Towards Meaningful Software Metrics Aggregation

Maria Ulan, Welf Löwe, Morgan Ericsson, Anna Wingkvist
Department of Computer Science and Media Technology
Linnaeus University, Växjö, Sweden
{maria.ulan | welf.lowe | morgan.ericsson | anna.wingkvist}@lnu.se

Abstract

Aggregation of software metrics is a challenging task, it is even more complex when it comes to considering weights to indicate the relative importance of software metrics. These weights are mostly determined manually, it results in subjective quality models, which are hard to interpret. To address this challenge, we propose an automated aggregation approach based on the joint distribution of software metrics. To evaluate the effectiveness of our approach, we conduct an empirical study on maintainability assessment for around 5 000 classes from open source software systems written in Java and compare our approach with a classical weighted linear combination approach in the context of maintainability scoring and anomaly detection. The results show that approaches assign similar scores, while our approach is more interpretable, sensitive, and actionable.

Index terms— Software metrics, Aggregation, Weights, Copula

1 Introduction

Quality models provide a basic understanding of what data to collect and what software metrics to use. However, they do not provide how software (sub-)characteristics should be quantified, and metrics should be aggregated.

The problem of metrics aggregation is addressed by the research community. Metrics are often defined at a method or class level, but quality assessment sometimes requires insights at the system level. One bad metric value can be evened out by other good metric values when summing them up or computing their *mean* [1]. Some effort has been directed into metrics aggregation based on inequality indices [2,3], and based on *thresholds* [4–8] to map source code level measurements to software system rating.

In this research, we do not consider aggregation along the structure of software artifacts, e.g., from classes to the system. We focus on another type of metrics aggregation, from low-level to higher-level quality properties, Mordal-Manet et al. call such type of aggregation *metrics composition* [9].

Different software quality models that use weighted metrics aggregation have been proposed, such as *QMOOD* [10], *QUAMOCO* [11], *SIG* [12], *SQALE* [13], and *SQUALE* [14]. The weights in these models are defined based on experts' opinions or surveys. It is questionable whether manual weighting and combination of the values with an arbitrary (not necessarily linear) function are acceptable operations for metrics of different scales and distributions.

As a countermeasure, we propose to use a probabilistic approach for metrics aggregation. In previous research, we considered software metrics to be equally important and developed a software metrics visualization tool. This tool allowed the user to define and manipulate quality models to reason about where quality problems were located, to detect patterns, correlations, and anomalies [15].

Here, we define metrics scores by probability as *complementary Cumulative Distribution Function* and link them with joint probability by the so-called *copula* function. We determine weights from the joint distribution and aggregate software metrics by weighted product of the scores. We formalize quality models to expresses quality as the probability of observing a software artifact with equal or better quality. This

In: D. Di Nucci, C. De Roover (eds.): Proceedings of the 18th Belgium-Netherlands Software Evolution Workshop, Brussels, Belgium, 28-11-2019, published at http://ceur-ws.org

approach is objective since it relies solely on data. It allows to modify quality models on the fly, and it creates a realistic scale since the distribution represents quality scores for a set of software artifacts.

2 Approach Overview

We consider a joint distribution of software metrics values, and for each software artifact, we assign a probabilistic score. W.l.o.g, we assume that all software metrics are defined such that larger values indicate lower quality. The joint distribution of software metrics provides the means of objective comparison of software artifacts in terms of their quality scores, which represent the relative rank of the software artifact within the set of all software artifacts observed so far, i.e., how good or bad a quality score compare to other quality scores.

Let $A = \{a_1, \ldots, a_k\}$ be a set of k software artifacts, and $M = \{m_1, \ldots, m_n\}$ be a set of n software metrics. Each software artifact is assessed by metrics from M, and the result of this assessment is represented as $k \times n$ *performance matrix* of metrics values.

We denote by $e_j(a_i)$ for $\forall i \in \{1, k\}, \forall j \in \{1, n\}$ an (i, j)-entry, which shows the degree of performance for an software artifact a_i measured for metric m_j. We denote by $E_j = [e_j(a_1), \ldots, e_j(a_k)]^T \in \mathcal{E}_j^k$ the j-th column of *performance matrix*, which represents metrics values for all software artifacts with respect to metric m_j where $\mathcal{E}_j$ is the domain of these values.

