Research Article

Maintainability Evaluation for Object Oriented Software Metrics Using Tool Cohesion Inheritance (COIN)

Amit Sharma[1], Prabhat Kumar Vishwakarma[2]

amit.krsharma123@gmail.com[1], prapoo2012@gmail.com[2]

Article History: Received: 10 November 2020; Revised: 12 January 2021; Accepted: 27 January 2021; Published online: 05 April 2021

Abstract: In modern era, maintainability is an important part for software development that covers approx. 70-75% of development cost of the software system. It can allow the customer to adapt the software quickly and easily in an agile manner. Object oriented software metrics plays an important role for the designing of software development. Its features can be categorized into the object oriented metrics and the hierarchies of the class. In this paper, a tool named as COIN can help for evaluating the maintainability factors of object oriented software system using metrics like cohesion, coupling, inheritance and other object oriented metrics as well as through which we can analyzed the all metrics for the software system for evaluating the maintainability factors and testability also.

Keywords: Maintainability, Object-Oriented, Software.

1. Introduction

There are many examples of the software system that have not met with consideration of the users and have been consigned to forgetfulness or some immense activities on the plan and the accomplishment of the software systems. There are many examples that can be modified and updated multiples times and still not accepted by the users in current scenario the computation of software system from the end of developer can be done at developer end and after deliver the product in the market the maintainability can be measured by the end user in the market end. On these they defined as the computer software "is measured by how easily and how effectively it can be used by the precise set of users", that provide various kind of support and finally gives the different approach for the working environment. Now, in modern era multiple concurrency was plays an important role where multiple this eras gives numerous sorts of supports, for the conduction of finite tasks and these tasks will be performing on different type of workplace.

In modernization, every user is too much busy and time plays an important for the user to develop the software, to access the same with too much comfort and also wants to access the same software from other workspace. They think towards the different one. In the above context the maintainability is a main approach through which the success ration and the failure ration can be evaluated with the help of feedback of the end user over the software system. Multiple types of software matrices are analyses and investigated for evaluating the maintainability, reliability and other aspects of object oriented software system. These metrics are cohesion, coupling, inheritance, and quality metrics and so on. Most of the researchers can predicting the maintainability and software quality using the same [1], [2], [3], [4], [6], reliability [7], [8], testability [10], [11] and reusability [13].

Cohesion and inheritance are coupled together to compute the reusability of classes and class hierarchy. It is too much common that when the system is cohesive then the feature of metrices are easily inherited from the class and reuse in the universal manner. However, it is very hard to inherit the completely coupled software systems. Such designs are by and large reused by means of launch of the classes called verbatim reuse [13]. Daly et al [14] has been performed a study on the relationship between the depth of inheritance and the maintainability for the object oriented Software systems. According to their detection, three kinds of inheritance of legacy end up being preferable viable over five levels of inheritance. On another finding, Rachel Harrison et al [8] suggested that deeper inheritance trees in general leads to increased maintainability. However, Harrison et al revealed a different result through their study in [2]. Their results indicate the software systems are very easy to understand of manipulate their versions either the three or five levels of inheritance. It also shoes that the large systems in general suffer from low degree of understandability with or without inheritance. Dallal [15], Alshayeb [16], Ahmed [17] have already studied the effect of cohesion and coupling for software systems on software maintenance efforts and it is easily can be modifiable and scalable. consider an object oriented java project that generate an abstract syntax and can help for the measurement for every one of the classes and class hierarchy. the cohesion inheritance (COIN) tool is an augmentation of the class IN tool that is similar for comparable reason in [20].

The COIN device works for Java tasks and helps in recognizing class hierarchy orders that might be more intricate according to the perspective of programming maintainability.

Principle highlights of COIN are as per the following.

a. It gives cohesion, coupling, and its metrics of a given Java project at class level and furthermore at class hierarchy of importance level.

b. It sends out the metric informational collection into an accounting page (Excel-sheet) for investigation.

c. It shows the various leveled construction of various class hierarchy of command in the given task.

d. It gives a graphical perspective on practicality components of class hierarchy.

e. It gives a rundown of estimations of object oriented maintainability factors (modifiability, understandability and testability) of all the class hierarchy of the software systems.

