A Benchmarking-inspired Approach to Determine Threshold Values for Metrics

Klaus Lochmann

Technische Universität München
Software & Systems Engineering
Munich, Germany
lochmann@in.tum.de

ABSTRACT

In order to assess software quality by software metrics, usually, thresholds for metric values are needed. A common problem is to define reasonable threshold values. One possible solution is to use a benchmarking approach: the threshold value for a metric is calculated based on the metric values of a set of systems, which are called benchmarking base. A relevant question is, how the used benchmarking base influences the result of the software quality assessment. Based on the quality assessment approach of Quamoco, we conduct a series of experiments, using different benchmarking bases. For each benchmarking base a quality assessment of a series of test systems is conducted. We analyze the whether the quality assessment results of the test systems are concordant for different benchmarking bases. The main findings are: (1) The bigger the benchmarking base, the less divergent are the rankings, and the less is the variance of the results. (2) The size of the systems contained within a benchmarking base does not influence the results, i.e. a benchmarking base containing small systems works equally well for small and large systems, and vice versa. These results show that benchmarking is a feasible approach for determining threshold values.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics; D.2.9 [**Software Engineering**]: Management—*Software Quality Assurance (SQA)*

General Terms

Measurement, Experimentation

Keywords

Quality model, Metric, Benchmarking

1. INTRODUCTION

In order to define and assess the quality of software systems, quality models have been introduced. Late approaches, such as *Quamoco* [14, 22] and *Squale* [16], define quality attributes, their operationalization by metrics and tools, and sophisticated assessment and aggregation approaches. Based on measurement data they produce an overall quality statement of a software system. In order to get to this overall quality statement, the quality models describe the relation of single metrics to quality attributes. Furthermore, it is necessary to map a value obtained for a metric to a utility w.r.t. a quality attribute, i.e. evaluating if the metric value is "good" or "bad". The mapping is called a utility function.

In our approach *Quamoco* we use partwise linear functions for specifing the utility function. The decision whether the linear function must be increasing or decreasing is taken by an expert. For instance, the expert decides that the higher the values of the metric "clone coverage", the lower the result regarding maintainability must get.

The main challenge of this approach is to define the exact numerical parameters of the utility function. A common solution to this problem is to define the thresholds based on a benchmarking approach [12]. The basic principle of benchmarking is to collect a measure for a (large) number of systems (called *benchmarking base*) and compare the measured value of the system under assessment to these values. This allows us to decide if the system is better, equally good, or worse compared to the benchmarking base.

The Quamoco approach uses the benchmarking base to statistically determine parameters for the utility functions. In essence, the parameters of the linear function are chosen so that a system with a metric value lower than that of all benchmarking systems evaluates to a utility of 0, a system with a metric value higher than that of all benchmarking system evaluates to a utility of 1.

The evaluations of the Quamoco approach in [22] found that the quality assessment results matches the expectations of experts for the corresponding systems. Furthermore, the assessment is able to detect quality improvements of a software system across different versions. Nonetheless, the impact of the chosen benchmarking approach on the quality assessment results remains an open and relevant question.

Ideally, we would use all existing software systems as a benchmarking base. This way, the generalizability of the resulting quality assessment would be maximized. However, for practical reasons, we must use available samples of software systems as benchmarking bases. Obviously, different benchmarking bases, lead to different quality assessment results.

For being able to produce generalizable quality statements nonetheless, we would like to find characteristics for satisfactory benchmarking bases. Two satisfactory benchmarking bases used for assessing the same set of systems should lead to the same assessment results.

We will test the following two characteristics of benchmarking bases: (1) one characteristic that arguably could have an influence on the quality assessment results is the size of the benchmarking base (i.e. the number of systems contained in the benchmarking base); (2) another characteristic with a possible influence on the quality assessment results is the size of the systems contained in the benchmarking base. For assessing rather large software systems, benchmarking bases containing large systems could produce better results.

DOI: 10.1145/2382756.2382782 http://doi.acm.org/10.1145/2382756.2382782

Other criteria, such as successful vs. not successful systems, are out-of-scope for this article. Furthermore, our investigations are limited to open source systems written in the programming language Java.

2. RELATED WORK

To the best of our knowledge there is only one approach directly concerned with benchmarking w.r.t. software code quality. Gruber et al. [7, 6] describe an approach for assessing software quality by a benchmarking approach. They provide a case study with a comprehensive evaluation of the validity of the assessment results using their approach. However, they do not investigate the impacts of different benchmarking bases.

Similar questions to the ones investigated in this paper arise in the area of metric-based approaches for quality assessments and software fault prediction, discussed in the following.

Metric-based Quality Assessments. Lanza and Marinescu [12] describe two approaches for finding threshold values for metrics: (a) statistical information and (b) common knowledge. The former essentially describes a benchmarking approach like ours, but without taking a detailed look at the influence of different benchmarking bases. The latter refers to thresholds determined by experts, based on a well-justified explanation.

