WILEY **Software**: Evolution and Process

REI: An integrated measure for software reusability

Ioannis Zozas[1] (ID) | **Apostolos Ampatzoglou**[2] (ID) | **Stamatia Bibi**[1] (ID) |
Alexander Chatzigeorgiou[2] (ID) | **Paris Avgeriou**[3] (ID) | **Ioannis Stamelos**[4]

[1] Department of Informatics and Telecommunications, University of Western Macedonia, Kozani, Greece

[2] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece

[3] Faculty of Science and Engineering, University of Groningen, Groningen, Netherlands

[4] School of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece

Correspondence
Ioannis Zozas, Department of Informatics and Telecommunications, University of Western Macedonia, Kozani, Greece.
Email: izozas@uowm.gr

Funding information
State Scholarships Foundation (IKY); European Social Fund and the Greek public

Abstract

To capitalize upon the benefits of software reuse, an efficient selection among candidate reusable assets should be performed in terms of functional fitness and adaptability. The reusability of assets is usually measured through reusability indices. However, these do not capture all facets of reusability, such as structural characteristics, external quality attributes, and documentation. In this paper, we propose a reusability index (REI) as a synthesis of various software metrics and evaluate its ability to quantify reuse, based on IEEE Standard on Software Metrics Validity. The proposed index is compared with existing ones through a case study on 80 reusable open-source assets. To illustrate the applicability of the proposed index, we performed a pilot study, where real-world reuse decisions have been compared with decisions imposed by the use of metrics (including REI). 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. The findings of the paper are discussed to understand the most important aspects in reusability assessment (interpretation of results), and interesting implications for research and practice are provided.[1]

KEYWORDS

metrics, quality model, reuseability, validation

1 | INTRODUCTION

Software reuse is considered as a key solution to increase software development productivity and decrease the number of bugs.[1,2] The process of software reuse entails the use of existing software assets to implement new software systems or extend existing ones.[3] Reuse can take place at multiple levels of granularity, ranging from reusing a few lines of code (usually through the white-box reuse process) to reusing complete components/libraries (usually as black-box reuse). The process of reusing software assets consists of two major tasks:

- *Reusable asset identification*. The *reuser* has to identify an asset (ie, a piece of source code such as method, class, package, or a component/library), which implements the functionality that he/she wants to reuse. This task is challenging because the available amount of assets (either in-house or open source) that can be reused is vast, offering a large variety of candidates. In many cases, the candidate reusable assets are not well organized and documented. The assets that are identified as functionally fitting are then evaluated, based on some criteria, and one is selected for reuse.

---

[1]Pre-print of article accepted to Journal of Software: Evolution and Process

- *Reusable asset adaptation.* After the asset has been selected for reuse, the *reuser* has to adapt it to fit the architecture of the target system. For example, in white-box reuse, source code needs to be changed, whereas in black-box reuse, dependencies need to be handled, and the API needs to be exploited and understood. If the asset is well structured, understandable, and maintainable, the effort required to adapt it is significantly lower. The assessment of adaptability is a nontrivial task; although there are plenty adaptability indicators, a specific asset cannot be safely characterized as easily adaptable or not, in isolation, due to the lack of well-established metric thresholds.

To summarize the requirements for asset selection per task: during identification, the functional fitness of the asset needs to be considered; during adaptation, the functionally equivalent assets must be evaluated based on the extent to which they enable ease of adaptation, eg, through sound documentation and internal structure. To guide "*reusers*" in the selection of the most adaptable asset (among the functionally fitting ones), reusability needs to be assessed, ie, the degree to which a certain asset can be reused in other systems. To this end, a wide range of reusability indices have been proposed[4,5]; these indices are calculated as functions of metric scores that quantify factors that influence reusability. However, several research efforts in the literature on reusability assessment (see Section 2) suffer from one or both of the following two limitations: (a) only highlight the parameters that should be considered in reusability assessment, thus, *not providing an aggregated reusability measure*, or (b) *only consider structural aspects of the software asset*, ignoring aspects such as documentation, external quality, etc. We note that the aforementioned reusability indices are only relevant for comparing functionally equivalent assets, because the main characteristic that an asset must have so as to be reused is to provide the necessary functionality. Additionally, the applicability of such indices is focusing on "in-house" and/or "open-source" reuse artifacts, and not third-party closed source code. The reason for this is that in-house and OSS code are available prior to their usage for comparison with competing assets, which is usually not the case for third-party closed-source code.

The goal of this study is to propose and validate a *reusability index* (REI) that overcomes these 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*). The novelty of the proposed index lies in the fact that a holistic (ie, not only dependent on structural aspects of software) and quantifiable measure of reusability is proposed. To validate the accuracy of the developed index, we have performed a case study on 80 well-known open source assets. In particular, we assess the reusability of the software assets, based on the proposed index and then contrast it to their actual reuse, as obtained by the Maven repository.* We note that despite the fact that this accurate reuse metric is available for a plethora of open source reusable assets (already stored in Maven repository), our proposed index can be useful for assessing the reusability of assets that (a) are not deployed on Maven, (b) are developed in-house, or (c) are of different levels of granularity (eg, classes, packages, etc.) for which no actual reuse data can be found.

The rest of the paper is organized as follows: In Section 2, we present related work (focusing on existing reusability models), and in Section 3 we describe in detail the proposed REI. In Section 4, we present the case study design that we have used for evaluation purposes, whose results we present in Section 5, and discuss in Section 7. In Section 6, we present an illustrative example discussing the applicability of REI in white-box artifact selection process. Finally, we present threats to validity in Section 8 and conclude the paper in Section 9. We note that this paper is an extended and revised version of the study obtaining the best paper award in the ICSR 2018 conference.[6] The main points of differentiation of this study compared with the original one published in ICSR are as follows: (a) we have expanded the related work section so as to include a detailed comparison to the state-of-the-art; (b) we have expanded our validation dataset from 30 to 80 projects; (c) we have investigated the sixth criterion of IEEE standard for metrics validation (namely tracking); and (d) we added two proof of concept pilot studies, which demonstrate the applicability of the model in practice and provided an initial assessment of the gained benefits.

2 | RELATED WORK

In this section, we present previous studies that are related to this paper. First, we describe the context of this paper, ie, software reuse, emphasizing on the main benefits and challenges of the domain. Subsequently, in Section 2.2, we present reusability models and indices that are comparable to our work.