This tool is very valuable in recognizing class hierarchies and for the measurement of the maintainability cost. It could be noticed that recognizing such progressions right off the bat in the plan stage would help create cost effective programming arrangements.

2. Software Metrics Tools

Several open source tools are used and available in the market for the evaluation of the parameters used in the designing of software system using the design metrics and the class level with their hierarchy and the levels are used for the packaging, in [21] CKJM tools is uses for the evaluations of various suites of the object oriented maintainability. On other side the tool, Eclipse plug-in 1.3.6 [22] used for the evaluations of data sets for both the class and its hierarchy. and supports Martin metrics suite as well. A third tool called JDepend is available as a Java plugin. JDepend creates plan quality measurements for every Java bundle and encourages in naturally assessing the plan quality regarding its extensibility, reusability, and practicality to adequately oversee bundle conditions [19]. (see additionally [18], [12]) Reliance Finder [9] incorporates measurements at different levels, for example, class level, technique level and bundle level. This aide in deciding reliance between various parts in a bundle. OOMeter is another device for measuring various qualities ascribes of curios delivered in a product improvement project. These incorporate plan level models and source codes.

As of late, Zhang [5] has presented a viable mechanized learning based way to deal with anticipate programming viability utilizing the genuine normal support endeavours, and a thorough arrangement of four-level progressive code measurements gathered by static code examination instruments. Recently, a survey of various existing and relevant software metric tools was made by the present authors in [20]. The tools are used for the analysis are eclipse 1.3.6, Vizz analyzer, object oriented meter, SD Metrics, JMT and dependency finder. it also includes, a new tool in the name of ClassIn (Class Inheritance) has also been introduced in [20] that support size and inheritance metrics at class level and class hierarchy level. The present tool ‘COIN’ extends ClassIn tool with additional metric support such as coupling, cohesion and size metrics. It also enables extracting metric data for factors like modifiability, understandability and testability at both classes level and hierarchy level. The existing tools support various size, cohesion, coupling, and package level metrics. However, most of the tools only focus on one or two types of metrics. For example, the main focus of most of the tools remains class level design metrics, size metrics, cohesion or coupling. The present tool is designed to give a broad range of class and level hierarchy of the depth of inheritance metrics, size, coupling metrics and cohesion metrics for analyzing at one place for the prediction of outside quality attributes, especially the software maintainability.

