Predicting Reliability changes using Object Oriented Metrics

PAUL ŢIRBAN, Babeş-Bolyai University Cluj Napoca

Reliability is an important factor but is hard to asses it during the development of an application. In this paper we propose an approach to predict reliability changes of object oriented applications using object oriented metrics from multiple versions of an application. Further on, after computing this criterion, the current work investigate how the new criterion that we propose correlates with changes in object oriented metrics using statistical methods. Analyzing results from the correlation, we can see that there is a strong correlation between our criterion and WMC, RFC and LOC metrics, a moderate correlation with CBO, LCOM5 and a very weak correlation with NOC metric. The approach was applied to seven projects (a total number of 1.462.178 classes from 678 total versions), with sizes ranging from small to medium, in order to assure statistical relevance and project diversity (not resumed to a single project with several versions).

1. INTRODUCTION

Complex systems needs to be developed within time and budget, are usually done incrementally over a long period of time and, after delivering them, developers have to maintain and improve them based on the new requirements.

Fore some of these systems (like safety critical systems for example) one of the most important quality factor to be considered is reliability, which expresses the degree to which a system is free of faults or can recover to a stable state after failure. Nina S Godbole points out that Quality goals must be clearly defined, effectively monitored and rigorously enforced [Godbole 2004] in order to deliver reliable software systems.

Being hard to asses reliability during the development cycle of a software, the assessment of reliability is usually done at the end of the cycle so that corrections to improve it cost more time and money. Also is very hard to find a metric that computes exactly the reliability of a software during the development phase and current methods of predicting reliability are mainly focusing on complexity of the software without taking into consideration the facts that object oriented programming imposes a special structure of the code and special interactions between constructs are not taken into consideration.

Since, using the current IDEs available, is easy to compute object oriented metrics for a version of a software and see how the metrics evolve between multiple versions we focused on finding a method to relate this changes in metrics to changes in reliability. After proposing our method of predicting reliability changes based on OO metrics, we applied statistical methods in order to validate if the changes in the metrics correlates with the changes in our criteria of predicting reliability changes.

The paper is organized as follows: Section 2 contains preliminary information about reliability, object oriented metrics used in our research and related work. In Section 3 we present our approach of

Author's address: Babeş-Bolyai University, str.M.Kogalniceanu 1, 400084, Cluj-Napoca, Romania, email: tirban@cs.ubbcluj.ro

In: Z. Budimac (ed.): Proceedings of the SQAMIA 2018: 7th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Novi Sad, Serbia, 27–30.8.2018. Also published online by CEUR Workshop Proceedings (http://ceur-ws.org, ISSN 1613-0073)

predicting reliability changes using OO metrics and the results of our case study. In the last section we conclude the paper and discuss future work.

2. PRELIMINARIES

2.1 Reliability as Software Quality Factor

Starting with the [McCall et al. 1977], Reliability has been introduced as a quality factor, and then it has been present as a software quality factor as shown in Table I.

Table I. Realiability in software quality models

SQ Model	Reliability	Subfactors
McCall	*	consistency, accuracy, fault tolerance
Boehm	*	self containedness, accuracy, completeness, robustness, integrity, consistency
FURPS	*	frequency and severity of failure, recoverability, predictability, accuracy, mean time between failures
ISO 9126	*	maturity, fault tolerance, recoverability, compliance
Dromey	*	*same as ISO 9126*
ISO 25010	*	maturity, availability, fault tolerance, recoverability

Due to the fact that applications have evolved and changed over time, the subfactors that characterized it changed, in order to capture the relevant aspects and requirements of applications. As part of reliability in earlier models, accuracy, which represents the measure of required precision in computation, was present, but has lost its importance in the newer models, being replaced by maturity and recoverability.

In ISO 25010, availability is also recognized as a key characteristic of software, capturing the aspect that software should be ready to be used and accessible. Looking at nowadays mobile and web applications the importance of availability is even more important.

”Degree to which a system, product or component meets needs for reliability under normal operation” [ISO25010 2011] defines Maturity subfactor. Maturity captures the aspect that the frequency of faults decreased in a mature application, since it has gone through a series of development cycles. Even in case of failures, the software product is able to re-establish its level of performance and recover data handled by application, which are the defining aspects of recoverability.

