World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:16, No:12, 2022

A Framework to Support Reuse in Object-Oriented Software Development

Fathi Taibi

Abstract**—**Reusability is a quality desired attribute in software products. Generally, it could be achieved through adopting development methods that promote it and achieving software qualities that have been linked with high reusability proneness. With the exponential growth in mobile application development, software reuse became an integral part in a substantial number of projects. Similarly, software reuse has become widely practiced in start-up companies. However, this has led to new emerging problems. Firstly, the reused code does not meet the required quality and secondly, the reuse intentions are dubious. This work aims to propose a framework to support reuse in Object-Oriented (OO) software development. The framework comprises a process that uses a proposed reusability assessment metric and a formal foundation to specify the elements of the reused code and the relationships between them. The framework is empirically evaluated using a wide range of open-source projects and mobile applications. The results are analyzed to help understand the reusability proneness of OO software and the possible means to improve it.

Keywords**—**Software reusability, software metrics, object-oriented software, modularity, low complexity, understandability.

I. Introduction

SINCE the early days of programming, some forms of improvised code reuse has been practiced. However, the usage of reusable components in industrial software development was first introduced in the late sixties by Douglas McIlroy [22]. Even though code reuse [9] is widely practiced in software development, other software artefact such as design skeletons and processes are reused as well. By reusing software, the cost of the development is reduced, the speed of development is increased, and reliability is improved [18], [12].

Agile methods are widely and successfully adopted in software development. For this reason, agile software project management is being considered for other industries [6]. Rapid and continuous delivery is one of the key principles that guide the management of agile projects today and is a major trend in the software industry. For example, a company like Amazon deploys code every 11.7 seconds. This hints clearly to a direct link between fast delivery and software reuse.

Software start-up companies are an interesting phenomenon to study in the context of software reuse. Since the reduction of time-to-market is one of the most important objectives in this context, exploiting code-reuse, development frameworks and design patterns make a lot of sense [15]. Specifying these patterns formally promotes their reusability even further. However, there are suggestions that design patterns should be used cautiously due to the possibility that they may hinder maintenance and evolution [16].

Human reusability assessment and fault prediction could be mimicked through neural networks [31]. The studied prediction models are predominately statistical based, based on machine learning or based on software metrics [4]. The identification of the appropriate metrics that can be used to perform the prediction or assessment is crucial. Readability or understandability of the source code is a factor that is often associated with software reusability proneness. Using naming conventions and writing useful comments are examples of techniques that can improve understandability. The usage of naming conventions has been found to be reliable if the names used are related to the concepts implemented [3]. Maintenance tasks are made difficult to carry in the presence of lexicon bad smells such as inconsistent term usage and odd grammatical structures [1]. Moreover, high complexity must be avoided since it is associated with programs that are less reusable, hard to test and maintain. Furthermore, structuring program code using modules that are highly cohesive [20] and highly independent [8] is a vital factor for reusability.

Excessive coupling between classes was found to be a very reliable predictor of faults in OO systems as indicated in [11] where it was found that Coupling Between Objects (CBO) is more reliable than Lack of Cohesion of Methods (LCOM) and several other OO design metrics in predicting faults. The combination of this metric (i.e., CBO) with metrics addressing the other aspects could form the basis of a reusability assessment approach since they allow measuring the factors related to the reusability proneness of program code while at the same time discarding defect-prone code from being reused. Finally, it is important to note that classes participating in antipatterns (i.e., bad smells, which are poorly designed classes) have been found to be more change and fault prone than those that do not [16].

Reusing existing code is beneficial only if the reused code possesses the required quality. The explosion in the amount of open-source software projects and mobile applications could be seen as a direct consequence of massive code reuse. This poses two major problems. The overall quality of most of these projects is unknown. Reusing them blindly can cause major problems. Worse, a considerable amount of these projects is reported as being malicious. This is especially true for mobile applications. Hence, it is important to have a framework to support reuse in OO software development by assessing the

F. Taibi is an independent researcher and a consultant in international development cooperation projects, Algeria (e-mail: ft.taibi@gmail.com).

reusability proneness of potential source code.

The remainder of the paper is organized as follows: Section II gives an overview of the problem and reviews related work. Section III details the proposed framework, which consists of a formal foundation to specify the manipulated source code, the proposed reusability assessment metric and the assessment process. Section IV provides the details of the empirical investigation performed in order to assess the reusability proneness of open-source projects and mobile applications. The analysis of the results helps in understanding the extent of this quality, the factors hindering it and the effectiveness of the proposed metric in reusability assessment prediction. Finally, Section V summaries the findings of this research and highlights future research directions.

