Volume 8, No. 3, March – April 2017
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

A Metrics Based Framework to Improve Maintainability of Reusable Software Components through Versioning

Anshul Kalia
Ph.D. Scholar, Dept. of RIC,
IKG-Punjab Technical University,
Kapurthala, India

Sumesh Sood
HOD, Dept. of Computer Applications,
IKG-Punjab Technical University Campus Dinanagar,
Dinanagar, India

***Abstract*:** An improvement in maintainability of software components has the potential to affect the maintenance practices. The maintainers are more likely to follow the maintenance process that improves the maintainability of software components. The study presents a method that is intended to improve the maintainability of reusable software components. The study floats the idea of improving maintainability of reusable software components through versioning. It states that on creating a new version of a reusable software component, the maintainability increases. It validates the concept through case study conducted on open source software. The reason for taking open source software as case study is that their probability to be reused is more as compared to other software components. The study makes use of metrics to calculate maintainability and other factors that affect it. The study is also compared with other related studies to exhibit its contribution.

***Keywords*:** Maintainability index, versioning, cyclomatic complexity, metrics.

I. INTRODUCTION

A study has been conducted to determine the effects of versioning on software components. The study has been conducted by using the metrics. Metrics are used to measure the maintainability and cyclomatic complexity of components. The maintainability of software components states their potential to be maintained. It states the ease of maintaining the components [22]. That is, how much effort will be required to maintain the components? If the effort required to maintain the software components is significantly high, then the cost of maintaining components will rise. It can cause budgetary problems for the project such as increasing the cost of maintaining the individual components as well as increasing the overall budget of the project. Now versioning of a software component can be referred to as the improvements, enhancements, upgradation in existing software components [25]. In other words, it can be said that a previous version of a software component is reused while making a new version of that component. While doing so there may be changes in software component. When a new version of a software component is created either a major change is made by adding a new functionality to the component or a minor change is made by making few significant changes in component. The types of changes made to the software components while versioning are reflected by using version numbers [25]. The study proposes a metrics based framework to improve the maintainability of reusable software components through versioning of software components. While proposing the framework the study floats an idea of creating a new version of a reusable software component. The reason is, with the creation of a new version the maintainability of reusable software component increases.

II. RESEARCH OBJECTIVE

The objective of this study is to analyze the maintainability of reusable software components. It tries to determine the factors that affect the maintainability of reusable software components. After determining those factors, it measures the effect of these factors on maintainability. That is, how the maintainability varies in accordance with these factors. Also, the study emphasizes on measuring the effects of versioning on maintainability of reusable software components. While doing such an exercise, the study takes an opportunity to propose a metrics based framework that is intended to improve the maintainability of reusable software components.

III. RESEARCH METHODOLOGY

The methodology that is adopted for conducting the study is to firstly determine the factors that affect the maintainability of reusable software components. It will then identify the metrics that can be used to measure the maintainability and the effect of determined factors on the maintainability. The study will propose a framework that will be intended to improve the maintainability of reusable software components. The proposed framework will be validated through a case study.

IV. FACTORS AFFECTING THE MAINTAINABILITY OF REUSABLE SOFTWARE COMPONENTS

The following factors are considered as the one which are responsible for affecting the maintainability of reusable software components:

A. Size

The factor size affects the maintainability of reusable software components. The more the size of components the more difficult it is to maintain the components. Also it takes more time to understand components large in size. That in turn, increases the effort of maintaining components and finally it has its effects on cost and time to delivery [7] [10].

B. Complexity

The complexity refers to the degree of ease with which the structure and architecture of component is presented [23]. The lesser is the complexity the more will

be maintainability of components. The complexity of a component is affected by the size of component [7]. Complexity is said to be directly proportional to the size of component.

C. *Compliance*

It refers to the compliance of components with the international standards to maintain the components. That is, it has been checked that whether the rules have been violated while developing, and maintaining components or not. If the rules have been violated, then it will have its effects on the maintainability of components [1] [28].

D. *Versioning*

Versioning refers to the improvements, enhancements, upgrading software components, and adding new functionality to the components. It also refers to removing errors, bugs, and problems which are faced in the older versions [25]. By this way of maintaining components, it serves two purposes. One it maintains the software component, second it creates a newer version of components. The newer version of software component should be in compliance with the present technological trends.

Though there exist other factors as well which are responsible for affecting the maintainability of reusable software components [16], but measuring the effect of all those factors on maintainability is a tedious task. So, the study is taking note of some of those factors which are considered as important from the title's view point of study. That is, only those factors are considered that are required to be measured from the title's point of view.

