ISO/IEC 25010 [112] software product quality models do not consider the quality of execution tracing but define metrics and measures which show similarities or overlap with execution tracing quality; however, they are different [113]. Below a brief summary is provided about these similarities in detail with the references to corresponding the standards.

Similarities or Overlaps

1. ISO/IEC 25010 Standard Family, Characteristic Security, Accountability measures, Section 8.7.4. in [113]:

   "Accountability measures are used to assess the degree to which the actions of an entity can be traced uniquely to the entity."

   (a) User audit trail completeness (ID: SAc-1-G):

   Definition: "How complete is the audit trail concerning the user access to the system or data?"

   Measurement Function: $X = A/B$

   A: Number of access recorded in all logs.

   B: Number of access to system or data actually tested.

   (b) System log retention (ID: SAc-2-S):

   Definition: "For what percent of the required system retention period is the system log retained in a stable storage?"

   Measurement function: $X = A/B$

   A: Duration for which the system log is actually retained in stable storage.

   B: Retention period specified for keeping the system log in a stable storage.

   Stable storage: "A stable storage is a classification of computer data storage technology that guarantees atomicity for any given write operation and allows software to be written that is robust against some hardware and power failures. Most often, stable storage functionality is achieved by mirroring data on separate disks via RAID technology."

2. ISO/IEC 25010 Standard Family, Characteristic Maintainability, Analysability Measures, Section 8.8.3. in [113]:

   "Analysability measures are used to assess the degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or cause of failure or to identify parts to be modified."

   (a) System log completeness (ID: MAn-1-G):

   Definition: "To what extent does the system record its operations in logs so that they are to be traceable?"

   Measurement function: $X = A/B$

   A: Number of logs that are actually recorded in the system.

   B: Number of logs for which audit trails are required during operation.

(b) Diagnosis function effectiveness (ID: MAn-2-S):

Definition: "What proportion of diagnosis functions meets the requirements of casual analysis?"

Measurement function: $X = A/B$

A: Number of diagnostic functions used for casual analysis.

B: Number of diagnostic functions implemented.

(c) Diagnosis function sufficiency (ID: MAn-3-S):

Definition: "What proportion of the required diagnosis functions has been implemented?"

Measurement function: $X = A/B$

A: Number of diagnostic functions implemented.

B: Number of diagnostic functions required.

3. ISO/IEC 9126 Standard Family, Characteristic Maintainability, Analysability Metrics, in [107, 108]:

(a) Audit trail capability

Purpose: "Can the user identify specific operations which caused failure? Can the maintainer easily find specific operations which caused failure?"

Measurement function: $X = A/B$

A: Number of data actually recorded during operation;

B: Number of data planned to be recorded, which is enough to monitor status of the software in operation

Method of application: "Observing the behaviour of users or maintainers who are trying to resolve failures."

(b) Diagnostic function support

Purpose: "How capable are the diagnostic functions to support causal analysis? Can the user identify the specific operation which caused failure? (The user may be able to avoid encountering the same failure with alternative operations.) Can the maintainer easily find the cause of failure?"

Measurement function: $X = A/B$

A: Number of failures which maintainers can diagnose using diagnostic functions to understand cause-effect relationship

B: Total number of registered failures

Method of application: "Observing the behaviour of users or maintainers who are trying to resolve failures using diagnostic functions."

(c) Failure analysis capability

Purpose: "Can the user identify specific operations which caused the failures? Can the maintainer easily find the cause of failure?"

Measurement function: $X = 1 - A/B$

A: Number of failures the causes of which are still not found;

B: Total number of registered failures

Method of application: "Observing the behaviour of users or maintainers who are trying to resolve failures."

(d) Failure analysis efficiency

Purpose: "Can the user efficiently analyse the cause of failure? (Users sometimes perform maintenance by setting parameters.) Can the maintainer easily find the cause of failure?"

Measurement function: $X = Sum(T)/N$ Remark: The standard provides recommendations regarding the time and number of failures to be considered.

T: Time

N: Number of failures

Method of application: "Observing the behaviour of users or maintainers who are trying to resolve failures."

(e) Status monitoring capability

Purpose: "Can the user identify specific operations which caused failure by getting monitored data during operation?"

Measurement function: $X = 1 - A/B$

A: Number of cases for which maintainers or users failed to get monitored data

B: Number of cases for which maintainers or users attempted to get monitored data recording status of software during operation

Method of application: "Observing the behaviour of users or maintainers who are trying to get monitored data recording status of software during operation."

(f) Activity recording

Purpose: "How thorough is the recording of the system status?"

Measurement function: $X = A/B$

A: Number of implemented data logging items as specified and confirmed in reviews

B: Number of data items to be logged defined in the specifications

Method of application: "Counting the number of items logged in the activity log as specified and compare it to the number of items to be logged."

(g) Readiness of diagnostic functions

Purpose: "How thorough is the provision of diagnostic functions?"

Measurement function: $X = A/B$

A: Number of diagnostic functions as specified and confirmed in reviews

B: Number of diagnostic functions required

Method of application: "Counting the number of the diagnostic functions specified and compare it to the number of the diagnostic functions required in the specifications."

Homonyms

QMEs, related to logging in a general sense, are defined in [102], the definitions of which make it clear that not execution tracing or software logging is addressed:

1. Expansion set of QMEs, No. 12: QME: Size of logs (number of logs).

Definition: “Log: a document used to record, describe or denote selected items identified during execution of a process or activity. Usually used with a modifier, such as issue, quality control, or defect. (PMBOK Guide 4th Ed.)”

2. Expansion Set of QMEs, No. 13: QME: Number of Document (including log records).

   Definition: “Document: (1) uniquely identified unit of information for human use, such as report, specification, manual or book, in printed or in electronic from [...] (2) equivalent to an item of documentation.”

The ISO/IEC 25010 standard [112] also defines quality measure elements (QME), some of which rely on the log as input but not on software logging [102]:

1. QME: (No.1) Number of Failures, input for the QME: Log of failures within an organisation
2. QME: (No.2) Number of faults, input for the QME: Log of failures within an organisation
3. QME: (No.3) Number of interruptions, input for the QME: Log of operations

B.2 Output of the Focus Groups

The collected quality property candidates represent the output of the seven focus groups as illustrated in the tables B.1, B.2, B.3, B.4, B.5, B.6, B.7. These tables contain the question processed in the focus group, the anonymous mark “P” with a consecutive number for the participants to show the assigned score of importance by each participant, and the sum of the scores for each collected item to highlight their importance in the sight of the focus group. Finally, the sum of scores for each participant and for all collected items are also indicated for verification purposes. Each participant had 20 scores to distribute among the collected items according to the importance of the items identified.

Table B.1: Data Collection in Focus Group 1

What properties influence execution tracing quality?	P1	P2	P3	P4	P5	P6	P7	P8	Sum of scores
be it designed, not ad-hoc	1	3	2	2	3	4	8	3	26
location in the code	2	2	2	2	2		4	3	17
be it centrally available	1	1	1	1					4
be access rights logged									0
easy configuration	3	2	2	2		3		2	14
be the structure of log the same at different applications belonging to the same application domain	1	1	1	1	2				6
well-structured	2	1	3	2	3	2			13
information content	2	3	2	3	3	2	6	3	24
data safety regulation satisfied	3	1	1	1		2	1	2	11
protection of personal data	1	1	1	1		1			5
be it easily legible, user-friendly		2	2		3				7
be the flow control identifiable, transaction could be followed over the component boundaries	2	1	1	2	3	4	1	3	17
no superfluous redundancies		1	1	1					3
consequent log-level all over the application	1	1	1	1	1	1		2	8
performance with regard to storage	1			1		1		2	5
Sum of scores:	20	20	20	20	20	20	20	20	160

Table B.2: Data Collection in Focus Group 2

What properties influence execution tracing quality?	P1	P2	P3	P4	P5	Sum of scores
be the information density of the log data good	2				2	4
event driven logging, where needed, more logs, where not needed, less logs	2	3	2	3		10
be the host, process, time, etc. identifiable based on the log	2	3	4	2	2	13
logging exceptions appropriately		3	3	2	1	9
performance with regard to storage and space utilization	2	2		1		5
be context information logged to identify higher-level actions		3			1	4
GDPR (protecting personal data)	1	2	1	1	3	8
be the logging trustworthy from technical point of view	2	2	3	3	3	13
be the logging trustworthy from audit point of view, no data tempering				2		2
structure and text of log messages should remain constant in time						0
be it designed what data are logged (it is frequently inadequate on newly developed software parts)				2	1	3
logging environment variables	1	2			1	4
performance (time behaviour)	4		5	2	2	13
be the logs accessible and processable even in the case of critical system errors	4		2	2	4	12
Sum of scores:	20	20	20	20	20	100

Table B.3: Data Collection in Focus Group 3

What properties influence execution tracing quality?	P1	P2	P3	P4	P5	P6	P7	Sum of scores
be well-structured, and clear	5	3	4	5			4	21
be the log messages definite, concise with appropriate details	7	2	5	4			3	21
define the data to be contained in the log	2	1	2	3	5		2	15
be the date of the event logged		1	1	2	1	5		10
the source of the log message could definitely be identified		1			1	5	2	9
be the version of the software logged		1	1		1	5		8
conformance with data protection regulations, protection of data, be the passwords not logged	2	1			5			8
be it readable in text format, possibly without an external tool				2		5		7
be the size of the log file not too big	2	2			2			6
search the log with ease		2	2		2			6
the language of logging be unified e.g. English			2				4	6
tampering could be ruled out		1		2			2	5
be the logging unified, the same error be logged with the same message each time, e.g. error code with a defined error message			2	2				4
tracing exceptions only once, be it concise but be there stack trace					1		3	4
be the log categorized (e.g.. Info, error)	2		1					3
performance of the search in the log		2						2
be the build date logged		1			1			2
performance of automatic parsing		1			1			2
authentication for the logging, so that one could see who contributed the log entry		1						1
Sum of scores:	20	20	20	20	20	20	20	140

Table B.4: Data Collection in Focus Group 4

What properties influence execution tracing quality?	P1	P2	P3	P4	P5	P6	P7	P8	Sum of scores
it should contain important information	5	5	4	10	5	7	8	8	52
be detailed	5	5	8		3	3	6	5	35
be the severity of the error logged	5				3	5	3	3	19
be clear and well-arranged			4	5	4	3	3		19
legibility		10		5		1			16
searchability	5				5	1		4	15
conciseness			4						4
the complexity of the software has an impact on the logging									0
Sum of scores:	20	20	20	20	20	20	20	20	160

Table B.5: Data Collection in Focus Group 5

What properties influence execution tracing quality?	P1	P2	P3	P4	P5	P6	P7	P8	Sum of scores
accuracy, the log should precisely point at the location of the error	20	10	10	5	10	10	8	5	78
Appropriate amount of information, conciseness		3	3		1	5	4	2	18
Error-free logging of information content		3	3	3	1	1	4	1	16
be well structured				3	1	1		4	9
be easily legible		3	3		1	1		1	9
be easy to search in the log		1	1	3	1			2	8
Error-free implementation of logging (not the errors of logs should be logged)					1		4	1	6
standard implementation for logging and using log frameworks				4	1				5
be the size of the log file manageable				2	1			1	4
be consistent (the same log entry at the same error)					1	1		2	4
keeping the chronology in the sequence of the log events					1	1		1	3
Sum of scores:	20	20	20	20	20	20	20	20	160

Table B.6: Data Collection in Focus Group 6

What properties influence execution tracing quality?	P1	P2	P3	P4	P5	P6	Sum of scores
be the log appropriately segmented in the case of multithreaded, multi-component applications	2	4	6	5	4	5	26
the real information content	3	5	4	4	3	3	22
how detailed the log is	5	3	3	2	3	2	18
be there log levels, and be they consistent	3	5	4		2	4	18
format of the log	5	3		6	2		16
size of the log	2		1	3		2	8
searchability			1		5	2	8
how repetitive the log is			1		1	2	4
Sum of scores:	20	20	20	20	20	20	120

Table B.7: Data Collection in Focus Group 7

What properties influence execution tracing quality?	P1	P2	P3	P4	P5	P6	P7	Sum of scores
be the variable values logged, be they logged in a legible manner not just object references, the same for functions	6	3	5	5	5		5	29
coverage, be the log detailed	3	4		3	3	6	4	23
errors could be localized in the source file, line number	6	2	3		7	4	1	23
information content could be interpreted		2	3	3	2	6	4	20
good legibility, formatting (e.g. not a full xml file in one line)		2	2	5		4	2	15
the call stack could be seen	1	2	5		2			10
be the log written possibly in one file or the different log files could be joined	2	2		2			1	7
log processing could be automatized	1	1		2			3	7
be the timestamp logged, also the fractals of the seconds	1	2	2		1			6
Sum of scores:	20	20	20	20	20	20	20	140

B.3 Common List of the Collected Quality Property Candidates from the Focus Groups

The common list of the quality property candidates is summarised in descending order according to their score of importance in table B.8. Zero importance score was assigned to item 40, 41, and 42 by the focus groups. Consequently, they are included in the list but they were excluded from the data coding process. The column "Merged Description" comprises the collected items, separated by semicolon, which were considered as identical from the point of view of data coding. The individual importance scores of the identical items were summed up and the cumulated score values put in the column "Importance Score". In addition, a label was assigned to each row in the column "Label", which describes the content of the column "Merged Description" with a couple of words. Finally, the related, and similar labels were grouped in clusters, which is described by the column "Matching Cluster".

Table B.8: Common List of the Execution Tracing Quality Property Candidates, Data Coding, Source: [69]

No.	Merged Description	Importance Score	Label	Matching Cluster
1	no superfluous redundancies; be the log messages definite, concise with appropriate details; be it detailed; conciseness; Appropriate amount of information, conciseness; how detailed the log is;how repetitive the log is; the call stack could be seen; coverage, be it detailed.	136	appropriate details in a concise manner	accuracy
2	location in the code; the source of the log message could definitely be identified; accuracy, the log should precisely point at the location of the error; error could be localized in the source file, line number.	127	accurate location in the source code	accuracy
3	information content; be the information density of the log data good; it should contain important information; the real information content; information content could be interpreted.	122	appropriate details in a concise manner	accuracy
4	well-structured; be well-structured, and clear; be it clear and well-arranged; be it well structured; be the log appropriately segmented in the case of multithreaded, multi-component applications; format of the log.	104	well-structured	legibility
5	be it easily legible, user-friendly; legibility; be it easily legible; be the variable values logged, be it in a legible manner not just object references, the same for functions; good legibility, formatting (e.g. not a full xml file in one line).	76	legibility	legibility
6	search the log with ease; searchability; be it easy to search in the log; searchability.	37	easy search	legibility
7	be it designed, not ad-hoc; be it designed what data are logged (it is frequently inadequate on newly developed software parts).	29	design	design and implementation
8	be the host, process, time, etc. identifiable based on the log; be the date of the event logged; be the timestamp logged, also the fractals of the seconds.	29	appropriate details in a concise manner	accuracy
9	data safety regulation satisfied; GDPR (protecting personal data); conformance with data protection regulations, protection of data, be the passwords not logged.	27	security	security
10	consequent log-level all over the application; be there log levels, and be they consistent.	26	consistency	consistency

Table B.8: Common List of the Execution Tracing Quality Property Candidates, Data Coding, Source: [69]

No.	Merged Description	Importance Score	Label	Matching Cluster
11	be the size of the log file not too big; be the size of the log file manageable; size of the log; be the log written possibly in one file or the different log files could be joined.	25	log size	design and implementation
12	be the log categorized (e.g.. Info, error); be the severity of the error logged.	22	log levels	accuracy
13	be the flow control identifiable, transaction could be followed over the component boundaries; be context information logged to identify higher-level actions.	21	ability to identify high level actions	accuracy
14	Error-free logging of information content.	16	log implementation reliability	design and implementation
15	define the data to be contained in the log.	15	design	design and implementation
16	easy configuration.	14	configuration	design and implementation
17	logging exceptions appropriately; tracing exceptions only once, be it concise but be there stack trace.	13	exception logging	accuracy
18	be the logging trustworthy from technical point of view.	13	log implementation reliability	design and implementation
19	performance (time behaviour).	13	performance	performance
20	be the logs accessible and processable even in the case of critical system errors.	12	access to logs	access to logs
21	performance with regard to storage; performance with regard to storage and space utilization.	10	performance	performance
22	event driven logging, where needed, there more logs, where not needed less logs.	10	design	design and implementation
23	be the version of the software logged.	8	appropriate details in a concise manner	accuracy
24	be the logging unified, the same error be logged with the same message each time, e.g. error code with a defined error message; consistent (the same log entry at the same error).	8	consistency	consistency
25	be the logging trustworthy from audit point of view, no data tempering; tampering could be ruled out.	7	security	security
26	be it readable in text format, possibly without an external tool.	7	legibility	legibility
27	log processing could be automatized;	7	design	design and implementation
28	be the structure of log the same at different applications belonging to the same application domain.	6	design	design and implementation
29	the language of logging be unified e.g. English.	6	legibility	legibility
30	Error-free implementation of logging (not the errors of logs be logged).	6	log implementation reliability	design and implementation

Table B.8: Common List of the Execution Tracing Quality Property Candidates, Data Coding, Source: [69]

No.	Merged Description	Importance Score	Label	Matching Cluster
31	protection of personal data.	5	security	security
32	standard implementation for logging and using log frameworks.	5	design	design and implementation
33	be it centrally available.	4	access to logs	access to logs
34	logging environment variables.	4	appropriate details in a concise manner	accuracy
35	keeping the chronology in the sequence of the log events.	3	sequential accuracy	accuracy
36	performance of the search in the log.	2	easy search	legibility
37	be the build date logged.	2	appropriate details in a concise manner	accuracy
38	performance of automatic parsing.	2	design	design and implementation
39	authentication for the logging, so that one could see who contributed the log entry.			
40	be access rights logged.	0	zero score of importance not coded	zero score of importance not coded
41	structure and text of log messages should remain constant in time.	0	zero score of importance not coded	zero score of importance not coded
42	the complexity of the software has an impact on the logging.	0	zero score of importance not coded	zero score of importance not coded

In addition, the merged item descriptions by clusters in table B.9 are also presented. This is a different view of table B.8 with the same columns grouped by the column “Matching Clusters” and the items are counted for each cluster.

Table B.9: Output of Data Coding Cycle Two, Merged Descriptions by Clusters, Source: [69]

Matching Clusters	Items in the Cluster	Score of Importance	Description
Access to logs	2	16	be the logs accessible and processable even in the case of critical system errors; be it centrally available;

Table B.9: Output of Data Coding Cycle Two, Merged Descriptions by Clusters, Source: [69]

Matching Clusters	Items in the Cluster	Score of Importance	Description
Accuracy	11	487	no superfluous redundancies; be the log messages definite, concise with appropriate details; be it detailed; conciseness; Appropriate amount of information, conciseness; how detailed the log is; how repetitive the log is; the call stack could be seen; coverage, be it detailed; location in the code; the source of the log message could definitely be identified; accuracy, the log should precisely point at the location of the error; error could be localized in the source file, line number; information content; be the information density of the log data good; it should contain important information; the real information content; information content could be interpreted; be the host, process, time, etc. identifiable based on the log; be the date of the event logged; be the timestamp logged, also the fractals of the seconds; be the log categorized (e.g.. Info, error); be the severity of the error logged; be the flow control identifiable, transaction could be followed over the component boundaries; be context information logged to identify higher-level actions; logging exceptions appropriately; tracing exceptions only once, be it concise but be there stack trace; be the version of the software logged; logging environment variables; keeping the chronology in the sequence of the log events; be the build date logged;
Consistency	2	34	consequent log-level all over the application; be there log levels, and be they consistent; be the logging unified, the same error be logged with the same message each time, e.g. error code with a defined error message; consistent (the same log entry at the same error);

Table B.9: Output of Data Coding Cycle Two, Merged Descriptions by Clusters, Source: [69]

Matching Clusters	Items in the Cluster	Score of Importance	Description
Design and Implementation	12	148	be it designed, not ad-hoc; be it designed what data are logged (it is frequently inadequate on newly developed software parts); be the size of the log file not too big; be the size of the log file manageable; size of the log; be the log written possibly in one file or the different log files could be joined; Error-free logging of information content; define the data to be contained in the log; easy configuration; be the logging trustworthy from technical point of view; event driven logging, where needed, there more logs, where not needed less logs; log processing could be automatized; be the structure of log the same at different applications belonging to the same application domain; Error-free implementation of logging (not the errors of logs be logged); standard implementation for logging and using log frameworks; performance of automatic parsing;
Legibility	6	232	well-structured; be well-structured, and clear; be it clear and well-arranged; be it well structured; be the log appropriately segmented in the case of multithreaded, multi-component applications; format of the log; be it easily legible, user-friendly; legibility; be it easily legible; be the variable values logged, be it in a legible manner not just object references, the same for functions; good legibility, formatting (e.g. not a full xml file in one line); search the log with ease; searchability; be it easy to search in the log; searchability; be it readable in text format, possibly without an external tool; the language of logging be unified e.g. English; performance of the search in the log;
Performance	2	23	performance (time behaviour); performance with regard to storage; performance with regard to storage and space utilization;
Security	4	40	data safety regulation satisfied; GDPR (protecting personal data); conformance with data protection regulations, protection of data, be the passwords not logged; be the logging trustworthy from audit point of view, no data tempering; tampering could be ruled out; authentication for the logging, so that one could see who contributed the log entry; protection of personal data;

The final clusters, i.e. quality properties, based on the output of the focus groups are introduced in table B.10. The original item descriptions are listed for each cluster and are separated by a semicolon in the column

"Description". These items stem from the column "Merged Description" of the table B.8. The column "Score of Importance" shows the summed up score values assigned by the focus groups to each item, while the number of items is also indicated in the column "Items in the Cluster".

Table B.10: Output of Data Coding Cycle, Final Clusters, Source: [69]

Clusters	Items in the Cluster	Score of Importance	Description
Accuracy	13	521	no superfluous redundancies; be the log messages definite, concise with appropriate details; be it detailed; conciseness; Appropriate amount of information, conciseness; how detailed the log is; how repetitive the log is; the call stack could be seen; coverage, be it detailed; location in the code; the source of the log message could definitely be identified; accuracy, the log should precisely point at the location of the error; error could be localized in the source file, line number; information content; be the information density of the log data good; it should contain important information; the real information content; information content could be interpreted; be the host, process, time, etc. identifiable based on the log; be the date of the event logged; be the timestamp logged, also the fractals of the seconds; be the log categorized (e.g.. Info, error); be the severity of the error logged; be the flow control identifiable, transaction could be followed over the component boundaries; be context information logged to identify higher-level actions; logging exceptions appropriately; tracing exceptions only once, be it concise but be there stack trace; be the version of the software logged; logging environment variables; keeping the chronology in the sequence of the log events; be the build date logged; consequent log-level all over the application; be there log levels, and be they consistent; be the logging unified, the same error be logged with the same message each time, e.g. error code with a defined error message; consistent (the same log entry at the same error);

Table B.10: Output of Data Coding Cycle, Final Clusters, Source: [69]

Clusters	Items in the Cluster	Score of Importance	Description
Legibility	6	232	well-structured; be well-structured, and clear; be it clear and well-arranged; be it well structured; be the log appropriately segmented in the case of multithreaded, multi-component applications; format of the log; be it easily legible, user-friendly; legibility; be it easily legible; be the variable values logged, be it in a legible manner not just object references, the same for functions; good legibility, formatting (e.g. not a full xml file in one line); search the log with ease; searchability; be it easy to search in the log; searchability; be it readable in text format, possibly without an external tool; the language of logging be unified e.g. English; performance of the search in the log;
Design and Implementation	16	187	be the logs accessible and processable even in the case of critical system errors; be it centrally available; be it designed, not ad-hoc; be it designed what data are logged (it is frequently inadequate on newly developed software parts); be the size of the log file not too big; be the size of the log file manageable; size of the log; be the log written possibly in one file or the different log files could be joined; Error-free logging of information content; define the data to be contained in the log; easy configuration; be the logging trustworthy from technical point of view; event driven logging, where needed, there more logs, where not needed less logs; log processing could be automatized; be the structure of log the same at different applications belonging to the same application domain; Error-free implementation of logging (not the errors of logs be logged); standard implementation for logging and using log frameworks; performance of automatic parsing; performance (time behaviour); performance with regard to storage; performance with regard to storage and space utilization;
Security	4	40	data safety regulation satisfied; GDPR (protecting personal data); conformance with data protection regulations, protection of data, be the passwords not logged; be the logging trustworthy from audit point of view, no data tempering; tampering could be ruled out; authentication for the logging, so that one could see who contributed the log entry; protection of personal data;

B.4 Execution Tracing Quality Property Candidates Extracted from the Literature

B.4.1 Queries Issued in the Scientific Document Databases

The queries performed to collect execution tracing related sources were recorded for supporting reproducibility. The logical query: (“execution tracing” OR logging) AND quality AND maintainability AND software AND (model OR framework) was issued in the different document databases in their own query-dialect as shown below:

1. ACM

   (a) **Search term:** recordAbstract:(+(“execution tracing” logging) +quality +maintainability +software +(model framework))

   (b) **Search term:** (+(“execution tracing” logging) +quality +maintainability +software +(model framework))

   Database: Full-text, hosted

   (c) **Search term:** (+(“execution tracing” logging) +quality +maintainability +software +(model framework))

   Database: Full-text, ACM Guide

2. EBSCO

   Search term: (“execution tracing” OR logging) AND quality AND maintainability AND software AND (model OR framework)

   Searching: Academic Search Premier DB

   Journal type: Peer-reviewed

3. IEEE

   Search term: (“execution tracing” OR logging) AND quality AND maintainability AND software AND (model OR framework)

   Search Type: Abstract search

4. Science Direct

   Search term: (“execution tracing” OR logging) AND quality AND maintainability AND software AND (model OR framework) AND (measure OR measurement))

   Subject: Computer Science

   Fields: title-abstr-key

5. Scopus

   Search term: (“execution tracing” OR logging) AND quality AND maintainability AND software AND (model OR framework)

Metadata: title, keyword, abstract

6. Web of Science

Search term: TS=((“execution tracing” OR logging) AND quality AND maintainability AND software AND (model OR framework))

Language: English

Databases: (1) Science Citation Index Expanded (SCI-EXPANDED), (2) Conference Proceedings Citation Index- Science (CPCI-S)

B.4.2 Transcription of Quality Properties from the Literature

In the course of data coding, the explicitly named quality properties were recorded and the properties that are not directly named just paraphrased were transcribed to explicit quality properties so that these could be used in the data coding process. E.g.: The wish articulated in the form: “[...] each of these [log] fields are separated by the |(pipe) symbol [...]” [184] was considered in the scope of the transcription as a requirement to possess the quality properties: legibility and structure of the execution trace file.