For each software artifact $a_i \in A$ and metric $m_j \in M$, we define a score $s_j(a_i)$, which indicates the degree to which this software artifact meets the requirements for the metric. Formally, for each metric m_j we define a score function s_j:

$$\begin{aligned} e_j(a) &: A \mapsto \mathcal{E}_j \\ s_j(e) &: \mathcal{E}_j \mapsto [0, 1] \end{aligned} \tag{1}$$

Based on the score functions s_j for each metric, our goal is to define an overall score function such that, for any software artifact, it indicates the degree to which this software artifact satisfies all metrics. Formally, we are looking for a function:

$$F(s_1, \ldots, s_n) : [0, 1]^n \mapsto [0, 1] \tag{2}$$

Such an aggregation function takes an n-tuple of metrics scores and returns a single overall score. We require the following properties:

1. If a software artifact does not meet the requirements for one of the metrics, the overall score should be close to zero.

$$F(s_1, \ldots, s_n) \to 0 \text{ as } s_j \to 0 \tag{3}$$

2. If all scores of one software artifact are greater or equal than all scores of another software artifact, the same should be true for the overall scores.

$$\begin{gathered} s_1^i \geq s_1^l \wedge \cdots \wedge s_n^i \geq s_n^l \Rightarrow \\ F(s_1^i, \ldots, s_n^i) \geq F(s_1^l, \ldots, s_n^l), \\ \text{where } s_j^i = s_j(e_j(a_i)), s_j^l = s_j(e_j(a_l)) \end{gathered} \tag{4}$$

3. If the software artifact perfectly meets all but one metric, the overall score is equal to that metrics score.

$$F(1, \ldots, 1, s_j, 1, \ldots, 1) = s_j \tag{5}$$

We propose to express the degree of satisfaction with respect to a metric using probability. We define the score function of Equation (1) as follows:

$$s_j(e_j(a)) = Pr(E_j > e_j(a)) = CCDF_{e_j}(a) \tag{6}$$

We calculate the *Complementary Cumulative Distribution Function (CCDF)*. This score represents the probability of finding another software artifact with an evaluation value greater than the given value. For a multi-criteria case, we can specify a joint distribution in terms of n marginal distributions and a so-called *copula* function [16]:

$$\begin{gathered} Cop(CCDF_{e_1}(a), \ldots, CCDF_{e_n}(a)) = \\ Pr(E_1 > e_1(a), \ldots, E_n > e_n(a)) \end{gathered} \tag{7}$$

The *copula* representation of a joint probability distribution allows us to model both marginal distributions and dependencies. The *copula* function *Cop* satisfies the signature (2) and fulfills the required properties (3), (4), and (5).

We consider a *weight vector*, where each w_i represents the relative importance of metric m_i compared to the others:

$$w = [w_1, \ldots, w_n]^T, \text{ where } \sum_{i=1}^{n} w_i = 1 \tag{8}$$

We compute weights using a non-linear exponential regression model for a sample of software artifacts mapping metrics scores of Equation(6) to copula value of Equation(7). Note that these weights regard dependencies between software metrics. Finally, we define software metrics aggregation as a weighted composition of metrics score functions:

$$F(s_1, \ldots, s_n) = \prod_{j=1}^{n} s_j^{w_j} \tag{9}$$

We consider a software artifact a_l to be better than or equally good as another software artifact a_i, if the

total score according to Equation (2) of a_l is greater than or equal the total score of a_i:

$$a_l \succeq a_i \Leftrightarrow F(a_l) \geq F(a_i) \tag{10}$$

Aggregation is defined as a composition of the product, exponential, and *CCDF* functions, which are monotonic functions. Hence, the score which is obtained by aggregation allows to rank set A of software artifacts with respect to metrics set M:

$$Rank(a_l) \leq Rank(a_i) \Leftrightarrow F(a_l) \geq F(a_i) \tag{11}$$

From a practical point of view, probabilities can be calculated empirically, and each score can be obtained as a ratio of the number of software artifacts with lower than a given metric value to the number $|A|$ of software artifacts.

