axioms

Article

Practical Consequences of Quality Views in Assessing Software Quality

Tamas Galli [1,*], Francisco Chiclana [1,2] and Francois Siewe [3]

1 Institute of Artificial Intelligence (IAI), Faculty of Computing, Engineering and Media, De Montfort University, Leicester LE1 9BH, UK; chiclana@dmu.ac.uk
2 Andalusian Research Institute on Data Science and Computational Intelligence (DaSCI), University of Granada, 18071 Granada, Spain
3 Software Technology Research Laboratory (STRL), Faculty of Computing, Engineering and Media, De Montfort University, Leicester LE1 9BH, UK; fsiewe@dmu.ac.uk
* Correspondence: tamas.galli@bcs.org or tamas.galli@my365.dmu.ac.uk

Abstract: The authors' previously published research delved into the theory of software product quality modelling, model views, concepts, and terminologies. They also analysed this specific field from the point of view of uncertainty, and possible descriptions based on fuzzy set theory and fuzzy logic. Laying a theoretical foundation was necessary; however, software professionals need a more tangible and practical approach for their everyday work. Consequently, the authors devote this paper to filling in this gap; it aims to illustrate how to interpret and utilise the previous findings, including the established taxonomy of the software product quality models. The developed fuzzy model's simplification is also presented with a Generalized Additive Model approximation. The paper does not require any formal knowledge of uncertainty computations and reasoning under uncertainty, nor does it need a deep understanding of quality modelling in terms of terminology, concepts, and meta-models, which were necessary to prepare the taxonomy and relevance ranking. The paper investigates how to determine the validity of statements based on a given software product quality model; moreover, it considers how to tailor and adjust quality models to the particular project's needs. The paper also describes how to apply different software product quality models for different quality views to take advantage of the automation potential offered for the measurement and assessment of source code quality. Furthermore, frequent pitfalls are illustrated with their corresponding resolutions, including an unmeasured quality property that is found to be important and needs to be included in the measurement and assessment process.

Keywords: software product quality model; quality assessment; quality view; tailoring quality models; SQALE; ISO25010; Generalized Additive Model; GAM

check for updates

Citation: Galli, T.; Chiclana, F.; Siewe, F. Practical Consequences of Quality Views in Assessing Software Quality. *Axioms* **2023**, *12*, 529. https://doi.org/10.3390/axioms12060529

Academic Editors: Oscar Humberto Montiel Ross and Oscar Castillo

Received: 31 January 2023
Revised: 21 April 2023
Accepted: 25 May 2023
Published: 28 May 2023

1. Introduction

Software quality has gained more influence in recent years, as evidenced by the exponential growth of the identified publications in this field by a mapping study [1]. Software quality assessment encompasses two major approaches: (1) the measurement and assessment of the processes that result in a software product, and (2) the measurement and assessment of a software product directly [2,3]. The former relies on the assumption that controlling the processes of software development will contribute to the final software quality [4]. In our previous article [2], we identified and analysed, from the perspectives academic and industrial relevance, the software product quality models published since 2000. Covering approximately a 20-year period (2000–2019), Table 1 shows the identified models with their relevance scores and corresponding publications ranges. These data can help to determine whether research interest and industrial activity is associated with the listed software product quality models.

Table 1. Ranking of the quality model classes by relevance score, including manual and automatic searches. Source: [2].

Ranking	**Model Class**	**Relevance Score**	**Quality Score Average**	**Publication Range after 2000**	**Google Relative Search Index, Average for 12 Months**
1	ISO25010 [5–13]	130	16.25	[2011; 2018]	30.02
2	ISO9126 [14–22]	120	13.33	[2000; 2017]	53.06
3	SQALE [23–30]	107	13.38	[2009; 2016]	18.33
4	Quamoco [31–34]	90	22.5	[2012; 2015]	0
5	EMISQ [35–37]	38	12.67	[2008; 2011]	0
6	SQUALE [38–41]	36	9	[2012; 2015]	n.a.
7	ADEQUATE [42,43]	18	9	[2005; 2009]	n.a.
8	COQUALMO [44,45]	15	7.5	[2008; 2008]	0.21
=9	FURPS [46–48]	10	3.33	[2005; 2005]	20.56
=9	SQAE and ISO9126 combination [49]	10	10	[2004; 2004]	0
=9	Ulan et al. [13]	10	10	[2018; 2018]	n.a.
10	Kim and Lee [50]	9	9	[2009; 2009]	n.a.
11	GEQUAMO [51]	5	5	[2003; 2003]	0
12	McCall et al. [52,53]	1	0.5	[2002; 2002]	n.a.
=13	2D Model [54]	0	0	n.a.	n.a.
=13	Boehm et al. [55]	0	0	n.a.	n.a.
=13	Dromey [56]	0	0	n.a.	n.a.
=13	GQM [57]	0	0	n.a.	40.73
=13	IEEE Metrics Framework Reaffirmed in 2009 [58]	0	0	n.a.	0
=13	Metrics Framework for Mobile Apps [59]	0	0	n.a.	0
=13	SATC [60]	0	0	n.a.	n.a.
=13	SQAE [61]	0	0	n.a.	n.a.
=13	SQUID [62]	0	0	n.a.	n.a.

The recorded automatic searches, as described in [2], were subsequently repeated for a 2-year period (2020–2022) using the IEEE and ACM databases and the obtained results published in [3]. The identified publications in the additional 2-year period, related to software product quality models [63,64], corroborated the previous trends, and no new software product quality models were identified.

Since the extent of quality that the individual software product quality models are able to address deviates from model to model significantly, further investigation was necessary to help with model selection in practice [65]. To this end, the notion of quality views was used to determine the theoretical maximum extent of quality that each model could capture [65]. Following the ISO/IEC 9126 [19] and ISO/IEC 25010 [10] standards, which defined three distinct quality views (see below definition), the quality models were classified based on their abilities to address and capture software product quality, so that fair software product quality measurement and assessment could be performed, which ultimately could help with model selection in practice.

The definition of the three quality views is as follows [10,19,62,65].

Internal quality view: 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: 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: Set of quality properties that determine how the user perceives software product quality, and how far the goals for which the software is used can be achieved.

Table 2 summarises the abovementioned classification reported in [65]. The abbreviations "I", "E", and "U" stand for internal, external, and quality in use views. The letter "D" is provided for quality models, the views of which depend on tailoring. Moreover, the term “undefined” in Table 2 is assigned to quality modelling approaches that do not define a quality model, do not demonstrate use through case studies, or do not define sample metrics or sample quality properties, and the published guidelines are unable to define a quality model based on the given approach. In addition, the column “Relevance Rank” shows the model’s relevance in the scientific and industrial community as explained in [2]. Table 2 evidences the significant differences in the identified software product quality models. There are many models unable to assess the external quality view and the quality in use view, which means that they cannot describe how the software behaves while it is used, nor how the end user perceives its quality.

Table 2. Taxonomy: software product quality model families and their quality views. Source: [65].

