Software Quality is Multidimensional: Let's play with Tensors

AUTOR 1
ENGENHARIA DE SOFTWARE
Brasil
autor1@eng.software.br

AUTOR 2
ENGENHARIA DE SOFTWARE
Brasil
autor2@eng.software.br

ABSTRACT

In the face of the dynamic nature of modern software systems, the quality of software products must be monitored and assured from different perspectives at the same time and throughout the development cycle. Although different characteristics and sub-characteristics can represent a software product quality, they have been historically studied and observed in a unidimensional way, which restricts the ability to observe the multiple relationships among quality characteristics. There is empirical evidence indicating the multidimensionality of this phenomenon. Therefore, an *n*-dimensional software quality base model is presented. The fundamentals of tensor algebra provide its basis, which explores the combination of different dimensions and relations of quality (internal, external, and in use) to suggest the level of quality of a software product. Preliminary results of its operationalization are presented to support this proposal.

CCS CONCEPTS

Software and its engineering → Software creation and management; Quality Assurance

KEYWORDS

Software Product Quality, Release Acceptance, Software Engineering, Software Quality Model, Software Metric and Measurement, Tensor, Empirical Software Engineering

ACM Reference format:

G. Gubbiotti, P. Malagò, S. Fin, S. Tacchi, L. Giovannini, D. Bisero, M. Madami, and G. Carlotti. 1997. SIG Proceedings Paper in word Format. In *Proceedings of ACM Brazilian Symposium on Software Engineering, Natal, Rio Grande do Norte Brazil, October 2020 (SBES'20)*, six pages. https://doi.org/10.1145/123 4

1 INTRODUCTION

The quality of software products has been an object of interest to software engineers and researchers for nearly five decades. The knowledge accumulated from the research related to the software quality product (SQP) enabled the identification of different factors and sub-factors that allow the description and observation of the phenomenon (quality). These factors and sub-factors have been organized and grouped by different quality models proposed over time, thereby establishing the definition of a hierarchical structure of characteristics and sub-characteristics describing the observed SQP properties. Likewise, we could see that the models proposed recently address the operationalization between these characteristics and sub-characteristics, said to be conceptual and abstract, to a set of metrics and measures to make the analyses more concrete and objective.

There is empirical evidence emphasizing the need to observe the multiple relationships between the quality characteristics and sub-characteristics and to expose and explain the multiple relationships and side effects among them when analyzing SQP [9-12].

However, we could also see that historically the quality of software products has been observed using a one-dimensional way [1-8]. The assessment of a quality characteristic per time without considering side effects and mutual influences among characteristics constraints the observation. These constraints can make quality goals unachievable in practice, in addition to inducing software engineers and managers to make inaccurate or even wrong decisions.

Therefore, this work presents a spatial software quality model. What primarily differentiates this model from the previous ones is the multidimensional observation of quality. In this spatial model, the analysis of software quality takes place in the tensor space. For instance, tensors are multidimensional arrays of numerical values; thus, generalize vectors to multiple dimensions.

Some starting ideas were discussed in [13]. This work goes one step further on describing an approach to observe SQP in the *n*-dimensional space. It considers the differences in the measurement scales and their need for aggregation and operationalization. This spatial model intends to support the following research question:

How to observe the software quality in a multidimensional way, based on the combination of different dimensions and quality relations (internal, external, in use) of the product throughout its development cycle?

The following sections depict the proposed model on how to use a multidimensional perspective to observe SQP in modern software engineering environments. Besides this introduction, Section 2 describes basic concepts regarding this research. Next, we present the mathematical model. Subsequently, we discuss the results from a preliminary and observational study. Section 5 presents some conclusions for this ongoing research.

2 BACKGROUND AND RELATED WORKS

2.1 Measurement and Metrics

Usually, one of the first questions that come to mind for software practitioners regards the metrics that should be used to monitor the quality of a software product throughout its life cycle. However, the most appropriate answer depends on the context. Different issues can influence the response, such as the strategic objectives of an organization [14], measurement objectives [15], programming languages, and paradigm [16], or the used development technologies. Therefore, when defining a measurement plan, the context information must be treated and observed according to the need of each stakeholder interest [17].

