STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXV**, Number 1, 2020
DOI: 10.24193/subbi.2020.1.01

EMPIRICAL VALIDATION OF OO METRICS IMPACT ON CHANGEABILITY

DANA CIUBĂNCAN AND PAUL ŢIRBAN

Abstract. In the context of current software development, where changes to an application are made from one sprint to another, it's more and more necessary for developers to be able to easily change the existing code. Starting from the existing literature and using two commercial projects with multiple versions and different architecture we are trying to confirm a correlation between a possible changeability indicator and variations of OO metrics. In our research we managed to provide empirical evidence of the relation between certain OO metrics that can be computed for a system and the changeability quality attribute.

1. Introduction

Nowadays in a software industry, where agile methodologies and fast increments of the software are written, the changes of the existing code are more and more frequent and time consuming. The challenge that developers face more and more is how to change easier and faster existing code of the projects. The challenge for the project managers is to estimate and, if possible, reduce the cost of system changes.

Changeability, sometimes referred as modifiability, had been considered as a software quality characteristics or subcharacteristics in almost all software quality models, proving the importance and impact it has on the overall quality of a software system. Several such models (as Boehm [5], ISO 9126 [3], and the current standard ISO 25010 [12]), consider changeability as a subcharacteristic of maintainability. However, our investigation has identified that despite its importance, these models do not offer ways to improve this factor or numerical values, respectively ranges, to evaluate changeability.

Received by the editors: 13 November 2019.

2010 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.8. **Software Engineering**: Metric — *Product Metrics*.

Key words and phrases. Software Quality, Changeability, Object Oriented Metrics.

Many studies [8, 9, 7, 15] are focusing on the topic of changeability that would help developers work with existing code easier and further develop it without making it more error prone.

The purpose of this work is to study the evolution of Changeability in the context of large software systems, that have been developed by teams (several developers), over a significant period of time, through the relation to software metrics. This objective has been split into the following research objectives:

- *RO1*: provide empirical evidence of the relation between software metrics and changeability;
- *RO2*: examine if the software architecture has an impact on changeability.
- *RO3*: investigate the relation between metrics, changeability over the development period (namely, analyze several versions of the application)

The rest of the paper is organized as follows: the next section details changeability as a software quality factor and the software metrics used in the study, and presents similar research approaches. Section 3 describes the methodology used, the case study and analyze the obtained results, while the last section states some conclusions and suggests future investigations.

2. Preliminaries and Related Work

2.1. **Changeability as Software Quality Factor.** Changeability, also referred to as modifiability in some contexts, has been given a lot of definitions, amongst which are also the following:

- "how well software can be changed" [18]
- "the cost of change and refers to the ease with which a software system can accommodate changes"[20]
- "the ease with which it can be modified to changes in the environment, requirements or functional specification"[9]

Different quality models specify changeability as part of their classification. Both Boehm [5] and ISO 9126 [3] quality models mention changeability as part of maintainability, while in the FURPS and McCall [13] models, it is mentioned as related to flexibility. Also, in the redesigned ISO 9126 model, called the SQuaRE's Model [3], the changeability is still part of maintainability, but under the name of modifiabilty.

Considering the classifications from the quality models, when talking about changeability there are two aspects to consider: *maintainability* and *flexibility*. In a lot of cases, the definitions of these two quality attributes are mainly about what encompasses changeability. When we are thinking about changes that impact the maintainability of a system, we need to think about changes that

correct existing faults, expand or implement a functionality or refactorization, done to improve the overall quality of the system. Whereas, when we think about flexibility, we should have in mind the changes that are done to the system as a full or to separate components in order to make them available for use in other applications and environments in which they were not originally designed.

In our research we focus on changes that have repercussions on the maintainability of a system, focusing on functional changes and code readability, reliability and understandability improvements.

