Software Quality Journal (2021) 29:863–899
https://doi.org/10.1007/s11219-021-09568-9

Copula-based software metrics aggregation

Maria Ulan[1] · **Welf Löwe**[1] · **Morgan Ericsson**[1] · **Anna Wingkvist**[1]

Accepted: 8 July 2021 / Published online: 24 August 2021

Abstract
A quality model is a conceptual decomposition of an abstract notion of quality into relevant, possibly conflicting characteristics and further into measurable metrics. For quality assessment and decision making, metrics values are aggregated to characteristics and ultimately to quality scores. Aggregation has often been problematic as quality models do not provide the semantics of aggregation. This makes it hard to formally reason about metrics, characteristics, and quality. We argue that aggregation needs to be interpretable and mathematically well defined in order to assess, to compare, and to improve quality. To address this challenge, we propose a probabilistic approach to aggregation and define quality scores based on joint distributions of absolute metrics values. To evaluate the proposed approach and its implementation under realistic conditions, we conduct empirical studies on *bug prediction* of ca. 5000 software classes, *maintainability* of ca. 15000 open-source software systems, and on the *information quality* of ca. 100000 real-world technical documents. We found that our approach is feasible, accurate, and scalable in performance.

Keywords Quality assessment · Quantitative methods · Software metrics · Aggregation · Multivariate statistical methods · Probabilistic models · Copula

1 Introduction

Software quality assessment helps to improve software and its development based on qualitative and quantitative data. Mathematical models are generally used to explain aspects of phenomena numerically at a statistically significant level, and quality models could serve as mathematical models for quality assessment.

✉ Maria Ulan
maria.ulan@lnu.se

Welf Löwe
welf.lowe@lnu.se

Morgan Ericsson
morgan.ericsson@lnu.se

Anna Wingkvist
anna.wingkvist@lnu.se

[1] Data-driven Software and Information Quality Group, Centre for Data Intensive Sciences and Applications, Linnaeus University, Växjö 351 95, Sweden

Software quality models have a hierarchical structure, where quality is defined in terms of (sub-)characteristics and metrics in a tree-like or directed layered graph structure. For example, software quality can be decomposed into the characteristics reliability and maintainability (Boehm et al., 1978; McCall et al., 1977), or into functionality, usability, reliability, performance, and supportability (Grady & Caswell, 1987). Metrics refine the (sub-) characteristics and, by definition, provide quantitative data. To get a complete picture of (sub-) characteristics and ultimately of quality, several metrics need to be assessed and then aggregated.

Metrics have different units, scale types (Fenton, 1994), and distributions of values, which makes aggregation challenging. Quality assessment tools often use different statistical assumptions about the metrics distributions and their independence and aggregation often performed in an ad-hoc manner. Thus, different tools give different recommendations for the same input (Lincke et al., 2008; Ericsson et al., 2013). Hence, there is a need to define the aggregation of metrics in quality models formally, based on accepted mathematical theories. This way, quality scores, the result of metrics aggregation, become uniquely interpretable and provide a common basis for decision making. The interpretation of aggregation should be aligned with intuitive reasoning, expert knowledge, and common sense, but also based on mathematical soundness to remove uncertainty, subjectivity, and degrees of freedom for quality assessment tool providers.

In short, the main objective of our research is to aggregate different metrics into a single score in a mathematically sound way that makes sense. As we need to assess quality repeatedly, the approach must also allow for an efficient and scalable implementation.

Note that this aggregation is related, but orthogonal to the task of integrating different metric values from one level (e.g., the level of individual classes) to a higher level (e.g., that of a package). This latter integrates metrics values of the *same* metric (e.g., lines of code) of different software artifacts of the same kind (e.g., classes) to a single value (e.g., the average) of all artifacts in a container level (e.g., a package). Instead, aggregation defines a score of a single software artifact (e.g., a class) incorporating metrics values of *the different* metrics (e.g., lines of code *and* cyclomatic complexity). This way it defines a quality score that unifies different aspects of quality of an artifact, each assessed by a different metric, and makes the artifacts comparable. In this paper, we distinguish *aggregation* (of different metrics of one artifact) from *integration* (of the same metric evaluated for different artifacts) and we only contribute to aggregation.

We suggest a probabilistic approach to quality assessment. We consider metrics as random variables and define quality models based on joint probabilities of the events related to the metrics. Probability distributions of multivariate random variables are generally more complex compared to univariate distributions, since there might be nonlinear dependencies between the random variables. Moreover, it is known that many relevant software quality metrics cannot even be described with univariate parametric probability distribution models (Barkmann et al., 2009; Ericsson et al., 2013). Therefore, our approach does not rely on parametric distribution models but uses numerical sample distributions instead. This allows for accurate computation of joint probabilities based on samples. Large sample sizes are necessary to cope with the high dimensionality of the random variable space (equal to the number of metrics in a quality model). To this end, we suggest an implementation that accommodates for efficient and scalable computation of joint probabilities. In summary, the work contributes with:

(i) an accurate and mathematically sound aggregation approach, where quality scores are based on joint probabilities.

(ii) an efficient implementation, that is scalable in performance.

The remainder of the paper is structured as follows. We formulate basic notations in Sect. 2, requirements for aggregation in Sect. 3, and summarize related work in Sect. 4. To address the requirements, we introduce a probabilistic approach for aggregation in Sect. 5. Section 6 evaluates our approach theoretically and empirically, based on Java software systems and XML technical documentations. Finally, Sect. 7 concludes the results and point out directions for future work. As complementary material, we present the theoretical foundations of the proposed approach in Appendix A, and R-scripts used in the experiments in Appendix B.

2 Background

2.1 Software quality

The notion of quality is diverse, complex, and ambiguous. Some define quality as *fitness for use* (Juran & Godfrey, 1999) others as *conformance to requirements* (Corsby, 1980). The standards ISO/IEC 25010:2010 (ISO, 2010) and IEEE 610.12-1990 (IEEE, 1990) define quality as a mix of both: *a degree to which a product satisfies requirements, customer and user needs, and/or expectations*. Alternatively, quality is also regarded as an *ideal* to put an effort to achieve (Garvin, 1984).

The way we measure software quality depends on how we define quality and what viewpoint we take (Kitchenham & Pfleeger, 1996), e.g., what goals we want to achieve with quality. Quality models are an attempt to define and quantify the notion of quality. There are several possible quality models, depending on the different goals.

In order to control the quality of software, it needs to be operationalized. To this end, quality models break down quality into (sub-)characteristics and then into metrics, mapping observations to numbers. They follow the "Factor-Criteria-Metrics" structure (McCall et al., 1977).

For example, the ISO/IEC 25010:2011 standard (ISO, 2010) decomposes *software quality* into eight *quality characteristics*: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability. Each quality characteristic is further decomposed into *quality sub-characteristics*. The standard assumes that quality metrics can measure sub-characteristics. There are many metrics proposed by the software engineering community (Chidamber & Kemerer, 1994; Henderson-Sellers, 1995; Martin, 2002). Source code metrics have a long tradition, and many of them are validated (Basili et al., 1996). Companies quite often use their own (set of) metrics and customized quality models (Wagner, 2013).

A quality model should also define the aggregation of numerical values from metrics to quality. However, neither the standard nor the metrics definitions provide clear definitions of how metrics should be aggregated, which degrades the quantification of quality to an implementation decision. This results in considerable variations in how it is implemented with negative effects on the repeatability and comparability of quality assessment (Lincke et al., 2008; Ericsson et al., 2013).

A software development process is a complex and multidimensional process developing, howsoever, the executable software product, its source code, documentation, user manuals, specifications, databases, etc. We define a *software artifact* as any output of an arbitrary

stage of a software development process. Software artifacts can be changed, merged, and modified during the development process. They are the objects of quality assessment and control.

Software metrics provide quantitative data for quality assessment. We define a *software metric* as a function mapping a *software artifact* to a numerical value referred to as the *metric value*. *Measurement* is the process of applying a metric to a software artifact to get a metric value. We avoid using the notion of measurement as a synonym for metric.

We neither aim to define (yet another notion of) quality nor (yet another set of) software metrics. We do not make any assumptions about possible distribution laws of software metrics. In general, their distributions laws can be approximated numerically by observing a (representative, sufficiently large) sample of the population (Hald, 2007). We focus on aggregation, i.e., how one can combine different software metrics into a single measure, that preserves properties of original data with a minimum of human supervision.

2.2 Distributions and copulas

We will treat software metrics as random variables and define here the necessary mathematical foundations.

The *cumulative distribution function* CDF_X of a random variable X is the probability that X will take a value less than or equal to x, i.e., $CDF_X(x) = Pr(X \leq x)$. It requires that X is measured at least on an ordinal scale, i.e., "less than or equal" ($\leq$) is defined on X. CDF_X defines the so-called *marginal probability distribution* of X.

The *empirical cumulative distribution function* $ECDF(x)$ is a good approximation of $CDF(x)$. It can be calculated as the relative frequency of observations in the sample $\hat{X}$ that are less or equal x, i.e.,

$$ECDF_X(x) = \frac{|\{\hat{x}|\hat{x} \in \hat{X}, \hat{x} \leq x\}|}{|\hat{X}|},$$

where $|\cdot|$ is the size of a set.

To model several metrics, we need to discuss different random variables and their so-called *joint probability distribution* instead of their individual marginal distributions. Different random variables might be dependent. To model their joint probability distribution, we refer to *Sklar's theorem* (Rüschendorf, 2009). It states that we can describe the joint probability distribution of a vector of random variable $(X_1, \ldots, X_k)$ by their marginal probability distributions $CDF_i(x) = P(X_i \leq x), i \in [1, k]$ and a *copula* function *Cop*. It allows us to separate the modeling of the marginal distributions from the dependence structure, which is expressed in suitable copula, i.e.,

$$CDF(x_1, \ldots, x_k) = Cop(CDF_1(x_1), \ldots, CDF_k(x_k))$$

Each $CDF_i(x_i)$ is a monotonously increasing function mapping X_i to the interval [0, 1]. Without loss of generality, we can assume that each value $x_i \in X_i$ occurs with a probability larger than 0. Hence, $CDF_i(x_i)$ is even a *strictly* monotonously increasing function. As a consequence, it is invertible for any given probability distribution of X_i, i.e., CDF_i^{-1} exists and is a function $[0, 1] \mapsto X_i$. The distribution of X_i and $CDF_i(x_i)$ uniquely defines the value $x_i = CDF_i^{-1}(CDF_i(x_i))$. Therefore, copula can be extracted from any joint distribution $CDF(\cdot)$ by using the *probability integral transform* as follows

$$Cop(u_1, \ldots, u_k) = CDF(CDF_1^{-1}(u_1), \ldots, CDF_k^{-1}(u_k)),$$

where $u_i = CDF_i(x_i)$.

There are several families of copulas; for details, we refer to Nelsen (2007). For example, if joint distribution $CDF(\cdot)$ is a multivariate normal distribution, it is called a *Gaussian copula*. *Archimedean copulas* allow to model dependence in arbitrarily high dimensions with only one parameter to determine the strength of dependence. The simplest example of Archimedean copula is the *independence copula*, applicable in the special case of independent random variables. It is defined as a product of the marginals, i.e.,

$$Cop(u_1, \ldots, u_k) = \prod_{i=1}^{k} u_i$$

The *Levi-frailty copulas* constitute yet another family (Mai & Scherer, 2009). They share some properties with Archimedean copulas regarding construction and analytical form. The weighted product of the marginals:[1]

$$Cop[\vec{\mathbf{w}}](u_1, \ldots, u_k) = \prod_{i=1}^{k} u_i^{w_i}, \sum_{i=1}^{k} w_i = 1$$

is one example from this family.

In practice, a nonparametric copula can be calculated directly from the empirical distribution. Assume a sample of size n of k random variables $(\hat{x}_1^j, \ldots, \hat{x}_k^j)$, with $1 \leq j \leq n$. Copula observations of a vector random variable observations are denoted as a k-tuple $(\hat{u}_1^j, \ldots, \hat{u}_k^j) = (CDF_1(\hat{x}_1^j), \ldots, CDF_k(\hat{x}_k^j))$. Then, the *empirical copula* is defined as follows.

$$Cop(u_1, \ldots, u_k) = \frac{1}{n} \sum_{j=1}^{n} \mathbb{1}(\hat{u}_1^j \leq u_1 \wedge \cdots \wedge \hat{u}_k^j \leq u_k),$$

where $\mathbb{1}$ is an *indicator function* with:

$$\mathbb{1}(cond) = \begin{cases} 1, & \text{if } cond \\ 0, & \text{otherwise} \end{cases}$$

We use the copula functions to define the aggregation of metrics, more details in Sect. 5. To compare with the state-of-the-art probabilistic approach, we use Levi-frailty copula, and to compare with a ground truth, we use empirical copulas, more details in Sect. 6.

3 Requirements for aggregation

An approach that aggregates different metrics of one artifact into a single quality score should make sense and it should allow for an efficient and scalable implementation. This section breaks down these objectives into the requirements of a suitable approach that we can assess either theoretically or in experiments.

[1] Note that they are parameterized with a suitable vector $\mathbf{w}$ of weights.