A metric is the numerical value of raw data extracted from an information source. The measure, on the other hand, can be understood as a number or a symbol assigned to an entity to characterize an attribute [19]. Therefore, the measure provides interpretation meaning to a metric. Although there is no single definition [20], metrics and measures are often associated with the evaluation of characteristics of the development process or of the software product itself, whether by looking at qualitative or quantitative aspects, as a way of perceiving and understand the software phenomenon. It makes no sense to observe a metric outside the context, without an interpretation method, and without evidence on its theoretical or experimental validation.

Over the last five decades, there are studies on the definition and choice of metrics and measures to represent properties associated with the software [18, 19]. Several metrics studied and used in the industry and academy appear in secondary studies that sought to their mapping [21-23].

The maturing of the discussions and the use of agile development practices, free software, and the Lean movement mentality contributed to the establishment of a confluent view on the cadence of the development cycle. They started to carry out development process activities continuously to deliver releases in short and frequent intervals of time [24].

Taking this view into consideration, from the last decade, it was possible to observe the flourishing of studies on the use of the continuous experimentation approach to support the decision making on software product versions. It is interesting to note that this strategy focuses on the observation of characteristics from the use of the software. As a consequence, the measures and metrics captured are predominantly related to quality in use [25-27].

2.1.2 Interpretation Metrics and Thresholds. From the moment a software engineer carefully chooses the metrics, it is necessary to have a mechanism to support of quantifying and providing judgment, in subjective terms, of how good or bad the results observed with the metrics can be interpreted. An interpretation function provides this behavior, which can be automatically defined or based on expert opinion [7].

An interpretation function characterizes and normalizes an evaluated measure. It makes use of the linear interpolation function between the value that comes from a metric, and its thresholds. The values assumed by an interpretation function correspond to an open range between 0 and 1, where 0 = absence of quality (worse) and 1 = maximum quality (best). This appliance was proposed and empirically validated [5,7,8], and it is a reference for the spatial model proposed in this work.

The definition of the threshold is decisive in the interpretation of the metrics and measures. Without such values, this activity becomes unworkable. It is quite a sensitive and non-trivial issue in software engineering since the logical and intangible nature of a software product makes it impossible to use parameters defined and supported by the laws of physics.

It can be noticed the application of two treatment approaches to defining these parameters, considering the reference value definition strategies used by the recently proposed quality model: i) based on the opinion of quality experts [7,8], and ii) obtained automatically from the benchmarking in free software product repositories [4,16,18,28-31]. In the analyzed studies, these approaches were used alone or in combination [5,6]. In the technical literature, the methods proposed for the automatic derivation of thresholds consider the analysis of the frequency of values observed in the statistical distribution of metrics and measures. Interviews, questionnaires, and workshops involving quality experts from the industry and academia have been the most used instruments to define thresholds based on the opinion of quality experts.

Some methods initially proposed to observe metrics in the object-oriented paradigm (OO) suggest that it is viable to obtain the thresholds by analyzing the average frequency of values found in distribution [16]. In [28], the authors proposed a strategy for detecting anomalies in source code (bad smells). Also, they used the value of the mean and standard deviation as the basis. As "low" reference value, $\mu + \sigma$ as "high" and 1.5 $(\mu + \sigma)$ as "very high" reference value. These methods take the assumption that the source code metrics distributions assume a normal distribution [29]. However, the distribution of software metrics, especially those of source code, hardly follows the normal distribution. Therefore, it is necessary to consider other cut-off points and fit of the distribution.

Still taking into account the automatic derivation strategies, some studies show that the analysis of quartiles and percentiles are more accurate compared to the use of the mean and standard deviation, and therefore, more suitable for benchmarking strategies [5,6,18,30,31].

The interquartile analysis approach *IRQ*(25%-75%) is regarded as more suitable for defining a reference threshold than using the mean and standard deviation. However, it may not capture significant values for a software metric. In [31] the author analysed OO metrics observing the percentiles (5%, 10%, 25%, 50%, 75%, 90%, 95%, 99%) and proposed the definition of a threshold range, such as "very frequent", "frequent", "less frequent" and "not frequent". The conclusion of this study shows that some OO metrics only begin to demonstrate observable from the 50% percentile and in some cases from the 90% percentile, even showing an abrupt growth. This proposal was adopted in [18] to investigating annotation metrics in source code. The "very frequent" value corresponds to the average obtained from the values found in the 90% percentile. The "frequent value" refers to the average derived from the values found in the 95% percentile, and the "less frequent" value comprises the average obtained from percentile 99%. In this study, the behavior observed in the distributions of annotation metrics in source code was similar to that found in [31].