II. Related Work

A model for the process involved in performing a pragmatic reuse task was proposed in [14]. This included how to capture the decisions of a developer regarding how each program element should be treated (i.e., a pragmatic-reuse plan). Partial tool support was provided, which can take the selected source code from its originating system and integrating it into the target system (i.e., the developer's one). A series of case studies and experiments were conducted using a variety of source systems and tasks. These experiments showed a significant decrease in the time that developers require to perform pragmatic reuse tasks, an increase in the likelihood that developers will successfully complete their reuse tasks, a decrease in the time required to identify infeasible reuse tasks, and an improved sense in the ability of developers to manage the risk in these tasks.

A tool supported quality model on maintainability and reusability of software was presented in [19]. It relied on user intuition in selecting a metric set for their projects where modularity and complexity were used to measure reusability. Modularity was measured based on the cohesion and coupling of classes while the internal and external complexity of classes was used to assess complexity.

An empirical investigation was conducted in [2] in order to study the ability of 29 internal class measures to estimate reuse proneness from the perspectives of inheritance and instantiation. These measures represent class attributes such as cohesion, coupling and size. Size and coupling attributes were found to be correlated to the reuse proneness of a class via inheritance and instantiation. The cohesion attribute has a positive impact on the reuse proneness of a class via instantiation only. Due to the large number of attributes used and the overlapping in the qualities they measure, the model lacked effectiveness.

A metric suite was proposed in [30] to measure the reusability of components in component-based software development. This suite consisted of the definition of five metrics in order to measure understandability, adaptability and portability factors of a given component. Statistical analysis of a number of JavaBeans components was used to set a confidence interval for each metric. The existence of meta-information was used to measure the understandability and the observability of a component. Adaptability and portability were measured based on metrics, rating customizability and external dependency, respectively.

A new coupling and cohesion metrics to rank the reusability of Java components was proposed in [10]. Interestingly, cohesion was measured as the degree of relativeness among the methods of a class (including transitive cohesion). A similar intuition was used for the proposed coupling metric. In comparison to some of the existing cohesion and coupling metrics, the experiments conducted revealed that the proposed metrics were better predictors of the amount of code that was added, modified or deleted in order to extend the functionality of the studied components.

An empirical study was conducted on software reuse in Java open-source project [13]. It was aimed at studying the extent of code reuse occurrence and third-party code usage. Black-box software reuse was found to be the predominant form of software reuse. Moreover, in 95% of the cases the amount of reused code exceeded the amount of the original one.

Assessing the reusability proneness of OO code at the class level obeys different considerations in comparison with the assessment of components in Component Based Software Engineering (CBSE). A component is considered in this context as a black box. Several reusability metrics were surveyed in [17]. The adaptability, interface, composability, complexity and understandability were used predominantly across the surveyed work.

An investigation into the applicability of software metrics in the software fault prediction was conducted in [24]. A total of 106 papers that were published between 1991 and 2011 were selected and classified according to metrics and context properties. The findings of this investigation showed that OO metrics were used nearly twice as often compared to traditional source code metrics or process metric. The metrics proposed by Chidamber and Kemerer's (CK) [5] were most frequently used. In comparison to size and complexity metrics, OO and process metrics have been reported to be better fault detectors, while process metrics are better predictors of post-release faults compared to any static code metrics [24].

A large number of mobile applications (265359) were analyzed in [7] and 1.62% (4295) of them were discovered to be victims of cloning. Each one of these applications was probably cloned several times. Additionally, 13.61% (36106) applications were rebranded including 88 malware and 169 malicious applications. Duplicative application content and library usage in Google Play was a subject of a large-scale investigation in [29] among other concerns. Interestingly, the amount of duplicative application content among the free applications was around 25%. Moreover, over half of the free Android applications use advertising libraries. Furthermore, an increase in the popularity of an application is correlated with the usage of native libraries, which is meant to optimize the user experience of the application.

In order to have a good reusability proneness predictor several factors should be considered while avoiding redundancy among the measured features. Although cohesion and coupling metrics have been proven to be good reusability predictors, they need to be combined with other important factors measuring the complexity, understandability and customizability among others.

