An Empirical Analysis of the Maintainability Evolution of Open Source Systems

Gerta Kapllani, Ilya Khomyakov, Ruzilya Mirgalimova, and Alberto Sillitti(✉)

Innopolis University, Innopolis, Russian Federation
g.kapllani@innopolis.university,
{i.khomyakov,r.mirgalimova,a.sillitti}@innopolis.ru

Abstract. Maintainability is a key factor for the evolution of an open source system due to the highly distributed development teams that contribute to many projects. In the literature there are a number of different approaches that has been developed to evaluate the maintainability of a product but almost each method has been developed in an independent way without leveraging on the existing work and with almost no independent evaluation of the performance of the models. In most of the cases, the models are only validated through a limited set of projects only by the people that propose the specific approach. This paper is a first step towards a different direction focusing on the independent application of the existing models to popular open source projects.

Keywords: Maintainability · Empirical software engineering · Software evolution

1 Introduction

Maintainability is one of the most important features of software systems [10]. Though, defining maintainability has been a topic of interest for many researchers over decades. A software application might fulfils all the traditional requirements and yet be of little use if the cost of maintaining it is too high. For such reason, maintainability has been considered one of the requirements that must be imposed to software products [3]. Several authors have investigated the models and metrics used for the best estimation of maintainability also known as prediction or software maintainability prediction models. Traditional models based on their accuracy of prediction have been represented by Shafiabady et al. [22]. Moreover, an analysis of the evolution of software maintenance models over the four past decades has been introduced by Lenarduzzi et al. [14]. Defining the right tools on code analysis to perform research studies on software maintenance prediction is also a crucial aspect and has received much attention in the last few years [18]. Recent papers [2,5,11,15,19] have identified tools that are available and can be used to apply maintenance models published in the studies in this area.

Published by Springer Nature Switzerland AG 2020
V. Ivanov et al. (Eds.): OSS 2020, IFIP AICT 582, pp. 78–86, 2020.
https://doi.org/10.1007/978-3-030-47240-5_8

The paper is organized as follows: Sect. 2 briefly analyzes the state of the art; Sect. 3 introduces our investigation; Sects. 4 and 5 present and discuss the results; finally, Sect. 6 draws the conclusions and introduces future work.

2 Related Research

During the years, several models have been proposed for measuring the maintainability of software. A considerable amount of studies have employed different models and techniques for predicting software maintenance using basic metrics and models: McCall's model in 1976 [20], Barry Boehm's quality model presented in 1978 [1], Sneed-Mercy Model in 1985 [23], Li-Henry Model in 1993 [16], Marcela Genero Model in 2004 [8]. Later, slightly different techniques were used from simple statistical models such as regression [25] to machine learning [7,24], and deep learning [9,13]. To predict maintainability of software, different metrics have been proposed in literature. Among the large variety of metrics, the most used ones are the OO metrics depth of the inheritance tree (DIT), response for a class (RFC), number of children (NOC), coupling between objects (CBO), lack of cohesion of methods (LCOM), and weighted method per class (WMC) [4]. Other metrics are popular, such as: number of methods (NOM), lines of code (LOC), number of semicolons in a class (SIZE1), number of properties (SIZE2), and CHANGE metrics as dependent variable to predict software maintainability by calculating the changed number of lines in the class during maintenance process [16]. Investigating different primary studies and secondary studies from the literature, we have found out that even if there are a number of different methods to predict maintainability, almost all the studies built their models from scratch and do not extend existing ones.

3 Our Investigation

We decided to analyse software maintainability change following the approach proposed in [17] for the jEdit open source project using the JHawk tool to get useful insights. jEdit is an open source project developed in Java available in SourceForge[1] with all existing versions with the related source code. We analyzed the application starting from version 3.2 up to version 5.5, a total of 12 versions. There are in total 41 versions but they have almost no differences compared to the selected ones. A significant difference compared to [17] is that we rely on general information outside the source code to examine and find potential relation between version numbers and changes to maintainability during the evolution of the project.