Table B.11: Data Coding, Transcription: Quality Properties from the Literature, Source: [69]

ID	Quality Property Related Text Extracted, Transcribed	Reference
1	-performant -flexible to extend -reliable -scalable to deal with increasing data volume -capable to deal with synchronisation	[233]
2	-capable to measure performance at different levels of the architecture -reliable not to loose any log entries	[12]
3	-not to impact application performance -monitoring use: be it adaptive to determine the sampling frequency to monitor a resource	[9]
4	-a logging mechanism be present -log files be legible -timestamps be logged, in the case of more nodes, the clocks be synchronized -logging be performant -different severity levels -unique identifier for actions -being able to process the log files automatically (e.g. filtering duplicate entries) -dashboard with the most important statistics -capability of performing queries on timestamps -prediction of the system availability based on logs	[184]

Table B.11: Data Coding, Transcription: Quality Properties from the Literature, Source: [69]

ID	Quality Property Related Text Extracted, Transcribed	Reference
5	-timestamp be logged -user id be logged -location information be logged -file system metadata (modified, accessed, changed times), file recovery be logged -log file could not be tempered	[232]
6	-accurate -secure (no fake actions, no log violation, avoid being compromised) -performant	[127]
7	-performance -no negative impact on the performance of the application -capability of fine-grained performance measurements across application component boundaries -capability to be changed by configuration -capability to determine the user actions based on the log	[212]
8	-trace code insertion be automatic, manual is error-prone	[213]
9	-capability for both analysis of errors and auditing -capability of usage statistic creation -capability of performance analysis -capability to measure quality of service metrics for Service-Level License Agreements (SLA) -capability of replaying and simulating a sequence of actions based on the log -providing enough information, if an error is logged	[238]
10	-locate the logging mechanism in separate modules, do not mix it into the code of the application -the design needs to consider the logging mechanism	[30]
11	-capability to identify components -capability to identify actions -capability to identify the global business flows based on the trace data -capability to identify the time of the action -usability for (1) mode of activation possibly with or without restarting the application, (2) access mode to the trace data through APIs, log files or a separate monitoring system, (3) data availability as the things happen or only after the trace session is finished	[122]
12	-low performance impact on the application -multi-level severity: warning error, fatal, unhandled exception -capability to measure memory consumption and process time -capability for configuration, disable or enable tracing -capability to configure the output format: txt, db etc.	[123]
13	-precision to log enough information -consistent naming, no misleading entries in the log files	[230]

Table B.11: Data Coding, Transcription: Quality Properties from the Literature, Source: [69]

ID	Quality Property Related Text Extracted, Transcribed	Reference
14	-capability for configuration -capability to log at different levels of severity -capability to measure performance with different granularities (not only response time but the quality of data returned during this time) -low performance impact on the application	[5]
15	-performance -capability to log enough information -capability to correlate actions on different nodes (distributed applications) -keep the perturbations minimal the tracing causes in the application -capability to identify the inputs and outputs where the data come from and where they go to -capability to identify task switches, which task when (real-time) -capability to log interrupts (real-time) -capability to log tick rate -capability to log CPU usage -capability to log memory usage -capability to log network utilisation -capability to log the state of the real-time kernel and answer questions such as: Which tasks are waiting for their turn to execute (waiting queue, list, or table)? Which task is running? Which tasks are blocked?	[231]

B.4.3 Clustering

As a further step in the data coding, all the identified text parts were listed from the publications related to the quality property candidates and the similar ones were grouped in one cluster while the distant ones kept in separate clusters. This way four clusters could be constructed as the tables illustrate below: “Design and Implementation”, “Accuracy”, “Performance”, moreover “Legibility and Interpretation”, which constitute the quality properties of execution tracing based on the identified literature.

Quality Property Cluster: Design and Implementation

The cluster contains all implementation related quality property candidates identified. This cluster possesses the highest variance with regard to its items in comparison to the other clusters introduced.

Table B.12: Quality Property Cluster: Design and Implementation, Source: [69]

Cluster Name: Design and Implementation

Publication ID	Quality Property Related Text Extracted, Transcribed
1	flexible to extend
1	reliable
1	scalable to deal with increasing data volume
1	capable to deal with synchronisation
2	capable to measure performance at different levels of the architecture
2	reliable not to loose any log entries
3	monitoring use: be it adaptive to determine the sampling frequency to monitor a resource
4	a logging mechanism be present
4	being able to process the log files automatically (e.g. filtering duplicate entries)
4	capability of performing queries on timestamps
4	prediction of the system availability based on logs
5	log file could not be tempered
6	secure (no fake actions, no log violation, avoid being compromised)
7	capability of fine-grained performance measurements across application component boundaries
7	capability to be changed by configuration
8	trace code insertion be automatic, manual is error-prone
9	capability of usage statistic creation
9	capability of performance analysis
9	capability to measure quality of service metrics for Service-Level License Agreements (SLA)
9	capability of replaying and simulating a sequence of actions based on the log
10	locate the logging mechanism in separate modules, do nor mix it into the code of the application
10	the design needs to consider the logging mechanism
11	usability for (1) mode of activation possibly with or without restarting the application, (2) access mode to the trace data through APIs, log files or a separate monitoring system, (3) data availability as the things happen or only after the trace session is finished
12	capability to measure memory consumption and process time
12	capability for configuration, disable or enable tracing
12	capability to configure the output format: txt, db etc.
14	capability for configuration
14	capability to measure performance with different granularities (not only response time but the quality of data returned during this time)
15	capability to correlate actions on different nodes (distributed applications)
15	keep the perturbations minimal the tracing causes in the application
15	capability to identify task switches, which task when (real-time)
15	capability to log interrupts (real-time)
15	capability to log tick rate
15	capability to log CPU usage
15	capability to log memory usage
15	capability to log network utilisation
15	capability to log the state of the real-time kernel: Which tasks are waiting for their turn to execute (waiting queue, list, or table)? Which task is running? Which tasks are blocked?

Quality Property Cluster: Accuracy

The cluster contains all accuracy related quality property candidates identified. This cluster possesses higher variance than the cluster performance but it is relatively homogeneous.

Table B.13: Quality Property Cluster: Accuracy, Source: [69]

Cluster Name: Accuracy	
Publication ID	Quality Property Related Text Extracted, Transcribed
4	timestamps be logged, in the case of more nodes, the clocks be synchronized
4	different severity levels
4	unique identifier for actions
5	timestamp be logged
5	user id be logged
5	location information be logged
5	file system metadata (modified, accessed, changed times), file recovery be logged
6	accurate
7	capability to determine the user actions based on the log
9	capability for both analysis of errors and auditing
9	providing enough information, if an error is logged
11	capability to identify components
11	capability to identify actions
11	capability to identify the global business flows based on the trace data
11	capability to identify the time of the action
12	multi-level severity: warning error, fatal, unhandled exception
13	precision to log enough information
13	consistent naming, no misleading entries in the log files
14	capability to log at different levels of severity
15	capability to log enough information
15	capability to identify the inputs and outputs where the data come from and where they go to

Quality Property Cluster: Performance

The cluster contains all performance related quality property candidates identified. It is homogeneous as the word “performance” or “performant” occur in each extracted item.

Quality Property Cluster: Legibility and Interpretation

This cluster of items require that the trace output be legible and the most important data could be interpreted also in an aggregated manner. This group is relatively small in comparison to the previous clusters.

Table B.14: Quality Property Cluster: Performance, Source: [69]

Cluster Name: Performance	
Publication ID	Quality Property Related Text Extracted, Transcribed
1	performant
3	not to impact application performance
4	logging be performant
6	performant
7	performance
7	no negative impact on the performance of the application
12	low performance impact on the application
14	low performance impact on the application
15	performance

Table B.15: Quality Property Cluster: Legibility and Interpretation, Source: [69]

Cluster Name: Legibility and Interpretation	
Publication ID	Quality Property
4	log files be legible
4	dashboard with the most important statistics

Appendix C

Model Construction

The charts and tables below stem from the author's journal publication in [68]. This material is necessary to include in the thesis to support the claims and explain the details of the research conducted. The questionnaire and the collected data were published as a separate data article in [67].

C.1 Data Processing and Exploratory Data Analysis

C.1.1 Rating the Input Variables of the Use Cases

The charts below depict the responses for each input variable of the defined use cases.

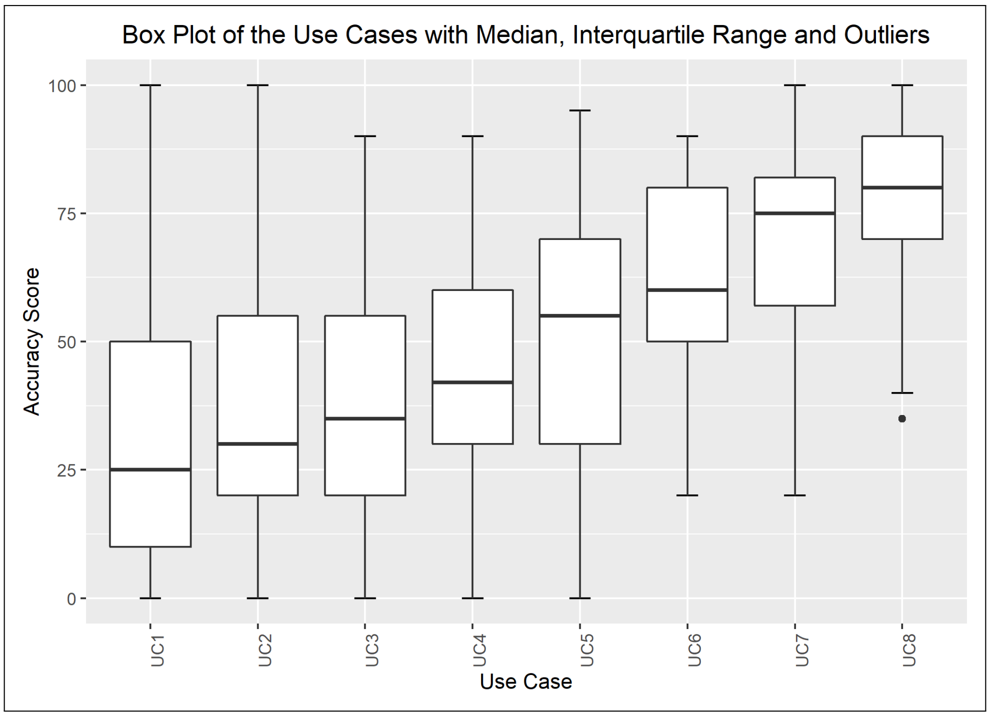

Figure C.1: Distribution of the Accuracy Scores Assigned to the Use Cases, Source: [68]

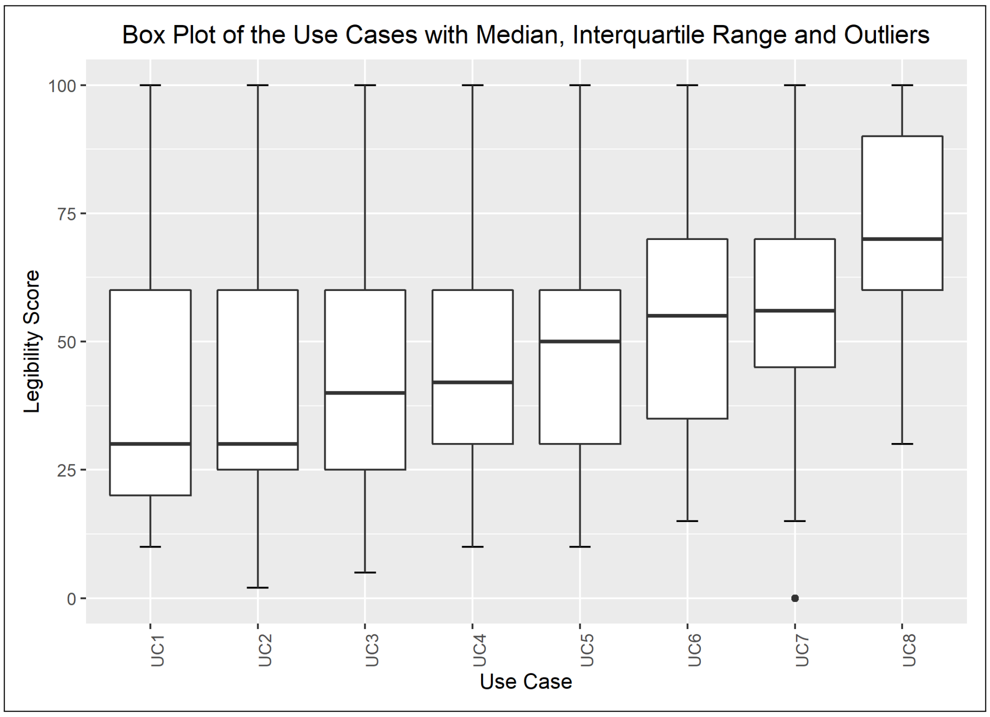

Figure C.2: Distribution of the Legibility Scores Assigned to the Use Cases, Source: [68]

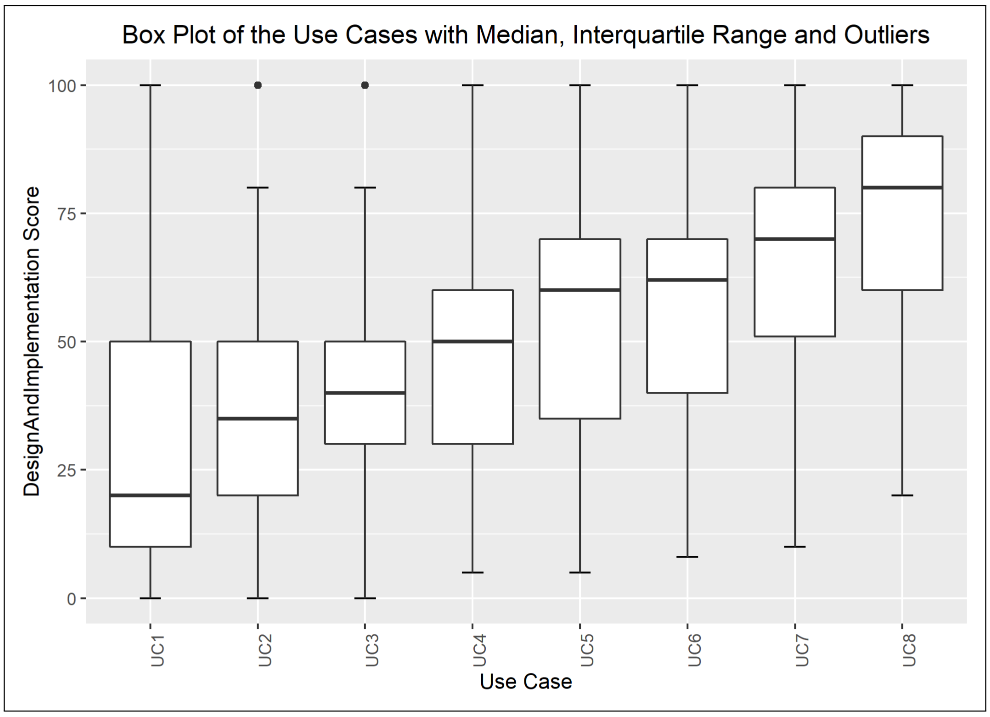

Figure C.3: Distribution of the Design and Implementation Scores Assigned to the Use Cases, Source: [68]

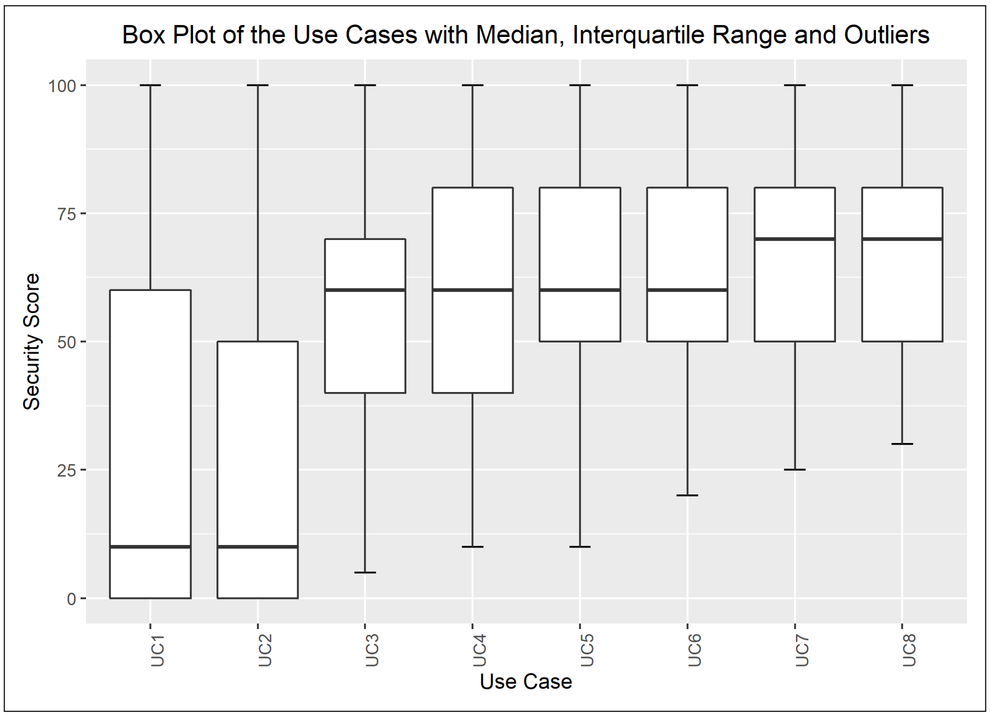

Figure C.4: Distribution of the Security Scores Assigned to the Use Cases, Source: [68]

C.1.2 Normality Checks of the Collected Data

The quantile-quantile plots below depict the distribution of each collected variable. The 0.95% confidence interval is also shown; moreover, the computed Sharpio-Wilk Normality Test values are added to the charts. The extreme input variable value combinations Q20 and Q22 contained solely identical values; therefore, they were excluded from the normality test.

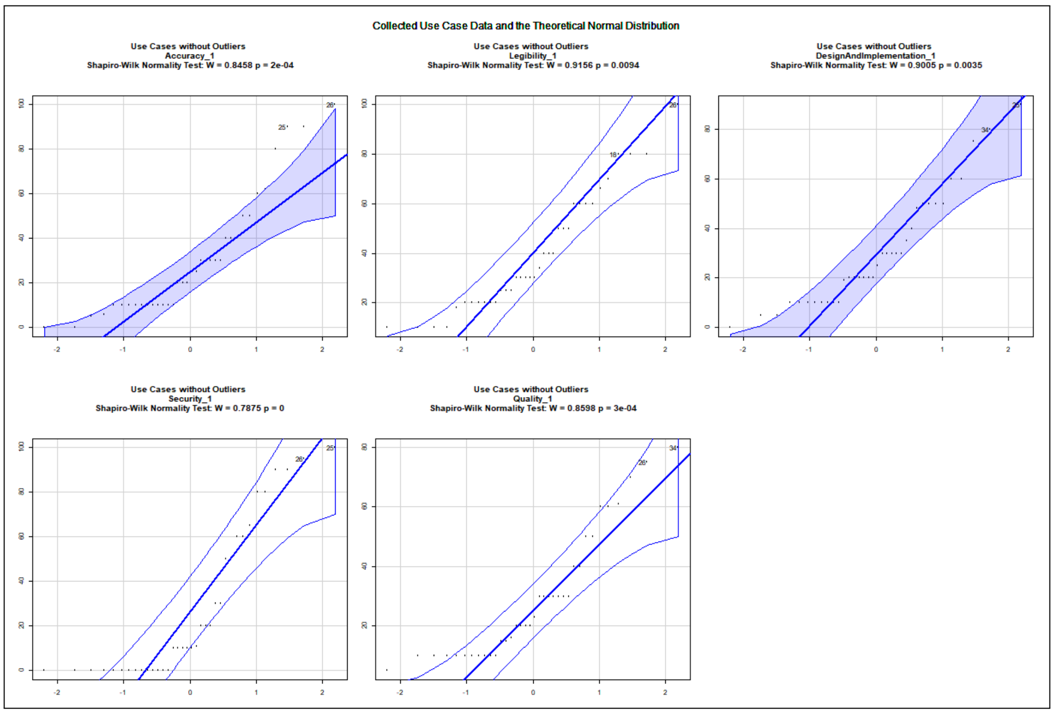

Figure C.5: Distribution of Collected Variables Regarding the Use Cases 1, Source: [68]

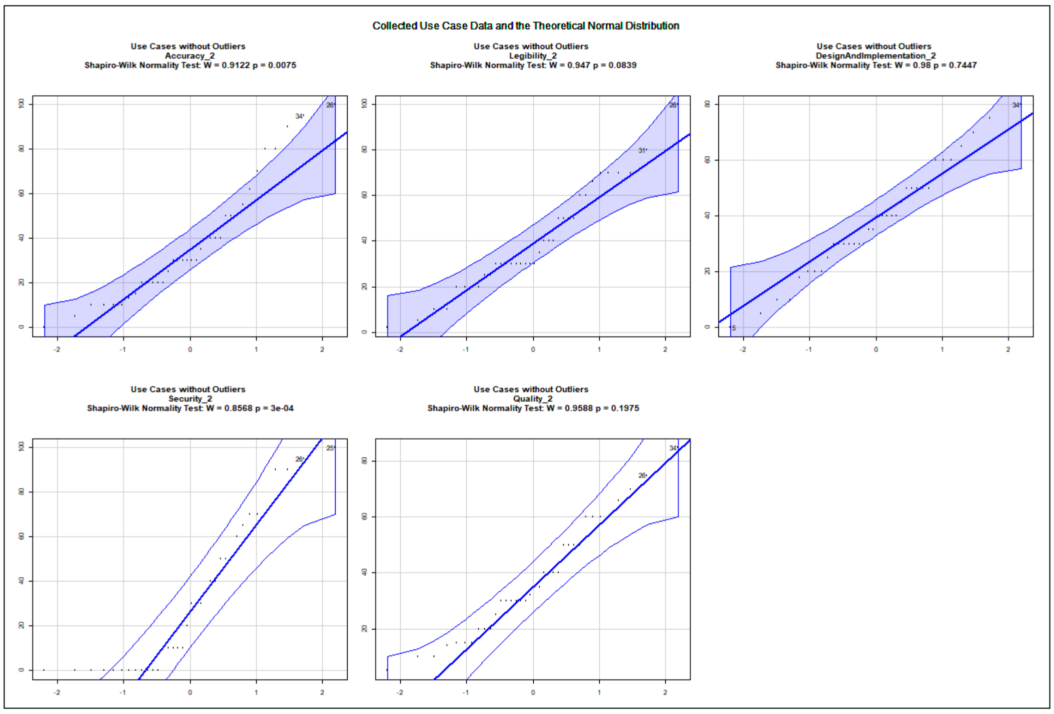

Figure C.6: Distribution of Collected Variables Regarding the Use Cases 2, Source: [68]

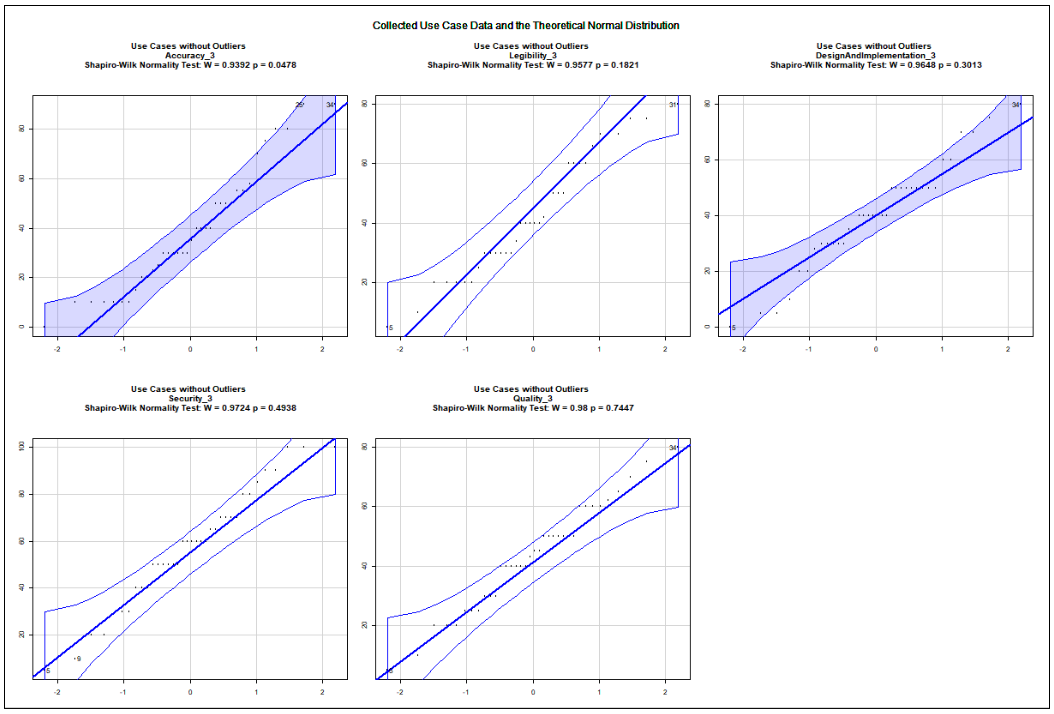

Figure C.7: Distribution of Collected Variables Regarding the Use Cases 3, Source: [68]

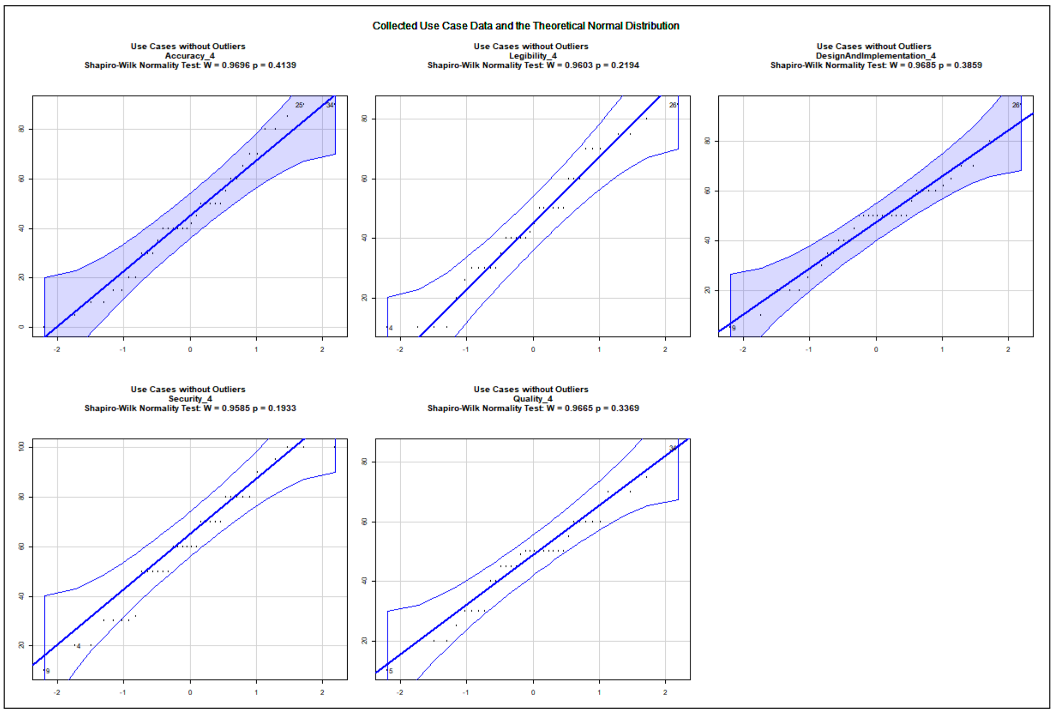

Figure C.8: Distribution of Collected Variables Regarding the Use Cases 4, Source: [68]