Quality models should be objective and unambiguous. Aggregation should map different values of different metrics to a single quality interval (Mordal et al., 2013). Each (aggregated) metric value should be well defined, easy to understand, and should allow for root-cause analysis (Heitlager et al., 2007).

Without loss of generality, we assume that values for aggregation are normalized to the interval [0, 1] and that 0 (1) represents the worst (best) possible quality. We use the *ECDF* for the normalization, cf. Sect. 2.2, which is mathematically well defined and interpretable in the context of software quality metrics. Note that there is no objectively correct way to normalize metrics, i.e., any normalization is based on some goals and assumptions. However, metrics have different units, scale types, and distributions of values, in general, and aggregation must not add "apples and oranges." Using *ECDF* for normalization solves this problem as it transforms metrics values to probabilities. In Sect. 5.1, we discuss in detail how we normalize the metrics.

To be mathematically well defined, we require metrics aggregation to implement an *aggregation operator* (Calvo et al., 2002), i.e., it should satisfy the following

Definition 1 *Aggregation Operator.* A function A that maps the (k-dimensional) unit cube onto the unit interval

$$A : [0,1]^k \mapsto [0,1] \tag{1}$$

is called a *aggregation operator* if it satisfies the following properties:

$$A(0,\dots,0) = 0 \tag{2}$$

$$A(1,\dots,1) = 1 \tag{3}$$

$$x_1 \leq y_1, ..., x_k \leq y_k \Rightarrow A(x_1,\dots,x_k) \leq A(y_1,\dots,y_k) \tag{4}$$

A special case of an aggregation operator is the *aggregation of a singleton*, i.e., the unary operator $A : [0,1] \mapsto [0,1]$, that is usually used to get basic statistics (e.g., min, max, mean, standard deviation, expected value, etc.) for a single variable. In the context of software quality, aggregation of a singleton could be used for *integration*, i.e., the mapping of the values of a single metric or aggregated quality score evaluated for different software artifacts, e.g., all classes of a package, which is not our scope here.

In the context of software quality, the *boundary condition* (cf. Eqs. (1) and (2)) states that if we aggregate only the worst (best) metrics values, the aggregate represents the worst (best) quality. The *Monotonicity* (cf. Eq. (3)) states that if the metrics values increase or remain equal, then the aggregate increases or remains equal.

We refer to artifacts with a quality score of 0, i.e., the worst possible score, as outliers. Considering a usually limited budget for a quality assurance, such outlier artifacts should be inspected first, even if they are not necessarily bad in all quality aspects.

We argue that a well-written software artifact is usually good or at least acceptable is all aspects. Conversely, software artifacts that are bad in terms of at least one and good/acceptable in other aspects should at least to be quality reviewed. For instance, a class that is not test-covered, but otherwise inconspicuous should still stick out. This is way each metric get a "veto" right in our aggregation approach. In order to easily spot software artifacts with extremely bad values in a single metric, we implement a stronger condition than Eq. (1) forcing the aggregated quality to be poor even if only a single metric indicates that. This poor quality cannot be evened out by better values for other metrics.

$$\lim_{x_i \to 0} A(x_1, \ldots, x_i, \ldots, x_k) = 0, \tag{5}$$

Equation (1') would arguably not make sense in *integration* contexts. For example, if there was one badly cloned file in a large system, the overall quality score of the whole system should not be zero.

Metrics-based quality assessment aims at predicting a ground truth in the software artifacts, e.g., a class is hard to maintain or prone to contain hidden bugs. In order to make sense, metrics aggregation should be an accurate predictor of such a ground truth.

However, the ground truth is often hard to predict regardless of the prediction method. The maintainability of a class, for instance, can be defined as the integral effort over all change activities cause by that class during its lifetime. It regards both (bug fixing, refactoring, and extending) changes and the effort in understanding the system prior to such changes. This is obviously hard to measure objectively and even harder to predict.

Therefore, quality assessment more often than not only aims at *comparing and ranking* software artifacts w.r.t. a ground truth and aggregation should support such a comparison. Since we cannot expect to predict the ground truth based on metrics, we define *accuracy* as the agreement of the ranking induced by the ground truth (e.g., the ranking of classes w.r.t. their maintainability) with the ranking induced by the aggregated quality score (aggregating, e.g., a set of maintainability metrics).[2]

To practically enable comparison and to suggest meaningful rankings, we require an aggregation also to be sensitive. The *sensitivity* of a set of vectors $\vec{\mathbf{x}}$ of values is the ratio between the number of unique aggregation outputs $A(\mathbf{x})$ and the set cardinality. In the context of software quality, this is the ratio between the number of different quality scores and the number of scored software artifacts.

However, we need to discuss differences of metrics values and quality scores that make sense. Let $\mathbf{x}$ and $\mathbf{y}$ be two vectors of metrics values of two artifacts, and $\mathbf{u}$ and $\mathbf{v}$ the respective normalized metrics values. If the artifacts are sufficiently close, i.e., $||\vec{u} - \vec{v}||_2 \leq \delta$ for some small δ, then decision makers cannot "sense" the difference between the corresponding software artifacts and perceive them as equal in quality. Then, it is acceptable that $A(\vec{x}) = A(\vec{y})$. Conversely, if $||\vec{u} - \vec{v}||_2$ is larger than δ, we expect $A(\vec{x}) \neq A(\vec{y})$.[3] In human reasoning, the observed sensitivity of aggregates, i.e., the question whether $A(\vec{x})$ is equal $A(\vec{y})$ or not depending on $||\vec{u} - \vec{v}||_2$, limits the suitability of an aggregation function (Dujmović, 2013). We set δ relative to the number of artifacts n: a artifact is perceived equal in quality if $\delta = [n]^{-1}$, where $[n]$ rounds to the nearest power of 10. The sensitivity of an aggregation operator A is defined as the number of artifacts that A distinguishes over the number of artifacts that it is expected to distinguish according the the above discussion.

We expect the computation of aggregated quality scores to perform well and to *scale* when large sets of software artifacts need to be assessed, i.e., $A \in \mathcal{O}(n \times k)$ for n artifacts and k metrics.

In summary, we require an aggregation approach:

R_1 to be mathematically well defined and fulfill Eqs. (1), (2), and (3),
R_2 to provide a mathematically sound interpretation of scores,

[2] The selection of a appropriate metric for measuring agreement of rankings is not trivial, but deliberately left out of the requirements discussion here. We will come back to it in the evaluation Sect. 6.

[3] We don't distinguish the importance of metrics for calculating this distance. This would be a meaningful generalization of the notion of sensitivity, especially, in the context of weighted copulas.

$\boldsymbol{R_3}$ to be able to combine metrics of different scales and distributions to a single quality score,
$\boldsymbol{R_4}$ to be accurate,
$\boldsymbol{R_5}$ to have an appropriate sensitivity, and
$\boldsymbol{R_6}$ to have a scalable performance.

We evaluate the requirements $\boldsymbol{R_1 - R_3}$ theoretically in Sect. 6.1 and the requirements $\boldsymbol{R_4 - R_6}$ empirically in a number of experiments described in the remainder of Sect. 6. We do not claim that there exists a *unique* approach that satisfies the requirements $\boldsymbol{R_1 - R_6}$.

Note that Eq. 4 is a not a requirement as there might be well-defined aggregation cases, where a "veto" does not play a crucial role to choose software artifacts for an inspection and, e.g., a threshold for metrics can be used instead.[4]

To resolve possible ambiguity in $\boldsymbol{R_4}$, $\boldsymbol{R_5}$, and $\boldsymbol{R_6}$, we quantify *"to be accurate," "to have an appropriate sensitivity,"* and *"to have a scalable performance"* as follows.

Accuracy is the agreement of a measured value to a standard or the true value. When the ground truth is available, we measure an agreement with the ground truth (i.e., correlation between an approach and the ground truth). When the ground truth is not available, we measure the agreement with the metric- and quality-goal-specific state-of-the-art approach (i.e., correlation between two approaches). Higher agreement means higher accuracy.

Sensitivity is the ability to distinguish as many data points as meaningful to distinguish in the raw data. We measure the ratio between unique outputs of an approach and the number of tuples in the raw data expected to be distinguished. An approach has the highest possible sensitivity if this ratio equals to 1, and it has the lowest possible sensitivity if the ratio is n^{-1} for n artifacts.

Scalable performance is the ability to accommodate rising resource demand gracefully. We analyze/measure the response time, i.e., the total computational time of an approach to aggregate metrics values to a quality score. An approach has a reasonable performance if the response time is analyzed to be at most linear in both number of software artifacts and number of metrics, and measured in the range of milliseconds on today's computers.

4 Related work

Metrics are often defined at a method or class level, but quality assessment sometimes requires insights at the component or system level, which also requires another type of aggregation. The distribution of metrics values is usually skewed, so well-known methods for this type, such as *mean* and *median*, are not representative enough. Therefore, some effort has been directed into metrics integration based on *thresholds* (Heitlager et al., 2007; Correia & Visser, 2008; Alves et al., 2011; Oliveira et al., 2014) to map source code level metrics to software system rating. However, in some cases, such integration can show degradation in software quality, while the code was actually improved (Mordal et al., 2013). Using inequality indices, e.g., the Gini index, instead of simple thresholds reduces this

[4] We do not consider such cases, since defining a thresholds should be based on both data and expert opinion. The latter does not allow for metrics and quality goal agnostic aggregation.

problem (Vasa et al., 2009; Serebrenik & Van Den Brand, 2010). Still, two very different distributions can have the same inequality index, since it does not capture where in the distribution the inequality occurs. In this paper, however, we do not address integration along the structure of software artifacts, e.g., from methods, to classes, and to the system. Instead, we suggest orthogonal aggregations along the structure of quality models, from low-level metrics, to quality (sub-)characteristics, and to a high-level notion of quality.

Aggregation of different software metrics to a unique measure has been researched before. Oman and Hagemeister (1994) introduced the *Maintainability Index*, a single-value indicator that uses a polynomial to aggregate four quality metrics: *Halstead Volume V*, *Cyclomatic Complexity G*, *Number of lines of code LOC*, and, in an improved version the *Percentage of comments CM*. To define the aggregation model coefficients, they assessed several systems in C and Pascal, using both metrics and expert ratings. Then, they applied statistical regression analysis to find the coefficients mapping the metrics to the experts' opinion[5]. The Maintainability Index aggregates averages but, since metrics distributions are skewed, taking the average implies a significant loss of information (Vasilescu et al., 2011). As all coefficients are negative, it is monotonically decreasing; a trivial transformation $MI' = -MI$ makes it satisfy Eq. (3). In general, aggregations using a weighted average do not satisfy Eq. (4). In general, however, aggregations using a weighted average do not satisfy Eq. (4), and if input is not normalized to a common unit and scale, they don't satisfy $\boldsymbol{R_3}$.

There are several practical quality models that extract a single quality score out of several metrics, in one step directly or over an intermediate aggregation to (sub-)characteristics and then further to a quality score. For example, in *QMOOD* (Bansiya & Davis, 2002) aggregation is based on weighted linear combination; in *SQUALE* (Mordal-Manet et al., 2009) aggregation is based on the combination of the values by a function; in *SQALE* (Letouzey & Coq, 2010) on fixed thresholds; and in *SIG* (Baggen et al., 2012) on thresholds derived from a benchmark. Such quality models, by definition, do not fulfill requirement $\boldsymbol{R_1}$; they do satisfy Eq. (3) but not Eqs. (1), (4), and (2).

Wagner et al. (2015) addressed the aggregation of metrics to abstract quality characteristics by an operationalized product hierarchical quality model *QUAMOCO* where aggregation is based on fixed thresholds and an additional mapping to grades. *QUAMOCO* satisfies Eqs. (1), (2), and (3). Moreover, it can even satisfy Eq. (4) by choosing an appropriate grading function. It uses the concept of *utility* from Multi-Attribute Utility/Value Theory (MAUT/MAVT) (Vincke, 1992) to represent the strength of a preference a decision maker (expert) has among software artifacts regarding a specific quality aspect (decision criterion). A frequent assumption in MAUT/ MAVT is that utility functions are independent of each other. However, it was shown in many decision problems that criteria are indeed dependent (Carlsson & Fullér, 1995).

Some approaches try to detect dependencies between metrics in order to mitigate the effect of assessing the same fundamental issue with different metrics. Most approaches do not consider interference with third metrics and assume linear dependencies. If metrics are independent, they are also linearly independent. However, the opposite is not necessarily true. If metrics are mutually independent, they are also pairwise independent. Again, the opposite is not necessarily true. For example, Gil and Lalouche (2017) study collinearity and show a strong pairwise correlation between size and metrics from the Chidamber and

[5] The Maintainability Index $MI = 171 - 5.2\ln(V) - 0.23G - 16.2\ln(LOC)$. A minor issue of *MI*: it uses the logarithms and $\log(0)$ is undefined. The improved version $MI = 171 - 5.2\log_2(V) - 0.23G - 16.2\log_2(LOC) + 50\sin(\sqrt{2.4CM})$ suffers from the same problems.

