A quality model for evaluating maintainability of object-oriented software systems

Özlem AKALIN[1] and Feza BUZLUCA[2]

[1] Istanbul Technical University, Istanbul/Turkey, akalino@itu.edu.tr
[2] Istanbul Technical University, Istanbul/Turkey, buzluca@itu.edu.tr

Abstract **- Measuring software maintainability is of vital importance for improving software product quality. Using a software quality model in the development life cycle, the quality of the system can be continuously evaluated and improved to reduce the maintenance cost. According to ISO/IEC 25010 Software Quality Models Standard, the maintainability characteristic of software product quality is composed of five sub characteristics; modularity, modifiability, reusability, analyzability and testability. This paper proposes a quality measurement model to evaluate the maintainability of software classes in terms of their reusability and modifiability characteristics in large-scale software systems. The model is based on software properties that are strongly related to reusability and modifiability, such as size, complexity, cohesion, coupling, and inheritance. First, our method categorizes metric values of software classes in the test system as low, medium and high. This categorization is done based on the average and median values for these metrics that are obtained from reference software systems. Then, the proposed measurement method uses the levels of the metrics to calculate the reusability and modifiability scores of each class in the system. The scores fall in one of the five categories; very low, low, medium, high, and very high. The developers of the software system can examine classes with low and very low scores and then refactor them if necessary. This continuous evaluation and refactoring during the development can increase the quality of the system and reduce maintenance costs. We applied our model on two large-scale industrial mobile applications and discussed the results with the development teams of the systems. We saw that our approach could reasonably grade classes on their reusability and modifiability characteristics.**

Keywords **- Quality Model, Software Metrics, Software Maintainability, Reusability, Modifiability.**

I. INTRODUCTION

High maintenance cost is considered as an important problem for software companies, because this complex process typically consumes 50-70% of the total effort allocated to a software development project [1], [2]. Main reasons for this high cost are frequently changing customer demands and technological developments that cause the software systems to be updated constantly, and the increasing complexity of the programs. It is expected that the software systems could be developed, modified, extended, and corrected in a short time without a degradation in its performance. Therefore, many researchers study on models for measuring the maintainability characteristic of the software product quality. The objective is to help developers in detecting modules that are not properly designed and need refactoring to decrease maintenance costs.

The maintainability is defined by IEEE standard glossary of Software Engineering as "the ease with which a software system or component can be modified to correct faults, improve the performance or other attributes, or adapt to a changed environment" [3]. According to ISO/IEC 25010 Software Quality Models Standard [4], the maintainability characteristic of software product quality is composed of five sub characteristics; modularity, modifiability, reusability, analyzability and testability.

In this paper, we propose a quality model for evaluating the maintainability of large-scale object-oriented software systems. Our model is based on the reusability and modifiability, because these sub characteristics are the main factors that affect the maintenance cost. Software developers need to know which classes can be reused at different stages of the systems or in other projects. Similarly, they need to know the cost of modifying classes for adding new features, refactoring or bug fixing. In our hierarchical model, to measure reusability and modifiability we use properties of object-oriented systems such as size, cohesion, coupling, complexity, and inheritance. We assigned software code metrics for each property. The metrics can be easily from programs obtained and using their values the maintainability of a system can be measured and enhanced. We tested our model on two large-scale industrial projects and discussed our findings with the developers of the projects. The results show that our model can be used to detect software classes that have low reusability or modifiability values because they were not properly designed. Such classes can be refactored to improve the maintainability of the system

The rest of the paper starts with related work in Section 2. Section 3 gives information definition of the maintainability in ISO / IEC 25010 standard and explains the structure of the proposed model. Steps of model construction is explained in Section 4. The empirical study and validation results are shared in Section 5. Final section concludes the paper.

II. RELEATED WORK

Many studies have used different prediction models for maintainability quality. Muthanna et al. [5], investigated the use of software design metrics to statistically evaluate the maintainability quality of large software systems. They model

can be applied only to procedural programs and it is not suitable for object-oriented software systems.