3. Classification of Maintainability Metrics

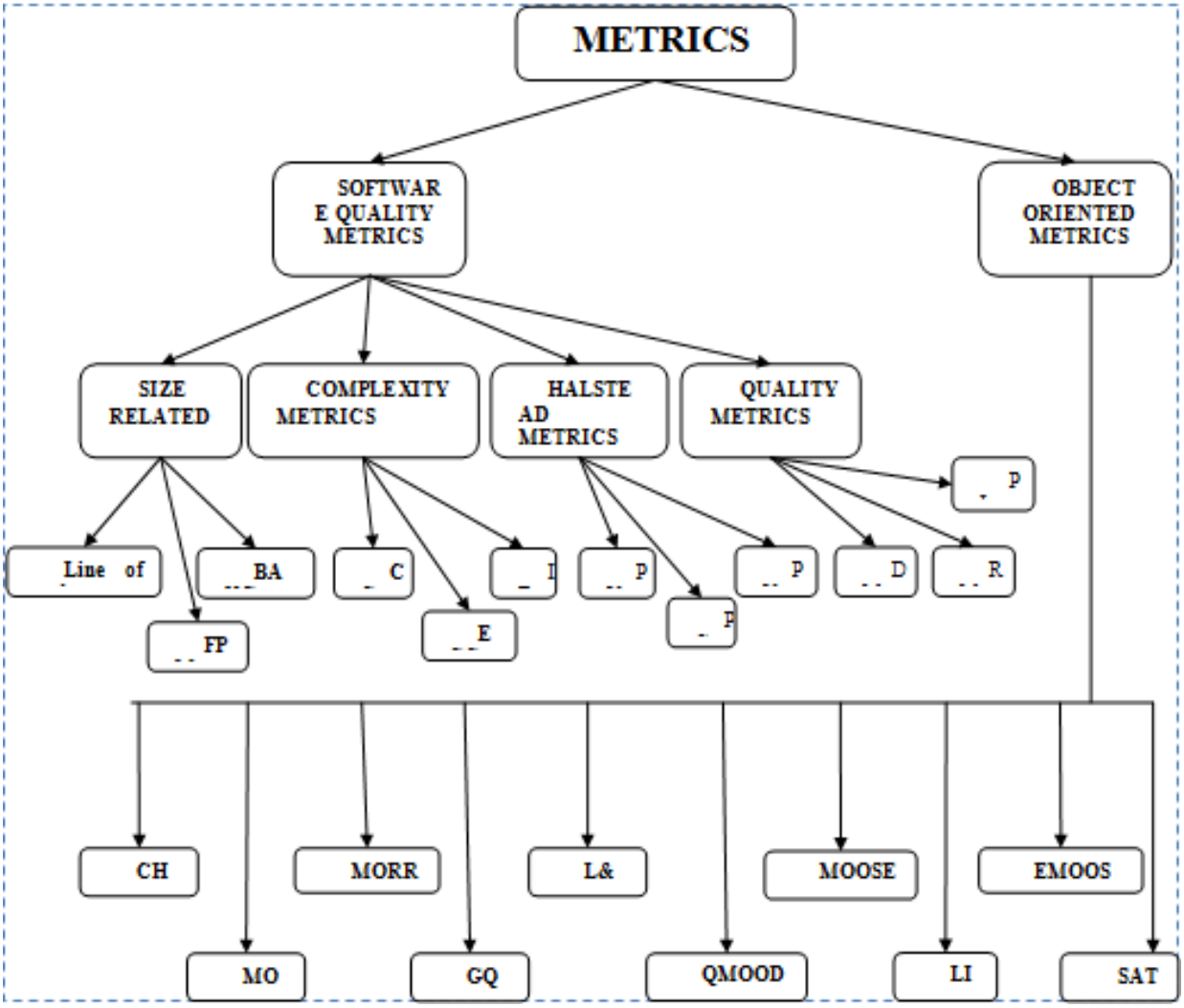


Figure 1. Classification of Metrics

In this phase, we can describe the functionality of the tool. Firstly, we can take the object oriented project as an input i.e. in file option open project and calculate the object oriented metrices for the class and class hierarchy. The data will be displayed on the mainframe window. The tool features are-

a) In the tab contained all the object oriented metrics like cohesion, inheritance, coupling and all the class and class level hierarchy. One may include either inherited attributes or / and inherited methods for measuring the inheritances cohesion metrics.

b) The 'Maintainability' tab contains modifiability, understandability and testability at class level and class hierarchy level.

c) The 'Analysis' tab contains analysis report of the class inheritance hierarchy.

d) The "HELP" button can be used for the user manual.

