Empirical Evaluation of Reusability Models

Andreea Cristina Lung, Simona Motogna[a] and Vladiela Petraşcu[b]
Babes-Bolyai University, Department of Computer Science, M. Kogalniceanu Street, Cluj-Napoca, Romania

Keywords: Software Quality, Reusability, Empirical Study.

Abstract: Many research efforts have been directed into ways to quantify the degree to which a software component can be reused in other systems. As such, a number of different reusability assessment models have been proposed in the literature, taking into account several metrics that can affect reusability and different approaches to measuring it. In this paper, we conduct a longitudinal reusability assessment by applying three reusability models to a number of different projects (libraries and frameworks) with the goal of studying the long-term evolution of reusability in open-source software. The exploratory part of the study consists of reproducing and applying the chosen models on three different-sized projects for several released versions of the software and studying the transformations of reusability over time and how these relate to certain changes in quality factors or size of the software. Results show a more intense variation of reusability in earlier versions and more stable values towards later versions in applications, and a clear influence of complexity, modularity and cohesion on reusability scores.

1 INTRODUCTION

In software development, reusability established itself as a vital aspect seeked by many organizations or individual developers in order to speed up their development times, increase productivity and boost performance of their teams. Reusability also has a positive impact on maintainability and overall quality of the final products. Regardless if the used code is an external library or a dependency for the project or a simply inserted third party component into a project, it is clear that software reuse is heavily used in one form or another.

Open-source code sharing platforms include libraries, frameworks and other assets easy to be reused. Finding the right component to integrate in a software system, on one hand, and preserving the reuse potential of developed code, on the other hand, might be challenging tasks. As a result, measuring the reusability level and monitoring it becomes an important aspect of such software systems.

In order to correctly quantify reusability, it is important to differentiate it from reuse. Literature defines *reuse* as the use of existing artifacts or knowledge to create new software (Frakes and Terry, 1996), and *reusability* as the extent to which a program can be reused in other applications (McCall et al., 1977) or, more generally, the degree to which an asset can be used in more than one system, or in building other assets (ISO/IEC 25010, 2011). While reuse has established itself as an activity that brings many savings to software development, reusability metrics aim to identify the assets that will facilitate those savings (Poulin, 1994).

Especially for software systems that are specifically created for reuse purposes, keeping a track record of how reusability evolves across the development of the project can become a very tedious, but significant task, as any moment of low consideration towards software quality could lead to a decrease in the reusability level, hence decreasing the re-use potential of the library or framework altogether and affecting how people who use it interact with it or driving away new potential users of that library. For this reason, factors affecting reusability should also be tracked during development.

Therefore, keeping the reusability level under control should be a goal that any such system should strive to achieve, and we aim to facilitate this process and provide a starting guideline through the current empirical study. We aim to investigate how reusability evolves in large software systems over significant periods of time, how it is influenced by characteristics of the code and which is the relation with the main-

[a] https://orcid.org/0000-0002-8208-6949
[b] https://orcid.org/0000-0002-7878-4349

Lung, A., Motogna, S. and Petraşcu, V.
Empirical Evaluation of Reusability Models.
DOI: 10.5220/0011143100003266
In *Proceedings of the 17th International Conference on Software Technologies (ICSOFT 2022)*, pages 265-275
ISBN: 978-989-758-588-3; ISSN: 2184-2833

tainability of the system.

The core contribution of our work lays in the fact that we study reusability in the context of software evolution, namely by investigating several versions of the software systems. The second contribution lies in the analysis of several reusability estimation models across the whole release cycle of the case study projects.

The rest of the paper is organized as follows. We start with presenting related work, in order to state the contribution brought by our investigation. In Section 3, we explain how we design our exploratory case study, formulating the research questions and describing the setup of the case study. Section 4 presents our findings in form of responses to the research questions and draw some lessons learned. Then, we identify the main threats to validity and discuss how we mitigate them. We end by drawing some conclusions and designing future research directions.

2 RELATED WORK

Reusability has been a research topic since its introduction in the first software quality model, proposed by McCall in 1977 (McCall et al., 1977). Several models for estimating the degree to which a software asset can be reused have been proposed and a number of comparative studies address the differences between them.

In (Poulin, 1994), the author summarizes various existing reusability estimation approaches and associated metrics, based on a taxonomy distinguishing between empirical and qualitative ones. While empirical approaches use objective, quantifiable software attributes as a basis for reusability metrics, qualitative ones rely on offering guidelines and assigning reusability estimators to components, based on compliance to those guidelines. The paper emphasizes the fact that, despite the considerable amount of approaches proposed in the literature - 11 empirical and 3 qualitative ones being considered in the study - work remains to be done in the area of introducing domain attributes and contextual information (along with internal attributes) as a basis for reusability metrics.

The authors of (Gui and Scott, 2009) propose new coupling and cohesion metrics supporting the ranking of software components extracted from the Internet by a search engine, according to their reusability level. Reusability is understood in terms of the adaptation effort (NLOC added/deleted/modified) required to integrate a component into a larger system. The novelty of their proposal lies in the fact that they take into account both the transitive nature of coupling/cohesion (indirect coupling/cohesion) and the functional complexity of software components. The performance of the newly proposed metrics in predicting reusability is evaluated using both linear regression and Spearman rank correlation approaches, on three software systems, each consisting of 20-25 components. In all cases, each of the new metrics performs better than the existing four it is compared against, showing the feasibility of the proposal.