Aggarwal et al. [6] developed a Fuzzy model to measure the maintainability of the software system. They defined four factors affecting maintainability, namely average number of live variable, average life span of variables, average cyclomatic complexity, and the comments ratio. Their model classifies maintainability as very good, good, average, poor and very poor. There are total 81 rules in the model for all inputs and outputs. They test the model only on the projects developed by undergraduate engineering students.

Bagheri and Gasevic [7] evaluated the maintainability of software product line feature models. They proposed different structural metrics in their study and used manually the three main sub-characteristics of the maintainability: analyzability, changeability and understandability.

Rizvi and Khan [8] investigated a maintainability model including understandability and modifiability sub-characteristics on software design phase. They proposed different design metrics for quality measurement and constructed Maintainability Estimation Model for Object-Oriented software in the design phase (MEMOOD). The model generates only project-based results and it do not give information about software classes.

Bansiya and Davis [9] built a hierarchical quality model for object-oriented designs named QMOOD, which contains structural and behavioral design properties of classes, objects, and their relationships. QMOOD has four layers which include Design Quality Attributes, Object Oriented Design Properties, Object-Oriented Design Metrics, Object Oriented Design Components.

In our study, we created a model that uses code metrics to measure the reusability and modifiability characteristics of software classes in object-oriented systems. We evaluated our model on two large-scale industrial projects and validated the results with developers of the projects.

III. Software Maintainability and Hierarchical Model

Software maintainability is defined as the necessary effort for addition of new features to the software system and elimination of defects. In the ISO / IEC 25010: 2011 "System and software quality models" standard, the maintainability quality characteristic consists of five sub-characteristics including modularity, modifiability, reusability, analyzability and testability. In our proposed model, we consider reusability and modifiability characteristics, because they are main factors that affect the maintenance cost. Reusability is defined as the degree to which a software component can be used in more than one system or in implementing other software components. Modifiability is expressed as the degree to which a software system can be efficiently modified without introducing defects or degrading existing product quality [4].

In this study, we propose a hierarchical model that uses low-level code metrics to measure the high-level quality characteristic maintainability. The general structure of the model is shown in Figure 1. The values of class-based code metrics are obtained from the software systems. These values are used to calculate values of properties such as size, complexity, cohesion, coupling and inheritance. Properties determine the values of reusability and modifiability quality characteristics for each class. Using these values, developers can find classes that increase the maintainability costs and need refactoring.

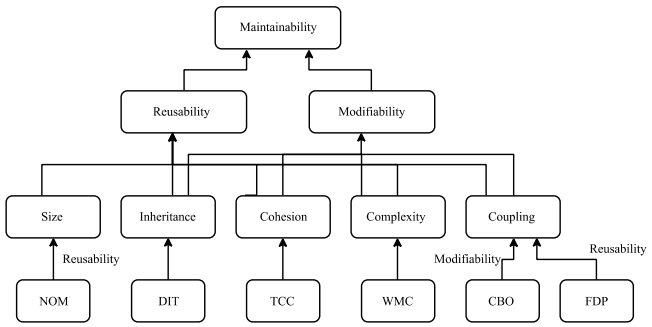


Figure 1: Hierarchical Structure of the Quality Model

IV. Construction of The Model

A. Property Selection

At the first stage of the model development, we select the main properties of object-oriented programs that can be used to determine reusability and modifiability values of classes. The selected properties and their influence on reusability and modifiability characteristics are explained below.

- Code size: The size of the source code negatively influences reusability of a class. For the simplicity of the model, effect of size on the modifiability is ignored because other factors are more dominant.
- Complexity: The complexity of the source code has a negative influence on the class' reusability and modifiability so that as complexity increases, reusability and modifiability of the class decreases.
- Cohesion: The cohesion of a class positively influences its reusability and modifiability. Therefore, when cohesion increases, the reusability and modifiability of the class increases.
- Coupling: The coupling between classes negatively influences the system's reusability and modifiability so that as the Coupling between classes increases, the reusability and modifiability of the system decreases.
- Inheritance: The degree of class inheritance has a negative influence on the reusability and modifiability of the system so that as the degree of class inheritance increases, the reusability and modifiability of the system decreases.