The quality model of Squale [20] mainly uses expert-based threshold values. For classical size and complexity metrics (e.g. size of methods, number of methods per class, nesting depth of methods) an expert defines a function mapping to the interval $[0, 3]$, 0 meaning a bad result and 3 a good result. For instance, the size of methods is mapped to this scale by the function $2^{\frac{70-SLOC}{21}}$. Obviously, the values 70 and 21 are based on expert knowledge about the sizes of methods. For rule-based static code analysis metrics a similar transformation is used, essentially being based on a defect density metric: $a^{\frac{w\times \text{numberofdefects}}{SLOC}}$. The constant a and the weight w are chosen by the expert. In contrast, in the Quamoco approach such parameters of utility functions are never set by experts, but only by the benchmarking approach.

Other articles (e.g. [5, 15]) mention the challenge of finding thresholds, but without making a contribution to this topic.

Fault Prediction. In software fault prediction [8] statistical models are built that capture the relation of different metrics to faults in the field. For a set of systems, for which both the metric values and the fault numbers are known, either statistical models (such as logistic regression) or machine learning models (such as neural networks) are applied. Determining utility functions based on such techniques is not possible in our quality model, because the "target quality" is not known.

However, like in our approach, also for statistical models and for machine learning models, the number of systems used for "learning" the predictor is important.

3. BACKGROUND: THE QUAMOCO APPROACH

In the research project Quamoco[1] a quality model for software products and accompanying methods for applying the quality model in practice have been developed. First, an explicit meta-model for quality models has been developed [22, 11, 14], which enables the construction of general quality models [13]. Second, based on the meta-model a "base model" [17] has been developed. It describes abstract quality attributes and operationalizes them by concrete measures and tools for Java source code. For constructing quality models and for conducting quality assessments, tool support was built [2]. Using the tool-support, several evaluations were conducted [14, 22]: The evaluation of the Quamoco approach showed that the quality assessment (1) is able to diversify software systems regarding different quality levels, (2) is in line with expert-based assessments (3) reflects quality improvements over different versions of a software product.

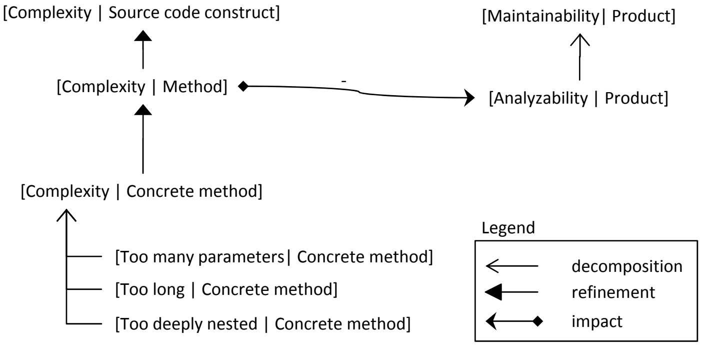

[Complexity | Source code construct]
[Maintainability| Product]
[Complexity | Method]
-
[Analyzability | Product]
[Complexity | Concrete method]
[Too many parameters| Concrete method]
[Too long | Concrete method]
[Too deeply nested | Concrete method]
Legend
decomposition
refinement
impact

Figure 1: Excerpt of the Quality Model

3.1 Structure of the Quality Model

As suggested in literature (see [4, 3]) the Quamoco quality model uses a product model of software as a backbone. The product model describes *Entities* of which software consists. The entities are organized like data models, with a specialization (*is-a*) and a decomposition relation (*part-of*). Typical entities include *Class* and *Expression*, whereby Expression is further refined by *Relational expression* and *Arithmetical expression*, which are in an *is-a* relation to Expression.

The main goal of the quality model is to describe properties of software. In the literature, the term property is defined as an attribute, which is used to characterize an object. Hence, in our quality model, we define a *Property* as an *Attribute* of an *Entity*. In the following we use the notation $[attribute|entity]$ for properties.

Typical quality attributes, like known from ISO 25010 [9], are expressed as properties referring to the entity *product*. For instance $[Maintainability|Product]$ and $[Reliability|Product]$. We also describe properties of more specific entities, such as $[Complexity|Method]$, or $[Correctness|Arithmetical\ expression]$.

The *Impact* relation plays a central role in the quality model. It describes a qualitative interrelation between properties. More precisely, an impact specifies that the degree to which an entity has a property influences the degree to which another entity has another property. The effect of the influence can be either positive or negative. If the impact has a positive effect, the degree to which the entity has the target property is increased if the entity has the source property; and vice-versa for a negative impact. Impacts are used to describe how specific properties (such as $[Complexity|Method]$) influence general quality attributes (such as $[Maintainability|Product]$).