ID	Relevance Rank	Name	Quality Views Considered	Predefined Quality Properties or Metrics Available	Also Process-Related Properties
1	1	ISO25010 [5–12]	I, E, U	Yes	No
2	2	ISO9126 [14–22]	I, E, U	Yes	No
3	3	SQALE [23–30]	I	Yes	No
4	4	Quamoco [31–34]	I, E, U	Yes	No
5	5	EMISQ [35–37]	I	Yes	No
6	6	SQUALE [38–41]	I, E	Yes	Yes
7	7	ADEQUATE [42,43]	I, E, U	Yes	Yes
8	8	COQUALMO [44,45]	I, E	Yes	Yes
9	9	FURPS [46–48]	I, E, (U)	Yes	Yes
10	9	SQAE and ISO9126 combination [49]	I, E	Yes	No
11	9	Ulan et al. [13]	I	Yes	No
12	10	Kim and Lee [50]	I	Yes	No
13	11	GEQUAMO [51]	I, E, U	Yes	Yes
14	12	McCall et al. [52,53]	I, E, (U)	Yes	Yes
15	13	2D Model [54]	Undefined	No	Undefined
16	13	Boehm et al. [55]	I, E, (U)	Yes	No
17	13	Dromey [56]	I	Yes	No
18	13	GQM [57]	D	No	D
19	13	IEEE Metrics Framework Reaffirmed in 2009 [58]	Undefined	No	Undefined
20	13	Metrics Framework for Mobile Apps [59]	Undefined	Yes	Undefined
21	13	SATC [60]	I, E	Yes	Yes
22	13	SQAE [61]	I	Yes	No
23	13	SQUID [62]	D	D	D

In the software product quality modelling domain, it is of extreme importance to understand that Table 2 presents the theoretical maximum quality in terms of the views that a given model is able to address. Thus, model implementations, such as SonarQube [66], that rely on the underlying SQALE model [30] are able to measure and assess the internal quality view only. To ensure a fair demonstration of the quality measurement and assessment results, the quality models and the quality views applied in the course of the measurement and assessment always need to be presented [65].

Both Tables 1 and 2 assist software professionals to make optimal decisions while selecting and tailoring software product quality models. This process is highlighted in Section 3.2 in detail with case studies. Section 2 briefly summarises the research methods used in the herein reported study, Section 3 shows the obtained results and Section 4 revisits some important points raised in the case studies, while Section 5 presents our concluding remarks. Appendix A includes the annotated R code applied to prepare the Generalized Additive Model (GAM) model in Section 3.1.

2. Materials and Methods

The abstract notion of software product quality and the complex mathematical computations applied in [67] for reasoning under uncertainty might obscure the practical use of the findings documented in [2,65,67] and discourage potential practitioners. Thus, from the point of view of research methods, the current study contributes to the body of research with (1) a model simplification by approximating the quality model published in [67] with a Generalized Additive Model (GAM) [68], and (2) a practical demonstration of the findings of the authors' previous publications [2,65,67].

Two fictional, but practical and realistic, case studies are detailed to show how to interpret and utilise the taxonomy of Table 2. The case studies aim to lead the reader through complex situations full of pitfalls, which might not necessarily appear in each setting. The presentation does not require any familiarity with the uncertainty computations applied in [67], nor a deep understanding of quality modelling in terms of terminology, concepts, and meta-models, which were necessary to prepare the taxonomy and relevance ranking as reported in [2,65].

3. Results

This section presents (1) an investigation of how execution tracing quality can be described by a GAM model [68], which will be referred to as model simplification; and (2) fictional but realistic case studies that show how software product quality can be modelled, measured, and assessed, and how quality models can be tailored to specific needs. The below model simplification is only useful to offer more insight into the extent to which a quality property impacts the execution tracing quality. This information is placed in context, as reported in Section 3.2, while tailoring the quality models for the considered case studies. Nevertheless, the computations in Section 3.1 are not strictly necessary to apply quality modelling.

3.1. Model Simplification

The GAM [68] modelling approach is used to approximate the fuzzy model describing the quality of execution tracing developed in [67]. Beyond constructing a simplified model, a further goal of the modelling approach is to understand the roles of the input variables from a different perspective.

The reason for choosing GAM lies in its capacity to capture non-linear relationships without the need to specify the assumed degrees of polynomials. Indeed, GAM assumes the below additive relationship between the functions of the input variables, which explicitly shows the impact of each variable on the output:

$$y = \beta_0 + s_1(x_1) + s_2(x_2) + \cdots + s_n(x_n) + \epsilon \tag{1}$$

where y is the output variable; $x_1, \ldots, x_n$ are the input variables; β_0 is the intersect value; $s_1, \ldots, s_n$ are smooth functions; and ϵ is the error term. Within the scope of the learning algorithm, ϵ is minimised while $\beta_0, s_1, \ldots, s_n$ are identified, and the smooth functions are constructed from basis functions [68,69].

3.1.1. Modelling Steps

The modelling steps are as follows: (1) generating data from the fuzzy model, which was reported in [67]; (2) building a simplified model with GAM; and (3) checking the performance of the constructed new model.

The fuzzy model has Accuracy, Legibility, Implementation and Security as input variables, and Execution Tracing Quality as its output variable, which also constitute the variables of the simplified model. All variables are assumed to take values in the $[0, 100]$ domain range. While generating the data, the values of the input variables were increased by 5, starting from a value of 0 until the whole domain was covered, while the output variable was recorded for each combination of values of the input variables. Thus, each input variable took the 21 selected values $\{0, 5, 10, \ldots, 100\}$, which means that each possible combination of the 4 input variable values with its corresponding output variable value generated 21^4 = 194,481 points in a 5-dimensional space. The generated data were published in [70] and the GAM model was trained using this data set. The annotated R code to construct the GAM model is given in Appendix A. Figure 1 shows the relationship of each input variable and the output variable assuming that the other input variables are zero. The curves are shifted with the intercept ($\beta_0 = 37.9$), so that the y value can depict the real output value (Execution Tracing Quality). The effective degree of freedom for each input variable is printed on the y axis, and it highlights a highly non-linear relationship in each case.

Figure 1 illustrates an increasing relationship between the output variable and each input variable individually, and the rate of increase in the output variable is higher for the input variables Accuracy and Implementation; the rate of increase in the output variable with respect to the input variable Legibility is positive but approaches zero in the subrange [40; 60]; meanwhile, the rate of increase in the output variable is lowest for the input variable Security. These results corroborate the findings of the fuzzy model obtained in [67].