V. METRICS USED TO MEASURE THE MAINTAINABILITY AND RELATED FACTORS

The factors mentioned in section 4 are measured and their effect on maintainability of reusable software components is analyzed with the help of metrics. Different metrics are used to serve this purpose. A list of such metrics is presented in this section of study.

A. *Lines of code*

The lines of code metrics is used to measure the size of a software component by counting the number of lines included in a program. The LOC metrics is also referred to as source lines of code (SLOC). It can be used to assess the effort required to develop and to maintain the program or software component [34].

There are two aspects through which the lines of code metrics measure the line of code in software. One is the physical lines of code and the other is logical lines of code. In physical lines of code, the metrics includes all the source lines of code, i.e. it counts all the code lines but excluding the comment lines. Whereas in logical lines of code, only the executable statements are counted irrespective of the language formats. The logical lines of code metrics are independent of language formats and style conventions. Therefore, the logical lines of code often remain the same [34].

The study makes use of logical lines of code metrics (LLOC) for measuring the size of software component.

B. *Cyclomatic complexity*

The cyclomatic complexity is a software metrics which is used to measure the complexity of software quantitatively. This metrics measures the independent path through a source code to calculate the complexity [17] [23]. The cyclomatic complexity can be calculated for functions, classes, methods, modules of software [4]. This metric produces a number indicating the complexity of software. The more the value of number, the more is the complexity. The more complexity of software means that it is difficult to maintain [4] [17]. The mathematical formula used to calculate cyclomatic complexity is [17] [23]:

Cyclomatic complexity (CC) = E-N+2P

Where P = Number of predicate nodes that contains conditions

E = Number of edges

N = Number of Nodes

C. *Compliance*

Compliance is actually a measure of software that determines whether software has been developed in accordance with the standard rules or not. If the rules are violated while developing the software, then the compliance of software is affected [1] [28].

The study calculates the compliance of software components in terms of percentage by using the formula [32]:

Compliance = ((Total number of rules – Number of rules violated) / Total number of rules) * 100

D. *Maintainability Index*

Maintainability Index is software metric that is used to measure the potential of source code to be maintained and to be supported. It gives a measure about the software that how easily its source code can be maintained [3] [14] [21]. Maintainability index is calculated by using a formula which includes other factors such as halstead volume, cyclomatic complexity, and lines of code [3].

After calculating maintainability index a numeric values is obtained ranging from 0 to 100. A higher numeric value means higher maintainability. The range from 0 to 100 is further classified in three levels to rate the maintainability. The level 20 to 100 represents high maintainability, the level 10 to 19 indicates moderate maintainability, and the level 0 to 9 indicates low maintainability [3] [21]. The mathematical formula used for calculating maintainability index is [3]:

MI = MAX (0, (171 – 5.2 * ln (Halstead Volume) – 0.23 * (Cyclomatic Complexity) – 16.2 * ln (Lines of Code)) * 100 / 171)

VI. FRAMEWORK TO IMPROVE MAINTAINABILITY OF REUSABLE SOFTWARE COMPONENTS

The study proposes a framework to improve the maintainability of reusable software components. It floats an idea of creating a new version of a software component to improve its maintainability. It should be noted that the maintainability is different than that of maintenance.

Maintainability – It refers to the degree of ease with which a software component can be maintained. Also it states the tendency of a software component to be maintained [22]. If the maintainability index of a software component increases, it shows that the component can be maintained more easily. Otherwise, if the maintainability index of a software component decreases, it shows that it will be difficult to maintain a software component [10].

Maintenance – Maintenance refers to the process of removing faults, errors, defects, and improving the performance of software components [13]. It is required that a component should have a greater maintainability index, so that there should be a requirement of less effort to maintain the software components.

Whenever, a component is under the process of maintenance, there are changes that occur while maintaining a software component. The quantum of changes may vary as per the requirement of changes. That is, there may occur minor changes or major changes in a software component while maintaining it [25]. These changes then made a new version of a reusable software component which has something different than its previous version. These changes in software components are generally represented using the version numbers. The version numbers are allocated using a numerical set of two or three values separated with '.' or dot. The first number in a version number represents the major changes or the addition of new functionality to the component, the second number in a version number represents the minor but significant changes, the third number in a version number represents the minor and few changes or the bug eliminations made to the software component [25].