Kemerer (CK) metrics suite (Chidamber & Kemerer, 1994). They study (only) linear and pairwise dependence. Similar restrictions apply to the dependency analysis of feature selection techniques used to construct defect prediction models. The interpretation of these models relies on selected software metrics. AutoSpearman (Jiarpakdee et al., 2018) considers *collinearity* and *multi-collinearity* to extract appropriate metrics automatically. It disregards nonlinear dependencies. Disregarding (all or certain kinds of) dependencies between metrics hampers the interpretability of the aggregation results and, hence, violates requirement $\boldsymbol{R_2}16.2\log_2(LOC) + 50\sin(\sqrt{2.4CM})$. Aggregation is defined with logistic regression or random forests leading to classification models, which in general cannot guarantee requirement $\boldsymbol{R_1}$.

We propose a probabilistic approach to metrics aggregation, which is not unprecedented. Morasca (2009) suggested to consider conditional probability representations for external software attributes, e.g., defining *reliability* as the probability of failures occurring in a given time interval with a given program and in a given environment, or *modifiability* as the probability that a given artifact, in a given modification environment, is modified with a given amount of effort etc. These probabilities can only be estimated. Morasca suggested to use *hazard function* to define a model to estimate the values. No aggregation was suggested so the approach fails requirement $\boldsymbol{R_3}$.

Bakota et al. (2011) suggested a *goodness* function, a continuous generalization of thresholds, and measure goodness for a software system by comparing metrics distributions of the assessed and other known systems. Aggregation is defined as a combination of goodness values, weights, and probabilities of this comparison. The authors suggest to aggregate at most three metrics at once to avoid the negative effect of their exponential increase in computational costs. This violates the requirement $\boldsymbol{R_6}$. To mitigate this problem, they use Monte Carlo randomization avoiding exponential complexity. However, then the accuracy of their approach decreases with the number of metrics, which would violate requirement $\boldsymbol{R_5}$. We compare this approach with ours in more detail in Sect. 6.2.2.

The idea of aggregation presented in this paper was first described by Ulan et al. (2018). Here, we extend it in three ways: (i) We added and motivated aggregation requirements and assessed the state of the art against them. (ii) To compare with the state-of-the-art probabilistic approach, we added a weighted aggregation to the original only unweighted one. (iii) We evaluated the practical requirements $\boldsymbol{R_4} - \boldsymbol{R_6}$ empirically in several production-scale experiments.

5 Copula-based approach to aggregation

In this section, we present a probabilistic approach to aggregation in order to make quality models both formally well defined and interpretable. We present basic mathematical ideas to make it easy to understand and to replicate the proposed approach. Theoretical foundations can be found in Appendix A, and R scripts implementing the approach in Appendix B, resp.

We assume that the view on quality and the goal of its assessment are given, along with the relevant metrics, their contribution direction to the goal, i.e., whether high values are indicators of good or bad quality in the given assessment context. The metrics values of software artifacts are given as well.

We consider software quality a *latent variable* (Borsboom et al., 2003); it cannot be observed directly, but is inferred from (software) metrics that measure different aspects of quality, e.g., product or a development process. Formally, we use a mathematical model:

$$Model(Quality) := (\mathcal{A}, \mathcal{M}, \mathbb{R}, \succeq) \tag{6}$$

$\mathcal{A}$ is a set of software artifacts and $\mathcal{M}$ is a vector of software metrics. Each metric is a function that maps from $\mathcal{A}$ to a subset of $\mathbb{R}$, e.g., a subrange of the natural numbers $\mathbb{N}$ (counts) or the rational numbers $\mathbb{Q}$ (ratios). $\succeq$ is a non-strict preference relation to compare software artifacts with respect to their quality.

We rest on a well-known model of decision making proposed by Simon (1960) and adapt it for multi-criteria ranking in the setting of software quality assessment. We decompose the process into three steps that are detailed in the subsections below.

1: Calculate metrics scores by normalizing metrics so that resulting values are in the range [0, 1] with the same direction (0 is worst, 1 best) using *ECDF* of the marginal distributions.
2: Aggregate individual metrics scores into a quality score using a copula function.
3: Rank software artifacts based on their quality scores.

5.1 Calculating metrics scores

Given software metrics $\mu_1, \cdots, \mu_k$, we represent the assessment result for software artifacts $a_1, \ldots, a_n$ as an n by k *quality matrix* $[m_{ji}]$ of metrics values. We denote by $m_{ji}, j \in \{1, n\}, i \in \{1, k\}$, the quality of an artifact a_j assessed with metric μ_i. We use $\mathbf{m}_i = [m_{1i}, \ldots, m_{ni}]^T \in M_i^n$ to denote the i-th column of the quality matrix, which represents metrics values for all software artifacts with respect to metric μ_i, where M_i is the domain of its values.

We denote by σ_i a *metric score function*. It normalizes the metrics values. As the original metrics values m_{ji}, their corresponding normalized values $s_{ji} = \sigma_i(m_{ji})$ still indicate the degree to which the software artifact $a_j \in \mathcal{A}$ performs in the metric μ_i. The metric and score functions have the following signatures.[6]

$$\begin{aligned} \mu_i &: \quad \mathcal{A} \mapsto M_i \\ \sigma_i[\mathbf{m}_i] &: \quad M_i \mapsto [0, 1] \end{aligned}$$

Software metrics can have different directions, i.e., large values indicate either poor or good quality in an assessment context. Because of their symmetry, we will focus on large values (and drop low values) to indicate higher satisfaction in this discussion.

However, software metrics might be defined such that both larger and smaller values indicate good (bad) quality. For example, the metric *instability* $I = \frac{Ce}{Ca+Ce}$, where Ce are outgoing dependencies and Ca incoming dependencies, indicates good quality for values close to 0 or to 1. We could transform this metric to meet with the required property that larger values indicate good quality, for example, by a simple transformation: $I' = 2 \times |I - 0.5|$. While we can come up with such transformation for (arguably any) concrete metrics, there is no objectively correct way to deduce such a transformation for any such metric without expert knowledge. Therefore, we assume that transformations of metrics into a uniform direction and an application of this transformation to the metrics values have been done prior to normalization.

[6] Note that σ_i is parameterized with $\mathbf{m}_i$, i.e., all observed metrics values for metric μ_i.

The performance of software artifact a_j according to the metric μ_i is defined as the probability of finding another software artifact with metrics value smaller or equal to the given value. We calculate such a normalized metric score using *empirical cumulative distribution function* (*ECDF*) as follows.

$$\sigma_i[\mathbf{m}_i](m_{ji}) = s_{ji} = \frac{1}{n}\sum_{j'=1}^{n} \mathbb{1}(m_{j'i} \leq m_{ji}), \tag{7}$$

where $\mathbb{1}$ is the *indicator function* with: $\mathbb{1}(cond) = 1$ if $cond$ and 0, otherwise.

The normalized metric score interpretation is the same for all metrics: the empirical probability of finding another software artifact that performs worse than the given one and 0 indicates the worst and 1 the best performance with respect to a specific metric in the concrete quality assessment context.

5.2 Aggregating metrics scores

Let s_{ji} be the normalized metric score of m_{ji} calculated by Eq. (5). This metrics value, in turn, is measured for a software artifact $a_j \in \mathcal{A}$ with a metric μ_i. Corresponding to the *quality matrix* $[m_{ji}]$ we define a *quality score matrix* $[s_{ji}]$. All scores are in [0, 1] and have the same interpretation and the same direction, which makes them comparable. Hence, it is possible to aggregate them.

We define an aggregation in terms of marginal distributions and a suitable *copula* function with a meaningful interpretation. It defines an *aggregated quality score* q_j of an artifact a_j.

$$A(s_{j1}, \ldots, s_{jk}) = q_j = Cop(s_{j1}, \ldots, s_{jk}) \tag{8}$$

A copula function that allows for a meaningful interpretation is the (empirical approximation of the) joint probability of the marginal scores. It can be interpreted as the probability of an artifact performing worse than or equally good as a_j in *all* metrics $\mathcal{M}$.

$$Cop(s_{j1}, \ldots, s_{jk}) = \frac{1}{n}\sum_{l=1}^{n} \mathbb{1}(s_{l1} \leq s_{j1} \wedge \cdots \wedge s_{lk} \leq s_{jk}) \tag{9}$$

The Levi-frailty copulas are another family of copula functions with a meaningful interpretation. They define the quality of an artifact as the weighted geometric product of the marginals:

$$Cop[\mathbf{w}](s_{j1}, \ldots, s_{jk}) = \prod_{i=1}^{k} s_{ji}^{w_i}, \quad \sum_{i=1}^{k} w_i = 1 \tag{10}$$

assuming that it is known that the relative importance of a metric μ_i for a (sub-)characteristic is known and quantified by the corresponding weight w_i.

Regardless of the copula employed, if a (sub-)characteristic c is defined as the aggregation of metrics $[\mu_1, \ldots, \mu_k]$, then the result of normalization and aggregation is the *quality score* of c. We interpret it as a score of how good a software artifact performs in a set of metrics, compared to other artifacts; 0 indicates bad and 1 good quality.

Note that these aggregated (sub-)characteristic quality scores are themselves metrics by definition as they map each artifact to a subrange $[0, 1] \in \mathbb{R}$. Hence, aggregation may

repeatedly be applied aggregating, e.g., metrics to sub-characteristics and these further to characteristics and to an overall score.

5.3 Ranking software artifacts

Once the quality scores are computed, the artifacts can be ranked simply by ordering according to their score values. We assign the same rank to artifacts in case their aggregated scores are equal (up to margin of error $\varepsilon = [n]^{-1}, n = |\mathcal{A}|$, and $[n]$ rounds to the nearest power of 10).

A software artifact a_j is better than or equally good as another artifact a_l, if the quality score of a_j is greater than or equal to the quality score of a_l.

$$a_j \succeq a_l \Leftrightarrow q_j \geq q_l \tag{11}$$

Software artifacts with bad quality should be prioritized and inspected first; therefore, we rank software artifacts based on their ((sub-)characteristic) quality scores in acceding order: Software artifact with the lowest (highest) possible aggregated score will be ranked as first (last).

In the next section, we build our approach on empirical copula to evaluate how well it fulfills the requirements, and we build our approach on Levi-frailty copula to study the agreement with the state-of-the-art probabilistic approach. Note that the process of aggregation and ranking described above can be built on any copula function.

6 Evaluation

In Sect. 6.1, we argue that our approach fulfills the theoretical requirements formulated in Sect. 3, i.e., it is mathematically well defined $(\boldsymbol{R_1})$, interpretable $(\boldsymbol{R_2})$, and to be able to combine metrics of different scales and distributions to a single quality score $(\boldsymbol{R_3})$.

To demonstrate that the proposed approach is valuable, it should also be shown that an aggregate metrics based on copulas leads to accurate $(\boldsymbol{R_4})$, sensitive $(\boldsymbol{R_5})$, and scaling $(\boldsymbol{R_6})$ assessments of quality. We evaluate empirically the fulfillment of the corresponding other requirements.

We confirm the accuracy $(\boldsymbol{R_4})$ in two studies described in Sect. 6.2. We consider two different contexts: bug prediction (with the ground truth available) and maintainability assessment (agreement with state-of-the-art probabilistic approach as a proxy of a ground truth). In the first study, we confirm the accuracy relative to other metric-based approaches applied in defect prediction. We compare ours with the baseline prediction models from a publicly available benchmark[7] by D'Ambros et al. (2010). These models aggregate different metrics to assess defect proneness on a software class level. We study whether or not our aggregation approach leads to an accurate assessment, i.e., we rank software classes by our aggregation results and by the predicted number of bugs for baseline approaches and compare these rankings against a ranked list of the same classes based on the ground truth, i.e., the observed post-release defects. In this study, we document that our aggregation is more accurate than the baseline approaches in most cases.

[7] http://bug.inf.usi.ch

We choose "error-proneness" as the assessed quality in this study as it gives an objective ground truth that is relatively easy to measure. Alternative evaluations comparing to expert-based rankings lack this objectivity. A defect prediction model is a function mapping metric values of a software artifact to a measure of its likelihood to be defective. The baseline models are aggregation models, i.e., weighted sum aggregation operators where weights are defined by regression. We do not aim to build the best performing defect predictor. In contrast, to illustrate the accuracy of our *unsupervised* aggregation, we study whether or not our approach could improve the performance of the existing baseline models for defect prediction, even when their weights are set using *supervised* regression, which is actually unfair to the disadvantage of our approach.

In the second study, we apply our approach to maintainability assessment and study the agreement with the alternative probabilistic approach of Bakota et al. (2011). We collect metrics values of classes of real-world software systems from publicly available *GitHub Java Corpus*[8] by Allamanis and Sutton (2013). In this study, we cannot compare to a ground truth and only show that our results are in agreement without deciding which one is more accurate.

We confirm sensitivity ($\boldsymbol{R_5}$) in Sect. 6.3, again in two studies. First, we show that our aggregation approach leads to an improved assessment sensitivity and anomaly detection compared to the alternative probabilistic approach of Bakota et al. (2011). Second, we apply our approach to information quality assessment of real-world technical documentations provided by our industrial partners and discuss sensitivity as a function of system size and number of metrics.

Finally, we confirm performance scalability ($\boldsymbol{R_6}$) in in Sect. 6.4. Using the same setup as in the sensitivity study, we assess how scalable (the implementation of) the quality score calculations of our approach are. We compare well to the alternative probabilistic approach, whose authors suggest to aggregate at most three metrics at once to avoid the negative effect of their exponential increase in computational costs.