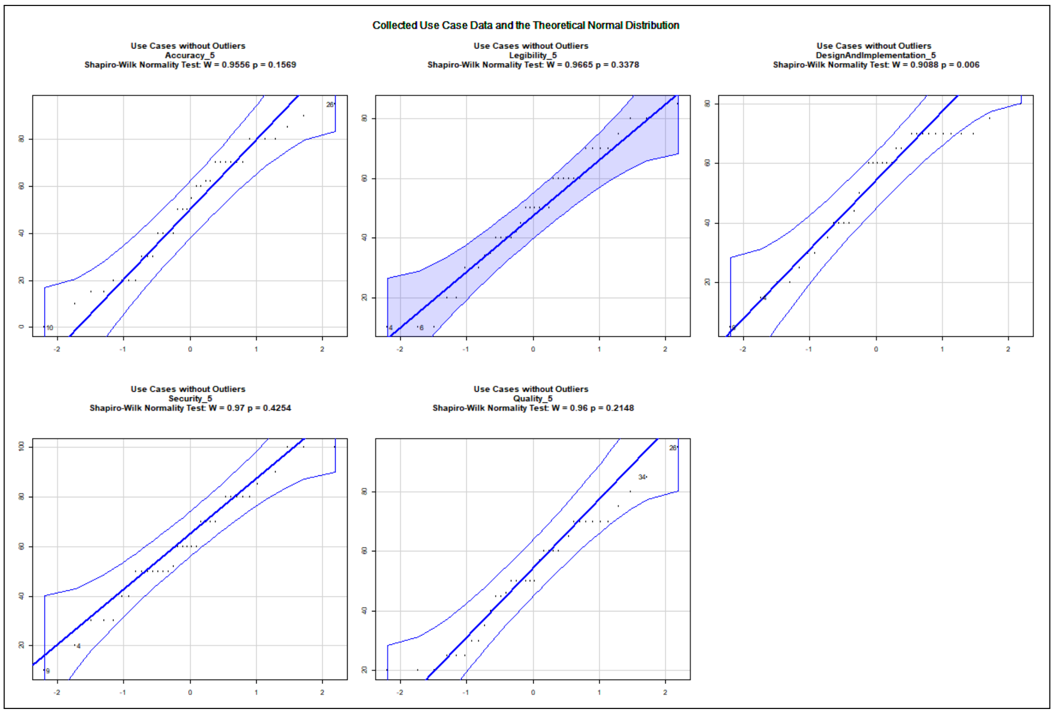

Figure C.9: Distribution of Collected Variables Regarding the Use Cases 5, Source: [68]

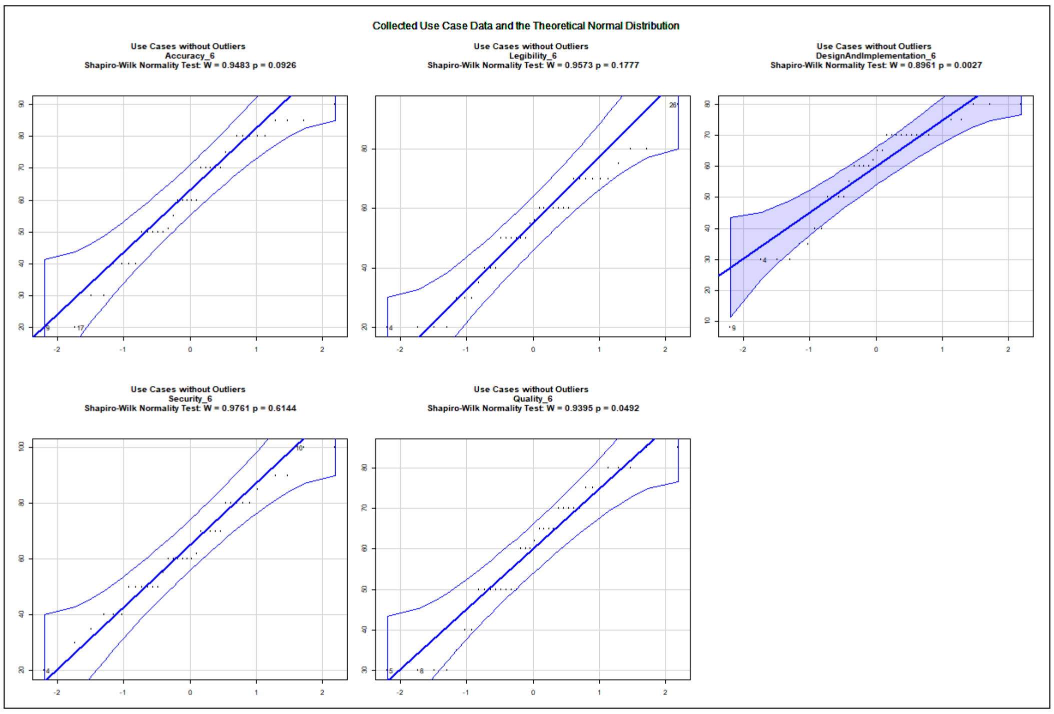

Figure C.10: Distribution of Collected Variables Regarding the Use Cases 6, Source: [68]

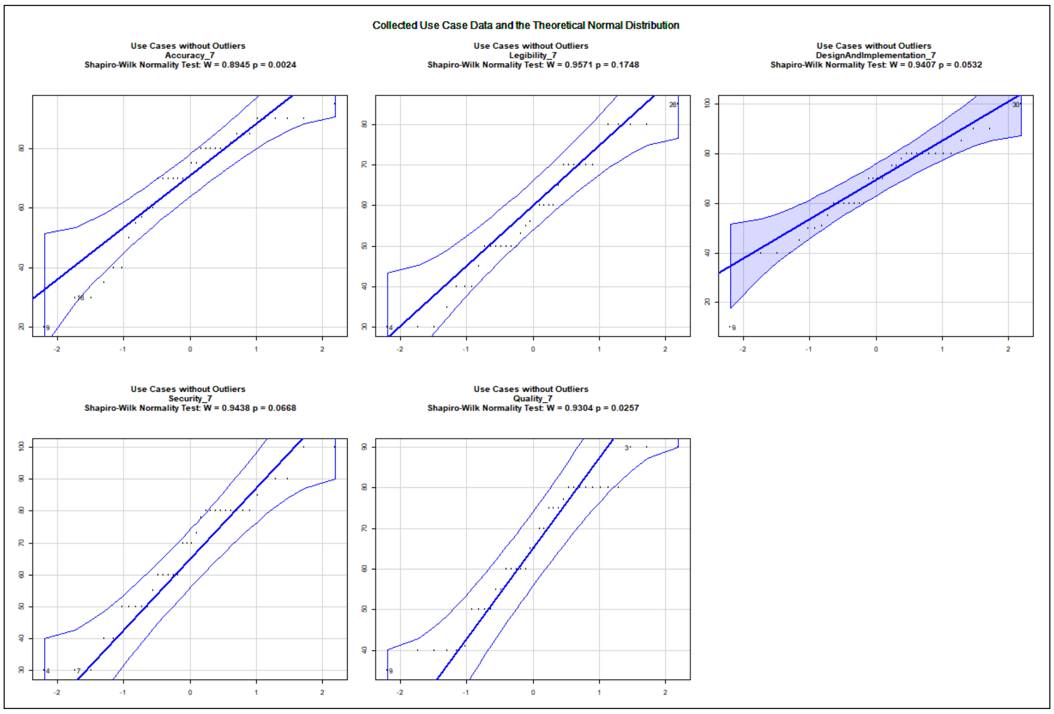

Figure C.11: Distribution of Collected Variables Regarding the Use Cases 7, Source: [68]

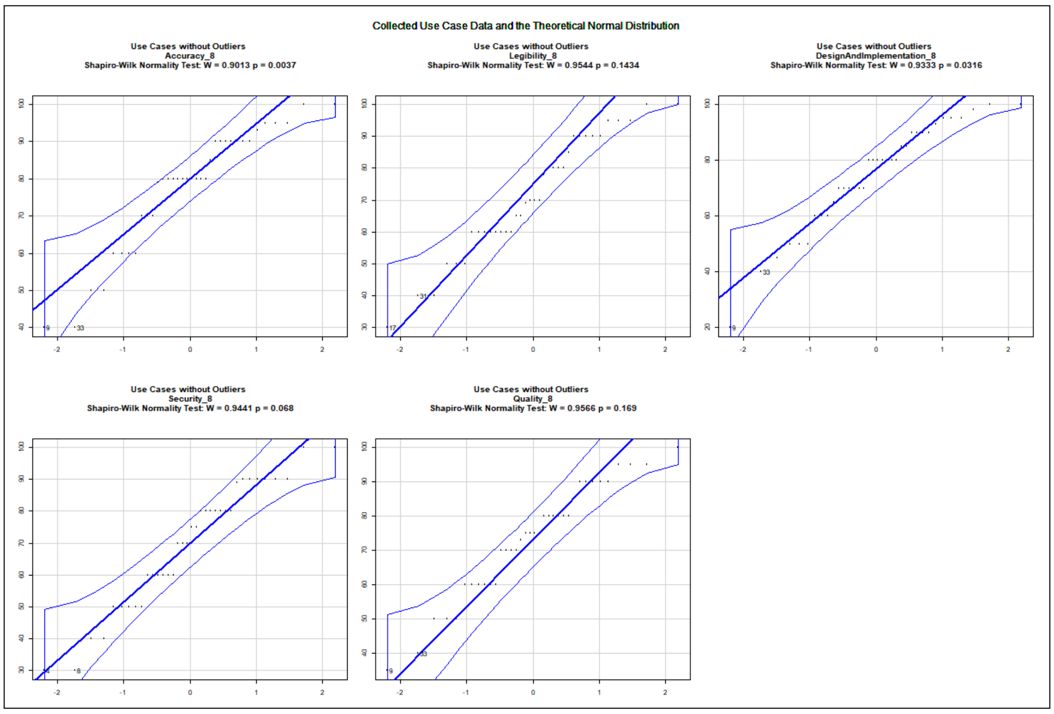

Figure C.12: Distribution of Collected Variables Regarding the Use Cases 8, Source: [68]

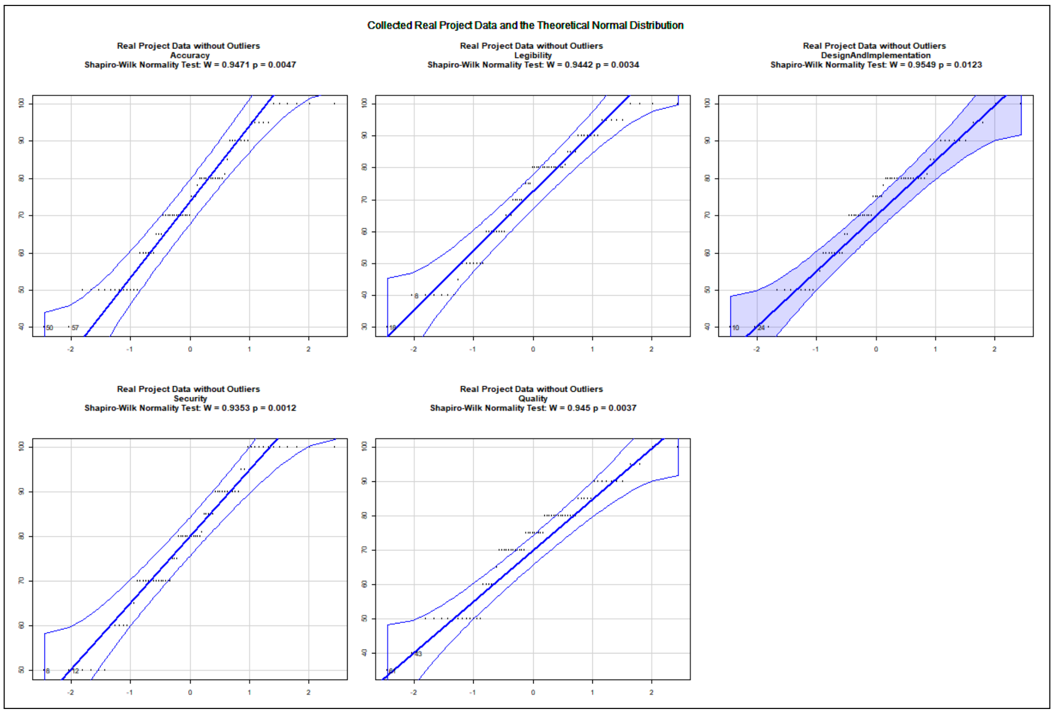

Figure C.13: Distribution of Collected Variables Regarding the Real Project Data, Source: [68]

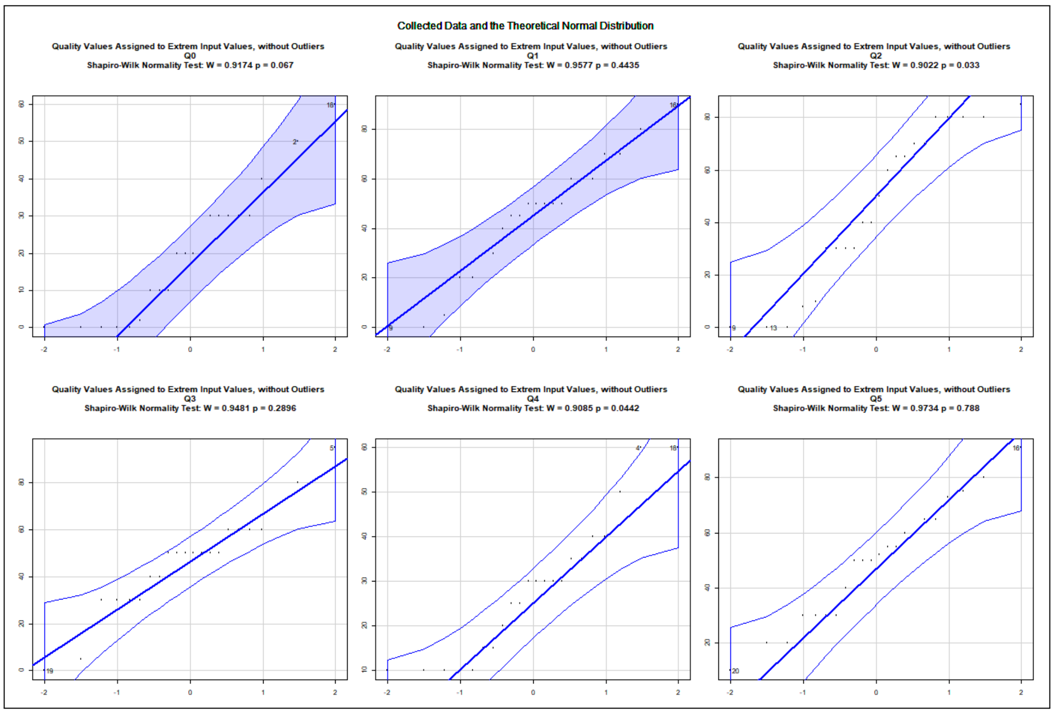

Figure C.14: Distribution of the Assigned Variables to the Extreme Input Variable Values, Q0-Q5, Source: [68]

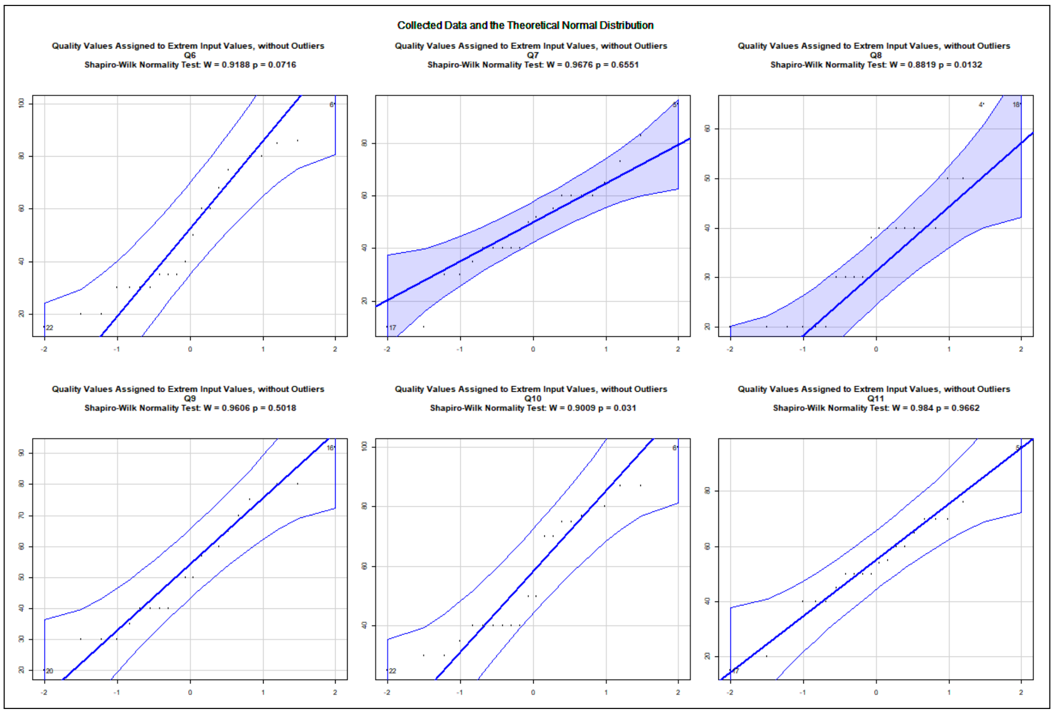

Figure C.15: Distribution of the Assigned Variables to the Extreme Input Variable Values, Q6-Q11, Source: [68]

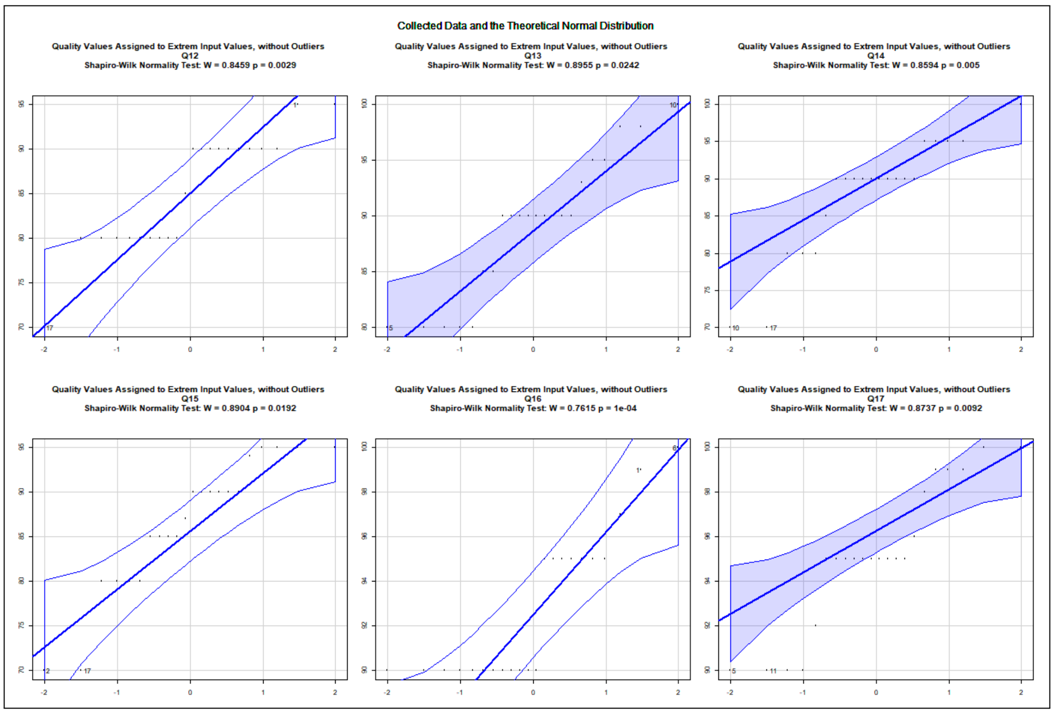

Figure C.16: Distribution of the Assigned Variables to the Extreme Input Variable Values, Q12-Q17, Source: [68]

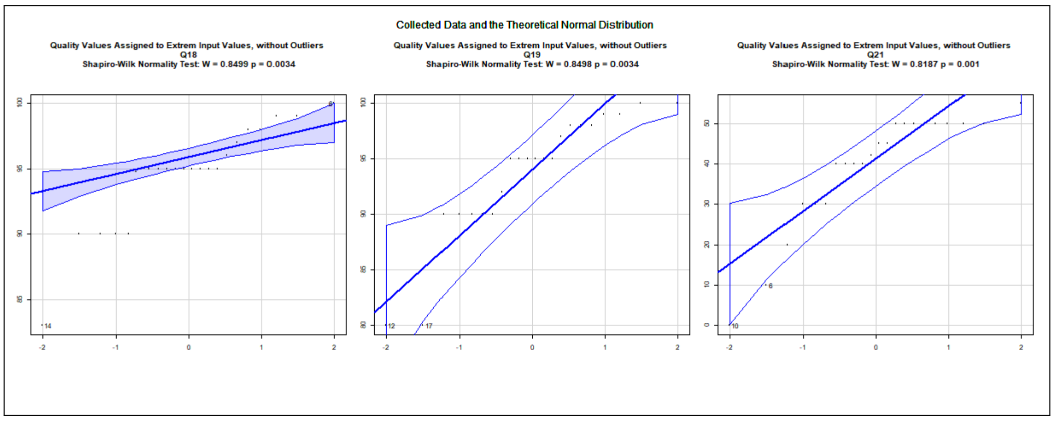

Figure C.17: Distribution of the Assigned Variables to the Extreme Input Variable Values, Q18-Q21, Source: [68]

C.2 Modelling

Figure C.18 and figure C.19 illustrate the different error indicators with regard to the maximal number of rules set in the course of the learning process. In addition, the corresponding data are demonstrated in table C.1 and in table C.2.

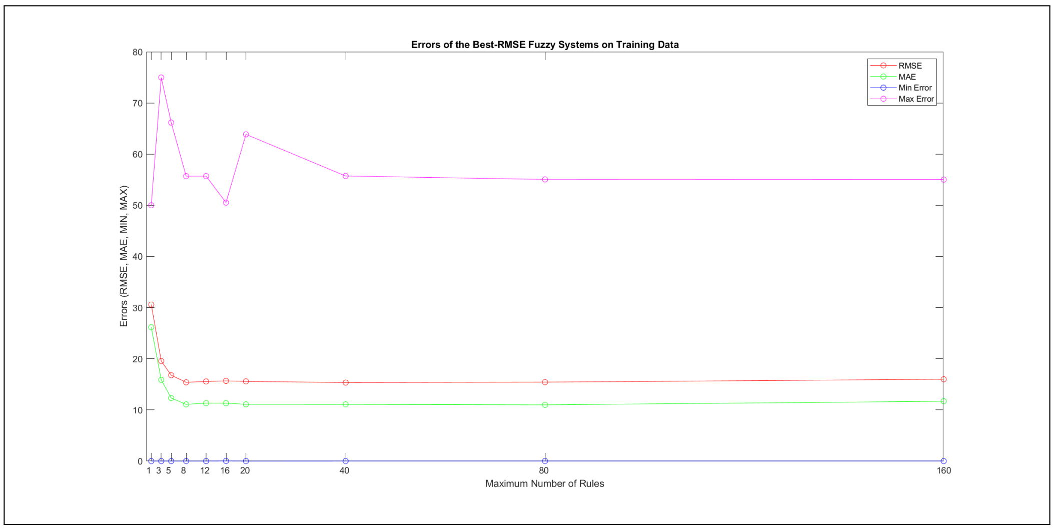

Figure C.18: Best RMSE Errors on the Training Data, Source: [68]

Table C.1: Best RSME Errors on the Training Data, Source: [68]

	Limit: 1 Rule	Limit: 3 Rules	Limit: 5 Rules	Limit: 8 Rules	Limit: 12 Rules	Limit: 16 Rules	Limit: 20 Rules	Limit: 40 Rules	Limit: 80 Rules	Limit: 160 Rules
RMSE	30.59485	19.56332	16.78509	15.39203	15.56531	15.67844	15.58645	15.34062	15.43388	15.99222
MAE	26.16265	15.88731	12.31566	11.08484	11.32251	11.31696	11.08999	11.08607	10.99209	11.69641
Min. Error	0	7.11E-15	7.36E-05	0.000461	0.003325	0.025736	0.000437	0.007998	0.007301	0.010426
Max. Error	50	74.97749	66.16576	55.69301	55.70125	50.522	63.85966	55.72273	55.05721	55.02147

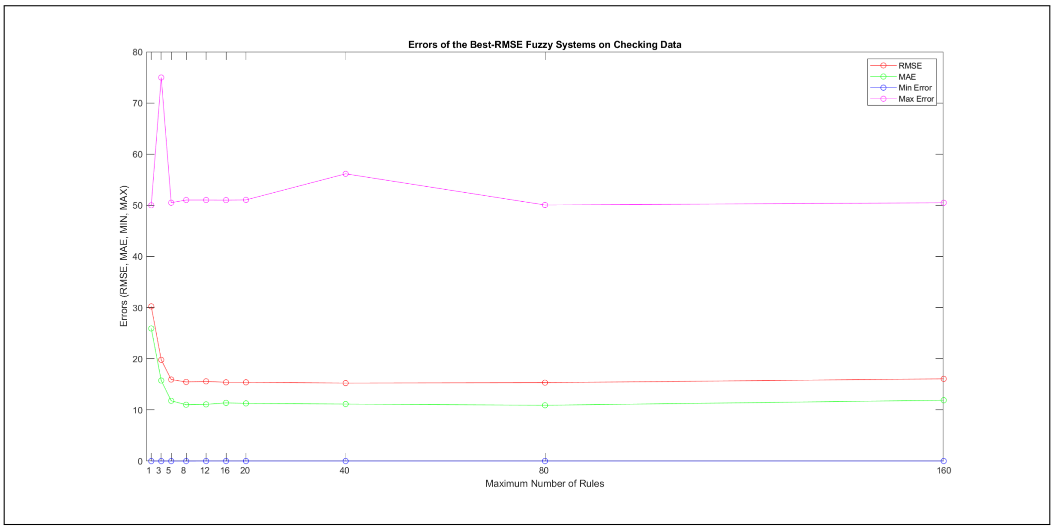

Figure C.19: Best RMSE Errors on the Checking Data, Source: [68]

Table C.2: Best RSME Errors on the Checking Data, Source: [68]

	Limit: 1 Rule	Limit: 3 Rules	Limit: 5 Rules	Limit: 8 Rules	Limit: 12 Rules	Limit: 16 Rules	Limit: 20 Rules	Limit: 40 Rules	Limit: 80 Rules	Limit: 160 Rules
RMSE	30.23458	19.79506	15.93676	15.46742	15.58643	15.39114	15.39485	15.23949	15.32682	16.07228
MAE	25.92113	15.74546	11.78425	11.01338	11.08251	11.37357	11.27761	11.14186	10.90988	11.8917
Min. Error	0	7.11E-15	5.79E-06	0.000461	0.014746	0.011548	0.011135	0.007144	0.007301	0.010426
Max. Error	50	74.97749	50.50592	51.03276	51.03274	51.01228	51.05456	56.15333	50.05721	50.49569

C.3 Extracted Rules Sets and Model Performance

This section contains the extracted rule sets in the course of run suite 2 according to the upper bounds for the maximal number of the linguistic rules specified for the algorithm. Each rule set extracted designates a distinct model. In addition, the charts are also presented which show the performance of each model on the checking and the corresponding training data. The original data has been sorted to ensure that the deviations from the desired targets are easy to notice.