Although the automatic percentile analysis approach is suitable for watching metrics in different domains, it cannot be generalized, especially if the context characteristics and measurement objectives are not clearly defined. There are many unknown confounding factors when benchmarking software repositories. In [5,6], the authors used a mixed approach, where they submitted the thresholds automatically obtained to the judgment and analysis of specialists. Already in [7,8], the authors used the opinion of experts for these definitions in a more analytical approach.

Finally, it is observable that the ideal scenario for dealing with the definition of thresholds for software product metrics and measures is the combined use of automatic approaches and expert opinion-based. If the organization has a historical database, it would be preferable to the use of free software repositories since the context

characteristics will be more similar. Alternatively, it can use a base of free software projects in the benchmarking strategy. Therefore, one must consider carefully measurement the plan must be elaborated. This plan must explain and addresses the issues discussed in this section. Only then will it be possible to represent software quality in multidimensional arrays, and thus, analyze the numerical representation of the multiple dimensions and attributes of quality in a spatial way. With that, we can expose and explain the side effects perceived by the data.

2.2 Basics of Tensors

N-dimensional vector spaces with p-linear applications are objects of study in multilinear algebra. In this sense, the notion of tensors can be understood as the generalization of the concepts of scalars, vectors, and matrices, for arrays of larger orders, that is, equal to or greater than three dimensions. Therefore, a tensor is a multidimensional array, and its dimensions are usually known as modes. Thus, the dimensionality of a tensor is defined by its order, represented by its modes. Therefore, a vector is a particular case of a tensor of order 0, just as a third-order tensor has mode 3.

Due to its expressiveness in modeling phenomena that are observed in a multidimensional way, tensors have typical applications in fields of physics and mathematics, such as studies on mechanical tension and gravity. Especially in this decade in the computer science area, the utilization of the tensors proved to be very efficient. It has been showing the maturity of its application in areas such as, e.g., signal processing and computer vision; natural language processing; smart cities, information retrieval on the web in the analysis of brain signal and various others applications as it is possible to see from a secondary study in [33].

Due to the lack of space, the formalization and the operations carried out in the tensor space will not be presented here. For instance, we adopted the notation and formalism defined in [32,33].

3 A MULTIDIMENSIONAL BASE MODEL

3.1 Structure and Operations

The proposed quality model follows the hierarchical structure of characteristics and sub-characteristics of the reference quality models in the software engineering literature. Also, it relies on the operationalization mechanism between these characteristics and their respective measures and metrics.

In *MODEL*, a quality configuration $\boldsymbol{SQC_i}$ corresponds to the measurement and evaluation of a particular version of the software in a certain period of its life cycle. Each $\boldsymbol{SQC_i}$ is composed of a set of tensors named $\boldsymbol{TQC_i}$, which represents a specific quality characteristic, defined in a reference model. In these tensors, each dimension $dim_{(N)}(\boldsymbol{TQC_i})$ represents a specific sub-characteristic of quality related to a particular quality characteristic. For example, to represent the Compatibility characteristic described in ISO 25010, it would have a tensor with two dimensions representing the sub-characteristics of Coexistence and Interoperability. The sub-characteristics, in turn, are represented numerically by a set of scalar measures, called evaluated measures am_i. Each evaluated measure describes a function called interpretation $\mathbf{FI}_{i,\dots,N} = f_i : X \to Y \mid x,y \in \mathbb{R}$ e $x,y > 0$ e $0 \leq f_i \leq 1$. An interpretation function represents the application of a mathematical function, which maps how a given measure is interpreted $am_{i,\dots,N} \in \mathbb{R} \mid am_i = \frac{\sum_{i=1}^{\mathbf{Tm}} \mathbf{FI}_i}{\mathbf{Tm}_i}$ e $i > 0$. When defining an interpretation function, a set of software metrics $\mathbf{m}_{1,\dots,N}$ is selected to provide an interpretation meaning, that is, the interpretation rule of an evaluated measure. Besides, we statistically standardize the measures, which means that the metrics collected at different scales and from various sources of information are standardized for the same scale. This standardization enables the proportional comparison of measurements in software products of different sizes.

Definition: a tensor represents a software quality configuration $\boldsymbol{SQC} \in \mathbb{R}^N$ which models and represents a set of tensors associated with characteristics describing the quality properties of the software product. Each dimension or axis of the tensor represents a specific characteristic established in reference models. It is defined as $\boldsymbol{SQC}_{(1,2,\dots,i)} \in \mathbb{R}^{I_1 \times I_2 \times,\dots,\times I_N} \mid i > 0$ e $N \geq 0$, where:

I_N = number of dimensions of a quality setting. Therefore, $dim_{(N)}(\boldsymbol{SQC_i})$ corresponds to the i-th dimension of a quality configuration. It depicts the set of measures observed in a quality characteristic.