2.2. **Object Oriented Metrics.** Software metrics have become useful in software engineering, playing an important role in understanding, controlling and improving the code, but also the development process. Earlier metrics include LOC (lines of code), lines of comments, or number of functions or modules. With the introduction of object oriented (OO) metrics, the usefulness of these metrics has significantly increased in project management, software quality or refactoring. Several metrics have been proposed [10, 11], and have been classified according to the characteristics of object oriented paradigm, namely coupling, inheritance, cohesion and structural complexity [19]. We selected the metrics such that each of this characteristic is captured.

- *LCOM5*: defined by Henderson-Sellers in 1996 [11], it computes the lack of cohesion of a class and decides into how many classes that class should be split (this metrics represents the latest update of the initially defined LCOM metric)
- *WMC*: part of the Chidamber & Kemerer metric suite [10], it represents the sum of the McCabe Cyclomatic Complexity values of the local methods and initialization blocks
- *CBO*: also defined in the Chidamber & Kemerer metric suite, it computed the number of classes that are directly used by a class
- *RFC*: part of the CK metric suite, computes the number of methods of a class to each is added the number of methods called from within them
- *DIT*: represents the length of the longest path from a class to a root class in the inheritance tree of that system; defined, as well, in the CK metric suite
- *NOC*: defined by Chidamber & Kermerer, it is opposite to DIT, it computes the number of subordinated classes a class has, by going down the class hierarchy tree
- *LOC*: as the name states, it computes the number of lines of code from a specific scope; in our research the metric will always have the *class* scope

- *SIZE2*: gives information about the size of a class, by simply counting the number of attributes and methods that are part of that class

The last two metrics are capturing the size of the software system and the size of classes, and included them in our study besides well known OO metrics.

2.3. **Related work.** From the existing literature we have found that there are some factors that can be used to determine the level of changeability of a system, as follows: the complexity of components, its size and the scope of the changes [8]. The more complex a component is the harder will be for changes to be made to it and the same applies for the size of components. The scope of the changes will decide if the change will be contained and it will only impact a couple of components or if it will have a larger impact across the full application or even other applications.

Other studies concluded that the architecture of a system is a major influencer of its changeability [9]. When designing the architecture of a system it is easier to incorporate changes that have been thought of, changes that have been foreseen than it is to incorporate unpredictable changes. Their approach is based on five steps (selecting a goal, describe the software architecture, elicitation of possible scenarios, evaluation of the scenarios, interpretation) in order to analyze software systems architectures for changeability. The result can be used to predict future maintenance cost and identify possible inflexibilities of the system, as well as giving the possibility of comparing different architectures.

The importance of changebility can also be argued by the number of other software quality attributes or activities that are influenced [8]. Extensibility, refactorization, scalability and functional flexibility are a few of the impacted concerns; all of them imply changes to the system. What does it mean that it has an impact on these concerns? It means that having a system with an increased changeability will positively impact the above-mentioned concerns or, respectively, negatively impact them if we have a decreased changeability. Being able to easily modify a system we will also be able to more easily refactor it, work on its understandability, and also reduce the effort needed for a developer to make changes. We will be able to extend its functionality as well with a reduced effort and cost in order to remain relevant and we will be able to more easily scale the system in order to meet the needs of nowadays' market.

There have been also some researches on the topic of changeability, trying to fill the gap by providing techniques and tactics for achieving changeability [7] or by analyzing different architectures in order to determine if changeability is achieved more by one or another [9].

Researchers from the Montreal University, Canada [15] tried connecting changeability to a more measurable concept: metrics. While trying to prove cohesion as a reliable indicator of changeability, they have investigated the connection between the design of the software system, its architecture, and changeability. To do this they have developed a change impact model. They identified thirteen types of changes that could happen on an object-oriented system and the types of links that connect classes with one another, then computed the impact of a change based on the change's types and the links between the classes that are involved in the change, that are an important factor on how easy will be for that change to happen. The impact is defined as the set of classes that require modifications as the result of the change [15]. The research has concluded that cohesion is not an indicator for changeability, by using the impact of change, but the model they have defined could be furtherly used to assess connection between the quality attribute and other software system quality concepts. An important conlcusion of this study is that cohesion by itself, computed using the LCC (Loose Class Cohesion) and LCOM (Lack of Cohesion of Methods), is not a reliable indicator for changeability [15].

