EasyChair Preprint

№ 4590

Supervised Machine Learning Approach for Software Maintainability Assessment

Stephane Nkeuga Ngueliekam and Mathurin Soh

November 18, 2020

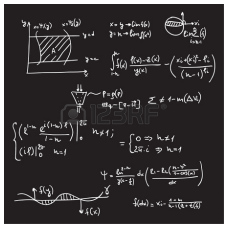
UNIVERSITY OF DSCHANG

Enterprise, Research and Development Forum (EREDEF-2020)

Artificial Intelligence, Digital Economy and African Transformation

Algebra • Analysis • Computer Science

mailto://dept.math-info@univ-dschang.org



SUPERVISED MACHINE LEARNING APPROACH FOR SOFTWARE MAINTAINABILITY ASSESSMENT

Stephane Nkeuga Ngueliekam [1] and Mathurin SOH[1]

[1] *URIFIA, Department of Mathematics and Computer Science, University of Dschang, Cameroon*

[*] Corresponding author : *stephanenkeuga@gamil.com,mathurinsoh@gmail.com*

ABSTRACT: In the highly competitive product market, the importance of quality is no longer an advantage, but a necessary factor for the success of the company. The work presented in this paper focuses on software quality assessment using a supervised machine learning approach. Software quality is a vague and multifaceted concept and varies from person to person. Thus, assessing quality depends on the perspective we have, making direct quality assessment very difficult. In this document we evaluate quality from the developer's point of view. The ISO 9126 standard defines six factors (internal and external) of high level to characterize the quality of a software product. In the following we use the standard to identify the quality requirements. The software being an abstract entity, the essential element for its evaluation is its source code. We use static code extractor to extract the metrics it contains. These metrics are used as inputs to the machine learning system. The machine learning system is used to discover the knowledge that is hidden in the data. This is done by making the best possible approximation to reality. Our work proposes a formula to calculate the quality of the software. To do this we use supervised machine learning algorithms to optimize the quality formula. Our quality formula thus offers developers an objective and independent view of the concept of quality on the one hand, but also to be able to build good quality products on the other hand.

KEYWORDS: software quality, quality metrics, quality factor, software product, supervised machine learning

1. Introduction

For almost half a century now, computer systems, and more particularly software and computer programs, have been part of the daily life of mankind. These softwares which are more and more omnipresent in our daily lives, whatever the sector of activity we are in, finance, education, health, army to name a few. This very growing presence has taken the notion of software quality from a need for comfort to a critical need. Indeed, any failure of a computer system is potentially a source of inconvenience that can, in certain borderline cases, cause serious damage to the economic integrity of organizations or, worse, to the physical integrity of individuals. For example, on Sunday, March 10, 2019, the crash of a Boeing 737 Max belonging to Ethiopian Airlines killed a total of 157 people[1]. The crash

is caused by a new software system called the Maneuvering Characteristics Augmentation System (MCAS), which the Boeing company has developed to solve the stability problems in certain flight conditions induced by the aircraft's new, more powerful engines. The crash of the Ariane 5 shuttle in 1996, which crashed 40 seconds after liftoff due to a computer bug in the autopilot system. A bug which caused a loss of 370 million dollars [2]. Or the online sales site "Ebay" unavailable for 22 hours in 1999, which resulted in a loss for the group estimated between 3 and 5 million dollars [2]. Thus all these cases of software failures mentioned above show the importance and urgency of evaluating the quality of the software before, during and after the development phase.

2. State of the art

Software quality is still a vague and multifaceted concept, which means different things for different people. In general, the way we measure the quality depends on the point of view we adopt [3]. This makes the direct evaluation of very difficult quality. In order to better quantify quality, researchers have developed Indirect models that attempt to measure the quality of software products using a set of attributes, characteristics and quality measures. An important assumption in the definition of these quality models is that the internal characteristics of the product (internal quality indicators) influence the external attributes of the product (quality of the product). of use), and by evaluating the internal characteristics of a product, one can derive reasonable conclusions on the external quality attributes of the product[3]. This approach based on on the product is frequently adopted by advocates of software metrology because that it offers an objective and independent view of the quality context. One of the earliest models of software product quality was suggested by McCall[4]. McCall's quality model defines software product quality as a hierarchy of factors, criteria and measures and was the first of several models of the same form. Another quality model was introduced by Boehm[5]. It is also an important predecessor of the current quality models. Boehm[5] takes into account the contemporary shortcomings of models that automatically and quantitatively evaluate software quality. Basically, his model attempts to qualitatively define software quality through a given set of attributes and measures. There are some recognizable parallels between McCall's model [4] and Boehm's model [5]. For example, both propose a structured hierarchical model with high-level, intermediate-level, and low-level features. International efforts have also led to the development of a standard for measuring the quality of software products, ISO 9126, which has its origins in the McCall[4] and Boehm[5] models. It is a generic model that allows users to develop their own software products. own criteria. This model does not reach the level of metrics. It leaves it up to users to choice of metrics to be implemented in their model. In this work, Dromeydromey underlines that software does not directly manifest high quality attributes. Software only has product characteristics that influence quality attributes. Poor product characteristics reduce its quality attributes. The dromeydromey model provides a bottom-up methodology for the development of quality models. All of these models vary in their hierarchical definition of quality, but they share a