2.1 | Software reuse

Software reuse according to McIlroy[7] is the "the process of creating software systems from existing software, rather than building software systems from scratch." Software reuse as a process targets in building software systems with lower development cost, increased efficiency and maintainability, and improved quality as mentioned by Leach,[8] who describes a variety of reuse types with respect to the level granularity. As referred by Leach,[8] reuse can be performed by utilizing a number of lines of code, functions, packages, subsystems, or even entire programs. Furthermore, according to Krueger,[9] software reuse may be applied to requirements, architectures, design documents, and design patterns, in

---

*Maven is one of the most popular reuse repositories because at this point it hosts more than 6 million projects. Because all Maven projects automatically download externally reused libraries from this repository, we consider the Maven Reuse measure an accurate proxy of actual reuse.

order to apply during all phases of the software development lifecycle. While the potentials of software reuse are widely recognized, there are also many challenges that software engineers face during the reuse process. Such difficulties include the differentiations in the requirements and design between the reusable elements and the target system that rise the need of formal procedures for selecting the appropriate reusable artifacts[10] and handling reuse expenses.[8] Concerning economic challenges, Jalender et al[11] denoted the need for building economic and organizational models within development process in order to implement the software reuse concept. Tripathy and Naik,[12] to confront the technical challenge of selecting reusable software artifacts, highlighted the need to establish a minimum quality level of certain properties, such as reliability, adaptability, and platform independence according to which reusable artifacts can be prioritized. Furthermore, Sojer et al[13] proposed a development process model to support software reuse, including project management and tool distribution. The model aimed to provide portable software artifacts without the need of rework, thus reducing related costs.

2.2 | Software reusability models and indices

In this section, we present the reusability models and indices that have been identified by a recent mapping study on design time-quality attributes.[5] For each reusability model/index, we present (a) the aspects of the software that are considered (eg, structural quality, documentation, etc.); (b) the way of synthesizing these aspects; and (c) the validation setup (ie, type of study, dataset, the way to quantify reuse as an independent variable). A summary of the metrics that exist in the literature is presented in Table 1.

First, Bansiya and Davies[14] proposed a hierarchical quality model, named QMOOD, for assessing the quality of object-oriented artifacts. The model provides functions that relate structural properties (eg, encapsulation, coupling, and cohesion) to high-level quality attributes (eg, reusability, flexibility, etc.). The metrics that QMOOD links to reusability are presented in Table 1. For evaluation purposes, Bansiya and Davis relied on human evaluators for assessing the validity of the model. Furthermore, Nair et al[15] assessed the reusability of a certain class based on the values of three metrics defined in the Chidamber and Kemerer suite.[16] Multifunctional regression was performed across metrics to define the REI which was evaluated in two medium-sized java projects.

Additionally, Kakarontzas et al[17] proposed an index for assessing the reuse potential of object-oriented software modules. The authors used the metrics introduced by Chidamber and Kemerer[16] and developed a reuse index (named FWBR) based on the results of a logistic regression

TABLE 1 Metrics associated with reusability

Metrics	14	17	15	19	18	20	21	22	23
Direct class coupling	X					X		X	X
Coupling between objects			X			X		X	X
Lack of cohesion between methods			X			X		X	
Cohesion among methods of class	X					X			X
Class interface size	X								
Response for a class		X	X					X	
Weighted methods for class		X	X			X	X	X	X
Design size in classes	X						X		
Number of classes			X			X	X		
Depth of inheritance		X	X					X	
Number of properties				X		X	X		
Setter methods				X					
Interface complexity				X			X		X
Number of external dependencies					X	X			
Documentation quality				X					X
Existence of meta information					X	X			X
Observability					X				X
Portability				X	X	X	X		X
Number of open bugs									X
Number of components						X	X		
Aggregation of metrics	Y	Y	Y	N	N	Y	Y	Y	N

performed on 29 OSS projects. Their validation compared FWBR with the two aforementioned indices (from Bansiya and Davis[14] and Nair and Selvarani[15]). As a proxy of reusability, the authors used classes D-layer.[17]

From another perspective, Sharma et al utilized artificial neural networks (AAN) to estimate the reusability of software components.[18] The rationale of the proposed 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.

Moreover, Washizak[20] 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. Fazal et al[21] proposed a reusability model based on novel and existing metrics from a series of project releases. A mathematical formula has been introduced to aggregate low level metrics to a REI. The proposed index has been validated in eight open source software packages. Goel et al[22] proposed a model based on object-oriented programming language metrics, which was validated on three C++ projects. The model utilized the Chidamber and Kemerer (CK) metrics suite,[16] as well as novel inheritance metrics. Finally, Hristov[23] proposed a reusability model based on several reuse factors (namely reuse, adaptability, price, maintainability, quality, availability, documentation, and complexity).

2.3 | Contributions compared with the state-of-the-art

The main limitations identified in the state-of-the-art reusability metrics are the following: (a) only a limited number of models provide an aggregated index for the calculation of reusability; (b) the models that provide a REI (ie, they do not only present factors that influence reusability) are considering only structural characteristics; (c) the used structural characteristics are usually concentrated at the lowest levels of hierarchical quality models; and (d) the accuracy of existing reusability indices is quite low. Therefore, in this study, we aim to propose a reusability model that will lead to an aggregated index, which will consist of high-level and low-level structural quality characteristics, but will also take into account other aspects of the reused artifacts, such as external quality, documentation, etc. The performance of the obtained index will be compared with state-of-the-art models, and it will be investigated whether it can advance the levels of accuracy already achieved.

3 | PROPOSED REUSABILITY INDEX (REI)

In this section, we present the proposed REI. The proposed index is calculated as a function of a set of metrics, each one weighted with a specific value. To perform candidate metric selection, we use a three-level hierarchical reusability model (see Figure 1). The model decomposes reusability
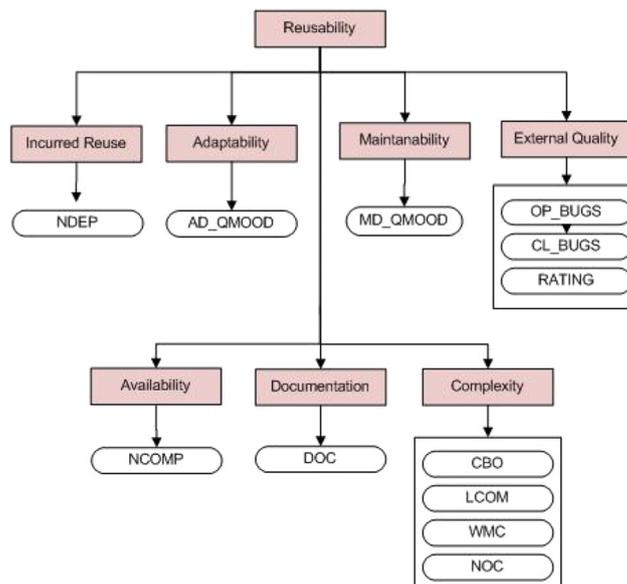

FIGURE 1 Reusability measurement model

to seven factors and then proposes metrics for each one of them. In Section 3.1, we present the reuse factors, whereas in Section 3.2 we present the metrics that we have selected for quantifying the quality subcharacteristics of the model.