A very comprehensive survey of the relations between ISO9126 quality categories [3] and OO metrics is presented in [17], in the form of a compendium in which relations for all quality characteristics and subcharacteristics with software metrics are stated and graded "related" or "highly related" . We should remark that all the metrics we have selected for our study are considered highly related to changeability.

3. Estimating Changeability changes using Object Oriented Metrics

3.1. **Approach.** The purpose of the initial study was to determine how can OO metrics for different software versions of the same software system provide relevant information for changeability, and if the collected data can be used to improve changeability of the system.

The chosen metrics reflect on a systems complexity and size, cohesion and coupling, factors that were studied before in relation with changeability. Our initial study has computed the metrics for all versions of the considered study, and our first finding was that some metrics display little variation between versions, and in terms of software system evolution they cannot provide useful values. We decided to eliminate them from our study, such that the final list of metrics contained only LCOM5, WMC, CBO, LOC and SIZE2 (values for DIT, RFC and NOC were eliminated)

We used SourceMeter [22], a free tool for computing different metrics that also supports projects written in Java. Due to the fact that different tools

compute metrics in different ways, we considered an important aspect that all metrics are computed with the same tool.

The next step was to determine a way to count the changes performed on a system. Computing this value has two purposes: 1) to find a relation between changes and metrics in different versions of the source code and 2) to use it in order to predict further possible changes. As suggested in similar studies [4], we restricted our count to simple/atomic changes, and took into consideration changes applied to: classes, methods and fields (variables). The type of changes that we considered are: add, delete, change type, change signature. Starting from this, we assessed the possible changes in changeability using the formula:

$$Changeability\_Possible\_Change = DIFF\_LOC/CHANGE\_COUNT$$

where:

$DIFF\_LOC$ = difference of LOC between two versions

$CHANGE\_COUNT$ = number of changes between two versions

After we computed the OO metrics for multiple versions of projects and the Changeability_Possible_Change (ChangeCoef) we tried to see if there exists any correlation that could indicate us what classes we should further investigate. The first step was to run the Shapiro-Wilk test [21], for each of the variables to see if they are normally distributed. Because the Shapiro-Wilk test is limited to a number of maximum 5000 observations, we also tested normal distribution with Anderson-Darling normality test [23]. 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) [1]. Given the fact that our data was not normally distributed, we used Spearman's correlation. The last step was to interpret the results from the correlations.

3.2. **Case Study and Results.** For our case study we chose two implementations with different architectures of the same health care system (considering almost the same functionalities). For privacy reasons we shall call these projects Project A and Project B. Both have been developed as commercial applications, over significant period of time, and in teams of developers with different levels of experience.

Project A was started 7 to 8 years ago and is still being used in production today. The system was built on a Liferay [16] platform; it used Java up to version 7 and jQuery [14] when needed for the UI. As for the architecture, when working with Liferay, a lot of functionality comes bundled in the core Liferay plugin and you can create new plugins for your specific functionality that you need to add than to Liferay. Liferay also generates a lot of code on

behalf of the developer. For example, when creating a new model, a couple of service classes will be generated, wrappers, exception classes and in the end also the model itself. This creates a lot of coupling and redundant code that deeply affects understandability. In our study, we only analyzed the main plugin that holds a big part of the functionality of the application.