Definition: a quality tensor, $\boldsymbol{TQC} \in \mathbb{R}^N$, models, and represents a quality characteristic of the software product established in reference models. Each dimension or axis of the tensor represents a specific sub-characteristic and is defined as follows:

$\boldsymbol{TQC}_{(1,2,\dots,i)} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}\ i > 0$ e $N \geq 0$, where:

I_N = number of dimensions of a quality tensor. Therefore, $dim_{(N)}(\boldsymbol{TQC_i})$ corresponds to the i-th dimension of a quality tensor. In Fig. 1, we can see the conceptual structure of the proposed model.

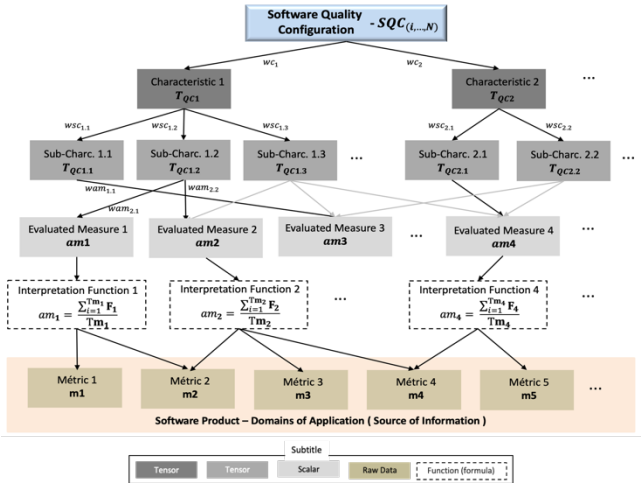

Figure 1: The conceptual structure of the model

It is necessary to define which software quality metrics will be extracted to start the operationalization in the proposed model, and mainly why they should be chosen. After that, the next step is to aggregate the measures evaluated in their respective sub-characteristics of quality.

3.1.1 Aggregation and Weighting. The weighting of different characteristics and sub-characteristics observed from the measures precedes the aggregation of the model. The weights represent the

relative importance of each measure evaluated regarding its respective sub-characteristic. In the same way, the sub-characteristics concern their characteristics, and such characteristics concern the system as a whole.

Unlike the definition of thresholds, weights must be defined by specialists. It is because the perception of quality is very subjective and very dependent on the context. The definition of how a measure or characteristic influences the overall quality of the system depends on how software engineers, managers, and users perceive the notion of quality. Consequently, the definition of weights also carries this subjective aspect [6]. Therefore, this calibration step is of paramount importance, and the software engineers must perform it meticulously.

Definition: let wam_i , wsc_i and wc_i the weights respectively associated with an evaluated measure am_i with a dimension of a quality tensor $dim_{(N)}(\boldsymbol{TQC}_i)$ and one dimension of a quality configuration $dim_{(N)}(\boldsymbol{SQC}_i)$. At the measure level, direct multiplication is carried out between the measure and its weight. At the level of sub-characteristics, through the slice tensor operation, each of the dimensions of a tensor is recovered. Then the Kronecker product is made by its respective weight. Finally, at the level of the characteristics, each dimension of a quality configuration is recovered, and the same procedure as in sub-characteristics is applied, where:

$$am_{\mathbf{1},\dots,\boldsymbol{i}} \in \mathbb{R} \mid am_i = am_i \times wam_i \text{ e } i > 0 \text{ and } wam_i > 0$$

$$dim_{(N)}(\boldsymbol{TQC}_i) \in \mathbb{R}^1 \mid dim_{(N)}(\boldsymbol{TQC}_i) = dim_{(N)}(\boldsymbol{TQC}_i) \otimes wsc_i \text{ e } i > 0 \text{ e } wsc_i > 0$$

$$dim_{(N)}(\boldsymbol{SQC}_i) \in \mathbb{R}^1 \mid dim_{(N)}(\boldsymbol{SQC}_i) = dim_{(N)}(\boldsymbol{SQC}_i) \otimes wc_i \text{ e } i > 0 \text{ e } wc_i > 0$$

Once the elements of the model were appropriately weighted, then the results can be aggregated and analyzed in a summarized way. The application of the tensor standard gives the aggregation function of the model. This strategy differs from that proposed in the recent software quality models so far, which perform the aggregation with the use of a weighted average.