3.1 | Reuse subcharacteristics

According to Hristov et al, reusability can be assessed by quantifying eight main factors: *incurred reuse, adaptability, price, maintainability, external quality, availability, documentation*, and *complexity*,[23] defined as follows:

- *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 (eg, during a new version). Maintainability is reflected in the source code.
- *External quality* describes the fulfillment of the asset's requirements and could be quantified by different aspects: (a) bugs of the asset and (b) rating from its users.
- *Availability* describes how easy it is to find an asset (eg, 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 internal structure of the asset and is depicted into many aspects of quality (eg, 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 highest the complexity, the lower the reusability. Additionally, from the aforementioned factors, we do not consider price, because both in-house (assets developed by the same company) and OSS reuse are, usually, not performed under a monetary transaction.

3.2 | Reuse metrics

In this section, we present the metrics that we selected for quantifying each of the reusability factors. We note that the metrics that we selected for the quantification of the reuse factors 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. However, the validity of the selected set will be evaluated in the conducted empirical study (see Section 5). The calculation of the proposed REI index is based on a synthesis of the most fitting among metrics. The synthesis process is explained in detail in Section 4.5. We note that in some cases, more than one metrics are used for each factor. The metrics are described in Table 2. Finally, although some repetition with Section 3.1 might exist here, the two subsections serve different goals: Section 3.1 is focusing on the quality properties in which reusability is decomposed to, according to Hristov, whereas Section 3.2 aims at mapping these properties to metrics that can be used for their quantification.

3.3 | Calculation of the REI

REI is calculated as an aggregation of the values of independent metrics (see Table 2) of the model, by performing backwards linear regression (using as independent variable the actual reuse from the Maven repository). The decision to apply backward regression was based on our intention to develop an index that depends on as few variables as possible: therefore, not all metrics in Table 2 will be used in the model, but the minimum number of metrics that can lead to the best fitting model. The nature of the algorithm (ie, backward linear regression) removes any possible bias, or convenience choice, for the selected metrics. 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 linear regression is a function, in which independent variables contribute toward the prediction of the dependent variable, with a specific weight, as follows:

REI = Constant +
Σ ($B_i \times$ metric$_i$), where Σ the summary of each metric i.

To perform the backward linear regression, we have developed a dataset from 15 open-source reusable assets. The design of the dataset construction is presented in detail in Section 4. Upon the execution of the backward linear regression, we ended up with a function of seven variables (ie, metrics). The variables of the function accompanied with their weights are presented in the first and the second column of Table 3, respectively. The third column of the table represents the standardized Beta of each factor, which can be used for comparing the importance of each

TABLE 2 Metrics of reusability model

Reuse Factor	Metric	Description
Incurred reuse	NDEP	Number of dependencies. As a way to quantify the extent to which the evaluated asset is based upon reuse, we use the number of external libraries.
Adaptability	AD_QMOOD	As a proxy of adaptability, we use an index defined by Bansiya and Davies,[14] as the ability of an asset to be easily adapted from the source systems that it has been developed for, to a target system (ie, adaptation to the new context)
Maintainability	MD_QMOOD	As a way to quantify maintainability, we use the metric for extendibility, as defined by Bansiya and Davies.[14] According to QMOOD, extendibility can be calculated as a function of several object-oriented characteristics that either benefit or hinder the easy maintenance of the system.
External quality	OP_BUGS	External quality can be quantified by measuring the number of open bugs reported in the issue tracker of each asset. The number of open bugs may represent the maturity of a certain software asset and the communities interest in that asset.
	CL_BUGS	The number of closed/resolved bugs as reported in the issue tracker of each asset is also an indicator of external quality. The number of closed/resolved bugs may represent the community's readyness and interest in a certain software asset.
	RATING	The average rating by the users of the software is used as a proxy for independent rating and certification
Documentation	DOC	To assess the amount, completeness, and quality of documentation, we suggest a manual inspection of the asset's website. We suggest a Likert-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.
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 and understandability.
	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,[24] 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. High WMC results in difficulties in maintaining and understanding the asset
	NOC	Number of classes provides an estimation of the amount of functionality offered by the asset.[2] The size of the asset needs to be considered, because smaller systems are expected to be less coupled, less complex, to have less classes as leaves in hierarchies, and use less inheritance trees

TABLE 3 Measure validation analysis

Criterion	Test	Variables
Predictability	Linear regression	Independent: Candidate assessors Dependent: Actual reuse
Correlation	Pearson correlation	Independent: Candidate assessors Dependent: Actual reuse
Consistency	Spearman correlation	Independent: Candidate assessors Dependent: Actual reuse
Tracking	Same as consistency, repeated along the existing project versions	
Discriminative power	Bayesian networks	Independent: Candidate assessors Dependent: Actual reuse
Reliability	All the aforementioned tests (separately for each reusable software type)	

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 5.1, because it corresponds to the predictive power of the REI. 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 (eg, OSS assets not deployed through the Maven repository, or in-house developed assets, or assets of lower level of granularity—eg, sets of classes, packages).

Based on Table 4, *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

TABLE 4　REI metric calculation coefficients

Metric (i)	B (i)	Standardized Beta
Constant	1267.909	
NDEP	−316.791	−0.524
OP_BUGS	2.661	0.202
NCOMP	5.858	0.736
DOC	2547.738	0.410
LCOM	7.477	0.280
WMC	−1081.78	−0.212
NOC	−11.295	−0.827

inversely proportional to reusability, whereas components availability and quality of documentation are proportional. The aforementioned structural metrics can be calculated with Percerons Client,[†] the number of bugs can be extracted from the Issue Tracker,[‡] whereas documentation can only be assessed with manual inspection. A more detailed discussion of the relationship among these factors and reusability is provided in Section 7.1. Finally, we need to note that although a different regression function could have been created for the different types of reusable artifacts (ie, libraries and frameworks), we opted not to do so. Although such a decision would eliminate the need for reliability checking, it would hurt the generalizability of the model, and its ability to be used outside the types of assets for which it has been trained. Moreover, setting up a different regression model for libraries and frameworks would pose a major threat to the validity of the index as the distinction between the two is not sharp (eg, many frameworks act also as libraries of components).

REI can be calculated as a function of seven metrics, among which components availability, size, number of dependencies, and quality of documentation are the most important ones.

4 | CASE STUDY DESIGN

To empirically investigate the validity of the proposed REI, we performed a case study on 80 open source reusable assets (ie, libraries and frameworks). In particular, we compare the validity of the obtained index (REI) to the validity of the two indices that produce a quantified assessment of reusability: (a) QMOOD REI[14] and (b) FWBM index proposed by Kakarontzas et al.[17] 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, because 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] In this section, we present (a) the goal of the case study and the derived research questions, (b) the description of cases and units of analysis, (c) the data collection procedure, and (d) the data analysis process. Additionally, we present the metric validation criteria.