The development phase of project B started in June 2018. It uses a microservice architecture that exposes REST APIs for the frontend microservices to call when displaying the UIs. Here the technology stack is up to date, using Java with Spring Boot [2] for the backend microservices and Angular 7 [6] for the frontend. We can say that this project represents a rewriting of project A with newer technologies. Since the project is only one year old, not all the functionality of project A is included in project B and neither is that the goal for project B, but to mainly recreate the main functionalities of project A. For our analysis, the classes from one microservice were taken into consideration, the microservice that contains the most similar functionality with the plugin chosen from project A.

The versions of the system were selected so that they are scattered across the implementation thus far of the systems, having variable time passed between versions, but long enough time so that the versions are not too similar.

Table 1 and Table 2 represents an excerpt of the computed data, and almost all differences between versions gave the same results. The tables show the results of Spearman correlation (*rho* and *p-value*) and the conclusion based on these values for *ChangeCoef* and the selected set of metrics. For both projects we can see that there is a correlation between the number of changes and the differences of the metrics in values.

Some remarks regarding the obtained results can be summarized in:

- the correlation for *ChangeCoef* and *LCOM5* is weaker than the correlations corresponding to the other metrics;
- the correlation for *ChangeCoef* and different metrics can be influenced by the architecture of the system - some correlations are "strong", others are "very strong".

So, regarding our first research objective, we can say that except LCOM, the conclusions stated by [17] are confirmed by our data. In case of LCOM: [17] considered that changeability and LCOM are highly related, while [15] concluded that cohesion (given by LCOM metric) is not an good indicator for changeability. Due to the variations in our results, and obtaining only "moderate" correlation for some cases, the conclusion will be to use LCOM or LCOM5 with care and together with other metrics when analyzing changeability of the system.

TABLE 1. Project A v1-3-3 and v1-3-14

	rho	p-value	correlation interpretation
ChangeCoef-LOC	0.7748249	< 2.2e-16	Strong
ChangeCoef-SIZE2	0.6923647	< 2.2e-16	Strong
ChangeCoef-WMC	0.8149562	< 2.2e-16	Very strong
ChangeCoef-CBO	0.6357793	< 2.2e-16	Strong
ChangeCoef-LCOM5	0.5867332	< 2.2e-16	Moderate

TABLE 2. Project B v1 and v2

	rho	p-value	correlation interpretation
ChangeCoef-LOC	0.9761374	< 2.2e-16	Very strong
ChangeCoef-SIZE2	0.8960441	< 2.2e-16	Very strong
ChangeCoef-WMC	0.8087428	< 2.2e-16	Very strong
ChangeCoef-CBO	0.826574	< 2.2e-16	Very strong
ChangeCoef-LCOM5	0.7780307	< 2.2e-16	Strong

Considering the second research objective, our computations showed that the architecture of the system can influence the relation between OO metrics and changeability, and newer technologies, such as microservices expose a higher correlation between OO metrics and changeability. The two tables were chosen as representatives of the performed computations (namely, most of the compared versions produced on average the same results), and we can notice higher values for *rho* in project B, leading to "very strong" correlation.

In order to achieve or third research objective, we developed a tool that based on the analysis described above can identify the situations in which changeability has recorded a significant increase, that can lead to costly situations in further development of the software system.

Figure 1 shows how metrics can be compared for different versions of the same application, giving the developer an indication where a certain metric has exposed a significant difference. For example, NM (Number of Methods) represents an increase between the two versions, but this is due to adding new functionalities to the application, and as a consequence WMC has changed, but not in a disturbing way.