In (Mijač and Stapic, 2015), the goal is to conduct a systematic literature survey in order to discover relevant reusability metrics for software components. A total of 39 papers have been investigated and summarized, leading to the identification of 36 quality factors influencing reusability and 37 reusability metrics (12 for black box components and 25 for white/glass box components).

The authors of (Ampatzoglou and Stamatia, 2018) propose a new reusability predictor for software assets, called Reusability Index (REI), that aggregates various metrics corresponding to both structural and external quality factors. REI is derived using backward regression, taking into account 7 such factors and associated metrics. The new measure is validated against two existing proposals for assesing reusability, using a dataset consisting of 15 projects extracted from the Maven repository, both libraries and frameworks. The ground-truth value is represented by the number of times a given asset has been reused, as reported by the Maven statistics.

Our study brings new elements to the body of knowledge in this domain by: (i) conducting an inspection of reusability evolution in software systems, by analysing how it changes in consecutive versions of these systems; (ii) comparing different reusability models by means of empirical investigation, namely exploratory case study.

3 CASE STUDY DESIGN

3.1 Research Goals

Applying a Goal-Question-Metric approach (Basili et al., 1994), the purpose of our study can be stated as follows:
Analyze *long term evolution of reusability*
using *three reusability computation models*
for the purpose *of evaluating them and their dependencies to software metrics*
from *the source code* ***point of view***
in the context of *several versions of large Java applications.*

This general objective was then formulated into research questions:

RQ1: *How does reusability evolve over long term?* We investigate if and how the reusability indexes vary during the evolution of the project, starting from early versions of the applications.

RQ2: *Which factors have an impact on reusability?* Different models use several metrics associated with quality factors. We inquire which of these factors makes a bigger impact on reusability. We call these factors internal, since they are involved in the analyzed computational models of reusability.

RQ3: *Which is the relation between reusability and maintainability?* Maintenance being one of the most time consuming activities in software lifecycle, the relation between reusability and maintainability has been intensively studied (Lee and Chang, 2000; Henry and Lattanzi, 1994). Even more, the current software quality standard ISO 25010 (ISO/IEC 25010, 2011) considers reusability as a subcharacteristic of maintainability. As a consequence, we are interested in how this relation is reflected in reusability models.

The investigation has been designed as an exploratory case study, since our intention is to evaluate the hypothesis formulated in the research questions and has been developed using adopted standard in the domain (Ralph, Paul (ed.), 2021) and similar practices described in literature (Kitchenham et al., 1995; Runeson and Höst, 2008).

3.2 Case study setup

To investigate the dependencies and differences between reusability models, we designed an exploratory case study in which three such models were applied to a set of three projects.

***Selection of Reusability Estimation Models*:** We have considered the following criteria when selecting a model as part of our case study:

- provides a numerical assessment of reusability, with clear calculation instructions;
- presents a solid proof of validity and has been tested on a generous benchmark to confirm its reliability;
- the metrics used are easily available and computable by existing tools.

The candidate models, means of computation and excluding criteria are presented in Table 1.

The final selection contains: the model proposed by the authors of (Papamichail et al., 2019) - acronymed by us as PDS, the Taibi model introduced in (Taibi, 2014), and QMOOD (Quality Model for Object-Oriented Design) - presented in (Bansiya and Davis, 2002).

The **PDS** model estimates reusability by aggregating six quality factors (cohesion, coupling, complexity, size, inheritance and documentation) and associated metrics, as shown in Table 2. For each of the metrics, a polynomial regression model is trained, translating its value into a reusability score. The ground-truth value is taken as the number of occurrences within the import statements of a chosen set of 3000 GitHub projects, extracted using an in-house tool called Agora. Below, are the steps we followed in implementing the PDS model, following the general method proposed by the authors:

1. Setup Agora locally, then download the set of 100 projects used as benchmark and the 3000 projects used for the ground-truth value;
2. For each of the 100 benchmark projects, compute all static analysis metrics considered, at class level, using SourceMeter;
3. For each class within each of the 100 benchmark projects, retrieve its reuse rate, as the number of times its name appears in the import statements of the 3000 projects dataset;
4. Eliminate the classes with 0 reuse rate (they are mostly private and cannot provide relevant information);
5. Extract a general distribution for each metric, using a binning strategy (a number of bins of fixed size is chosen, each containing all the classes having the value of the metric within the bin boundaries);
6. Assign a reusability score to each bin, by summing up the reuse rates of all classes belonging to the bin;
7. Normalize the scores from the previous step between 0 and 1;
8. Create the polynomial regression model corresponding to each metric (that translates its value into a reusability score), using pairs of data of the shape [BinCenter, AssociatedReusabilityScore]. The optimal degree for the polynomial is obtained by using the elbow method on the Root-Mean-Square-Error (RMSE);
9. Compute the reusability score corresponding to each of the six quality factors, by aggregating the scores of its associated metrics, using the formula (Papamichail et al., 2019)

$$Score_{factor} = \frac{\sum_{i=1}^{N} w_i * Score_{metric_i}}{\sum_{i=1}^{N} w_i}.$$

   Above, N is the number of metrics used for the quality factor considered, $Score_{metric_i}$ is the reusability score predicted by the regression model for $metric_i, i = \overline{1,N}$ and w_i is the weight corresponding to $metric_i$ (computed as the Pearson correlation between the values of the metric and the associated reuse rates).
10. Compute the final reusability score of the component, by aggregating the scores corresponding to all factors

Table 1: Synthesis of Reusability Estimation Models.