B. Metric selection

At the second stage of the model development, we determine metrics that should be used to measure the selected properties. Metrics are selected based on observations of authors about the metrics and on results of different research conducted in recent years [6-11]. Selected metrics are explained below.

Coupling Between Object Classes (CBO) has been proposed

by Chidamber & Kemerer [13]. If a class is coupled to more classes, its reusability and modifiability possibilities become lower.

Weighted Methods per Class (WMC) metric represents the sum of complexities of all methods in a class. Since high value of WMC metric indicates error-proneness, its reusability decreases and changes over the class take longer.

Depth of Inheritance Tree (DIT) metric calculates the depth of inheritance of given class in the class hierarchy. The deeper class have more properties and more methods, thus the possibility of its modifiability and reusability decreases.

Tight Class Cohesion (TCC) is a metric used to the relative number of methods directly connected via accesses of attributes. It is easy to refactor and understand the classes with high TCC so it has a positive effect on modifiability of a class.

Foreign Data Providers (FDP) is the number of different foreign attributes accessed by a method. since high FDP indicate that the class includes methods with different responsibilities, its reusability may become more unmanaged.

Number of Methods (NOM) metric shows the total number of methods of given class. High NOM value shows that the class has too much responsibility so it negatively affects reusability of a class.

The properties of the class, the metric value corresponding to the property, and the effect of the characteristic of the metric are shown Table 1.

C. Constructing the Prediction Model

Since values of different metrics change in very different ranges, to create an understandable model, first we categorize the values of the metrics as low (LOW), medium (MED), and high (HIGH) based on their typical values such as mean, median and standard deviation. The typical values for categorizing the metrics are obtained from open source projects that reached certain maturity. We used iPlasma tool [12] to gather metrics for each class from the reference open source software systems including WordPress-Android [16], FastHub [17], RxJava [18]. Since DIT metric values are in ordinal scale and may include asymmetric outlier values, median calculation (Med_m) is used for tagging it as shown in equation 1. Mean calculation ($Mean_m$) and standard deviation (Std) are used for other metrics, which are interval scale. Tagging rule for NOM, CBO, TCC, WMC, FDP metrics is shown in equation 2.

Our model assigns a numeric value (-1, 0, or 1) as score to each category based on the effect of the metric on the characteristic. For example, if a metric has a HIGH value and its affects the measured characteristic positively than this metric has the score +1. Table 2 shows the rules used to assign scores to metrics.

$$tag_c = \begin{cases} HIGH\ if (M_c > Med_m) \\ MED\ if (M_c = Med_m) \\ LOW\ if (M_c < Med_m) \end{cases} \quad (1)$$

Table 1: The corresponding properties of the metrics and the metric's effect on the characteristic

Property	Metric	The metric's effect on the characteristic	
		Reusability	***Modifiability***
Coupling	CBO	-	Negative
Coupling	FDP	Negative	-
Code Size	NOM	Negative	-
Inheritance	DIT	Negative	Negative
Cohesion	TCC	Positive	Positive
Complexity	WMC	Negative	Negative

Table 2: Score values corresponding tag values

	$Score_c$	
tag_c	Values for positive effect	Values for negative effect
HIGH	1	-1
MED	0	0
LOW	-1	1

$$tag_c = \begin{cases} HIGH\ if (M_c \geq Mean_m + Std) \\ MED\ if (M_c < Mean_m + Std\ or \\ \qquad M_c < Mean_m - Std) \\ LOW\ if (M_c \geq Mean_m r + Std) \end{cases} \quad (2)$$

Using the scores and the effect of metrics given in Table 1 the model calculates the output values for reusability and modifiability of each software class. The output value is the sum of all related metric scores for the quality characteristic as shown in equation 3.