C.3.1 Maximal Number of Linguistic Rules: 1

1. If (Legibility is good) and (Security is medium), then (Quality is medium)

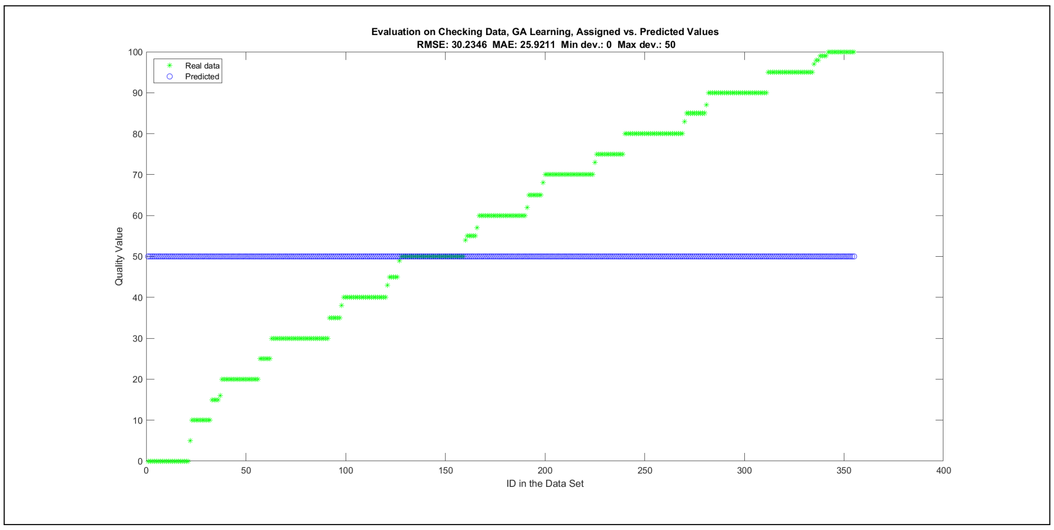

Figure C.20: Evaluation of the Best Performing 1-Rule Model with Regard to Maximum Error on Checking Data, Source: [68]

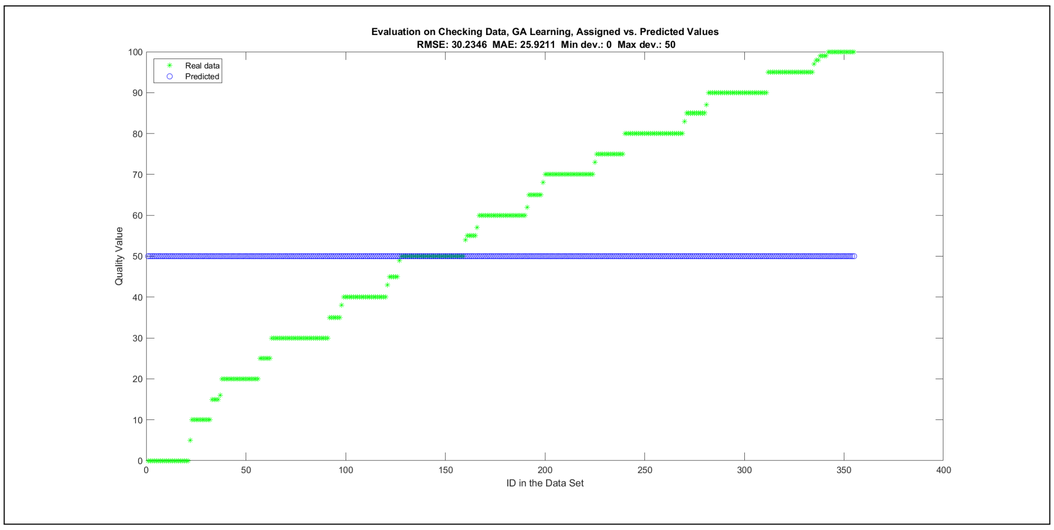

Figure C.21: Evaluation of the Best Performing 1-Rule Model with Regard to RMSE Error on Checking Data, Source: [68]

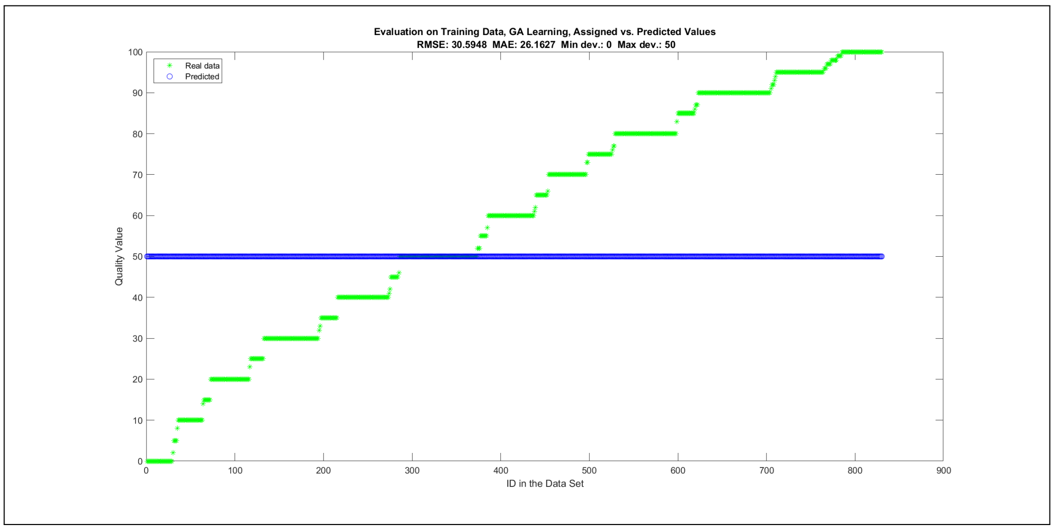

Figure C.22: Evaluation of the Best Performing 1-Rule Model with Regard to Maximum Error on Training Data, Source: [68]

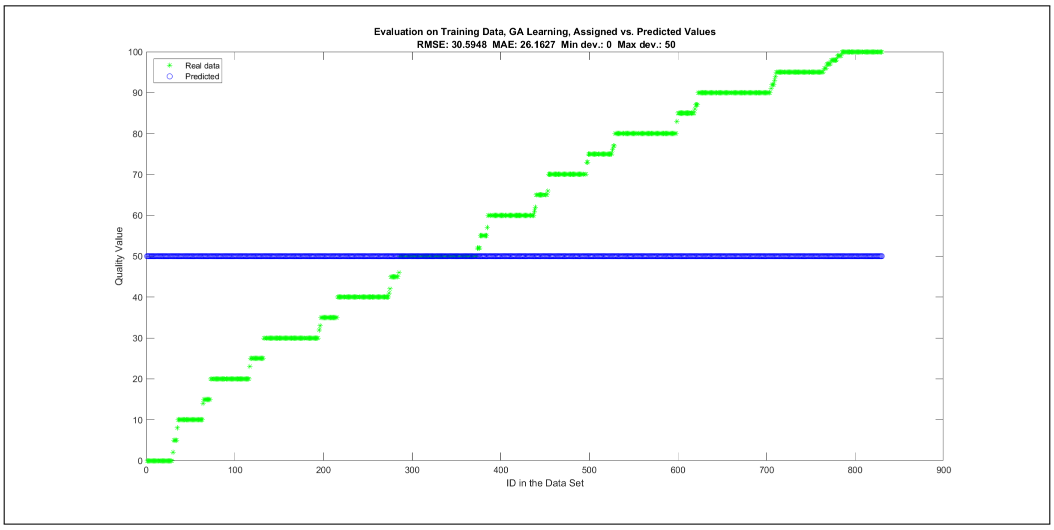

Figure C.23: Evaluation of the Best Performing 1-Rule Model with Regard to RMSE Error on Training Data, Source: [68]

C.3.2 Maximal Number of Linguistic Rules: 3

1. If (Legibility is poor) and (DesignAndImplementation is medium), then (Quality is medium)
2. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is good) and (Security is medium), then (Quality is good)
3. If (Accuracy is poor) and (Security is poor), then (Quality is poor)

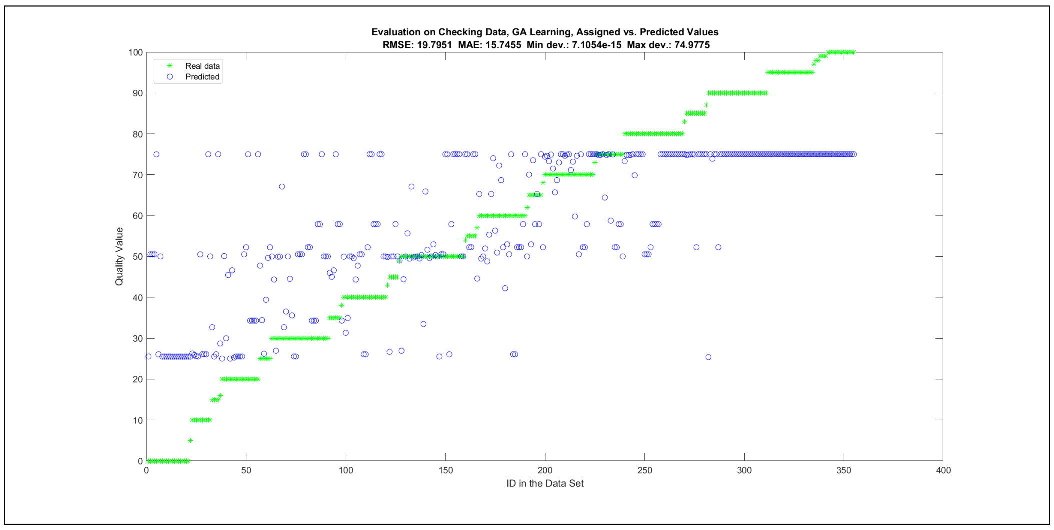

Figure C.24: Evaluation of the Best Performing 3-Rule Model with Regard to Maximum Error on Checking Data, Source: [68]

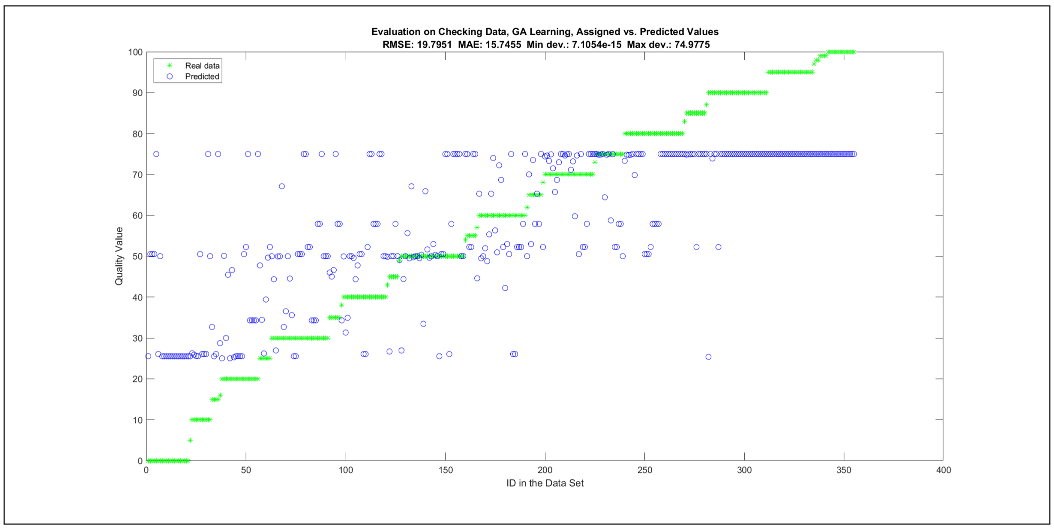

Figure C.25: Evaluation of the Best Performing 3-Rule Model with Regard to RMSE Error on Checking Data, Source: [68]

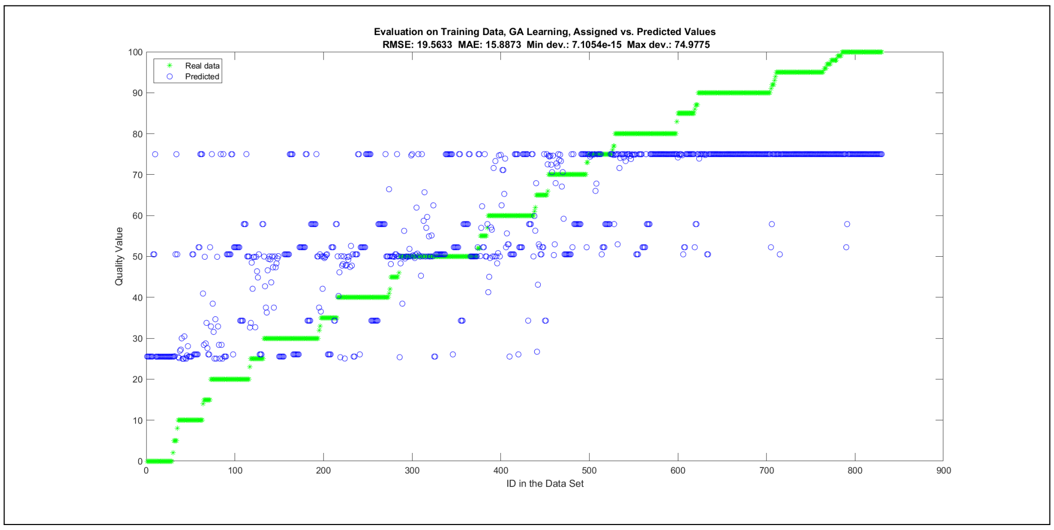

Figure C.26: Evaluation of the Best Performing 3-Rule Model with Regard to Maximum Error on Training Data, Source: [68]

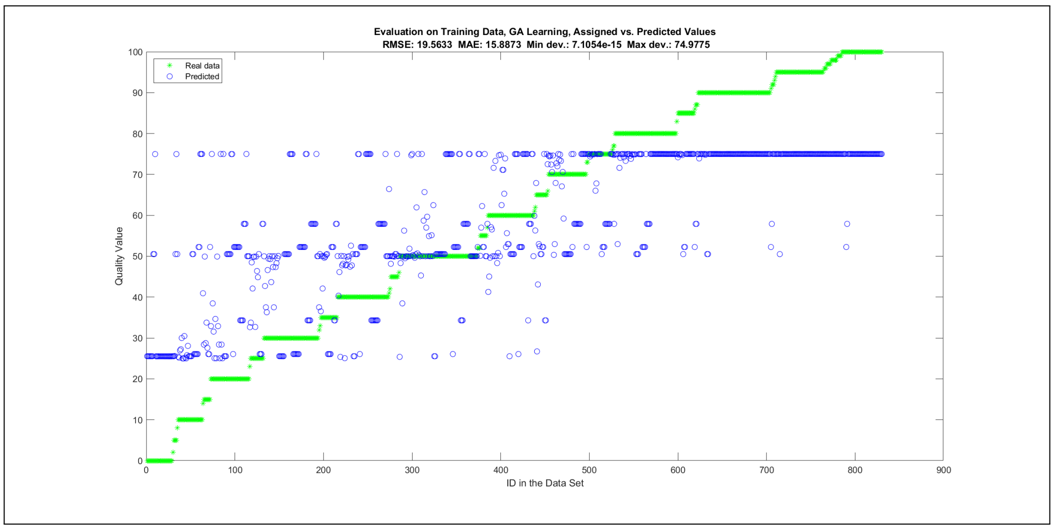

Figure C.27: Evaluation of the Best Performing 3-Rule Model with Regard to RMSE Error on Training Data, Source: [68]

C.3.3 Maximal Number of Linguistic Rules: 5

1. If (Legibility is poor) and (DesignAndImplementation is good) and (Security is medium), then (Quality is medium)
2. If (Accuracy is poor) and (Legibility is good) and (DesignAndImplementation is good) and (Security is good), then (Quality is poor)
3. If (Accuracy is medium) and (Legibility is good) and (DesignAndImplementation is medium), then (Quality is medium)
4. If (Accuracy is medium) and (Legibility is good) and (DesignAndImplementation is good) and (Security is good), then (Quality is very good)

5. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is poor), then (Quality is poor)

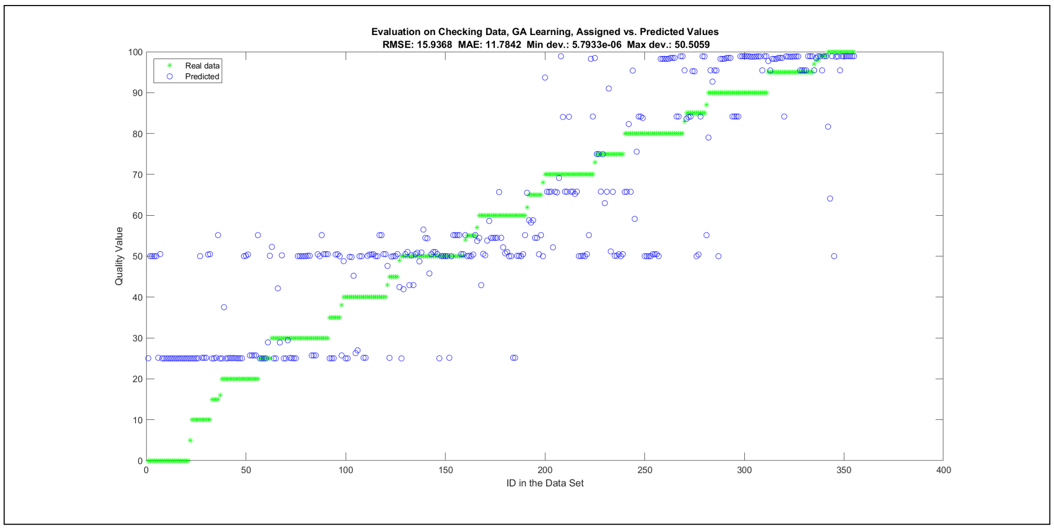

Figure C.28: Evaluation of the Best Performing 5-Rule Model with Regard to Maximum Error on Checking Data, Source: [68]

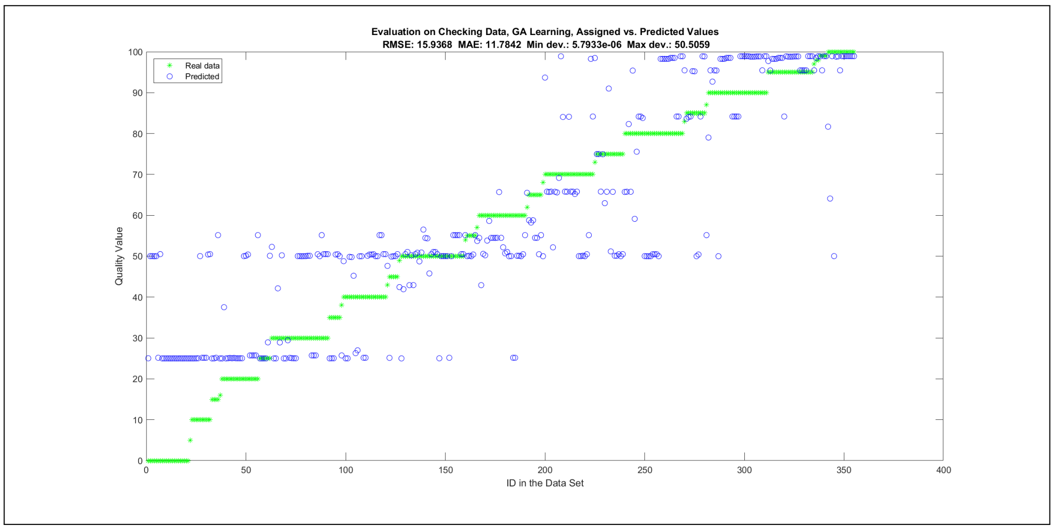

Figure C.29: Evaluation of the Best Performing 5-Rule Model with Regard to RMSE Error on Checking Data, Source: [68]

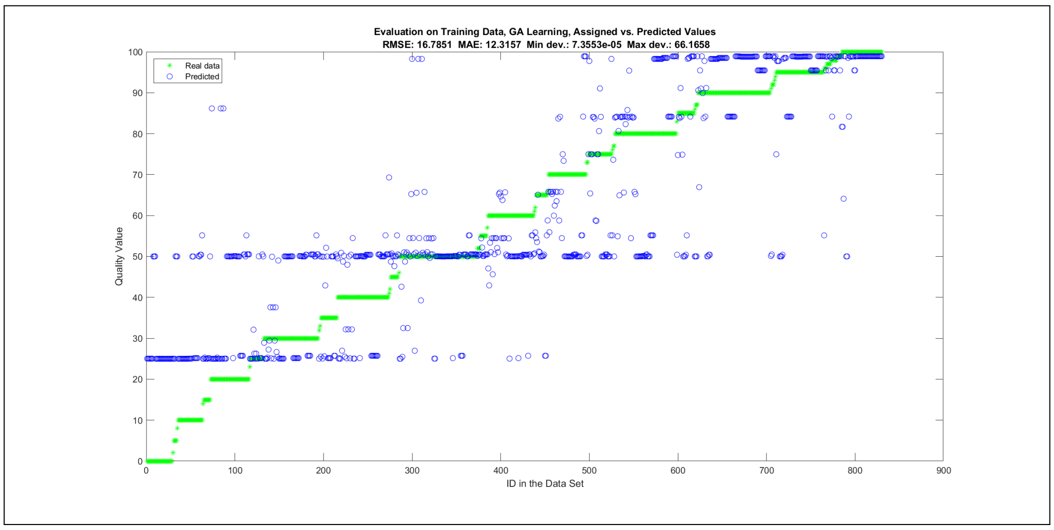

Figure C.30: Evaluation of the Best Performing 5-Rule Model with Regard to Maximum Error on Training Data, Source: [68]

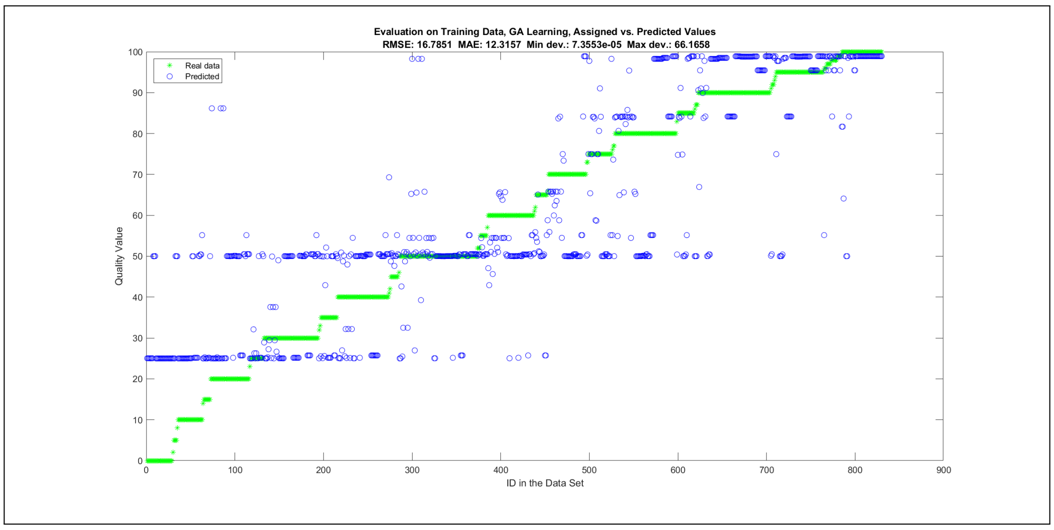

Figure C.31: Evaluation of the Best Performing 5-Rule Model with Regard to RMSE Error on Training Data, Source: [68]

C.3.4 Maximal Number of Linguistic Rules: 8

1. If (Accuracy is medium) and (Legibility is good) and (DesignAndImplementation is good) and (Security is poor), then (Quality is medium)
2. If (Accuracy is good) and (Legibility is good) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is very good)
3. If (Accuracy is poor) and (Legibility is medium), then (Quality is poor)
4. If (Accuracy is good) and (Legibility is good) and (DesignAndImplementation is poor), then (Quality is medium)
5. If (Accuracy is medium) and (Legibility is good) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is poor)

6. If (Legibility is poor) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)

7. If (Accuracy is medium) and (Legibility is good) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is good)

8. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is medium), then (Quality is medium)

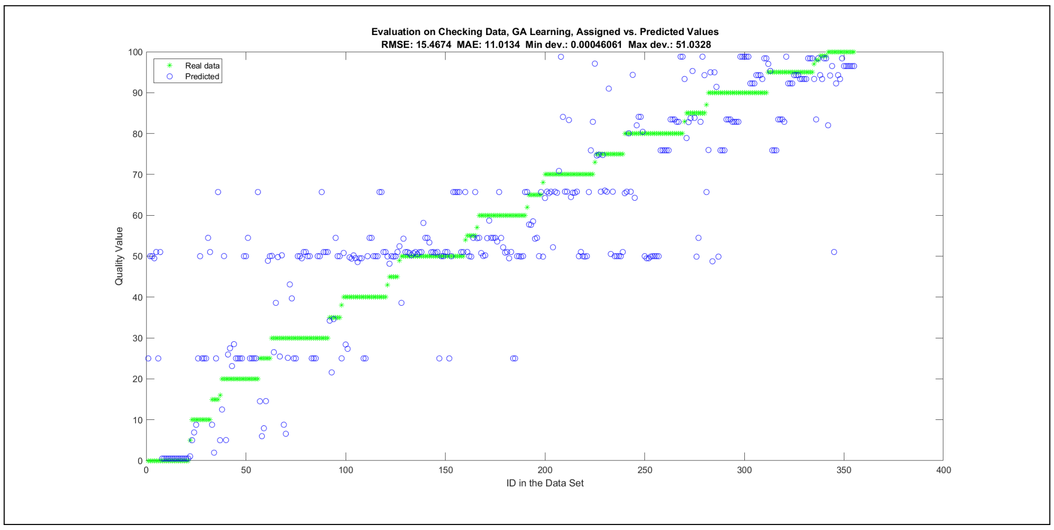

Figure C.32: Evaluation of the Best Performing 8-Rule Model with Regard to Maximum Error on Checking Data, Source: [68]

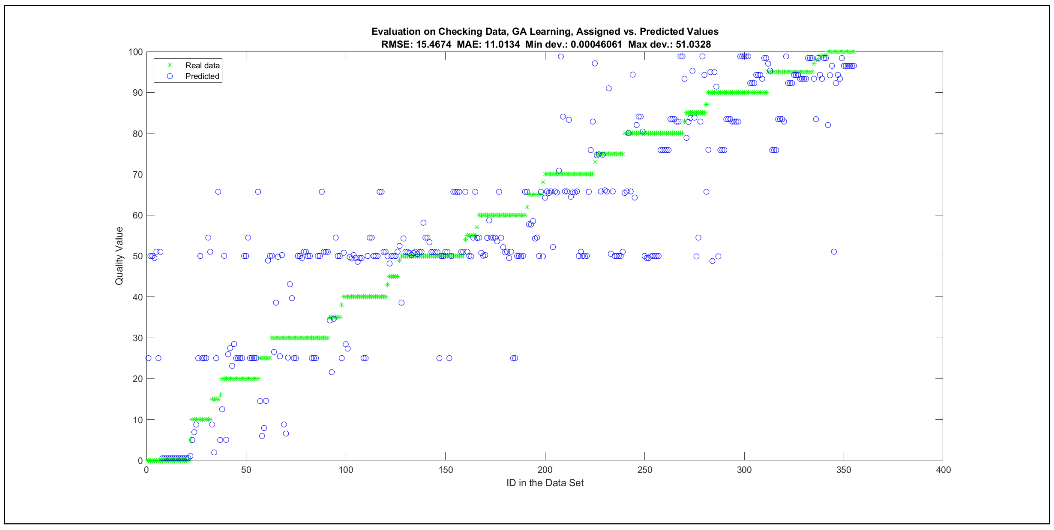

Figure C.33: Evaluation of the Best Performing 8-Rule Model with Regard to RMSE Error on Checking Data, Source: [68]

