Reusability Index: A Measure for Assessing Software Assets Reusability

Apostolos Ampatzoglou, Stamatia Bibi, Alexander Chatzigeorgiou, Paris Avgeriou and Ioannis Stamelos

Dept. of Informatics, Aristotle University of Thessaloniki, Greece
Dept. of Informatics & Telecommunications, University of Western Macedonia, Greece
Dept. of Applied Informatics, University of Macedonia, Greece
Dept. of Mathematics and Computer Science, University of Groningen, Netherlands

apostolos.ampatzoglou@gmail.com, sbibi@uowm.gr, achat@uom.gr, paris@cs.rug.nl, stamelos@csd.auth.gr

Abstract. The reusability of assets is usually measured through reusability indices. However, these indices either do not synthesize their constituent metrics into an aggregate or they do not capture all facets of reusability, such as structural characteristics, external qualities, and their documentation. To alleviate these shortcomings, we introduce a reusability index (REI) as a synthesis of various software metrics that cover a number of related reusability aspects. Furthermore, we evaluate its ability to quantify reuse, by comparing it to existing indices through a case study on 15 reusable open-source assets (i.e., libraries and frameworks). The results of the study suggest that the proposed index presents the highest predictive and discriminative power, it is the most consistent in ranking reusable assets, and the most strongly correlated to their levels of reuse.

Keywords: reusability; quality model; metrics; validation

1 Introduction

Assessing the reusability of a software asset (i.e., the degree to which it can be reused in other systems) is an important step towards successfully applying reuse practices. To this end, a wide range of reusability models have been proposed [4]; these models usually determine high level quality attributes affecting reusability and each such attribute is quantified by a set of metrics. However, existing reusability models (see Section 2 for a detailed description) suffer from either of two limitations: (a) they only deal with the quality attributes that affect reusability (e.g., coupling and cohesion, etc.) and not reusability per se, i.e. they do not provide an aggregated reusability measure or index, or (b) they only consider structural aspects of the software asset, ignoring aspects such as documentation, external quality, etc.

In this study we propose Reusability Index (REI), an index that overcomes said limitations by: synthesizing various metrics that influence reusability—related to limitation (a); and considering multiple aspects of reusability, such as structural quality, external quality, documentation, availability, etc.—related to limitation (b). In particular, REI is calculated by synthesizing seven metrics that correspond to both structural (e.g., complexity, maintainability, etc.) and non-structural (e.g., documentation, bugs, etc.) quality characteristics. To validate the accuracy of the developed index, we have performed a case study on 15 well-known open source assets (i.e., libraries and frameworks). In particular, we first assess the reusability of the assets, based on the proposed index and other indices from the literature and subsequently contrast them to the actual reuse of those assets (the assessment of actual reuse in a particular context is further discussed in Section 3.3).

In Section 2 we present related work and in Section 3 we describe the proposed REusability Index (REI). In Section 4, we present the study design that was used for evaluation purposes (i.e., comparing the assessing power of REI and existing indices from the literature to the actual reuse). The evaluation results are presented and discussed in Sections 5 and 6. We present threats to validity in Section 7, and conclude the paper in Section 8.

2 Related Work

In this section, we present the reusability models / indices that have been identified by a recent mapping study [4]. For each reusability model / index, we present: (a) the aspects of the software that are considered (e.g., structural quality, documentation, etc.), (b) the way of synthesizing these aspects, and (c) the validation setup.

Bansiya and Davis [6] proposed a hierarchical quality model, named QMOOD, for assessing the quality of object-oriented artifacts and relied on human evaluators to assess its validity. The model provides functions that relate structural properties (e.g., encapsulation, coupling and cohesion) to high-level quality attributes, one of which is reusability. Furthermore, Nair et al. [18] examine the reusability of a certain class based on the values of three metrics defined in the Chidamber and Kemerer suite [8]. Multifunctional regression was performed across metrics to define the Reusability Index which was evaluated in two medium-sized java projects. Additionally, Kakarontzas et al. [14] proposed an index for assessing the reuse potential of object-oriented software modules. The authors used the metrics introduced by Chidamber and Kemerer [8] and developed a reuse index (named FWBR) based on the results of a logistic regression performed on 29 OSS projects. Their validation compared FWBR with the two aforementioned indices (from [6] and [18]). As a proxy of reusability, the authors used classes D-layer [14].

From another perspective, Sharma et al. utilized Artificial Neural Networks (AAN) to estimate the reusability of software components [20]. The rationale of their model is that structural metrics cannot be the sole predictors of components reusability, in the sense that reusability can be performed at other levels of granularity as well. Thus, they proposed four factors and several metrics affecting component reusability, namely: (a) customizability, measured as the number of setter methods per total number of properties, (b) interface complexity measured in scale from low to high, (c) understandability, depending on the appropriateness of the documentation (demos, manuals, etc.), and (d) portability measured in scale from low to high. The authors developed a network from 40 Java components and tested their results in 12 components presenting promising results. Finally, Washizak [21] suggested a metric-suite capturing the reusability of components, decomposed to understandability, adaptability, and portability. The validity of the model was evaluated with 125 components against expert estimations.