$$Output_c = \sum_{k=0}^{n} Score_{c,k} \quad (3)$$

Output values change in the range between [-5,5] for reusability and [-4,4] for modifiability. In order to categorize the reusability and modifiability characteristics for each class, we also tag the output values. The output rule used in the reusability and modifiability model is shown Table 3. As a result of the output rules, we obtain a data set in the matrix form with n rows and m+1 columns for prediction where n is number of class and m is number of metric. As each column refers to a software metric, each row of the matrix refers to

Table 3: Output rules for tag values

Output	Reusability	Modifiability
VERY HIGH	$4 \leq Output_c \leq 5$	$3 \leq Output_c \leq 4$
HIGH	$2 \leq Output_c \leq 3$	$1 \leq Output_c \leq 2$
MED	$-1 \leq Output_c \leq 1$	$-1 \leq Output_c \leq 0$
LOW	$-3 \leq Output_c \leq -2$	$-3 \leq Output_c \leq -2$
VERY LOW	$-5 \leq Output_c \leq -4$	$Output_c = -4$

metric values for a software class. The last element of the row is the output tag value of software class, which is used to

categorize the reusability or modifiability characteristic.

Output intervals are not strict and can be flexible for different projects. For example, in order to find about ten percent of the worst-developed part of a project in terms of reusability, only those with an output value of "-5" can be defined as VERY LOW. Similarly, only the output value "4" can be defined as VERY HIGH to find the best developed about ten percent of a project in terms of modifiability.

V. Empirical Study and Evaluation

A. Model Results for Empirical Study

Two android mobile applications that are developed in industrial projects are examined for the evaluation of the model. They are very popular commercial projects that are extensively used in real-world. Project A has 1071 classes and Project B has 450 classes. Project A and Project B have developed in 18 months and 12 months, respectively. Since the projects were written with Java language, mean and median values in the model are obtained from open source projects written in Java. Open source project selection criteria are as follows: a) software must be written in the same language with software project which is selected to evaluate; b) open source software is selected as possible as from the similar topics to the selected project for evaluation; c) software must be at a certain maturity.

We applied our prediction model on a live release of the Project A and Project B. To tag classes according to reusability and modifiability characteristics, we first calculated mean and median values for each metrics using selected open source projects. Then we tagged metric values for Project A and Project B in accordance with the median and mean values. The metric tag values' numbers obtained for the projects are shown in Table 4 and Table 5. To create output values, we have labeled classes using the proposed model for each project. Some of the observed results are shown as:

- Project A has 173 classes labeled as VERY HIGH and 165 classes labeled as HIGH for reusability so it is considered that approximately thirty-third percent of Project A is highly reusable. Otherwise, Project A has 84 classes labeled as VERY LOW and 187 classes labeled as LOW for reusability so it can be said that about twenty-five percent of the project is quite low in terms of reusability.
- Project B has 14 classes labeled as VERY HIGH and 50 classes labeled as HIGH for reusability so it is considered that approximately fifteen percent of Project B is highly reusable. Otherwise, Project B has 18 classes labeled as VERY LOW and 166 classes labeled as LOW for reusability so it can be said that reusability of the project for about forty percent is rather difficult.
- Project A has 96 classes labeled as VERY HIGH and 229 classes labeled as HIGH for modifiability so it is considered that it is quite easy to make a modification for about thirty percent of the project. Otherwise, Project A has 67 classes labeled as VERY LOW and 270 classes labeled as LOW for modifiability so it can be said that about thirty-third percent of the project is quite low in terms of modifiability.
- Project B has 17 classes labeled as VERY HIGH and 34 classes labeled as HIGH for modifiability so it is considered that approximately ten percent of Project B is extremely easy to modify. Otherwise, Project B has 5 classes labeled as VERY LOW and 159 classes labeled as LOW for modifiability so it can be said that modifiability of the project for about thirty-third percent is rather difficult.