III. The Proposed Framework

The proposed framework is aimed at specifying the manipulated code and proposing reusability assessment metric and process. The proposed metric and approach are an extension of the one proposed in [25] and [26] based on the findings of the initial empirical investigation as well as the formal specification of the elements of the manipulated code and the relationships between them.

A. *Formal Foundation*

Using the formal specification language Z [23], below is a formal specification of the manipulated code where: 'Name' is the set of all valid names of the elements of a program, 'Type' is a set of all possible valid types (incl. void) and 'CodeLine' represents any line of code with three sub-types (SimpleCodeLine, CommentCodeLine and MixedCodeLine). For the sake of practicality, three visibility levels are considered for the elements of a program: public, private and protected. Similarly, three types of relationships between classes are considered: aggregation, inheritance, and any other form of dependency (i.e., association). Figs. 1 and 2 show a formal specification of the types, parameters, attributes, constructors, methods, classes, files, and projects of the manipulated code.

```
[Name, Type, CodeLine]
Visibility ::= private | public | protected
Rel ::= aggregated_by | derived_from | associated_with

┌──────── Parameter ─────────
│ name: Name
│ type: Type
└────────────────────────────

┌──────── Attribute ─────────
│ Parameter
│ ┌──────────────────
│ │ visibility: Visibility
│ └──────────────────
└────────────────────────────

┌──────── Constructor ───────
│ name: Name
│ visibility: Visibility
│ params: ℙ Parameter
└────────────────────────────

┌──────── Method ────────────
│ Constructor
│ ┌──────────────────
│ │ return: Type
│ └──────────────────
└────────────────────────────
```

Fig. 1 Formal specification in Z of the types, parameters, attributes, constructors, and methods of the manipulated code

In a program, a parameter has a name and a type associated with it, whereas an attribute has all elements of a parameter in addition to a visibility. Furthermore, a constructor has a name, a visibility and a set of parameters, whereas a method has all the elements of a constructor in addition to a return type.

```
┌──────── Class ─────────────
│ name: Name
│ visibility: Visibility
│ attributes: ℙ Attribute
│ methods: ℙ Method
└────────────────────────────

┌──────── File ──────────────
│ name: Name
│ lines: ℙ1 CodeLine
│ classes: ℙ1 Class
│ rels: ℙ ( Class × Class ) → Rel
└────────────────────────────

┌──────── Project ───────────
│ name: Name
│ files: ℙ1 File
└────────────────────────────
```

Fig. 2 Formal specification in Z of the classes, files, and projects of the manipulated code

A class has a name, a visibility, a set of attributes and methods. A program code file has a name, a non-empty set of code lines, a non-empty set of classes and a set of relationships between these classes. Finally, a project has a name and comprises a non-empty set of files.

B. *The Proposed Reusability Assessment Metric*

The reusability of a class is assessed by considering the factors Understandability (U), Low Complexity (LC) and Modularity (M). A brief description of how each factor is calculated is given below:

- U is a value between 0 and 1 that is assessed through the signification or relevance of names used for a class, fields and methods (Relevance Of Identifiers - ROI), the rate of code comments and their correlation with the names used (Correlation Identifiers Comments - CIC). CIC is calculated using a similarity metric derived from the Longest Common Substring, N-Grams and the Levshtein distance algorithms [27], [28]. CIC is calculated for the whole file, i.e., classes in the same file have the same CIC. However, ROI is assessed manually (i.e., expert rating) by two different experts and the average value is taken.
- LC is a value between 0 and 1 calculated using a Weighted Cyclomatic Complexity (WCC) value of a class, the Number of Methods (NM) per class where the threshold 7 is used as per the recommendations in [21], the Depth of Inheritance Tree (DIT) where 5 is used as a threshold and the Response For a Class (RFC). WCC is calculated as the sum of the weights of the individual methods of the class regarding their cyclomatic complexity and dividing it by the Number of Methods (NM).
- M in this context is a value between 0 and 1 that is assessed through measuring the cohesion (through LCOM) and coupling (through CBO) of the class.

Several other factors were also considered. They include factors such as the size, customizability and stability. Some were discarded because they overlap with the factors already considered (e.g., size is correlated with criteria such as WCC and NM) while others were discarded due to the absence of reliable metrics that can measure them (e.g., stability).

The reusability of a given class R is calculated using:

$$R = \sum_{i=1}^{n} \lambda_i \times F_i \qquad (1)$$

where F_i are the factors used in the reusability proneness assessment and λ_i are tuning parameters ($\Sigma\ \lambda_i = 1$).