4.1 | Metric validation criteria

To investigate the validity of the proposed REI and compare it with two other reusability indices, we employ the properties described in the 1061 IEEE Standard for Software Quality Metrics.[25] The standard defines six metric validation criteria and suggests the statistical test that shall be used for evaluating every criterion, as follows:

- *Correlation* assesses the association between a quality factor and the metric under study to warrant using the metric as a substitute for the factor. The criterion is quantified by using a correlation coefficient.[25]

- *Consistency* assesses whether there is consistency between the ranks of the quality factor values (over a set of software components) and the ranks of the corresponding metric values. Consistency determines if a metric can accurately rank a set of artifacts and is quantified with the coefficient of rank correlation.[25]

- *Tracking* assesses if values of the metric under study can follow changes in the levels of the quality characteristic. Tracking is quantified by using the coefficient of rank correlation for a set of project versions .[25]

[†]http://www.percerons.com

[‡]https://guides.github.com/features/issues/

- *Predictability* assesses the accuracy with which a metric applied at a time point t1 is able to predict the levels of the quality factor at a time point t2. The criterion is quantified through the standard estimation error for a univariate regression model.[25]

- *Discriminative power* assesses if the metric under study is capable of discriminating between high-quality and low-quality components. Discriminative power can be quantified through the precision and recall[26] of a classification approach, using categorical variables.

- *Reliability* assesses if the metric under study can fulfill all five aforementioned validation criteria, over a sufficient number of applications. This criterion can offer evidence that a metric can perform its intended function consistently. This criterion can be assessed by replicating the previous tests (for each of the aforementioned criteria) to various software systems.[25]

4.2 | Research objectives and research questions

The aim of this study, expressed through a GQM formulation, is: *to analyze REI and other reusability indices (ie, 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 software reuse*. Driven by this goal, two research questions have been set:

RQ$_1$:

What is the correlation, consistency, tracking, predictive, and discriminative power of REI compared with existing reusability indices?

RQ$_2$:

What is the reliability of REI as an assessor of assets reusability, compared with 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 five validity criteria (ie, correlation, consistency, tracking, predictability, and discriminative power). For the first research question, we employ a single dataset comprising all examined projects of the case study. The second research question aims to investigate validity in terms of reliability. Reliability is examined separately because, according to its definition, each of the other five validation criteria should be tested on different projects. In particular, for this research question, we created two groups of software projects (as explained in Section 4.4) and then the results are cross-checked to assess the metric's reliability.

4.3 | Cases and units of analysis

This study is a holistic multiple-case study, ie, each case comprises a unit of analysis. Specifically, the cases of the study are open source reusable assets found in the widely known Maven repository. Thirty of the most reused software assets[27] were selected to participate in the analysis, as well as 50 random assets. The ranking of these cases, based on reuse potential, was enabled by exploiting a previous work of Constantinou et al[27] that explored the Google Code repository and identified the most reused libraries from more than 1000 open source projects.

Therefore, our study was performed on 80 Java open source software assets, as presented in Table 5. In particular, in Table 5, we present the name of the software asset and its goal. The last two columns of the table correspond to the size of the asset in terms of number of classes and the number of times that the asset has been reused through the Maven repository. This last column will be the proxy of actual reuse in our study. This selection is sensible, in the sense that all Maven projects, worldwide, are automatically connecting to this repository for downloading the binary files of the assets that they reuse. Some statistics describing our sample are presented in Figure 2. In particular, the bar chart illustrates the distribution of the size (NOC) of the selected projects.

4.4 | Data collection

For each case (ie, software asset), we have recorded 17 variables, as follows:

- *Demographics*: two variables (ie, project and type). As project type, we split the dataset in the middle by characterizing artifacts as frequently reused or seldomly reused (cut-off 50 reuses from Maven repository).

- *Metrics for REI calculation*: 12 variables (ie, the variables presented in Table 2). 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 4.3, 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[17] and QMOOD[14] (see Section 2). Therefore, we recorded two variables, each one capturing the score of these indices for the assets under evaluation.

TABLE 5 Reusable assets demographics

Project	Goal	Size (NOC)	Maven Reuse (MR)
Apache Axis	Web development	636	116
Apache Log4j[a]	Logging	221	12.384
Apache wicket	Web development	831	7.149
ASM	Bytecode reader	23	1.145
Commons-cli	CMD line management	22	1.852
Commons-io[a]	File management	115	11.935
Commons-lang	Text management	131	8.002
GeoToolKit[a]	Geospatial	204	848
Groovy	Programming language	1659	3.979
Guva	Parallel programming	467	13.986
ImageJ	Visualization	374	487
iText[a]	Text management	553	242
JavaX XML/saaj[a]	XML management	28	149
jDom	XML management	62	554
jFree	Visualization	587	298
Joda-time[a]	Time, date library	166	7.128
Jopt simple	CMD line management	50	293
Lucene	Text management	731	6.031
Plexus[a]	Software development	83	3.570
POI	File management	1196	1.036
scala xml	XML management	123	614
slf4j[a]	Logging	29	21.484
Slick 2D	Visualization	431	274
Spring framework[a]	Web development	276	240
Struts	Web development	750	134
Superfly	User authentication	20	248
WiQuery[a]	Web development	66	24
Wiring	Microcontrollers	76	5.498
Wro4j[a]	Web development	243	127
Xstream	XML management	325	1.732
Acyclic	Code analyzers	3	15
Ant[a]	Build tools	80	479
Apache_Commons_Pool	Objects pool	28	500
Apache_FtpServer_Core[a]	FTP clients and servers	57	11
Bean_Scripting_Framework	JVM languages	20	59
BoneCP_Core_Library	JDBC pools	12	114
Common_Eclipse_Runtime	OSGI frameworks	26	12
Concurrent	Concurrency libraries	47	14
Dagger	Dependency injection	15	29
Disk_LRU_Cache	Cache implementations	1	96
EclipseLink_JPA	JPA implementations	119	32
Exlipse_Xtend_Runtime_Library	Language runtime	12	26
Expression_Language	Expression languages	28	1
Grgit	Git tools	17	10

(Continues)

TABLE 5 (Continued)