Table 4: Reusability results for Project A and Project B

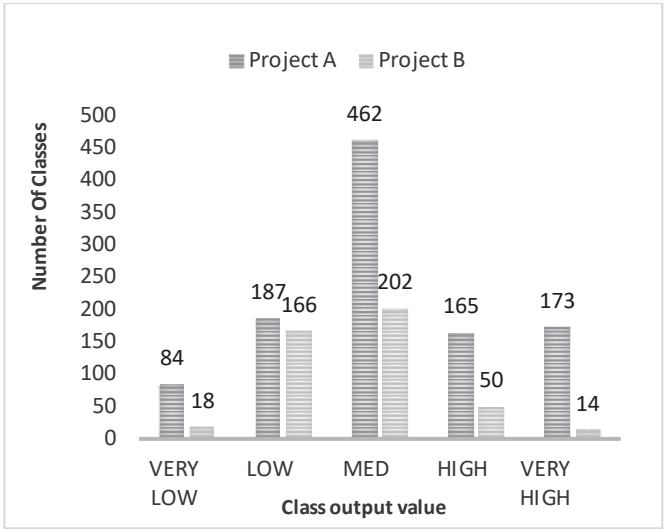
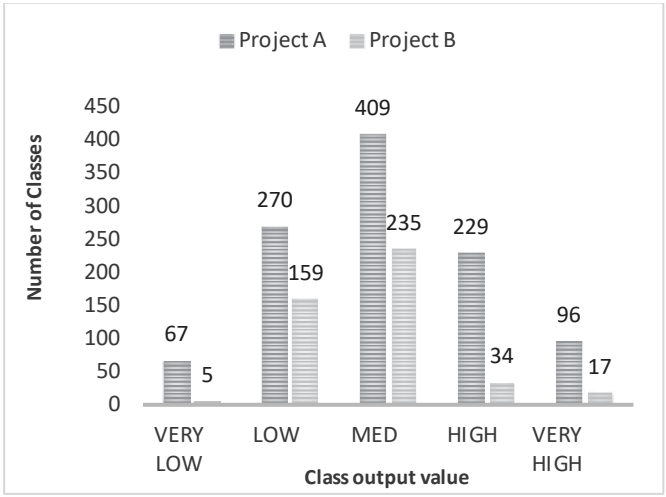


Table 5: Modifiability results for Project A and Project B



Class-based example results based on the values obtained from the model result are shown in Table 6 and Table 7. We interpreted the output values in terms of reusability and

Table 6: Class-based Reusability results

	FDP	NOM	DIT	TCC	WMC	Output
Class 1	LOW	HIGH	LOW	LOW	HIGH	MED
Class 2	LOW	HIGH	LOW	HIGH	LOW	HIGH
Class 3	HIGH	HIGH	MED	LOW	HIGH	VERY LOW

Table 7: Class-based Modifiability results

	CBO	DIT	TCC	WMC	Output
Class 1	LOW	LOW	LOW	HIGH	MED
Class 2	LOW	LOW	HIGH	LOW	VERY HIGH
Class 3	HIGH	MED	LOW	HIGH	VERY LOW

modifiability through the metrics that the classes have. The observed results for Table 6 and Table 7 are commented as:

- Class 1 has the low coupling, inheritance, cohesion and high complexity and size so its reusability is medium level. Also, since it has low coupling, inheritance, cohesion and high complexity, its modifiability is medium level. Thus, we can infer that its reusability and modifiability is medium in terms of maintainability.
- Class 2 has the low coupling, inheritance, complexity and high cohesion and size so its reusability is high level. Moreover, since it has low coupling, inheritance, complexity and high cohesion, its modifiability is very high level. As can be inferred, its reusability is high and also its modifiability is very high in terms of maintainability
- Class 3 has high coupling, size, complexity and low cohesion so its reusability is very low level. Also, since it has high coupling, complexity and low cohesion, its modifiability is very low level so that its reusability and modifiability is very low in terms of maintainability.

By examining the results obtained from the model, it becomes possible to deduce from which direction the class should be developed.