In order to identify the "hotspots" regarding chageability, the values for each metric are split equally into four quartiles. The diagnostics are done on different level: danger, medium and simple threat. The levels represent how much of a threat do the class pose. The predefined threat levels that can be chosen are:

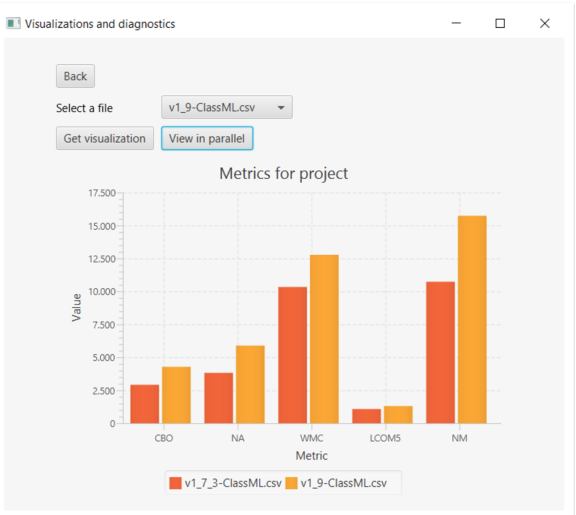

FIGURE 1. Parallel visualization for metrics from 2 versions (NA (Number of Attributes) + NM(Number of Methods) = SIZE2

- *Danger*: classes having all metric values in each metric's forth quartile
- *Medium*: classes having all metric values in each metric's third quartile or above
- *Threat*: classes having all metric values in each metric's second quartile or above.

The classes that are displayed as part of the diagnosis can also be viewed as the classes that need adjustments if you are concerned with the threat level that you have chosen. This means that if you chose the medium threat level, as shown in Figure 2, you receive both the classes having metric values in the third quartile but also the ones having all the metric values in the fourth quartile since those are the ones that are more important to be changed.

The tool can be a useful assistant to developers in early detection of situations that can become out of control, especially in large software systems, with a significant number of versions, developed by big teams over large period of time. However, this tool need to be correlated with a through inspection of the code in order to avoid false results (exemplified in Figure 1).

The biggest threat to validity is given by the small amount of data used in the analysis. Finding real project from the industry was hard and because of this the study should be extended on other projects or on open source projects.

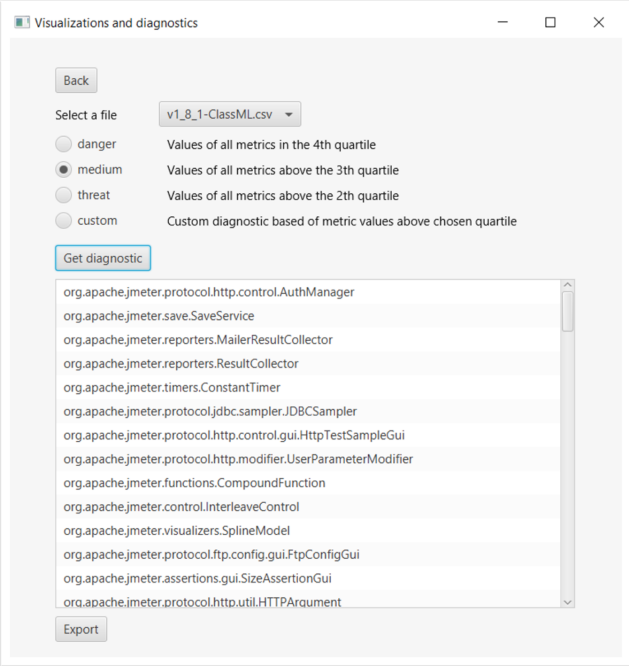

FIGURE 2. Classes exposing a medium threat to changeability

Even if we selected the metrics so that key characteristics of software were covered, a further extension of them is required.

4. Conclusions and future work

Changeability is a quality attribute that has an impact on multiple disciplines that are involved in developing and launching a product. Right now, its improvement has been targeted indirectly by enhancements done on other quality attributes that are more easily assessed and controlled. There is still a lot to be learned about changeability and with this paper we have provided a better understanding of what influences changeability and what is its relation with OO metrics. In case of large software systems, controlling the changeability during the development phase can improve future maintenance and refactoring.

With our research we believe that we are a step closer in providing empirical evidence of the relation between certain OO metrics that can be computed for a system and the changeability quality attribute. We also showed one possibility in which the collected data about metrics can be applied to detect situations

in which changeability expose a significant increase between versions of an application.