[1] http://www.quamoco.de/

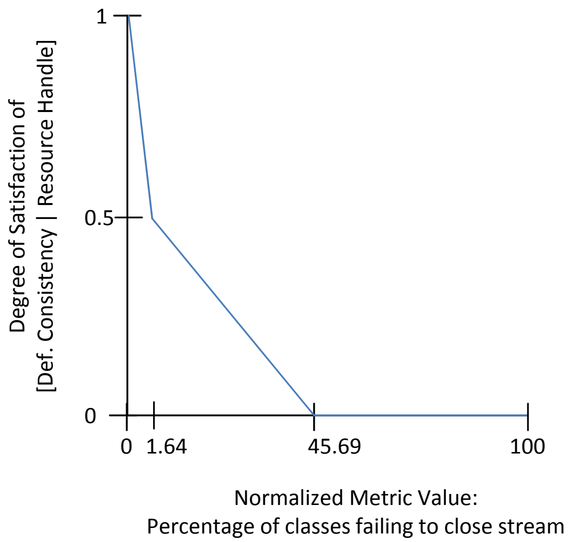


Figure 2: Utility Function for a Rule-Based Metric

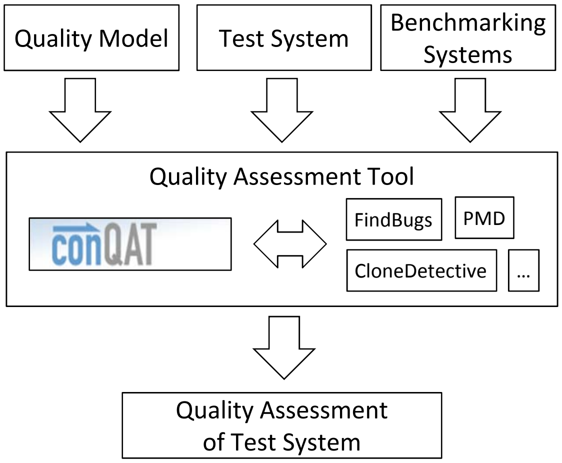


Figure 3: Quality Assessment Tool

Quality models built this way usually consist of a large number of properties characterizing specific entities. These properties are organized hierarchically using two relations:

1. If a property B *refines* a property A, it means that B is more special regarding the entity it characterizes than A. More precisely, if a property B refines a property A, then the entity of B must be in a *is-a* relation to the entity of A.
2. A property can be *decomposed* into sub-properties by specializing its attribute, while the entity stays the same.

Figure 1 shows an excerpt of the quality model for Java. It defines the property [*Complexity*|*Source code construct*], describing that there is a general notion of complexity in source code. This property is refined for more special entities, such as methods and classes. In object-oriented programming languages methods can be further split into abstract and concrete methods. The complexity of concrete methods is than decomposed to more tangible properties describing the number of parameters, the length, and the nesting depth of a method. The property [*Complexity*|*Method*] has a negative influence on the analyzability, because complex methods demand more effort for being understood by the developer.

3.2 Quality Assessment Approach

Assessing the quality means determining the degree of satisfaction of properties in the quality model. The satisfaction of each property is evaluated on a scale $[0, 1]$, 0 meaning that the property is not satisfied, and 1 meaning that the property is fully satisfied.

The principle of the quality assessment approach is to measure leaf-properties by metrics and then aggregate the values alongside the property hierarchy. Introducing all the details of the aggregation is out of scope, and can be found in [14]. In the following, we will only introduce the details regarding the utility functions and threshold values for metrics.

The main challenge is to derive a value for the satisfaction of a property by the measurement data of a metric. We will show by an example how this is done. The normalized metric value for the FindBugs rule *OS_OPEN_STREAM* is "percentage of classes, which open, but do not close a stream". This metric is used to measure the property [*Definition and Usage Consistency*|*Resource Handle*]. To calculate the degree of satifaction for this property by the given metric, the mapping function shown in Figure 2 is used. The property is fully satisfied if the metric is 0, it gets a satisfaction of 0.5 if 1.64% of the classes have the given problem, and it is not satisfied if 45.69% or more of the classes have the given problem.

The three threshold values t_1, t_2, and t_3 for the utility function (0, 1.64 and 45.69 in the example) have been determined using the benchmarking approach. For each system of the benchmarking base the normalized metric value has been calculated, resulting in values $x_1, \ldots, x_n$. The values of the thresholds are then the minimum, median, and maximum of the non-outlier values:

$$\begin{aligned} t_1 &= \mathit{min}(\{x : x \geq Q_{25\%}(x_1, \ldots, x_n) - 1.5 \cdot IQR(x_1, \ldots, x_n)\}) \\ t_2 &= \mathit{median}(x_1, \ldots, x_n) \\ t_3 &= \mathit{max}(\{x : x \leq Q_{75\%}(x_1, \ldots, x_n) + 1.5 \cdot IQR(x_1, \ldots, x_n)\}) \end{aligned}$$

whereby

Q_p denotes the p-percentice
$IQR(x_1, \ldots, x_n)$ denotes the inter-quartile-range:
$IQR(x_1, \ldots, x_n) = Q_{75\%}(x_1, \ldots, x_n) - Q_{25\%}(x_1, \ldots, x_n)$

These functions assure that outlier values are ignored; for t_1 the minimum non-outliner value is taken; for t_3 the maximum non-outlier value. For t_2 the median of all values is taken.