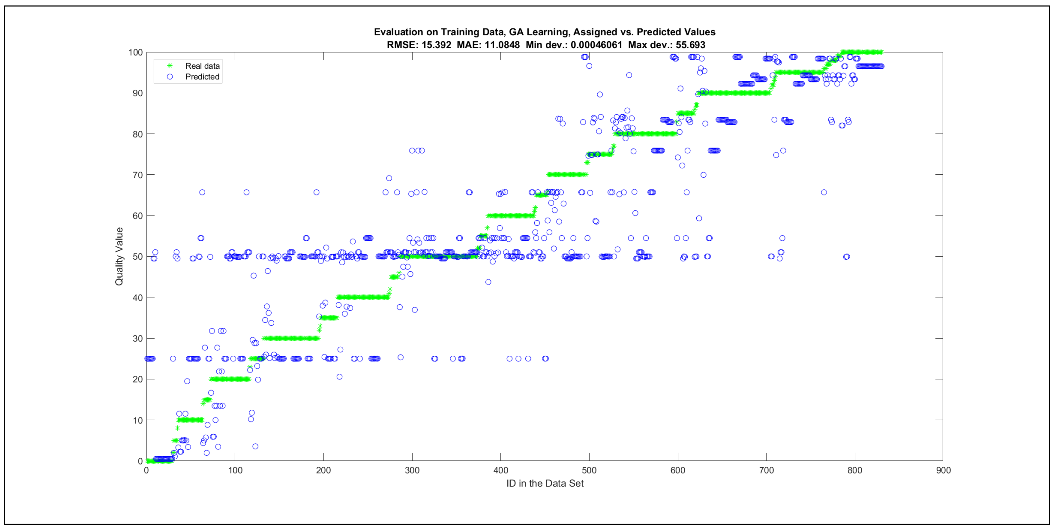

Figure C.34: Evaluation of the Best Performing 8-Rule Model with Regard to Maximum Error on Training Data, Source: [68]

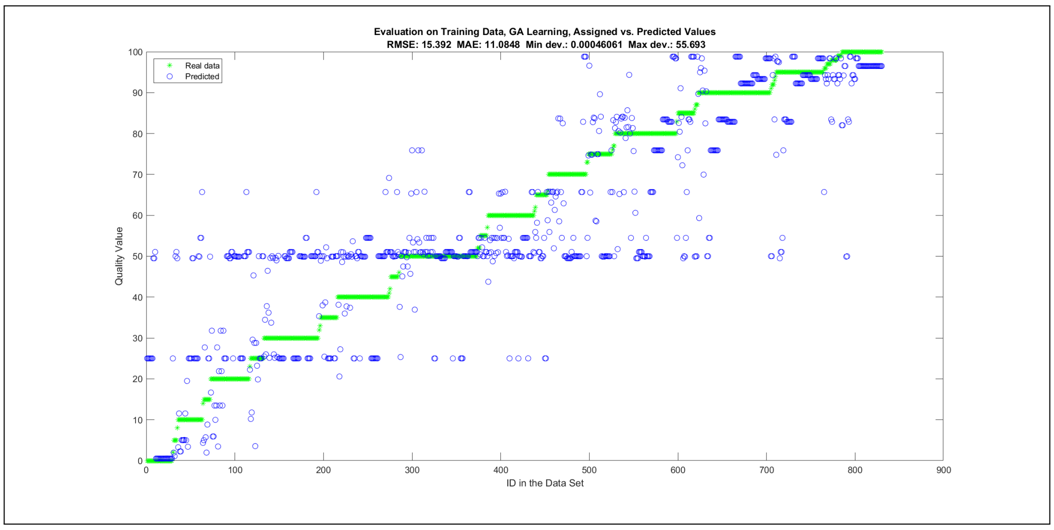

Figure C.35: Evaluation of the Best Performing 8-Rule Model with Regard to RMSE Error on Training Data, Source: [68]

C.3.5 Maximal Number of Linguistic Rules: 12

1. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)

2. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is good) and (Security is poor), then (Quality is medium)

3. If (Accuracy is poor) and (Legibility is poor) and (Security is medium), then (Quality is poor)

4. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is medium)

5. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is poor), then (Quality is very poor)

6. If (Accuracy is poor) and (Legibility is good) and (DesignAndImplementation is good) and (Security is good), then (Quality is poor)
7. If (Legibility is poor) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is poor)
8. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is poor)
9. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is good) and (Security is medium), then (Quality is very good)
10. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is poor)
11. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is medium)
12. If (Accuracy is medium) and (Legibility is poor) and (Security is medium), then (Quality is good)

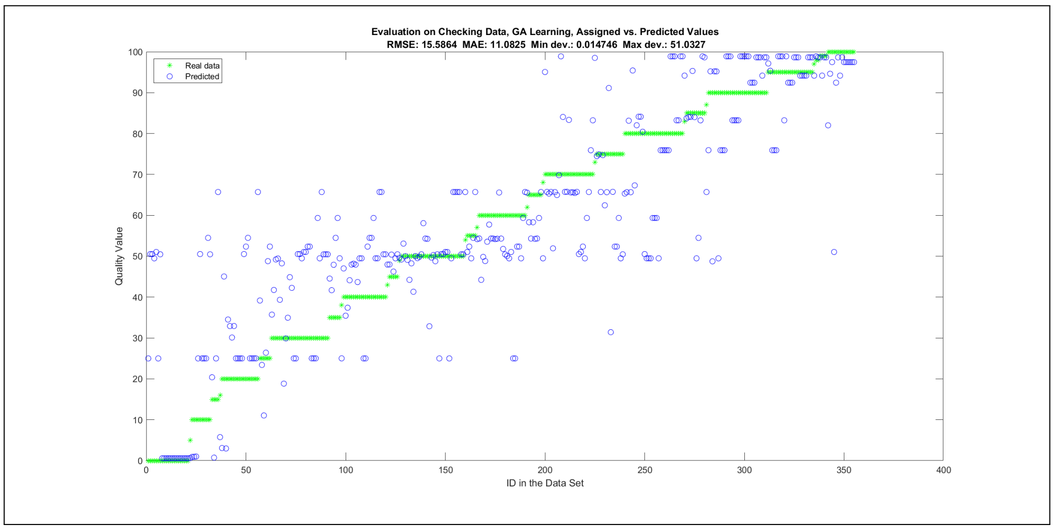

Figure C.36: Evaluation of the Best Performing 12-Rule Model with Regard to Maximum Error on Checking Data, Source: [68]

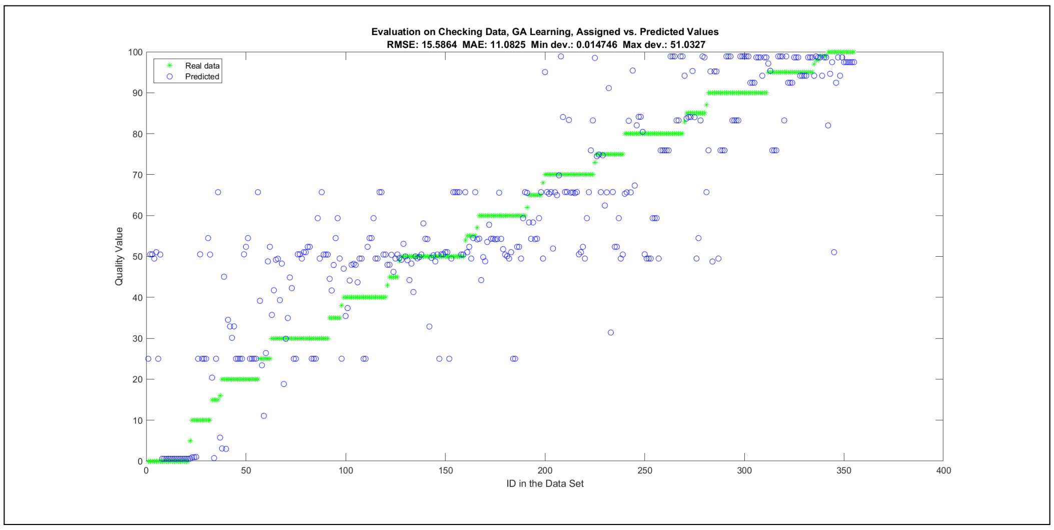

Figure C.37: Evaluation of the Best Performing 12-Rule Model with Regard to RMSE Error on Checking Data, Source: [68]

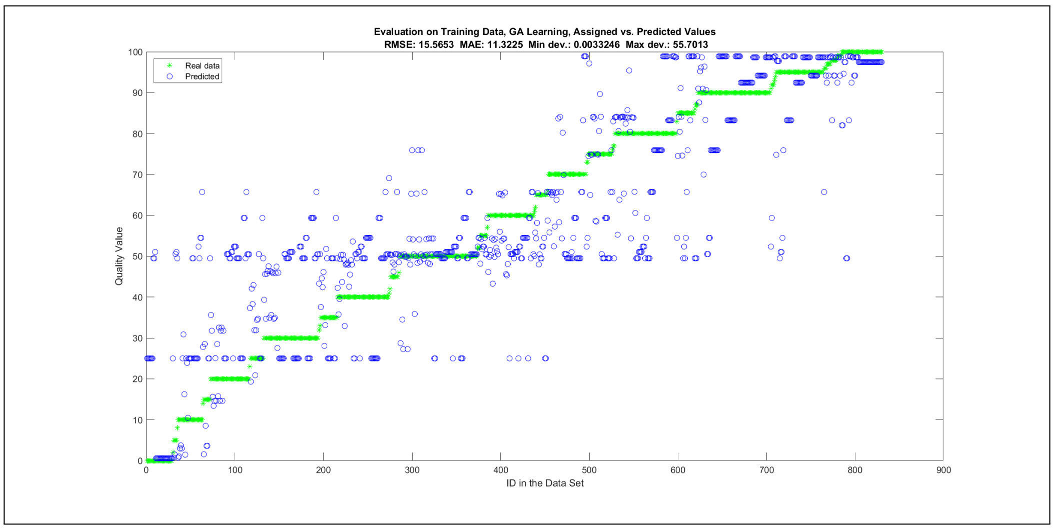

Figure C.38: Evaluation of the Best Performing 12-Rule Model with Regard to Maximum Error on Training Data, Source: [68]

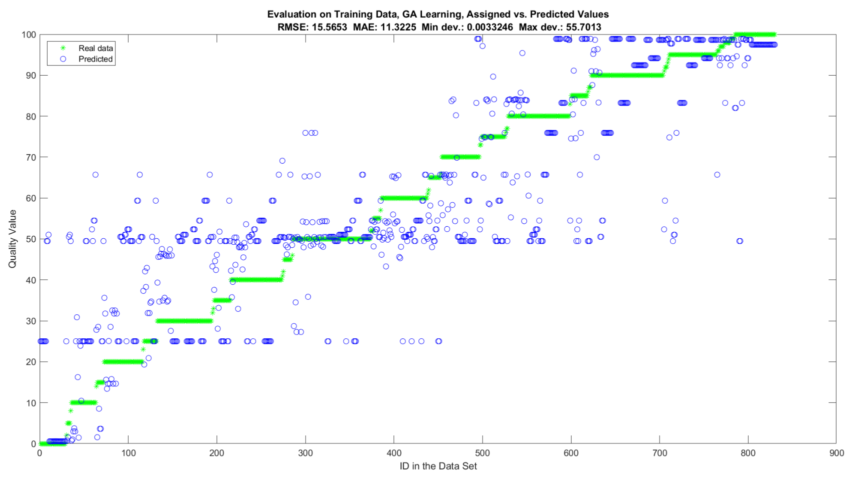

Figure C.39: Evaluation of the Best Performing 12-Rule Model with Regard to RMSE Error on Training Data, Source: [68]

C.3.6 Maximal Number of Linguistic Rules: 16

1. If (Legibility is good) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is good)
2. If (Accuracy is poor) and (Legibility is good) and (DesignAndImplementation is poor) and (Security is good), then (Quality is good)
3. If (Accuracy is medium) and (Legibility is good) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)
4. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is poor)
5. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is medium)
6. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is good) and (Security is medium), then (Quality is very good)
7. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is good) and (Security is poor), then (Quality is poor)
8. If (Legibility is poor) and (DesignAndImplementation is good) and (Security is medium), then (Quality is medium)
9. If (Legibility is poor) and (DesignAndImplementation is medium) and (Security is good), then (Quality is medium)
10. If (Accuracy is good) and (Legibility is good) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is good)
11. If (Accuracy is good) and (Legibility is medium) and (Security is medium), then (Quality is good)

12. If (Accuracy is poor) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)
13. If (Accuracy is medium) and (Legibility is good) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is very poor)
14. If (Accuracy is poor) and (Legibility is poor) and (Security is good), then (Quality is very poor)
15. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is medium), then (Quality is medium)
16. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is poor), then (Quality is poor)

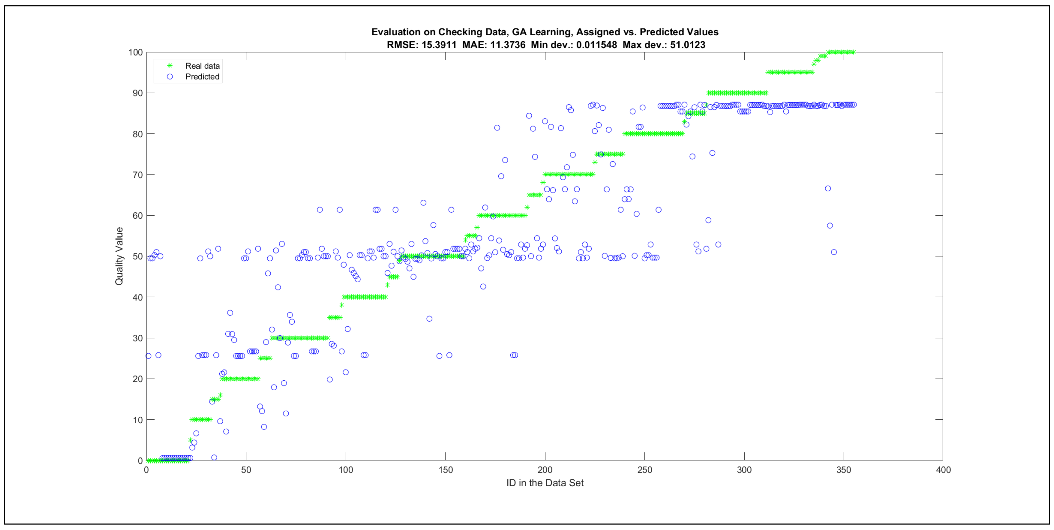

Figure C.40: Evaluation of the Best Performing 16-Rule Model with Regard to Maximum Error on Checking Data, Source: [68]

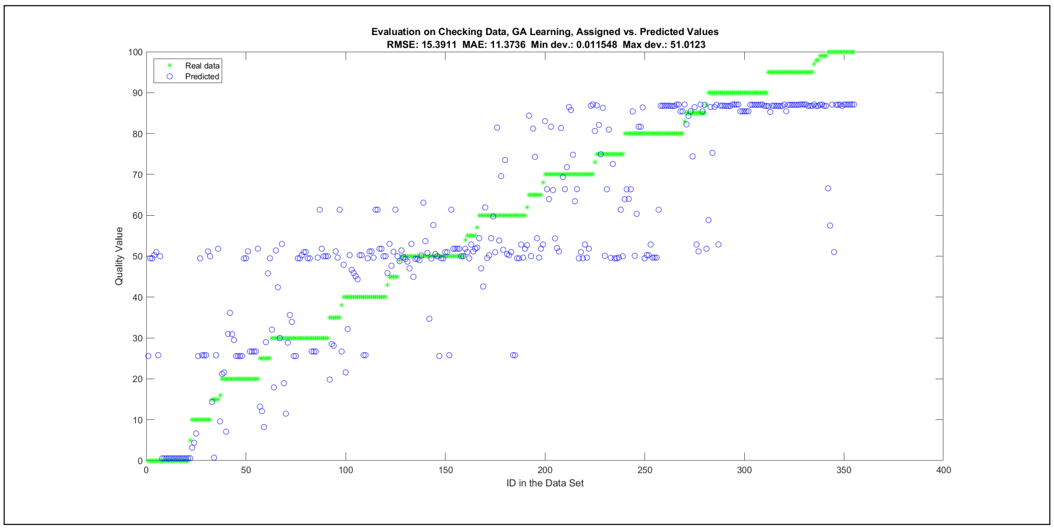

Figure C.41: Evaluation of the Best Performing 16-Rule Model with Regard to RMSE Error on Checking Data, Source: [68]

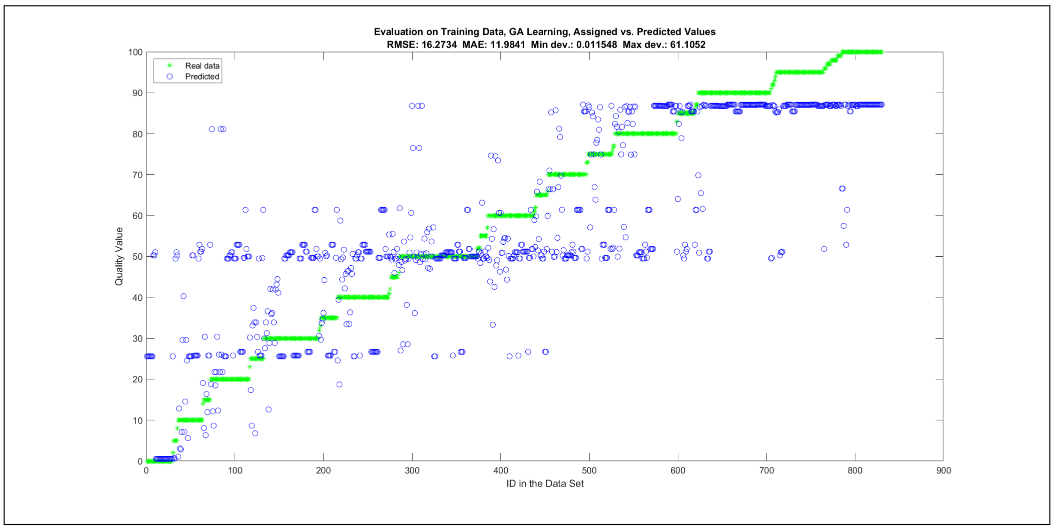

Figure C.42: Evaluation of the Best Performing 16-Rule Model with Regard to Maximum Error on Training Data, Source: [68]

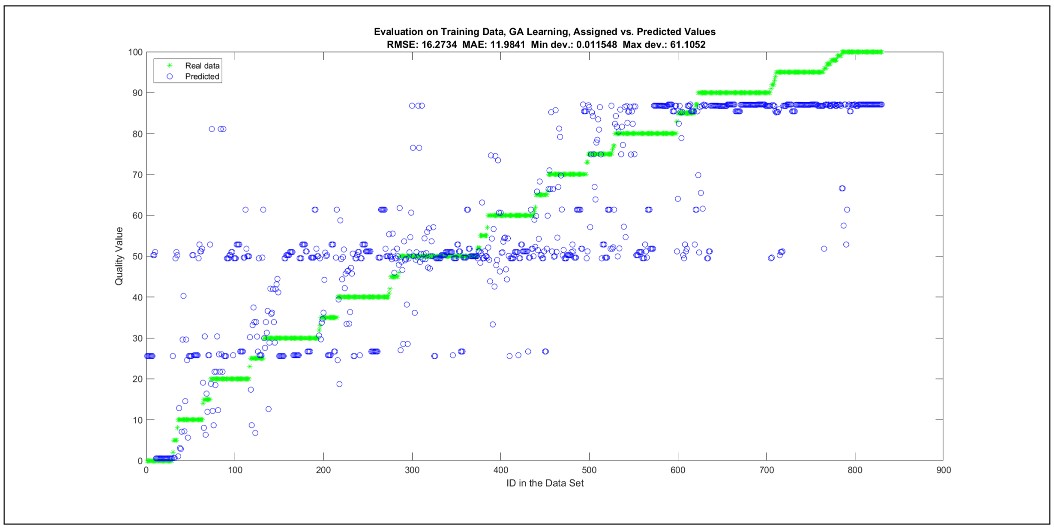

Figure C.43: Evaluation of the Best Performing 16-Rule Model with Regard to RMSE Error on Training Data, Source: [68]

C.3.7 Maximal Number of Linguistic Rules: 20

1. If (Accuracy is good) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is medium)
2. If (Legibility is poor) and (DesignAndImplementation is good), then (Quality is medium)
3. If (Accuracy is medium) and (Legibility is good) and (DesignAndImplementation is medium) and (Security is good), then (Quality is good)
4. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is poor), then (Quality is good)
5. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is medium)

6. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is good), then (Quality is good)
7. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is poor)
8. If (DesignAndImplementation is poor) and (Security is poor), then (Quality is poor)
9. If (Accuracy is medium) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is medium)
10. If (Accuracy is poor) and (Legibility is poor) and (Security is poor), then (Quality is very poor)
11. If (Accuracy is poor) and (DesignAndImplementation is good) and (Security is medium), then (Quality is very poor)
12. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is poor), then (Quality is good)
13. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is good)
14. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is good) and (Security is good), then (Quality is medium)
15. If (Accuracy is poor) and (DesignAndImplementation is poor), then (Quality is very poor)
16. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is medium), then (Quality is good)
17. If (Accuracy is good) and (Legibility is good) and (DesignAndImplementation is good) and (Security is medium), then (Quality is very good)
18. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is poor), then (Quality is poor)
19. If (Accuracy is good) and (Legibility is good) and (DesignAndImplementation is good) and (Security is poor), then (Quality is medium)

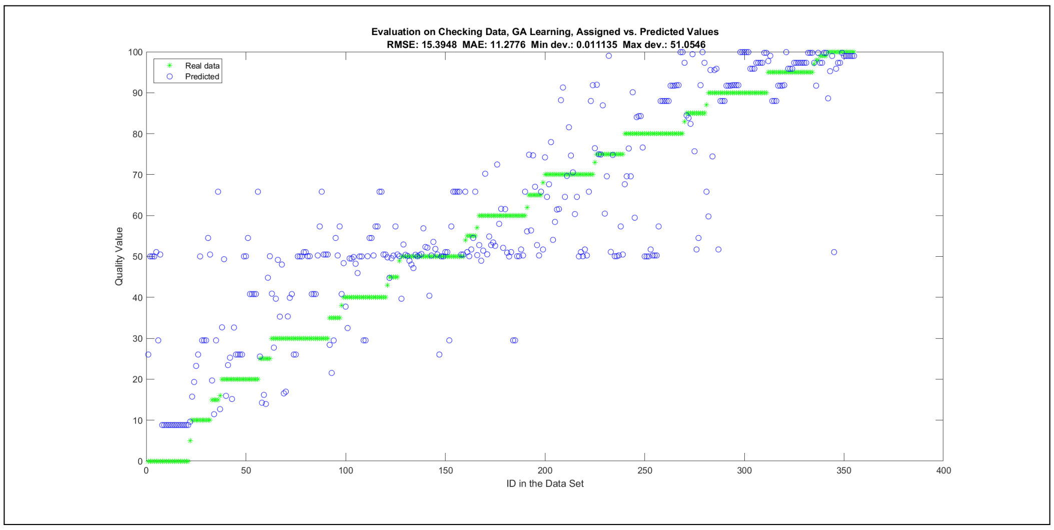

Figure C.44: Evaluation of the Best Performing Model with Regard to Maximum Error on Checking Data, with Upper Bound 20 Rules, Source: [68]

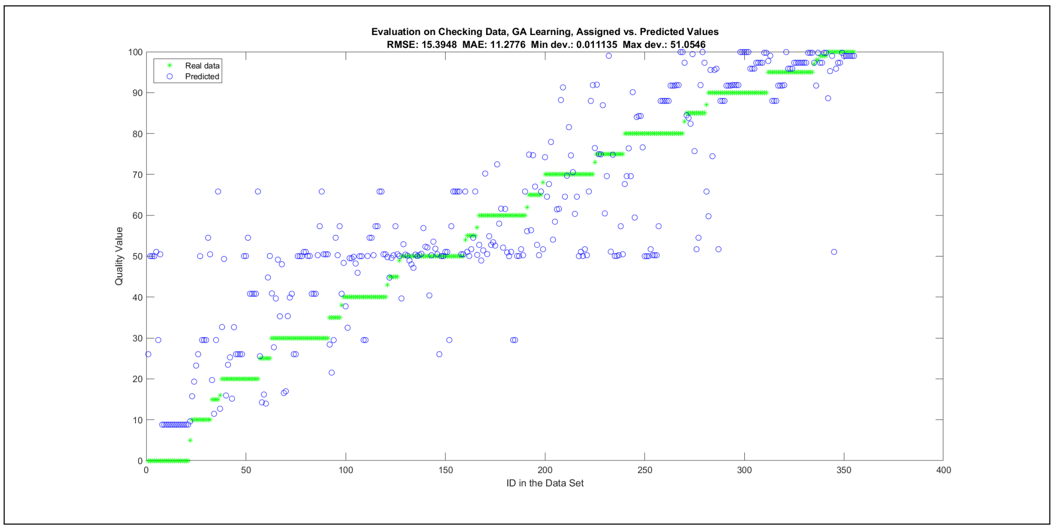

Figure C.45: Evaluation of the Best Performing Model with Regard to RMSE Error on Checking Data, with Upper Bound 20 Rules, Source: [68]

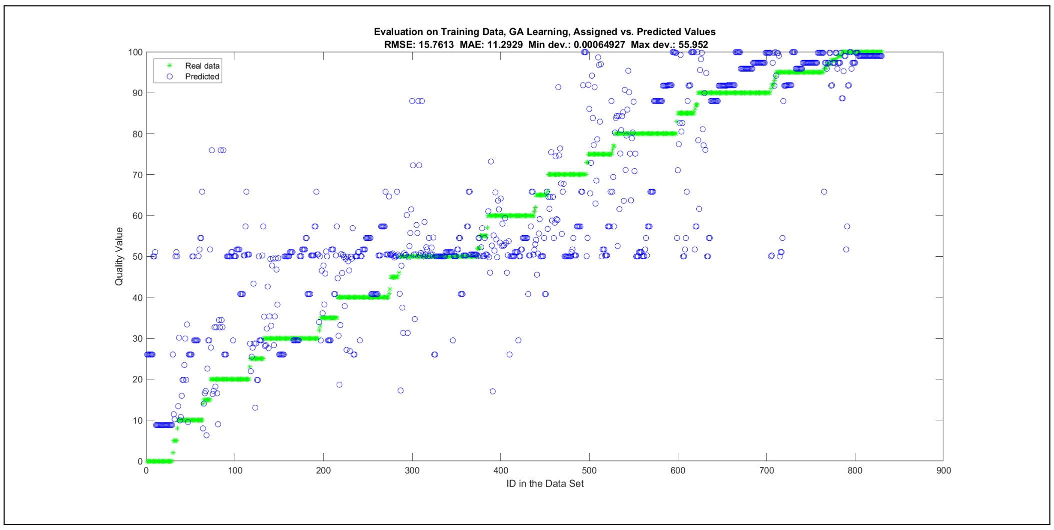

Figure C.46: Evaluation of the Best Performing Model with Regard to Maximum Error on Training Data, with Upper Bound 20 Rules, Source: [68]

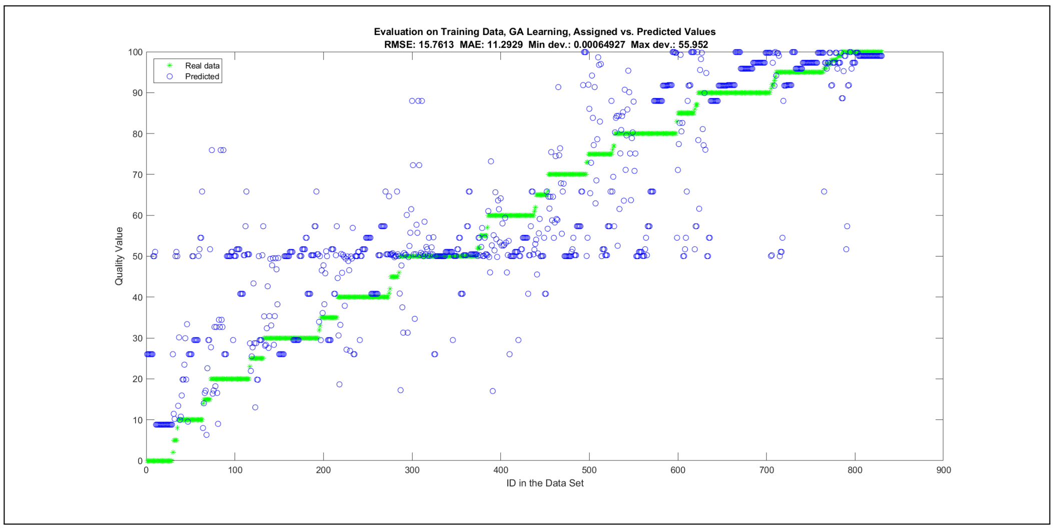

Figure C.47: Evaluation of the Best Performing Model with Regard to RMSE Error on Training Data, with Upper Bound 20 Rules, Source: [68]