For the empirical evaluations, we implemented all algorithms and statistical analyses in *R*.[9] We use the following *R* packages: *dplyr*[10] for extracting and filtering the data from given database, *ggplot2*[11] for visualization, *fitdistrplus*[12] for calculating basic statistics and for testing distribution laws, and *copula*[13] to generate additional data for additional sensitivity and performance tests. We calculate metrics and quality scores empirically by using multi-thread computations and optimized algorithms from the *R* package *Emcdf*.[14]

The quality score calculations and all measurements were performed and analyzed on the same computer under the same conditions. For each software class and documentation file, we obtained quality scores using third-party software. The metrics data were collected with the *VizzMaintenance*[15] Eclipse plug-in for software systems, and with the *Quality Monitor*[16] for the technical documentations.

[8] http://groups.inf.ed.ac.uk/cup/javaGithub/
[9] The R Project for Statistical Computing, https://www.r-project.org
[10] dplyr, https://cran.r-project.org/web/packages/dplyr
[11] ggplot2, https://cran.r-project.org/web/packages/ggplot2
[12] fitdistrplus, https://cran.r-project.org/web/packages/fitdistrplus
[13] copula, https://cran.r-project.org/web/packages/copula
[14] Emcdf, https://cran.r-project.org/web/packages/Emcdf
[15] VizzMaintenance, http://www.arisa.se/products.php
[16] Quality Monitor™, http://iqm.arisa.se/iqmonitor

6.1 Theoretical evaluation of $R_1 - R_3$

By definition, copulas are monotonically non-decreasing functions for each of their variables. Hence, the *boundary condition* (Eqs. (1) and (2)) and *monotonicity* (Eq. (3)) hold. Also, by definition:

$$\forall i \lim_{u_i \to 0} Cop(u_1, \ldots, u_i, \ldots, u_n) = 0,$$

hence, the stronger condition of Eq. (4) holds. Altogether, the suggested aggregation technique is mathematically well defined and $\boldsymbol{R_1}$ holds.

We interpret the score of as metric as the probability that an artifact with lower satisfaction w.r.t. this metric can be found. We interpret the aggregated quality scores according to the copula function used. This includes but is not limited to the two copulas suggested here.

The unweighted aggregation of scores computed on the joint probability of the marginals considers possible statistical dependencies between metrics but assumes that metrics are equally important. The weighted aggregation of scores considers weights to represent different importance of metrics but assumes that metrics are statistically independent. In the unweighted case, the quality score of an artifact is defined as the probability of another artifact existing that is strictly better in each metric. In the weighted case, the quality score of an artifact is defined as the joint probability of independent metric scores weighted according to the (application-specific) importance of the metrics. Both provide an intuitive interpretation of scores that is the same on every level of aggregation and, hence, $\boldsymbol{R_2}$ holds.

Copulas allow us to aggregate different metrics with different domains, scales, and directions of impact to a single value from the [0, 1] interval as it is a probability. Hence, $\boldsymbol{R_3}$ holds.

We just showed that $\boldsymbol{R_1 - R_3}$ hold for any copula function, especially, for empirical and Levi-frailty copulas that we will use in evaluation.

6.2 Evaluation of accuracy R_4

In Sect. 6.2.1, we evaluate accuracy in comparison of alternative metric-based approaches in the context of bug prediction with a ground truth at hand. In Sect. 6.2.2, we evaluate accuracy as alignment to an alternative probabilistic and (metric-based) approach in the context of maintainability assessment, since there is no ground truth available.

6.2.1 Comparison to alternative metric-based approaches

Evaluation Method and Measures There are many approaches to bug prediction. Some of them rely on more than one metric; hence, aggregation is needed. We use a publicly available benchmark (D'Ambros et al., 2010) that contains statistical data of five software systems, several sets of metrics, and an extensive comparison of several bug prediction approaches. We use the metrics values from this benchmark, perform aggregation, and relate the aggregated score with a known number of post-release defects as a ground truth. We compare other metrics-based approaches using the same set of metrics with ours. We study whether our aggregation improves bugs prediction models in terms of accuracy, i.e., in terms of agreement with the ground truth.

We apply our approach for bug prediction on a software class level. More specifically, we rank software classes according to their defect proneness (ground truth) and based on their quality scores obtained by our and alternative approaches. The alternative approaches are based on different sets of metrics, aggregated with generalized linear regression models. The (different alternative) regression models are trained to map (different sets of metrics) to the number of bugs. In other words, we compare our model of *unsupervised* aggregation with models of *supervised* regression against the ground truth.

We apply the same *measure* of predictive power as used in the related study:

Correlation. We measure the *Spearman's rho correlation* between rankings to assess the ordering, relative spacing, and possible functional dependency.

Quality Model and Dataset Description The bug prediction dataset (D'Ambros et al., 2010) consists of software class level metrics values and numbers of post-release bugs from five open-source software systems written in Java: *eclipse*[17] (997 classes), *equinox*[18] (324 classes), *lucene*[19] (691 classes), *mylyn*[20] (1 862 classes), and *pde*[21] (1 497 classes).

We compare metrics-based approaches using different metrics sets that require aggregation (see Table 1). We consider eight MOSER metrics, since bugs might be caused by changes (Moser et al., 2008). We also consider its subset NFIX+NR, since past defects (Zimmermann et al., 2007) and the number of changes (Graves et al., 2000) might be good predictors. We consider six CK metrics and eleven OO metrics. We consider both, the combination CK+OO and CK and OO in isolation, since complex components might be error-phone (Basili et al., 1996).

We also compare to the best performing approaches from a benchmark, called *WCHU* and *LDHH* proposed by D'Ambros et al., who argue that source code metrics change is a better approximation of code churn, and better describe the entropy of changes (D'Ambros et al., 2010). These approaches use several versions of a software system to sample the history of source code metrics. The *WCHU* (weighted churn of source code metrics) approach computes for each metric its churn as a sum of deltas of source code metrics values for each consecutive pair of samples. It contains seventeen metrics, each the churn of a metric from CK+OO. It uses weights to acknowledge that many small changes are more relevant than a few big changes. The *LDHH* (linearly decayed entropy of source code metrics) approach uses the churn to compute the entropy of source code metric changes. It contains seventeen metrics, each the entropy of a metric change from CK+OO. It accounts for how much a class changed.

We analyzed five open-source software systems to perform separate studies and compare the results to observe potential differences between them. We do 50-fold cross-validation, i.e., for each software system, we use 90% of the classes as a training set to build the regression models and the remaining 10% as a validation set. For our unsupervised aggregation model, we use the same validation data (ignoring the training data) to compute *ECDF* for the comparison with supervised models. For each model and fold, we compute the Spearman correlation of two rankings: the classes ordered by the actual number of bugs and by the aggregated

[17] Eclipse JDT Core, https://eclipse.org/jdt/core/, prediction release 3.4.
[18] Equinox framework, https://eclipse.org/equinox/, prediction release 3.4
[19] Apache Lucene, https://lucene.apache.org/, prediction release 2.4.0
[20] Mylyn, https://eclipse.org/mylyn/, prediction release 3.1
[21] Eclipse PDE UI, https://eclipse.org/pde/pde-ui/, prediction release 3.4.1

quality score for our approach and classes ordered by the number of predicted bugs for the other metrics-based approaches. The reported values of Spearman's rho are averages over 50-fold.

Summary of Results Table 2 shows the results of the correlation of the approaches applied for each software system. All correlations are significant at the 0.01 level, i.e., 99% confidence interval and $p < 0.01$. For each software system and set of metrics, we compared our copula-based approach with the generalized linear regression approaches. For each pairwise comparison, the better of the two values is highlighted by a gray background, and in boldface when there is a significant difference.

We observe that aggregation by our approach often improves the results. More specifically, aggregation using our approach performs better than the respective regression-based approach in the majority of cases. Only in 8 out of 35 cases, the alternative metrics-based approaches perform better. However, only two of them (highlighted in italic) show significant differences.

Ranking based on the quality scores obtained by our approach is, in most cases, closer to the ground truth ranking than rankings obtained by alternative metrics-based approaches that use generalized linear regression. This lets us conclude that our approach has higher predictive power, i.e., is more accurate than these alternatives.

6.2.2 Comparison to an alternative probabilistic approach

Evaluation Method and Measures Bakota et al. (2011) proposed an ISO/IEC 9126 based quality model, which also uses concepts of probability to handle the ambiguity coming from the lack of consensus on software quality. To the best of our knowledge, it is the only related research that investigates both the probabilistic nature of quality and aggregation of metrics. Therefore, we study whether aggregated quality scores obtained by our approach correlate with goodness functions obtained by this related state-of-the-art approach. The approaches are applied to maintainability assessment under the same experimental settings. Particularly, we use the same independent metrics and *integrate* metrics calculated for classes to the system level; recall *aggregation of a singleton* introduced in Sect. 3.

The two approaches use the same principle assumption—the probabilistic nature of quality—to measure software characteristics. They should agree in their results when applied to the same settings, i.e., the same systems, quality model, metrics, and weights Lincke et al. (2008). Indeed, both approaches use the same quality model and metrics, the *expected value* $\mathbb{E}$ to integrate class to system-level scores, and the same weights to indicate the importance of a metric to quality.

In Bakota et al., weights are obtained manually from an online survey where experts were asked about their opinions. However, since these weights are not reported, we generate synthetic weights then used in both approaches.

Approach 1 (Bakota et al., 2011). For each system S and each metric μ, we integrate the metric values of the system's classes c to the system level using the expected value of these values $E[\mu(c)] = \mu(S)$. For each metric μ of a training system S_0, a *goodness function* g is calculated as $\Delta_i = |\mu(S_0) - \mu(S_i)|$, where S_i are the systems from the benchmark. Then, these goodness functions are aggregated into the given system's goodness

Table 1 Metrics of the bug prediction models

Type	Metric	Description
MOSER	NR	Number of revisions
MOSER	NREF	Number of refactorings
MOSER	NFIX	Number of bug-fixings
MOSER	NAUTH	Number of authors who committed
MOSER	LINES	Lines added and removed
MOSER	CHURN	Code churn (sum, max, average)
MOSER	CHGSET	Change set size (maximum and average)
MOSER	AGE	Age and weighted age
CK	WMC	Weighted Method Count
CK	DIT	Depth of Inheritance Tree
CK	RFC	Response For Class
CK	NOC	Number Of Children
CK	CBO	Coupling Between Objects
CK	LCOM	Lack of Cohesion in Methods
OO	FanIn	Number of other classes that reference the class
OO	FanOut	Number of other classes referenced by the class
OO	NOA	Number of attributes
OO	NOPA	Number of public attributes
OO	NOPRA	Number of private attributes
OO	NOAI	Number of attributes inherited
OO	LOC	Number of lines of code
OO	NOM	Number of methods
OO	NOPM	Number of public methods
OO	NOPRM	Number of private methods
OO	NOMI	Number of methods inherited

by taking an integral $\int f(y)g_1(x_1)\dots g_n(x_n)\,dx\,dy$, where x traverses the domain of goodness functions, y traverses the simplex where each point represents possible weights, and f is a density function.

Approach 2 (our approach). For each class c of a system S and for each metric μ, we calculate the score $s(c)$ of the metric value $m(c) = \mu(c)$ using Eq. (5) based on the *ECDF* derived from metric values of a training systems $S_i \neq S$ from the benchmark.

Table 2 Spearman correlation between rankings obtained by the different approaches and the ranking from the ground truth

	Eclipse	Mylyn	Equinox	PDE	Lucene
MOSER Copula of MOSER metrics	0.323 0.365	0.284 *0.269*	0.534 0.573	0.165 0.211	0.238 0.267
NFIX+NR Copula of NFIX+NR	0.381 0.378	0.091 0.128	0.567 0.602	0.255 *0.228*	0.277 0.291
CK Copula of CK metrics	0.377 0.404	0.226 0.234	0.484 0.519	0.256 0.259	0.216 0.231
OO Copula of OO metrics	0.395 0.429	0.297 **0.305**	0.49 0.499	0.263 0.259	0.214 0.207
CK+OO Copula of CK+OO metrics	0.39 0.415	0.299 0.297	0.453 0.491	0.284 0.278	0.214 0.221
WCHU Copula of WCHU metrics	0.419 0.443	0.279 0.281	0.56 **0.598**	0.278 0.293	0.285 0.326
LDHH Copula of LDHH metrics	0.408 **0.451**	0.272 0.267	0.53 0.573	0.296 **0.317**	0.333 **0.375**

For each c, we then aggregate $s(c)$ into a quality score using the *Levi-frailty copula* defined in Eq. (8). Finally, we integrate the quality scores for classes to a system quality score as the expected value of the class quality scores.

We collect metrics values of classes of real-world software systems. We then split the systems into training and test systems. The training systems are used to calculate the goodness functions for Approach 1 and *ECDF* for Approach 2, which are applied to the test systems. The result of both approaches is quality scores and lists of software systems from the test set ranked according to their quality scores.

We compare the approaches using the following *measures*:

Correlation. We measure *Spearman's correlation* (Spearman, 1904) between maintainability scores to assess the ordering, relative spacing, and possible functional dependency.

Agreement. We apply *Bland–Altman* statistics (Bland & Altman, 1999) between the maintainability scores coming from the two approaches to see a potential bias between the differences and to estimate an agreement interval between the approaches.