common difficulty. The models are vague in their definition of the details and lower-level measures needed to obtain a quantitative assessment of product quality. This lack of specificity in these models provides little guidance to software developers who need to build quality products. Another difficulty with previous models was the inability to account for the dependency between quality attributes. If several high-level attributes are used to refer to product quality, in general, only a subset of these attributes would be relevant for each different viewpoint. In his work, Bansiya [3] successfully made an association between metrics and high-level quality attributes. Bansiya[3] redefined the ISO 9126 quality factors for the object-oriented paradigm. According to Bansiya[3], since quality is calculated by an aggregate of criteria, the importance of all criteria for quality may not be equal, therefore, the influence of a quality criterion can be modified by a weighting factor.
However, previous models have provided an excellent framework from which to build on. possible to proceed. New measures, relationships and weights can be evaluated and defined in the context of these earlier models. Artificial intelligence is defined as a science whose goal is to do, by a machine, what a human being achieves using his or her intelligence[6]. Machine learning is an application of artificial intelligence (AI) that gives systems the ability to learn and improve automatically from experience without being explicitly programmed to do so. It is applied in several fields such as medicine, robotics and also software engineering. In software engineering machine learning can be used to predict the quality of a software from the early stages of its development cycle. Machine learning techniques have been used in many different problem areas, as this field focuses on building algorithms that have the ability to improve their performance automatically through experience. The application of machine learning techniques to software engineering. Pierre Oum Sack[7] has proposed a quality assessment model based on a machine learning and model transformation approach. This model allows to predict software quality using machine learning and graph theory.
Salma Hamza [8] has proposed in her thesis a pragmatic approach to measure the quality of software applications based on software components. This model allows to predict software quality from metrics in the component domain such as component-level metrics, application-level metrics and interface-level metrics.

3. Methodology

Kitchenham cites a more detailed list of quality aspects in this work : [9]. This list composed by Garvin [10] summarizes these aspects in points of view. It defines the user's viewpoint, which sees quality as relevant to the user's purpose; the manufacturer's viewpoint, which sees quality as conforming to specifications; the product's viewpoint, which sees quality as related to the inherent characteristics of the product; and the value-based viewpoint, which sees quality as dependent on what a customer is willing to pay for it. Our work focuses on the evaluation of software quality using source code measures and thus has a product quality view. How can an abstract quality concept such as maintainability or reliability be objectively evaluated? To answer this question we proceed in 5 steps:

- Definition of quality requirements in terms of quality factors (maintainability, functional capability) according to an ISO 9126 quality model.
- Association of quality metrics, and calculation of formulas for factor values and quality criteria. Definition of the satisfaction rates for the quality factor. The factor is accepted or not according to a threshold value.
- Acquisition, pre-processing and construction of the learning and test data set.
- Construction of the quality prediction model. It is done in the Weka environment.
- Evaluation of the performance of the quality prediction model.

The first step is to define the quality requirements in terms of quality factors. Factors such as maintainability represent high level attributes in quality models such as ISO 9126[11]. These attributes are not directly measurable and need to be broken down into criteria. The decomposition process is applied until the attribute entities are obtained and are measurable. These attributes can be simple or derived. The ISO 9126[11] defines six quality factors grouped into internal and external quality factors, i.e:

- internal factors: functional capacity, maintainability;
- external factors: ease of use, portability, efficiency and reliability

However, each characteristic is relevant to each different point of view. In what follows, we evaluate quality from the software developer's point of view, this means that we evaluate the internal factors of the product and more precisely the maintainability of the software. Thus the factor is broken down into criteria and attributes as shown in the following table:

Criteria	Attributes	Measured properties
Modifiability	Modularity	encapsulation, cohesion, coupling
	Documentation	Comment
	Localization of changes	Cohesion, abstraction, polymorphism
Testability	Structure	complexity, abstraction, inheritance
	Concision	Complexity, cohesion, coupling
Stability	Code structure	complexity, abstraction, code size
Analysability	Structure	Complexity
	Coupling	coupling

Table 1: Maintainability Criteria and Attributes [7]

In the previous table we have refined the maintainability factor into criteria and then each criterion into a measurable entity through code metrics, thus moving the metric to a central position in the software quality assessment process. In the quality model instanciated by Bansiya [3], we distinguish two main types of metrics: simple metrics and derived metrics. He succeeds in assigning design metrics to design properties. It combines the metrics in a meaningful way. Bansiya[3] uses weighted design properties to construct a quality attribute. It sets up several formulas so the weightings and the combination of the properties are defined by Bansiya[3]. After decomposing the maintainability factor into criteria and the criteria into measurable attributes. We can deduce the following formula for maintainability:

Maintenability=$\alpha * Analysability + \beta * Changeability + \lambda * Stability + \Theta * Testability$
Where the weights $\alpha, \beta, \lambda, \Theta$ are obtained through the weighting process used by Bansiya[3]. Thus in this work we will use the following formulas for the different criteria of maintainability:
Analysability = $0.5 * WMC + 0.5 * CBO$
Changeability = $0.25 * LCOM + 0.25 * DIT + 0.25 * RFC + 0.25 * CBO$
Stability = $0.5 * WMC + 0.5 * LOC$
Testability = $0.25 * DIT + 0.25 * CBO + 0.25 * LCOM + 0.25 * LOC$

These different criteria are based on a set of metrics. The metrics are chosen according to the attributes and measured properties described in [7] and [12, 13, 14].

4. Results and experimentation

In this section we present the implementation of the quality assessment model we have developed. To do so, we use the following tools and environments:

- The CK (Chidamber-Kemerer metrics) tool for the extraction of code metrics[15]
- Weka for the machine learning system. Machine learning will be used to extract knowledge. This knowledge will allow the optimization of the quality model by more precisely quantifying the dependencies between the quality criteria and a set of metrics.[16].

For the conduct of our experiment, we use data sets from four software projects namely Vuze (Azureus Java BitTorrent client) v5.2.0.0 [18], apache-ant-1.10.7-src[19], glassfish-5. 0.1[17] and the KC1 repository entitled "Software Defect Prediction" which is one of the defect data sets of the NASA Metric Data Program (MDP)[20]. We perform the experiment with the following algorithms:

- the naive bayes classifier
- decision trees: J48, RandomTree
- linear regression

- the M5P algorithm

Data set with percentage (80,20)

Software project	Correctly classified instances	Misclassified instances
KC1	89.6552%	10.3448%
APACHE ANT	96.2766%	3.7234%
AZEURUS	98.6749%	1.3251%
GLASSFISH	94.9836 %	5.0164%

Table 2: Results of the classifications obtained by the NaiveBayes Classifier on 4 software projects

Data set with percentage (80,20)

Software project	Correctly classified instances	Misclassified instances
KC1	100%	0%
APACHE ANT	99.2021%	0.7979%
AZEURUS	99.6466 %	0.3534%
GLASSFISH	100 %	0%

Table 3: Classification results obtained by the RandomTree decision tree on 4 software projects

Data set with percentage (80,20)

Software project	linear equation	correlation coefficient
KC1	MAINTENABILITY = 0.0506 * CBO + 0.0506 * DIT + 0.0506 * LOCM + 0.0337 * RFC + 0.0833 * WMC + 0.1667 * STABILITY + 0.1152 * CHANGEABILITY + 0.1825 * TESTABILITY + 0.1667 * ANALYSABILITY +	1
APACHE ANT	maintenabilite = 0.0887 * cbo + 0.1037 * wmc + 0.047 * dit + 0.0203 * rfc + 0.047 * lcom + 0.0886 * loc + 0.1666 * analysability + 0.126 * stability + 0.1435 * testability + 0.1687 * changeability	1
AZEURUS	maintenability = 0.087 * cbo + 0.103 * wmc + 0.045 * dit + 0.02 * rfc + 0.045 * lcom + 0.085 * loc + 0.165 * analysability + 0.129 * stability + 0.168 * changeability + 0.153 * testability	1