B. Model Validation and Threads to Validity

The proposed methodology is validated with the help of development teams. We use Analytical Hierarchical Process (AHP) to evaluate teams' opinions. It is a structured technique for analyzing complex multi-attribute decisions. It provides the decision maker to set priorities and make the best decision for solving a problem. Users of the AHP decompose their decision problem into a hierarchy of more easily comprehended sub-problems, each of which can be analyzed independently [14].

We prepared a survey for each property of the reusability and modifiability characteristics to obtain metric coefficients through AHP. In our study, we aim to evaluate maintainability

	Classes				
OUTPUT	HIGH	VERY HIGH	VERY LOW	LOW	LOW
Model_Reu	4	5	1	2	2
AHP_Reu	1.23	1.22	0.94	0.9	0.97
Model_Reu: Reusability Calculated with Model					
AHP_Reu: Reusability Calculated with AHP					

Table 8: Reusability values for Pearson Correlation Calculation

	Classes				
OUTPUT	VERY HIGH	MED	MED	HIGH	LOW
Model_Mod	5	3	3	4	2
AHP_Mod	1.12	0.99	0.96	1.0	0.8
Model_Mod: Modifiability Calculated with Model					
AHP_Mod: Modifiability Calculated with AHP					

Table 9: Modifiability values for Pearson Correlation Calculation

model's validation using the survey results. During the survey as size, cohesion, coupling, inheritance and complexity properties are calculated for reusability, cohesion, coupling, inheritance and complexity properties are calculated for modifiability. After the opinions of the teams were taken as a result of the survey, metric coefficients were obtained with AHP. Results are calculated for both reusability and modifiability characteristic. To identify the relationships between the model and AHP results, we analyzed Pearson correlation matrix for each value. In order to calculate the correlation between AHP and the model results, we assigned numeral values as 5, 4, 3, 2, 1 for VERY HIGH, HIGH, MED, LOW, VERY LOW model results, respectively.

Each correlation matrix has a correlation coefficient (R) for each result pair and R can range from - 1 and +1. While the value of R, +1, represents perfect positive correlation, the value of R, - 1, represents perfect negative correlation. While correlation coefficient values between 0.70 and 1.00 are accepted as a strong positive correlation in Pearson Correlation Analysis, values between - 1.00 and - 0.70 are accepted as a strong negative correlation [15].

The metrics for all the five properties of reusability and for all the four properties of modifiability were measured for both the projects. Initially, metric values were normalized and then were multiplied by their corresponding weight values according to AHP result for the projects. The results calculated with AHP and the proposed models is shown in Table 8 and Table 9.

Table 10 presents Pearson correlation coefficient between the proposed model and AHP model which is created as a result of the survey. As can be inferred, all the two coefficients depict significant positive correlation between the proposed

Table 10: Pearson Correlation Results for Project A and Project B

	Project A		Project B	
	R	R^2	R	R^2
Reusability	0.7679	0.5866	0.8819	0.7777
Modifiability	0.8282	0.681	0.7581	0.5747

model and AHP model. Therefore, it is clear that the proposed model may be able to predict a class's reusability and modifiability in terms of maintainability. The flow that occurs as a result of construction and evaluation stages has been shown Figure 2.

We need to explain some threats to validity of the proposed work for providing completeness and accuracy. The maintainability prediction model has been tested on the software projects developed in Java language. However, it is likely to be valid for the software projects which developed different object-oriented programing languages. Further research should be done to evaluate on different projects. Moreover, we used only 6 different source code metrics for development of the quality model in this study. Some of other static source code object-oriented metrics can be used for examining different properties except the properties in the proposed model.

VI. CONCLUSION

We have developed a quality model to quantify maintainability of the software classes in terms of their reusability and modifiability. We used two industrial software projects to evaluate the performance of our model in the real world. We structurally categorized software classes using code metrics and compared statistically our results to the developer teams' opinions. Consequently, our model shows that object-oriented metrics can effectively be used as predictors to evaluate maintainability of software systems.