Ranking distance. We measure the *Kendall tau distance* (Kendal, 1948) between the two rankings, which counts the number of pairwise disagreements between two lists.

Quality Model and Dataset Description We applied metrics from the well-known CK metrics suite (Chidamber & Kemerer, 1994), which was selected based on the evidence of correlating with maintainability (Riaz et al., 2009).

Because of the scalability constraints of (Bakota et al., 2011), we do not aggregate more than three metrics at once. *"Empirically we found that the approach works well enough if the number of incoming edges is not higher than three for every aggregate node"* (Bakota et al., 2011). Therefore, we compare a very limited quality model with only three metrics.

Bakota et al. did not consider dependencies between metrics. Therefore, we selected the three metrics from the metrics suite with the lowest pairwise correlation: *DIT* (Depth of Inheritance Tree), *LCOM* (Lack of Cohesion in Methods), and *NOC* (Number of Children).

The chosen metrics, by definition, measure different aspects of quality. However, their aggregation provides an overview on quality rather than each single metric in isolation. Note we do not aim to define a new notion of quality. Instead, these metrics were chosen for fair comparison of two alternative approaches.

The weights of the metrics should not have a large influence on the results of the comparison of aggregations one and the same quality models, as long as they are the same for the aggregation approach and software system. As Bakota et al. (2011) did not report weights, we generated ten random combinations of weights. While complete randomness was a choice, we followed the quality model for maintainability used in the comparison study of Lincke et al. (2008) and generated weights in such a way that it is more likely that *LCOM* has a higher importance than *DIT*, which in turn is likely to get a higher importance than *NOC*.

We analyzed the *GitHub Java Corpus* (Allamanis & Sutton, 2013), a snapshot of open-source Java systems from October 2012 that contains code from 14 807 projects across a wide variety of domains amounting to 352 312 696 lines of code in 2 130 264 files. The systems included in the corpus were chosen by the following criteria: (i) they are written in Java, (ii) they are available on GitHub, and (iii) they were forked at least once.

We split the 14 807 projects 75%/25% into training and test sets.[22]

Summary of Results We find that under the same settings and assumptions, the approaches produce similar quality scores.

Figure 1a shows the correlation of the quality scores of the two compared approaches (scatter plot, bottom left), the distribution of scores of each approach (histograms top left for Approach 1 and bottom right for Approach 2) and Spearman's correlation coefficient (top right).

First, we observe a strong correlation between the two approaches.

Second, we observe an agreement between the approaches. In the *Bland–Altman plot*, cf. Figure 1b, each point corresponds to 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 horizontal (blue) line in the middle represents the mean difference between scores and the horizontal (red) lines on the top and the bottom, the 95% confidence interval ($mean \pm 1.96$ *Std. Dev.*). The span of the confidence interval of differences is about 0.1 (with 2 the maximum possible span), which is sufficiently small to conclude that the two approaches agree.

Third, the *Kendall tau distance* is 0.13185. It means that less than 14% of the system differ between the rankings of the two approaches.

We conclude that the two probabilistic approaches agree when applied under the same settings and assumptions. We cannot decide that either of them is more accurate as we lack a ground truth of maintainability.

6.3 Evaluation of sensitivity R_5

In Sect. 6.3.1, we evaluate sensitivity in comparison with an alternative probabilistic and (metric-based) approach of Bakota et al. In Sect. 6.3.2, we evaluate sensitivity of our approach in the context of information quality assessment.

6.3.1 Sensitivity in comparison with an alternative probabilistic approach

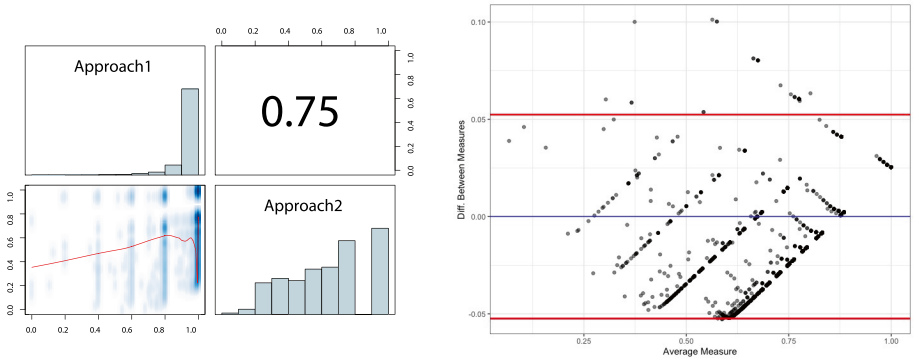
Evaluation Method and Measures We apply the same maintainability assessment as described in Sect. 6.2.2. However, here we assess the sensitivity with the following *measures*:

Anomaly detection. We determine the ratio between the number of detected outliers and the total number of outliers in a sample dataset for both approaches.

Overall Sensitivity. We determine the ratio between unique scores and the population size N for both approaches. Two scores, s_1 and s_2, are deemed identical if they are sufficiently close, i.e., $|s_1 - s_2| < \delta$, with $\delta = [n]^{-1}, n = |\mathcal{A}|$, and $[n]$ rounds to the nearest power of 10.

Sensitivity Ratio. We determine the ratio between unique scores and the unique tuples in the raw data for both approaches. It equals to 1 if an aggregation distinguishes as many data points as the raw data, and it approaches 0 for aggregation to the same score for all data points.

[22] The full list of the systems and the train-test split we used in the evaluation, http://groups.inf.ed.ac.uk/cup/javaGithub/



(a) Correlation and Distributions **(b)** Bland-Altman plot

Fig. 1 Agreement of maintainability scores between the approaches

Quality Model and Dataset Description We use the same quality model and the same dataset as described in Sect. 6.2.2.

Summary of Results We find that under the same settings and assumptions, our Approach 2 outperforms the probabilistic Approach 1 in detecting anomalies, and it has a higher sensitivity, i.e., it allows to differentiate more software systems regarding their maintainability. In detail:

First, the *anomaly detection* ratios equals 71.36% for Approach 1 and 92.59% for Approach 2.

Second, the *overall sensitivity* is 0.08828 for Approach 1 and 0.25196 for Approach 2, which is almost three times larger. The latter means that every fourth system has a different quality score.

Third, Approach 1 has a *sensitivity ratio* of 0.34305, Approach 2 a *sensitivity ratio* of 0.97909, i.e., almost all of the original sensitivity is maintained. Also, the histogram in the upper left of Fig. 1a confirms that a single quality score dominates in Approach 1. In contrast to that, the histogram in the lower right of Fig. 1a shows that Approach 2 can differentiate the software systems.

We conclude that the Approach 2 is more sensitive than Approach 1. Despite the fact that the approaches are by-and-large in agreement when applied under the same settings, they differ in terms of sensitivity. Note it does not mean that Approach 2 is more accurate than Approach 1.

6.3.2 Sensitivity in information quality assessment

Evaluation Method and Measures To be applicable under realistic conditions, sensitivity should scale to large systems. To test that, we apply our approach to information quality assessment of real-world documentations provided by our industrial partners. Here, we assess the sensitivity when our approach is applied to documentations of growing sizes. We aggregate 40 metrics and *measure*:

Overall Sensitivity as defined before.

Quality Model and Dataset Description We use a real-world quality model to assess information quality. It was customized for an industrial partner, a Swedish provider of telecommunication backbone infrastructures. The hard- and software components of this infrastructure are documented in operator, troubleshooting, and installation manuals using XML-based specifications. These specifications are semi-formal, but need to follow common guidelines. They are written in-house and by subcontractors inside Sweden and around the globe. Hence, there is a need to effectively and efficiently check and assure the quality of these documentations (Ericsson et al., 2012). The information quality model is based on the 40 metrics listed in Table 3. They are used to evaluate 10 sub-characteristics: Text Complexity, File Complexity, Hierarchy Complexity, Referential Complexity, Cloning Issues, Anti-patterns, Language Issues, Validity Issues, Utilization, and Others.

We analyzed 91 documentations. The majority of them consisted of a set of *XML* documents in proprietary *XML*, in *DITA XML*, or in *XHTML* format. We excluded 17 documentations that used other formats (proprietary PDF structures). We assessed the metrics for each of the remaining 74 documentation. To select similar documentations for further analysis, we applied statistical tests to check if the metrics values come from populations with similar distributions and, therefore, dropped another three documentations.[23] The final benchmark consisted of 238 versions of 71 XML documentations with $N = 93\,049$ XML documents and a dataset of $2\,595\,674$ metrics values in total.

Based on this seed of technical documentation metric values, we generate multivariate synthetic data with population sizes $N = \{10^3, 10^4, 10^5, 10^6\}$. The generated data contained the same metrics with the same properties (basic statistics of metrics, metrics distributions, and metrics dependencies) as observed in the seed. Generating synthetic data was necessary in order to vary the problem size for sensitivity (and later performance) assessments. To account for the possible effect of randomness and uncertainty, we generated ten different synthetic datasets for each population size N.

Summary of Results We find that the *sensitivity* decreases with the number of metrics and the sample population size increasing. It seems to approach zero for a sufficiently large number of metrics and population sizes, cf. Fig. 2.

However, our approach allows more than one level of aggregation, as defined in hierarchical quality models. Still, score calculation and interpretation remain the same on all levels of aggregation. Recall the real-world quality model that aggregates 40 metrics to ten characteristics first and then further to a single quality score (see Table 3). For this quality model, we also assessed the sensitivity of a two-step aggregation, where scores were first aggregated to the ten characteristic scores and then, with the same approach, to the actual quality score.[24]

[23] More precisely, we applied the nonparametric *Kolmogorov–Smirnov* (Hollander & Wolfe, 1999) and *Tukey–Duckworth* (Siegel and Tukey, 1960) tests. We considered both tests because *Kolmogorov–Smirnov* requires variables to be at least ordinal from the same distribution, while *Tukey–Duckworth* does not have such strong assumptions. The latter only requires that there are no repeating values in the combined sample. Only documentations that passed both of the tests with the significance cutoff at 0.05 were considered for further analysis. We admit the multiple testing problem. However, adjustment of p-values may reduce the chance of making type I errors, but may increase the chance of making type II errors. To this end, we decided to select documentations that passed both tests, but don't claim statistical evidence that the metrics values come from populations with the same distributions.

[24] One might assume that a similar multi-level approach is also mitigation of the scalability issues of Bakota et al. (2011). However, this would require expert-generated weights not only on the metrics level but also on the level of the (sub-) characteristics.

Table 3 Metrics of the information quality model

Sub-characteristic	Metric	Description
Text Complexity	N_ TextSize	Text size
Text Complexity	N_ SentenceSize	Sentence size
Text Complexity	R_ StepListFrac	Fraction of item list advices
File Complexity	N_ XMLSize	Number of all XML nodes
Hierarchy Complexity	N_ SubSectionDepth	Subsection depth
Hierarchy Complexity	N_ SubSectionWidth	Subsection width
Referential Complexity	N_ InternalReferences	Number of all internal References
Referential Complexity	N_ ExternalReferences	Number of all external References
Referential Complexity	N_ CrossReferences	Number of all references
Referential Complexity	N_ CrossPackReferences	Number of all cross package references
Referential Complexity	N_ InEdgeCount	Number of incoming references
Referential Complexity	N_ OutEdgeCount	Number of outgoing references
Cloning Issues	R_ XMLUniqueness	XML uniqueness
Cloning Issues	R_ XMLSimilarity	XML similarity to other documents
Cloning Issues	R_ TextUniqueness	Text uniqueness
Cloning Issues	R_ TextSimilarity	Text similarity to other documents
Cloning Issues	N_ TextSimilar	Number of text-similar documents
Anti-patterns	R_ Stability	Central document
Anti-patterns	N_ Isolated	Number of isolated documents
Anti-patterns	N_ BackRefs	Number of cyclic references
Anti-patterns	N_ BrokenRefs	Number of broken references
Anti-patterns	N_ PointAbstraction	Number of sections with one subsection
Anti-patterns	R_ RevOverTime	Change frequency
Language Issues	N_ LanguageChecks	Lack of language checks
Language Issues	N_ GrammarIssues	Number of all grammar issues
Language Issues	N_ SpellingIssues	Number of all spelling issues
Language Issues	N_ StyleIssues	Number of all style issues
Language Issues	N_ TerminologyIssues	Number of all terminology issues
Validity Issues	N_ CorrectedXMLissues	Number of all corrected XML issues
Validity Issues	N_ OtherXMLissues	Number of all other XML issues
Validity Issues	N_ MissingInfo	Missing/erroneous information tags
Utilization	N_ absVisitTime	Visit time
Utilization	N_ absVisitors	Number of visitors
Utilization	N_ absVisits	Number of visits
Utilization	R_avgClicksPerVisit	Number of clicks per visit
Utilization	R_avgClicksPerVisitor	Number of clicks per visitors
Utilization	N_absClicks	Number of clicks
Others	B_ nonRecommended	Non-recommended document type
Others	B_ unresolvedImpr	Unresolved improvement suggestions
Others	B_ unresolvedTR	Unresolved errors in TR documents