GLASSFISH	maintenability = 0.1048 * cbo + 0.1407 * wmc + 0.0275 * dit + 0.0027 * rfc + 0.0275 * lcom + 0.0882 * loc + 0.0955 * analysability + 0.1232 * stability + 0.2392 * changeability + 0.1508 * testability	1

Table 4: Results of the classifications obtained by the linear regression algorithm on 4 software projects

From the prediction models obtained, we notice that decision trees are more efficient on all four projects.
We therefore present the decision trees obtained by the RandomTree algorithm respectively on:

About the KC1 project

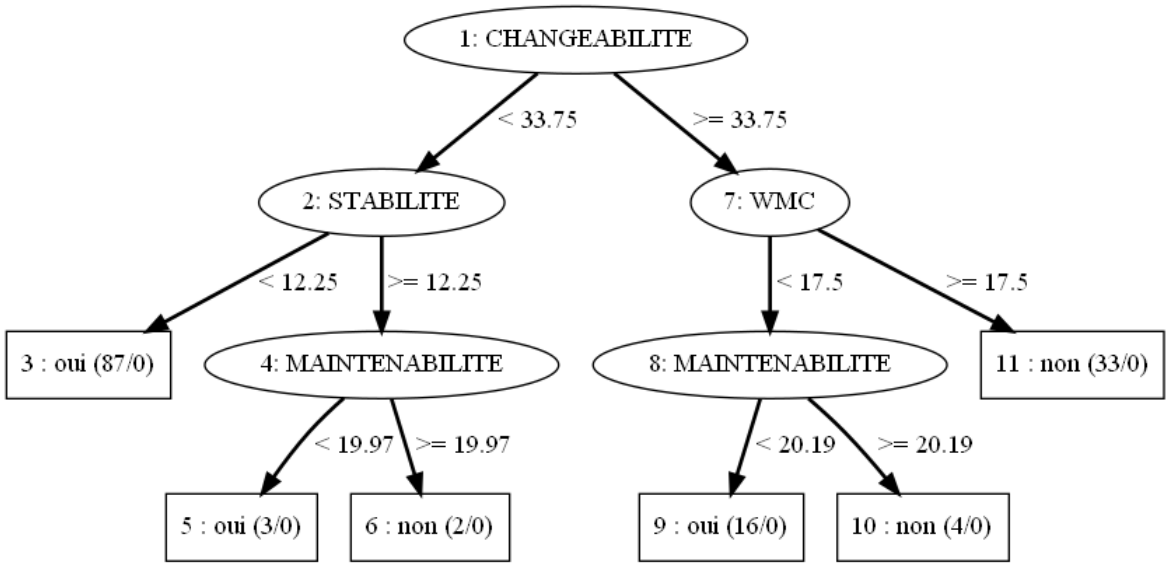


Figure 1: Decision tree obtained on the KC1 project

By examining the tree produced by the KC1 project, we discover the following rules about maintainability:

1. $CHANGEABILITY < 33.75 \; and \; STABILITY < 12.25$
2. $CHANGEABILITY < 33.75 \; and \; STABILITY >= 12.25 \; and \; MAINTENABILITY < 19.97$
3. $CHANGEABILITY >= 33.75 \; and \; WMC < 17.5 \; and \; MAINTENABILITY < 20.19$

Taking these rules into account, maintainability depends on the WMC metrics and the changeability and stability criteria. By detailing these criteria, we deduce that maintainability ultimately depends on the metrics *LCOM, DIT, RFC, CBO, WMC, LOC*.
Moreover, considering the M5P algorithm, we deduce the following maintainability formula: **MAINTENABILITY = 0.0506 * CBO + 0.0506 * DIT + 0.0506 * LOCM + 0.0337 * RFC + 0.25 * WMC + 0.1152 * CHANGEABILITY + 0.1825 * TESTABILITY** In this formula, maintainability depends on the metrics CBO, DIT, LCOM, RFC, WMC and the changeability and testability criteria. By developing this formula we have the same metrics as those discovered by the rules from the decision trees.

5. Conclusion

In this article we presented our software quality assessment approach based on a supervised machine learning approach. This learning process allowed us to discover the knowledge between the data. This knowledge allowed us to give a better approximation of the maintainability formula. This work offers developers an objective and independent vision of the concept of quality, but also allows them to build good quality products. It would be interesting to extend the work in this field in order to find a quality model that would allow the evaluation of all the quality factors of a given software. This could allow an overall appreciation of software quality by the various actors involved in the process of designing, developing, using and evolving software.