3.3 Tool Support

For creating quality models conforming to the meta-model a *quality model editor* has been developed. This editor enables the graphical creation and modification of quality models. Based on the quality assessment framework ConQAT[2] a *quality assessment tool* for automatically conducting quality assessments based on the quality model is provided. The tools and the Java quality model are available online[3].

Figure 3 summarizes the usage of the tools for conducting a quality assessment based on a given quality model and a given benchmarking base. The quality assessment tool reads a quality model, sets its threshold values using the benchmarking systems, and

[2] `http://www.conqat.org/`
[3] `http://www4.in.tum.de/~lochmann/WoSQ2012/`

DOI: 10.1145/2382756.2382782

http://doi.acm.org/10.1145/2382756.2382782

then calculates the quality assessment of the given test system. As a result, the degree of satisfaction of all properties of the quality model are reported. For the case study in this paper, only the aggregated value for the root-quality attribute $[Quality|Product]$ is used. Additionally, ConQAT generates a detailed log file containing warning and error messages for debugging.

4. STUDY DESIGN & PROCEDURE

In this study we conduct quality assessments for a set of test systems using different benchmarking bases. For the analyses in this paper, the value aggregated to the root quality attribute $[Quality|Product]$ will be used as quality assessment result. The goal is to investigate the influence of different benchmarking bases on the quality assessment results.

4.1 Research Questions

The study is aligned according to the following three research questions:

RQ 1: Do different benchmarking bases of the same size have an influence on the quality assessment result?
We use different benchmarking bases containing the same number of systems for conducting quality assessments of the test systems. We investigate the influence of the benchmarking bases on the assessment results.

RQ 2: Does the size of the benchmarking base have an influence on the quality assessment results?
We investigate, whether the number of systems contained in a benchmarking base has an influence on the quality assessment results of the test systems.

RQ 3: Does the size of the systems used in the benchmarking base have an influence on the quality assessment results?
A reasonable supposition is that a benchmarking base containing small systems (large systems) is best suited for quality assessments of small system (large systems). We will investigate whether this assumption holds.

4.2 Study Procedure

The investigation of all three research questions involves the comparison of the quality assessment results produced by different benchmarking bases for their accordance. To derive criteria for accordance, we discuss ways in which the quality assessment results may get distorted:

- The ranking of the systems according to the overall quality $[Quality|Product]$ may change. For instance, for benchmarking base A, system X may get a better result than system Y, for benchmarking base B, system Y may get a better result than system X. Note that the ranking of the systems is independet from the absolute values the systems get.
- The absolute values of the quality assessments may be shifted by a certain offset, while their ranking remains the same. For instance, for benchmarking base A the system X and Y may get values x_A and y_A, with $x_A < y_A$, for benchmarking base B they get x_B and y_B with $x_B < y_B$, but with $x_A \neq x_B$ and $y_A \neq y_B$.

We use the following two criteria for measuring these distortions:

C1: Ranking. For two benchmarking bases A and B, we calculate the ranking of a set of test systems. Then, we calculate the Spearmans rank correlation coefficient [19] to assess, whether the rankings are conform or not.

C2: Variance. For a system X and for benchmarking bases $A_1, \ldots, A_n$, we get values $x_1, \ldots, x_n$. We calculate the standard deviation $stdev(x_1, \ldots, x_n)$ to assess how much the values of one system differ for the used benchmarking bases.

4.2.1 RQ 1 Different Systems

We do a quality assessment for all test systems, with 10 randomly generated benchmarking bases, containing 500 systems (10-100 kLOC) each.

We calculate the criterion *C1 Ranking* for each pair of benchmarking bases. We consider the influence of the benchmarking base not significant, if all pairs of rankings have a statistically significant correlation of more than 0.9.

Furthermore, we calculate the criterion *C2 Variance* for all test systems. Then, we calculate the minimum, average, and maximum standard deviation. We consider the influence of the benchmarking base not significant, if the maximum standard deviation is below 0.1.

4.2.2 RQ 2 Different Benchmarking Base Sizes

We will use benchmarking bases containing different numbers of systems, namely 10, 20, 50, 100, 200, 500. For each of these size classes we randomly generate 5 benchmarking bases out of all available Java systems between 10 and 100 kLOC. We analyze all available test systems with each benchmarking base.

First, within each size class, we calculate criterion *C1 Ranking* for each pair of rankings. This results in 10 rank correlations for each of the 6 size classes. We visualize the interrelation of the benchmarking bases' size and the conformance of the rankings through a scatterplot. In order to do a statistical test, we formulate the following hypothesis:
H_{1_0}: There is no correlation between the size of the benchmarking base and the conformance of rankings.
H_{1_A}: There is a correlation between the size of the benchmarking base and the conformance of rankings.
The hypothesis H_1 is tested by the Pearson's product-moment correlation with a significance level of $\alpha = 0.95$, i.e. H_{1_0} is rejected for a p-value less than 0.05.