Paper describing the model	Computation	Excluding criteria
Reusability Index: A Measure for Assessing Software Assets Reusability (Ampatzoglou and Stamatia, 2018)	Backwards Linear Regression	Calculation only at component and asset level
Measuring the Reusability of Software Components using Static Analysis Metrics and Reuse Rate Information (Papamichail et al., 2019)	Hierarchical approach, polynomial regression	**accepted**
A Reusability Evaluation Model for OO-Based Software Components (Sandhu and Singh, 2008)	Neuro-fuzzy inference engine	Only structural quality factors considered
Estimation of Software Reusability for Component based System using Soft Computing Techniques (Singh et al., 2014)	Neural Networks	Requires implementation of neural network in Matlab
Empirical Analysis of the Reusability of Object-Oriented Program Code in Open-Source Software (Taibi, 2014)	Based on three quality factors	**accepted**
A hierarchical model for object-oriented design quality assessment (Bansiya and Davis, 2002)	Hierarchical, based on empirical and anecdotal info	**accepted**
Estimation of Software Reusability: An Engineering Approach (Nair and Selvarani, 2010)	empirical, weighted combination of polynomials	only structural code quality characteristics

considered in the model, using the formula

$$ReusabilityScore = \frac{1}{M}\sum_{i=1}^{M} Score_{factor_i}.$$

The model proposed by **Taibi** (Taibi, 2014) is based on three factors impacting the reusability of a class: modularity (M), low complexity (LC) and understandability (U). As shown in Table 2, M is estimated based on the CBO and LCOM metrics, LC based on ACC, DIT and NM, while U is computed by aggregating the ROI and CIC metrics. The steps we followed in order to estimate the reusability of a class component are:

1. Run SourceMeter in order to obtain the values of the five metrics quantifying M and LC for the class (namely CBO/LCOM and ACC/DIT/NM, respectively);
2. Apply a similarity-based comparison of class names to code identifiers (for ROI) and comments to identifiers (for CIC), using a pretrained NLP model provided by the Gensim library (Gensim, 2021), in order to asses the values of the two metrics quantifying U;
3. Compute the reusability score F corresponding to each of the three factors (see formula in (Taibi, 2014).
4. Compute the final reusability score corresponding to the class, based on the formula

$$R = \sum_{i=1}^{n} \lambda_i * F_i,$$

where F_i and λ_i, $i = \overline{1,n}$ are the reusability scores corresponding to the considered factors and the tuning parameters, respectively.

Regarding the tuning parameters, a heuristic method has been used, and extensive experiments have been conducted, in order to determine the appropriate values. The best results have been reported for $\alpha_j = 1.5$ for CC, DIT and CBO, $\alpha_j = 1$ for NM and LCOM, $\lambda_i = 0.3$ for U and $\lambda_i = 0.35$ for M and LC (more details can be found in (Taibi, 2014)).

QMOOD is a model used to assess high-level quality factors of object-oriented systems: reusability, flexibility, understandability, functionality, extensibility and effectiveness. Each quality factor is mapped to some design properties, whose quantification relies on appropriate metrics. In case of reusability, the corresponding design properties are coupling, cohesion, design size and messaging, the four associated metrics used to estimate them being listed in Table 2. The reusability score associated to a component is computed using the formula (Bansiya and Davis, 2002)

$$\begin{aligned} Reusability = &-0.25 * Coupling + \\ &0.25 * Cohesion \\ &+ 0.5 * Messaging \\ &+ 0.5 * DesignSize \end{aligned}$$

Our own tool was developed implementing the computational methods corresponding to these models, while the metrics were computed using SourceMeter (SourceMeter, 2022). All the charts from next section have been generated with this tool.

Table 2: Overview of quality factors and metrics used in the analyzed models.