Definition: the aggregation of model elements is defined by the Frobenius norm, which is the standard for tensors. Let a tensor $\boldsymbol{SQC} \in \mathbb{R}^N$. The overall system quality is obtained as follow:

$$\|\boldsymbol{SQC}\|_{\mathcal{F}} = \sqrt{\sum_{i_1}^{I_1} \sum_{i_2}^{I_2} \dots \sum_{i_N}^{I_N} \boldsymbol{SQC}(i_1, i_2, \dots, i_n)^2} \quad (1)$$

The aggregated value per se is not the most relevant aspect of the model. The most important thing is that it captures and represents the subjective notion of software quality in the multidimensional space to understand the variations that have occurred over the life cycle of the product. We present an abstraction of the general view of the model in Fig. 2.

3.1.2 Comparison between software quality configurations. The comparison activity between different versions of software products is part of day-to-day development, and it occurs in various granularities.

It is critical, especially in modern software development environments, such as in continuous development and experimentation, where the decision and choice between a candidate version A and a candidate version B are given by the results obtained from the execution of controlled experiments. Therefore, software engineers and managers must have mechanisms that help them make comparisons and support data-driven and evidence-based decision making.

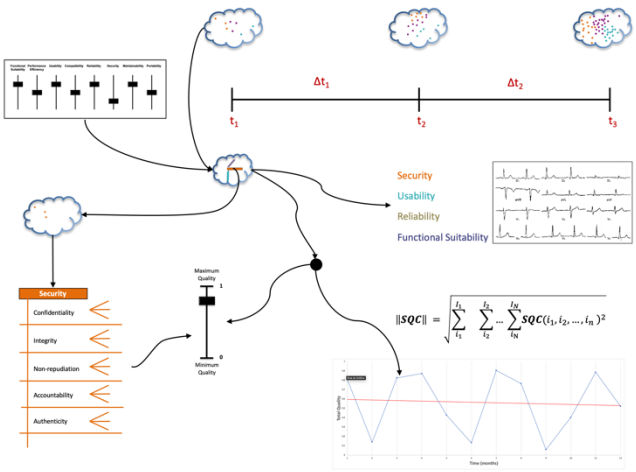

Figure 2: An abstraction of the model

In the proposed model, two types of comparisons (partial and total) are possible. Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be two tensors defining two quality configurations $\boldsymbol{SQC}_i$, and $\boldsymbol{SQC}_{i+N}$ that represent two versions of different software products.

Definition: two tensors of quality $\boldsymbol{X}$ and $\boldsymbol{Y}$ are partially comparable if and only if: (a) there is at least one $dim_{(i)}(\boldsymbol{X}) \cap dim_{(i)}(\boldsymbol{Y}) \neq \emptyset \mid i \text{ e } N > 0$

Definition: two $\boldsymbol{X}$ and $\boldsymbol{Y}$ quality tensors are fully comparable if and only if: (a) for any $dim_{(i)}(\boldsymbol{X}) \cap dim_{(i)}(\boldsymbol{Y}) \neq \emptyset \mid i \text{ e } N > 0$ and (b) there is the same amount of dimensions

A measure of similarity between vectors highlighted in the literature and widely used in areas such as information retrieval, indexing, is the cosine difference between two vectors [35]. We use the measure of similarity between vectors to quantify the comparison between two different versions of the software. Thus, the interpretation is that more closer to zero is the cosine of the angle θ more similar two vectors are. Consequently, two releases are equivalent to whether the vectors that represent them have similarity equal one.

An intrinsic behavior to vector spaces is the orthogonality. This situation will occur when the tensors do not have any characteristics or sub-characteristics of quality in common. Thus, their comparison does not make sense, and they should not be compared. As the condition of comparison between two tensors defined in this model requires that there be at least one dimension in common between them. So, the proposed model does not compute orthogonal tensors.

The orientation of the vectors is another behavior that needs to be addressed. Vectors with opposite directions have no convergence point, even though they have the same module. The analysis of the sine and cosine signals are taken in place to address this issue.