Project	Goal	Size (NOC)	Maven Reuse (MR)
Hessian	Distributed communication	72	61
HtmlUnit	Web testing	123	60
Java_Native_Access	Native acess tools	55	3
JavaMail_API	Mail clients	253	24
Jaxen	Xpath libraries	57	47
JBoss_AOP_Framework	Aspect oriented	200	16
Jboss_Logging_Programming_Interface	Logging frameworks	11	51
Jclouds_Compute_Core	Cloud computing	29	64
Jedis	Redis clients	13	1
Jettison	JSON libraries	16	13
Jmock	Mocking	24	291
JSch	SSH libraries	61	175
JSP_API	Java specifications	45	762
JUnitBenchmarks	Microbenchmarks	15	9
Liferay_Util_Taglib[a]	JSP tag libraries	71	209
Metrics_Core	Application metrics	8	255
Mule_Core	Enterprise integration	898	102
Neko_HTML	HTML parsers	12	43
Nimbus_JOSE_JWT	Encryption libraries	95	35
OAuth2_For_Spring_Security	OAuth libraries	126	1
Okio	I/O libraries	5	31
ORMLite_Android	Object/relational mapping	50	79
Pig	Hadoop query engines	293	10
ReflectASM	Reflection libraries	1	164
REST_Assured	Testing frameworks	51	122
Scala_Actors_Library	Actor frameworks	10	15
Scala_Parser_Combinators	Parser generators	7	6
Scopt	Command line parsers	3	1
SLF4J_JCL_Binding	Logging bridges	5	34
Snappy_Java	Compression libraries	10	23
StAX_API[a]	XML processing	37	536
SwitchYard_Plugin	Maven plugins	6	67
The_MongoDB_Asynchronous_Driver	MongoDB clients	29	29
Unirest_Java	HTTP clients	17	183
Value	Annotation processing tools	5	2

[a]The projects with an asterisk have been used as the training set for the analysis. Their selection was random, although an equal distribution between highly reused and less reused projects was targeted. However, this cannot be supported with evidence, because there is no clear threshold for separating highly from low reused assets.

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[28] created to facilitate empirical studies.
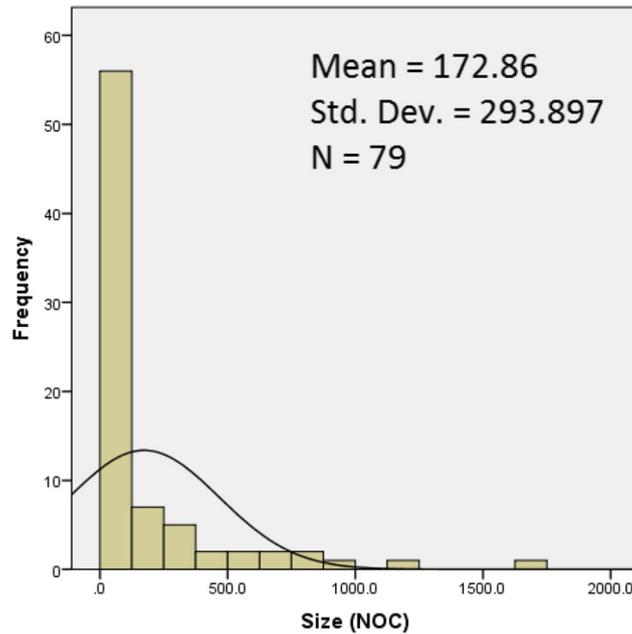
FIGURE 2 Descriptive statistics on the size of the sample projects

4.5 | Data analysis

The collected variables (see previous section) will be analyzed against the five criteria of the 1061 IEEE Standard (see Section 4.9 of the 1061 IEEE Standard) as shown in Table 5. As a pre-step to the analysis, we perform backward linear regression to extract an equation that will be able to calculate the *REI index* (see Section 3.3). We note that for the training of REI we have used 15 projects, whereas the rest 65 were used for testing purposes. The use of backward regression will guarantee that nonsignificant variables are excluded from the equation. Next, to answer each research question, 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 well-known standards for each performed analysis. In particular, regarding:

- *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 (ie, mean standard error). While investigating predictability, we produced a separate linear regression model for each assessor (univariate analysis).

- *Correlation, consistency, and tracking* we use the correlation coefficients (coeff.) and the levels of statistical significance (sig.). 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* is represented as 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 three mutually exclusive categories (representing low, medium, and high metric values) adopting equal frequency binning.[29] Then, Bayesian classifiers[30] 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 (ie, frequently reused and seldomly reused). The extent to which the results on the projects are in agreement (eg, is the same metric the most valid assessor asset reusability for both types?) represents the reliability of the considered index.

5 | RESULTS

In this section, we present the results of the empirical validation of the proposed REI. 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. Finally, in Section 5.2, we summarize the results of RQ_2, ie, the assessment of REI reliability. We note that in this section we present the raw results of our analysis and answer the research questions. Any interpretation of results and implications to researchers and practitioners are collectively discussed in Section 6.

5.1 | RQ₁—Correlation, consistency, tracking, predictive, and discriminative power of REI

In this section, we answer RQ₁ by comparing the relation of REI, QMOOD_R, and FWBR to actual reuse, in terms of correlation, consistency, predictive, and discriminative power. The results on correlation, consistency, and predictive power are cumulatively presented in Table 6. The rows of Table 6 are organized/grouped by validity criterion. In particular, for every group of rows (ie, criterion), we present a set of success indicators. For example, regarding predictive power, we present three success indicators, ie, R-square, standard error, and significance of the model.[26] Statistically significant results are denoted with italic fonts.

Based on the results presented in Table 6, REI is the optimal assessor of software asset reusability, because: (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 = 0.532). 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.[30] Through Bayesian classifiers, we tested the ability of REI to correctly classify software assets in three classes, with respect to their reuse (see Section 4.5). The accuracy of the classification is presented in Table 7, through three well-known success indicators: namely precision, recall, and f-measure.[26] Precision quantifies the positive predictive power of the model (ie, TP/ (TP + FP)), and recall evaluates the extent to which the model captures all correctly classified artifacts (ie, TP/ (TP + FN)). F-measure is a way to synthesize precision and recall in a single measure, because 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 two-fold cross validation.[30]

By interpreting the results presented in Table 7, 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 with the QMOOD REI 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 | RQ₂—Reliability of of REI

In this section, we present the results of evaluating the reliability of the three indices that we have investigated. To assess reliability, we split our dataset into two subsets: the first containing frequently reused artifacts, whereas the second seldomly reused artifacts. All the tests discussed in Section 5.2 are replicated for both sets, and the results are compared. The outcome of this analysis is outlined in Table 8, which is organized by validity criterion. For each validity criterion, we present all success indicators for both frequently (F) and seldomly (S) reused artifacts. With italic fonts, we denote statistically significant results.