C.3.8 Maximal Number of Linguistic Rules: 40

1. If (Accuracy is poor) and (Security is poor), then (Quality is very poor)
2. If (Legibility is good) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is good)
3. If (Accuracy is good) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is poor)
4. If (Legibility is poor) and (DesignAndImplementation is good) and (Security is poor), then (Quality is very poor)
5. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is good)

6. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is good), then (Quality is good)
7. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is good) and (Security is poor), then (Quality is poor)
8. If (Accuracy is poor) and (Legibility is good) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is good)
9. If (Accuracy is good) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is very good)
10. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is poor), then (Quality is very good)
11. If (Accuracy is medium) and (DesignAndImplementation is medium) and (Security is good), then (Quality is medium)
12. If (Accuracy is poor) and (Legibility is good) and (DesignAndImplementation is medium), then (Quality is medium)
13. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is poor)
14. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is poor), then (Quality is medium)
15. If (Accuracy is good) and (Legibility is good) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is good)
16. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is good)
17. If (Accuracy is poor) and (Legibility is good) and (DesignAndImplementation is good) and (Security is good), then (Quality is poor)
18. If (Accuracy is medium) and (Legibility is good) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is medium)
19. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is good)
20. If (Accuracy is medium) and (Legibility is good) and (Security is good), then (Quality is very good)
21. If (Accuracy is medium) and (Legibility is poor), then (Quality is medium)
22. If (Legibility is poor) and (DesignAndImplementation is poor) and (Security is good), then (Quality is poor)
23. If (Accuracy is good) and (Legibility is good) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is very poor)

24. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is very good)

25. If (Accuracy is medium) and (Legibility is good), then (Quality is good)

26. If (Accuracy is medium) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is poor)

27. If (Accuracy is poor) and (Legibility is good) and (Security is good), then (Quality is very poor)

28. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is poor), then (Quality is poor)

29. If (Legibility is poor) and (Security is poor), then (Quality is very poor)

30. If (Accuracy is good) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is medium)

31. If (Accuracy is medium) and (Legibility is good) and (DesignAndImplementation is poor) and (Security is good), then (Quality is good)

32. If (Accuracy is poor) and (Legibility is poor) and (Security is good), then (Quality is poor)

33. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is poor)

34. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is medium)

35. If (DesignAndImplementation is poor) and (Security is medium), then (Quality is very poor)

36. If (Accuracy is good) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is poor)

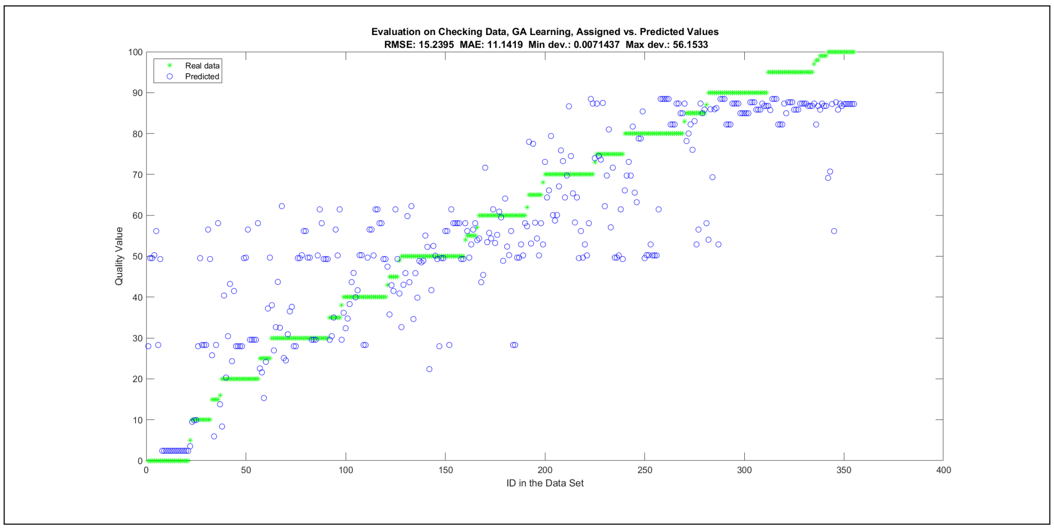

Figure C.48: Evaluation of the Best Performing Model with Regard to Maximum Error on Checking Data, with Upper Bound 40 Rules, Source: [68]

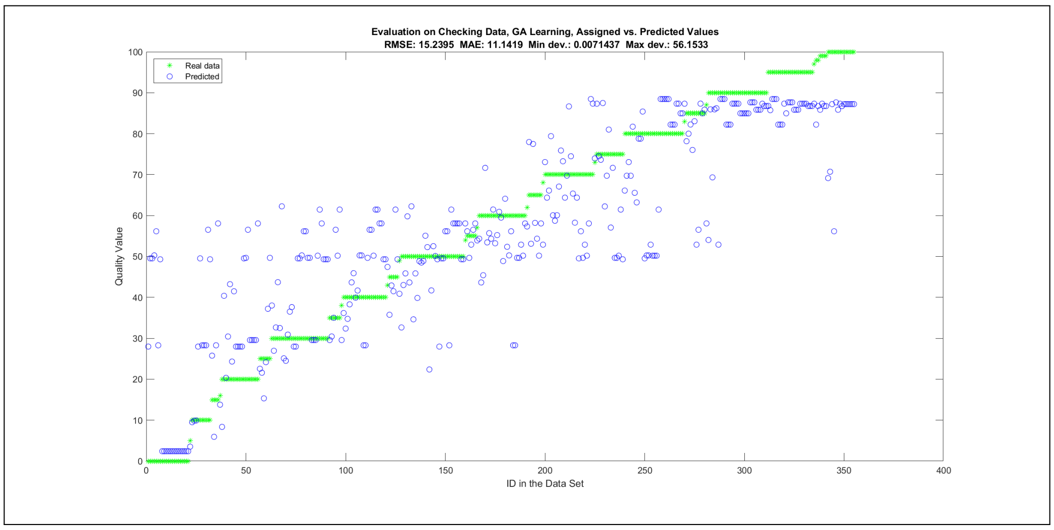

Figure C.49: Evaluation of the Best Performing Model with Regard to RMSE Error on Checking Data, with Upper Bound 40 Rules, Source: [68]

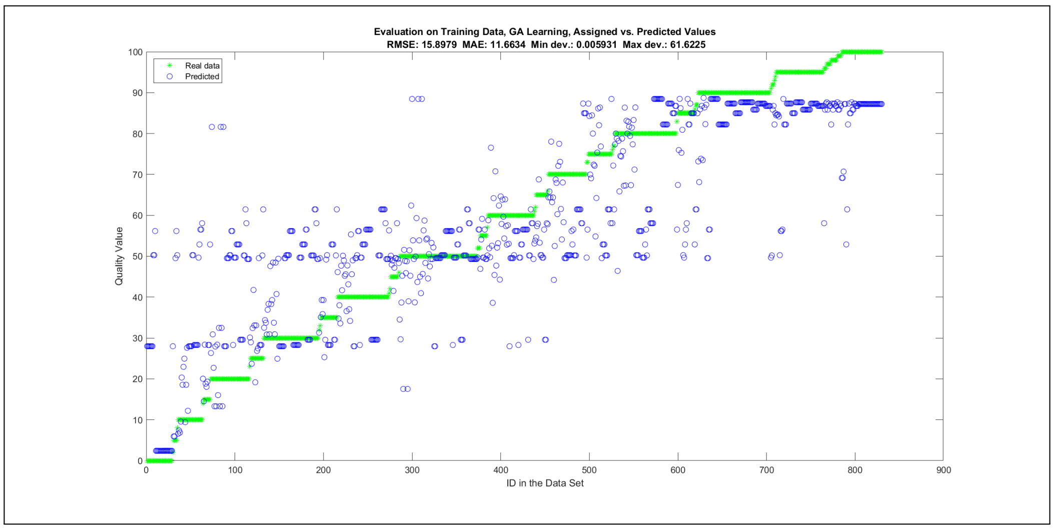

Figure C.50: Evaluation of the Best Performing Model with Regard to Maximum Error on Training Data, with Upper Bound 40 Rules, Source: [68]

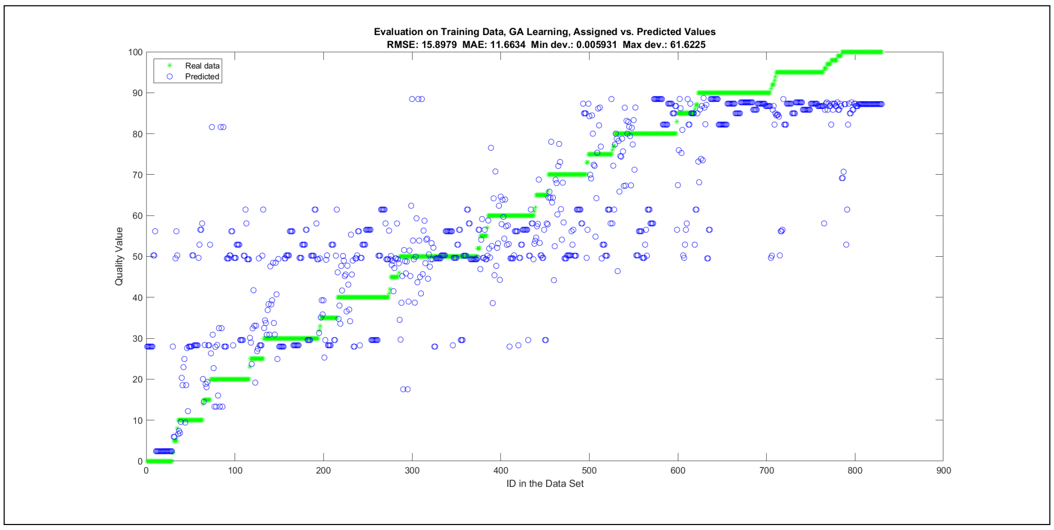

Figure C.51: Evaluation of the Best Performing Model with Regard to RMSE Error on Training Data, with Upper Bound 40 Rules, Source: [68]

C.3.9 Maximal Number of Linguistic Rules: 80

1. If (Accuracy is good) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is poor), then (Quality is very poor)
2. If (Accuracy is medium) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is very poor)
3. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)
4. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)
5. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is good), then (Quality is poor)
6. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is good), then (Quality is very poor)
7. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is very poor)
8. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is poor)
9. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is very good)
10. If (Accuracy is poor) and (Legibility is good) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is poor)
11. If (Accuracy is poor) and (Legibility is good) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is poor)

12. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is medium)
13. If (Accuracy is poor) and (Legibility is good) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is very poor)
14. If (Accuracy is poor) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)
15. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is poor)
16. If (Accuracy is medium) and (Legibility is good) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)
17. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is very poor)
18. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is good)
19. If (Legibility is poor) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is very poor)
20. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is good) and (Security is medium), then (Quality is medium)
21. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is good)
22. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is good) and (Security is poor), then (Quality is medium)
23. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is poor)
24. If (Accuracy is poor) and (DesignAndImplementation is good) and (Security is medium), then (Quality is medium)
25. If (Accuracy is medium) and (Legibility is good) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is very good)
26. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is poor), then (Quality is good)
27. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is good) and (Security is poor), then (Quality is very poor)
28. If (Accuracy is poor) and (Legibility is good) and (Security is good), then (Quality is poor)
29. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is medium), then (Quality is medium)

30. If (Legibility is medium) and (DesignAndImplementation is good) and (Security is medium), then (Quality is very good)

31. If (Accuracy is good) and (Legibility is good) and (DesignAndImplementation is poor) and (Security is good), then (Quality is medium)

32. If (Accuracy is good) and (Legibility is poor), then (Quality is medium)

33. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is poor)

34. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is poor)

35. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is medium), then (Quality is very poor)

36. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is good), then (Quality is very poor)

37. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is good)

38. If (Accuracy is poor) and (Legibility is good) and (DesignAndImplementation is good) and (Security is poor), then (Quality is medium)

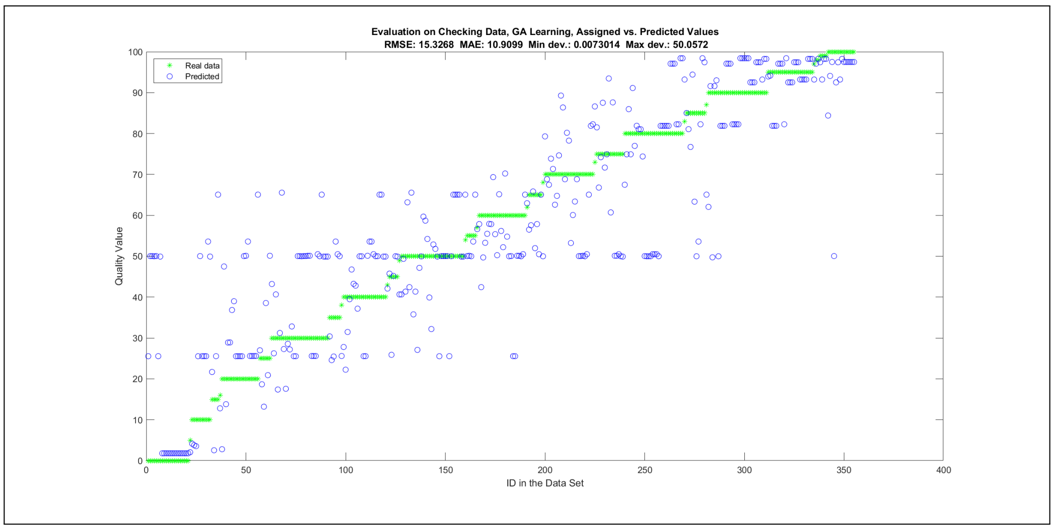

Figure C.52: Evaluation of the Best Performing Model with Regard to Maximum Error on Checking Data, with Upper Bound 80 Rules, Source: [68]

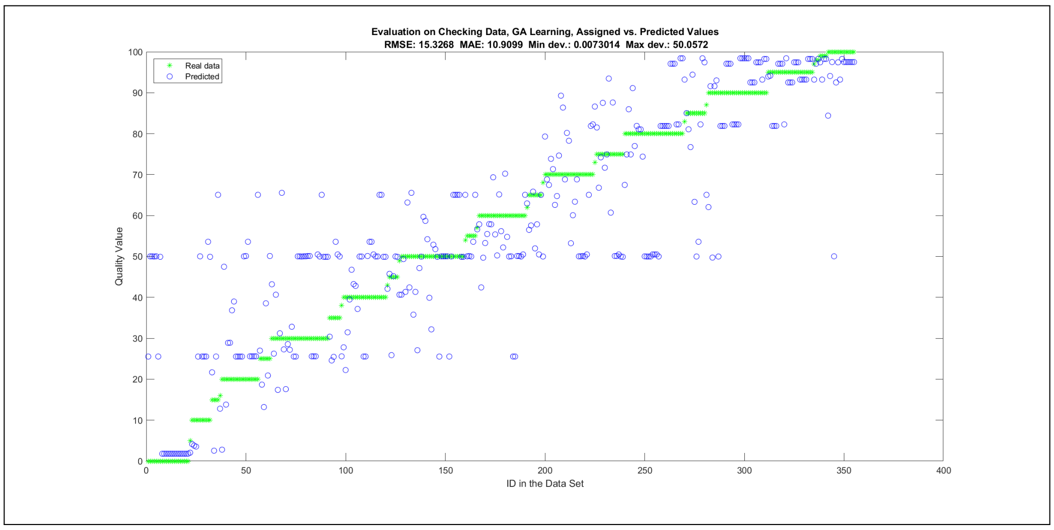

Figure C.53: Evaluation of the Best Performing Model with Regard to RMSE Error on Checking Data, with Upper Bound 80 Rules, Source: [68]

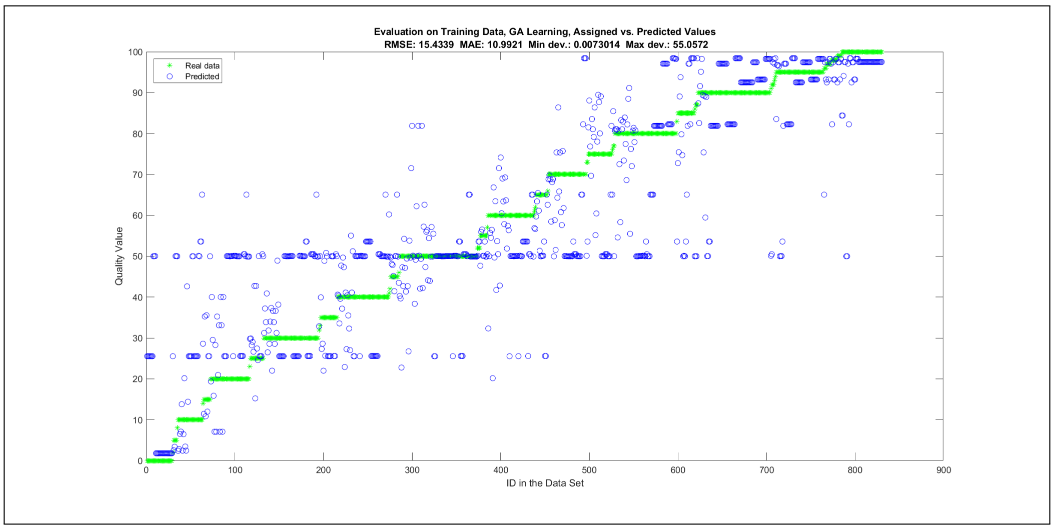

Figure C.54: Evaluation of the Best Performing Model with Regard to Maximum Error on Training Data, with Upper Bound 80 Rules, Source: [68]

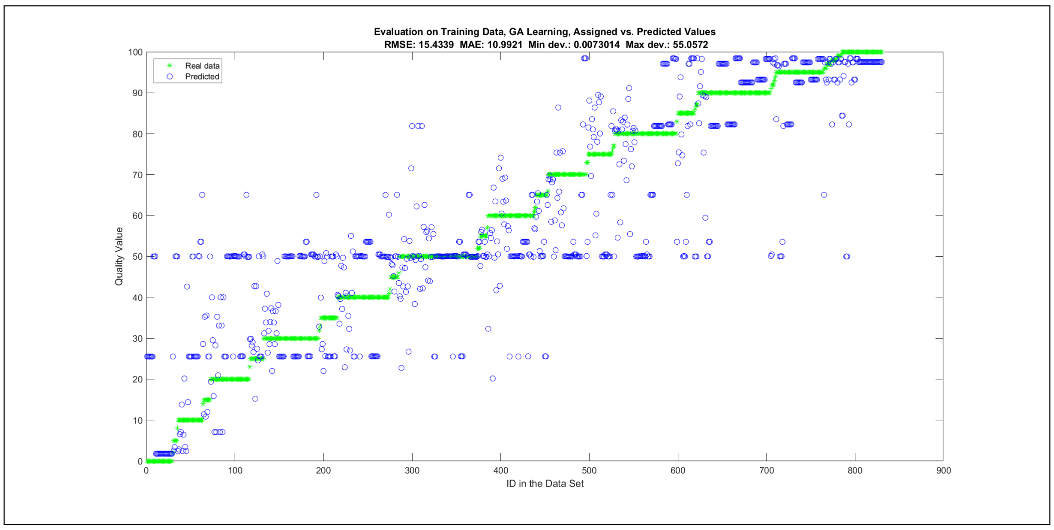

Figure C.55: Evaluation of the Best Performing Model with Regard to RMSE Error on Training Data, with Upper Bound 80 Rules, Source: [68]

C.3.10 Maximal Number of Linguistic Rules: 160

1. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is good) and (Security is poor), then (Quality is poor)
2. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is very poor)
3. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is poor)
4. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is good), then (Quality is medium)
5. If (Accuracy is poor) and (Legibility is good) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is very poor)
6. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is poor)
7. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is poor)
8. If (Accuracy is good) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is poor)
9. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)
10. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)
11. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is medium)

12. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is poor), then (Quality is very poor)
13. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is poor)
14. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is medium)
15. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is very poor)
16. If (Accuracy is poor) and (Legibility is good) and (DesignAndImplementation is good) and (Security is medium), then (Quality is poor)
17. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is medium), then (Quality is very poor)
18. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is medium), then (Quality is poor)
19. If (Accuracy is medium) and (Legibility is good) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is very good)
20. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is very poor)
21. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)
22. If (Accuracy is good) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)
23. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is medium)
24. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is good)
25. If (Accuracy is medium) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is poor)
26. If (Accuracy is good) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is poor), then (Quality is very poor)
27. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is poor), then (Quality is very poor)
28. If (Accuracy is poor) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is good), then (Quality is very poor)
29. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is medium), then (Quality is poor)

30. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is poor) and (Security is good), then (Quality is poor)

31. If (Accuracy is medium) and (DesignAndImplementation is medium) and (Security is poor), then (Quality is poor)

32. If (Accuracy is medium) and (Legibility is poor) and (DesignAndImplementation is medium) and (Security is good), then (Quality is medium)

33. If (Accuracy is good) and (Legibility is poor) and (DesignAndImplementation is good) and (Security is medium), then (Quality is medium)

34. If (Accuracy is poor) and (Legibility is medium) and (DesignAndImplementation is poor) and (Security is good), then (Quality is poor)

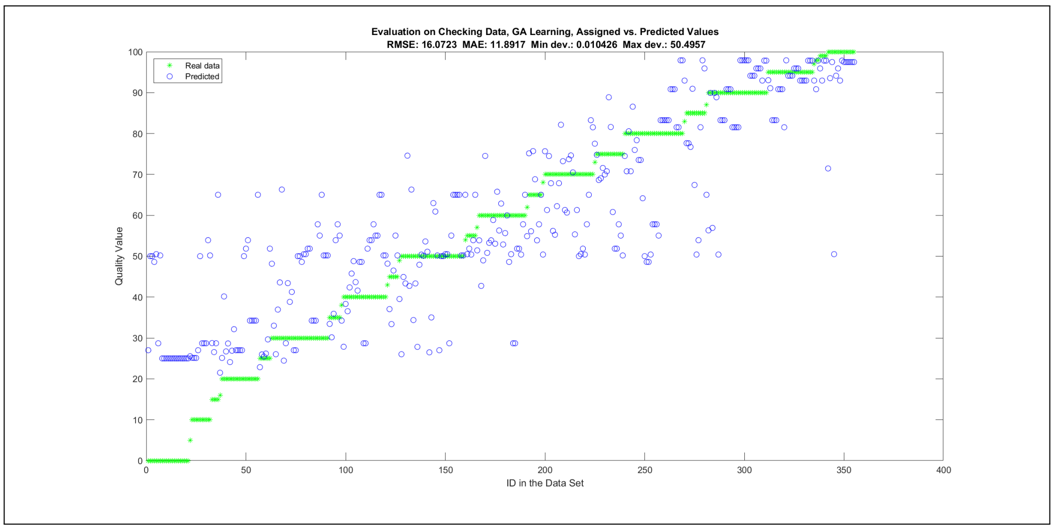

Figure C.56: Evaluation of the Best Performing Model with Regard to Maximum Error on Checking Data, with Upper Bound 160 Rules, Source: [68]

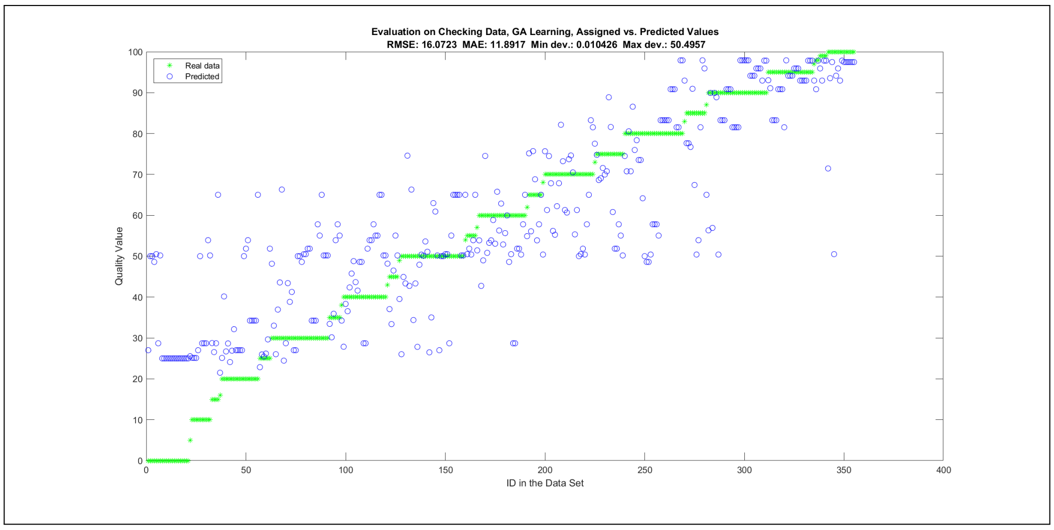

Figure C.57: Evaluation of the Best Performing Model with Regard to RMSE Error on Checking Data, with Upper Bound 160 Rules, Source: [68]

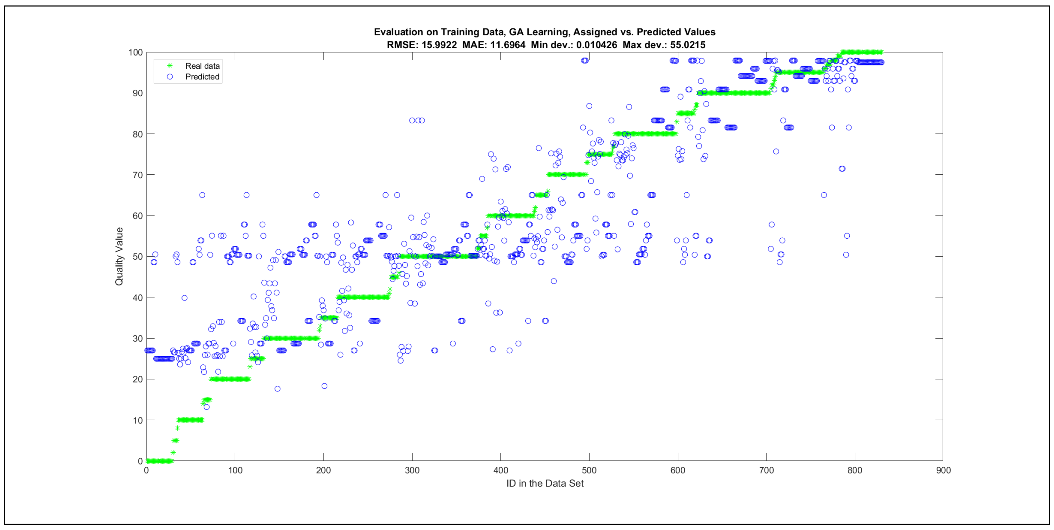

Figure C.58: Evaluation of the Best Performing Model with Regard to Maximum Error on Training Data, with Upper Bound 160 Rules, Source: [68]

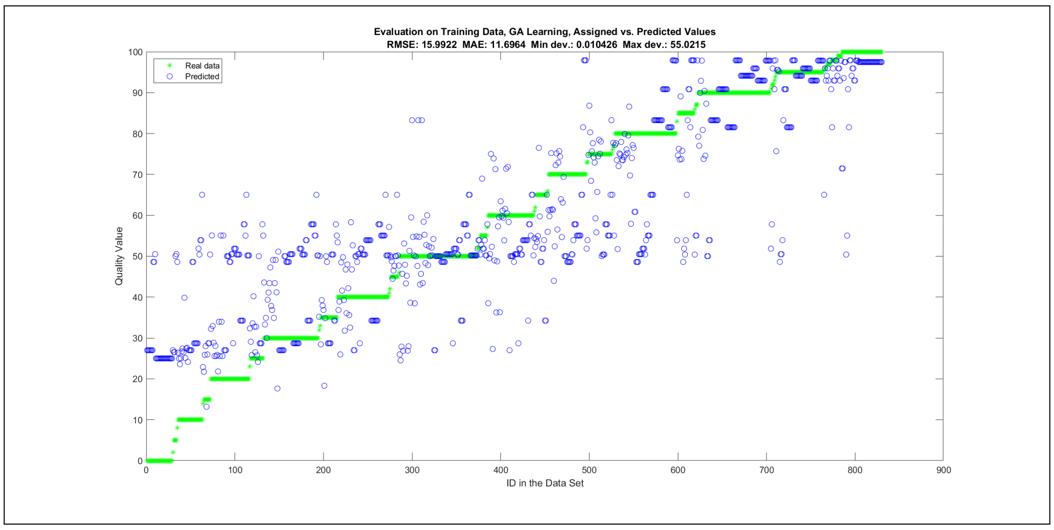

Figure C.59: Evaluation of the Best Performing Model with Regard to RMSE Error on Training Data, with Upper Bound 160 Rules, Source: [68]

C.4 Evaluation Charts I.