LC and M are given more weights than U since the latter factor was found to be slightly less significant than the former two factors in measuring the reusability proneness of a class. Currently, the weight 0.35 is used for M and LC and 0.3 for U. Finally, M, LC and U are calculated as a weighted average of the respective metrics used in their calculation.

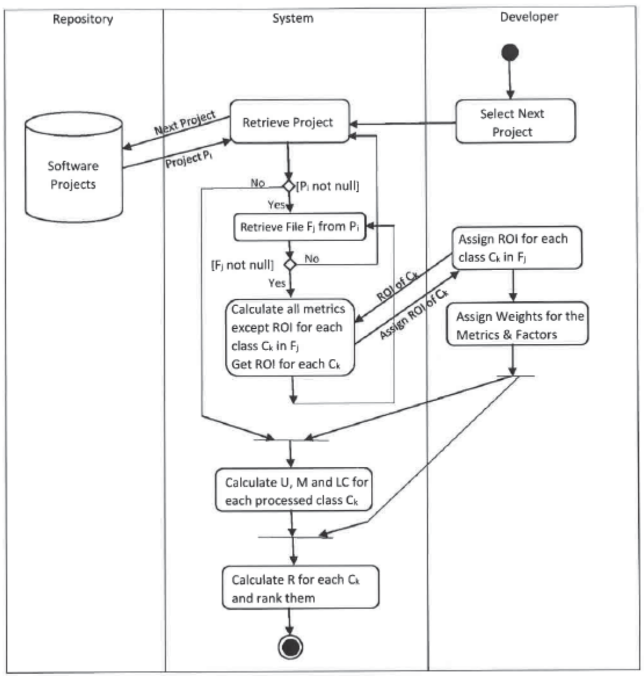
Repository
System
Developer
Select Next Project
Retrieve Project
Next Project
Project P_i
Software Projects
No
[P_i not null]
Yes
Retrieve File F_j from P_i
[F_j not null]
No
Yes
Assign ROI for each class C_k in F_j
ROI of C_k
Assign ROI of C_k
Calculate all metrics except ROI for each class C_k in F_j
Get ROI for each C_k
Assign Weights for the Metrics & Factors
Calculate U, M and LC for each processed class C_k
Calculate R for each C_k and rank them

Fig. 3 The reusability assessment process

C. *The Proposed Reusability Assessment Process*

The reusability assessment process is initiated by a developer that requests the selection of the next project from a software repository. This repository contains a diverse set of randomly selected OO projects and mobile applications. If there is an unprocessed project, then its files are selected one by one and all the required metrics are calculated systematically for each one of their classes except for ROI that is assessed manually by the developer(s). Once there are no more projects to process, the factors U, M and LC are calculated systematically for each processed class based on the calculated metrics of the previous step and the weights assigned by the developer. The final step of the process consists of calculating the reusability proneness metric R for each class and then raking them accordingly. A heuristic method was used to find their weights using a set of classes with known reuse potential. These weights could also be chosen according to the qualities required by a developer in search of reusable modules. LC and M are given more weight than U since the latter factor was found to be slightly less significant than the former two factors in measuring the reusability proneness of a class. Fig. 3 gives a graphical illustration of the reusability assessment process described above.

IV. Evaluation

47 projects and applications were randomly selected from various open-source sites and Android markets such as [35]. They represent various types of applications such as Brain and Puzzle, Business, Communication, Education, Game, Social, Lifestyle, Utility, etc. They incorporated a total 809 files comprising 2247 classes with a total of 120795 Line of Code (LOC). Table I shows the details of the selected applications.

TABLE I
Details of the Projects and Applications Used in the Evaluation

	Max	Min	Median	Mean	StdDev
#Files	86.00	2.00	14.00	17.21	14.47
#Classes	133.00	11.00	35.00	47.81	31.27
Size (LOC)	13676.00	182.00	1344.00	2570.11	2862.24
%Comments	53.83%	0.00%	5.79%	9.33%	10.63%

The relatively large variation (StdDev) in the size and the percentage of comments is a consequence of the randomness used in choosing the software projects. One of the projects was considerably larger than the rest; it included 86 modules and 133 classes. Only 5 projects included less than 20 classes and only 4 of them included more than 100 classes. Moreover, the Android applications with no source code led to a null percentage of code comments as the latter cannot be decompiled. A converter [34] was used in order to retrieve the individual class files because these projects had package files only (i.e. *.apk). It translates a ‘dex’ file (available from the *.apk file) into a ‘jar’ file that contains the individual classes of an application. A Java decompiler [36] was then used to obtain the source code. Hence, for these applications, CIC was not used to calculate the factor U.