2.2 Object Oriented Metrics

Object oriented metrics provide information about code and help developers in discovering design flows and are widely used in software development. There are multiple tools that allows us to compute the metrics, some integrated in IDEs or frameworks and some independent, that compute OO metrics: SourceMeter [SourceMeter 2018], Eclipse Metrics [Metrics 2016], NDepend[Ndepend 2016], JHawk[JHawk 2016].

Until now a lot of metrics have been proposed and new metrics continue to appear in the literature regularly. Some examples are the metrics proposed by [Abreu 1993], [Abreu and Rogerio 1994], [Chidamber and Kemerer 1994], [Li and Henry 1993], and the MOOD metrics proposed in [Abreu 1995]. [Marinescu 2002] has classified these metrics according to four internal characteristics that are essential to object-orientation: coupling, inheritance, cohesion and structural complexity.

In the current study we have chosen the following metrics: WMC, CBO, RFC, LCOM5, NOC and LOC that you can find bellow as they are defined by SourceMeter [SourceMeter 2018].

—*Weighted Methods per Class* (WMC) measures the complexity of the class expressed as the number of independent control flow paths in it. It is calculated as the sum of the McCabes Cyclomatic Complexity (McCC) values of its local methods and init blocks; [SourceMeter 2018]

—*Coupling Between Objects* (CBO) measures number of directly used other classes (e.g. by inheritance, function call, type reference, attribute reference). Classes using many other classes highly depend on their environment, so it is difficult to test or reuse them; furthermore, they are very sensitive to the changes in the system; [SourceMeter 2018]

—*Response for a Class* (RFC) metric is defined as the number of local (i.e. not inherited) methods in the class (NLM) plus the number of directly invoked other methods by its methods or attribute initializations (NOI); [SourceMeter 2018]

—*Lack of Cohesion in Methods 5* (LCOM5) measures the lack of cohesion and computes into how many coherent classes the class could be split. The value of the metric is the number of connected components in the graph not counting those, which contain only constructors, destructors, getters, or setters; [SourceMeter 2018]

—*Number of Children* (NOC) measures the number of classes, interfaces, enums and annotations which are directly derived from the class; [SourceMeter 2018]

—*Lines of Code* (LOC) measures the number of code lines of the class, including empty and comment lines, as well as its local methods; however, its nested, anonymous, and local classes are not included. [SourceMeter 2018]

The chosen metrics were selected starting from the metrics proposed in CK Metrics [Chidamber and Kemerer 1994] and the ones that are computed by SourceMeter [SourceMeter 2018].

2.3 Related work

The systematic literature review provided in [Jabangwe et al. 2015] showed that the most investigated quality characteristic is reliability and most of the studies are linked to fault-proneness.

WMC, DIT, RFC, NOC and CBO from CK Metrics appear to be useful to predict class fault-proneness during the high- and low-level design phases of the life-cycle, as presented in [Basili et al. 1996].

In MetricsReloaded plugin from Intellij [MetricsReloaded 2017] a criterion for assessing Reliability at the method level is introduced ***Quality Criteria Profile (Reliability)***:

$$QCP\_RLBTY = N + (2 * NEST) + (3 * V(g)) + BRANCH + \\ + CONTROL + EXEC$$

where: N is the Halstead Length metric for a method, NEST is the maximum nesting depth of each method, V(g) is the cyclomatic complexity of each non-abstract method, BRANCH is the total number of non-structured branch statements in each method, CONTROL is total number of control statements in each method and EXEC is the total number of executable statements in each method.

Relationship between existing class level OO metrics and fault proneness over the multiple releases is investigated in [Rathore and Gupta 2012]. Univariate logistic regression analysis to evaluate each metric (WMC, DIT, NOC, CBO, RFC, LCOM, CC) for their potential to predict faults independently is used.

[Lincke and Lowe 2006] links software quality factors to object oriented metrics, based on the semantics of the syntactical constructions that are involved in evaluation of the factor and the computation

of the metric. The approach had resulted in a Compendium [Lincke and Lowe 2013] containing 37 quality attributes (factors and criteria) and 23 metrics, together with their influences.