The metrics used in the aforementioned studies are summarized in Table 1. Based on Table 1, we can observe that the proposed models are either depending upon only structural characteristics, or they do not provide a way to aggregate the proposed metrics into a single reusability index. In this paper, we overcome this limitation, by synthesizing metrics that consider multiple aspects of software reusability, into a quantifiable reusability index (which is shown in the last column of Table1).

Table 1. Metrics Associated with Reusability

	Metrics	[6]	[14]	[18]	[20]	[21]	REI
Complexity - Structural Quality	*Direct Class Coupling*	X					
	Coupling between objects			X			
	Lack of cohesion between methods			X			X
	Cohesion among methods of class	X					
	Class interface size	X					
	Response for a class		X	X			
	Weighted methods for class		X	X			X
	Design size in classes	X					
	Number of Classes			X			X
	Depth of inheritance		X	X			
	Number of properties				X		
	Setter methods				X		
Adapta-bility	*Interface Complexity*				X		
	Number of External dependencies					X	X
External Quality	*Documentation quality*				X		X
	Existence of meta information					X	
	Observability					X	
	Portability				X	X	
	Number of open bugs						X
Availa -bilty	*Number of Components*						X
	Aggregation of Metrics	YES	YES	YES	NO	NO	YES

3 Proposed reusability index

In this section we present the proposed Reusability Index (***REI),*** which is calculated as a function of a set of metrics, each one weighted with a specific value. To select these metrics we consider the reusability model of Hristov et al. [13] that consists of eight main factors. Subsequently, we select metrics for each factor (see Fig. 1).

3.1 Reuse Factors

According to Hristov et al. [13] reusability can be assessed by quantifying eight main factors: incurred reuse, maintainability, adaptability, price, external quality, availability, documentation, and complexity. As this model consists of both structural and non-structural qualities, we consider it a fitting starting point, as we do not want to limit the proposed reusability index only to structural quality characteristics.

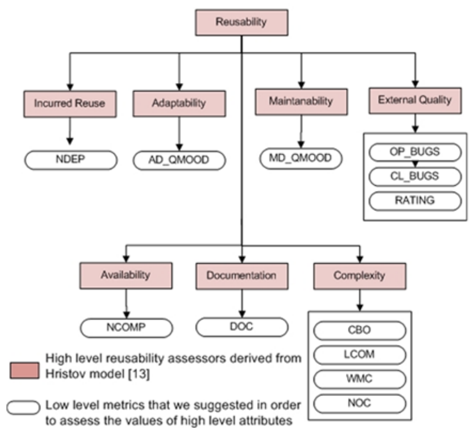

Fig. 1. Reusability measurement model

Incurred Reuse indicates the extent to which a software asset is built upon reused components. ***Adaptability*** is reflecting the ease of asset adaptation, when reused in a new system. ***Price*** indicates how expensive or cheap an asset is. ***Maintainability*** represents the extent to which an asset can be extended after delivery. ***External Quality*** describes the fulfillment of the asset's requirements and could be quantified by two different aspects: (a) bugs of the asset, and (b) rating from its users. ***Availability*** describes how easy it is to find an asset (e.g., instantly, after search, unavailable, etc.). ***Documentation*** reflects the provision of documents related to an asset. The existence of such documents makes the asset easier to understand and reuse. ***Complexity*** reflects the structure of the asset, and is depicted into many aspects of quality (e.g., the easiness to understand and adapt in a new context). Component/System complexity is measured through size, coupling, cohesion, and method complexity. We note that the relationship between reusability and these factors is not always positive. For example, the higher the complexity, the lower the reusability. Additionally, from the aforementioned factors, we do not consider price, since both in-house (assets developed by the same company) and OSS reuse, are usually not associated with a cost model (but not always).

3.2 Reuse Metrics

In this section we present the metrics that are used for quantifying the reusability factors. We note that the selected metrics are only few of the potential candidates.

Therefore, we acknowledge the existence of alternative metrics, and we do not claim that the set we selected consists of optimal reusability predictors. In total, 12 metrics have been addressed for the scope of this study in an attempt to avoid missing important attributes. The metrics are described in Table 2.