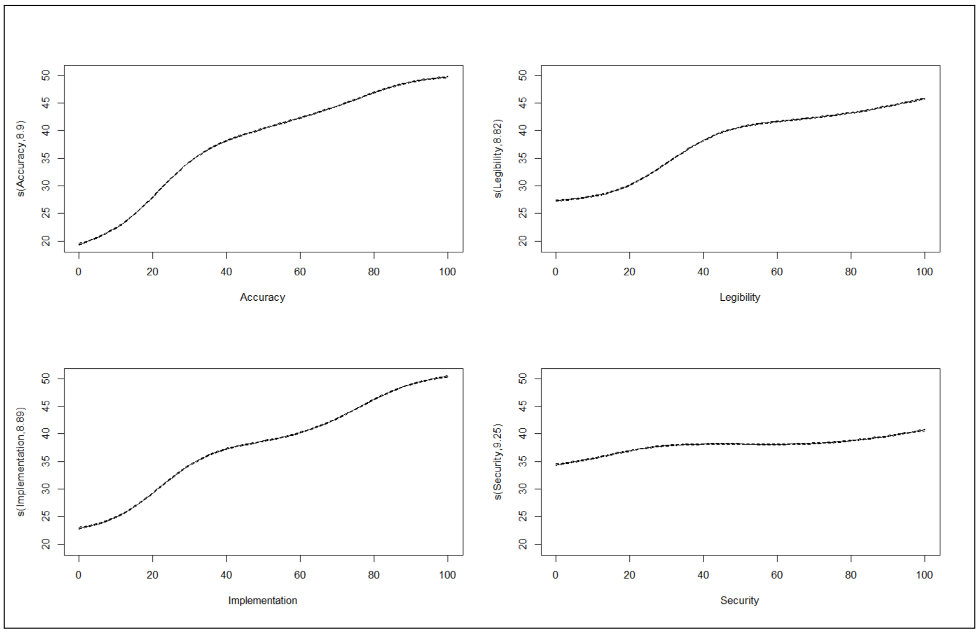

Figure 1. Impact of the inputs.

3.1.2. Checking the Constructed Model

This sections aims to answer questions such as the following: To what extent can the newly created GAM model capture the variance of the data set produced by the fuzzy model? Are the contributions of the smooth functions to the output variable significant? Is the number of basis functions, which make up the smooth functions, large enough to describe the patterns in the data set? Do the input variables have parallel curves that result in a poorly fitting and unstable model?

- Since the R^2 value of the simplified model was 0.775, it can be stated that 77.5% of the variability of the output variable of the fuzzy model was explained by the variability of the output variable of the GAM model, which can be interpreted as an acceptable fit for the simplified model. Table 3 presents the smooth terms with their effective degrees of freedom and the performed F-test statistics with the corresponding p-values. All smooth terms are statistically significant. Thus, the identified smooth functions contribute significantly to the output variable.
- Considering the research settings of [67] and the data generation from the fuzzy model described above, it is unlikely that the inputs have parallel curves, although this was examined.

Linear correlation: Table 4 provides the Pearson correlation between each pair of variables, and it is obvious that there is a lack of linear correlation between the pairs of input variables. The strongest linear correlations between the input variables and the output variable are for the variables Accuracy and Implementation, while the linear correlation of the input variable Security with the output variable is nearly negligible, which conforms with Figure 1.

Concurvity: GAM concurvity measures the extent up to which a "smooth term in a model can be approximated by one or more of the other smooth terms in the model" (concurvity measures: https://stat.ethz.ch/R-manual/R-devel/library/mgcv/html/concurvity.html, accessed on 20 April 2023). GAM concurvity is measured in the range $[0, 1]$, with no problem identified with a 0 value, while total concurvity is achieved when value 1 is obtained. Table 5 shows the concurvity values. Considering the worst case, all values lie in the vicinity of zero, indicating that the input variables do not have a non-linear relationship.

- The smooth functions are constructed from basis functions to model the patterns found in the data set. If the number of basis functions is too low, then some patterns are missed in the data. The p-values in Table 6 show no significance. (The mgcv R library offers a function to perform this computation, as demonstrated in Appendix A. This check resulted in increasing the number of basis functions for the input variable Security to avoid a significant p-value.) This means that the number of basis functions is appropriate to cover the existing patterns in the data set.

Table 3. Smooth terms and their significance.

Smooth Term	Effective Degree of Freedom	*F*-Statistic	*p*-Value
s(Accuracy)	8.898	33,515.5	$<2 \times 10^{-16}$
s(Legibility)	8.815	14,164.6	$<2 \times 10^{-16}$
s(Implementation)	8.893	26,134.6	$<2 \times 10^{-16}$
s(Security)	9.247	685.9	$<2 \times 10^{-16}$

Table 4. Pearson correlation matrix.

	Accuracy	Legibility	Implementation	Security	Execution Tracing Quality
Accuracy	1	1.402361×10^{-5}	1.402361×10^{-5}	1.402361×10^{-5}	0.57322295
Legibility	1.402361×10^{-5}	1	1.402361×10^{-5}	1.402361×10^{-5}	0.36988615
Implementation	1.402361×10^{-5}	1.402361×10^{-5}	1	1.402361×10^{-5}	0.51562623
Security	1.402361×10^{-5}	1.402361×10^{-5}	1.402361×10^{-5}	1	0.08651949
Execution Tracing Quality	0.5732230	0.3698862	0.5156262	0.08651949	1

Table 5. Concurvity metrics.

Case	s(Accuracy)	s(Legibility)	s(Implementation)	s(Security)
worst	2.557907×10^{-8}	2.557907×10^{-8}	2.557907×10^{-8}	2.746695×10^{-8}
observed	5.336778×10^{-9}	4.135488×10^{-9}	4.515394×10^{-9}	7.145365×10^{-9}
estimate	3.516405×10^{-9}	3.516405×10^{-9}	3.516405×10^{-9}	4.230810×10^{-9}

Table 6. Number of basis functions.

Smooth Function	Effective Degree of Freedom	*p*-Value
s(Accuracy)	8.90	0.68
s(Legibility)	8.82	0.82
s(Implementation)	8.89	0.30
s(Security)	9.25	0.60

3.1.3. Conclusions of Model Simplification

The model simplification confirmed that input variables Accuracy and Implementation have the strongest effect on the output variable Execution Tracing Quality. The input variable Security only has a marginal effect on the output variable Execution Tracing Quality, while the input variable Legibility in the range [0; 40] also has a strong impact on the output variable Execution Tracing Quality. These insights are important if we place execution tracing quality into context, and the case studies in Section 3.2 illustrate the application with regard to the overall software product quality.

3.2. Case Studies

The following two projects are used as case studies for demonstration purposes: (1) the first project, which is already running, applies quality measurements and the stakeholders would like to know the validity of the statements based on these measurements; (2) the second project, which has recently been launched, requires a software product quality model to be selected to satisfy the assessment of non-functional requirements and the wishes of the stakeholders. In addition, the second project is divided into two parts: (2a) the first part leads the reader through the quality model selection and tailoring process via the application of two different software product quality models, one for the internal quality view to take advantage of automation, and the other for the external and quality in use views where automation is not possible at present; (2b) the second part demonstrates how to consider new findings and their implementation as new requirements in the quality measurement and assessment process. The considered case studies form part of the corresponding author's PhD thesis [3].