Factors	Metrics		Reusability models		
	Short name	Meaning	PDS	Taibi	QMOOD
Complexity	ACC	Average Cyclomatic Complexity		✓	
	NL	Nesting Level	✓		
	NLE	Nesting Level Else-If	✓		
	WMC	Weighted Methods per Class	✓		
Coupling / Modularity	CBO	Coupling Between Objects	✓	✓	✓
	CBOI	Coupling Between Objects Inverse	✓		
	NII	Number of Incoming Invocations	✓		
	NOI	Number of Outgoing Invocations	✓		
	RFC	Response set For Class	✓		
Cohesion	LCOM5	Lack of Cohesion in Methods 5	✓	✓	✓
Understandability	ROI	Relevance of Identifiers		✓	
	CIC	Correlation Identifiers Comments		✓	
Documentation	AD	API Documentation	✓		
	CD	Comment Density	✓		
	CLOC	Comment Lines of Code	✓		
	DLOC	Documentation Lines of Code	✓		
	PDA	Public Documented API	✓		
Inheritance	DIT	Depth of Inheritance Tree	✓	✓	
Size/Messaging	LOC	Lines of Code	✓		
	LLOC	Logical Lines of Code	✓		
	TNA	Total Number of Attributes	✓		
	NOC	Number of Children			✓
	NM	Number of Methods		✓	
	NPM	Number of Public Methods			✓
	NG	Number of Getters	✓		
	TNOS	Total Number of Statements	✓		

***Selection of Applications*:** The projects on which we have applied the previously presented models are: **JUnit4** (JUnit4, 2021) - a widely adopted unit testing framework for Java, based on annotations, **Mockito** (Mockito, 2021) - a popular mocking framework for unit tests in Java, and **Atmosphere** (Atmosphere, 2021) - a framework used for building asynchronous Web applications. There are all mature systems, but quite different in size, adoption and popularity levels. Size and reputation indicators of these applications are summarized in Table 3.

Table 3: Details about size and reputation of applications.

	Mockito	**JUnit4**	**Atmosph.**
Lines of code	90,926	47,402	61,841
No of classes	1,970	1,505	899
Versions	605	28	219
Used by	107k	1.8m	134
GitHub stars	8.2k	12.1K	3.5

The research approach consisted in implementing the three mentioned models of reusability applied to a set of versions of the selected applications and investigating, on one hand, the differences in each application, and on the other hand analyzing the differences between the reusability estimation models. In order to get a longitudinal view with respect to the evolution of reusability throughout the lifetime of the selected projects, we have applied the chosen reusability models on 23 versions of JUnit (out of 28), 45 versions of Mockito (out of 605) and 30 versions of Atmosphere (out of 219).

4 RESULTS

This section will offer answers to the formulated research questions based on quantitative analysis of different computed metrics.

***RQ1:** How Does Reusability Evolve over Long Term?* In order to investigate the evolution of reusability indexes throughout selected projects, we computed the reusability indexes based on the three estimation models, for the most significant versions of each se-

lected application: PDS shown in Figure 1, Taibi in Figure 2, respectively QMOOD in Figure 3. We perform a side-by-side comparison of the plotted reusability curves produced by each model across the release version history of the analyzed projects.

Analyzing the reusability evolution graphs, we can observe that there is indeed some correlation between the curves predicted by the 3 models. In particular, we notice a common evolution pattern between the values obtained with the PDS model and the ones obtained with Taibi, for the Atmosphere and Mockito projects. For JUnit4, the two models seem to perform proportionally inverse. PDS values are on an ascending curve, while reusability reported by Taibi seems to decrease sharply at the same point PDS is picking up in value and continues to maintain an almost constant value, with a very slight increase between the following versions. QMOOD seems to yield the most different results out of the 3 models.

We find that there is very little correlation between this model's predicted values and the other two. For the first project, we can observe that QMOOD average values seem to be increasing by each version, whereas PDS and Taibi have classified its reusability as decreasing. We can, however, observe a slight correlation between QMOOD and PDS where the JUnit4 project is concerned. Both models categorize the project's reusability level overall to be increasing, though in the case of QMOOD the angle of the curve is much sharper.

Lesson Learned: There are two conclusions that we can draw from this exploration: Firstly, we can say that the variation tends to appear in earlier versions, and the later versions of the applications become somewhat constant in terms of reusability scores. Secondly, we can conclude that the PDS and Taibi models perform similarly in terms of average reusability evolution tracking, with QMOOD displaying very slight similarity to the other two.

RQ2: *Which Factors Have an Impact on Reusability?*

Table 2 summarizes the quality factors and associated metrics used by the three computation models. We first remark the overwhelming number of metrics considered by PDS compared with the other two models. PDS also includes metrics associated with all factors that have been proven to influence reusability (Mehboob et al., 2021) (nominated in first column of the table). The Taibi model considers complexity, coupling, cohesion, inheritance and size in its computation, while QMOOD is using only four metrics (associated to coupling, cohesion and size/messaging), and ignores complexity and inheritance.

Then, we perform an in depth analysis in order to investigate the relationships between constituent factors and reusability for different models. The tool allows the selection of reusability score corresponding to a model, respectively the factors: cohesion, complexity coupling, documentation, inheritance or size. The strongest influences detected are:

PDS - Complexity - the complexity attribute seems to have the biggest impact on the final reusability curves observed for the PDS model. This is visible in Figure 4. We can observe a very strong correlation between complexity and the final reusability values for each of the projects. However, in the particular case of Atmosphere, the relationship between reusability and complexity is an inverse one, which does not align with the other two projects, for which the relationship is a direct one. Intuitively, however, it seems more natural to have reusability decreasing along with increasing complexity, although this is not the case for the first project.

Taibi - Modularity - we observe a very strong direct correlation between the evolution of modularity and the one of reusability in the Taibi model (see Figure 5).