JHawk is Java-based open source framework which can analyse source code while performing static analysis generating graphical form results. It takes Java code as input and generates code metrics. We are interested in gathering features such as average cyclomatic complexity (CC), number of lines of code (LOC), and maintainability index (MI).

[1] https://sourceforge.net/projects/jedit/files/jedit/.

3.1 Collected Metrics

In this study we have focused on the following metrics:

- **Cyclomatic Complexity (CC):** it is used to measure the complexity of a piece of code from the point of view of the possible execution paths. In particular, it measures the independent path through a source code as a proxy of its complexity. The higher the value, the more complex is the system and this fact results in difficulty to maintain the code. We have measured the average of CC which is stated by McCabe to be cyclomatic complexity per function in a file and is calculated as the sum of the CC of all function definitions, divided by the number of function definitions in the file.
- **Lines of code (LOC):** it is used to measure the size of a software by counting the number of lines included in a program. There are multiple ways to count the lines of code but in this work we focus on the simplest definition that consider the physical lines of code of the source files.
- **Maintainability index (MI):** it is based on Halstead Volume (HV) metric, the cyclomatic complexity (CC) metric, and the average number of lines per code per module (LOC) metric. The higher this value, the more maintainable the software results [21]. If MI of software increases it means that the software can be maintained easily, otherwise if it decreases the degree of difficulty to maintain the software is high [12]. Values of MI index as follows:
  - MI $<$ 65: difficult to maintain
  - MI 65–85: maintainable
  - MI $>$ 85: good maintainability

We have to highlight the fact that in JHawk MI is reported in two forms, one that takes into account comments of the program and ones which does not. In our database we have reported both of them, but the final result of the graphs below do contain only the ones with comments. It is stated that the reason that MI formula with comments does exist is because the formula to find comments is not standardized and comments are considered to be subjective from developer to developer and do vary from project to project. In such cases it is recommended to use the MI without comments as reported from official source development of the JHawk tool. However, we think that comments are an important part of the source code, therefore we considered them in our study.

4 Results

In Fig. 1 we present the results achieved from data collection of MI for each version of jEdit, as we have mentioned in the above sections.

We can notice that from version 3.2 to 4.2 there is a constant value of MI of about 130. Upon release of version 4.0, there is a tremendous decrease of MI to about 58. Then, with next releases up to version 5.3.0, there seem to be a slight increase about 68. The last versions have a value of MI up to 144.

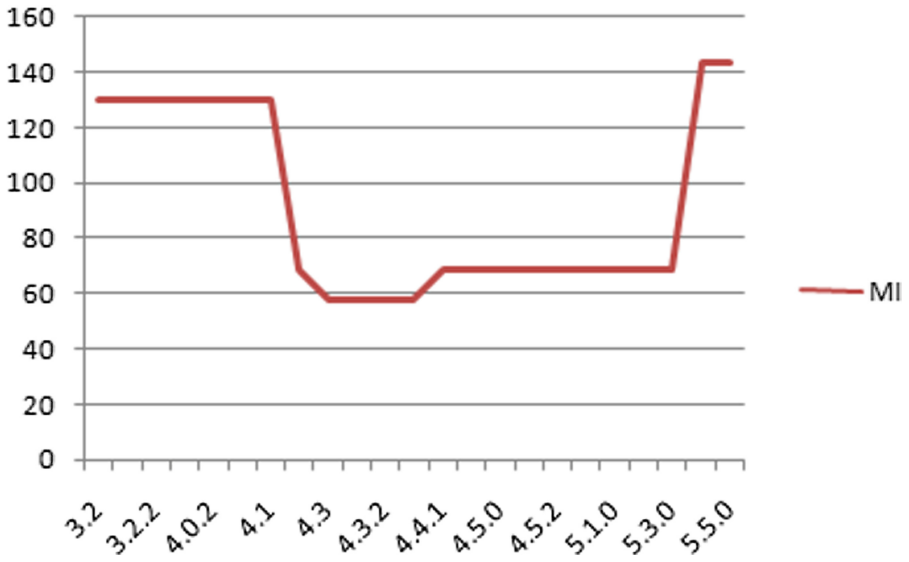

Fig. 1. MI evolution.