While the different quality properties are measured and the quality metrics are computed, it is important to consider the measurement scales and data aggregation. If aggregation is needed, then masking, threshold, and compensation effects have to be considered [23]. For different metric scales, normalisation needs to be performed [71]. In the considered case studies, quality metrics are all measured in the range [0; 1] and, therefore, scale normalisation is not necessary. The quality metrics are assigned to quality sub-characteristics, which are assigned to quality characteristics constituting a hierarchy, although the quality sub-characteristics are not aggregated to one individual metric, which helps to avoid the issues described in [23].

3.2.1. Case Study 1: Validity of Statements Based on a Given Software Product Quality Model

Background:	Based on the SQALE model, SonarQube [66] is a widespread tool for quality measurement and assessment, with easy-to-use integration into the project life-cycle [2,3,30]. Technical debt is one major element of the SQALE definition [30], which refers to any non-conformance of coding standards described in the form of a rule set. Each identified non-conformance is also associated with a remediation cost in time units to improve the given defect. The identified defects can also be associated with non-remediation costs, which describe the impact of the defect on the system as a whole. However, the estimation of non-remediation costs is vague and error-prone; therefore, the identified defects are usually classified based on criticality from blocker to information on an ordinal scale. Company A decides to introduce measures to apply SonarQube in each software development project and the company defines its own rule sets to identify technical debts using the tool. A project team at company A has already integrated SonarQube [66] into their project but they would like to know to what extent the tool is able to measure software quality as a whole and to what extent the team can rely on the measurement results. This is also an extremely important question when transferring projects for maintenance to external service providers, where software quality can have a direct impact on the financial negotiations.
Step 1:	Participants in the development project check the documentation of SonarQube and they find that SQALE is the underlying software product quality model.

Step 2: After successfully identifying the software product quality model, the project team looks up SQALE in Table 2 and they find that SQALE is able to measure and assess only one of the three different manifestations of software product quality, which is a fraction of the whole.

Result: The project team draws the conclusion that SonarQube can measure and assess how the source code is written; moreover, the measurements are performed in a consistent and precise manner. However, SonarQube cannot measure and assess (i) the architecture, (ii) how the software behaves while it runs, and (iii) whether the end user is satisfied with the product. Consequently, SonarQube's results might be valid for source code quality, depending on the rule set applied during the quality measurement, but these results cannot be considered valid for software product quality in general. Measuring and assessing software product quality as a whole requires the measurement and assessment of all three quality views: internal, external, and quality in use. Thus, the SonarQube results need to be supplemented with the measurement and assessment of the operational behaviour of the software and with the measurement and assessment of the end user feedback to be able to judge software quality as a whole.

3.2.2. Case Study 2a: Selecting a Quality Model for Measurement and Assessment, and Tailoring It to Specific Project Needs

Background: This case study shows how a software product quality model can be selected and the quality requirements refined with regard to a certain project. Eeles's checklist [48] can be of assistance to collect non-functional and architectural requirements. The stakeholders formulate the following quality guidelines: (1) source code quality must be measured and assessed, (2) the quality of the operational software must be measured and assessed, and (3) the perception of the end user regarding the software must also be measured, assessed, and reported after each defined milestone in the project's life-cycle.

Step 1: Considering the stakeholders' guidelines, members of the project team recognise that the selected software product quality model needs the capability to measure all three distinct quality views: internal, external, and quality in use. Thus, as per Table 2, the following models are eligible: ISO/IEC 25010 [10], ISO/IEC 9126 [19], Quamoco [31–34], ADEQUATE [42,43], GEQUAMO [51], GQM [57], SQUID [62].

Step 2: The project team needs to select a software product quality model for measurement and assessment. The GQM [57] and SQUID [62] approaches would require the creation of a quality model individually, which the project team would avoid. The project team wishes to use a widespread quality model with constant research interest and industrial use cases, which narrows down the set of eligible quality models to ISO/IEC 25010 [10] and ISO/IEC 9126 [19]. From these two models, the project team selects ISO/IEC 25010 [10] because it supersedes the predecessor standard ISO/IEC 9126 [19].

Step 3: After familiarising themselves with the extensive definition of the ISO/IEC 25010 [10] standard, the project team considers how to measure and assess software product quality for each distinct quality view. This provides the project team with an important insight: they need to set up static code analysers individually and interpret their results for the selected high-level quality properties that they use from the standard. Kim and Lee published a similar study in [50]. However, the effort required for the whole setup motivates the team to look for alternatives. They come to the conclusion that they will use a different quality model for the measurement and assessment of the internal quality view, which offers automation potential and requires less effort while integrating into the development pipeline. Consulting Table 2 and the study in [2], they decide to use the SQALE model [30], which offers easy integration into the software life-cycle through tool implementation including SonarQube [66]; moreover, they make the decision to accept the default rule set for their target programming language.

Step 4: The project team outlines the measurement and assessment for the external and quality in use views, based on the ISO/IEC 25010 standard [10], which makes it possible to investigate the operational software and the feedback of the end user. As the standard encourages tailoring to specific project needs, they select the most important high-level quality properties to measure and assess, in agreement with the stakeholders. The selection can be performed in many different ways but the simplest one seems to be constant sum scaling [72], where each stakeholder receives a constant score value to be distributed among the most important high-level quality properties in his/her opinion. Where the score values accumulate, those quality properties represent the most valuable assets for the stakeholders and they need to be considered while measuring and assessing quality. After the discussion and selection process, the following high-level quality properties are chosen for the specific project: (1) for the external quality view, (a) performance efficiency and (b) reliability; (2) for quality in use, (a) satisfaction.

Step 5: The project team defines new quality metrics to measure the selected high-level quality properties and has them approved by the stakeholders. Each metric has a measurement method and a continuous scale in the range [0; 1], where 1 stands for the best possible value.

Step 6: The project team defines the acceptable metric ranges for each metric in Table 7. After approval by the stakeholder, the following acceptable ranges are defined: (1) response time metric over 0.55, (2) crash metric over 0.5, (3) manual intervention metric over 0.5, and (4) usefulness goal metric over 0.7. Defining a fine-granular ordinal scale for the assessment is also possible, with many ranges in the acceptable and non-acceptable domain and with a transient one in between.

Step 7: Quality measurement: The project team measures the following values: (a) the external interface mean response time is 20 msec, while its maximally allowed response time amounts to 30 msec; (b) 2 crashes in the timeframe of the system test, (c) 1 manual intervention during the system test so that the software's operational state can be maintained, and (d) three end users rate the software's usefulness from the point of view of achieving their work goals with the scores 6, 9, and 7. These quality metric elements result in the following quality metrics: (1) response time measure—0.6, (2) crash measure—0.33, (3) manual intervention measure—0.5, and (4) usefulness goal measure—0.73.

Step 8: Quality assessment: Not all of the quality metrics achieve the defined quality targets determined in Step 6, which means that the software cannot be released without improvements.

Result: The project team successfully defines and carries out the first quality measurement affecting all three quality views. Based on the defined acceptable ranges, they make a decision regarding whether the software can be released or not. If they define transient ranges between the acceptable and non-acceptable metrics' ranges, and if the measured values fall into those ranges, then the software release is possible after the stakeholders' approval.