Table 2. Proposed Metrics

Reuse Factor	**Metric**	**Description/ Calculation Method**
Incurred Reuse	*NDEP*	The number of reused libraries in the project. It can be measured through the number of files in lib folder, or dependencies in the pom.xml (in case of Maven).
Adaptability	*AD_QMOOD*	As a proxy of adaptability, we use an index defined by Bansiya and Davies [6], as the ability of an asset to be easily adapted from the source systems that it has been developed for, to a target system (i.e., adaptation to the new context) **= -0.25 DCC + 0.25 CAM + 0.5 CIS + 0.5 DSC** DCC is calculated as the number of different classes that a class is related to. CAM is calculated using the summation of intersection of parameters of a method with the maximum independent set of all parameter types in the class. CIS is calculated as the number of public methods in a class. DSC is the total number of classes
Maintainability	*MD_QMOOD*	As a way to quantify maintainability, we use the metric for extendibility, as defined by Bansiya and Davies [6]. **= 0.25*ANA- 0.5*DCC+0.5*NOH+0.5*NOP** ANA is calculated as the average number of classes from which a class inherits information. NOH is the number of class hierarchies in the design. NOP is derived as a count of methods that can exhibit polymorphic behavior.
External Quality	*OP_BUGS*	The number of Open Bugs reported in the Issue Tracker of each asset.
	CL_BUGS	The number of Closed bugs as reported in the Issue Tracker of each asset.
	RATING	The average rating by the users of the software is a proxy for independent rating.
Documentation	*DOC*	To assess the amount, completeness, and quality of documentation, we suggest a manual inspection of the asset's website. We suggest a scale defined as follows: H—complete, rich, and easily accessible documentation. M—one of the aforementioned characteristics is not at a satisfactory level. L—two of the previous characteristics are not at a satisfactory level.
Availability	*NCOMP*	Number of Components. The number of independent components that have been identified for the specific asset. This is used as an indicator of how many concrete functionalities can be found into the asset. The methodology that is used to identify components from open source projects and populate the repository has been proposed by Ampatzoglou et al. [3] is based on the identification of reusable sets of classes, by applying a path-based strong component algorithm.
Complexity	*CBO*	Coupling between Objects. CBO measures the number of classes that the class is connected to, in terms of method calls, field accesses, inheritance, arguments, return types and exceptions. High coupling is related to low maintainability.
	LCOM	Lack of Cohesion of Methods. LCOM measures the dissimilarity of pairs of methods, in terms of the attributes being accessed. High Lack of Cohesion is an indicator of violating the Single Responsibility Principle [15], which suggests that each class should provide the system with only one functionality
	WMC	Weighted Method per Class. WMC is calculated as the average Cyclomatic Complexity (CC) among methods of a class.
	NOC	Number of Classes provides an estimation of the amount of functionality offered by the asset. The size of the asset needs to be considered, since smaller systems are expected to be less coupled, less complex.

3.3 Calculation of REI

REI is calculated as an aggregation of the values of independent metrics of the model, by performing Backwards Linear Regression on 15 Open-Source Software projects (OSS)[1]. The response (dependent) variable is the actual reusability according to Maven downloads[2]. The number of Maven downloads is an accurate measure of actual reuse, since each download from Maven corresponds to one case in which the asset has been reused in practice. That is because declaring the dependency to a Maven library in the POM file of a sample project, automatically downloads the library and puts it in the local Maven repository when the project is built (i.e. a download corresponds to an actual project having a dependency to that library). We note that even though we employed Maven downloads as an accurate reuse metric for our validation, our proposed index is not limited to open source reusable assets already stored in Maven. It can be equally useful for assessing the reusability of assets that: (a) are open-source but not deployed on Maven, or (b) are developed in-house, or (c) are of different levels of granularity (e.g., classes, packages, etc.) for which no actual reuse data can be found.

Regression is applied by initially including all predictor variables (see metrics on Table 2) and then by removing predictors, in a stepwise fashion, until there is no justifiable reason to remove any other predictor variable. The decision to apply backward regression was based on our intention to develop an index that is based on as few variables as possible. This reduction is expected to be beneficial regarding its applicability in the sense that: (a) it will be easier to calculate, and (b) it will depend upon fewer tools for automatic calculation. The end outcome of Backwards Linear Regression is a function, in which independent variables contribute towards the prediction of the dependent variable, with a specific weight, as follows:

$$REI = Constant + \sum_{i=0}^{i<num\_metrics} B(i) * metric(i)$$

To calculate REI we ended-up with a function of seven variables (i.e., metrics). The variables of the function accompanied with their weights and the standard beta coefficient are presented in Table 3, respectively. The standardized Beta of each factor can be used for comparing the importance of each metric in the calculation of REI. Finally, the sign of Beta denotes if the factor is positively or negatively correlated to the reusability of the asset. The accuracy of the index is presented in Section 5.2, since it corresponds to the predictive power of the REI index. The coefficients of the model as presented in Table 3 can be used to assess the reusability of assets whose actual levels of reuse are not available (e.g., OSS assets not deployed in Maven repository, or in-house developed assets, or assets of lower level of granularity—e.g., classes, or packages).

[1] Due to space limitations we present the 15 OSS projects that are used as a training set for the Backwards Linear Regression, together with the test set of the validation in Section 4.

[2] The number of downloads is retrieved from https://mvnrepository.com. The value is obtained by the “used by” artifacts tag which is expected to be more accurate.

Based on Table 3, ***components availability (NCOMP)*** and ***size (NOC)*** of the software asset are the most important metrics that influence its reusability, followed by ***number of dependencies (NDEP)*** and ***quality of documentation (DOC)***. From these metrics, size and number of dependencies are inversely proportional to reusability, whereas components availability and quality of documentation are proportional. A more detailed discussion of the relationship among these factors and reusability is provided in Section 5.1.