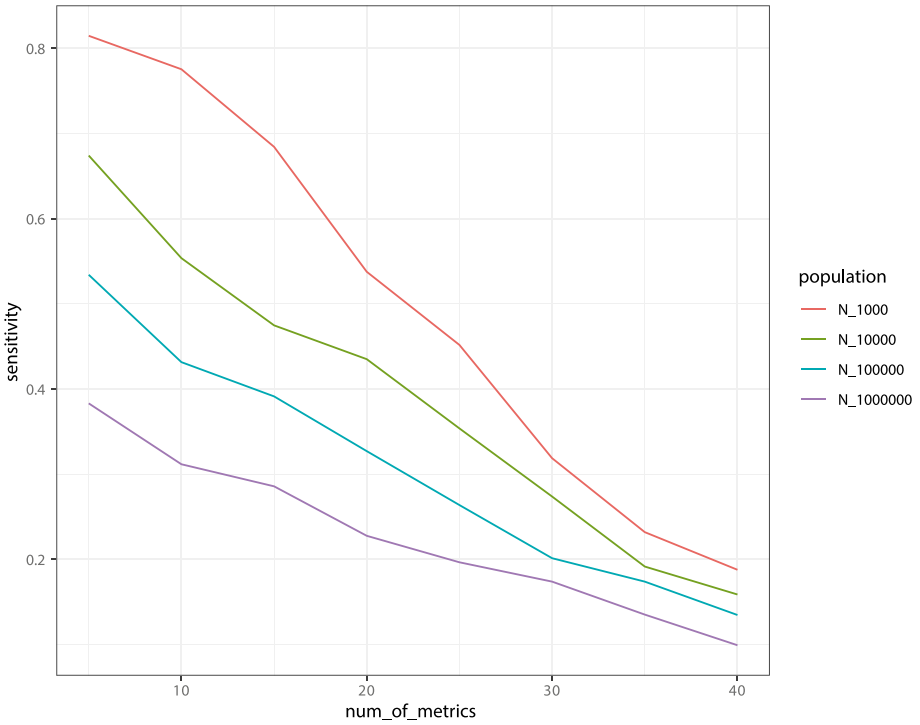


Fig. 2 Sensitivity

Both aggregations decline in sensitivity when the population grows. The two-step approach, however, is able to maintain a good level of sensitivity even for very large population sizes, cf. Fig. 3.

We find that our approach has an appropriate sensitivity even when aggregating a large number of metrics and large population sizes.

6.4 Evaluation of scalable performance R_6

Evaluation Method and Measures For performance evaluation, we use the same real-world information quality scenario as described in Sect. 6.3. Here, we assess the scalable performance, i.e., we *measure*:

Response time in milliseconds of the quality score calculation for a single artifact.

For each data point, we measured performance ten times (for ten different synthetic datasets) and take the median.

Our approach is based on a numerical computation of sample probabilities. Hence, we expect a decline in performance with growing sample population sizes. Still, while adding more metrics and artifacts, the performance should remain acceptable.

Quality Model and Dataset Description We use the same quality model and the same multivariate synthetic dataset as described in Sect. 6.3.

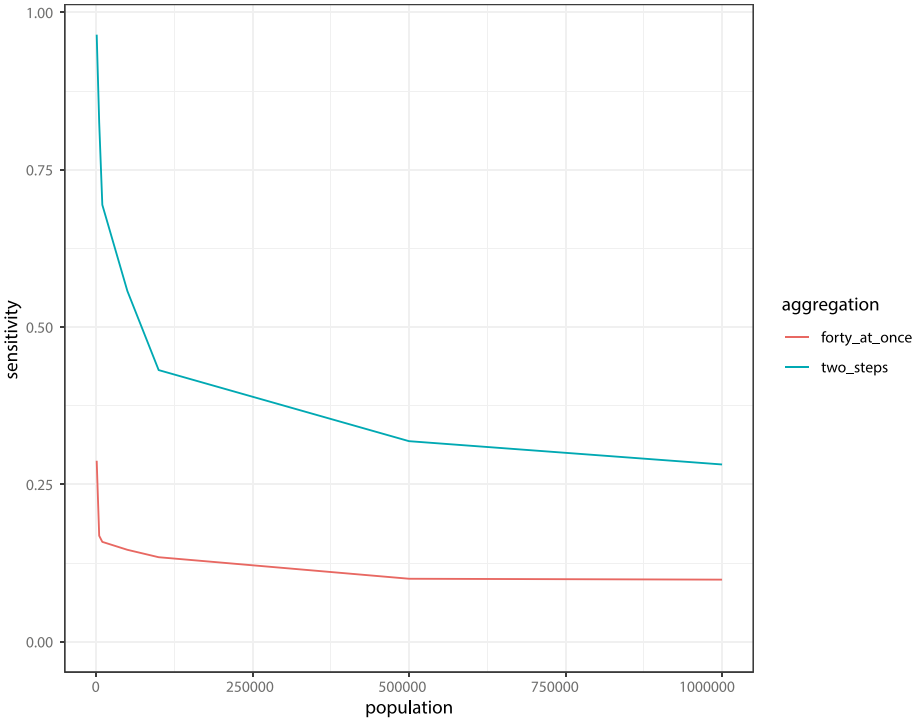


Fig. 3 Sensitivity (two-step approach versa 40 metrics at once)

Summary of Results The performance graphs, displayed in Fig. 4, show the number of metrics to aggregate and the response times. We use multi-thread computations (7 hardware threads), but even a single-threaded implementation (at most a factor of 7 slower)

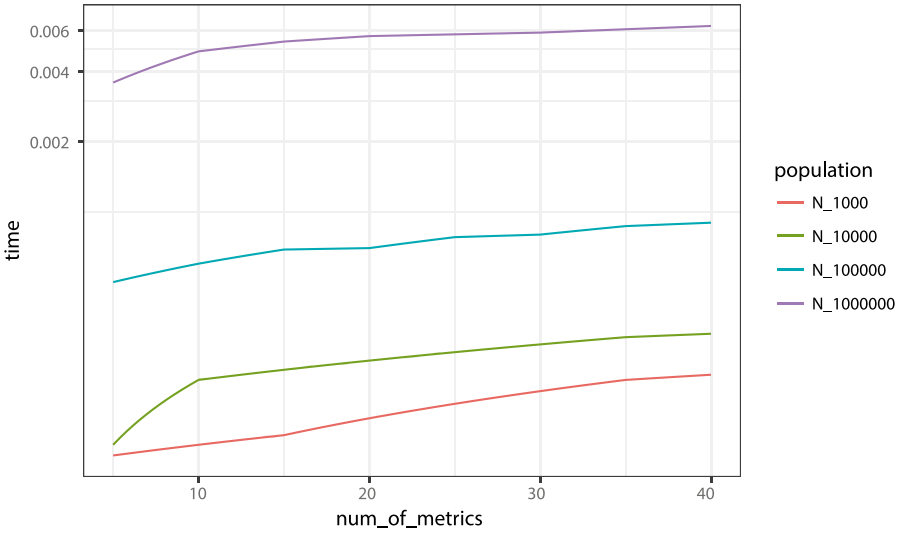


Fig. 4 Performance in *sec*.

was efficient and scalable in performance. While the response time of a score calculation increases with the number of metrics and the sample population size, as expected, it remains in the range of a few milliseconds. This is considered acceptable.

We conclude that our approach has a scalable performance when applied to large numbers of software artifacts and metrics.

6.5 Discussion

Accuracy We tested different supervised approaches to bug prediction against our aggregation trained using unsupervised learning. Given a set of metrics, the models suggest an overall ranking of potentially buggy classes. Our unsupervised approach often outperforms the rankings suggested by alternative metrics-based approaches based on generalized linear regression (supervised learning).

While it turns out that our (unsupervised) aggregation approach is quite competitive with (supervised) approaches in defect prediction, it is not our claim to replace supervised approaches, in general. However, it showed that this unsupervised aggregation might be considered as a suitable alternative when ground truth data are lacking.

Moreover, we found that our approach provides quality results that are similar to an alternative probabilistic approach, in correlation, agreement, and ranking distance. Altogether, we may claim that $\boldsymbol{R_4}$ holds.

Sensitivity Our aggregation approach outperforms the alternative probabilistic approach in terms of sensitivity, both in detecting anomalies and in differentiating the quality of software artifact.

In general, we found that the proposed aggregation is sensitive for moderate numbers of metrics, i.e., it provides sufficiently many different scores. However, for large numbers of metrics, the aggregated scores approach zero. In that case, our approach allows to aggregate metrics to sub-characteristics, and then to quality scores, which maintains a high sensitivity even for large population sizes. If needed, aggregation can be done in even more than two steps to increase sensitivity. Altogether, we may claim that $\boldsymbol{R_5}$ holds.

Performance Finally, we found that our approach admits implementations with scalable performance, i.e., $\boldsymbol{R_6}$ holds.

6.6 Threats to validity and limitations

Our experimental findings are generalizable, and we have high confidence that other researchers, given the same experimental setup, would come to the same conclusions. We discuss possible threats to validity and limitations and describe how we try to mitigate them.

External Validity. We have compared the approaches on the same open-source software systems. However, open-source and industrial development might differ greatly. We minimized this threat by also using data provided by our industrial partners when comparison was not necessary.

A second threat concerns the statistical conclusions. It is important to have a representative population to be able to interpret metrics-based assessments (Kitchenham et al., 1995). Bernoulli's *theorema aureum*, the "golden theorem," states that after sufficient many trials, the observed relative frequency (empirical probability) of an outcome does not differ from the outcome probability (theoretical probability) with a high degree of certainty (Hald, 2007).

The open-source software systems in our experiments are large enough to draw statistically meaningful conclusions regarding accuracy when a ground truth was available in the defect prediction scenario. However, they are not large enough to make general conclusions when comparing with alternative probabilistic approach.

The selection of a large dataset from technical documentation projects for information quality assessment aims experimental evaluations on representative and sufficiently large sample sizes. We generated the samples with the same metrics and the same properties (basic statistics of metrics, metrics distributions, and metrics dependencies) as observed in the real-world projects. There is no reason why the evaluation should be different in software projects and documentations implemented in different information representation languages. However, we cannot exclude this thread to validity.

Internal validity Experiments are clearly defined to make them reproducible. The *GitHub Java Corpus* and the *Bug Prediction Dataset* were chosen to replicate the proposed experiments. Metrics data extraction, collection, and analysis can be repeated yielding the same results. Metrics tool selection has an impact on the metric values and the interpretation (Lincke et al., 2008). Therefore, we assessed all software and information quality metrics with the same tool, resp., described in peer-reviewed publications. For the software metrics, the VizzAnalyzer tool and its architecture are described in (Strein et al., 2007). For technical documentation metrics, the Quality Monitor tool and its architecture are described in (Ericsson et al., 2012).

Limitations. We do not judge metrics values or scores. For instance, we do not discuss any thresholds for low, moderate, high, and very high quality, resp. Instead, metrics and quality scores are probabilities, which make aggregation results for all levels to be in the same interval [0, 1]. This mapping provides a unified understanding and allows to rank artifacts. We could, e.g., highlight "bad/good" software artifacts for every aggregation level by applying a typical approach to set quantiles as thresholds (Alves et al., 2010).

We apply a two-step aggregation approach on the quality model structure provided by our industrial partners. It improves the sensitivity compared to an aggregation of forty metrics at once. However, the two-step aggregation approach is not associative, i.e., the final result depends on the choice of grouping. This might be considered a limitation.

For our approach, we require that quality scores reveal extreme values, cf. Eq. (4), but require only (1) for $\boldsymbol{R_1}$. However, Eq. (4) makes our aggregation approach sensitive to artifacts that have extremely bad values for one metric even if they have the best values for others. We do not bother to easily spot artifacts that are extremely good in one metric. We do not aim at fining Pareto-optimal (maximal or minimal) solutions either.[25]

7 Conclusion and future work

Humans are overwhelmed with too many different aspects of artifacts, e.g., too many metrics or sub-characteristics, hence, the need for aggregation. On the other hand, humans might distrust aggregated scores that come without meaningful interpretation. For a given software artifact, our aggregated quality score corresponds to the probability of finding another artifact with worse or equal overall quality, which gives a simple and clear

[25] We could drop the requirement (4) for our approach and use copulas to estimate (non-) convex Pareto fronts (Binois et al., 2015).

interpretation. This was confirmed by our industrial partners, especially, when we visualized the quality scores. They became understandable and interpretable even for managers without strong technical or mathematical background.

Our probabilistic approach to the aggregation of metrics based on copulas meets the following requirements: It is mathematically well defined ($\boldsymbol{R_1}$), it provides a clear interpretation of scores ($\boldsymbol{R_2}$), it allows to combine different metrics with different units, scales, and distributions ($\boldsymbol{R_3}$), it is accurate ($\boldsymbol{R_4}$) and sensitive ($\boldsymbol{R_5}$), and it allows for implementations with scalable assessment performance ($\boldsymbol{R_6}$).

We proved $\boldsymbol{R_1}, \boldsymbol{R_3}$, argued for $\boldsymbol{R_2}$, and confirmed $\boldsymbol{R_4}$, $\boldsymbol{R_5}$, and $\boldsymbol{R_6}$ experimentally. Therefore, we conducted an empirical study on *Bug prediction* of ca. 5 000 software classes, on *Maintainability* assessment of ca. 15 000 open-source software systems, and on *Information Quality* assessment of ca. 100 000 real-world technical documents. These studies also showed that our approach is well applicable under realistic conditions. In short, our approach is not only theoretically sound, it is also accurate, sensitive, identifies anomalies, and scales in performance, whereas compared approaches lack one or several of these properties.