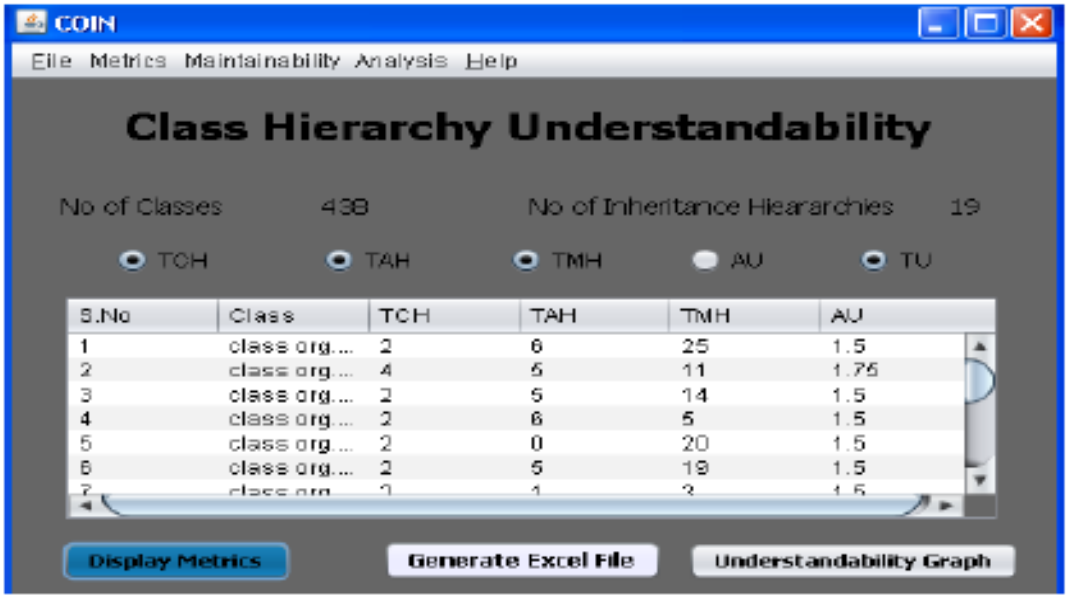


Figure 2. Understandability of Class Hierarchy Level Snapshot of Tool Page

The above diagram shows that the total number of class and class hierarchy of the object oriented projects. In which the user can select the metrics data and this data can be displayed in an excel format.

In figure 3, we can show the 'Average Understandability' factor for each of the class hierarchies along with MaxDIT and NOCC values which are highly correlated with understandability factor. In figure 3 we can display the list of metrics for class inheritance hierarchy. low maintainability and high maintainability will be represented as the green and red color for the representation of class hierarchy. The highlighted metric data represents correlation with maintainability factors.