We also have to consider the other two features collected (Fig. 2, 3 and 4). It is of high relevance to mention that the value of the average CC has been higher for versions of low MI index in other words for early releases. A slightly different trend follows the LOC metric. For higher values of LOC there is also a higher value for MI and vice versa. We can notice the trend of size metric for each version. For an increase of size from versions 4.3 to 4.4.1, there is a decrease in MI in these versions. For a decrease of size in versions 4.5 to 5.3, there is a slight increase of MI. Surprisingly, for an increase of size in 5.4.0 and 5.5.0, there is also an increase in MI.

From Fig. 3 and 4, we can state that for an increase in CC from version 4.2 to 4.3, there is a decrease in MI. Moreover, for a decrease in CC from versions 4.3.3 to 4.4.1, there is an increase in MI. The same result follows 5.3.0 to 5.4.0 with a decrease in CC and an increase in MI. These outputs clearly highlight the fact that a high value of CC means more complexity added to the system which increases the degree of difficulty for the system to be maintained.

5 Discussion

In this section we are going to investigate further the results obtained from previous section. We are going to test hypothesis of possible correlation of LOC, CC with MI and LOC with CC. Since the MI is defined as a function that include LOC and CC as independent variables, we expect that MI is highly correlated to them but we want to investigate how strong is the effect of such variables on the overall values of the MI.

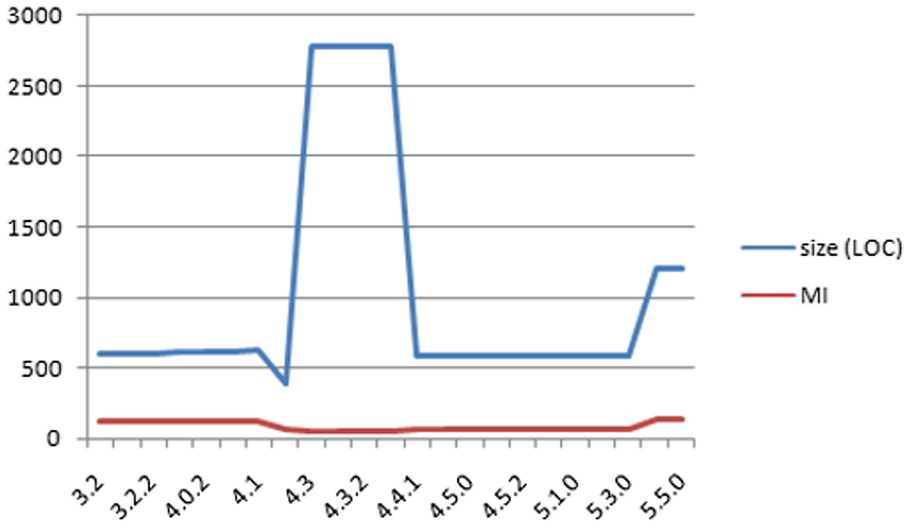

Fig. 2. LOC and MI evolutions.