Chidamber and Kemerer Java Metrics [33] and C and C++ Code Counter [32] tools were used to calculate CC, LCOM, CBO, NM, DIT and RFC. A small prototype tool was developed to calculate WCC and CIC while ROI was assessed manually as indicated previously. The results were the thoroughly analyzed. Fig. 4 shows the reusability of each class in the studied projects and applications. The results are sorted for a better analysis.

The overall reusability of the studied classes was good with an average R of 0.73. Only 396 classes (17.62%) had a reusability below 0.5 while 212 classes (9.43%) had a reusability between 0.5 and 0.7. All the remaining 1639 classes (72.94%) had a reusability greater or equal to 0.7 as shown in Fig. 5.

The impact of the factors used to measure reusability was studied by calculating the correlation between them and R. Classes were again categorized into three categories based on their reusability score. Table II shows the overall corresponding correlation as well as the correlation for the individual categories.

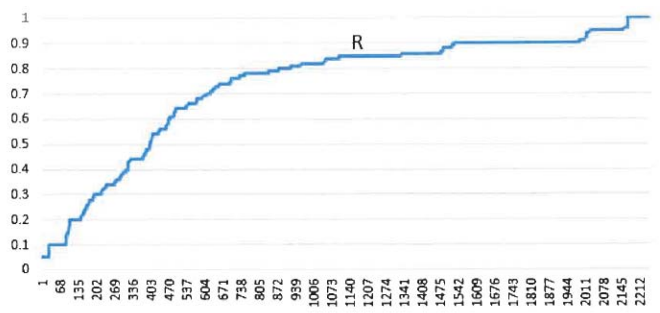

Fig. 4 The reusability proneness of the evaluated classes

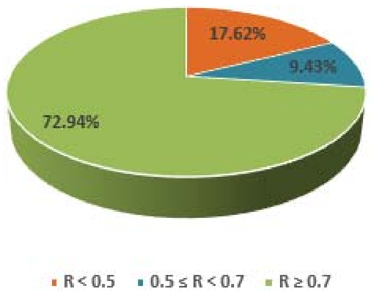

Fig. 5 The distribution of the reusability proneness of the evaluated classes

TABLE II
CORRELATION BETWEEN THE FACTORS USED AND THE PROPOSED REUSABILITY METRIC

FACTOR	All classes	Classes with $R < 0.5$	Classes with $0.5 \leq R < 0.7$	Classes with $R \geq 0.7$
M	0.743	0.813	0.275	0.786
LC	0.501	0.698	0.499	0.297
U	0.248	0.299	0.003	0.539

Overall, the factor M has the highest positive correlation to R followed by LC. There was a relatively poor correlation between U and R since the original source code (with comments) was not available for most projects and applications used in the evaluation. For classes with a low reusability ($R < 0.5$), the factor M has the highest correlation followed by LC and then U. For classes with high reusability ($R \geq 0.7$), the factor M has the highest correlation followed by U and then LC. Finally, for classes with average reusability ($0.5 \leq R < 0.7$), the factor LC has the highest correlation followed by M while U has almost no correlation at all. Hence:

- The value of M is more correlated to classes with high and low reusability.
- The value of LC is far more correlated to classes with low reusability.
- The value of U is relatively correlated to classes with high reusability.

These results indicate that the factors used to calculate R are valid reusability proneness predictors as they allow the identification (and eventually the reuse) of classes with high reusability while highlighting those with a poor reusability.

In order to study the predictive capability of the proposed metric, the result obtained for each class was compared to a value (R*) also between 0 and 1 combining human assessment, online rating and online reviews. For the latter two parameters, the value assigned is the same for all the classes of a given project. For the former parameter, each class was assessed individually and given a score. Table III shows the results obtained.