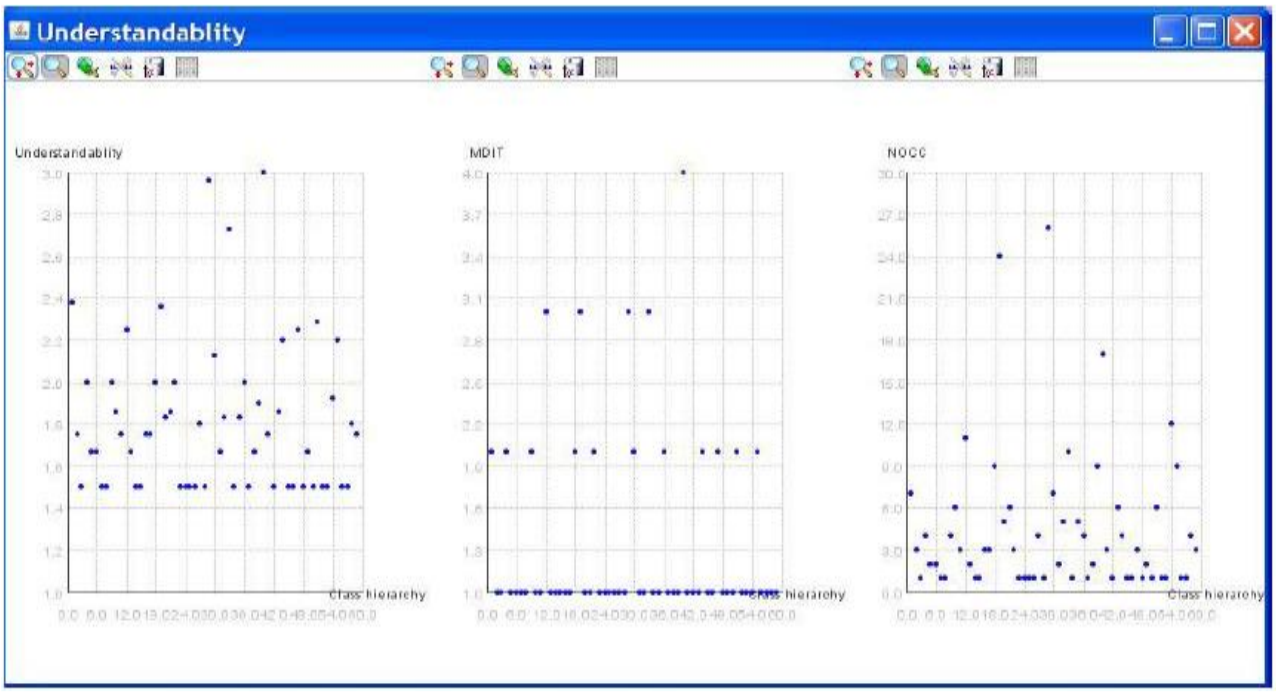
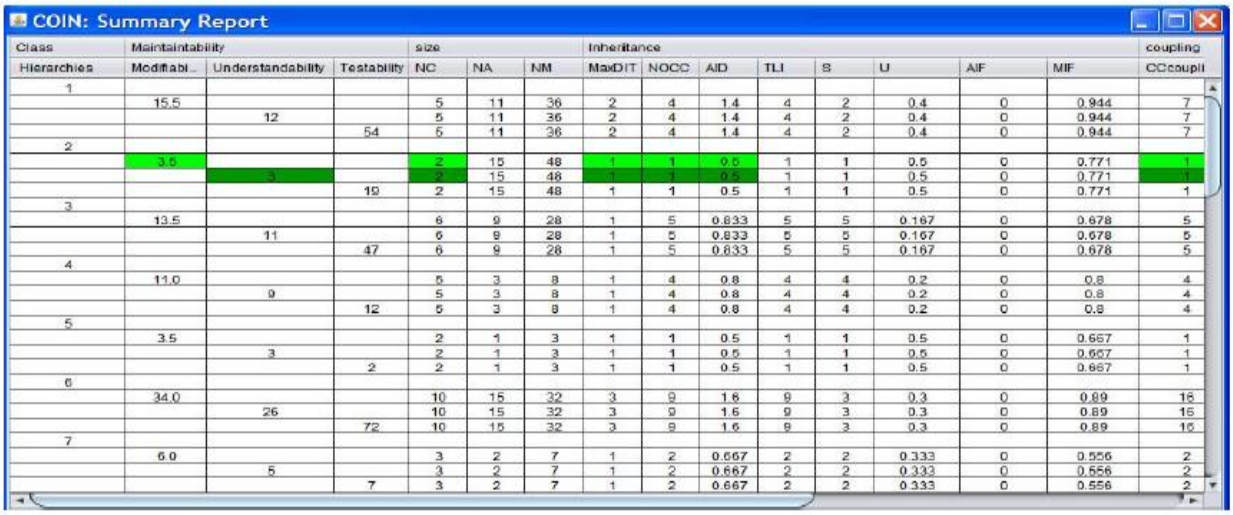


Figure 3. Understandability Factor for Different Hierarchies



COIN: Summary Report

Class	Maintaintability			size			Inheritance								coupling
Hierarchies	Modifiabi...	Understandability	Testability	NC	NA	NM	MaxDIT	NOCC	AID	TLI	S	U	AIF	MIF	CCcoupli
1															
	15.5			5	11	36	2	4	1.4	4	2	0.4	0	0.944	7
		12		5	11	36	2	4	1.4	4	2	0.4	0	0.944	7
			54	5	11	36	2	4	1.4	4	2	0.4	0	0.944	7
2															
	3.5			2	15	48	1	1	0.5	1	1	0.5	0	0.771	1
		3		2	15	48	1	1	0.5	1	1	0.5	0	0.771	1
			19	2	15	48	1	1	0.5	1	1	0.5	0	0.771	1
3															
	13.5			6	9	28	1	5	0.833	5	5	0.167	0	0.678	5
		11		6	9	28	1	5	0.833	5	5	0.167	0	0.678	5
			47	6	9	28	1	5	0.833	5	5	0.167	0	0.678	5
4															
	11.0			5	3	8	1	4	0.8	4	4	0.2	0	0.8	4
		9		5	3	8	1	4	0.8	4	4	0.2	0	0.8	4
			12	5	3	8	1	4	0.8	4	4	0.2	0	0.8	4
5															
	3.5			2	1	3	1	1	0.5	1	1	0.5	0	0.667	1
		3		2	1	3	1	1	0.5	1	1	0.5	0	0.667	1
			2	2	1	3	1	1	0.5	1	1	0.5	0	0.667	1
6															
	34.0			10	15	32	3	9	1.6	9	3	0.3	0	0.89	16
		26		10	15	32	3	9	1.6	9	3	0.3	0	0.89	16
			72	10	15	32	3	9	1.6	9	3	0.3	0	0.89	16
7															
	6.0			3	2	7	1	2	0.667	2	2	0.333	0	0.556	2
		5		3	2	7	1	2	0.667	2	2	0.333	0	0.556	2
			7	3	2	7	1	2	0.667	2	2	0.333	0	0.556	2