The evaluation charts show the pairwise effect of the input variables on the output before the adjustments of the mini focus group.

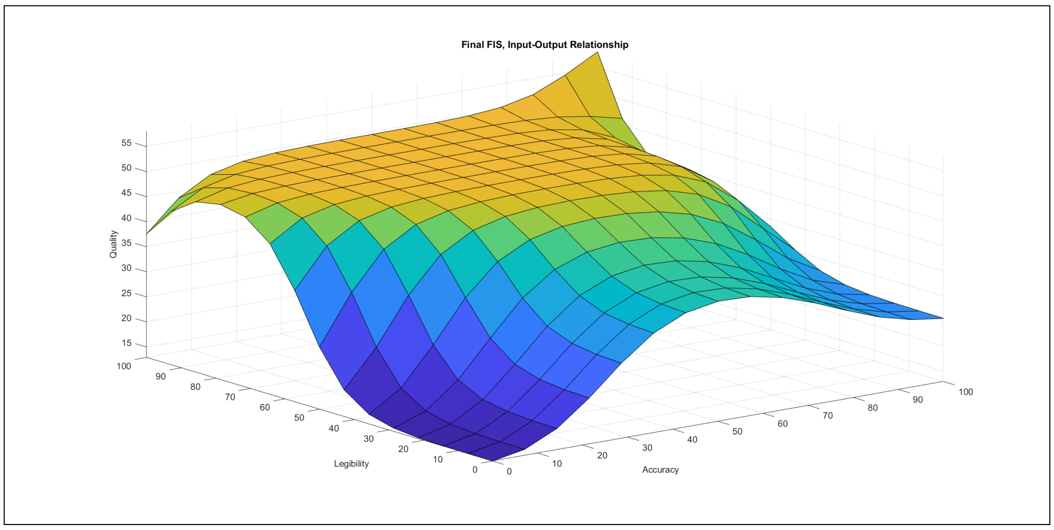

Figure C.60: Effect of the Input Accuracy and Legibility on Execution Tracing Quality, Source: [68]

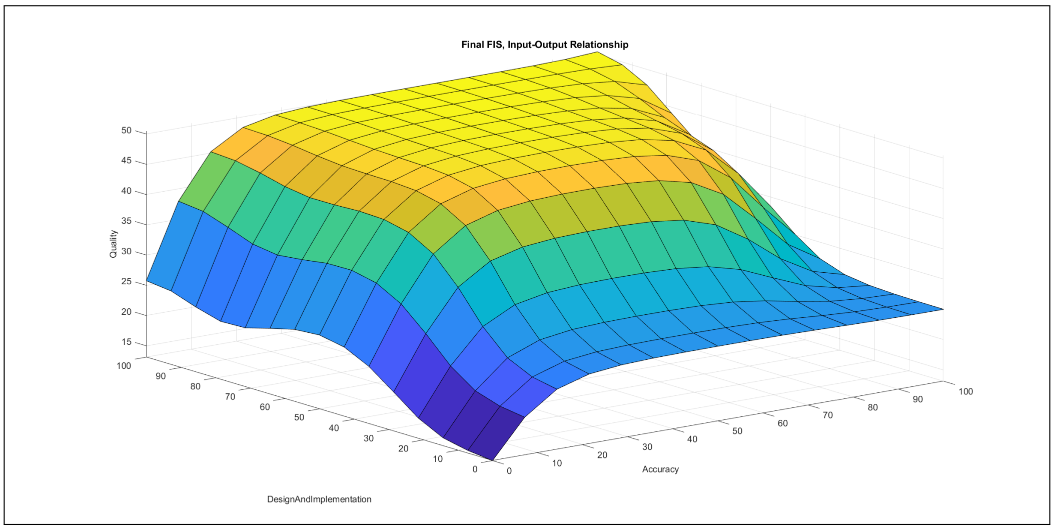

Figure C.61: Effect of the Input Accuracy and Design and Implementation on Execution Tracing Quality, Source: [68]

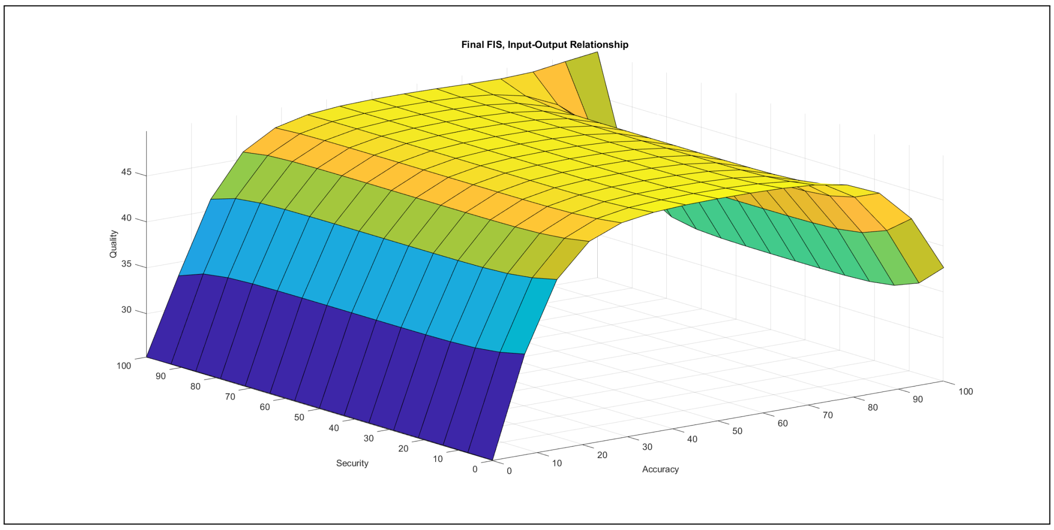

Figure C.62: Effect of the Input Accuracy and Security on Execution Tracing Quality, Source: [68]

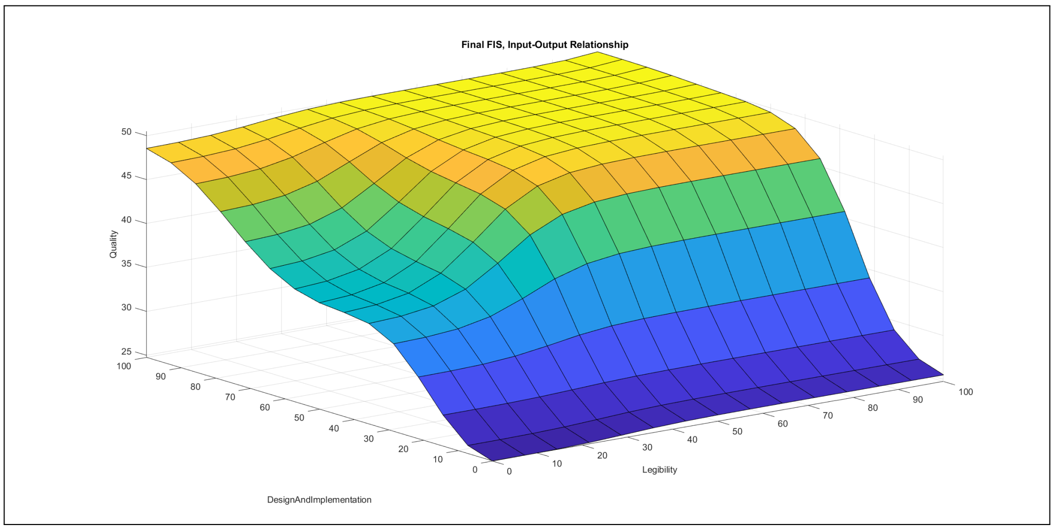

Figure C.63: Effect of the Input Legibility and Design and Implementation on Execution Tracing Quality, Source: [68]

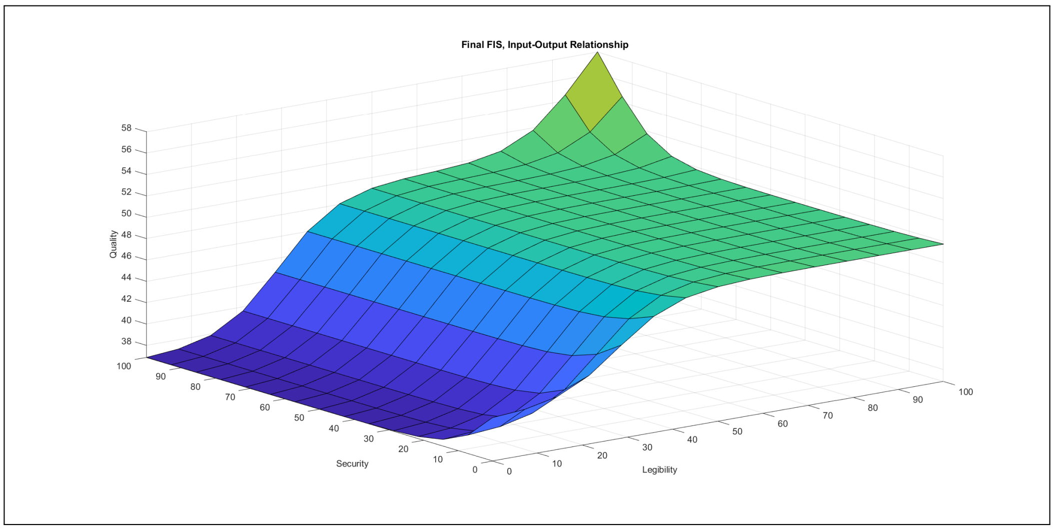

Figure C.64: Effect of the Input Legibility and Security on Execution Tracing Quality, Source: [68]

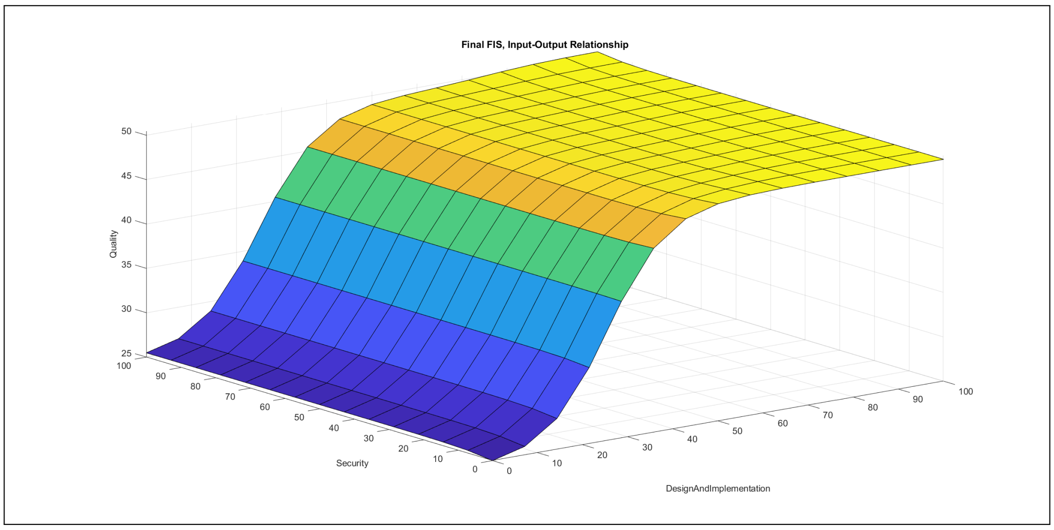

Figure C.65: Effect of the Input Design and Implementation and Security on Execution Tracing Quality, Source: [68]

C.5 Evaluation Charts II.

The evaluation charts show the pairwise effect of the input variables on the output after carrying out the adjustments in the mini focus group.

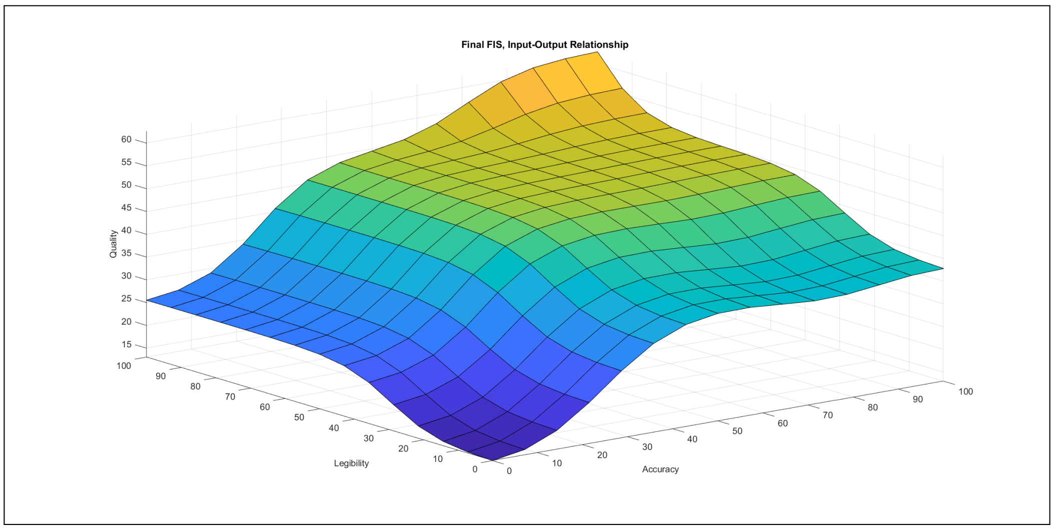

Figure C.66: Effect of the Input Accuracy and Legibility on Execution Tracing Quality After Adjusting, Source: [68]

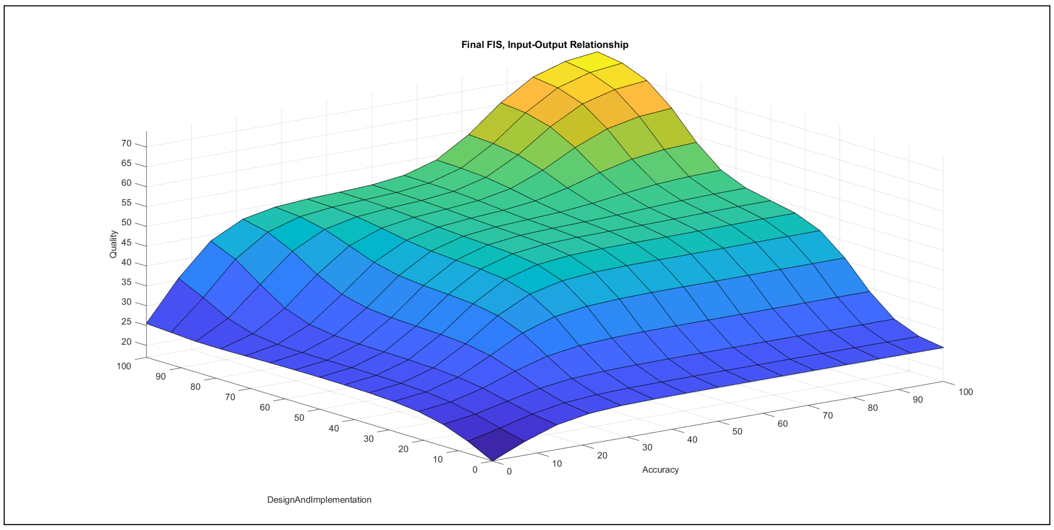

Figure C.67: Effect of the Input Accuracy and Design and Implementation on Execution Tracing Quality After Adjusting, Source: [68]

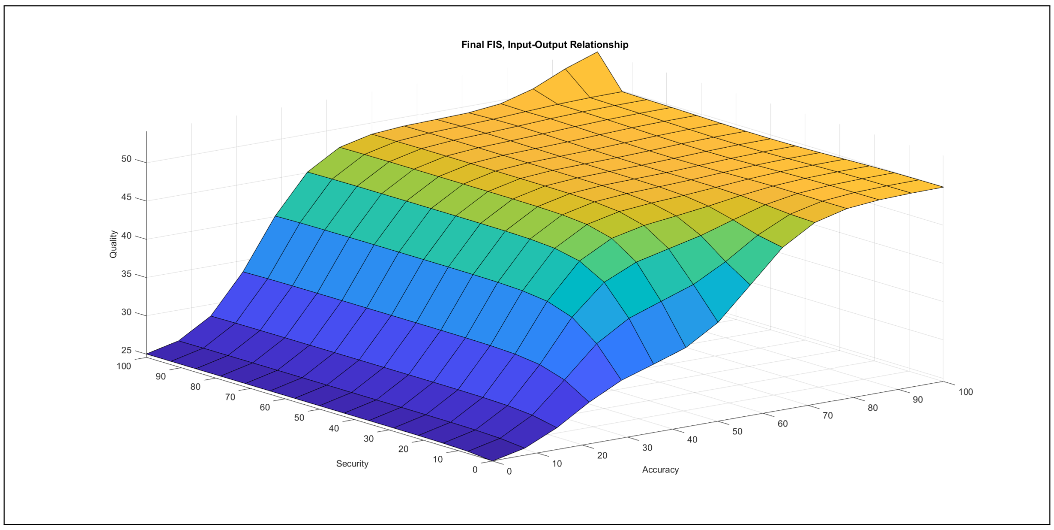

Figure C.68: Effect of the Input Accuracy and Security on Execution Tracing Quality After Adjusting, Source: [68]

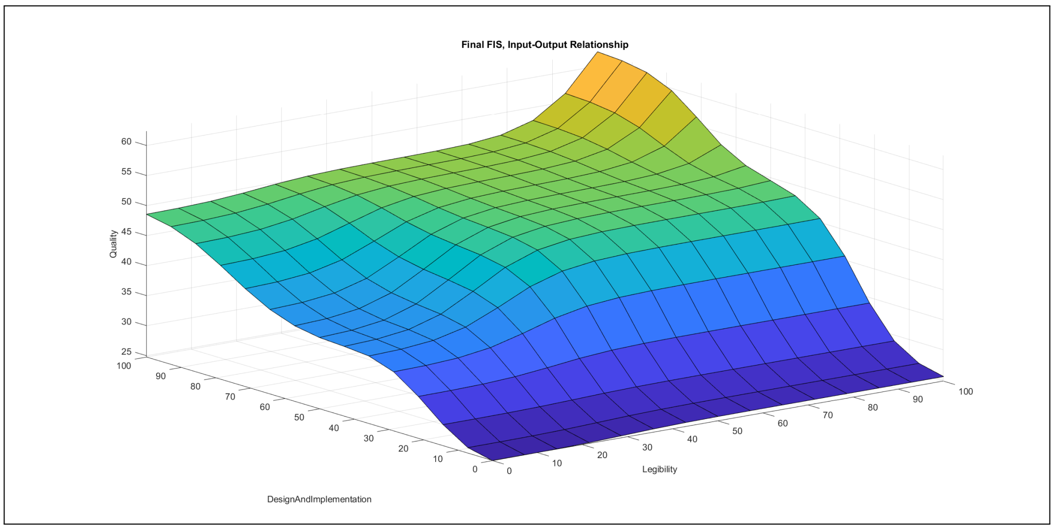

Figure C.69: Effect of the Input Legibility and Design and Implementation on Execution Tracing Quality After Adjusting, Source: [68]

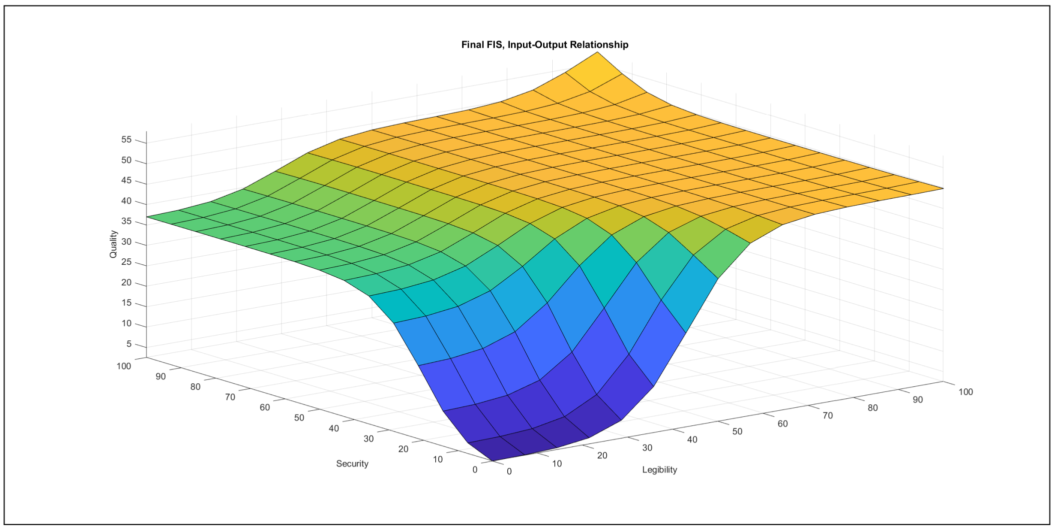

Figure C.70: Effect of the Input Legibility and Security on Execution Tracing Quality After Adjusting, Source: [68]

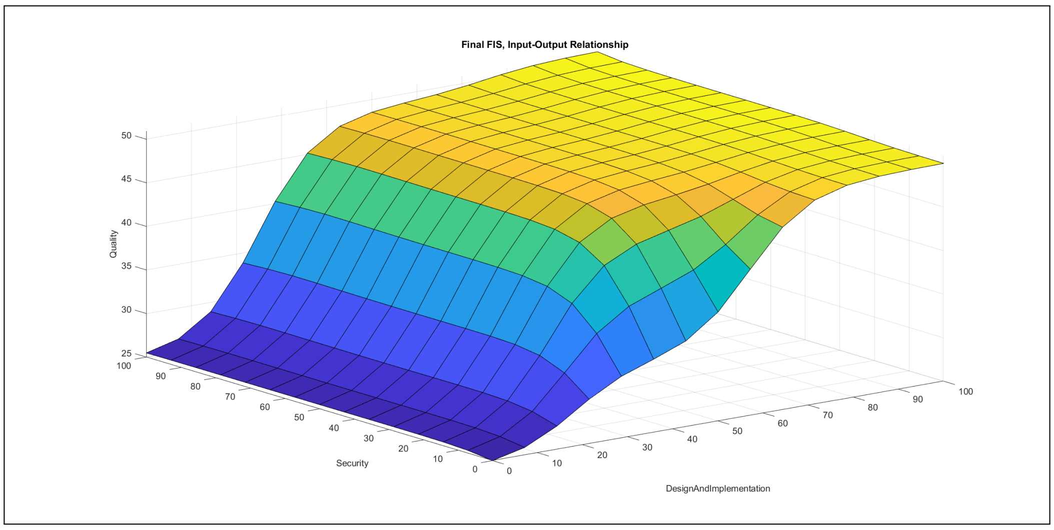

Figure C.71: Effect of the Input Design and Implementation and Security on Execution Tracing Quality After Adjusting, Source: [68]

C.6 The ANFIS Approach

This section presents the training results of the ANFIS approach performed to verify the results of the genetic learning. In the scope of both the (1) back-propagation and (2) hybrid approaches, a separate model has been trained with one of the initial step sizes from the set: $\{0.1, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5\}$. The best-performing models with regard to the maximal errors and RMSE errors were selected in the course of 100 learning epochs and are demonstrated below in figures C.72, C.73, C.74, and C.75.