QMOOD - Messaging (quantified in the model as the Number of Public Methods) - we distinguish a very obvious inverse correlation between the average values of the Messaging attribute and the reusability ones reported by the QMOOD model. Hence, we can say that messaging seems to have a negative influence on reusability (as also noted in Figure 6). We also find Cohesion to have a meaningful impact on the final results obtained for QMOOD, but in a positive manner.

Lesson Learned: Our case studies confirmed previous results from literature, in terms of the impact that complexity, cohesion, coupling, size, inheritance and documentation have on reusability. However, the metrics associated with these characteristics present distinct correlations to reusability.

RQ3: *Which Is the Relation between Reusability and Maintainability?*

We have used *Maintainability Index* - MI (Oman and Hagemeister, 1992) as the measure associated to maintainability. The results of our evaluation are depicted in Figures 7, 8, respectively 9. The most prominent correlation to maintainability is displayed by the QMOOD model, which succeeds in tracking maintainability evolution trends for 2 out of the 3 projects analyzed. The relationship seems to be a strong one and is backed up by the design of the model itself, which uses a number of overlapping metrics for both reusability and maintainability, so a good estimation of maintainability should also translate into a good estimation of reusability.

As far as the other two models are concerned, PDS does not seem to correlate very much to maintainabil-

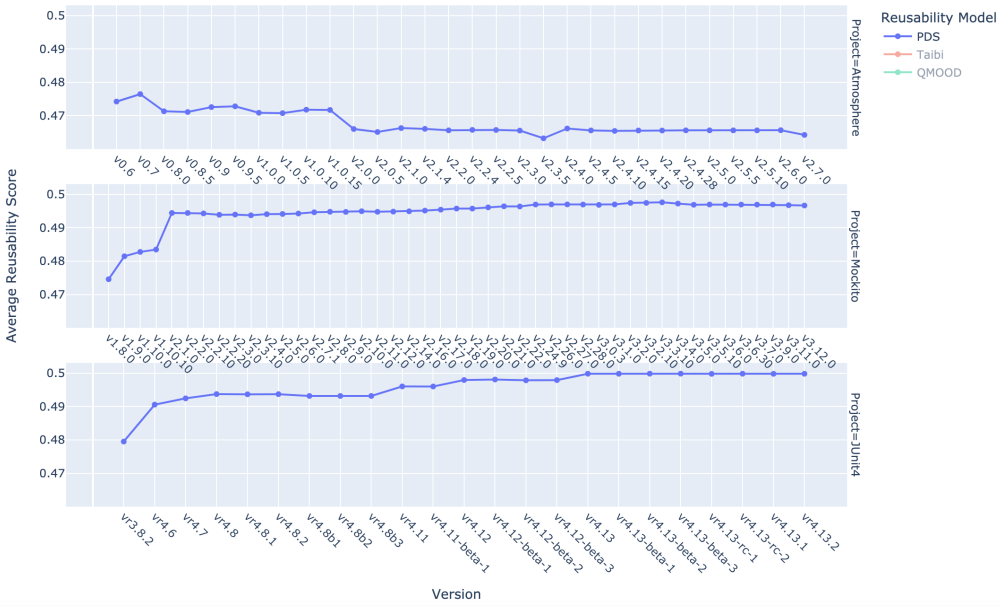

Figure 1: PDS Reusability Evolution (tool view with PDS model selected).

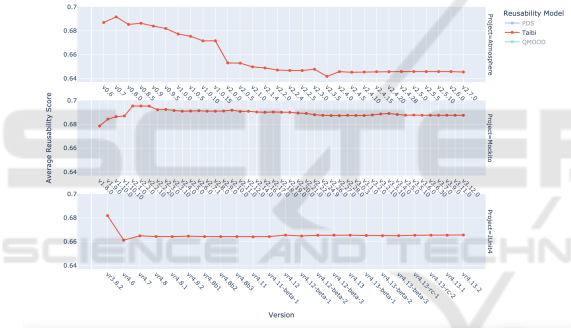

Figure 2: Taibi Reusability Evolution (tool view with Taibi model selected).

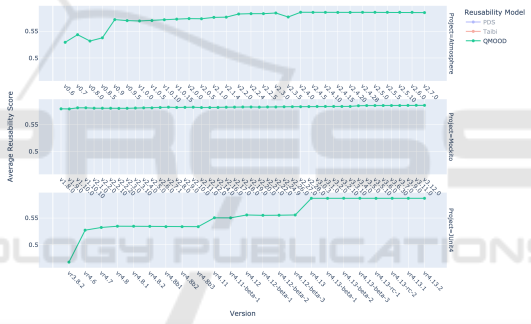

Figure 3: QMOOD Reusability Evolution (tool view with QMOOD model selected).