Our model results provide different advantages in the software development phase. It generally helps to determine software class quality in terms of their maintainability level. Software developers can focus on software classes that have low quality. Moreover, the model gives information about reasons for their low or high quality levels, such as cohesion, coupling etc. Hence, developers can easily refactor classes to increase their quality level and significantly reduce the maintenance cost of the system.

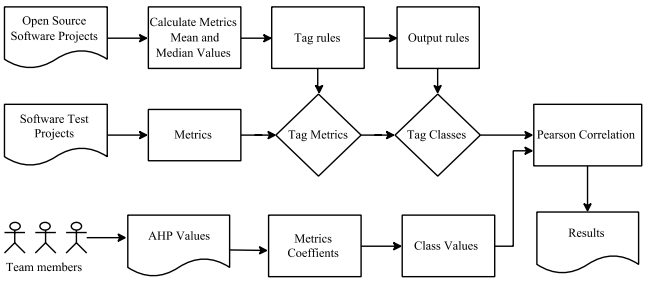


Figure 2: Construction and Evaluation of the Quality Model

REFERENCES

[1] Pigoski, Thomas M. *Practical software maintenance: best practices for managing your software investment,* Wiley Publishing, 1996.
[2] Sommerville, *Software Engineering*, 6th ed., Harlow, Addison-Wesley, 2001.S
[3] IEEE STD 610.2: *IEEE Standard Glossary of Software Engineering Terminology*, 1990.
[4] ISO/IEC 25010. ISO/IEC 25010:2011: Systems And Software Engineering – Systems And Soft-Ware Quality Requirements And Evaluation (Square) – System And Software Quality Models. Geneva: ISO, 2011.
[5] Muthanna, S., et al. "A maintainability model for industrial software systems using design level metrics." Reverse Engineering, 2000. Proceedings. Seventh Working Conference on. IEEE, 2000.
[6] Aggarwal, Krishan K., Yogesh Singh, and Jitender Kumar Chhabra. "An integrated measure of software maintainability." Reliability and maintainability symposium. Proceedings. Annual. IEEE, 2002.
[7] Bagheri, Ebrahim, and Dragan Gasevic. "Assessing the maintainability of software product line feature models using structural metrics." Software Quality Journal 19.3: 579-612,2011.
[8] Rizvi, S. W. A., and Raees A. Khan. "Maintainability estimation model for object-oriented software in design phase (memood).",2010.
[9] Bansiya, Jagdish, and Carl G. Davis. "A hierarchical model for object-oriented design quality assessment." IEEE Transactions on software engineering 28.1: 4-17, 2002.
[10] Genero, Marcela, et al. "Using metrics to predict OO information systems maintainability." International Conference on Advanced Information Systems Engineering. Springer, Berlin, Heidelberg, 2001.
[11] Singh, Charu, Amrendra Pratap, and Abhishek Singhal. "Estimation of software reusability for component based system using soft computing techniques." *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference-*. IEEE, 2014.
[12] C. Marinescu, R. Marinescu, P. Mihancea, D. Ratiu, and R. Wettel, "iPlasma:An integrated platform for quality assessment of object-oriented design," In Proceedings of the 21st IEEE International Conference on Software Maintenance, pp. 77-80, 2005.
[13] Chidamber, Shyam R., and Chris F. Kemerer. "A metrics suite for object oriented design." IEEE Transactions on software engineering 20.6 (1994): 476-493.
[14] Sharma, A., Kumar, R., Grover, P.S.: Estimation of Quality for Software Components - an Empirical Approach. ACM SIGSOFT Software Engineering Notes 33(5), 1–10 (2008)-639.
[15] Dagpinar, Melis, and Jens H. Jahnke. "Predicting maintainability with object-oriented metrics-an empirical comparison." Reverse Engineering, WCRE 2003. Proceedings. 10th Working Conference on. IEEE, 2003.
[16] https://github.com/wordpress-mobile/WordPress-Android/
[17] https://github.com/k0shk0sh/FastHub
[18] https://github.com/ReactiveX/RxJava