3.2.3. Case Study 2b: Adjusting the Selected Quality Model to Include a Further Quality Property

Background: After deploying the software, the development team and the stakeholders regularly evaluate the quality measurements and trends. Moreover, they also consider the feedback from the end users, and they analyse the error reports.

While examining the error reports and their resolution times, they find that identifying the causes of errors requires far more time than expected. Thus, the software's maintainability property—more specifically, analysability—requires attention.

Step 1: The development team as responsible for the software's maintenance, in agreement with the stakeholders, includes the maintainability quality property with its analysability sub-property into the quality measurement and assessment process. They identify different publications [3,67,73] describing the Execution Tracing Quality, which can exactly be linked to the analysability quality property. Thus, the development team adds Execution Tracing Quality to the metrics that they measure for the external quality view. Furthermore, they make the decision to simplify the computation of this metric and they use weighted averaging to aggregate the quality metric elements instead of fuzzy logic or GAM. They determine the following weights (Section 3.1 justifies the weights in general but the weights may differ for each project and they can be adjusted to the specific context of use): (1) ET-1 (Accuracy)—0.3; (2) ET-2 (Legibility)—0.3; (3) ET-3 (Design and Implementation of the Trace Mechanism)—0.3; (4) ET-4 (Security)—0.1. Thus, the development team, in agreement with the stakeholders, measures and assesses maintainability as well. Table 8 introduces all measured characteristics, sub-characteristics, and quality metrics with the corresponding quality metric elements.

Step 2: The project team prepares guidelines on how to measure the quality metric elements for Execution Tracing Quality; moreover, they determine the acceptable and non-acceptable ranges for the metric in agreement with the stakeholders. They conclude that the Execution Tracing Quality metric is acceptable above 0.7 and non-acceptable below 0.5. In addition, they agree that they will consider only this quality metric for maintainability at present.

Step 3: *Quality measurement:* The development team determines the quality metric elements based on their defined guidelines for Execution Tracing Quality: Accuracy—4, Legibility—5, Design and Implementation—7, and Security—7. This results in a value of 0.55, computed with the formula defined in Table 8. All other metrics in Table 8 are computed as illustrated above.

Step 4: *Quality assessment:* The project team examines whether all quality metrics fall into the defined acceptable ranges. If there is only one metric that falls into the non-acceptable range, then the software must not be released. If transient ranges are also defined, i.e., one or more ranges exist between the acceptable and non-acceptable ranges, and a metric falls into that range, then the software might be released if the stakeholders approve it. Execution Tracing Quality does not fall into the acceptable range, so the defined quality targets are not met, but it does not fall into the non-acceptable range either. Consequently, the project team contacts the stakeholders to make a decision regarding whether the software can be released without improvement. Even if the software can be released, the quality of execution tracing needs to be improved before the next release.

Result: The project team finds that the tailored software product quality model in Section 3.2.2 does not cover a quality characteristic that is important for the given context of use. Consequently, they add the identified quality characteristic to the model; moreover, they define a quality metric and couple it to the newly added quality characteristic. This tailored software product quality model is used for quality measurement and assessment in the project's further life-cycle. If future evaluations uncover any missing but important quality properties, then they can be incorporated into the quality model and into the quality measurement and assessment process in a similar manner.

Table 7. Case study 2a: tailoring the ISO/IEC 25010 quality model to the needs of the specific project. Source: [3].

View [1]	Quality Characteristic	Purpose	Quality Sub-Characteristic	Quality Metric	Measurement Method	Computation Formula
E	Performance efficiency	It expresses how well the software performs	Time behaviour	Response time metric	Determine the most important business operations in the system or at external interfaces and measure the response time for them. Compute the metric for each business operation determined as defined by the formula.	$\frac{1}{1+\frac{time_{response}}{time_{maxallowed}}}$
E	Reliability	It expresses how reliable the software is	Availability	Crash metric [2]	Count the number of crashes and the number of "freezes" (when the software is available but it does not respond) in a given timeframe, and then compute the metric as defined by the formula.	$\frac{1}{1+count_{crash}+count_{freeze}}$
			Fault tolerance	Manual intervention metric	Count the number of manual interventions that are necessary to maintain the operational state of the software in a given timeframe, and then compute the metric as defined by the formula.	$\frac{1}{1+count_{manualintervention}}$
U	Satisfaction	It expresses how satisfied the end user is when he/she uses the software	Usefulness	Usefulness goal metric	The end users assess how easily they can achieve their goals with the use of the software. The user assessment results in a score in the range $[0,1]$. The higher the value is, the higher the user's satisfaction is. Compute the metric as defined by the formula.	$\frac{\sum_{n=1}^{count_{users}} assessment_n}{10*count_{users}}$

[1] I: internal quality view, E: external quality view, U: quality in use view. [2] If the software starts up quickly and automatically, then the up-time ratio does not appropriately mirror the availability.

Table 8. Case study 2b: tailoring the ISO/IEC 25010 quality model to the needs of the specific project, including maintainability. Source: [3].

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

[1] I: internal quality view, E: external quality view, U: quality in use view. [2] If the software starts up quickly and automatically, then the up-time ratio does not appropriately mirror the availability.

4. Discussion

The above two fictional, but practical and realistic, case studies led the reader through the quality measurement and assessment process, while identifying potential pitfalls and illustrating how to handle them.

Case study 1 showed how to identify the quality views, i.e., the theoretical maximum amount of quality on which a software tool for analysis relies. The SQALE model and its implementations, including SonarQube [66], are unable to handle the external and quality in use views, which means that, based on SQALE, valid statements cannot be made regarding the software's direct runtime behaviour and the quality as perceived by the end user. 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 not valid. 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 software product quality as a whole. For the above reasons, the statement "non-remediation costs estimated by SQALE can be claimed", which is based on SQALE indices, made in [27], may not be considered valid in a general sense but solely for the internal quality view.

Case study 2a and case study 2b explain how to (i) select a software product quality model for a given project; (ii) tailor the selected model to the project's needs; and (iii) combine two different quality models to take advantage of automation. In addition, both case studies show the desired close cooperation between the development team and the stakeholders. Furthermore, case study 2b shows how an emerging issue in connection with the tailored software product quality model is found, and how it is resolved in the long term; thus, it illustrates the adaptive nature of the quality measurement and assessment process. If the software maintenance is outsourced to personnel who differ from the original development team, then they also need to be involved in discussions to identify the quality characteristics to measure and to construct the quality metrics, especially in the above scenario, where software maintainability needs improvement.

5. Conclusions

This paper demonstrates the practical importance of the notion of quality views and the taxonomy established in [65]. Applying the taxonomy [65] and selecting quality models for projects does not require in-depth knowledge of software product quality modelling, meta-models, and reasoning under uncertainty [67]. The paper thoroughly explains how to select a software product quality model, how to tailor it to specific project needs, and how to adjust it if it is challenged; thus, the adaptive nature of software product quality measurement and assessment is exemplified.