Table 3. REI Metric Calculation Coefficients

Metric (i)	B(i)	Std. Beta	Metric (i)	B(i)	Std. Beta
Constant	1.267,909		DOC	2.547,738	0,410
NDEP	-316,791	-0,524	LCOM	7,477	0,280
OP_BUGS	2,661	0,202	WMC	-1.081,78	-0,212
NCOMP	5,858	0,736	NOC	-11,295	-0,827

4 Case Study Design

To empirically investigate the validity of the proposed reusability index, we performed a case study on 15 open source reusable assets (i.e., libraries and frameworks), that are of course different from those that are used for developing the regression model (see Section 3.3). The study aims at comparing the validity of the obtained index (REI) to the validity of the two indices that produce a quantified assessment of reusability: (a) the QMOOD reusability index [6] and (b) the FWBM index proposed by Kakarontzas et al. [14]. The reusability obtained by each index is contrasted to the actual reuse frequency of the asset, as obtained by the Maven repository. QMOOD_R and FWBR have been selected for this comparison, since they provide clear calculation instructions, as well as a numerical assessment of reusability (similarly to REI), and their calculations can be easily automated with tools. The case study has been designed and reported according to the guidelines of Runeson et al. [19].

To investigate the validity of the proposed reusability index (REI) and compare it with two other reusability indices, we employ the properties described in the 1061 IEEE Standard for Software Quality Metrics [1]. The standard defines six metric validation criteria (namely: correlation, consistency, predictability, discriminative power, and reliability) and suggests the statistical test that shall be used for evaluating every criterion. We note that although the IEEE 1061 Standard introduces a sixth criterion, i.e., tracking, it has not been considered in this study since it would require to record the values of all metrics along the evolution of the software which is a heavy-weight process for an after-the-fact analysis. Therefore, we decided to omit this analysis from this study and propose it as a standalone research effort that would complementarily study the evolution of the levels of reusability for software assets.

Research Objectives and Research Questions. The aim of this study, based on GQM, is to *analyze* REI and other reusability indices (namely FWBM and QMOOD) for the purpose of evaluation *with respect to* their validity when assessing the reusability of software assets, *from the point of view of* software engineers *in the context of* open-source software reuse. Driven by this goal, two research questions have been set:

RQ$_1$: What is the correlation, consistency, predictability and discriminative power of REI compared to existing reusability indices?

RQ$_2$: What is the reliability of REI as an assessor of assets reusability, compared to existing reusability indices?

The first research question aims to investigate the validity of REI in comparison to the other indices, with respect to the first four validity criteria (i.e. correlation, consistency, predictability and discriminative power). For the first research question, we employ a single dataset comprising of all examined projects belonging to the test set of the case study. The second research question aims to investigate validity in terms of the fifth criterion: Reliability is examined separately since, according to its definition, each of the other four validation criteria should be tested on different projects.

Cases and Units of Analysis. This study is a holistic multiple-case study, i.e. each case comprises a unit of analysis. Specifically, the cases of the study are open source reusable assets (i.e., open source software libraries and development frameworks) found in the widely-known Maven repository. Thirty of the most reused software assets were selected based on their reuse potential [9] (see Table 4). Out of these assets, 15 were used as a training set to calculate REI (see Section 3.3) and the rest 15 as a test set for this case study. Each software asset can be categorized either as framework or library (see parenthesis in Table 4), which are the standardized methods for reusing third-party code. To classify an asset as a library or framework, we used its level of offered functionalities. Thus, we consider that a library performs specific, well-defined operations; whereas a framework is a skeleton through which the application defines operations[3].

Table 4. Selected Projects for Training and Test Set

Test Set Project		**Training Set Project**	
GeoToolKit (L)	Apache Axis (F)	jDom (L)	jFree (L)
ASM (L)	Plexus (F)	Commons-lang (L)	Commons-io (L)
Commons-cli (L)	POI (L)	Spring Framework (F)	slf4j (L)
Struts (F)	Slick 2D (L)	Joda-time (L)	Apache wicket (F)
Guava (F)	Wiring (F)	Jopt Simple (L)	Groovy (F)
WiQuery (L)	Wro4j (L)	scala xml (L)	iText (L)
ImageJ(L)	Xstream (L)	Lucene (L)	Superfly (L)
JavaX XML/saaj (F)		Apache Log4j (L)	

[3] See https://martinfowler.com/bliki/InversionOfControl.html

Data Collection. For each case (i.e., software asset), we have recorded seventeen variables, as follows: ***Demographics***: 2 variables (i.e., project, type). ***Metrics for REI Calculation***: 12 variables (i.e., the variables presented in Table 1). These variables are going to be used as the independent variables for testing correlation, consistency predictability and discriminative power. ***Actual Reuse***: We used Maven Reuse (MR), as presented in Section 3.3, as the variable that captures the actual reuse of the software asset. This variable is going to be used as the dependent variable in all tests. ***Compared Indices***: We compare the validity of the proposed index against two existing reusability indices, namely FWBR [14] and QMOOD [6] (see Section 2). Therefore, we recorded two variables, each one capturing the score of these indices for the assets under evaluation. The metrics have been collected in multiple ways: (a) the actual reuse metrics has been manually recorded based on the statistics provided by the Maven Repository website; (b) opened and closed bugs have been recorded based on the issue tracker data of projects; (c) rating has been recorded from the stars that each project has been assigned by the users in GitHub; (d) documentation was manually evaluated, based on the projects' webpages; and (e) the rest of the structural metrics, have been calculated using the Percerons Client tool. Percerons is an online platform [2] created to facilitate empirical studies.