Figure 4. Maintainability Factor Hierarchy Shows List of Metric

4. Tool Architecture

In the figure no 4 the architecture of the tool will be defining in three modules-

Module 1- in this module we can take an object oriented software project considered as an input and can be easily compute the number of classes in the project using reflection classes. These reflection classes API allow the code in java for computing the object and classes objects run time and reflection classes also allows for the inspection of methods, field and classes in the object oriented projects. The abstract syntax tree also records the java documents and the source code comment that can be stored in the java program.

Module 2-in this module we can calculate the cohesion between the class member using the abstract syntax tree i.e. AST.

Module 3- in this model the metrics tool calculates the datasets of metrics at class and class hierarchy level both.

This tool provides the output as in the form of datasets for the metrics in excel format, class hierarchy and the graph for analyzing the maintainability of the class inheritance hierarchy.

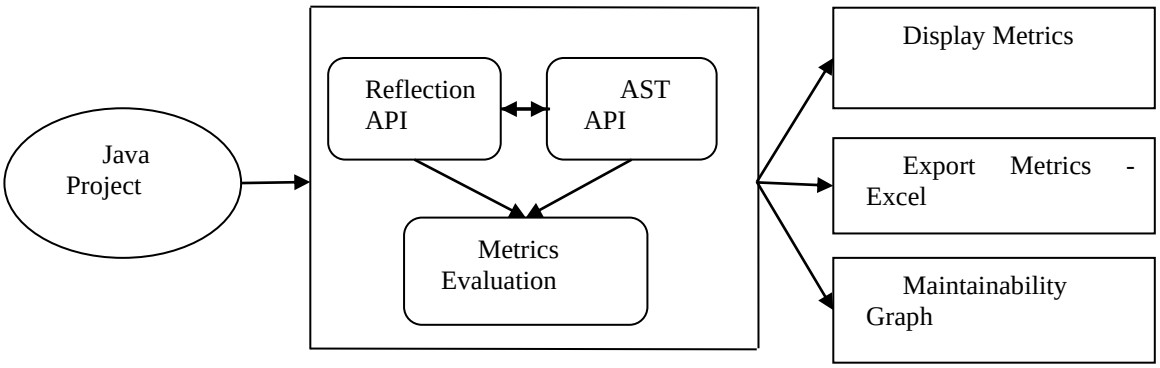


Figure 5. COIN Tool Architecture

5. Illustration

In order to demonstrate the usefulness of COIN tool, two example cases are presented in this section. In the first case, a class hierarchy is taken with a high value of modifiability, understandability, testability where as in second case these values are very low.

In case 1, MaxDIT = 6 and NOCC = 132, which is reasonably high. As can be seen from the data in Table III, maintainability factors are also high due to high depth and width of class hierarchies. Similarly, testability parameter Tassert turns out to be quite high due to high values of MaxDIT, NOCC and TMI. In the second case,

MaxDIT = NOCC = 1. So, the modifiability, understandability and the testability are all having low metric values.