The proposed aggregation approach makes it possible to express the score for a software artifact as the probability to observe something with equal or worse metrics values, based on all software artifacts observed. Once the quality scores are computed, the software artifacts can trivially be ranked by the score by simply ordering the values from smallest to largest. We assign the same rank for software artifacts in case their total scores are equal. Low (high) ranks correspond to high (low) probabilities. This interpretation is the same on all levels of aggregation, from metrics scores to the total quality scores.

3 Preliminary Evaluation

We apply our approach to assess *Maintainability* and compare the results with the aggregation approach based on a weighted linear combination of software metrics. We measure the difference between rankings obtained by these approaches and study the agreement between aggregated scores. Finally, we compare approaches by means of sensitivity, and the ability to detect extreme values and Pareto optimal solutions.

In the following subsections, we investigate Java classes and their quality assessment using two research questions:

RQ1 How effective is our approach for a quality assessment?

RQ2 How actionable is our approach by means of sensitivity and anomaly detection?

3.1 Quality Model Description

We consider a quality model for maintainability assessment of classes, which relies on well-known software metrics from *Chidamber & Kemerer* [17] software metrics suit:

CBO, Coupling Between Objects

DIT, Depth of Inheritance Tree

LCOM, Lack of Cohesion in Methods

NOC, Number Of Children

RFC, Response For a Class

WMC, Weighted Method Count
(using Cyclomatic Complexity as method weight)

3.2 Data Set Description

We chose to investigate three open-source software systems. The systems were chosen by such criteria: (i) they are written in Java, (ii) available in GitHub, (iii) they were forked at least once, (iv) they are sufficiently large (several tens of thousands of lines of code and several hundreds of classes), and (v) they have been under active development for several years. The projects we selected are three well-known and frequently used systems: *JabRef*[1], *JUnit*[2], and *RxJava*.[3] Table 1 shows descriptive statistics for these systems.

Table 1: Descriptive statistics of investigated systems

	JabRef	**JUnit**	**RxJava**
Number of classes (NOC)	1 532	1 119	2 744
Lines of code (LOC)	136 039	44 082	378 987
Version	4.3.1	5.3.2	3.0.0

3.3 Measures

The result of the aggregation is a maintainability score, and a ranked list of software artifacts according to their maintainability score. To evaluate our approach, we compare it to a well-known approach considering the following measures:

Correlation We study the *Spearman's correlation* [18] between maintainability scores to assess the ordering, relative spacing, and possible functional dependency.

Ranking distance We measure a distance between the two rankings based on the *Kendall tau distance*, which counts the number of pairwise disagreements between two lists [19].

[1] JabRef, Graphical Java application for managing BibTeX and biblatex databases, `https://github.com/JabRef/jabref`

[2] JUnit, A framework to write repeatable tests for the Java programming language, `https://github.com/junit-team/junit5`

[3] RxJava, Reactive Extensions for the JVM – a library for composing asynchronous and event-based programs using observable sequences for the Java VM, `https://github.com/ReactiveX/RxJava`

Agreement We measure agreement between maintainability scores using *Bland-Altman* statistics [20].

To evaluate if the aggregated scores can be used to detect extreme values and Pareto optimal solutions, we consider the following measures:

Sensitivity We study a variety of values to understand a percentage of software artifacts that have the same maintainability score. The *overall sensitivity* is the ratio of unique scores and the number of software artifacts.

Anomaly detection We compare approaches in terms of their ability to detect anomalies (extreme values and Pareto optimal solutions) using a ratio of the number of detected anomalies and the total number of anomalies in a sample data set.

3.4 Preliminary Results and Analysis

We implemented all algorithms and statistical analyses in R[4]. The metrics data for analysis was collected with *VizzMaintenance*.[5] We collected the metrics values for classes of *JabRef*, *JUnit*, and *RxJava* software systems (5 317 classes in total). We considered their packages structure to group classes and applied *Kolmogorov-Smirnov* statistical test [21] to select a subset for further statistical analysis, which was composed of 5 101 classes. Moreover, we consider the quality assessment of each system separately to study potential differences between software systems. We apply our aggregation approach (See Equation (12)) and compare the results with a weighted linear sum of metrics (see Equation(13)), which we normalized by the *min-max* transformation.

$$s_{CBO}^{w_1} \times s_{DIT}^{w_2} \times s_{LCOM}^{w_3} \times s_{NOC}^{w_4} \times s_{RFC}^{w_5} \times s_{WMC}^{w_6} \quad (12)$$

$$w_1 \times CBO + w_2 \times DIT + w_3 \times LCOM + w_4 \times NOC + w_5 \times RFC + w_6 \times WMC \quad (13)$$

RQ1-effectiveness

We compare approaches within a single software system and the merged data set. First, we study a correlation between aggregation results. Second, we rank software classes based on maintainability scores obtained by two approaches. Table 2 shows *Kendall Tau* distance and *Spearman's rho* correlation. We observe a strong correlation between maintainability scores and low distance between rankings.