According to the quadrant in the three-dimensional coordinate system (Euclidean plane), the values of sine and cosine vary, and they can assume negative values. As in the proposed model, the values assumed by quality are always positive, between 0 and 1, it means that a vectorized tensor will be projected on the 1st quadrant, more specifically on the 1st octant, so the values of sine and cosine will always be positive. It reduces the analysis space for vectors projected in the 1st quadrant, which simplifies it and prevents such an anomaly in the model.

These intrinsic properties of vector spaces, if not observed and adequately treated, can lead to anomalies and unpredictable behaviors in the perception of managers and software engineers, who might mistakenly decide due to radical and intangible changes caused in the development cycle of the software product.

3.1.3 Capturing and exposing the multiple relationships between quality characteristics. According to empirical evidence, the mutual influences between quality characteristics restrict us from setting maximum quality measurement objectives for all quality characteristics at the same time. In other words, it is unlikely to define a quality configuration that considers in which the absolute totality of quality characteristics simultaneously [9,11,34]. Thus, planning the objectives of measurement product quality without considering the mutual relationships between the different quality attributes can lead to unreachable situations during development. These situations have an impact on effort, costs, and time on the produced version. To capture the expectation of planned quality requirements in a quantitative way, in addition to lead with these trade-offs and conflicts between quality attributes, the model provides a quality equalizer mechanism.

Most studies on balancing quality attributes use ISO 9126 as a reference, since its update, ISO 25010, is more recent. However, in [34], the authors propose a quality attribute relationship matrix based on ISO 25010. After analyzing the conflicts found in this study, some of them were resolved due to the results presented in [12]. With this, it was able to mitigate four conflicts of the ten pointed out, as shown in Table 1.

Table 1: Mitigation of the Quality Attributes Conflicts in *MODEL*

[34]	[12]	Proposed Mitigation	Quality Characteristic
± –	–	–	Maintainability X Reliability

After reducing these conflicts, a new matrix was generated to be the reference for calibrating the trade-offs between the characteristics of ISO 25010 initially adopted in the model, as shown in Fig. 3. The other relationships still follow the proposal presented in [34].

ISO 25010	Functional Suitability	Performance Efficiency	Usability	Compatibility	Reliability	Security	Maintainability	Portability
Functional suitability	❖	–	+	o	+	–	+	o
Performance efficiency	–	❖	–	–	o	–	–	–
Usability	+	–	❖	o	+	o	o	o
Compatibility	o	o	o	❖	o	–	±	+
Reliability	+	o	+	o	❖	o	+	o
Security	o	–	–	–	+	❖	o	o
Maintainability	+	–	o	+	+	±	❖	+
Portability	o	–	o	+	o	o	+	❖

Figure 3: Balance matrix between characteristics of ISO25010

This matrix is not symmetric. This type of matrix presents such an idiosyncrasy because of the multiplying effect that an attribute A causes on an attribute B is not necessarily the same as the rising effect that an attribute B will cause on an attribute A [36].

So, there are conflicts, and they need to be treated when planning the releases, in an agreement view of quality experts, managers, software engineers, and the user's priority to achieve possible measurement objectives for a version of the software product.

Once these objectives were defined, the proposed model quantifies and represents them in a quality tensor, thus making it possible to compare the planned quality requirements and the results accomplished in the software release. It is typical in software projects to compare the planned with the executed.

To capture the subjective perception expressed in the planned quality requirements and turn them into a numerical representation that can be quantitatively comparable, the model presents the characterization of the quality from the user perspective through a characteristics equalizer on a visual analog scale (VAS). VAS are continuous rating scales. This type of scale presents an advantage over the ordinal one, with a discrete classification, such as the 5-point scale, which is: answers are not restricted to a discrete set of points. So, the exact nuances can be measured. Besides, more statistical tests can be used to analyze the data collected on variables represented on the VAS scale, making the fit made to the distribution more reliable. This type of scale is particularly useful for capturing subjective and relative perceptions in responses to online questionnaires. On a VAS scale, each chosen position corresponds to a point on the scale; therefore, a possible value assumed by the variable [37].

Afterward catching the position defined on the scale, a transformation from the VAS to a ratio scale is performed. Subsequently, this transformation, the perception of prioritization of the planned quality requirements are quantified, and it becomes able to be represented in a quality setting $\boldsymbol{SQC}_{i\,(planned)}$. At the end of the production cycle, the quality is calculated, aggregated, and the results are represented in $\boldsymbol{SQC}_{i\,(realized)}$. In this way, it allows us to apply the defined comparison rules.