***Data Analysis*.** To answer each RQ we will use three variables as candidate assessors of actual reuse: REI, QMOOD_R, and FWBR. The reporting of the empirical results will be performed, based on the performed analysis, using the Candidate Assessors as *Independent Variable*, and Actual reuse as the *Dependent Variable*:

- ***Predictability:*** we present the level of statistical significance of the effect (sig.) of the independent variable on the dependent (how important is the predictor in the model), and the accuracy of the model (i.e., mean standard error). While investigating predictability, we produced a separate linear regression model for each assessor (univariate analysis).
- ***Correlation*** and ***Consistency:*** we use the correlation coefficients (coeff.) and the levels of statistical significance (sig.) of Pearson and Spearman Correlation, respectively. The value of the coefficient denotes the degree to which the value (or ranking for Consistency) of the actual reuse is in analogy to the value (or rank) of the assessor.
- ***Discriminative Power:*** represents the ability of the independent variable to classify an asset into meaningful groups (as defined by the values of the dependent variables). The values of the dependent variable have been classified into 3 mutually exclusive categories (representing low, medium and high metric values) adopting equal frequency binning [7]. Then Bayesian classifiers [12] are applied in order to derive estimates regarding the discrete values of the dependent variables. The positive predictive power of the model is then calculated (precision) along with the sensitivity of the model (recall) and the models accuracy (f-measure).
- ***Reliability:*** we present the results of all the aforementioned tests, separately for the two types of reusable software types (i.e., libraries and development frameworks). The extent to which the results on the projects are in agreement (e.g., is the same metric the most valid assessor asset reusability for both types?) represents the reliability of REI.

5 Results

In this section, we present the results of the empirical validation of the proposed reusability index. The section is divided into two parts: In Section 5.1, the results of RQ_1 regarding the correlation, consistency, predictive and discriminative power of the REI are presented. In Section 5.2 we summarize the results of RQ_2, i.e., the assessment of REI reliability. We note that Section 5 only presents the raw results of our analysis and answers the research questions. Any interpretation of results and implications to researchers and practitioners are collectively discussed in Section 6.

5.1 RQ_1 — Correlation, Consistency, Predictive and Discriminative Power of REI Cases and Units of Analysis

In this section we answer RQ_1 by comparing the relation of REI, QMOOD_R, and FWBR to actual reuse, in terms of correlation, consistency, predictive and discriminative power. The results are cumulatively presented in Table 5. The rows of Table 5 are organized / grouped by validity criterion. In particular, for every group of rows (i.e., criterion) we present a set of success indicators. For example, regarding predictive power we present three success indicators, i.e., R-square, standard error, and significance of the model [10]. Statistically significant results are denoted with italic fonts.

Table 5. Correlation, Consistency and Predictive Power

Validity Criterion	**Success Indicator**	**REI**	**QMOOD_R**	**FWBR**	**Validity Criterion**	**Success Indicator**	**REI**	**QMOOD_R**	**FWBR**
Predictive Power	R-square	40.1%	4.0%	2.5%	Discriminative Power	Precision	53%	33%	33%
	Std. Error	4698.11	5270.36	5311.23		Recall	66%	33%	16%
	Significance	*0.08*	0.28	0.40		F-measure	60%	33%	22%
Corre-lation	Coefficient	0.633	0.200	-0.158	Consistency	Coefficient	0.587	0.330	-0.075
	Significance	*0.00*	0.28	0.40		Significance	*0.01*	*0.07*	0.69

Based on the results presented in Table 5, REI is the optimal assessor of software asset reusability, since: (a) it offers prediction significant at the 0.10 level, (b) it is strongly correlated to the actual value of the reuse (Pearson correlation coefficient > 0.6), and (c) it ranks software assets most consistently with respect to their reuse (Spearman correlation coefficient = 0.587). The second most valid assessor is QMOOD_R. Finally, we note that the only index that produces statistically significant results for all criteria at the 0.10 level is REI. QMOOD_R is able to provide a statistically significant ranking of software assets, however, with a moderate correlation.
To assess the discriminative power of the three indices, we employed Bayesian classifiers [12]. Through Bayesian classifiers we tested the ability of REI to correctly classify software assets in three classes (low, medium, and high reusability), with respect to their reuse (see Section 4). The accuracy of the classification is presented in Table

5, through three well-known success indicators: namely precision, recall, and F-measure [10]. Precision quantifies the positive predictive power of the model (i.e., TP / (TP + FP), and recall evaluates the extent to which the model captures all correctly classified artifacts (i.e., TP / (TP + FN). F-measure is a way to synthesize precision and recall in a single measure, since in the majority of cases there are trade-offs between the two indicators. To calculate these measures we split the dataset in a training and a test group in a random manner, using a 2-fold cross validation [12]. By interpreting the results presented in Table 5 we can suggest that REI is the index with the highest discriminative power. In particular, REI has shown the highest precision, recall, and f-measure. Therefore it has the ability to most accurately classify software assets into reuse categories.