Furthermore, it is shown how to validate a statement based on a software product quality model and how to decide whether the statement holds in general for software product quality or only for one of its specific views. It is of extreme importance that no valid statements are made on software product quality as a whole based on quality models that are able to address only the internal view of quality, i.e., the source code and documentation quality. Only such models allow for full automation at present, and human intervention is necessary to measure and assess the external and quality in use views.

In conclusion, software product quality measurements and assessments must always be presented with the applied quality models and their quality views to be able to interpret the results in a fair manner.

Author Contributions: Conceptualization, T.G.; methodology, T.G.; validation, F.C. and F.S.; formal analysis, T.G.; investigation, T.G.; resources, T.G.; data curation, T.G.; writing—original draft preparation, T.G.; writing—review and editing, F.C. and F.S.; visualization, T.G.; supervision, F.C. and F.S.; project administration, T.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data used for model simplification were made available at Zenodo, see reference [70] for further information.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Annotated R Code for Model Construction

```
library(readxl)
library(mgcv)

# data file is published: Galli, Tamas; Chiclana, Francisco;  Siewe, Francois. (2023). Data Set
# Generated by the Fuzzy Model Constructed to  Describe Execution Tracing Quality [Data set].
# Zenodo. https://doi.org/10.5281/zenodo.7841041
INPUT_FILE5 <- "./input/derived_data_from_gafis_model_stepsize5.csv"

# the output of the fuzzy model was collected while the inputs were increased by a step
# size 5 in the range [0; 100], i.e., 21 steps from 0 to 100
# we have 4 inputs, which result in 194481 rows (21^4 = 194481 rows)
df_input5 <- read.csv(INPUT_FILE5, sep = ",")
colnames(df_input5) <- c("Accuracy", "Legibility", "Implementation", "Security", "Quality")

# checking the data structure and variable types for the data frame
str(df_input5)

# fitting GAM, number of basis functions for the predictor Security was increased to k=15
# so that we could get appropriate effective degree of freedom (EDF) with gam.check()
model_gam5 <- gam(Quality ~ s(Accuracy) + s(Legibility) + s(Implementation) + s(Security, k=15),
data=df_input5, method="REML")

# plotting GAM
# shifting the standard errors and the curve with the intercept
# to get directly interpretable results with regard to the outcome
plot(model_gam5, all.terms = TRUE, pages = 1, se=TRUE,
     seWithMean = TRUE, shift = coef(model_gam5)[1])

# printing model summary, including R-squared
summary(model_gam5)

# looking at model parameters
coef(model_gam5)

# GAM check, whether we have enough number of basis functions for each predictor
gam.check(model_gam5)

# checking whether we have concurvity among the predictors
concurvity(model_gam5, full=TRUE)

# checking Pearson correlation among the predictors and outcome;
# scale of the predictors and outcome: [0; 100]
cor(df_input5)
```

References

1. Kokol, P. Software Quality: How Much Does It Matter? *Electronics* **2022**, *11*, 2485. [CrossRef]
2. Galli, T.; Chiclana, F.; Siewe, F. Software Product Quality Models, Developments, Trends and Evaluation. *SN Comput. Sci.* **2020**, *1*, 154. [CrossRef]
3. Galli, T. Data Science Techniques for Modelling Execution Tracing Quality. Ph.D. Thesis, Institute of Artificial Intelligence, Faculty of Computing, Engineering and Media, De Montfort University, Leicester, UK, May 2022.
4. Kitchenham, B.; Pfleeger, S. Software Quality: The Elusive Target. *IEEE Softw.* **1996**, *13*, 12–21. [CrossRef]
5. Ouhbi, S.; Idri, A.; Fernández-Alemán, J.L.; Toval, A.; Benjelloun, H. Applying ISO/IEC 25010 on mobile personal health records. In *Proceedings of the HEALTHINF 2015—8th International Conference on Health Informatics, Proceedings; Part of 8th International Joint Conference on Biomedical Engineering Systems and Technologies, BIOSTEC 2015*; SciTePress: Setubal, Portugal, 2015; pp. 405–412.
6. Idri, A.; Bachiri, M.; Fernández-Alemán, J.L. A Framework for Evaluating the Software Product Quality of Pregnancy Monitoring Mobile Personal Health Records. *J. Med Syst.* **2016**, *40*, 50. [CrossRef] [PubMed]
7. Forouzani, S.; Chiam, Y.K.; Forouzani, S. Method for assessing software quality using source code analysis. In *Proceedings of the ACM International Conference Proceeding Series*; Association for Computing Machinery: New York, NY, USA, 2016; pp. 166–170. [CrossRef]
8. Domínguez-Mayo, F.J.; Escalona, M.J.; Mejías, M.; Ross, M.; Staples, G. Quality evaluation for Model-Driven Web Engineering methodologies. *Inf. Softw. Technol.* **2012**, *54*, 1265–1282. [CrossRef]
9. Idri, A.; Bachiri, M.; Fernandez-Aleman, J.L.; Toval, A. *Experiment Design of Free Pregnancy Monitoring Mobile Personal Health Records Quality Evaluation*; IEEE: Piscataway, NJ, USA, 2016; pp. 1–6. [CrossRef]
10. *ISO/IEC 25010:2011*; Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models. International Organization for Standardization: Geneva, Switzerland, 2011.
11. Shen, P.; Ding, X.; Ren, W.; Yang, C. Research on Software Quality Assurance Based on Software Quality Standards and Technology Management. In Proceedings of the 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Busan, Republic of Korea, 27–29 June 2018; pp. 385–390. [CrossRef]
12. Liu, X.; Zhang, Y.; Yu, X.; Liu, Z. A Software Quality Quantifying Method Based on Preference and Benchmark Data. In Proceedings of the 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Busan, Republic of Korea, 27–29 June 2018; pp. 375–379. [CrossRef]
13. Ulan, M.; Hönel, S.; Martins, R.M.; Ericsson, M.; Löwe, W.; Wingkvist, A.; Kerren, A. Quality Models Inside Out: Interactive Visualization of Software Metrics by Means of Joint Probabilities. In Proceedings of the 2018 IEEE Working Conference on Software Visualization (VISSOFT), Madrid, Spain, 24–25 September 2018; pp. 65–75. [CrossRef]
14. Kanellopoulos, Y.; Tjortjis, C.; Heitlager, I.; Visser, J. 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, Athens, Greece, 1–4 April 2008; pp. 63–72. [CrossRef]
15. Vetro, A.; Zazworka, N.; Seaman, C.; Shull, F. Using the ISO/IEC 9126 product quality model to classify defects: A controlled experiment. In Proceedings of the 16th International Conference on Evaluation Assessment in Software Engineering (EASE 2012), Ciudad Real, Spain, 14–15 May 2012; pp. 187–196. [CrossRef]
16. Parthasarathy, S.; Sharma, S. Impact of customization over software quality in ERP projects: An empirical study. *Softw. Qual. J.* **2017**, *25*, 581–598. [CrossRef]
17. Li, Y.; Man, Z. A Fuzzy Comprehensive Quality Evaluation for the Digitizing Software of Ethnic Antiquarian Resources. In Proceedings of the 2008 International Conference on Computer Science and Software Engineering, Wuhan, China, 12–14 December 2008; Volume 5, pp. 1271–1274. [CrossRef]
18. Hu, W.; Loeffler, T.; Wegener, J. Quality model based on ISO/IEC 9126 for internal quality of MATLAB/Simulink/Stateflow models. In Proceedings of the 2012 IEEE International Conference on Industrial Technology, Athens, Greece, 19–21 March 2012; pp. 325–330. [CrossRef]
19. *ISO/IEC 9126-1:2001*; Software Engineering—Product Quality—Part 1: Quality Model. International Organization for Standardization: Geneva, Switzerland, 2001.
20. Liang, S.K.; Lien, C.T. Selecting the Optimal ERP Software by Combining the ISO 9126 Standard and Fuzzy AHP Approach. *Contemp. Manag. Res.* **2006**, *3*, 23. [CrossRef]
21. Correia, J.; Visser, J. Certification of Technical Quality of Software Products. In Proceedings of the International Workshop on Foundations and Techniques for Open Source Software Certification, Madrid, Spain, 23 September 2008; pp. 35–51.
22. Andreou, A.S.; Tziakouris, M. A quality framework for developing and evaluating original software components. *Inf. Softw. Technol.* **2007**, *49*, 122–141. [CrossRef]
23. Letouzey, J.L.; Coq, T. The SQALE Analysis Model: An Analysis Model Compliant with the Representation Condition for Assessing the Quality of Software Source Code. In Proceedings of the 2010 Second International Conference on Advances in System Testing and Validation Lifecycle, Nice, France, 22–27 August 2010; pp. 43–48.
24. Letouzey, J.L. Managing Large Application Portfolio with Technical Debt Related Measures. In Proceedings of the Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), Rome, Italy, 14–15 September 2016; p. 181. [CrossRef]