Second, within each size class, we calculate criterion *C2 Variance* for each test system. Due to the high number of values (a value for each test system per benchmarking base) a scatterplot is not suited for visualizing the interrelation of benchmarking bases' size and variance. We, thus, plot a line diagram with the minimum, median, and maximum standard deviation for each size class. In order to do a statistical test, we formulate the following hypothesis:
H_{2_0}: There is no correlation between the size of the benchmarking base and the average variance of the results.
H_{2_A}: There is a correlation between the size of the benchmarking base and the average variance of the results.
The hypothesis H_2 is tested by the Pearson's product-moment correlation with a significance level of $\alpha = 0.95$, i.e. H_{2_0} is rejected for a p-value less than 0.05.

4.2.3 RQ 3 Different System Sizes

We use benchmarking bases containing systems of different size classes: the first class contains 100 systems of sizes between 5 and 10 kLoC, the second one contains 100 systems of sizes between 50 and 150 kLoC. Of each class of benchmarking base, 10 actual benchmarking bases are generated randomly. The two criteria C1 and C2 are evaluated for two sets of test systems. The first set

DOI: 10.1145/2382756.2382782 http://doi.acm.org/10.1145/2382756.2382782

of test systems contains 150 systems between 5 and 10 kLoc, the second contains 150 systems between 50 and 150 kLoc. We compare the average rank correlation coefficient (C1) and the average standard deviation (C2) for each combination of benchmarking base class and test system set. This way, we assess whether the benchmarking base consisting of small systems works better for the small than for the large test systems; and vice versa for the benchmarking base consisting of large systems. Due to the low number of data points, a statistical test of the correlation between systems' sizes and the two criteria is not possible.

5. STUDY OBJECTS

In this section, we describe the used quality model and the Java systems used as benchmarking base and test systems.

5.1 The Quality Model

The used quality model for Java source code is, of course, an essential part of the study. We already described the structure, the quality assessment approach and the tool support in Section 3. Here, we will give more details about the contents of the Java quality model.

The quality model describes the source code quality of programs written in Java. It is based on rules of the static code checkers *FindBugs* and *PMD*. Each rule is modeled as a metric for measuring a certain property, resulting in 369 such metrics. Additionally, 9 classical software metrics (such as *clone coverage*, *number of parameters per method*) calculated by *ConQAT* are part of the model. The property hierarchy consists of 566 properties. Additionally, 14 properties are used to describe the quality attributes of ISO 25010. The quality attributes are targeted by 370 impacts, which originate from 144 properties.

Since the quality model is based on source code properties, not all quality attributes are covered equally. Only five top-level quality attributes (maintainability, functional suitability, reliability, and performance efficiency) have a high coverage by impacts. Some quality attributes, such as usability, are not associated with any measure. It is comprehensible that maintainability and reliability mostly rely on souce code properties, whereas usability cannot be assessed by looking at the source code.

5.2 Java Systems

For conducting experiments with different benchmarking bases and test systems it is important to have a large number of Java systems available. We used the repository SDS [21, 10], containing about 18.000 open-source Java projects. These projects have been mostly retrieved from open source databases such as *Sourceforge* through a web-crawling approach. In essence, this repository contains mirrors of the version control repositories of the before-mentioned databases. Such repositories usually contain not only the current version of a software, but also *branches* and *tags*. Thus, we used a heuristic to identify the directory containing the current version.

The SDS repository only contains the source code, not the binaries. For the quality assessment however, binaries compiled with the debug-option of the Java compiler are needed. We compiled all projects in a batch approach, because the effort to manually configure and compile them is prohibitive. The compilation of all 18.000 projects took about 30 hours, executed in parallel on 12 personal computers. Of all available projects about 6.000 were compiled successfully. Others could not be compiled because of missing external libraries or because of code needed to be generated during the build process.

We use all successfully compiled systems of the SDS repository larger than 5.000 LoC, which are 2041 systems. We exclude smaller systems, because many open source repositories contain software projects initiated by single persons without finishing them; these projects then remain in the repository without ever being used [1, 18]. In our experience, these unused systems are mostly smaller than 5.000 LoC.

6. STUDY RESULTS

In the following we present the study results organized according to the research questions. The raw data of the quality analyses with the different benchmarking bases can be found online[4].

6.1 RQ 1 Different Systems

Table 1 shows the correlation coefficients of the rankings for all pairs of benchmarking bases. The minimal correlation between two pairs is 0.9912 for BB_4 and BB_8. Thus, all values are above 0.9 and the criterion defined in the study design is satisfied.

For each test system we calculated the standard deviation between the results of the different benchmarking bases. The minimal standard deviation was 0.0002, the average 0.0061, and the maximum 0.0235. Since the maximum is below 0.1, the criterion in the study design is satisfied.

6.2 RQ 2 Different Benchmarking Base Sizes

For each of the six size classes we calculated the rank correlations of all pairs of benchmarking bases. Table 2 exemplarily shows the correlation coefficients for the size classes 10 and 500.