[4]The R Project for Statistical Computing, `https://www.r-project.org`

[5]VizzMaintenance, Eclipse plug-in, `http://www.arisa.se/products.php`

Table 2: Agreement between approaches

	Correlation (Spearman)	**Distance (Kendall)**
JabRef	0.93397	0.04829
jUnit	0.98899	0.02483
RxJava	0.96978	0.03083
Merged	0.98953	0.03382

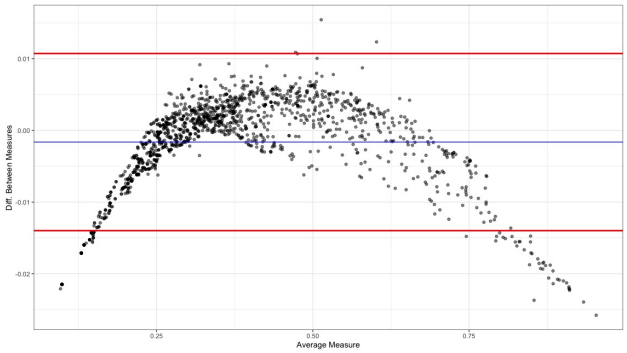

Figure 1: Bland-Altman plot for JabRef

Third, we study an agreement, in the Bland-Altman plot each class is represented by a point with the average of the maintainability scores obtained by two approaches as the x-value and the difference between these two scores as the y-value. The blue line represents the mean difference between scores and the red lines the 95% confidence interval ($mean \pm 1.96SD$). We can observe that plots for *JabRef* and *RxJava* have a similar shape (cf. Figure 1, Figure 3) compare to *jUnit* (cf. Figure 2). We can observe a similar shape for merged data set (cf. Figure 4), since in total *JabRef* and *RxJava* have almost four times more classes than *jUnit*. We can observe that in all plots measurements are mostly concentrated near the blue line and only a few of them are outside of the red lines. The difference for *jUnit* is slightly smaller than for *JabRef* and *RxJava*. In sum, we conclude that the approaches agree, i.e., aggregation results do not differ statistically, and may be used interchangeably for the ranking of software classes.

RQ2-actionability

First, we study the variety of values for each metric and number of extreme values, which we define by means of outliers. We detected 19 extreme values in total. In Table 3 we can observe that metrics have quite low sensitivity, for each metric 40 values on average are unique.

We consider a multi-objective optimization problem based on metrics, and we detect five possible Pareto optimal solutions, i.e., none of the metrics values can be improved without degrading some of the other metrics values. Second, we study the sensitivity and abil-

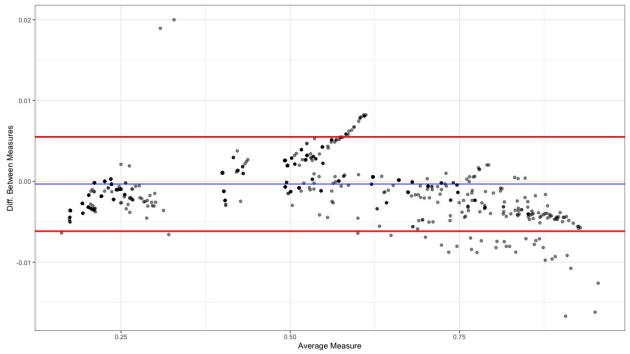

Figure 2: Bland-Altman plot for jUnit

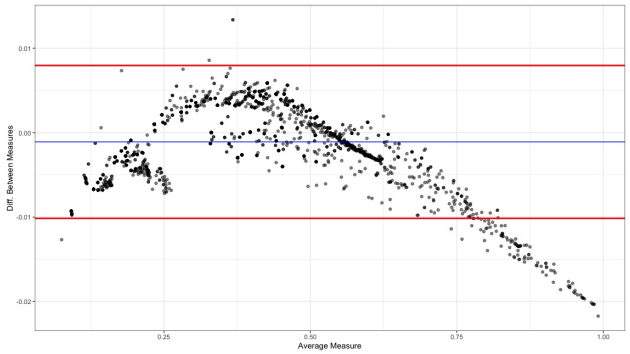

Figure 3: Bland-Altman plot for RxJava

ity to detect anomalies (extreme values and Pareto optimal solutions) for both approaches. In Table 4 we can observe that our approach is more sensitive and more suitable for anomaly detection.