Neural network are also used to predict software readiness. In [J.Quah and Thwin 2002] compares the prediction results from multiple regression model and neural network model, where object-oriented design metrics are used as the independent variables (for coupling - CBO, RFC, IC, CBM, for inheritance - DIT, NOC, for complexity - WMC, AMC, for memory allocation - NOMA), and number of faults as dependent variable.

3. ASSESSING RELIABILITY USING OBJECT ORIENTED METRICS

3.1 Approach

This paper proposes a new method for assessing reliability changes in a project using OO metrics in object oriented software. The main focus of this research is to introduce a way to measure reliability using object oriented metrics - WRC and to see if changes of this indicator correlates with object oriented metrics that we took under consideration.

We selected the metrics that we use when defining our own criterion, based on the information from ARISA Compendium, the experience that we gathered when trying to asses Maintainability changes as described in [Motogna et al. 2016], but also starting from the *Quality Criteria Profile (Reliability)* criterion from MetricsReloaded. Based on this metric from MetricsReloaded and on the information from ARISA [Lincke and Lowe 2013] we define *Weighted Reliability per Class* indicator as:

$$WRC = \frac{\sum_{m \in class\ C} M\_RELIABILITY(m) \times WMC}{no\_of\_methods\_in\_class\_C}$$

where:

$$M\_RELIABILITY = HPL + (2 * NL) + (3 * McCC) + NOS$$

where: HPL is the Halstead program length metric for a method; NL is the complexity of the method expressed as the depth of the maximum embeddedness of its conditional, iteration and exception handling block scopes; McCC is complexity of the method expressed as the number of independent control flow paths in it; NOS is number of statements in the method.

After computing WRC for each class from each version of the projects, we used statistical analysis using R Studio in order to see if the variations of WRC correlates with WMC, CBO, LCOM5, RFC, NOC and LOC metrics. Starting from [Marinescu 2002] research that classify metrics as describing four internal characteristics of the object oriented paradigm (inheritance, structural complexity, coupling and cohesion) we selected the above metrics so that there is at least one representative metric for each characteristic from [Marinescu 2002].

For this the first step was to run the Shapiro-Wilk test [Royston 1982], for each of the variables to see if they are normally distributed. Now because the Shapiro-Wilk test is limited to a number of maximum 5000 observations, we also tested normal distribution with Anderson-Darling normality test [Stephens 1974]. After this the next step was to use either Pearson's correlation (if both variables are normally distributed) or Spearman's correlation (if one or both variables are not normally distributed) [Correlations 2018]. The last step was to interpret the results from the correlations. The flow of the process can be see in Figure 1.

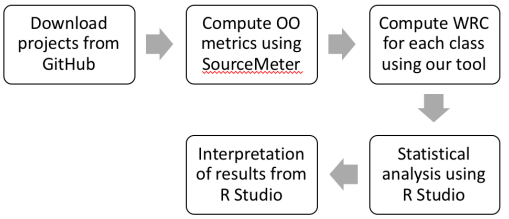

Fig. 1. Our approach of assessing correlation between Reliability and OO metrics.

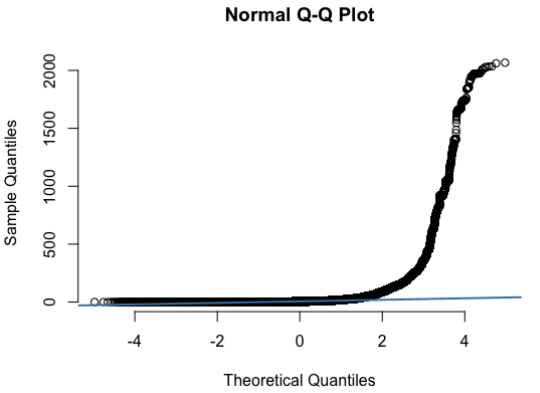

Fig. 2. WMC distribution plot in R Studio.