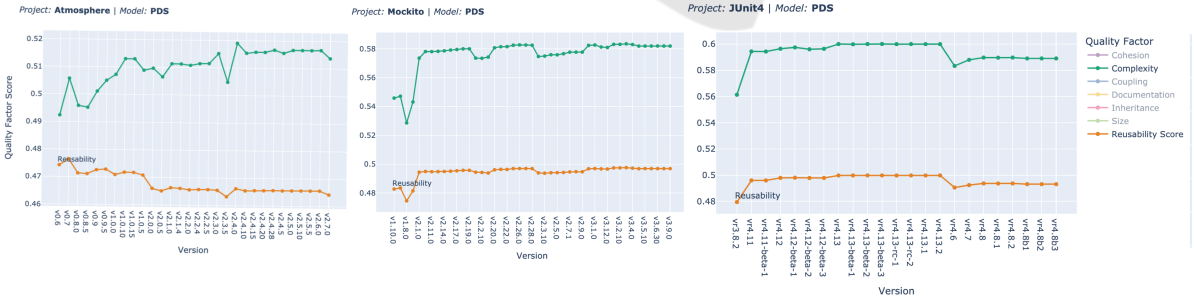

Figure 4: PDS Complexity impact on Reusability.

ity, however Taibi succeeds in returning close values to the ones calculated for maintainability, although their evolution curves seem to have little resemblance. **Lesson Learned:** Considering the strong interdependency between reusability and maintainability, the results obtained in this case study are not very convincing. From our point of view, they in fact represent another argument why maintainability index is not a good indicator for maintainability (Molnar and Motogna, 2021), (Heitlager et al., 2007), as it is deprecated: it was defined in 1992, considering modular and procedural programming languages, and ignor-

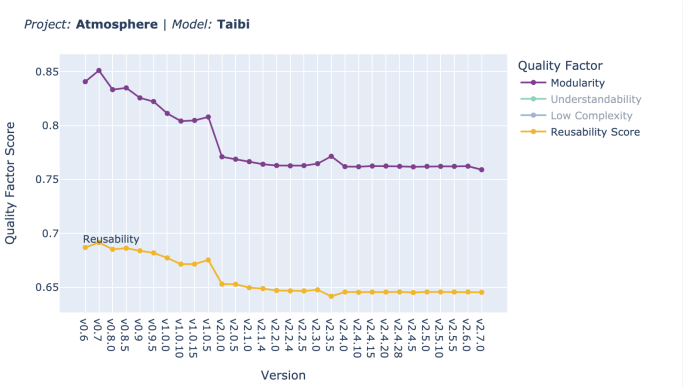

Figure 5: Taibi Modularity impact on Reusability.

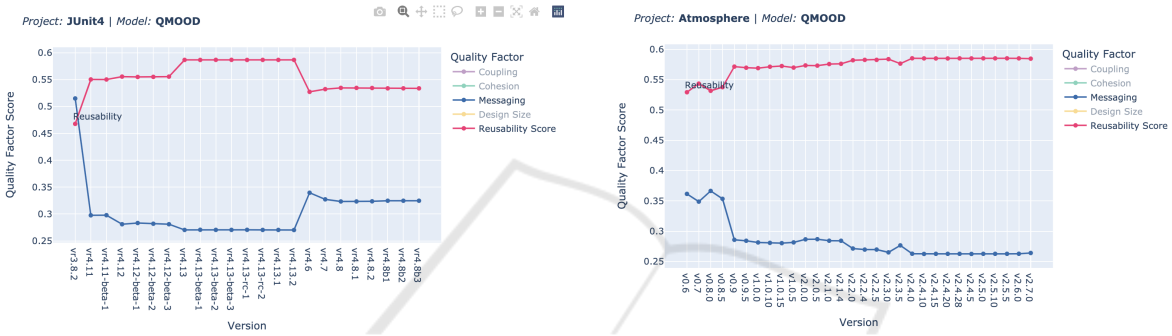

Figure 6: QMOOD Messaging impact on Reusability.

ing object oriented features that defined new relations such as inheritance, coupling and cohesion. However, considering that MI is still used as indicator for refactoring by practitioners, and it is implemented in several metric tools, we considered the investigation between MI and reusability to be relevant.

5 THREATS TO VALIDITY

The case study was conducted according to the existing standard (Ralph, Paul (ed.), 2021) and practices (Runeson and Höst, 2008). We started by formulating the objective and research questions, then selecting the reusability models and target applications. Data collection and analysis was performed with the help of the developed tool.

Internal threats were mitigated by using as much as possible existing tools, already used in previous research studies, and also by manually examining the source code. However, the major threat is represented by the implementation of the reusability models, given the fact that the existing references have not always been clear enough (such as Design Size in QMOOD, Relevance of Identifiers and Correlation Identifiers Comments in Taibi model). We address this threat by looking for supplementary references for these metrics in our implementation.

External threats are representing by the selection of target applications, and we tried to address this threat by considering open source projects, providing access to source code and exposing a significant trust due to their intense use by the development community. Section 3.2 contains details about the selection process. Another threat might be represented by a small amount of analyzed projects. This is one aspect that we consider for future improvement, but at this stage of the project, we give a closer attention to having a significant number of versions for a project rather than a large number of projects.