REI has proven to be the most valid assessor of software asset reusability, when compared to the QMOOD reusability index and FWBR. In particular, REI excels in all criteria (namely correlation, consistency, predictive and discriminative power) being the only one providing statistically significant assessments.

5.2 RQ_2 — Reliability of the REI

In this section we present the results of evaluating the reliability of the three indices. To assess reliability, we split our test set into two subsets: frameworks and utility libraries. All the tests discussed in Section 5.1 are replicated for both sets and the results are compared. The outcome of this analysis is outlined in Table 6.

Table 6. Reliability

Validity Criterion	**Asset Type**	**Success Indicator**	**REI**	**QMOOD**	**FWBR**	**Validity Criterion**	**Asset Type**	**Success Indicator**	**REI**	**QMOOD**	**FWBR**
Predictive Power	L	R-square	35.0%	4.3%	5.6%	Consistency	L	Coeff.	0.570	0.270	-0.142
		Std. Error	4600.10	5581.19	5543.38			Sig.	*0.00*	0.22	0.52
		Sig.	*0.00*	0.35	0.29		F	Coeff.	0.476	0.071	0.143
	F	R-square	64.9%	13.4%	0.4%			Sig.	0.23	0.86	0.73
		Std. Error	3016.44	4734.91	5077.62	Discriminative power	L	Precision	50%	25%	13%
		Sig.	*0.01*	0.37	0.87			Recall	50%	25%	25%
Correlation	L	Coeff.	0.591	0.206	-0.236			F-measure	50%	25%	17%
		Sig.	*0.00*	0.35	0.29		F	Precision	85%	28%	31%
	F	Coeff	0.805	0.366	0.06			Recall	75%	38%	38%
		Sig.	*0.01*	0.37	0.87			F-measure	75%	31%	34%

For each validity criterion we present all success indicators for both libraries (L) and frameworks (F). With italics we denote statistically significant results. The results of Table 6 suggest that in most of the cases, the reusability index is more accurate in the group of frameworks rather than the libraries. The fact that the Frameworks dataset is of small size and still produces statistically significant results, further highlights the fitness of REI. Concerning reliability, REI has been validated as a reliable metric regarding correlation, predictive and discriminative power, but not regarding consistency. In particular, REI was not able to accurately rank the eight frameworks of the dataset. Nevertheless, compared to the other indices, REI achieves the highest levels of reliability.

The reliability analysis suggested that REI is consistently the most valid assessor of software asset reuse, regardless of the dataset. However, the ranking ability of the proposed index needs further investigation. Nevertheless, REI is the most reliable assessor of reusability, compared to the other indices.

6 Discussion

In this section we interpret the results obtained by our case study and provide some interesting implications for researchers and practitioners.

Interpretation of Results. The validation of the proposed reusability index on 15 open assets, suggested that REI is capable of providing accurate reusability assessments. REI outperforms the other examined indices (i.e., QMOOD_R and FWBR) and presents significant improvement in terms of estimation accuracy and classification efficiency. We believe that the main advantage of REI, compared to state-of-the-art indices is the fact that it synthesizes both structural aspects of quality (e.g., source code complexity metrics) and non-structural quality aspects (e.g., documentation, correctness, etc.). This finding can be considered intuitive in the sense that nowadays, software development produces a large data foot-print (e.g. bug trackers, issue trackers), and taking the diversity of the collected data into account provides a more holistic and accurate evaluation of software quality. Although the majority of reusability models and indices emphasize on low-level structural quality attributes (e.g., cohesion, complexity, etc.—quantified through source code structural metrics) the results of this study highlight the importance of evaluating non-structural artifacts. The contribution of the different types of characteristics is explained as follows:

- ***Low-level structural characteristics*** (complexity, cohesion, and size in classes). These are very important when assessing software assets reusability, in the sense that they highly affect the understandability of the reusable asset along its adaptation and maintenance. Although size can be related to reusability into two ways (i.e., as the amount of code that you need to understand before reuse or as the amount of offered functionality), we can observe that size is negatively affecting reuse (i.e., smaller assets are more probable to be reused). Therefore, the first interpretation of the relationship appears to be stronger than the second.

- ***High-level structural characteristics*** (number of dependencies and available components). First, the number of dependencies to other assets (i.e., an architectural level metric) seems to outperform low-level coupling metrics in terms of importance when assessing component reusability. This observation can be considered intuitive since while reusing a software asset developers are usually not interfering with structural asset dependencies, they are forced to "inherit" the external dependencies of the asset. Specifically, assets, whose reuse imply importing multiple external libraries (and thus require more configuration time), seem to be less re-used in practice by developers. Second, the number of available components, as quantified in this study, provides an assessment of modularity, which denotes how well a software asset can be decomposed to sub-components. This information is important while assessing reuse in the sense that a modular software is easier to understand and modify.
- ***Non-structural characteristics*** (quality of documentation and number of open bugs). First, documentation is an important factor that indicates the level of help and guidance that a reuser may receive during the adoption of a component. As expected, assets with a lot of documentation, are more likely to be reused. Second, open bugs suggest the number of pending corrections for a particular asset. Although this finding might be considered as unexpected, in the sense that assets with less bugs should be more attractive for reuse, we found that this number essentially acts as an indicator of the maturity of the asset. The results show that average and high values of OP_BUGS metric are indicators of higher reusability.