TABLE 6 Correlation, consistency, and predictive power

Validity Criterion	Success Indicator	REI	QMOOD_R	FWBR
Predictive power	R-square	38.4%	3.2%	2.4%
	Standard error	5298.11	6270.36	6811.23
	Significance	*0.09*	0.31	0.41
Correlation	Coefficient	0.625	0.198	-0.148
	Significance	*0.00*	0.22	0.38
Consistency	Coefficient	0.532	0.378	0.035
	Significance	*0.01*	*0.08*	0.79
Tracking	AVG (coefficient)	*0.425*	*0.233*	0.042
	%of cases sig < 0.05	*75%*	*55%*	35%

TABLE 7 Discriminative power

Success Indicator	REI	QMOOD_R	FWBR
Precision	60%	44%	38%
Recall	70%	40%	14%
F-measure	64%	41%	20%

TABLE 8 Reliability

Validity Criterion	Asset Type	Success Indicator	REI	QMOOD_R	FWBR
Predictive power	F	R-square	37.0%	5.3%	7.6%
		Significance	0.00	0.33	0.25
	S	R-square	68.9%	18.4%	1.4%
		Significance	0.01	0.33	0.92
Correlation	F	Coefficient	0.551	0.192	-0.216
		Significance	0.00	0.38	0.24
	S	Coefficient	0.795	0.344	0.11
		Significance	0.01	0.37	0.87
Consistency	F	Coefficient	0.555	0.282	-0.132
		Significance	0.02	0.25	0.42
	S	Coefficient	0.576	0.271	0.243
		Significance	0.20	0.85	0.72
Discriminative power	F	Precision	55%	32%	18%
		Recall	55%	30%	25%
		F-measure	55%	31%	21%
	S	Precision	82%	22%	29%
		Recall	75%	38%	38%
		F-measure	78%	27%	32%

The results of Table 8 suggest that in most of the cases, the reusability indices are more accurate in the group of seldomly reused artifacts rather than the frequently reused. Concerning reliability, REI has been validated as a reliable metric regarding correlation, consistence, predictive, and discriminative power. In conclusion, compared with 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 with the other indices.

6 | ILLUSTRATIVE EXAMPLE

In this section, we present two illustrative examples and the lessons that we have learned from using the proposed metrics for white-box reuse purposes. In particular, we examined (a) the reuse process that was introduced by Lambropoulos et al,[10] in which a mobile application was developed, based to the best possible extent on open-source software reuse; and (b) replicate the reuse process that was introduced by Ampatzoglou et al[4] for providing an implementation of the Risk game, based on reusable components. The first project was a movie management application (see Figure 3) that reused artifacts from eight open-source software systems from the same application domain, whereas the second project was a re-implementation of the famous Risk game that reused artifacts from four open-source software systems.

In total, out of 25 requirements for both examples, the corresponding software engineer reused code while implementing 11 of them (44%). To perform the artifact selection process for these reuse activities, in total 138 candidates for reuse have been compared. As mentioned in the introduction, these candidate components are functionally equivalent, ie, offer exactly the same functionality (or more accurately a minimum common functionality). To compare the reuse candidates, the software engineer used expert opinion and ended up in the most fitting artifacts for the target application. The goal of these illustrative examples was to compare the ability of the three examined metrics (ie, REI, FWBR, QMOOD) to subsume the expert's opinion and guide reuse artifact selection. We note that this analysis is a posteriori one; the decisions of Lambropoulos et al[10] and Ampatzoglou et al[4] were not considering these metrics, but were only taken based on the subjective opinion of the developers. Out of the 11 reuse activities, only eight of them were involving the reuse of more than one alternative for reuse (ie, in the rest: the decision was only between to reuse or to build from scratch). The reused components were either application-domain specific entities (eg, Movie, Country, Continent entities, etc.) or platform-specific (eg, an Android pager, recycler view, etc.). The components that have been reused out of these activities were Retrofit, RecyclerView, Pager, and the Movie Entity (for the Movie app) and the Country, Continent, Map, and Resources (for the Risk game). Some demographics on the reuse candidates (only the eight for which reuse alternatives existed) and actually reused classes are presented in Table 9. The reusable assets have been identified through the Percerons Client that has been populated with OSS projects from the domains of movie management and risk games. We remind that Percerons components database is populated with sets of classes that are independent chunks of code built around a central class, providing the target functionality.[28] The most common reason for the rejection of a reusable asset for reuse was architectural nonconformance. For example, the mobile application was targeting the use of the Model-View-Presenter pattern.[10] Therefore, any reusable asset that was identified, but was not fitting the target architecture, was rejected, although it offered the required functionality, and it might have been of appropriate quality.
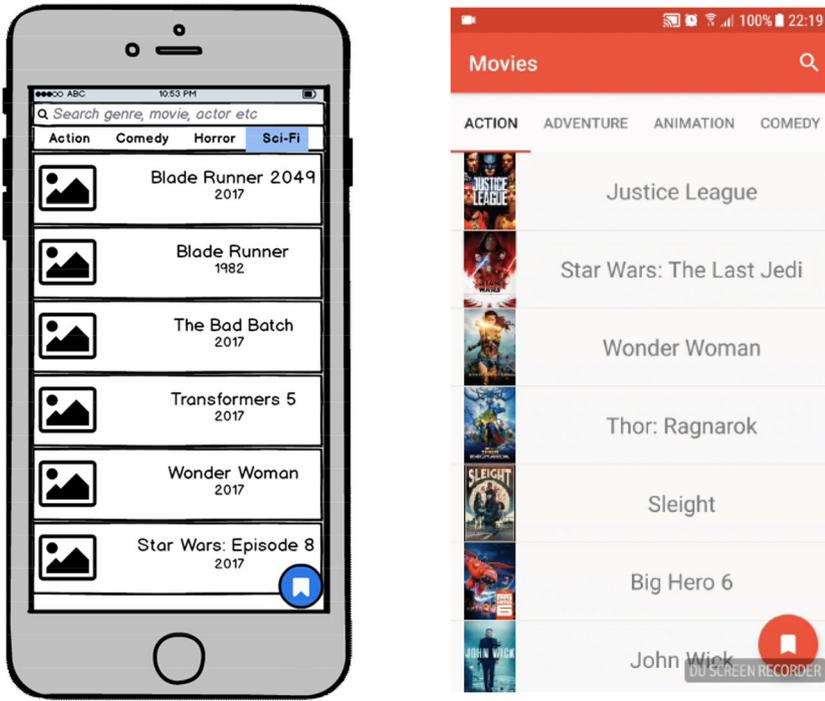
FIGURE 3 Movie management mock-up and final product

TABLE 9 Illustrative example reuse demographics

Reused Components	#Projects with Reuse Candidates	#Classes Reused	#Classes Rejected for Reuse
Retrofit	8	1	10
RecyclerView	7	1	8
Pager	8	12	31
Movie entity	6	1	7
Country	4	2	11
Continent	4	4	13
Resources	4	5	28
Map	4	1	3
Total	12[a]	27	111

[a]Some projects in the rows are duplicated.