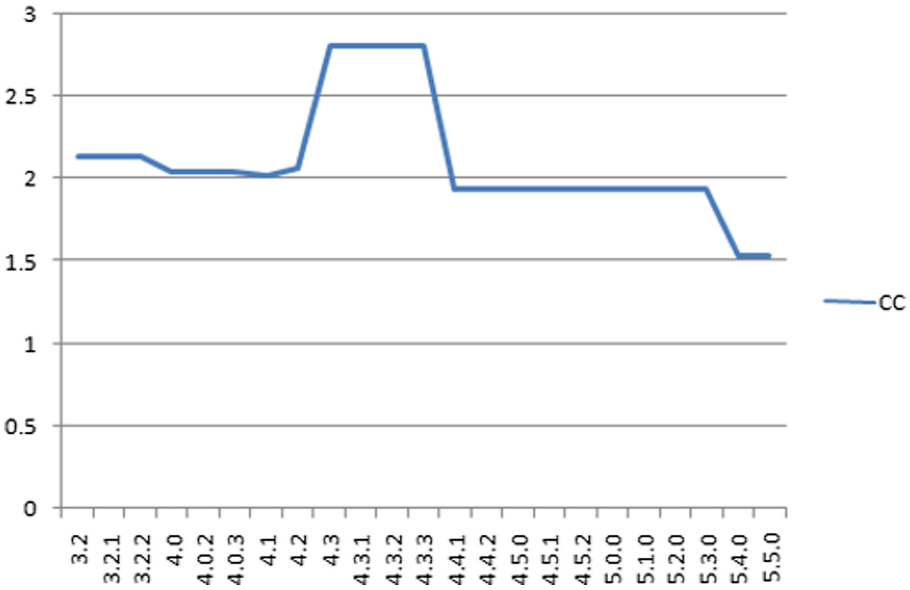

Fig. 3. CC evolution.

We use the same data provided in the initial steps of our work collected with the JHawk tool. To test correlation, we are postulating hypothesis in Table 1. In overall, correlation claims the strength of a relationship. We use the Cohen definition of correlation [6] where he defined a value less than 0.3 as a weak

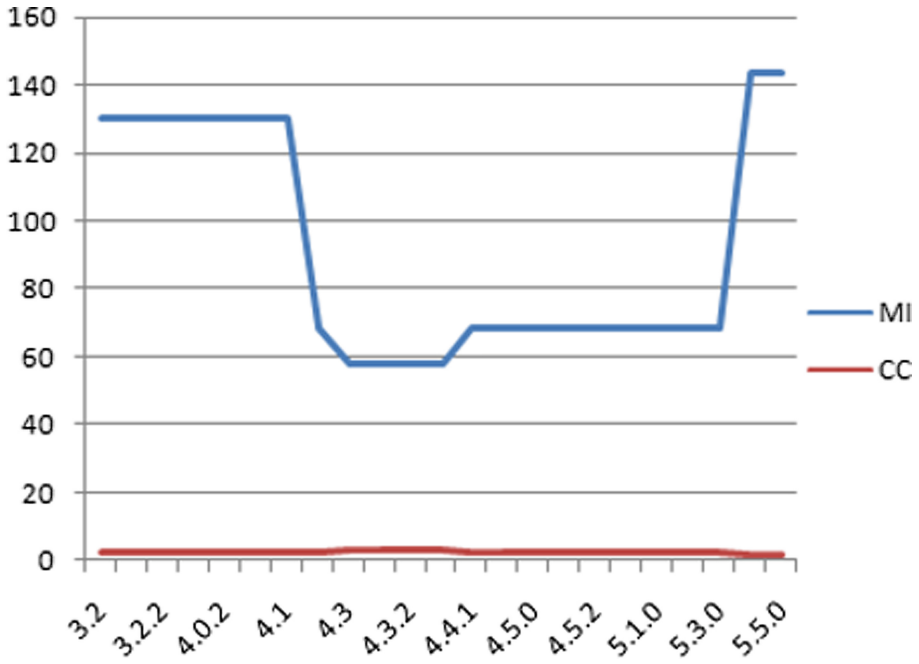

Fig. 4. CC and MI evolutions.

correlation, 0.3 to 0.5 as medium correlation, and greater than 0.5 as strong correlation. We are interested to understand the strength of such correlation. To measure the strength of relationship in our data variables we use Spearman rank correlation for the reason that it does not assume any linear relation between variables and it does not assume any specific distribution of data. We analyze the data in Python, using a Jupyter Notebook with the stats.py statistical package.