C.6.1 Evaluation on the Checking Data

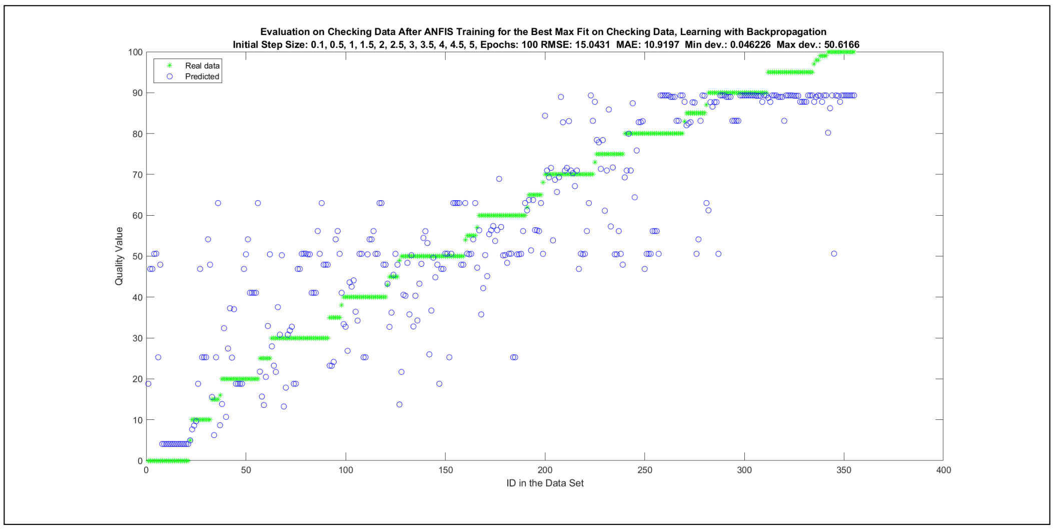

Figure C.72: Best Maximal-Error Model Trained by ANFIS with Back-propagation and Initial Step Size 2.5, Source: [68]

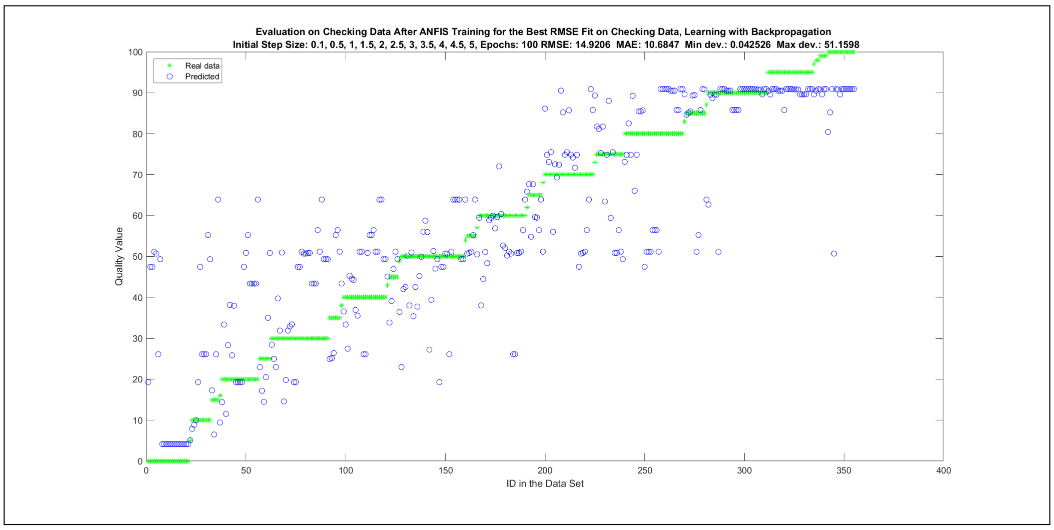

Figure C.73: Best RMSE-Error Model Trained by ANFIS with Back-propagation and Initial Step Size 3, Source: [68]

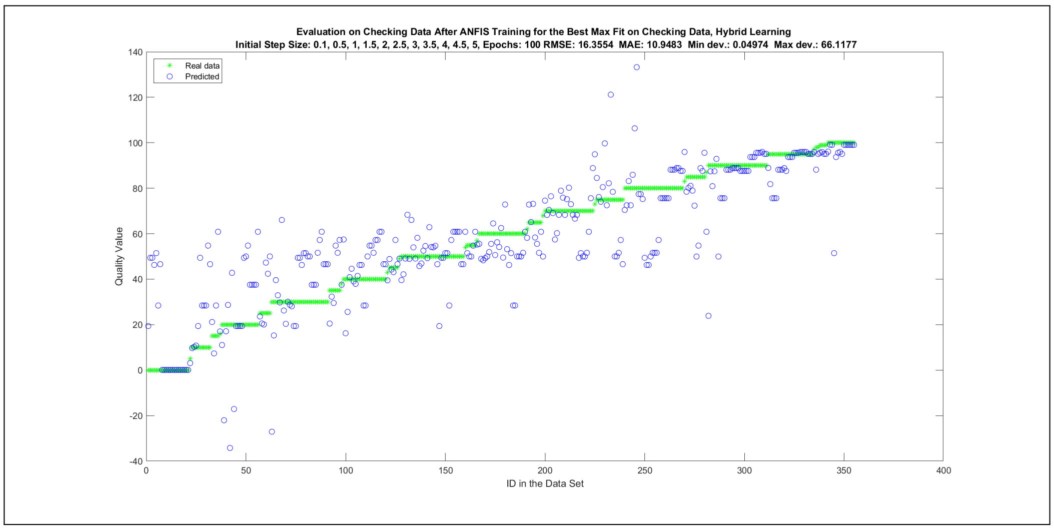

Figure C.74: Best Maximal-Error Model Trained by ANFIS with Hybrid Approach and Initial Step Size 3.5, Source: [68]

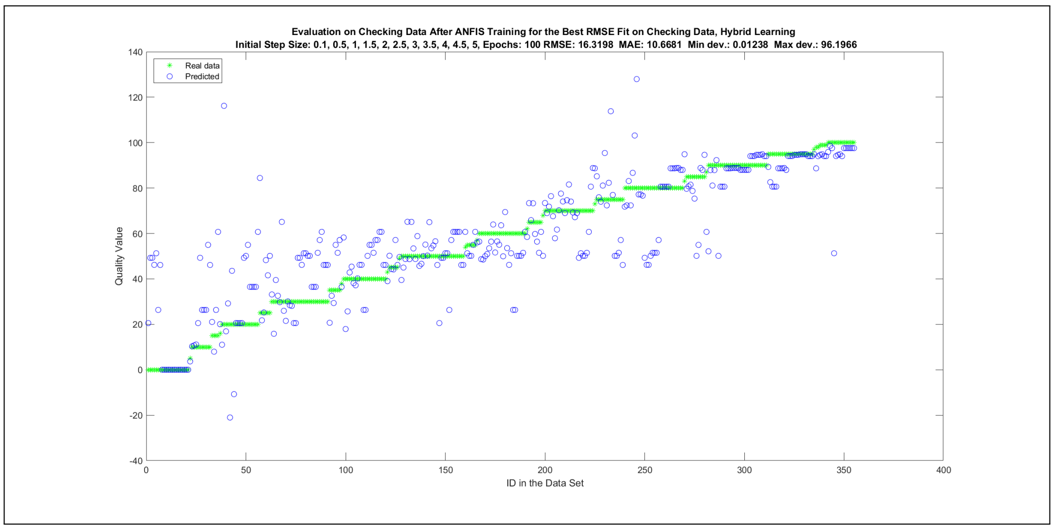

Figure C.75: Best RMSE-Error Model Trained by ANFIS with Hybrid Approach and Initial Step Size 5, Source: [68]

C.6.2 The Change of the RMSE Indicator as a Function of the Training Epochs

The change of the RMSE error indicator are depicted in figures C.76, C.78, C.80 and C.82 to highlight the convergence of the error on the training and on the checking data. In addition, the adaptation of the initial step sizes during the learning process are depicted in figures C.77, C.79, C.81 and C.83.

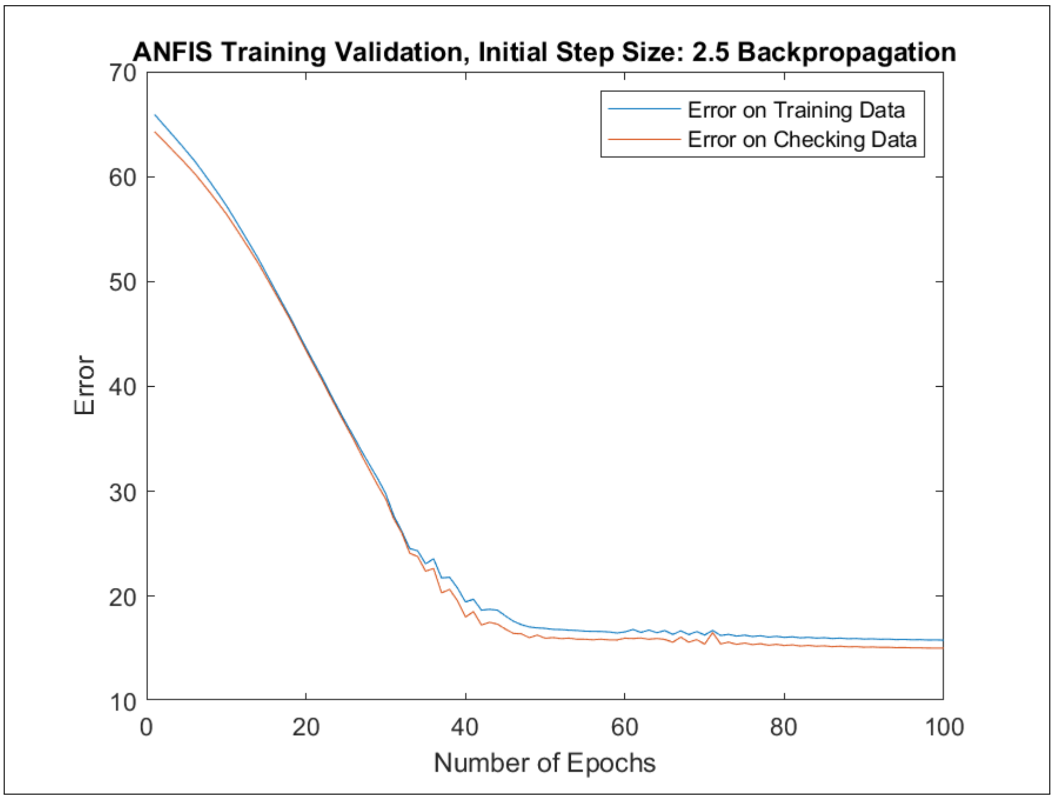

Figure C.76: Best Maximal-Error Model Trained by ANFIS with Back-propagation and Initial Step Size 2.5, Source: [68]

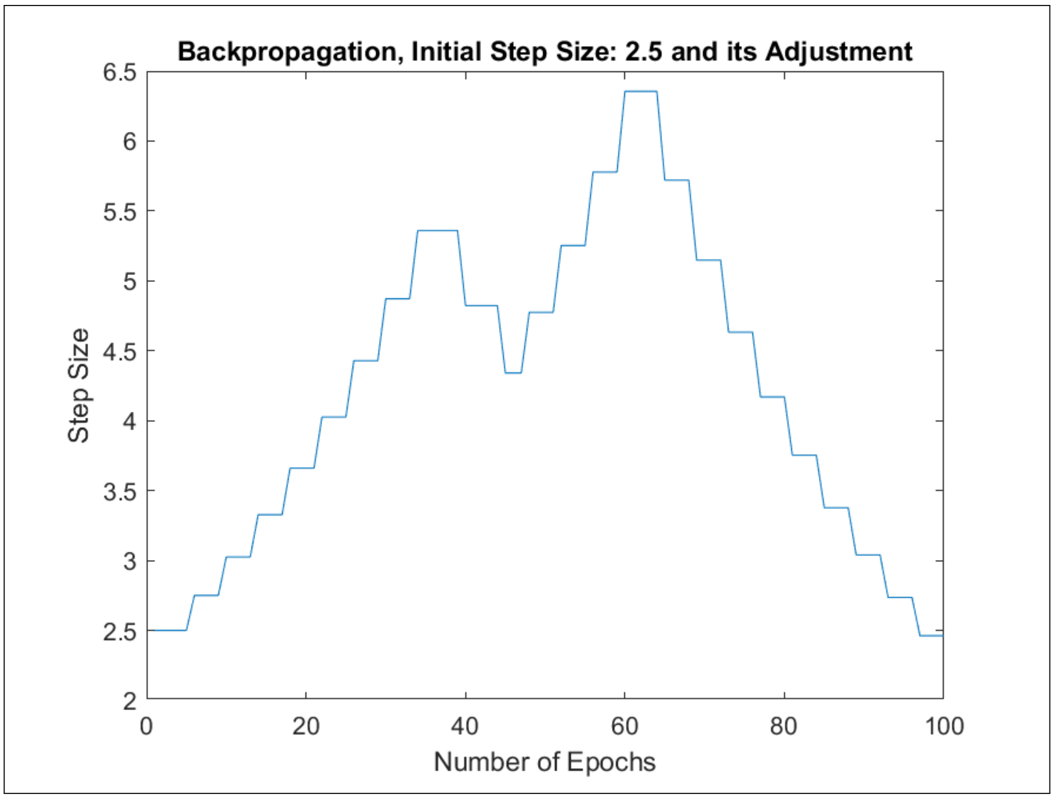

Figure C.77: Best Maximal-Error Model, Adaptation of the Initial Step Size 2.5, Source: [68]

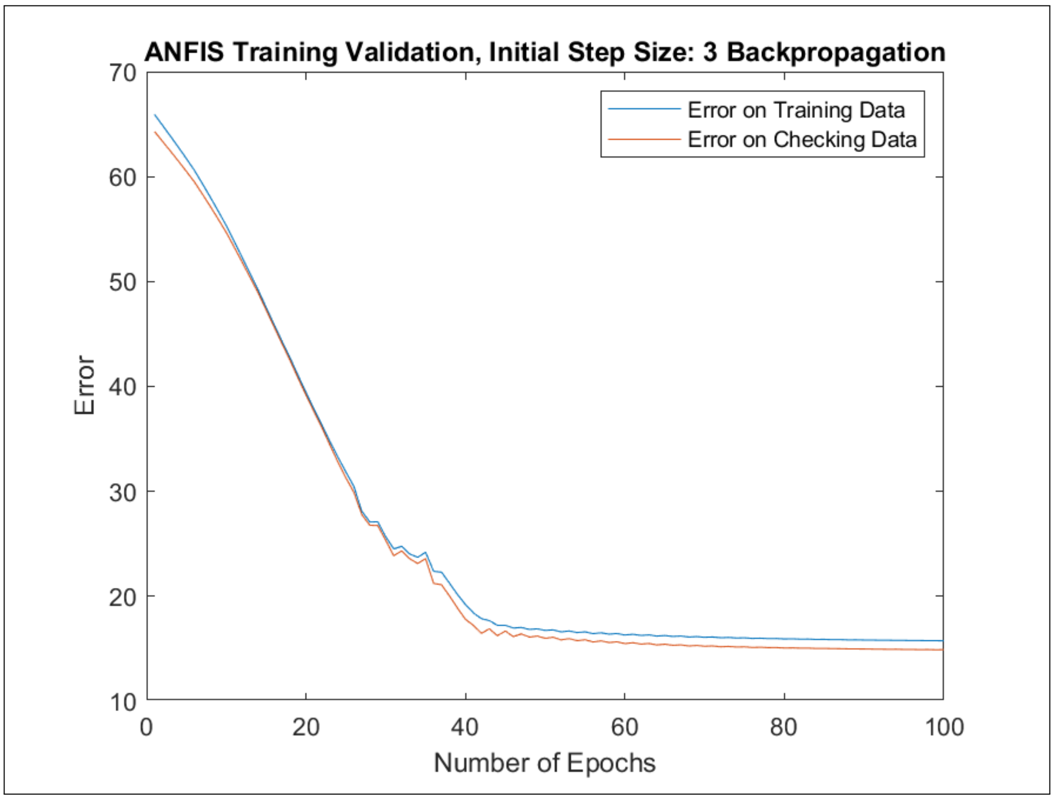

Figure C.78: Best RMSE-Error Model Trained by ANFIS with Back-propagation and Initial Step Size 3, Source: [68]

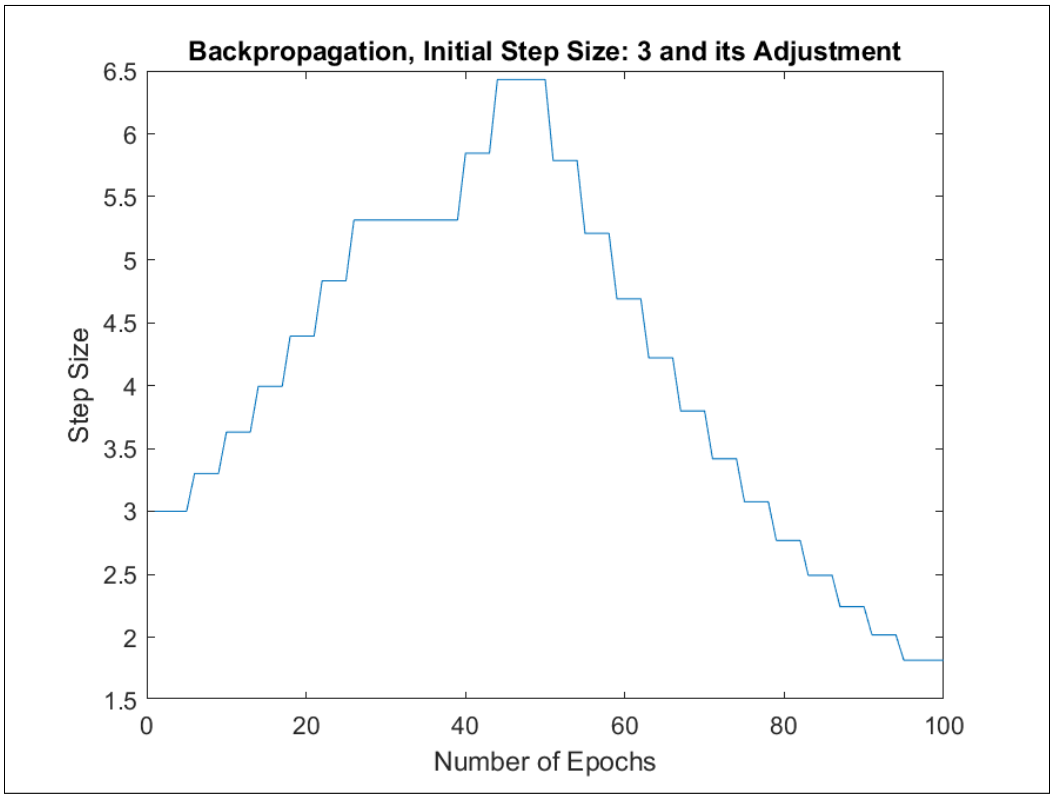

Figure C.79: Best RMSE-Error Model, Adaptation of Initial Step Size 3, Source: [68]

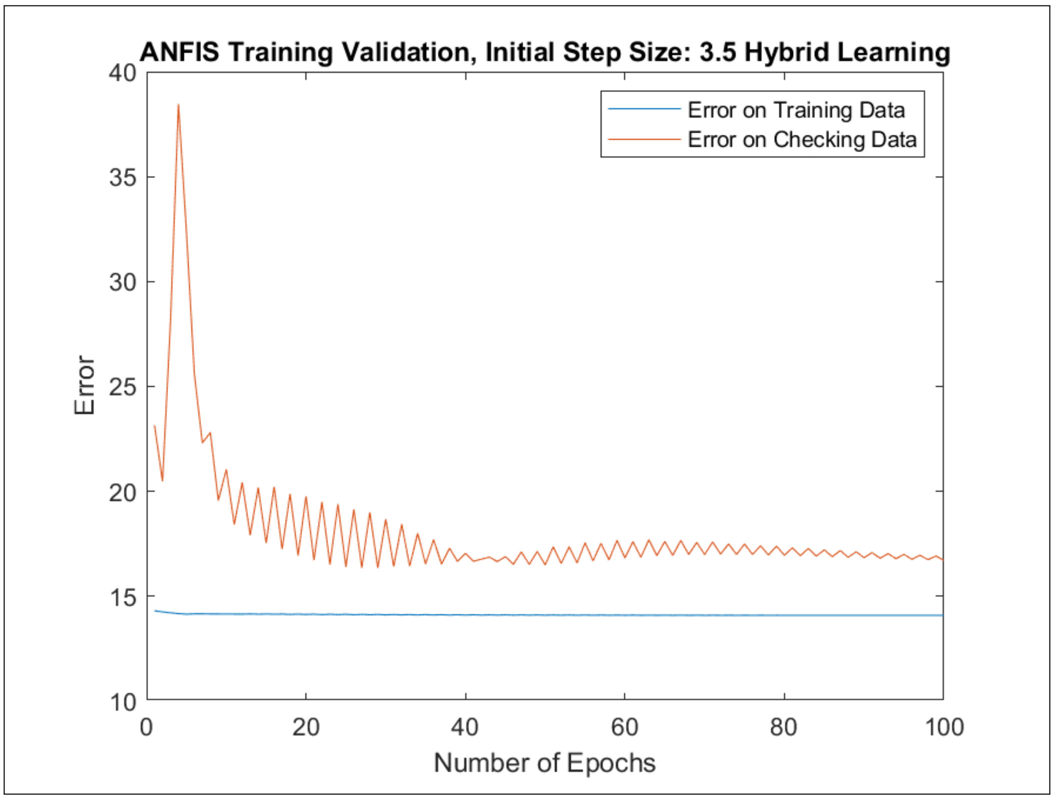

Figure C.80: Best Maximal-Error Model Trained by ANFIS with Hybrid Approach and Initial Step Size 3.5, Source: [68]

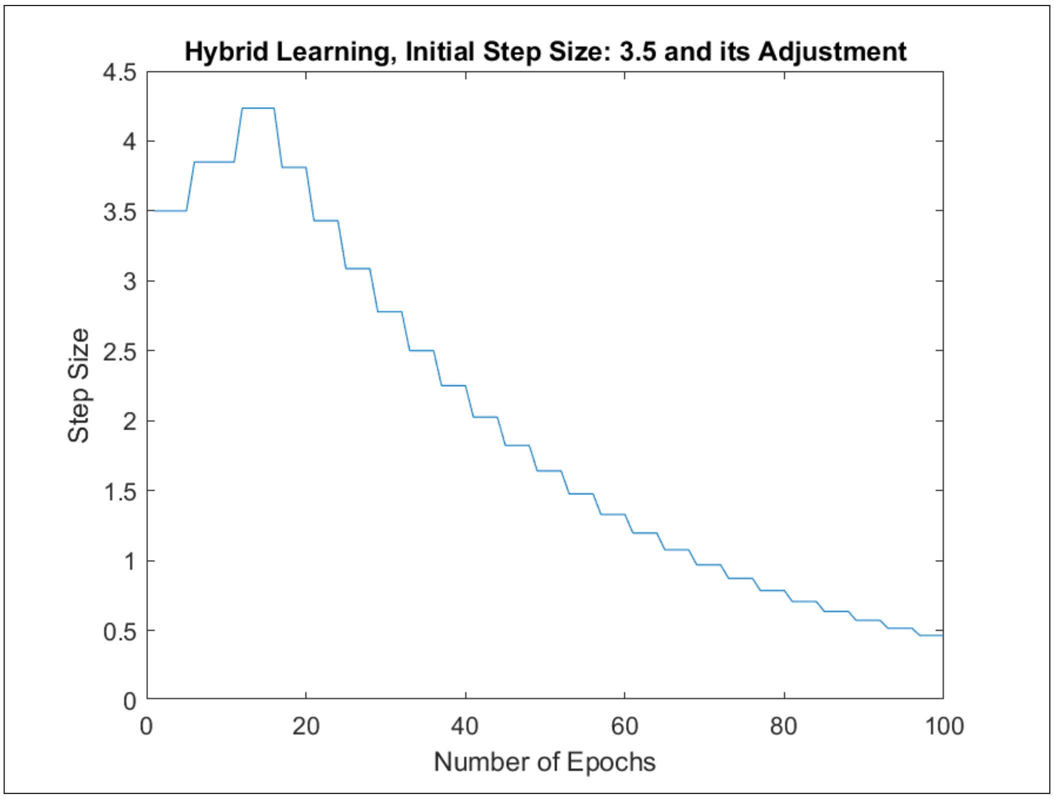

Figure C.81: Best Maximal-Error Model, Adaptation of the Initial Step Size 3.5, Source: [68]

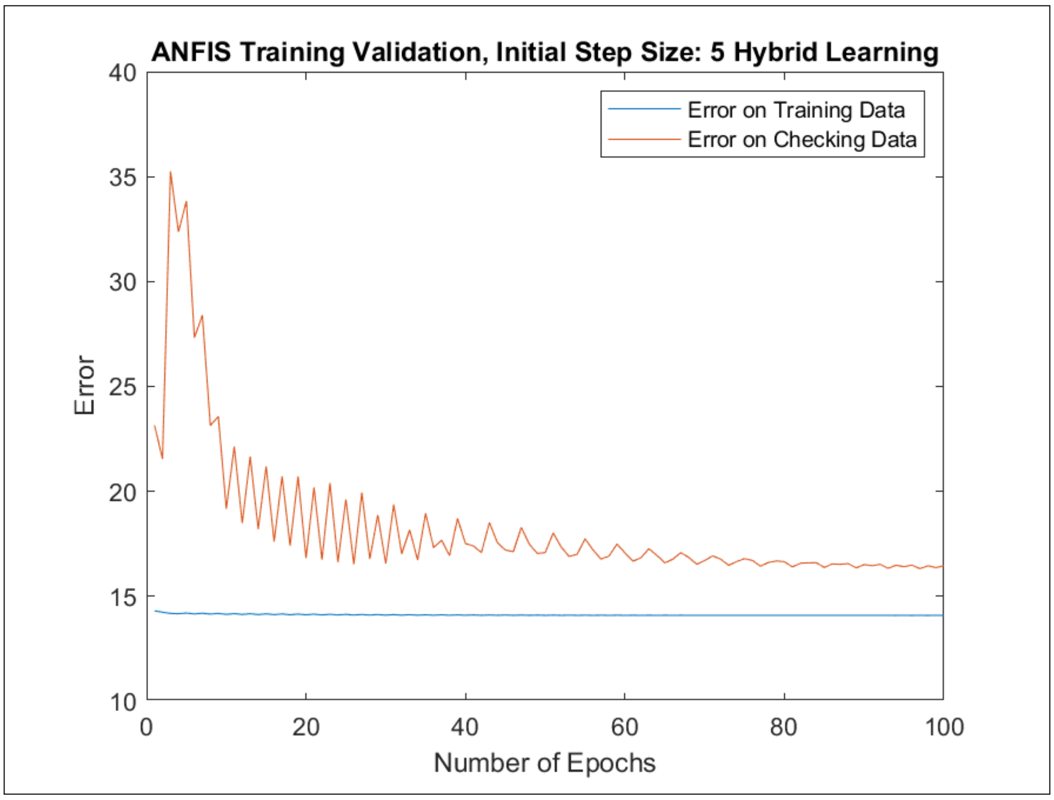

Figure C.82: Best RMSE-Error Model Trained by ANFIS with Hybrid Approach and Initial Step Size 5, Source: [68]

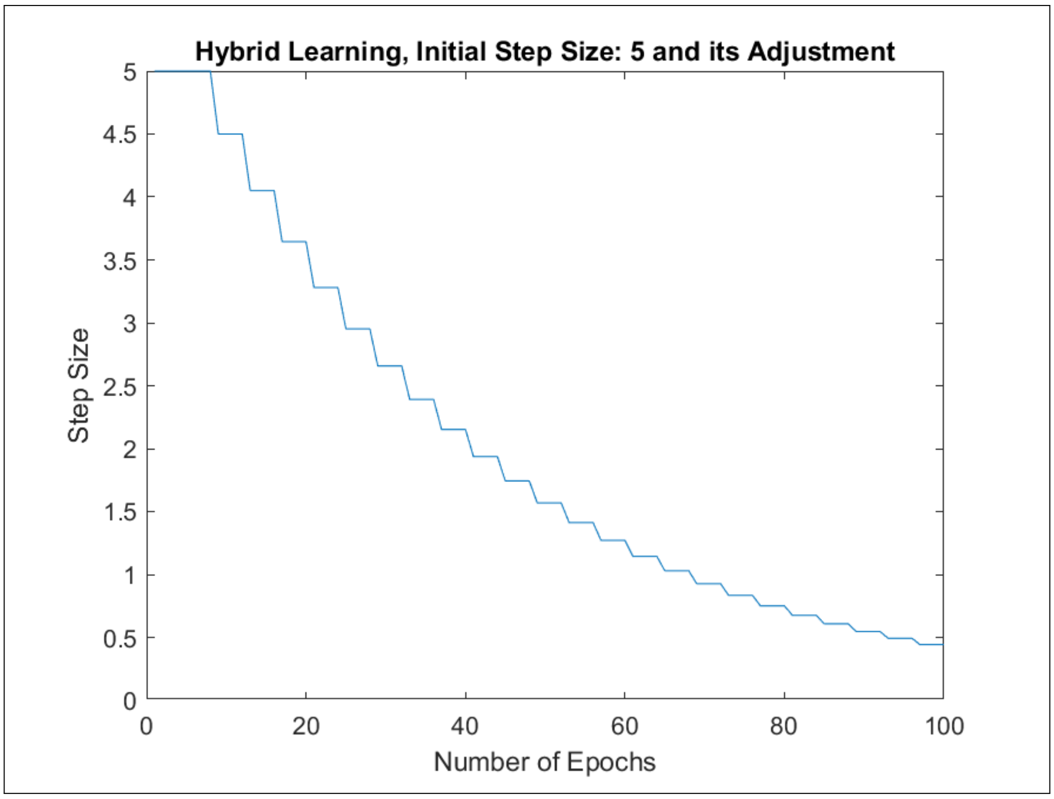

Figure C.83: Best RMSE-Error Model, Adaptation of the Initial Step Size 5, Source: [68]