25. Letouzey, J.L. The SQALE method for evaluating Technical Debt. In Proceedings of the Third International Workshop on Managing Technical Debt (MTD), Zurich, Switzerland, 5 June 2012; pp. 31–36. [CrossRef]
26. Letouzey, J.; Coq, T. The SQALE Models for Assessing the Quality of Real Time Source Code. 2010. Available online: https://pdfs.semanticscholar.org/4dd3/a72d79eb2f62fe04410106dc9fcc27835ce5.pdf?ga=2.24224186.1861301954.1500303973-1157276278.1497961025 (accessed on 17 July 2017).
27. Letouzey, J.L.; Ilkiewicz, M. Managing Technical Debt with the SQALE Method. *IEEE Softw.* **2012**, *29*, 44–51. [CrossRef]
28. Letouzey, J.L.; Coq, T. The SQALE Quality and Analysis Models for Assessing the Quality of Ada Source Code. 2009. Available online: http://www.adalog.fr/publicat/sqale.pdf (accessed on 17 July 2017).
29. Hegeman, J.H. On the Quality of Quality Models. Master's Thesis, University Twente, Enschede, The Netherlands, 2011.
30. Letouzey, J.L. The SQALE Method for Managing Technical Debt, Definition Document V1.1. 2016. Available online: http://www.sqale.org/wp-content/uploads//08/SQALE-Method-EN-V1-1.pdf (accessed on 2 August 2017).
31. Gleirscher, M.; Golubitskiy, D.; Irlbeck, M.; Wagner, S. Introduction of static quality analysis in small- and medium-sized software enterprises: Experiences from technology transfer. *Softw. Qual. J.* **2014**, *22*, 499–542. [CrossRef]
32. Wagner, S.; Lochmann, K.; Heinemann, L.; as, M.K.; Trendowicz, A.; Plösch, R.; Seidl, A.; Goeb, A.; Streit, J. The Quamoco Product Quality Modelling and Assessment Approach. In Proceedings of the 34th International Conference on Software Engineering, ICSE '12, Zurich, Switzerland, 2–9 June 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 1133–1142.
33. Wagner, S.; Lochmann, K.; Winter, S.; Deissenboeck, F.; Juergens, E.; Herrmannsdoerfer, M.; Heinemann, L.; Kläs, M.; Trendowicz, A.; Heidrich, J.; et al. The Quamoco Quality Meta-Model. October, 2012. Available online: https://mediatum.ub.tum.de/attfile/1110600/hd2/incoming/2012-Jul/517198.pdf (accessed on 18 November 2017).
34. Wagner, S.; Goeb, A.; Heinemann, L.; Kläs, M.; Lampasona, C.; Lochmann, K.; Mayr, A.; Plösch, R.; Seidl, A.; Streit, J.; et al. Operationalised product quality models and assessment: The Quamoco approach. *Inf. Softw. Technol.* **2015**, *62*, 101–123. [CrossRef]
35. Kothapalli, C.; Ganesh, S.G.; Singh, H.K.; Radhika, D.V.; Rajaram, T.; Ravikanth, K.; Gupta, S.; Rao, K. Continual monitoring of Code Quality. In Proceedings of the 4th India Software Engineering Conference 2011, ISEC'11, Kerala, India, 24–27 February 2011; pp. 175–184. [CrossRef]
36. Plösch, R.; Gruber, H.; Hentschel, A.; Körner, C.; Pomberger, G.; Schiffer, S.; Saft, M.; Storck, S. The EMISQ method and its tool support-expert-based evaluation of internal software quality. *Innov. Syst. Softw. Eng.* **2008**, *4*, 3–15. [CrossRef]
37. Plösch, R.; Gruber, H.; Körner, C.; Saft, M. A Method for Continuous Code Quality Management Using Static Analysis. In Proceedings of the 2010 Seventh International Conference on the Quality of Information and Communications Technology, Porto, Portugal, 29 September–2 October 2010; pp. 370–375. [CrossRef]
38. Mordal-Manet, K.; Balmas, F.; Denier, S.; Ducasse, S.; Wertz, H.; Laval, J.; Bellingard, F.; Vaillergues, P. The Squale Model—A Practice-based Industrial Quality Model. Available online: https://hal.inria.fr/inria-00637364 (accessed on 6 March 2018).
39. Laval, J.; Bergel, A.; Ducasse, S. Assessing the Quality of your Software with MoQam. Available online: https://hal.inria.fr/inria-00498482 (accessed on 6 March 2018).
40. Balmas, F.; Bellingard, F.; Denier, S.; Ducasse, S.; Franchet, B.; Laval, J.; Mordal-Manet, K.; Vaillergues, P. Practices in the Squale Quality Model (Squale Deliverable 1.3). October, 2010. Available online: http://www.squale.org/quality-models-site/research-deliverables/WP1.3Practices-in-the-Squale-Quality-Modelv2.pdf (accessed on 16 Novenber 2017).
41. INRIA RMoD, Paris 8, Qualixo. Technical Model for Remediation (Workpackage 2.2). 2010. Available online: http://www.squale.org/quality-models-site/research-deliverables/WP2.2Technical-Model-for-Remediationv1.pdf (accessed on 16 November 2017).
42. Khaddaj, S.; Horgan, G. A Proposed Adaptable Quality Model for Software Quality Assurance. *J. Comput. Sci.* **2005**, *1*, 482–487. [CrossRef]
43. Horgan, G.; Khaddaj, S. Use of an adaptable quality model approach in a production support environment. *J. Syst. Softw.* **2009**, *82*, 730–738. [CrossRef]
44. Boehm, B.; Chulani, S. *Modeling Software Defect Introduction and Removal—COQUALMO (Constructive QUALity Model)*; Technical Report, USC-CSE Technical Report; University of Southern California, Center for Software Engineering: Los Angeles, CA, USA, 1999.
45. Madachy, R.; Boehm, B. Assessing Quality Processes with ODC COQUALMO. In *Making Globally Distributed Software Development a Success Story*; Springer: Berlin/Heidelberg, Germany, 2008; Volume 5007, pp. 198–209. [CrossRef]
46. Grady, R.B.; Caswell, D.L. *Software Metrics: Establishing a Company-Wide Program*; Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1987.
47. Grady, R.B. *Practical Software Metrics for Project Management and Process Improvement*; Prentice Hall: Upper Saddle River, NJ, USA, 1992.
48. Eeles, P. Capturing Architectural Requirements. 2005. Available online: https://www.ibm.com/developerworks/rational/library/4706-pdf.pdf (accessed on 19 April 2018).
49. Côté, M.A.; Suryn, W.; Martin, R.A.; Laporte, C.Y. Evolving a Corporate Software Quality Assessment Exercise: A Migration Path to ISO/IEC 9126. *Softw. Qual. Prof.* **2004**, *6*, 4–17.
50. Kim, C.; Lee, K. Software Quality Model for Consumer Electronics Product. In Proceedings of the 9th International Conference on Quality Software, Jeju, Republic of Korea, 24–25 August 2008; pp. 390–395.
51. Georgiadou, E. GEQUAMO—A Generic, Multilayered, Customisable, Software Quality Model. *Softw. Qual. J.* **2003**, *11*, 313–323. [CrossRef]