Metrics	NC	NM	TMI	Max DIT	NO CC	AID	TLCOM3	TLCOM4	TM	TU	Tassert
Case-1	133	1193	14021	6	132	3.6992	8527	3332	871	625	12537
Case-2	2	6	0	1	1	0.5	1	1	3.5	3	1

6. Conclusion

A comprehensive metric tool COIN has been developed that supports various design level metrics and also help determine their correlation with three main software maintainability factors like modifiability, understandability and testability. It displays the data both in tabular and graphical form. Using an analysis of metric data set of a large number of class hierarchies, it is concluded that MaxDIT, NOCC, AID and TMI are highly correlated with the maintainability factors. In essence, the tool may prove to be very useful for software companies and developers to assess class structures for their maintainability.

References

1. Hanenberg, S., Kleinschmager, S., Robbes, R., Tanter, É. and Stefik, A. 2014. An empirical study on the impact of static typing on software maintainability. Emp. Softw.
2. Harrison, R., Counsell, S. and Nithi, R. 2000. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. J. Sys. and Softw. 52, 2 (2000), 173-179.
3. Chen, J. C. and Huang, S. J. 2009. An empirical analysis of the impact of software development problem factors on software maintainability. J. Syst. Softw. 82, 6 (2009), 981- 992.
4. Genero, M., Manso, E., Visaggio, A., Canfora, G. and Piattini, M. 2007. Building measure-based prediction models for UML class diagram maintainability. Emp. Softw. Eng. 12, 5 (2007), 517-549.
5. Zhang, W., Huang, L., Ng, V. and Ge, J. 2015.SMPLearner: learning to predict software maintainability. Auto. Softw. Eng. 22, 1 (2015), 111-141.
6. Basili, V. R., Briand, L. C. and Melo, W. L. 1996. A validation of object-oriented design metrics as quality indicators. IEEE Trans. Soft. Eng. 22, 10, 751-761.
7. Subramanyam, R. and Krishnan, M. S. 2003. Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. IEEE Trans. Soft. Eng. 29, 4 (2003), 297-310.
8. Cartwright, M. and Shepperd, M. 2000. An empirical investigation of an object-oriented software system. IEEE Trans. Soft. Eng. 26, 8 (2000), 786-796.
9. Dependency Finder tool, www.depfind.sourceforge.net/
10. Baudry, B. and Le Traon, Y. 2005. Measuring design testability of a UML class diagram. Info. and Softw. Techn. 47, 13 (2005), 859-879.
11. Bruntink, M. and van Deursen, A. 2006. An empirical study into class testability. J. Syst. and Softw. 79, 9 (2006), 1219-1232.
12. VizzAnalyzer tool, www.arisa.se.
13. Karunanithi, S. and Bieman, J. M. 1993. Candidate reuse metrics for object oriented and Ada software. In IEEE METRICS, (May 1993), 120-128.
14. Daly, J., Brooks, A., Miller, J., Roper, M. and Wood, M. 1996. Evaluating inheritance depth on the maintainability of object-oriented software. Emp. Softw. Eng. 1, 2(1996), 109-132.
15. Dallal, J. A. 2011. Measuring the Discriminative Power of Object-Oriented Class Cohesion Metrics. IEEE Trans. Soft. Eng. 37, 6 (2011), 788-804.
16. Alshayeb, M. 2013. On the relationship of class stability and maintainability. IET Softw. 7, 6 (2013), 339-347.
17. Ahmed, M. A. and Al-Jamimi, H. A. 2013. Machine learning approaches for predicting software maintainability: a fuzzy-based transparent model. IETSoftw. 7, 6 (2013), 317-326.
18. JMT tool, http://www-ivs.cs.uni-magdeburg.de/sweng/agruppe/forschung/tools/.
19. JDepend tool, http://clarkware.com/software/JDepend.html.
20. Reddy, B. R. and Ojha, A. 2014. ClassIN: A Class Inheritance Metric Tool. In New Perspectives in Information Systems and Technologies, Volume 2, (15-18April 2014), 113-119.
21. CKJM metric tool, www.spinellis.gr/sw/ckjm/.
22. Eclipse plug-in 1.3.6 tool,www.sourceforge.net/projects/metrics/.