To investigate the ability of metrics to discriminate between reused and not-reused classes, we have performed (a) Spearman correlation analysis and (b) independent sample t-test analysis, separately for each requirement. The results for Spearman correlation are presented in Figure 4. Based on the bar charts of Figure 4, we can observe that REI is consistently strongly correlated to the decision of the expert (mean value: 0.440, standard deviation: 0.06), whereas FWBM (proposed by Kakarontzas et al[17]) which is in two cases very close to REI (in one case produced better results) is showing larger deviation (mean value: 0.336, standard deviation: 0.13). In particular, we can observe that in two cases (retrofit and
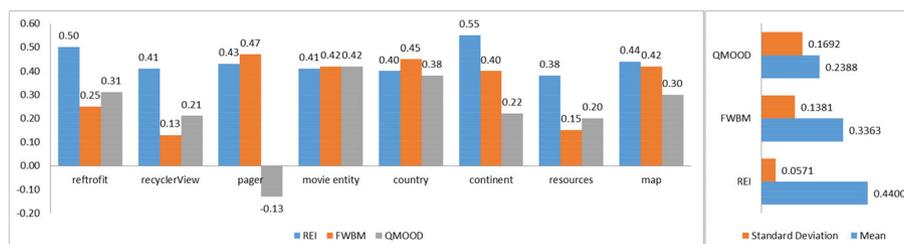

FIGURE 4 Ability of metrics to guide reuse process

recyclerView) REI was the only metric strongly correlated to the expert's opinion. In four cases (ie, Pager, Continent, Country, and Map), REI and FWBM were both strongly correlated, for another (Movie Entity) all three metrics were strongly correlated, whereas in one case (Resource) there was no strong correlation.

To perform the independent sample t-tests, the complete dataset has been treated as a whole, because the group of reused classes was having only one member for some requirements. The results of the analysis suggested that both REI and FWBM are able to discriminate between reused and non-reused classes (sig. < 0.05) in the Mann-Whitney test. Based on the aforementioned analysis in the illustrative example data, we can suggest that REI is able to guide the artifact selection to reuse process. Nevertheless, this result should be treated with caution in the sense that it is only based on two illustrative examples, of small size, and is based on the expert judgment of one developer in each case.

7 | DISCUSSION

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

7.1 | Interpretation of results

The validation of the proposed REI on 80 open source software assets suggested that the associated REI is capable of providing accurate reusability assessments. REI outperforms the other examined indices (ie, 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 with state-of-the-art indices, is the fact that it synthesizes both structural aspect of quality (eg, source code complexity metrics) and nonstructural quality aspects (eg, documentation, correctness, etc.). This finding can be considered intuitive in the sense that nowadays, software development produces a large data footprint (eg, commit records, bug trackers, and 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 internal quality attributes (eg, cohesion, complexity, etc.—quantified through source code structural metrics), the results of this study highlight the importance of evaluating nonstructural artifacts, while assessing reusability. More specifically, the contribution of the different types of characteristics to the REI index is explained as follows:

- *Low-level structural characteristics* (complexity cohesion, and size in classes). Low-level structural quality attributes (ie, *cohesion* and *complexity*) 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 (ie, 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 (ie, 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* (ie, 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 because while reusing a software asset developers are usually not interfering with internal 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 reused 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 subcomponents. This information is important while assessing reuse in the sense that modular software is easier to understand and modify.

- *Nonstructural 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. This number is indicative of the maturity of the asset and also the level of user interest that this asset concentrates. 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[23]—see Section 3), only two are not directly participating in the calculation of REI (ie, 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, ie, by borrowing equations from the QMOOD model, were suboptimal, or (b) the metrics are subsumed by the other structural quality metrics that participate in the calculation of REI. In particular, in the literature it is very frequently mentioned that LCOM, WMC, and 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.

Finally, regarding the weights that are assigned to each metric based on the linear regression, we need to note they are not restrictive to the software engineers. In particular, the goal of this paper is not to prevent the decision maker from fine-tuning the weights of each variable, but to provide a default model, trained and tested in a substantial corpus of reusable assets, and demonstrated through proof of concept studies. Nevertheless, we highlight that the software engineer is free to fine-tune the model if he/she wishes to do so and validate the model in more specialized data. However, in such a case, the validation that is performed as part of this manuscript is not applicable.

7.2 | Implications to researchers and practitioners

By taking into consideration the aforementioned results in this section, we summarize implications to researchers and practitioners. The major findings of this study show that reusability models/indices for assessing the quality of a software asset need to further concentrate on the inclusion of nonstructural factors. The following implications to researchers and practitioners have been derived:

- *(research)* An interesting extension for this study would be the cross validation of the model by trying different partitionings of the dataset (in training and testing sets), so as to enhance the reliability of the obtained results. However, such an attempt would as a side-effect produce multiple models (one from each partitioning) and would require further investigation on the circumstances that constitute each model more fitting. Therefore, we believe that this is a very intersting extension of this work.

- *(research)* Introduce formal metrics and procedures for quantifying quality aspects that till now are measured adopting ad-hoc procedures. Attributes like Documentation, External Quality, and Availability are underexplored and usually measured subjectively. More formal definitions of these factors could further increase the accuracy and adoption of reusability metrics

- *(research)* Evaluate the proposed reusability model on inner source development (see Stol et al[31]). 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 (eg, 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, because for in-house components, it is not possible to obtain an external, objective reusability measure such as Maven reusability (MR).

- *(research)* Further validate REI with the effort required to adopt the asset in a fully operating mode in new software. Clearly, it is important to select the right asset that will require less time, effort, cost, and modifications while being reused.

- *(research)* Similarly to every empirical endeavor, we encourage the replication of REI validation in larger samples of reusable assets, examining different types of applications. Through these replications, the research community will test the reliability of REI and lead to additional accuracy enhancements.

- *(practice)* The proposed REI 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.

- *(practice)* 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.

- *(practice)* The two-fold analysis that we adopt in this paper (ie, prediction and classification) enables practitioners to select the most fitting one for their purposes. In particular, the classification of assets to low, medium, and high reusable one, 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.

8 | THREATS TO VALIDITY

In this section, we discuss the threats to validity that we have identified for this study based on Runeson and Höst[19] classification schema that considers construct, internal, external, and reliability validity.