Table 3: Metrics variety of values

	Number of Extreme Values	Sensitivity
CBO	3	0.00768
DIT	7	0.00109
LCOM	3	0.05746
NOC	2	0.00365
RFC	2	0.01848
WMC	2	0.02086

3.5 Discussion

We define metric scores by means of probability, as it provides a simple interpretation for a quality score by means of the joint distribution. In contrast, quality scores obtained by a weighted linear combination of metrics do not provide clear interpretation, especially when metrics are incomparable. We assume that larger metrics values indicate worse quality, however both too small and too large values can be problematic for some of the software metrics. Note that it is not a limitation since we could transform metrics to have this property. We extracted weights from joint distribution, which we consider as a ground truth. This might be a threat to internal validity. We compare our approach with a weighted linear combination of metrics, it might be a treat as well since we do not compare it with other approaches. In this preliminary evaluation, we consider six metrics, three software systems written in Java, and focus on maintainability. This might be a threat to external validity.

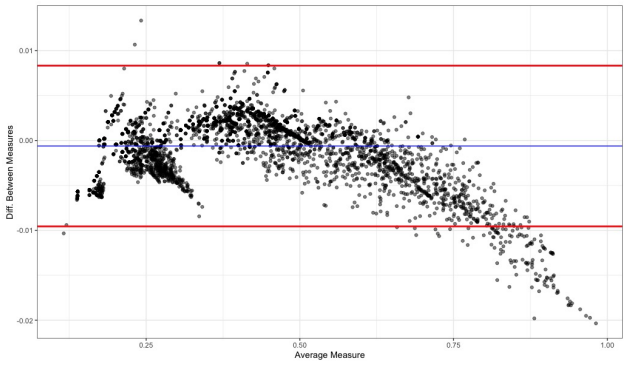

Figure 4: Bland-Altman plot for Merged data

Table 4: Comparison of approaches by actionability

	Aggregation (Eq.12)	Aggregation (Eq.13)
Sensitivity	0.41317	0.31656
Extreme values	0.94736	0.63158
Pareto optimal solutions	1	0.6

4 Conclusion and Future Work

In conclusion, we defined an automated aggregation approach for software quality assessment. We defined probabilistic scores based on software metrics distributions and aggregate them using the weighted product, we obtained the weights from joint distribution. To evaluate the effectiveness and actionability of our approach, we conducted an empirical study for maintainability assessment. We collected *CBO, DIT, LCOM, NOC, RFC,* and *WMC* metrics from *Chidamber & Kemerer* metrics suit for classes of *JabRef*, *JUnit*, and *RxJava* software systems, and compared our approach with a weighted linear combination of metrics. The results showed that the approaches agree and can be used interchangeably for ranking software artifacts. However, our approach is more effective and actionable, i.e., it has clear interpretation, higher sensitivity, and is better at detecting extreme values and Pareto optimal solutions.

Our approach is mathematically well-defined since generalization is not questionable, and can be theoretically validated. For example, we can conduct simulation experiments to study the deviation between our and other approaches depending on the number of classes, number of metrics, levels of aggregation, etc.

However, there is still a need for empirical validation of our approach. In the future, we plan to evaluate our approach on other data sets, such as *The GitHub Java corpus*, which contains around 15 000 software systems [22]. We also plan to compare our approach with Bakota et al. probabilistic approach [23].

Acknowledgments

We thank the anonymous reviewers whose comments and suggestions helped us improve and clarify the research onto paper.

References

[1] Bogdan Vasilescu, Alexander Serebrenik, and Mark Van den Brand. By no means: A study on aggregating software metrics. In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*, pages 23–26. ACM, 2011.