Table 1. Hypotheses.

No.	Hypotheses
1	**Null:** LOC and MI are unrelated **Ha:** LOC is positively correlated with MI
2	**Null:** CC and MI are unrelated **Ha:** CC is positively correlated with MI
3	**Null:** LOC and CC are unrelated **Ha:** LOC is positively correlated with CC

For this project, Spearman correlation coefficients are:

1. 0.129 for LOC and MI with p-value 0.522 which aims result is not significant, so we can not reject hypothesis null.
2. −0.098 with a p-value 0.628, again not significant so we can not reject hypothesis null.
3. 0.513 with p-value 0.006 is significant therefore we can reject hypothesis null and conclude that a relationship exists.

We have performed the same kind of analysis over other 4 open source projects:

- SoundHelix (0.6-0.9 versions): a Java framework for algorithmic random music composition. It can generate random songs, play them, and it is highly customizable using XML configuration as described in SourceForge.net.
- jWebUnit (1.0-3.2): a Java framework for testing web applications.
- jXLS (0.7-2.7.1): a Java library for writing Excel files using XLS templates and reading data from Excel into Java objects using XML configuration.
- jTDS (0.1-1.3.1): an open source driver for Microsoft SQL Server

We went again through same process described for the analysis of jEdit to collect LOC, MI, and CC metrics, and perform the correlation analysis as reported in Table 2.

Table 2. Correlations in all the considered projects.

Project	LOC-MI (pval)	CC-MI (pval)	LOC-CC (pval)
jEdit	0.129 (0.522)	−0.098 (0.628)	**0.513 (0.006)**
SoundHelix	−0.816 (0.184)	0.333 (0.667)	−0.816 (0.184)
JTDS	**0.733 (0.000)**	**−0.578 (0.000)**	−0.249 (0.161)
JXLS	**0.695 (0.000)**	**−0.812 (0.000)**	**−0.567 (0.000)**
JWebUnit	−0.320 (0.226)	0.200 (0.457)	**−0.789 (0.00)**

As we can notice in Table 2, the values in red identify significant values. All projects include some significant values except SoundHelix. There is a strong correlation of the LOC metric and MI in JXLS and JTDS and a strong correlation of CC and LOC in jEdit project. Also to be highlighted is the fact that exist strong negative correlations. The not significant results obtained from the SoundHelix project might derive from the fact that it is the project with the least data collected, this could have affected the overall analysis since only 4 versions were analyzed.

6 Conclusions and Future Work

The paper has introduced a preliminary analysis of the maintainability of 5 popular open source projects and how strongly it is correlated with some basic code metrics (lines of code and cyclomatic complexity) that are used in the definition of the Maintainability Index.

A systematic analysis of the maintainability of popular open source projects to understand how the code is managed and how the development team address the evolution of the code is important to increase the level of awareness in the usage of such products and to understand the possible effects in the long term.