Figure 4a shows the scatter-plot of benchmarking bases' sizes and correlation coefficients. The visual impression of the scatterplot suggest a positive correlation between the two plotted variables. This suggestion is statistically tested by hypothesis H_1: We calculate the pearsons correlation coefficient for the values. It results in a correlation coefficient of 0.87 and a p-value $< 2.2E$-16. Thus, hypothesis H_{1_0} is rejected in favor of H_{1_A}: There is a statistically significant positive correlation between the size of the benchmarking base and the rank correlation.

Because of Figure 4a we assume an exponential relation between these two variables. Thus, we added the exponential regression fitting line, and calculate an exponential correlation, resulting in a correlation coefficient of 0.91 and a p-value $< 2.2E$-16.

Figure 4b shows a line diagram of benchmarking bases' sizes and the standard deviation. The visual impression of the scatterplot suggest a negative correlation between the two plotted variables. This suggestion is statistically tested by hypothesis H_2: We calculate the pearsons correlation coefficient for the values. It results in a correlation coefficient of -0.38 and a p-value $< 2.2E$-16. Thus, hypothesis H_{2_0} is rejected in favor of H_{2_A}: There is a statistically significant negative correlation between the size of the benchmarking base and the standard deviation.

Because of Figure 4b we assume an exponential relation between these two variables. Thus, we added the exponential regression fitting line, and calculate an exponential correlation, resulting in a correlation coefficient of -0.49 and a p-value $< 2.2E$-16.

6.3 RQ 3 Different System Sizes

Table 3a shows the average rank correlation coefficients for benchmarking bases containing systems of different sizes that have been

[4] `http://www4.in.tum.de/~lochmann/WoSQ2012/`

DOI: 10.1145/2382756.2382782 http://doi.acm.org/10.1145/2382756.2382782

	BB_1	BB_2	BB_3	BB_4	BB_5	BB_6	BB_7	BB_8	BB_9	BB_{10}
BB_1	1	0,9985	0,9967	0,9926	0,9974	0,9969	0,9980	0,9982	0,9963	0,9965
BB_2		1	0,9976	0,9938	0,9981	0,9980	0,9986	0,9974	0,9968	0,9976
BB_3			1	0,9947	0,9968	0,9971	0,9975	0,9962	0,9977	0,9966
BB_4				1	0,9929	0,9922	0,9933	0,9912	0,9942	0,9923
BB_5					1	0,9977	0,9983	0,9966	0,9956	0,9978
BB_6						1	0,9979	0,9971	0,9965	0,9978
BB_7							1	0,9971	0,9963	0,9978
BB_8								1	0,9961	0,9962
BB_9									1	0,9955
BB_{10}										1

Table 1: RQ 1 – Rank Correlations[5] of Different Benchmarking Bases of the Same Size

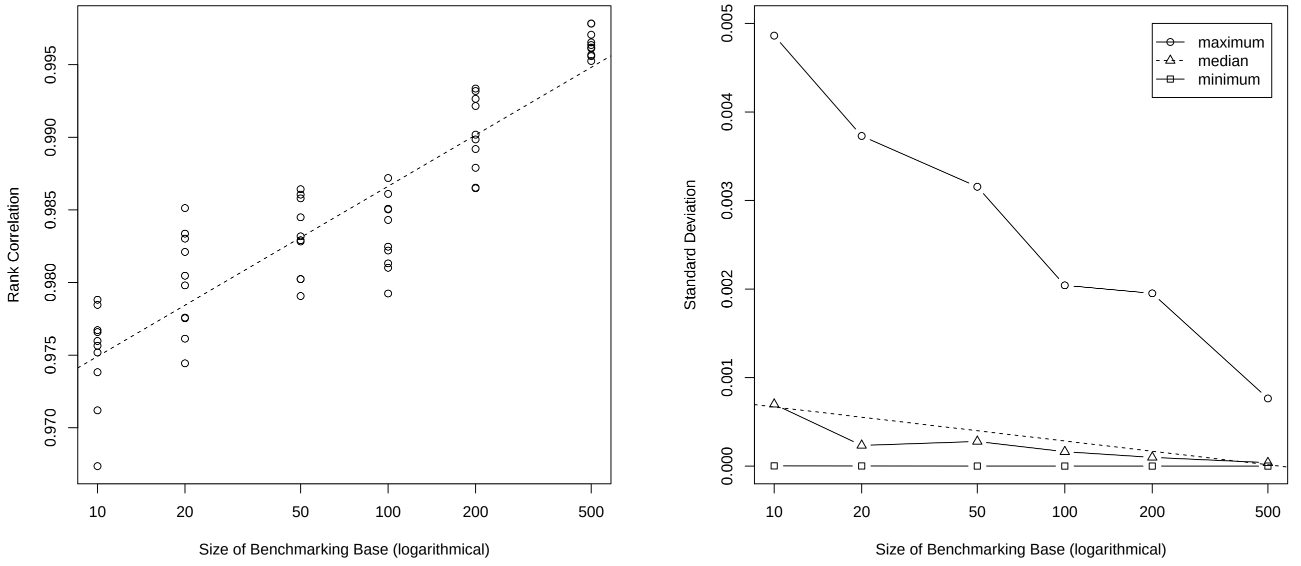
	BB_1	BB_2	BB_3	BB_4	BB_5
BB_1	1	0,9788	0,9712	0,9767	0,9785
BB_2		1	0,9752	0,9756	0,9766
BB_3			1	0,9760	0,9674
BB_4				1	0,9738
BB_5					1