TABLE III
PREDICTIVE CAPABILITY OF THE PROPOSED REUSABILITY METRIC

	All classes	Classes with $R < 0.5$	Classes with $0.5 \leq R < 0.7$	Classes with $R \geq 0.7$
Correlation (R, R*)	0.586	0.793	0.003	0.612

Overall, there was a good positive correlation between the reusability calculated using the proposed metric (R) and the value (R*) that combines manual expert assessment, online ratings and reviews. This correlation was excellent for classes with poor reusability and good for the ones with high reusability. This supports further the validity of the proposed metric as it clearly allows discarding classes with poor reusability and identifying the ones with high reusability proneness.

The internal validity of the proposed metric is achieved through the clear correlation that exists between the factors used and the reusability proneness of a given class. Especially for classes with high reusability (to be potentially reused) and those with low reusability (to be excluded from reuse). These factors were measured using well established and validated metrics, which support this validity even more. Additionally, manual intervention was minimized in order to avoid errors in measurements. This was combined with the cross checking (twice) of the results obtained automatically. This was aimed at finding any abnormal values, which is a sign of construct validity. Finally, even though the number of studied classes in the empirical investigation is not very substantial, various types of projects and applications were used and were randomly selected from various open-source websites. This is sign of external validity and shows that the results obtained can be replicated to a larger number of classes from other sources.

V. CONCLUSION AND FUTURE WORK

A framework was proposed in this paper to support reuse in OO software projects. The framework comprised a process that uses a proposed reusability metric and a formal foundation to specify the elements of the reused code and the relationships between them. The proposed metric combined carefully selected factors with a strong correlation to reusability proneness.

The proposed framework was empirically evaluated using a diverse set of randomly selected open-source projects and mobile applications. A total of 2247 classes were assessed using the proposed metric. The overall reusability of these classes was good with an average R of 0.73. Only 17.62% of them have a low usability ($R < 0.5$). Moreover, the factor M was found to be more correlated to classes with high reusability and those with low reusability whereas the factor LC was found to be more correlated to classes with low reusability. Furthermore, a good positive correlation between the reusability calculated using the proposed metric and a value that combines manual expert assessment, online ratings and reviews was found. This correlation was strong for classes with high and low reusability. Hence, the proposed metric R is a valid reusability proneness predictor as it allows the identification (and eventually the reuse) of classes with high reusability while highlighting those with a poor reusability.

REFERENCES

[1] Abebe, S. L., Kessler, F. B., Haiduc, S., Tonella, P. and Marcus, A. (2011). The Effect of Lexicon Bad Smells on Concept Location in Source Code, Proceedings of the 11th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM), pp. 125 – 134.

[2] Al-Dallal, J. and Morasca, S. (2014). Predicting object-oriented class reuse proneness using internal quality attributes," Empirical Software Engineering, 19(4), 775-821.

[3] Anquetil, N. and Lethbdige, T. (1998) . Assessing the Relevance of Identifier Names in Legacy System, In Proc of the Centre for Advanced Studies on Collaborative Research Conference.

[4] Catal, C. (2011). Software fault prediction: A literature review and current trends. Expert Systems with Applications, 38(4), 4626-4636.

[5] Chidamber, S.R. and Kemerer, C.F. (1994). A metrics suite for object oriented design. *IEEE Transactions on software engineering*, *20*(6), 476-493.

[6] Conforto, E. C., Salum, F., Amaral, D. C., da Silva, S. L., & Magnanini de Almeida, L. F. (2014). Can agile project management be adopted by industries other than software development? Project Management Journal, 45(3), 21–34.

[7] Crussell, J., Gibler, C. and Chen, H. (2013). AnDarwin: Scalable Detection of Semantically Similar Android Applications, Lecture Notes in Computer Science, pp. 182-199.

[8] Darcy, D. and Kemerer, C. (2005). OO Metrics in Practice, IEEE Software, 22(6), 17-19.

[9] Frakes, W. and Kang, K. "Software Reuse Research: Status and Future," IEEE Transactions on Software Engineering, vol. 31, no. 7, pp. 529-536, 2005.

[10] Gui, G. and Scott, P. D. (2006). Coupling and cohesion measures for evaluation of component reusability, Proceedings of the 2006 international workshop on Mining software repositories, pp. 18 – 21.

[11] Gyimothy, T., Ferenc, R. and Siket, I. (2005). Empirical Validation of Object Oriented Metrics on Open Source Software for Fault Prediction, IEEE Transactions on Software Engineering, 31(10), 897-910.

[12] Haefliger, S., Von-Krogh, G. and Spaeth, S. "Code Reuse in Open Source Software," Management Science, vol. 54, no. 1, pp. 180-193, 2008.