6 CONCLUSIONS AND FUTURE WORK

We successfully implemented three different reusability assessment models and applied them to a set of open-source projects, that gave us the opportunity to study reusability scores throughout significant versions of selected projects, so that long term evolution of reusability can be observed. We also investigated the long-term relationship of various source

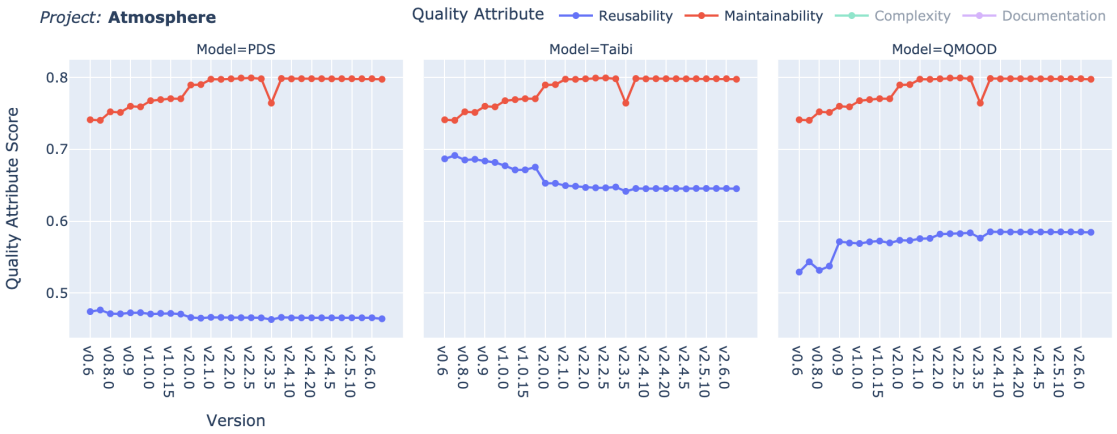

Figure 7: Relationship between reusability and maintainability - Atmosphere.

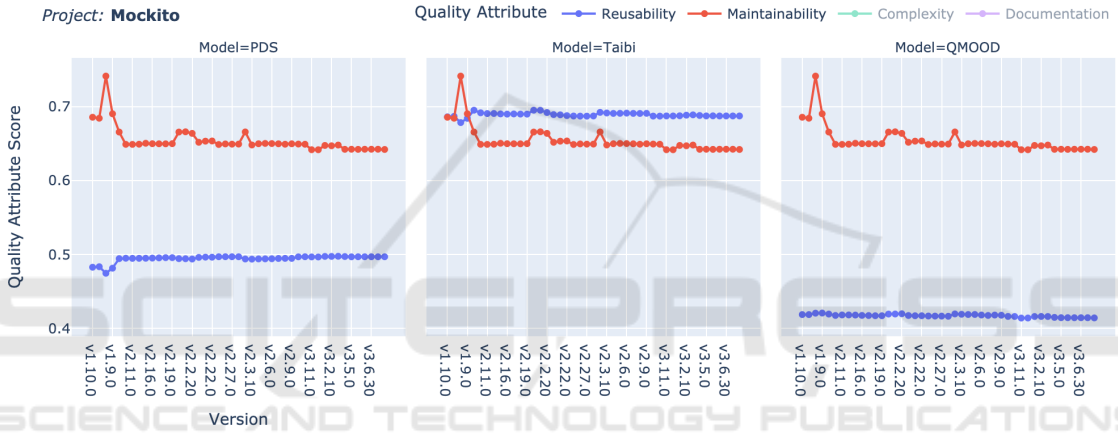

Figure 8: Relationship between reusability and maintainability - Mockito.

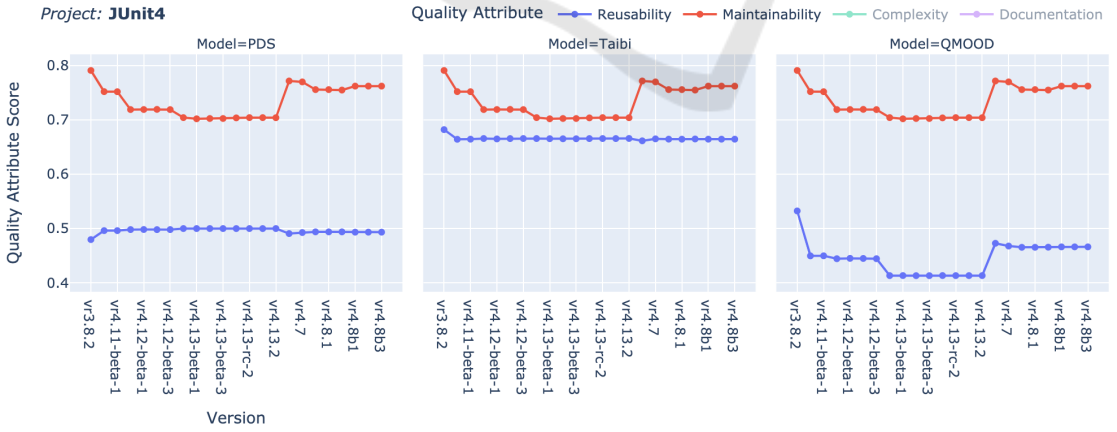

Figure 9: Relationship between reusability and maintainability - JUnit4.

code quality factors to measured reusability and found the ones that have the biggest influence on the evolution of the reusability degree. Finally, relation between reusability and maintainability was subject of exploration.