[2] Rajesh Vasa, Markus Lumpe, Philip Branch, and Oscar Nierstrasz. Comparative analysis of evolving software systems using the gini coefficient. In *2009 IEEE International Conference on Software Maintenance*, pages 179–188. IEEE, 2009.

[3] Alexander Serebrenik and Mark van den Brand. Theil index for aggregation of software metrics values. In *2010 IEEE International Conference on Software Maintenance*, pages 1–9. IEEE, 2010.

[4] Ilja Heitlager, Tobias Kuipers, and Joost Visser. A practical model for measuring maintainability. In *null*, pages 30–39. IEEE, 2007.

[5] José Pedro Correia and Joost Visser. Certification of technical quality of software products. In *Proc. of the Int'l Workshop on Foundations and Techniques for Open Source Software Certification*, pages 35–51, 2008.

[6] Tiago L Alves, José Pedro Correia, and Joost Visser. Benchmark-based aggregation of metrics to ratings. In *2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, pages 20–29. IEEE, 2011.

[7] Paloma Oliveira, Fernando P Lima, Marco Tulio Valente, and Alexander Serebrenik. Rttool: A tool for extracting relative thresholds for source code metrics. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 629–632. IEEE, 2014.

[8] Kazuhiro Yamashita, Changyun Huang, Meiyappan Nagappan, Yasutaka Kamei, Audris Mockus, Ahmed E Hassan, and Naoyasu Ubayashi. Thresholds for size and complexity metrics: A case study from the perspective of defect density. In *2016 IEEE international conference on software quality, reliability and security (QRS)*, pages 191–201. IEEE, 2016.

[9] Karine Mordal, Nicolas Anquetil, Jannik Laval, Alexander Serebrenik, Bogdan Vasilescu, and Stéphane Ducasse. Software quality metrics aggregation in industry. *Journal of Software: Evolution and Process*, 25(10):1117–1135, 2013.

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

[11] Stefan Wagner, Andreas Goeb, Lars Heinemann, Michael Kläs, Constanza Lampasona, Klaus Lochmann, Alois Mayr, Reinhold Plösch, Andreas Seidl, Jonathan Streit, et al. Operationalised product quality models and assessment: The quamoco approach. *Information and Software Technology*, 62:101–123, 2015.

[12] Robert Baggen, José Pedro Correia, Katrin Schill, and Joost Visser. Standardized code quality benchmarking for improving software maintainability. *Software Quality Journal*, 20(2):287–307, 2012.

[13] Jean-Louis Letouzey and Thierry Coq. The sqale analysis model: An analysis model compliant with the representation condition for assessing the quality of software source code. In *Advances in System Testing and Validation Lifecycle (VALID), 2010 Second International Conference on*, pages 43–48. IEEE, 2010.

[14] K. Mordal-Manet, F. Balmas, S. Denier, S. Ducasse, H. Wertz, J. Laval, F. Bellingard, and P. Vaillergues. The squale model; a practice-based industrial quality model. In *2009 IEEE Int. Conf. on Software Maintenance (ICSM)*, pages 531–534, Sept 2009.

[15] Maria Ulan, Sebastian Hönel, Rafael M Martins, Morgan Ericsson, Welf Löwe, Anna Wingkvist, and Andreas Kerren. Quality models inside out: Interactive visualization of software metrics by means of joint probabilities. In *2018 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 65–75. IEEE, 2018.

[16] Roger B Nelsen. *An introduction to copulas*. Springer Science & Business Media, 2007.

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

[18] C. Spearman. General intelligence, objectively determined and measured. *The American Journal of Psychology*, 15(2):201–292, 1904.

[19] Maurice Kendall. *Rank correlation methods*. Griffin, 1948.

[20] J Martin Bland and Douglas Altman. Measuring agreement in method comparison studies. *Statistical methods in medical research*, 8(2):135–160, 1999.

[21] Myles Hollander and Douglas A Wolfe. *Nonparametric statistical methods*. Wiley-Interscience, 1999.

[22] Miltiadis Allamanis and Charles Sutton. Mining source code repositories at massive scale using language modeling. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 207–216. IEEE Press, 2013.

[23] Tibor Bakota, Péter Hegedűs, Péter Körtvélyesi, Rudolf Ferenc, and Tibor Gyimóthy. A probabilistic software quality model. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 243–252. IEEE, 2011.