(a) Benchmarking Bases of Size 10

	BB_1	BB_2	BB_3	BB_4	BB_5
BB_1	1	0,9971	0,9965	0,9962	0,9961
BB_2		1	0,9957	0,9978	0,9964
BB_3			1	0,9956	0,9953
BB_4				1	0,9978
BB_5					1

(b) Benchmarking Bases of Size 500

Table 2: RQ 2 – Rank Correlations[5] of Different Benchmarking Bases



(a) Size of Benchmark Base versus Rank Correlation (b) Size of Benchmark Base versus Standard Deviation

Figure 4: RQ 2 – Influence of the Size of the Benchmarking Base[6]

		Test Systems	
		small	large
Systems in Benchmarking Base	small	0.974	0.967
	large	0.994	0.992

(a) Average Rank Correlation

		Test Systems	
		small	large
Systems in Benchmarking Base	small	0.013	0.017
	large	0.006	0.010

(b) Average Standard Deviation

Table 3: RQ 3 – Benchmarking Bases containing Systems of Different Sizes

[5] The values in the table are the correlation coefficients. The p-value for all these correlations is far below 0.05.

[6] The dashed lines represent an exponential regression fitting.

applied to test systems of different sizes. We can see that the benchmarking base containing large systems got better rank correlations for both small and large test systems. The benchmarking base containing small systems got a lower correlation for both small and large test systems.

Table 3b shows the average standard deviation for benchmarking bases containing systems of different sizes that have been applied to test systems of different sizes. Like for the correlations, also here we can see that benchmarking bases containing large systems achieved the better results (i.e. its results have lower standard deviation) than the benchmarking base containing small systems.

7. THREATS TO VALIDITY

Construct Validity. The criteria of ranking correlation and standard deviation may not adequatly represent the matching between different results of quality assessments.

Internal Validity. The creation of random benchmarking bases may be biased. Each benchmarking base was constructed by randomly selecting systems from all available Java systems. Thus, some systems may be contained in more than one of the generated benchmarking bases. However, the number of available systems (2041) is far larger than the benchmarking bases (from 10 to 500 systems). Thus, the overlapping between randomly generated benchmarking bases should not be significant.

Between the systems contained in the benchmarking base and the test systems there may be overlappings. This may have an influence on the results. However, we deliberately selected the benchmarking base and the test systems in this way, because it reflects the approach that would likely be taken in an industrial setting: the systems developed by a company would likely be included in the benchmarking base while being test systems at the same time.

Regarding RQ 3, the benchmarking bases containing the large systems contain more source code in terms of LoC than the benchmarking bases containing small systems. This could have lead to the better results of the benchmarking bases containing larger systems.

External Validity. The threat regarding generalizability of the results is twofold: First, only open source systems have been used as study objects. It is unknown if the results also apply for software products originating from other domains. Second, we only used one single quality model for Java source code. Whether these results are valid for other quality models is unclear. However, for quality models using static code analysis, we assume good generalizability, because of the large extent of the quality model.

8. CONCLUSION

When applying software metrics for quality assessments, utility functions are needed. Defining adequate parameters for utility functions is challenging. In the Quamoco quality assessment approach we used a benchmarking-inspired paradigm to automatically determine parameters based on a set of comparison systems, called benchmarking base.

It is a relevant research question how the selection of systems in the benchmarking base influences the result of quality assessments, and how the benchmarking base should be chosen to obtain valid and comparable quality assessment results.

To investigate these research questions we conducted a series of experiments with randomly generated benchmarking bases of different sizes and containing systems of different size. The main findings of these experiments are:

1. The size of the benchmarking base matters: the bigger the benchmarking base, the smaller is the influence of the randomly chosen systems. There is a statistically significant correlation between the size of the benchmarking base and both the rank correlation and the variance of the quality assessment results. For a smaller benchmarking base, the rank correlation is lower and the standard deviation higher, than for a larger benchmarking base.

2. Our supposition that a benchmarking base containing small (large) systems works best for assessing small (large) systems, was not confirmed. Surprisingly, a benchmarking base containing large systems produced better result for both assessing large and small systems, than a benchmarking base containing small systems.

Based on these findings, we conclude that defining parameters for utility functions based on a benchmarking approach is feasible. Furthermore, we found evidence that the single most important factor for a satisfactory benchmarking base is its size. For a randomly generated benchmarking base of sufficient size, neither the actually selected systems, nor the size of the systems contained in it, has a major influence.

For further research there are still several interesting questions. First, the minimum size for a reliable benchmarking base could be determined. Second, different criteria for systems contained in the benchmarking base could be investigated. While in this article only the size of the systems was considered, also the influence of the application domain (business information systems vs. embedded systems) could be investigated.