The multiple perspectives from which the REI index assesses reusability are further highlighted by the fact that from the seven factors that affect reusability (according to Hristov et al. [13]—see Section 3), only two are not directly participating in the calculation of REI (i.e., maintainability and adaptability). Although we did not originally expect this, we can interpret it as follows: either (a) the metrics that we have used for assessing these parameters, i.e., by borrowing equations from the QMOOD model, were sub-optimal, or (b) the metrics are subsumed by the other structural quality metrics that participate in the calculation of REI. Based on the literature LCOM, WMC, NOC have a strong influence on maintainability and extendibility. Therefore a synthesized index (like REI) does not seem to benefit from including extra metrics in its calculation that are correlated to other metrics that participate in the calculation.

Implications to researchers and practitioners*.** The major findings of this study show that reusability indices need to further focus on the inclusion of non-structural factors. We encourage ***researchers to introduce formal metrics and procedures for quantifying quality aspects that till now are evaluated by adopting ad-hoc procedures. Attributes like Documentation, External Quality and Availability are underexplored and usually assessed subjectively. More formal definitions of these factors could further increase the accuracy and adoption of reusability metrics. Additionally, we believe that researchers should evaluate the proposed reusability model on inner source development ([21]). From such a study, it would be interesting to observe differences in the parameters and the weight that will participate in the calculation of REI. A possible reason for deviation, is the belief that reusable assets that have been developed inside a single company might present similarities in terms of some factors (e.g., documentation, open bugs, etc.). In that case it might it be interesting to investigate the introduction of new metrics customized to the specificities of each software company. This is particularly important, since for in-house components it is not possible to obtain an external, objective reusability measure such as Maven Reusability (MR). Finally, we suggest to further validate REI, based on the effort required to adopt the asset in a fully operating mode in a new software. Clearly it is important to select the right asset that will require less time, effort, cost and modifications while being reused. Similarly to any empirical endeavor, we encourage the replication of REI validation in larger samples of reusable assets, examining different types of applications, in order to further refine its accuracy.

Regarding practitioners, the proposed reusability index will be a useful tool for aiding practitioners to select the most important factors (and the associated metrics) to be used when assessing in-house reusable assets, or OSS assets that are not deployed on the Maven repository. The fact that the majority of metrics that are used for quantifying REI can be automatically calculated from available tools, in conjunction with its straightforward calculation, is expected to boost the adoption of the index, and its practical benefits. The two-fold analysis that we adopt in this paper (i.e., prediction and classification) enables practitioners to select the most fitting one for their purposes. In particular, the classification of assets to low, medium, and highly reusable, provides a coarse-grained, but more accurate approach. Such an approach can be useful when software engineers are not interested in quantifying the actual value of reusability, but when they are just interested in characterization purposes.

7 Threats to Validity

In this section we discuss the threats to validity, based on the classification schema of Runeson et al. [19]. **Construct Validity** in our case refers to whether all the relevant reusability metrics have been explored in the proposed index. To mitigate this risk we considered in the calculation of the index a plethora of reusability aspects representing both structural and non-structural qualities such as, adaptability, maintainability, quality, availability, documentation, reusability and complexity each of which synthesized by the values of 12 metrics as depicted in Table 2. Furthermore, as already mentioned in Section 3 the selected metrics are established metrics for the respective factors, although we do not claim they are the most optimal ones**.** Additionally although the number of downloads from Maven is considered an accurate assessor of reuse (see Section 3.3) we need to acknowledge the selection of this metric as a possible threat to construct validity. **Internal Validity** is related to the examination of causal relations. Our results pinpoint particular metrics that affect significantly the reuse potential of a certain project but still we do not infer causal relationships.
Concerning generalizability of results, known as **External Validity** we should mention that different data sets could cause differentiations in the results. Still this risk is mitigated by the fact that the analysis was performed selecting a pool of projects that are well-known and popular in the practitioners community [9] forming a representative sample for analysis. However, a replication of this study in a larger project set

and in an industrial setting would be valuable in verifying the current findings. Regarding the reproducibility of the study known as **Reliability**, we believe that the research process documented thoroughly in Section 4 ensures the safe replication of our study by any interested researcher. However, researcher bias could have been introduced in the data collection phase, while quantifying the metric value of the level of documentation provided for each project. In that case, the first two authors gathered data on the documentation variable, adopting a manual recording process. The results were further validated by the third and fourth author.