The positive results motivate us in further extending the research by applying the research on more projects but also to add additional OO metrics. Also another direction that we will try to explore is to take in consideration also the effort needed to make a change.

References

[1] R correlations. https://www.statmethods.net/stats/correlations.html, 2019. Online; accessed 10 June 2019.

[2] Spring boot. https://spring.io/projects/spring-boot, 2020. Online; accessed 10 January 2020.

[3] 9126-1:2001, I. Software engineering - product quality. http://www.iso.org, 2019. Online; accessed 10 June 2019.

[4] Ajrnal Chaumun, M., Kabaili, H., Keller, R. K., Lustman, F., and Saint-Denis, G. Design properties and object-oriented software changeability. In *Proceedings of the Fourth European Conference on Software Maintenance and Reengineering* (2000), pp. 45–54.

[5] Al-Qutaish. R. Quality Models in Software Engineering Litrature: An Analytical and Comparative Study. *Journal of American Science 6*, 3 (2010), 166–175.

[6] Angular. Angular. https://angular.io, 2020. Online; accessed 10 January 2020.

[7] Bachmann, F., Bass, L., and Nord, R. Modifiability tactics. 63.

[8] Barbacci, M. R. Software quality attributes: Modifiability and usability. http://www.ieee.org.ar/downloads/Barbacci-05-notas1.pdf, 2003.

[9] Bengtsson, P., Lassing, N., Bosch, J., and Vliet, H. Analyzing software architectures for modifiability. https://www.researchgate.net/ publication/30499164_Analyzing_Software_Architectures_for_Modifiability, 2009.

[10] Chidamber, S., and Kemerer, C. A Metric Suite for Object- Oriented Design. *IEEE Transactions on Software Engineering 20*, 6 (1994), 476–493.

[11] Henderson-Sellers, B. Software metrics. *Hernel Hempstead: Prentice Hall* (1996).

[12] ISO25010. Iso25010 description information. http://iso25000.com/index.php/en/iso-25000-standards/iso-25010, 2019. Online; accessed 10 June 2019.

[13] J. McCall, P. R., and Walters, G. Factors in software quality. *Nat Tech.Information Service 1* (1977).

[14] jQuery. jquery. https://jquery.com/, 2020. Online; accessed January 2020.

[15] Kabaili, H., Keller, R. K., and Lustman, F. Cohesion as changeability indicator in object-oriented systems. In *Proceedings Fifth European Conference on Software Maintenance and Reengineering* (March 2001), pp. 39–46.

[16] Liferay. Digital experience software tailored to your needs. https://www.liferay.com/, 2020. Online; accessed 10 January 2020.

[17] Lincke, R., and Lowe, W. Foundations for defining software metrics, 2006.

[18] Lincke, R., and Lowe, W. Compendium of software quality standards and metrics. http://www.arisa.se/compendium/quality-metrics-compendium.html, 2019. Online; accessed 10 June 2019.

[19] Marinescu, R. *Measurement and Quality in Object Oriented Design.* PhD thesis, Faculty of Automatics and Computer Science, University of Timisoara, 2002.

[20] Northrop, L. Achieving product qualities through software architecture practices. https://apps.dtic.mil/dtic/tr/fulltext/u2/a631504.pdf, 2004.
[21] Royston, J. P. 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.
[22] SourceMeter. www.sourcemeter.com, "sourcemeter 8.2.0". https://www.sourcemeter.com/resources/java, 2019.
[23] Stephens, M. A. Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association 69*, 347 (1974), 730–737.

Babeş-Bolyai University, Department of Computer Science, 1 M. Kogălniceanu Street, 400084 Cluj-Napoca, Romania
Email address: cdis1553@scs.ubbcluj.ro
Email address: tirban@cs.ubbcluj.ro