[13] Heinemann, L., Deissenboeck, F., Gleirscher, M., Hummel, B. and Irlbeck, M. (2011). On the extent and nature of software reuse in open source Java projects, Proceedings of the 12th international conference on Top productivity through software reuse, Klaus Schmid (Ed.). SpringerVerlag, pp. 207-222.

[14] Holmes, R., & Walker, R. J. (2012). Systematizing pragmatic software reuse. *ACM Transactions on Software Engineering and Methodology*, *21*(4), 20.

[15] Jansen, S., Brinkkemper, S., Hunink, I., & Demir, C. (2008). Pragmatic and opportunistic reuse in innovative start-up companies. *IEEE software*, *25*(6), 42-49.

[16] Khomh, F., Di-Penta, M., Gueheneuc, Y.G. and Antoniol, G. (2012). An exploratory study of the impact of antipatterns on class change- and fault-proneness, Empirical Software Engineering, 17(3), 243-275.

[17] Kumar, V., Sharma, A., Kumar, R., & Grover, P. S. (2012). Quality aspects for component-based systems: A metrics-based approach. Software: Practice and Experience, 42(12), 1531-1548.

[18] Land, R., Sundmark, D., Luders, F., Krasteva, I. and Causevic, A. "Reuse with Software Components - A Survey of Industrial State of Practice," Formal Foundations of Reuse and Domain Engineering, Lecture Notes in Computer Science, vol. 5791, pp. 150-159, 2009.

[19] Lee, Y. and Chang, K. H. (2000). Reusability and maintainability metrics for object-oriented software," Proceedings of the ACM-SE 38th annual on Southeast regional conference, pp.88-94.

[20] Marcus, A., Poshyvanyk, D., & Ferenc, R. (2008). Using the conceptual cohesion of classes for fault prediction in object-oriented systems. IEEE Transactions on Software Engineering, 34(2), 287-300.

[21] McConnell, S. (2004). Code Complete: A Practical Handbook of Software Construction, 2nd Edition. Microsoft Press.

[22] McIlroy, D. "Mass-produced software components," In Proc 1968 NATO Conference on Software Engineering, Buxton, J.M., Naur, P., Randell, B. (eds.), pp. 138-155, Petroceli/Charter, New York, 1969.

[23] Potter, B., Sinclair, J., & Till, D. (1996). Introduction to Formal Specification and Z. Prentice-Hall.

[24] Radjenović, D., Heričko, M., Torkar, R. and Živkovič, A. (2013). Software fault prediction metrics: A systematic literature review, Information and Software Technology, 55(8), 1397-1418.

[25] Taibi, F. (2014). 'Empirical Analysis of the Reusability of Object-Oriented Program Code in Open-Source Software'. World Academy of Science, Engineering and Technology, International Journal of Computer and Information Engineering, 8(1), 118 - 124.

[26] Taibi, F. (2013). 'Reusability of open-source program code: a conceptual model and empirical investigation'. ACM SIGSOFT Software Engineering Notes, 38(4), 1-5.

[27] Taibi, F., Alam, M. J. & Abdullah J. (2010). " On Differencing Object-Oriented Formal Specifications" Journal of Object Technology 9(1), 183-198.

[28] Taibi, F., Abbou, F. M. & Alam, M. J. (2008). "A Matching Approach for Object-Oriented Formal Specifications." Journal of Object Technology 7(8), 139-153.

[29] Viennot, N., Garcia, E., & Nieh, J. (2014, June). A measurement study of Google Plasy. In ACM SIGMETRICS Performance Evaluation Review, 42(1), 221-233.

[30] Washizaki, H., Yamamoto, H. and Fukazawa, Y. (2003). A metrics suite for measuring reusability of software components, Proceedings of the 9th Software Metrics Symposium, pp. 211-223.

[31] Wong, W. E., Debroy, V., Golden, R., Xu, X. and Thuraisingham, B. (2012). Effective Software Fault Localization Using an RBF Neural Network. IEEE Transactions on Reliability, 61(1), 149-169.

[32] CCCC, http://cccc.sourceforge.net/, November 2022.

[33] CKJM, https://www.spinellis.gr/sw/ckjm/, November 2022.

[34] Dex2jar, https://github.com/pxb1988/dex2jar, November 2022.

[35] Google Play, https://play.google.com/store/apps, November 2022.

[36] JAD, https://varaneckas.com/jad/, November 2022.