3.2 Case Study and Results

Gathering data for study was not easy, since there are no databases from where metrics from multiple versions of projects to be considered. Moreover, it's hard to find data from commercial projects, and because of this we moved our attention to the open source projects. The first thing that we did was to select some projects from GitHub of different sizes and with different number of releases and to retrieve the releases of those projects using our own script. Project used in our study are log4j, [Log4j 2018], ElasticSearch [Elasticsearch 2018], Guava [Guava 2018], Apache Camel [Camel 2018], Apache Cassandra [Cassandra 2018], Apache Groovy [Groovy 2018], Apache JMeter [JMeter 2018]. Details about the projects used can be found in Table II. Looking at Table II we can see that there is a total number of 1.462.178 classes from 678 versions (the smallest project is log4j with 64 versions and a total number of classes of 25.410 in those 64 versions).

Table II. Details of projects used in the case study

Project	Number of versions	Total number of classes
log4j	64	25.410
elasticsearch	84	309.000
guava	44	220.147
apache-camel	28	80.729
apache-cassandra	220	466.992
apache-groovy	167	276.489
apache-jmeter	71	83.411

A total number of 1.462.178 classes from 678 versions were used in the statistical analysis.

After this the next challenge that we faced was to automatically compute the OO metrics that we needed in our research. For this we searched for a tool that can do this and after trying multiple tools we stopped at SourceMeter since it was the most suitable for our needs. Using SourceMeter we computed WMC, CBO, LCOM5, RFC, NOC and LOC metrics for our project and then using our own tool computed WRC for each class. Table III shows the summary of the metrics computed by SourceMeter and the summary for our WRC criterion computed with our own tool for log4j [Log4j 2018] - the project with the lowest total number of classes, and appache-cassandra [Cassandra 2018] - the project with the highest number of total classes. Looking at the values we can see that there are differences from

project to project, which are normal since there are differences in the size of the projects, but we can also see that the values are related to the tool that was used to compute the metrics (in our case SourceMeter) and using a different tools different values can be obtained, that's why we used data computed with the same tool in order to have valid data for computing the changes and correlations. In Table IV we can see a summary of the metrics on the data from all projects, data that was used for analyzing correlations described in Table V.

Table III. Summary of metrics used in the case study for 2 projects (log4j and apache-cassandra)

Metric	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
log4j						
WMC	0.00	2.00	5.00	11.37	14.00	223.00
CBO	0.000	1.000	3.00	3.534	5.000	52.000
RFC	0.00	3.00	7.00	10.46	13.00	212.00
LCOM5	0.000	1.000	1.000	1.543	2.000	29.000
NOC	0.000	0.000	0.000	0.247	0.000	32.000
LOC	2.00	18.00	48.00	92.62	112.00	1686.00
WRC	0.0	30.0	107.0	234.7	276.0	5234.0
apache-cassandra						
WMC	0.00	2.00	5.00	15.84	14.00	1048.00
CBO	0.000	1.000	3.000	6.179	8.000	149.000
RFC	0.00	3.00	7.00	13.86	16.00	617.00
LCOM5	0.000	1.000	1.000	1.693	2.000	109.000
NOC	0.000	0.000	0.000	0.261	0.000	95.000
LOC	1.00	14.00	34.00	93.72	89.00	4347.00
WRC	0.0	13.0	64.0	288.9	239.0	16315.0

Table IV. Summary of all the metrics for all the classes from all versions of the projects in one table

Metric	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
WMC	0.00	2.00	4.00	12.96	11.00	2067.00
CBO	0.000	1.000	3.000	5.488	7.000	149.000
RFC	0.00	3.00	6.00	12.82	14.00	1840.00
LCOM5	0.000	1.000	1.000	1.872	2.000	1400.000
NOC	0.0000	0.0000	0.0000	0.4182	0.0000	1454.0000
LOC	1.00	11.00	27.00	78.33	72.00	19067.00
WRC	0.0	15.0	56.0	227.7	194.0	60211.0

Using R Studio we gathered all the classes from all versions of the projects in one table and starting the correlation analysis. After running normal distribution tests (using the Shapiro-Wilk and Anderson-Darling normality test) for WRC, WMC, CBO, LCOM5, RFC, NOC and LOC we determined that all variables are not normally distributed (for all the p-value $<$ 2.2e-16). For example the distribution of WMC can be seen in Figure 2.