References

1. Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., McLeod, G., Merritt, M.: Characteristics of Software Quality. North Holland, Amsterdam (1978)
2. Bordeleau, F., Meirelles, P., Sillitti, A.: Fifteen years of open source software evolution. In: 15th International Conference on Open Source Systems (OSS 2019), Montreal, Quebec, Canada, 26–27 May 2019
3. Cheaito, R., Frappier, M., Matwin, S., Mili, A., Crabtree, D.: Defining and measuring maintainability. Technical report, University of Ottawa, Department of Computer Science (1995)
4. Chidamber, S.R., Kemerer, C.F.: Towards a Metrics Suite for Object Oriented Design. Center for Information Systems Research, Slan School of Management, Cambridge (1991)
5. Ciancarini, P., Missiroli, M., Sillitti, A.: Preferred tools for agile development: a sociocultural perspective? In: Technology of Object-Oriented Languages and Systems (TOOLS 50+1), Innopolis, Tatarstan, Russian Federation, 14–19 October 2019
6. Cohen, J.: Statistical Power Analysis for the Behavioral Sciences. Routledge, Abingdon (1988)
7. Elish, M.O.: Application of treenet in predicting object-oriented software maintainability: a comparative study. In: 2009 13th European Conference on Software Maintenance and Reengineering, pp. 69–78. IEEE (2009)
8. Genero, M., Piattini, M., Manso, E., Cantone G.: Building UML Class Diagram Maintainability Prediction Models Based on Early Metrics. IEEE (2004)
9. Jha, S., et al.: Deep learning approach for software maintainability metrics prediction. IEEE Access **7**, 61840–61855 (2019)
10. Kan, S.H.: Metrics and Models in Software Quality Engineering. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
11. Khomyakov, I., Sillitti, A.: A novel approach for collecting and sharing software metrics data. In: 34th ACM Symposium on Applied Computing (SAC 2019), Limassol, Cyprus, 8–12 April 2019
12. Khondhu, J., Capiluppi, A., Stol, K.-J.: Is it all lost? A study of inactive open source projects. In: Petrinja, E., Succi, G., El Ioini, N., Sillitti, A. (eds.) OSS 2013. IAICT, vol. 404, pp. 61–79. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38928-3_5
13. Kumar, L., Rath, S.K.: Hybrid functional link artificial neural network approach for predicting maintainability of object-oriented software. J. Syst. Softw. **121**, 170–190 (2016)

14. Lenarduzzi, V., Sillitti, A., Taibi, D.: Analyzing forty years of software maintenance models. In: 39th International Conference on Software Engineering (ICSE 2017), Buenos Aires, Argentina, 20–28 May 2017
15. Lenarduzzi, V., Sillitti, A., Taibi, D.: A survey on code analysis tools for software maintenance prediction. In: Ciancarini, P., Mazzara, M., Messina, A., Sillitti, A., Succi, G. (eds.) SEDA 2018. AISC, vol. 925, pp. 165–175. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-14687-0_15
16. Li, W., Henry, S.: Object-oriented metrics that predict maintainability. J. Syst. Softw. **23**, 111–122 (1993)
17. Molnar, A., Motogna, S.: Discovering maintainability changes in large software systems. In: IWSM Mensura 17: Proceedings of the 27th International Workshop on Software Measurement (2017)
18. Moser, R., Pedrycz, W., Sillitti, A., Succi, G.: A model to identify refactoring effort during maintenance by mining source code repositories. In: Jedlitschka, A., Salo, O. (eds.) PROFES 2008. LNCS, vol. 5089, pp. 360–370. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69566-0_29
19. Petrinja, E., Sillitti, A., Succi, G.: Comparing OpenBRR, QSOS, and OMM assessment models. In: Ågerfalk, P., Boldyreff, C., González-Barahona, J.M., Madey, G.R., Noll, J. (eds.) OSS 2010. IAICT, vol. 319, pp. 224–238. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13244-5_18
20. Rawashdeh, A., Matalkah, B.: A new software quality model for evaluating COTS components. J. Comput. Sci. **2**, 373–381 (2006)
21. Scotto, M., Sillitti, A., Succi, G., Vernazza, T.: Dealing with software metrics collection and analysis: a relational approach. Stud. Inform. Univ. Suger **3**(3), 343–366 (2004)
22. Shafiabady, A., Mahrin, M.N., Samadi, M.: Investigation of software maintainability prediction models. In: 18th International Conference on Advanced Communication Technology (ICACT) (2016)
23. Sneed, H.M., Merey, A.: Automated software quality assurance. IEEE Trans. Softw. Eng. **SE–11**(9), 909–916 (1985)
24. Van Koten, C., Gray, A.: An application of Bayesian network for predicting object-oriented software maintainability. Inf. Softw. Technol. **48**, 59–67 (2006)
25. Zhou, Y., Leung, H.: Predicting object-oriented software maintainability using multivariate adaptive regression splines. J. Syst. Softw. **80**, 1349–1361 (2007)