Construct validity defines how effectively a study measures what it intends to measure. In our case, this 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 internal and external quality, such as adaptability, maintainability, quality, availability, documentation, reusability, and complexity, each of which synthesized by the values of 12 metrics as depicted in Figure 1. Furthermore, as already mentioned in Section 3.2, the selected metrics are established metrics for the respective factors, although we do not claim they are the most optimal ones. *Internal validity* is related to the examination of causal relations. Our results pinpoint particular quality 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, coming from different application domains or contexts, could cause differentiations in the results. Still this risk is mitigated by the fact that the analysis was performed by selecting a pool of diverse projects that are quite well-known and popular in the practitioners community[27] 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. Additionally, we note that the method is only validated with open-source components, and its validation with in-house components is still pending. Nevertheless, the method is not applicable to third-party components, because their prior evaluation might not be possible. Regarding the reproducibility of the study known as *Reliability*, we believe that the followed 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 authors.

9 | 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 REI, which decomposes reusability to seven quality factors quantifying each one of them with certain metrics. Nonstructural metrics along with low-level and high-level structural metrics synthesize the proposed REI. 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.[25] The results from the holistic multiple-case study on 80 OSS projects suggested that REI is capable of providing accurate reusability assessments. REI outperforms the other examined indices (ie, QMOOD_R and FWBR) and presents significant improvement in terms of estimation accuracy and classification efficiency. Based on these results, implications for researchers and practitioners have been provided.

ORCID

Ioannis Zozas https://orcid.org/0000-0003-2159-1332
Apostolos Ampatzoglou https://orcid.org/0000-0002-5764-7302
Stamatia Bibi https://orcid.org/0000-0003-4248-3752
Alexander Chatzigeorgiou https://orcid.org/0000-0002-5381-8418
Paris Avgeriou https://orcid.org/0000-0002-7101-0754

REFERENCES

1. Baldassarre M. T., Bianchi A., Caivano D., Visaggio G. An industrial case study on reuse oriented development. 21st International Conference on Software Maintenance (ICSM'05). IEEE Computer Society. 2005.

2. Morisio M., Romano D., Stamelos I. 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.

3. Mili H., Mili F., Mili A., Reusing software: issues and research directions. IEEE Transactions on Software Engineering. 21 6. pp. 528-562. 1991.

4. Ampatzoglou A., Stamelos I., Gkortzis A., Deligiannis I. 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.

5. Arvanitou EM, Ampatzoglou A, Chatzigeorgiou A, Galster M, Avgeriou P. A mapping study on design-time quality attributes and metrics. *J Syst Softw.* 2017;127:52-77.

6. Ampatzoglou A, Bibi S, Chatzigeorgiou A, Avgeriou P, Stamelos I. Reusability index: a measure for assessing software assets reusability. In: Capilla R, Gallina B, Cetina C, eds. *New Opportunities for Software Reuse, vol 10826*. ICSR 2018. *Lecture Notes in Computer Science*. Cham: Springer; 2018:43-58.

7. McIlroy, D. Mass-produced software components. NATO Conference. pp. 88–98. 1968.

8. Leach R. *Software Reuse: Methods, Models and Costs (2nd edition)*. New York. NY. USA: McGraw-Hill Inc.; 2011:131-165.

9. Krueger, C. Software reuse. ACM Comput. Surv. 24, 2. pp. 131-183. June. 1992.

10. Lambropoulos A., Ampatzoglou A., Bibi S., Chatzigeorgiou A., Stamelos I., REACT: a process for improving open-source software reuse. QUATIC. 11th International Conference on the Quality of Information and Communications Technology. IEEE Computer Society. 2018.

11. Jalender B, Gowtham N, Kumar KP, Murahari K, Sampath K. Technical impediments to software reuse. *Int J Eng Sci Technol*. 2010;2(11):6136-6139.

12. Tripathy P, Naik K. *Reuse and Domain Engineering. Software Evolution and Maintenance*. Hoboken, NJ: John Wiley & Sons. ISBN 978-0-470-60341-3; 2014:325-357.

13. Sojer M, Henkel J. *Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments*. Rochester: Social Science Research Network; 2011.

14. Bansiya J., Davis C. G. A hierarchical model for object-oriented design quality assessment. IEEE Transactions on software engineering. 28. 1. pp. 4-17. 2002.

15. Nair TRG, Selvarani R. Estimation of software reusability: an engineering approach. *SIGSOFT Softw Eng Notes*. 2010;35(1):1-6.

16. Chidamber S. R., Kemerer C. F. A metrics suite for object oriented design. IEEE Transactions on software engineering. 20 6. pp. 476-493. 1994.

17. Kakarontzas G, Constantinou E, Ampatzoglou A, Stamelos I. Layer assessment of object-oriented software: a metric facilitating white-box reuse. *J Syst Softw*. 2013;86(2):349-366.

18. Sharma A, Grover PS, Kumar R. Reusability assessment for software components. *SIGSOFT Softw Eng Notes*. 2009;34(2):1-6.

19. Runeson R, Host M, Rainer A, Regnell B. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons; 2012:119-162.

20. Washizaki H., Yamamoto H., Fukazawa Y. A metrics suite for measuring reusability of software components. Software Metrics Symposium, 2003. Proceedings. Ninth International. IEEE. 2003.

21. Fazal, E. A., Mahmood, A. K. Oxley, A. An evolutionary study of reusability in Open Source Software. Computer & Information Science (ICCIS), 2012 International Conference on 12-14 June 2012. pp. 967-972. 2012.

22. Goel BM, Bhatia PK. Analysis of reusability of object-oriented systems using object-oriented metrics. *SIGSOFT Softw Eng Notes*. 2013;38(4):1-5.

23. Hristov D. Structuring software reusability metrics for component-based software development. ICSEA. The Seventh International Conference on Software Engineering Advances. 2012.

24. Martin RC. *Agile Software Development: Principles, Patterns and Practices*. New Jersey: Prentice Hall; 2003:85-135.

25. IEEE Standard for a Software Quality Metrics Methodology. IEEE Standards 1061-1998. IEEE Computer Society. 31 December 1998 (reaffirmed 9 December 2009).

26. Field A. *Discovering Statistics Using IBM SPSS statistics*. SAGE Publications Ltd; 2013:354-356.

27. Constantinou E., Ampatzoglou A., Stamelos I. Quantifying reuse in OSS: a large-scale empirical study. International Journal of Open Source Software and Processes (IJOSSP). 5 3. pp. 1-19. 2014.

28. Ampatzoglou A., Gkortzis A., Charalampidou S., Avgeriou P. 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. Baltimore. USA. Octomber. 2013.

29. Bibi S., Ampatzoglou A., Stamelos I. A Bayesian belief network for modeling open source software maintenance productivity. 12th International Conference on Open Source Software Systems (OSS). Springer. pp. 32-44. 2016.

30. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten I. The WEKA data mining software: an update. *ACM SIGKDD explo news*. 2019;11(1):10-18.

31. Stol KJ, Avgeriou P, Ali Babar M, Lucas Y, Fitzgerald B. Key factors for adopting inner source. *Trans Softw Eng Methodol*. 2014;23(2):18.