References

[1] "Le cauchemar du 737 MAX ne cesse de s'aggraver, Un rapport accablant des enquêteurs de la Chambre US montre la pire défaillance de sécurité dans l'avion cloué au sol à cause des problèmes logiciels", "Stan Adkens", = "https://embarque.developpez.com/actu/296371/Le-cauchemar-du-737-MAX-ne-cesse-de-s-aggraver-un-rapport-accablant-des-enqueteurs-de-la-Chambre-US-montre-la-pire-defaillance-de-securite-dans-l-avion-cloue-au-sol-a-cause-des-problemes-logiciels/", "2020 (consulté le 11, mars 2020)".

[2] Analyse et conception d'un modèle de qualité logicielle, Mordal, Karine, 2012.

[3] A hierarchical model for object-oriented design quality assessment, Bansiya, Jagdish and Davis, Carl G., IEEE Transactions on software engineering, 28, 1, 4–17, 2002, IEEE.

[4] Factors in software quality, volume I, McCall, JA and Richards, PG and Walters, GF, NTIS Springfield, 1977.

[5] Software development cost estimation approaches?A survey, Boehm, Barry and Abts, Chris and Chulani, Sunita, Annals of software engineering, 10, 1-4, 177–205, 2000, Springer.

[6] Intelligence artificielle vulgarisée, Aurélien Vannieuwenhuyze, Editions ENI, 978-2409020735, 2019, , , , http://gen.lib.rus.ec/book/index.php?md5=26f43e35c6fdb481829140c07fb85925.

[7] Contribution à l?étude de la qualité du logiciel, Sack, Pierre Marie Oum Oum, 2009, Université du Littoral Côte d'Opale.

[8] A pragmatic approach to measure the quality of Component–Based Software Applications, Hamza, Salma, https://tel.archives-ouvertes.fr/tel-01256822, 2014LORIS356, Université de Bretagne Sud, 2014, Dec, Software component ; Quality metrics ; Quality model ; Predictive model ; Composant logiciel ; Métriques de qualité ; Modèle de qualité ; Modèle prédictif, Theses, https://tel.archives-ouvertes.fr/tel-01256822/file/2014TheseHamzaS.pdf, tel-01256822, v1.

[9] Software quality: the elusive target [special issues section], Kitchenham, Barbara and Pfleeger, Shari Lawrence, IEEE software, 13, 1, 12–21, 1996, Ieee.

[10] Assessing software quality attributes with source code metrics, Jetter, Andreas and Gall, Harald and Pinzger, Martin and Knab, Patrick, 2006, Citeseer.

[11] "L'état de l'art : l'ingénierie des besoins", "Yves Constantinidis", "https://yves-constantinidis.com/doc/yves-758.htm", "2013 (consulté le 3, fevrier 2020)".

[12] A mapping study on design-time quality attributes and metrics, Arvanitou, Elvira Maria and Ampatzoglou, Apostolos and Chatzigeorgiou, Alexander and Galster, Matthias and Avgeriou, Paris, Journal of Systems and Software, 127, 52–77, 2017, Elsevier.

[13] Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review, Jabangwe, Ronald and Börstler, Jürgen and Šmite, Darja and Wohlin, Claes, Empirical Software Engineering, 20, 3, 640–693, 2015, Springer.

[14] A metrics suite for object oriented design, Chidamber, Shyam R and Kemerer, Chris F, IEEE Transactions on software engineering, 20, 6, 476–493, 1994, IEEE.

[15] Java code metrics calculator (CK), Maurício Aniche, 2015, consulté 21 juin 2020, Available in https://github.com/mauricioaniche/ck/.

[16] Leatn Weka : Simple easy learning, tutoriallspoint.com, 2020, consulté 29 juin 2020, https://www.tutorialspoint.com/weka/index.htm.

[17] "GlassFish The Open Source Java EE Reference Implementation", "", "https://javaee.github.io/glassfish/download", "2017 (consulté le 15,mai 2020)".

[18] "Vuze (Azureus Java BitTorrent client) for Mac OS X v5.2.0.0", "www.afterdawn.com", "https://www.afterdawn.com/software/source-codes/index.cfm?54", 2013, (consulté le 14 ,juin 2020).

[19] "Index of /ant/source", "Apache Fondation", "https://downloads.apache.org/ant/source/", "2020 (consulté le 18 ,mai 2020)".

[20] "PROMISE Software Engineering Repository: Public Datasets", "A. Gunes Koru", "http://promise.site.uottawa.ca/SERepository/datasets-page.html", "2005 (consulté le 02, fevrier 2020)".