As defined, the aggregation of the model is made from the calculation of the norm of quality tensors. The model uses this aggregated representation of quality to analyze the mutual relations between the characteristics in a quantitative way and from the data obtained in the release. In order to analyze the effects of multiple relationships between quality characteristics, the correlation matrix between the tensors is calculated, considering the norm for each tensor. The comprehension of this matrix provides us with a perception of how each quality characteristic impacts another one. When the observed correlation is positive, it indicates that the relationship between the characteristics occurs directly; that is, when the values of a characteristic A increase, then the values of a characteristic B also increase. When a negative correlation is given, it indicates that the relationship between the characteristics occurs inversely; that is, when the values of a characteristic A increase, then B decrease. When the correlation is null, it indicates that the two characteristics do not influence each other; that is, they are independent.

4 PRELIMINARY OBSERVATIONAL STUDY

It was executed an initial observational and exploratory study to investigate the feasibility of the proposed model to represent software quality in a multidimensional way. The study was carried out on a small product, called *PW*. It is a software system that aims to support the diagnosis of neglected tropical diseases. This project was executed through a partnership between *UF* and the *YY* Foundation. The development team was composed of undergraduate students at *UF*, as well as students and professors from the masters and doctorate programs at *PPG*. In Fig. 4. It is presented an illustrative view of results from the operationalization of *MODEL* in the *PW* project.

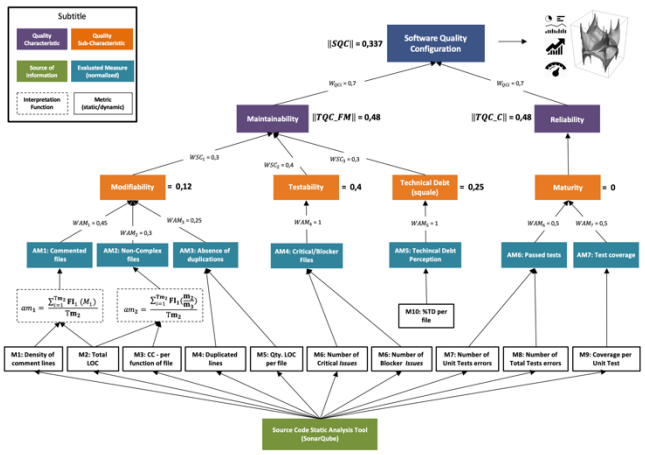

Figure 4: Operationalizing the *MODEL* to *PW* project

The version analyzed was the 1st release. The project in this version contained 1.081 LOC, 18 files, 28 classes, nine directories, 120 commits, and was written in Python. Seven measures were observed, involving ten metrics extracted with the SonarQube tool. One evaluated measure was defined in the SQUALE model[4] and the others in the Q-Rapids model[7]. The evaluated measures were represented in two quality tensors, and these defined the quality configuration.

5 CONCLUSIONS

In this research, it is provided the basis of the mathematical model to represent the multidimensionality of software quality. Besides, it presented the formalism of the model properties and the necessary steps for its instantiation and operationalization.

Some initial indications of the model feasibility of representing the software quality in the multidimensional space were taken after setting up the model and using it in a small, but a real project. As an emerging idea and ongoing research, further investigation is necessary to provide a better characterization of its efficiency and efficacy in managing and capturing all nuances regarding the quality of a software product.

These results encouraged us to evolve the model, and empirically evaluate it. The next steps include the planning and execution of experimental studies to support the characterization and understanding of the proposed model.

ACKNOWLEDGMENTS

This work was supported by the University of *UF* and the CAPES. Prof. *PROF* is a CNPq researcher (grant 000000/0000)