9. REFERENCES

[1] K. Beecher, A. Capiluppi, and C. Boldyreff. Identifying exogenous drivers and evolutionary stages in FLOSS projects. *Journal of Systems and Software*, 82(5):739–750, 2009.

[2] F. Deissenboeck, L. Heinemann, M. Herrmannsdoerfer, K. Lochmann, and S. Wagner. The Quamoco Tool Chain for Quality Modeling and Assessment. In Proc. of the *International Conference on Software Engineering (ICSE '11)*. ACM, May 2011.

[3] F. Deissenboeck, Stefan Wagner, M. Pizka, S. Teuchert, and J.-F. Girard. An Activity-Based Quality Model for Maintainability. In Proc. of the *International Conference on Software Maintenance (ICSM '07)*. IEEE Computer Society, October 2007.

[4] R. G. Dromey. A model for software product quality. *IEEE Transactions on Software Engineering*, 21(2):146–162, 1995.

[5] R. Fourati, N. Bouassida, and H. Abdallah. A Metric-Based Approach for Anti-pattern Detection in UML Designs. *Computer And Information Science*, 2011(364):17–33, 2011.

[6] H. Gruber. Benchmarking-oriented Assessment of Source Code Quality, PhD thesis, Johannes Kepler Universität. 2010.

DOI: 10.1145/2382756.2382782 http://doi.acm.org/10.1145/2382756.2382782

[7] H. Gruber, R. Plösch, and M. Saft. On the validity of benchmarking for evaluating code quality. In Proc. of the *International Conferences on Software Measurement (IWSM/MetriKon/Mensura '10)*. Shaker, 2010.

[8] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A Systematic Review of Fault Prediction Performance in Software Engineering. *IEEE Transactions on Software Engineering*, PP(99):1, 2011.

[9] ISO. 25010, Systems and software engineering – System and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

[10] Joel Ossher, Sushil Bajracharya, Erik Linstead, Pierre Baldi, and Cristina Lopes. SourcererDB: An aggregated repository of statically analyzed and cross-linked open source Java projects. In Proc. of the *International Working Conference on Mining Software Repositories (MSR '09)*. IEEE Computer Society, June 2009.

[11] M. Kläs, C. Lampasona, S. Nunnenmacher, S. Wagner, M. Herrmannsdoerfer, and K. Lochmann. How to Evaluate Meta-Models for Software Quality? In Proc. of the *International Conferences on Software Measurement (IWSM/MetriKon/Mensura '10)*. Shaker, 2010.

[12] M. Lanza and R. Marinescu. *Object-oriented metrics in practice: Using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer, Berlin u.a, 2006.

[13] K. Lochmann and A. Goeb. A Unifying Model for Software Quality. In Proc. of the *International Workshop on Software Quality (WoSQ '11)*. ACM, September 2011.

[14] K. Lochmann and L. Heinemann. Integrating Quality Models and Static Analysis for Comprehensive Quality Assessment. In Proc. of the *International Workshop on Emerging Trends in Software Metrics (WETSoM '11)*. ACM, May 2011.

[15] R. Marinescu. Detection strategies: metrics-based rules for detecting design flaws. In Proc. of the *International Conference on Software Maintenance (ICSM '04)*. IEEE Computer Society, September 2004.

[16] K. Mordal-Manet, F. Balmas, S. Denier, S. Ducasse, H. Wertz, J. Laval, F. Bellingard, and P. Vaillergues. The Squale Model – A Practice-based Industrial Quality Model. In Proc. of the *International Conference on Software Maintenance (ICSM '09)*. IEEE Computer Society, September 2009.

[17] Quamoco Base Model. `http://www.quamoco.de/tools`, last accessed on 11.07.2012, 2012.

[18] A. Rainer and S. Gale. Evaluating the Quality and Quantity of Data on Open Source Software Projects. In Proc. of the *International Conference on Open Source Systems (ICOSS '05)*. UHRA, June 2005.

[19] C. Spearman. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology*, 15(1):72–101, 1904.

[20] Squale Project. The Squale Quality Model, `http://www.squale.org/quality-models-site/research-deliverables/WP1.3_Practices-in-the-Squale-Quality-Model_v2.pdf`, last accessed on 11.07.2012. 2010.

[21] Sushil Bajracharya, Joel Ossher, and Cristina Lopes. Sourcerer: An internet-scale software repository. In Proc. of the *Workshop on Search-Driven Development–Users, Infrastructure, Tools and Evaluation (SUITE '09)*. IEEE Computer Society, June 2009.

[22] S. Wagner, K. Lochmann, L. Heinemann, M. Kläs, A. Trendowicz, R. Plösch, A. Seidl, A. Goeb, and J. Streit. The Quamoco Product Quality Modelling and Assessment Approach. In Proc. of the *International Conference on Software Engineering (ICSE '12)*. ACM, June 2012.

DOI: 10.1145/2382756.2382782

http://doi.acm.org/10.1145/2382756.2382782