Our findings are useful for both research and practitioners' communities. From a research point of view, we bring empirical evidence in comparing reusability models defined in literature. For practitioners, the developed tool can be integrated in or-

der to track and control reusability scores, by setting acceptable thresholds within projects. High quality of code can be maintained in an automatic manner, ensuring that no poorly written and low-reusability components are introduced in the codebase – this is especially useful for projects that serve as libraries or frameworks, because the degree of reusability for these must always be maintained high.

As future research directions, we consider this study as a starting point that can be continued by:

- **Adding more Reusability Assessment Models** to the performed analysis, including fuzzy, neural network-based ones and the whole spectrum of machine learning models that has been proposed in the literature as valid possible ways of assessing reusability. This would be a great enhancement to the current work and would definitely provide promising results;
- **Extending the Data Set** of applications, in order to increase the confidence of the evaluation;
- **Performing a more in Depth Analysis of Reusability** at the level of components and classes.

ACKNOWLEDGEMENTS

This study was partially funded by the 2022 Development fund of UBB.

REFERENCES

Ampatzoglou, A. and Stamatia, B. a. (2018). Reusability index: A measure for assessing software assets reusability. In Capilla, R., Gallina, B., and Cetina, C., editors, *New Opportunities for Software Reuse*, pages 43–58, Cham. Springer International Publishing.

Atmosphere (2021). https://github.com/Atmosphere/atmosphere. Online, accessed 01-03-2022.

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

Basili, V., Caldiera, G., and Rombach, H. D. (1994). The goal question metric approach. In *Encyclopedia of software engineering*.

Frakes, W. and Terry, C. (1996). Software reuse: Metrics and models. *ACM Comput. Surv.*, 28:415–435.

Gensim (2021). https://radimrehurek.com/gensim/. Online, accessed 01-03-2022.

Gui, G. and Scott, P. D. (2009). Measuring software component reusability by coupling and cohesion metrics. *Journal of Computers*, pages 797–805.

Heitlager, I., Kuipers, T., and Visser, J. (2007). A practical model for measuring maintainability. In *Quality of Information and Communications Technology, 6th International Conference on the Quality of Information and Communications Technology, QUATIC 2007, Lisbon, Portugal, September 12-14, 2007, Proceedings*, pages 30–39.

Henry, S. and Lattanzi, M. (1994). Measurement of software maintainability and reusability in the object oriented paradigm. http://hdl.handle.net/10919/19813.

ISO/IEC 25010 (2011). Systems and software engineering. http://www.iso.org. Accessed: 2015.

JUnit4 (2021). https://junit.org/junit4/. Online, accessed 01-03-2022.

Kitchenham, B., Pickard, L., and Pfleeger, S. L. (1995). Case studies for method and tool evaluation. *IEEE Softw.*, 12(4):52–62.

Lee, Y. and Chang, K. H. (2000). Reusability and maintainability metrics for object-oriented software. page 88–94. Association for Computing Machinery.

McCall, J., Richards, P., and Walters, G. (1977). Factors in software quality. *Nat Tech.Information Service*, 1:0–0.

Mehboob, B., Chong, C. Y., Lee, S., and Lim, J. (2021). Reusability affecting factors and software metrics for reusability: A systematic literature review. *Software: Practice and Experience*, 51.

Mijač, M. and Stapic, Z. (2015). Reusability metrics of software components: Survey.

Mockito (2021). https://site.mockito.org/. Online, accessed 01-03-2022.

Molnar, A.-J. and Motogna, S. (2021). A study of maintainability in evolving open-source software. In Ali, R., Kaindl, H., and Maciaszek, L. A., editors, *Evaluation of Novel Approaches to Software Engineering*, pages 261–282. Springer International Publishing.

Nair, T. G. and Selvarani, R. (2010). Estimation of software reusability: An engineering approach. *SIGSOFT Softw. Eng. Notes*, 35(1):1–6.

Oman, P. and Hagemeister, J. (1992). Metrics for assessing a software system's maintainability. In *Proceedings Conference on Software Maintenance 1992*, pages 337–344.

Papamichail, M. D., Diamantopoulos, T., and Symeonidis, A. L. (2019). Measuring the reusability of software components using static analysis metrics and reuse rate information. *Journal of Systems and Software*, 158:110423.

Poulin, J. S. (1994). Measuring software reusability. In *Proceedings of the Third International Conference on Software Reuse: Advances in Software Reusability*, pages 126–138. Society Press.

Ralph, Paul (ed.) (2021). ACM Sigsoft Empirical Standards for Software Engineering Research, version 0.2.0.

Runeson, P. and Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14:131–164.

Sandhu, P. and Singh, H. (2008). A reusability evaluation model for oo-based software components. *World*

Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering, 2:912–917.

Singh, C., Pratap, A., and Singhal, A. (2014). Estimation of software reusability for component based system using soft computing techniques. In *2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence)*, pages 788–794.

SourceMeter (2022). https://www.sourcemeter.com/. Online, accessed 01-03-2022.

Taibi, F. (2014). Empirical analysis of the reusability of object-oriented program code in open-source software. *International Journal of Computer and Information Engineering*, 8(1):118 – 124.