Since the variables are not normally distributed we continue the analysis using the Spearman's correlation and tested the correlation between WRC and object oriented metrics. Results of the analysis can be seen in Table V.

Analyzing results from the correlation, available in Table V we can see that there is a strong correlation between our method and WMC, RFC and LOC metrics, a moderate correlation with CBO, LCOM5

Table V. Results of Spearman's correlation

	rho	p-value	correlation interpretation
WRC-WMC	0.7428248	$< 2.2e\text{-}16$	strong
WRC-CBO	0.4281653	$< 2.2e\text{-}16$	moderate
WRC-LCOM5	0.4279371	$< 2.2e\text{-}16$	moderate
WRC-RFC	0.7427667	$< 2.2e\text{-}16$	strong
WRC-NOC	0.09204933	$< 2.2e\text{-}16$	very weak
WRC-LOC	0.780197	$< 2.2e\text{-}16$	strong

and a very weak correlation with NOC metric. Looking at the p-value, we can see that it is lower than 0.05 so this means that the correlations are statistically significant.

Changes in WMC, RFC and LOC metrics will strongly impact the Reliability, CBO and LCOM5 will have a moderate impact and changes in NOC will have a weak impact.

In [Lincke and Lowe 2013] WMC, CBO, RFC, NOC and LOC are categorized as related to Reliability and LCOM is Highly Related. We can see that some correlations from ARISA Compendium are also reflected in the present study, but there are some changes that might result from the fact that different tools are used to compute the metrics and values differ from tool to tool, or from the fact that data from the study is limited to the projects that we managed to gather and extending the data might allow us to find a better correlation. In order to draw a stronger conclusion further study needs to be done using different tools and more data (as discussed also in the future work section).

3.3 Threats to Validity

The selection of metrics was done to ensure that key characteristics of the software are covered, but a further extension of this selection is required to be investigated in order to better capture the impact of each metric. Also since there are differences between how each tool compute the metrics and because gathering enough data is a challenge another threat to validity is the size of the sample data.

4. CONCLUSIONS AND FUTURE WORK

The paper propose an approach to predict reliability of object oriented applications using object oriented metrics from multiple versions of an application.

We proved that correlations can be found between Reliability and OO metrics using statistical analysis. Analyzing results from the correlation, we showed that there is a strong correlation between our method and WMC, RFC and LOC metrics, a moderate correlation with CBO, LCOM5 and a very weak correlation with NOC metric.

The approach was applied to seven projects, with sizes ranging from small to medium, in order to assure statistical relevance and project diversity (not resumed to a single project with several versions).

Regarding the future work one of the aspects that needs to be covered is extending the selection of metrics and also the size of the sample data. Also the method of predicting reliability can be further improved by taking in consideration the thresholds of the metrics and weighting more the values of metrics that are outside of the thresholds. The approach can be extended to other quality factors and this can be the subject of a different research, for example to validate and improve the approach to assess the maintainability, using several object oriented metrics as proposed in [Motogna et al. 2016].

REFERENCES

F.B. Abreu. 1993. Metrics for Object Oriented Environment. In *Proceedings of the 3rd International Conference on Software Quality, Tahoe, Nevada, EUA, October 4th - 6th*. 67–75.

F.B. Abreu. 1995. The MOOD Metrics Set. In *9th European Conference on Object-Oriented Programming (ECOOP'95) Workshop Metrics*.

F.B. Abreu and Carapuca Rogerio. 1994. Candidate Metrics for Object- Oriented Software within a Taxonomy Framework. In *Journal of systems software 26*. 359–368.

Victor R. Basili, Lionel C. Briand, and Walcélio L. Melo. 1996. A Validation of Object-Oriented Design Metrics As Quality Indicators. *IEEE Trans. Softw. Eng.* 22, 10 (1996), 751–761.

Apache Camel. 2018. Apache Camel. https://github.com/apache/camel. (2018). Online; accessed 1 March 2018.

Apache Cassandra. 2018. Apache Cassandra. https://github.com/apache/cassandra. (2018). Online; accessed 1 March 2018.