Our general aggregation approach can improve prediction models. For instance, based on the same set of metrics, our unsupervised aggregation almost always outperforms the supervised regression approaches. Moreover, our approach agrees with the alternative probabilistic approach (Bakota et al., 2011) that was specifically designed for maintainability assessment. In contrast to this probabilistic approach, our approach differentiates more artifacts, detects more outliers, and scales in performance.

From a larger perspective, the suggested approach closes a gap in reliable and reproducible quality assessment uncovered in (Ericsson et al., 2013). Due to different aggregations, the same quality model and the same metrics for assessing the same software system or technical documentation could still lead to different assessment results and even to different interpretations. Using our general, well-defined, interpretable, sensitive, and scalable aggregation approach by default would turn the quality-model- and metric-based assessment of software and information quality into a deterministic, and hence, reliable and reproducible process.

The suggested aggregation approach is subject to parameterization: The weights of the metrics can be adjusted manually to the quality goal. Alternatively, weights can also be completely derived from metrics data (Ulan et al., 2021), which would remove subjective decisions from the aggregation approach.

In this paper, we formulated several requirements for aggregation to be appropriate. These requirements include mathematical soundness, in which the "veto power" of each single metric may or may not be included. This decision, in turn, would allow or exclude certain alternative aggregation approaches, such as an aggregation based on the (weighted) average of metrics values. In the future, we plan to complement these requirements with a public benchmark suite to allow for an objective quantitative comparison of aggregation approaches. This benchmark suite can then also be used to compare our probabilistic approach to the state-of-the-art probabilistic approach of Bakota et al. (2011) against a common objective ground truth, which the present study is lacking.

While we showed that our approach has an appropriate sensitivity, the assessment measure for sensitivity was based on an unweighted distance of the normalized metrics scores. If aggregation uses weights, improved sensitivity score will use the same weights in the distance measure for defining distinguishable artifacts. On the same note, sensitivity may not tell the entire story as different values could be concentrated around the uninteresting scores close to 1 or, vice versa, around the relevant scores close to 0. In this paper, we did not study

the spread of the score distribution. However, we observe in Fig. 1a that the scores of our approach 2 are widely spread, whereas they flock around 1 for the alternative approach. Future work will consider the spread or score distributions, e.g., using the Gini index.

Also, we plan to complement our aggregation approach with the *integration* of metrics at different software component levels from method level to class, package, and system level. Moreover, we plan to extend our approach to copulas considering both dependencies and weights at the same time. Future work will also investigate how to derive the importance weights from metric observations using unsupervised and supervise machine learning approaches.

Appendix A: probabilistic nature of quality

In this appendix, we present the theoretical foundations of proposed approach described in Sect. 5. We discuss the probabilistic nature of quality and show how metrics quality scores can be expressed and aggregated.

Note that randomness is a function of what is known (Crutchfield & Feldman, 2003). Once a software system is known, metrics become constants. For an unknown software system, they are uncertain but can be described with probability distributions.

We exemplify all the theoretical concepts on one of the top 10 most popular programming languages, Java. To illustrate some basic ideas, as an example we use well-known and widely used software metric LOC. Note that it is just an example, proposed approach can be applied to any programming language and the context when aggregation of metrics makes sense.

The probabilistic nature of quality

Reasoning problems are often better understood from a probabilistic point of view (Oaksford & Chater, 2009). Quality, the degree to which software artifacts fit their use and conform to requirements, can be expressed using probabilities.

The process of developing a software system that implements a specific, well-defined functionality is a random process, since the exact outcome of the software artifacts, i.e., the system decomposition into classes and methods and their implementation, cannot be predicted with certainty. The result depends, among other things, on the non-functional requirements, constraints in the development process, and the preferences and talents of the developers.

Probability spaces can model random processes. By definition, a *probability space* is a triple (Ω, Σ, P) with

1. the *sample space* Ω, a non-empty set of possible outcomes,
2. the *events* $\Sigma \subseteq 2^{\Omega}$, a set of subsets of the sample space, and
3. the *probability measure* $P : \Sigma \to [0, 1]$ assigning probabilities to events.

The pair (Ω, Σ) is called a *measurable space*. Each execution of a random process delivers an outcome $\omega \in \Omega$. All events in Σ that contain ω are said to *have occurred*. The probability of an event E is the *relative frequency* of any of its outcomes, i.e., the ratio between the occurrence of $\omega \in E$ and the occurrence of any outcome $\omega \in \Omega$.

For sake of mathematical correctness, it should be noted that:

4. The events $\sum$ form a $_\sigma$-algebra:
   (a) $\Omega \in \Sigma$,
   (b) Σ is closed under complement,
   (c) Σ is closed under countable unions.
5. The probability measure is countably additive, $P(\emptyset) = 0$ and $P(\Omega) = 1$.

Illustration 1 The outcome of the random process of developing a software system (in Java) can be modeled as a sample space that is the set of all subsets of syntactically correct Java classes. To keep the example simple, we only consider the development of a single class and define the sample space Ω as the set of all syntactically correct Java classes. We assume that such a process is incomplete if it does not produce a syntactically correct class. Singleton sets of individual classes $\{\omega\}$ and the set all classes Ω are *events* in Σ. Because of 4b), all sets $\Omega - \{\omega\}$ for each class $_\omega$ and the empty set $\emptyset$ are in Σ, as well. It is trivial to see that the probability of the event $_\Omega$ occurring is $P(\Omega) = 1$. Recall that an event $E \in \Sigma$ occurs if the outcome $_{\omega \in E}$. Since any process execution delivers a class $\omega \in \Omega$ as an outcome, $P(\Omega) = 1$. The probability of any other event occurring, i.e., a specific class being delivered, is next to impossible to define. Therefore, we do not aim to reason about individual classes and instead choose to map the original measurable space, i.e., the sample space and the sigma-algebra, to another, more abstract measurable space using software metrics.

A function $\mu : \Omega \to O$ that maps between two measurable spaces $(\Omega, \Sigma) \to (O, S)$ is called *measurable function* if the pre-image of an event A under μ is in Σ for every $A \in S$. The function is called a *metric* if O is numerical, i.e., $O \subseteq \mathbb{R}$. For metrics, O is often a subrange of the natural numbers $\mathbb{N}$ (counts) or the rational numbers $\mathbb{Q}$ (ratios). For all events $A \in S$ of the target measurable space, the event $E = \{\omega \in \Omega : \mu(\omega) \in A\}$ must be in Σ. The notation $Pr(X \in A)$ is a commonly used shorthand for $P(E) = P(\{\omega \in \Omega : \mu(\omega) \in A\})$.

A *random variable* is a measurable function from a set of possible outcomes to a measurable space, so we can consider software metrics as random variables. In quality models, several metrics are aggregated, e.g., to sub-characteristics and characteristics. Hence, these (sub-)characteristics consisting of several metrics are multivariate random variables, or so-called *random vectors*.

Illustration 2 The number of lines of code, LOC, in a Java class ω defined in a file is the number of newline characters in this file. The function $LOC : \Omega \to \mathbb{N}^+$ is a metric since the set of positive numbers $_{\mathbb{N}^+}$ is numerical and LOC defines a mapping between the measurable spaces (Ω, Σ) and $(\mathbb{N}^+, S)$. If we want to reason about classes of the same size, we can model the set of events S to contain singleton sets $\{n\}$ for each possible number $n \in \mathbb{N}^+$ of LOC and $\mathbb{N}^+$ itself. Again, because of 4b), all sets $\mathbb{N}^+ - \{n\}$ for each possible number n of $_{LOC}$ and the empty set $\emptyset$ are in S. Finally, since the pre-image of every event A under $_{LOC}$ must also be in Σ, all sets of classes Ω_n with the same number n of LOC and their complements $\Omega - \Omega_n$ are in Σ.

A probability space (Ω, Σ, P) and a measurable function μ induce another probability space (O, S, Pr). While it might be difficult to analytically or empirically model the

probability measures P, it is easier to reason about (O, S, Pr) and to determine the probability measures Pr empirically, especially if μ is a metric.

Illustration 3 It is hard and of low relevance to determine the probability of a software development process having a certain class ω as its outcome, i.e., $P(\omega' \in \{\omega\}) = P(\omega' = \omega)$. It is easier and more meaningful to determine the probability of this process leading to a class with a certain size n, i.e., $Pr(LOC(\omega') \in \{n\}) = Pr(LOC(\omega') = n)$.

A set of metrics quantifies each quality characteristic of a software artifact. For each software artifact and metric $\mu_i \in \{\mu_1 \cdots, \mu_n\}$, we denote the possible metric value of applying the metric to the given artifact by $\mu_i(artifact)$. We use the shorthand m_i to express $\mu_i(artifact)$.

Metrics scores and aggregation

Metrics scores

The algebraic structure of the set of real numbers $\mathbb{R}$ allows to define basic statistics, e.g., *variance*, *expected value*, and *distribution* of a random variable. In general, the *probability density function* (PDF) and its corresponding *cumulative distribution function* (CDF) provide a statistical overview of the distribution of numerical (metric) values. Note that for a given random variable, there is a unique CDF but more than one valid PDF.

The *cumulative distribution function* CDF_μ of a measurable function $\mu : \Omega \to O$ is the function given by:

$$\begin{aligned} CDF_\mu &: \Omega \to [0, 1] \\ CDF_\mu(\omega) &= P(\{\omega' \in \Omega : \mu(\omega') \leq \mu(\omega)\}). \end{aligned} \tag{12}$$

If the sample space can be disregarded:

$$\begin{aligned} CDF_\mu &: O \to [0, 1] \\ CDF_\mu(m) &= Pr(\mu \leq m) = Pr(\mu \in \{o \in O : o \leq m\}) \end{aligned} \tag{13}$$

Illustration 4 The cumulative distribution function CDF_{LOC} of LOC maps each class $\omega \in \Omega$ to the probability that the outcome of the software development process described in Illustration 1 is a class $\omega' \in \Omega$ with $LOC(\omega') \leq LOC(\omega)$, cf. Eqs. (10) and (11). However, we are often only interested in $CDF_{LOC}(n)$, i.e., the probability that the outcome of this software development process is a class $\omega' \in \Omega$ with $LOC(\omega') \leq n$, cf. Eqs. (12)–(13).

We denote the *i-th* metrics value obtained for *j-th* software artifact by m_i^j, where $i \in \{1, \ldots, |metrics|\}$ and $j \in \{1, \ldots, |artifacts|\}$.

For each metric $\mu_i \in \mu$, we define a quality score S_i corresponding to the probability of finding, for a given artifact, another artifact with worse or equal quality. We denote the *i-th* score value obtained for *j-th* software artifact by s_i^j. For metric μ_i, where small values are indicators of bad quality, this is the probability of observing a μ_i value less than or equal to m_i^j. In this case, the score of m_i^j is the $CDF_{\mu_i}(m_i^j)$. For metric μ_i, where large values are indicators of bad quality, it is the probability observing a μ_i value greater than or equal

to m_i^j. In this case, the score of m_i^j is a *complementary cumulative distribution function* $CCDF_{\mu_i}(m_i^j) = 1 - CDF_{\mu_i}(m_i^j)$

This way, the interpretation of aggregation is the same on all aggregation levels, regardless of whether the aggregated values are metrics or aggregates. Recall that on all levels low scores are considered bad and large scores good.

$$s_i^j = \begin{cases} Pr(\mu_i \leq m_i^j), & \text{if low values are bad for } \mu_i \\ Pr(\mu_i \geq m_i^j), & \text{otherwise} \end{cases} \tag{14}$$

Scores are special metrics that can also be understood as random variables. We denote the random score variable S_i corresponding to the random variable μ_i from a concrete score value s_i^j corresponding to a concrete metrics value m_i^j as defined by Eq. (14).

Illustration 5 Direction of the contribution of a metric to (a certain notion of) quality depends on the quality view and the goal. For example, low degrees of documentation of public APIs are considered bad for maintainability and, hence, we use CDF for scoring LOC metric. In contrast, low values of LOC are considered good for maintainability; hence, we use CCDF for scoring.

Aggregation

Let a (sub-)characteristic c be an aggregation of metric random variables $[\mu_1, \ldots, \mu_k]$ with corresponding score random variables $[S_1, \ldots, S_k]$. An aggregated score value s_c can now be defined as joint CDF for a given vector of corresponding score values $s_1, \ldots s_k$:

$$s_c(s_1, \ldots, s_k) = CDF_{[S_1,\ldots,S_k]}(s_1, \ldots, s_k) = Pr(S_1 \leq s_1, \ldots, S_k \leq s_k) \tag{15}$$

Again, aggregated scores are special scores and, special scores are metrics and random variables.

Copula representation of a joint probability distribution allows the marginal distributions to be modeled separately from the dependence structure, i.e., dependencies are introduced by an appropriate copula function (Nelsen, 2007). Applying *Sklar's theorem* (Rüschendorf, 2009), an aggregation of scores of a (sub-)characteristic c is a specific copula function *Cop*.

$$s_c(s_1, \ldots, s_k) = Cop(s_1, \ldots, s_k) \tag{16}$$

Appendix B: R scripts

In this appendix, we present R scripts needed to implement proposed approach described in Sect. 5.