REFERENCES

[1] McCall, J.A. & Richards P.K. & Walters G.F, Factors in Software Quality. Rome Air Development Center Reports, US Dept. of Commerce, USA, 1977.
[2] Boehm B. W., Characteristics of Software Quality, Volume 1, TRW, 1978.
[3] Dromey, R. G. A Model for Software Product Quality. IEEE Trans. Softw. Eng., v. 21, n. 2, p. 146–162, fev. 2013
[4] Mordal-Manet K. et al., The squale model – A Practice-Based Industrial Quality Model, ICSM, 2009
[5] Wagner S. et al., The quamoco product quality modeling and assessment approach, ICSE, 2012
[6] Siavvas M. G. et al., QATCH - An adaptive framework for software product quality assessment, ESWA Journal, 2017
[7] López, L. et al. Q-Rapids Tool Prototype: Supporting Decision-Makers in Managing Quality in Rapid Software Development. CAiSE Forum. Anais: Lecture Notes in Business Information Processing.Springer, 2018
[8] Choras, M. et al. Measuring and Improving Agile Processes in a Small-size Software Development Company. IEEE Access, v. PP, p. 1, 2020.
[9] Henningsson K. & Wohlin C., Understanding the relations between software quality attributes - a survey approach. In Proceedings of 12th ICSQ, 2002
[10] Aldaajeh S. & Asghar T. & Abed A.K. & Zakaullah M., Communing Different Views on Quality Attributes Relationships' Nature. EJSR, 2012.
[11] Svahnberg M. & Henningsson K., Consolidating different views of quality attribute relationships, WoSQ ICSE, 2009.
[12] Zulzalil, H. et al. A Case Study to Identify Quality Attributes Relationships for Web-based Applications. 2008.
[13] XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[14] Buse R.P.L. & Zimmermann T., Information needs for software development analytics. ICSE, 2012.
[15] Basili, V. R.; Caldiera, G.; Rombach, D. H. The Goal Question Metric Approach. In: Encyclopedia of SWE. [s.l.] John Wiley & Sons, 1994.
[16] Lanza, M.; Marinescu, R. OO Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of OO Systems, 2010.
[17] Dybå T. & Sjøberg D. & Cruzes D., What works for whom, where, when, and why? On the role of context in empirical software engineering, ESEM, 2012.
[18] Lima, P. et al. A Metrics Suite for code annotation assessment. Journal of Systems and Software, v. 137, p. 163–183, 2018.
[19] Fenton N.; Bieman J., Software Metrics-A Rigorous and Practical Approach, 3th, CRC Press, 2014.
[20] Meneely, A.; Smith, B.; Willians, L. Validating Software Metrics: A Spectrum of Philosophies. ACM Trans. Softw. Eng. Methodol., 2013.
[21] Kitchenham B., What's up with software metrics? – A preliminary mapping study, JSS, 2010.
[22] Tahir, A.; Macdonell, S. G. A systematic mapping study on dynamic metrics and software quality. Software Maintenance (ICSM), 2012.
[23] Elberzhager, F.; Munch, J.; Nha, V. T. N. A systematic mapping study on the combination of static and dynamic quality assurance techniques. IST, 2012.
[24] Fitzgerald B. & K.J. Stol., Continuous software engineering: A roadmap and agenda, JSS, 2017.
[25] Lindgren, E.; Munch, J. Software Development as an Experiment System: A Qualitative Survey on the State of the Practice. Lecture Notes in BIP, 2015.
[26] Kevic, K. et al. Characterizing Experimentation in Continuous Deployment: A Case Study on Bing. ICSE-SEIP, 2017
[27] Fabijan A. & Dmitrie P. & Olsson H. & Bosch J., The Evolution of Continuous Experimentation in Software Product Development, ICSE, 2017.
[28] Marinescu, R.; Ratiu, D. Quantifying the quality of object-oriented design: The factor-strategy model, WCRE, 2004.
[29] Lavazza L. & Morasca S., An Empirical Evaluation of Distribution-based Thresholds for Internal Software Measures, PROMISE, 2016.
[30] Lochmann, K. A Benchmarking-Inspired Approach to Determine Threshold Values for Metrics. SIGSOFT Softw. Eng. Notes, v. 37, n. 6, p. 1–8, nov. 2012.
[31] Meirelles, P. R. M. Monitoring Source Code Metrics in Free Software Projects. [s.l.] Institute of Mathematics and Statistics of University of São Paulo, 2013.
[32] Kolda, T.; Bader, B. Tensor Decompositions and Applications. SIAM, 2009.
[33] Papalexakis, E. E.; Faloutsos, C.; Sidiropoulos, N. D. Tensors for Data Mining and Data Fusion: Models, Applications, and Scalable Algorithms. TIST, 2016.
[34] Haoues, M. et al. A guideline for software architecture selection based on ISO 25010 quality-related characteristics. IJSAEM, 2017.
[35] Baeza Y. R.; Ribeiro N., B. Modern information retrieval. ACM press 1999.
[36] Wiegers, K.; BEATTY, J. Software Requirements 3, Microsoft Press, 2013.
[37] Funke, F.; Reips, U. Why Semantic Differentials in Web-Based Research Should Be Made from VAS and Not from 5-Pst. Scales, Field Methods, 2012.