52. Benedicenti, L.; Wang, V.W.; Paranjape, R. A quality assessment model for Java code. In Proceedings of the Canadian Conference on Electrical and Computer Engineering, Winnipeg, MB, Canada, 12–15 May 2002; Volume 2, pp. 687–690.
53. McCall, J.A.; Richards, P.K.; Walters, G.F. Factors in Software Quality, Concept and Definitions of Software Quality. 1977. Available online: http://www.dtic.mil/dtic/tr/fulltext/u2/a049014.pdf (accessed on 6 March 2018).
54. Zhang, L.; Li, L.; Gao, H. 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, Washington, DC, USA, 10–12 December 2008; pp. 1147–1152. [CrossRef]
55. Boehm, B.W.; Brown, J.R.; Lipow, M. Quantitative Evaluation of Software Quality. In Proceedings of the 2nd International Conference on Software Engineering, San Francisco, CA, USA, 13–15 October 1976.
56. Dromey, R. A Model for Software Product Quality. *IEEE Trans. Softw. Eng.* **1995**, *21*, 146–162. [CrossRef]
57. van Solingen, R.; Berghout, E. *The Goal/Question/Metric Method a Practical Guide for Quality Improvement of Software Development*; McGraw Hill Publishing: London, UK, 1999.
58. *IEEE Stdandard 1061-1998*; IEEE Standard for a Software Quality Metrics Methodology; IEEE: Piscataway, NJ, USA, 1998.
59. Franke, D.; Weise, C. Providing a software quality framework for testing of mobile applications. In Proceedings of the 4th IEEE International Conference on Software Testing, Verification, and Validation (ICST 2011), Berlin, Germany, 21–25 March 2011; pp. 431–434. [CrossRef]
60. Hyatt, L.E.; Rosenberg, L.H. A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality. In *Proceedings of the Product Assurance Symposium and Software Product Assurance Workshop, 19–21 March 1996*; EAS SP-377; European Space Agency: Paris, France, 1996.
61. Martin, R.A.; Shafer, L.H. 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, Chicago, IL, USA, 23–27 May 1996.
62. Kitchenham, B.; Linkman, S.; Pasquini, A.; Nanni, V. The SQUID approach to defining a quality model. *Softw. Qual. J.* **1997**, *6*, 211–233. [CrossRef]
63. Han, S.; Sinha, R.; Lowe, A. Assessing Support for Industry Standards in Reference Medical Software Architectures. In Proceedings of the IECON 2020 the 46th Annual Conference of the IEEE Industrial Electronics Society, Singapore, 18–21 October 2020; pp. 3403–3407. [CrossRef]
64. Bernardes Boarim, J.; Cavalcanti da Rocha, A.R. CRM Systems Quality Evaluation. In Proceedings of the XX Brazilian Symposium on Software Quality, SBQS '21, New York, NY, USA, 8–11 November 2021. [CrossRef]
65. Galli, T.; Chiclana, F.; Siewe, F. On the Use of Quality Models to Address Distinct Quality Views. *Appl. Syst. Innov.* **2021**, *4*, 41. [CrossRef]
66. SonarSource. SonarQube. 2017. Available online: https://www.sonarqube.org (accessed on 16 February 2018).
67. Galli, T.; Chiclana, F.; Siewe, F. Genetic Algorithm-Based Fuzzy Inference System for Describing Execution Tracing Quality. *Mathematics* **2021**, *9*, 2822. [CrossRef]
68. Hastie, T.; Tibshirani, R. Generalized Additive Models. *Stat. Sci.* **1986**, *1*, 297–310. [CrossRef]
69. Larsen, K. GAM: The Predictive Modeling Silver Bullet. 2015. Available online: Multithreadedstitchfix.com (accessed on 3 April 2023).
70. Galli, T.; Chiclana, F.; Siewe, F. *Data Set Generated by the Fuzzy Model Constructed to Describe Execution Tracing Quality [Data Set]*; Zenodo: Geneva, Switzerland, 2023. [CrossRef]
71. Poryazov, S.A.; Saranova, E.T.; Andonov, V.S. Overall Model Normalization towards Adequate Prediction and Presentation of QoE in Overall Telecommunication Systems. In Proceedings of the 2019 14th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS), Nis, Serbia, 23–25 October 2019; pp. 360–363. [CrossRef]
72. Malhotra, N.H. *Marketingkutatas (Translated Title: Marketing Research)*; Akademia Kiado: Budapest, Hungary, 2009.
73. Galli, T.; Chiclana, F.; Siewe, F. Quality Properties of Execution Tracing, an Empirical Study. *Appl. Syst. Innov.* **2021**, *4*, 20. [CrossRef]