S. R. Chidamber and C. F. Kemerer. 1994. A Metrics Suite for Object-Oriented Design. *IEEE Trans. Soft Ware Eng.* 20, 6 (1994), 476–493.

R Correlations. 2018. R Correlations. https://www.statmethods.net/stats/correlations.html. (2018). Online; accessed 1 March 2018.

Elasticsearch. 2018. Elasticsearch - Open Source, Distributed, RESTful Search Engine. https://github.com/elastic/elasticsearch. (2018). Online; accessed 1 March 2018.

Nina S Godbole. 2004. *Software quality assurance: Principles and practice*. Alpha Science Int'l Ltd.

Apache Groovy. 2018. Apache Groovy. https://github.com/apache/groovy. (2018). Online; accessed 1 March 2018.

Guava. 2018. Guava: Google Core Libraries for Java. https://github.com/google/guava/. (2018). Online; accessed 1 March 2018.

ISO25010. 2011. ISO25010 description information. http://iso25000.com/index.php/en/iso-25000-standards/iso-25010. (2011). Online; accessed 1 March 2018.

Ronald Jabangwe, Jürgen Börstler, Darja Smite, and Claes Wohlin. 2015. Empirical Evidence on the Link Between Object-oriented Measures and External Quality Attributes: A Systematic Literature Review. *Empirical Softw. Engg.* 20, 3 (2015), 640–693.

JHawk. 2016. http://www.virtualmachinery.com/jhawkprod.htm. (2016). Online; 23- Jan- 2016.

Apache JMeter. 2018. Apache JMeter. https://github.com/apache/jmeter. (2018). Online; accessed 1 March 2018.

J.Quah and Mie Mier Thet Thwin. 2002. Prediction of Software Readiness Using Neural Network. In *ICITA2002*.

W. Li and S. Henry. 1993. Maintenance Metrics for the Object Oriented Paradigm. *IEEE Proc. First International Software Metrics Symp* (1993), 52–60.

Rudiger Lincke and Welf Lowe. 2006. Foundations for Defining Software Metrics. (2006).

R. Lincke and W. Lowe. 2013. Compendium of Software Quality Standards and Metrics. http://www.arisa.se/compendium/quality-metrics-compendium.html. (2013). Online; accessed 1 March 2018.

Log4j. 2018. Apache log4j. https://github.com/apache/log4j. (2018). Online; accessed 1 March 2018.

R. Marinescu. 2002. *Measurement and Quality in Object Oriented Design*. Ph.D. Dissertation. Faculty of Automatics and Computer Science, University of Timisoara.

J.A. McCall, P.K. Richards, and G.F. Walters. 1977. Factors in Software Quality. *Griffiths Air Force Base, N.Y. Rome Air Development Center Air Force Systems Command* (1977).

Metrics. 2016. Metrics.sourceforge.net, "Metrics 1.3.6". http://metrics.sourceforge.net.. (2016).

MetricsReloaded. 2017. MetricsReloaded - Automated code metrics plugin for IntelliJ IDEA. https://github.com/BasLeijdekkers/MetricsReloaded. (2017). Online; accessed 1 March 2018.

S. Motogna, A. Vescan, C. Serban, and P. Tirban. 2016. An approach to assess maintainability change. In *2016 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*. 1–6. DOI:http://dx.doi.org/10.1109/AQTR.2016.7501279

Ndepend. 2016. http://www.ndepend.com. (2016).

S. S. Rathore and A. Gupta. 2012. Validating the Effectiveness of Object-Oriented Metrics over Multiple Releases for Predicting Fault Proneness. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*.

J. P. Royston. 1982. An Extension of Shapiro and Wilk's W Test for Normality to Large Samples. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 31, 2 (1982), 115–124. http://www.jstor.org/stable/2347973

SourceMeter. 2018. www.sourcemeter.com, "SourceMeter 8.2.0". https://www.sourcemeter.com/resources/java/.. (2018).

M. A. Stephens. 1974. EDF Statistics for Goodness of Fit and Some Comparisons. *J. Amer. Statist. Assoc.* 69, 347 (1974), 730–737. DOI:http://dx.doi.org/10.1080/01621459.1974.10480196