8 Conclusions

The selection of the most fitting and adaptable asset is one of the main challenges of the software reuse process as it depends on the assessment of a variety of quality aspects characterizing the candidate assets. In this study we presented and validated the Reusability Index (REI), which decomposes reusability to seven quality factors quantifying each one of them with certain metrics. Non-structural metrics along with low- and high-level structural metrics synthesize the proposed reusability index. Based on this model, REI is derived by applying backward regression. To investigate the validity of REI we have employed a two-step evaluation process that validates the proposed index against: (a) well-known reusability indices found in literature, and (b) the metric validation criteria defined in the 1061-1998 IEEE Standard for a Software Quality Metrics [1]. The results from the holistic multiple-case study on 15 OSS projects suggested that REI is capable of providing accurate reusability assessments. REI outperforms the other examined indices (i.e., QMOOD_R and FWBR) and presents significant improvement in terms of estimation accuracy and classification efficiency. Based on these results, we provide implications for researchers and practitioners.

9 Acknowledgement

This work was financially supported by the action "Strengthening Human Resources Research Potential via Doctorate Research" of the Operational Program "Human Resources Development Program, Education and Lifelong Learning, 2014-2020", implemented from State Scholarship Foundation (IKY) and co-financed by the European Social Fund and the Greek public (National Strategic Reference Framework (NSRF) 2014 – 2020).

10 References

1. 1061-1998: IEEE Standard for a Software Quality Metrics Methodology, IEEE Standards, IEEE Computer Society, 31 December 1998 (reaffirmed 9 December 2009).
2. A. Ampatzoglou, I. Stamelos, A. Gkortzis, and I. Deligiannis, “Methodology on Extracting Reusable Software Candidate Components from Open Source Games”, Proceeding of the 16th International Academic MindTrek Conference, ACM, pp. 93–100, Finland, 2012.

3. A. Ampatzoglou, A. Gkortzis, S. Charalampidou, and P. Avgeriou, “An Embedded Multiple-Case Study on OSS Design Quality Assessment across Domains”, 7th International Symposium on Empirical Software Engineering and Measurement (ESEM’ 13), ACM/IEEE Computer Society, pp. 255-258, Octomber 2013, Baltimore, USA
4. E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, M. Galster and P. Avgeriou. “A mapping study on design-time quality attributes and metrics”, Journal of Systems and Software, 127, (2017): 52-77.
5. M. T. Baldassarre, A. Bianchi, D. Caivano, and G. Visaggio, “An industrial case study on reuse oriented development”, 21st International Conference on Software Maintenance (ICSM’05), IEEE Computer Society
6. J. Bansiya and C. G. Davis. "A hierarchical model for object-oriented design quality assessment." IEEE Transactions on software engineering 28.1 (2002): 4-17.
7. S. Bibi, A. Ampatzoglou, and I. Stamelos. A Bayesian Belief Network for Modeling Open Source Software Maintenance Productivity. In 12th International Conference on Open Source Software Systems (OSS), Springer, (2016): 32-44.
8. S.R. Chidamber, C.F. Kemerer. “A metrics suite for object oriented design”. IEEE Transactions on software engineering, 20(6), (1994): 476-493.
9. E. Constantinou, A. Ampatzoglou, I. Stamelos. Quantifying reuse in OSS: A large-scale empirical study. Int. Journal of Open Source Software and Processes (IJOSSP), 2014.
10. A. Field, 2013. Discovering Statistics using IBM SPSS Statistics. SAGE Publications Ltd.
11. G. Gui, P. D. Scott, Ranking reusability of software components using coupling metrics, Journal of Systems and Software, Volume 80, Issue 9, September 2007, Pages 1450-1459,
12. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, & I. Witten. The WEKA data mining software: an update. ACM SIGKDD explorations newsletter, 11(1), (2009): 10-18.
13. D. Hristov, "Structuring software reusability metrics for component-based software development." 7th International Conference on Software Engineering Advances. 2012.
14. G. Kakarontzas, E. Constantinou, A. Ampatzoglou, and I. Stamelos: Layer assessment of object-oriented software: A metric facilitating white-box reuse. Journal of Systems and Software 86(2): 349-366 (2013)
15. R.C. Martin “Agile software development: principles, patterns and practices”, Prentice Hall, New Jersey. 2003.
16. H. Mili, F. Mili, and A. Mili, "Reusing software: Issues and research directions." IEEE Transactions on Software Engineering 21(6), pp.528-562, 1991.
17. M. Morisio, D. Romano, and I. Stamelos, “Quality, productivity, and learning in framework-based development: an exploratory case study”, Transactions on Software Engineering, IEEE Computer Society, 28 (9), pp. 876–888, September 2002.
18. T. R. G. Nair and R. Selvarani. 2010. Estimation of software reusability: an engineering approach. SIGSOFT Softw. Eng. Notes 35, 1 (January 2010), 1-6.
19. P. Runeson, M. Höst, A. Rainer and B. Regnell. “Case Study Research in Software Engineering: Guidelines and Examples”, John Wiley & Sons, 2012
20. A. Sharma, P. S. Grover, and R. Kumar. 2009. Reusability assessment for software components. SIGSOFT Softw. Eng. Notes 34, 2 (February 2009), 1-6.
21. H. Washizaki, H. Yamamoto, and Y. Fukazawa. "A metrics suite for measuring reusability of software components", 9th International Software Metrics Symposium, IEEE, 2003.