Calculating the Scores (cf. Eq. (5))

Unweighted and Weighted Aggregation (cf. Eqs. (7), (8))

```
#scoring when large values are good
scoringCDF <- function(x) {
  n<-length(x)
  score<-vector("numeric",n)
  for (i in 1:n){sc<-sum(x<=x[i])/n
score[i]<-sc}
  return(score)
}
#scoring when small values are good
scoringCCDF <- function(x) {
  n<-length(x)
  score<-vector("numeric",n)
  for (i in 1:n){sc<-sum(x>=x[i])/n
score[i]<-sc}
  return(score)
}
```

```
library(Emcdf)
options(digits = 7) #set precision
num = coreNum() - 1 #set thread number
obj = initF(scores, num) #initialize threads
Cop<-emcdf(obj, scores) #compute empirical copula of metrics scores

wCop<-function (input, w) { #input must be a matrix
M <- matrix(nrow = nrow(input), ncol = ncol(input))
    for (j in 1:ncol(input)) {
            M[, j] <- input[, j]
        }
wsc <- matrix(nrow = nrow(input), ncol = ncol(input))#calculate weighted scores
    for (j in 1:ncol(input)) {
        wsc[, j] <- M[, j]^w[j]
        }
wagg <- apply(wsc, 1, prod) #product of weighted scores
    return(wagg)
        }
```

Ranking (cf. Eq. (9))

```
Ranking<-function(x){
        ranking<-as.numeric(factor(rank(x))) #assign same ranks for the same values
        return(ranking)
}
```

Acknowledgements The authors thank Ericsson and Sigma Technology for providing real-world datasets and Softwerk for providing access to their Quality Monitor. The research was supported by The Knowledge Foundation within the project “Software technology for self-adaptive systems” (ref. number 20150088). We thank the anonymous reviewers whose comments and suggestions helped us improve and clarify the research onto paper.

Funding Open access funding provided by Linnaeus University.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Allamanis, M., & Sutton, C. (2013). Mining source code repositories at massive scale using language modeling. In *Proceedings of the 10th Working Conference on Mining Software Repositories,* IEEE Press, pp. 207–216.

Alves, T. L., Correia, J. P., & Visser, J. (2011). 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*, IEEE, pp. 20–29.

Alves, T. L., Ypma, C., & Visser, J. (2010). Deriving metric thresholds from benchmark data. In *2010 IEEE International Conference on Software Maintenance*, IEEE, pp. 1–10.

Baggen, R., Correia, J. P., Schill, K., & Visser, J. (2012). Standardized code quality benchmarking for improving software maintainability. *Software Quality Journal, 20*(2), 287–307.

Bakota, T., Hegedűs, P., Körtvélyesi, P., Ferenc, R., & Gyimóthy, T. (2011). A probabilistic software quality model. In *2011 27th IEEE International Conference on Software Maintenance (ICSM),* IEEE, pp. 243–252.

Bansiya, J., & Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering, 28*(1), 4–17.

Barkmann, H., Lincke, R., & Löwe, W. (2009). Quantitative evaluation of software quality metrics in open-source projects. In *23rd International Conference on Advanced Information Networking and Applications, AINA 2009, Workshops Proceedings, Bradford, United Kingdom, May 26-29, 2009*, pp. 1067–1072.

Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering, 22*(10), 751–761.

Binois, M., Rullière, D., & Roustant, O. (2015). On the estimation of pareto fronts from the point of view of copula theory. *Information Sciences, 324,* 270–285.

Bland, J. M., & Altman, D. (1999). Measuring agreement in method comparison studies. *Statistical Methods in Medical Research, 8*(2), 135–160.

Boehm, B. W., Brown, J. R., & Kaspar, H. (1978). Characteristics of software quality. North-Holland.

Borsboom, D., Mellenbergh, G. J., & Van Heerden, J. (2003). The theoretical status of latent variables. *Psychological Review, 110*(2), 203.

Calvo, T., Kolesárová, A., Komorníková, M., & Mesiar, R. (2002). Aggregation operators: properties, classes and construction methods. In *Aggregation operators*. Springer, pp. 3–104.

Carlsson, C., & Fullér, R. (1995). Multiple criteria decision making: the case for interdependence. *Computers and Operations Research, 22*(3), 251–260.

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering, 20*(6), 476–493.

Correia, J. P., & Visser, J. (2008). Certification of technical quality of software products. In *Proc. of the Int'l Workshop on Foundations and Techniques for Open Source Software Certification*, pp. 35–51.

Crosby, P. (1980). *Quality is free: The art of making quality certain*. Signet.

Crutchfield, J. P., & Feldman, D. P. (2003). Regularities unseen, randomness observed: Levels of entropy convergence. *Chaos: An Interdisciplinary Journal of Nonlinear Science 13*, 1, 25–54.

D'Ambros, M., Lanza, M., & Robbes, R. (2010). An extensive comparison of bug prediction approaches. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, IEEE, pp. 31–41.

Dujmović, J. (2013). Aggregation operators and observable properties of human reasoning. In *Aggregation Functions in Theory and in Practise*. Springer, pp. 5–16.

Ericsson, M., Löwe, W., Olsson, T., Toll, D., & Wingkvist, A. (2013). A study of the effect of data normalization on software and information quality assessment. In *20th Asia-Pacific Software Engineering Conf., APSEC 2013, Ratchathewi, Bangkok, Thailand, December 2-5, 2013 - Volume 2*, pp. 55–60.

Ericsson, M., Wingkvist, A., & Löwe, W. (2012). The design and implementation of a software infrastructure for iq assessment. *International Journal of Information Quality, 3*(1), 49–70.

Fenton, N. (1994). Software measurement: A necessary scientific basis. *IEEE Transactions on Software Engineering, 20*(3), 199–206.

Garvin, D. (1984). What does product quality really mean? *Sloan Management Review, 25,* 25–45.

Gil, Y., & Lalouche, G. (2017). On the correlation between size and metric validity. *Empirical Software Engineering, 22*(5), 2585–2611.

Grady, R. B., & Caswell, D. L. (1987). Software metrics: establishing a company-wide program. Prentice Hall.

Graves, T. L., Karr, A. F., Marron, J. S., & Siy, H. (2000). Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering, 26*(7), 653–661.

Hald, A. (2007). James bernoulli's law of large numbers for the binomial, 1713, and its generalization. *A History of Parametric Statistical Inference from Bernoulli to Fisher, 1713–1935,* 11–15.

Heitlager, I., Kuipers, T., & Visser, J. (2007). A practical model for measuring maintainability. In *6th international conference on the quality of information and communications technology (QUATIC 2007)*, IEEE, pp. 30–39.

Henderson-Sellers, B. (1995). *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc.

Hollander, M., & Wolfe, D. A. (1999). Nonparametric statistical methods. Wiley-Interscience.

IEEE. (1990). Ieee std 610.12-1990, standard glossary of software engineering terminology.

ISO/IEC. (2010). Iso/iec 25010 system and software quality models. Technical Report.

Jiarpakdee, J., Tantithamthavorn, C., & Treude, C. (2018). Autospearman: Automatically mitigating correlated metrics for interpreting defect models. In *Proceeding of the International Conference on Software Maintenance and Evolution (ICSME)*, pp. 92–103.

Juran, J., & Godfrey, A. B. (1999). Quality handbook. Republished McGraw-Hill, 173–178.

Kendall, M. (1948). Rank correlation methods. Griffin.

Kitchenham, B., & Pfleeger, S. L. (1996). Software quality: the elusive target [special issues section]. *IEEE Software, 13*(1), 12–21.

Kitchenham, B., Pfleeger, S. L., & Fenton, N. (1995). Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering, 21*(12), 929–944.

Letouzey, J.-L., & Coq, T. (2010). The sqale analysis model: An analysis model compliant with the representation condition for assessing the quality of software source code. In *2010 Second International Conference on Advances in System Testing and Validation Lifecycle*, IEEE, pp. 43–48.

Lincke, R., Lundberg, J., & Löwe, W. (2008). Comparing software metrics tools. In *Proc. of Int. Symp. on Software Testing and Analysis*, ISSTA '08, ACM, pp. 131–142.

Mai, J.-F., & Scherer, M. (2009). Lévy-frailty copulas. *Journal of Multivariate Analysis, 100*(7), 1567–1585.

Martin, R. C. (2002). Agile software development: principles, patterns, and practices. Prentice Hall.

McCall, J. A., Richards, P. K., & Walters, G. F. (1977). Factors in software quality. volume i. concepts and definitions of software quality. Technical Report. GENERAL ELECTRIC CO SUNNYVALE CA.

Morasca, S. (2009). A probability-based approach for measuring external attributes of software artifacts. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, IEEE Computer Society, pp. 44–55.

Mordal, K., Anquetil, N., Laval, J., Serebrenik, A., Vasilescu, B., & Ducasse, S. (2013). Software quality metrics aggregation in industry. *Journal of Software: Evolution and Process, 25*(10), 1117–1135.

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

Moser, R., Pedrycz, W., & Succi, G. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th International Conference on Software Engineering*, pp. 181–190.

Nelsen, R. B. (2007). *An introduction to copulas*. Springer Science & Business Media.

Oaksford, M., & Chater, N. (2009). Précis of bayesian rationality: The probabilistic approach to human reasoning. *Behavioral and Brain Sciences, 32*(1), 69–84.

Oliveira, P., Lima, F. P., Valente, M. T., & Serebrenik, A. (2014). Rttool: A tool for extracting relative thresholds for source code metrics. In *2014 IEEE International Conference on Software Maintenance and Evolution*, IEEE, pp. 629–632.

Oman, P., & Hagemeister, J. (1994). Construction and testing of polynomials predicting software maintainability. *Journal of Systems and Software, 24*(3), 251–266.

Riaz, M., Mendes, E., & Tempero, E. (2009). A systematic review of software maintainability prediction and metrics. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, IEEE Computer Society, pp. 367–377.

Rüschendorf, L. (2009). On the distributional transform, sklar's theorem, and the empirical copula process. *Journal of Statistical Planning and Inference, 139*(11), 3921–3927.

Serebrenik, A., & van den Brand, M. (2010). Theil index for aggregation of software metrics values. In *2010 IEEE International Conference on Software Maintenance*, IEEE, pp. 1–9.

Siegel, S., & Tukey, J. (1960). A nonparametric sum of ranks procedure for relative spread in unpaired samples. *Journal of the American Statistical Association, 55*(291), 429–445.

Simon, H. (1960). *The new science of management decision.* Harper & Brothers.

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

Strein, D., Lincke, R., Lundberg, J., & Löwe, W. (2007). An extensible meta-model for program analysis. *IEEE Transactions on Software Engineering, 9,* 592–607.

Ulan, M., Löwe, W., Ericsson, M., & Wingkvist, A. (2018). Introducing quality models based on joint probabilities. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ACM, pp. 216–217.

Ulan, M., Löwe, W., Ericsson, M., & Wingkvist, A. (2021). Weighted software quality scoring and its application to defect prediction. *Empirical Software Engineering*, Accepted for publication.

Vasa, R., Lumpe, M., Branch, P., & Nierstrasz, O. (2009). Comparative analysis of evolving software systems using the gini coefficient. In *2009 IEEE International Conference on Software Maintenance*, IEEE, pp. 179–188.

Vasilescu, B., Serebrenik, A., & Van den Brand, M. (2011). By no means: A study on aggregating software metrics. In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*, ACM, pp. 23–26.

Vincke, P. (1992). Multicriteria decision-aid. John Wiley & Sons.

Wagner, S. (2013). Software product quality control. Springer.

Wagner, S., Goeb, A., Heinemann, L., Kläs, M., Lampasona, C., Lochmann, K., et al. (2015). Operationalised product quality models and assessment: The quamoco approach. *Information and Software Technology, 62,* 101–123.

Zimmermann, T., Premraj, R., & Zeller, A. (2007). Predicting defects for eclipse. In *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*, IEEE, pp. 9–9.

Maria Ulan is a doctoral student at the Department of Computer Science and Media Technology of Linnaeus University, Sweden. Her main focus is on multi-criteria software quality scoring and software metrics aggregation.

Prof. Dr. Welf Löwe holds the chair in software technology at Linnaeus University in Växjö (Sweden) since 2002. Before, he studied at TU Dresden (Germany), received a PhD from TH Karlsruhe (Germany), was a postdoc at ICSE Berkeley (CA, USA), and assistant professor at TH Karlsruhe. He is interested in technology for software construction, analysis, optimization, and runtime support. He is also co-founder of Softwerk, Aimo, and DueDive

Dr. Morgan Ericsson is an Associate Professor of Computer Science at Linnaeus University. His main research interests include software quality and metrics, empirical software engineering, and data mining for software engineering data.

Dr. Anna Wingkvist is an Associate Professor in Computer Science at Linnaeus University, Sweden. Her academic background is in information systems development, methodological and research methods reasoning, and project management. Since completing her PhD in 2009, her scientific interest and publications are mainly in the information quality domain. In 2011, she was awarded a Marie Curie Fellowship research grant.