The study proposes that when a new version of a reusable software component is created its maintainability improves [10]. The metrics should be used for measuring the maintainability, complexity and other factors as mention in the section 5. It should be used for measuring both the older and newer versions of reusable software components. When the metrics are used to measure the older version, it should be helpful in exhibiting the weak areas of software components, and when the metrics are used to measure the newer versions, it should be helpful in exhibiting the improvements in software components. By way of using metrics, one can assess the exact state of older and newer versions of reusable software components.

This framework has been constructed to serve the requirements of consumer and the producer of reusable software components. As it is a well known fact that the reusable software components can be viewed from two viewpoints such as from the consumer and from the developer's point of view [13]. Now, this framework facilitates both the consumers and the producers of reusable software components. For consumers: The consumers of reusable software components before selecting a component for use can make use of metrics to assess the maintainability of different versions of reusable software components. They should be able to select those versions of a component that have the high maintainability index. This way of selecting reusable software components will be benefitted in the maintenance effort of components [2] [13]. For producers: The producers of reusable software components should be able to produce quality components and with high maintainability index when using this framework. They should be able to assess the exact state of reusable software components before performing the act of maintenance. The metrics can be used to assess the components. The framework will thus be helpful in giving direction to the maintenance of components. The producers can increase the maintainability of reusable software components in order to have better selling prospects [2] [13].

A. Activities to be performed in framework

This section specifies the activities to be performed when using this framework. The activities are again classified according to the consumers and the producers of reusable software components [13].

1) *For consumers:*
   a. The metrics should be used to measure the maintainability index of different versions of a component.
   b. The version of a component with higher maintainability index should be selected.
2) *For producers:*
   a. The metrics should be used to assess the exact state of reusable software components before performing the act of maintenance.
   b. It will then identify the weak areas.
   c. The maintainability index can then be increased by applying the required efforts.

B. Case study

The study has taken a case of open source software named as 'NUnit' [24]. The study considers the five versions of the current release of NUnit 3.0, i.e. NUnit 3.4.1, 3.2.1, 3.2.0, 3.0.1, 3.0.0. NUnit is a unit testing framework for Microsoft.Net. It is a unit testing tool. It is written in C#.Net. It serves for the testing needs of all .Net languages. It serves the same purpose as is done by the JUnit for java applications. NUnit 3.0 is open source software and is released under the MIT license [24]. This allows for the use of NUnit in free applications and in commercial applications and also in libraries without any limitations [24].

The reason for selecting NUnit as a case study is that it is open source software [24]. The possibility of open source software to be reused is more. That is, the probability of open source software to be reused is more as compared to the non-open source software (proprietor based software). This is because, the open source software is available freely, and their source code is available freely. So making changes in open source software is easier and creating new version of open source software is easier. This makes the open source software more compatible with other software.

C. Results for different versions of NUnit

The study has obtained the results for five versions of current release of NUnit 3.0. These versions are NUnit 3.4.1, NUnit 3.2.1, NUnit 3.2.0, NUnit 3.0.1, and NUnit 3.0.0 [24]. The results for these versions of NUnit are obtained by using two software tools such as 'visual studio 2015', and 'NDepend' [19] [28].

Visual studio 2015 is used as a tool to measure the maintainability index of different versions of NUnit. The

visual studio 2015 provides an in-built functionality to measure the maintainability index of software. In visual studio 2015, there is a feature called 'Analyze' that runs analysis on software and produces the code metrics results [19]. The study has used this functionality to measure the maintainability index of NUnit.

NDepend tool is used to measure the cyclomatic complexity, size, and compliance of NUnit software. The cyclomatic complexity is measured on method basis [28]. The size is measured on the basis of logical lines of code (LLOC) [28]. Compliance of NUnit is measured with the help of summary provided by NDepend for rules violation [28] [31].

1) NUnit 3.4.1:

Table 1: Version and Maintainability Index of NUnit 3.4.1

Version Name	MI	Avg. CC	Size	Compliance*
NUnit 3.4.1	170.19	2.17	27042LOC	93.19%

2) NUnit 3.2.1:

Table 2: Version and Maintainability Index of NUnit 3.2.1

Version Name	MI	Avg. CC	Size	Compliance*
NUnit 3.2.1	169.09	2.17	26733LOC	93.87%

3) NUnit 3.2.0:

Table 3: Version and Maintainability Index of NUnit 3.2.0

Version Name	MI	Avg. CC	Size	Compliance*
NUnit 3.2.0	167.82	2.17	26506LOC	93.87%

4) NUnit 3.0.1:

Table 4: Version and Maintainability Index of NUnit 3.0.1

Version Name	MI	Avg. CC	Size	Compliance*
NUnit 3.0.1	166	2.14	25298LOC	93.87%

5) NUnit 3.0.0:

Table 5: Version and Maintainability Index of NUnit 3.0.0

Version Name	MI	Avg. CC	Size	Compliance*
NUnit 3.0.0	165.95	2.14	25160LOC	93.87%

6) Comparison of different versions of NUnit:

Table 6: Comparison of Versions and Maintainability Index of NUnit

Version Name	MI	CC	Size	Compliance*
NUnit 3.4.1	170.19	2.17	27042LOC	93.19%
NUnit 3.2.1	169.09	2.17	26733LOC	93.87%
NUnit 3.2.0	167.82	2.17	26506LOC	93.87%
NUnit 3.0.1	166	2.14	25298LOC	93.87%
NUnit 3.0.0	165.95	2.14	25160LOC	93.87%

* Compliance percentage has been calculated considering the critical rules violations [28].

From the comparison of different versions of NUnit it has been observed that with the increase in versions the maintainability index increases. With the increase in maintainability index and versioning of the software, the size of software increases i.e. lines of code increases. But with the increase in size the complexity of software remains almost consistent, just with few variations. Also the compliance of software remains intact.

Generally, there remains the possibility that with the increase in size the complexity may increase and compliance may decrease. But in this case, both the complexity and the compliance remain consistent. The main objective of the study to improve maintainability of software through versioning is being fulfilled. That is, it is evident that the maintainability index increases with the creation of new version of software. Thus, it can be considered as one of the methods to improve the maintainability of software.

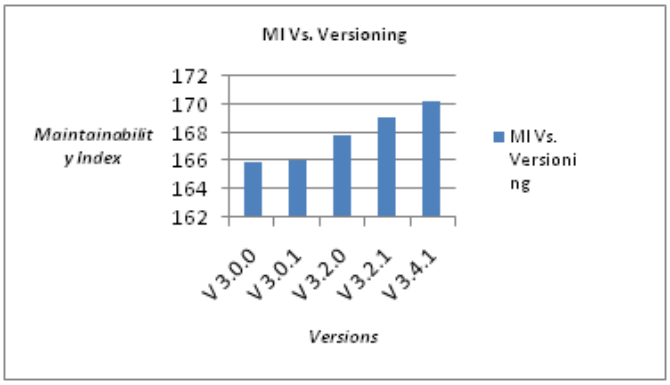

Figure 1: Maintainability Index and Versions of NUnit

A graph between maintainability index and versions of NUnit software shows that the maintainability index increases as the version of software increases. It can be seen from the graph that when there is a small change from version '3.0.0' to '3.0.1', the increase in maintainability index is also small [25]. When there is a significant increase from version '3.0.1' to '3.2.0', there is a significant increase in maintainability index. This significant increase in maintainability index is also reflected from version '3.2.1' to '3.4.1' [25].

VII. RELATED WORK

Kwon in 1997 conducted a study that investigated the implementation of maintenance activities in conjunction with software reuse. Kwon proposes a model that integrates the activities of software reuse with software maintenance. It proposes to integrate the software maintenance activities with software configuration management activities. It also provides with a prototype named as 'TERRA' to implement the proposed model. It addresses the problems faced by software during maintenance such as lack of integrated maintenance and support environment, lack of integrated tools and methods for maintenance related activities of reusable software components [13]. But the study lacks in evaluating that how the maintenance of reusable software components will be affected by integrating it with software configuration management activities. The study conducted in this paper fills this gap by evaluating the effect of software versioning on reusable software components.

Voas in 1998 conducted a research that was intended to upgrade the maintenance process for software components. The study discusses the maintenance issues concerning incompatibility, reliability, and third-party software components. It states an instance of year 2000 (Y2K) problem where the actual life of software exceeds its estimated lifecycle [33]. That in turn causes a problem for maintaining the software. The problem mentioned also relates to the maintenance problems of legacy software components. The solution to these problems lies in versioning of software components. Versioning extends the life of software components, makes the components compatible with the present technological platform, and will be able to increase the reliability of software components as it is a legacy component.

In 2007 Koponen conducted a research on the maintenance processes of open source software projects through defect and version management systems. The study has designed a process framework for maintenance of open source software and tries to measure these processes by using metrics. The study also presents an evaluation framework to measure the maintenance processes of open source software through defect and version management system. The study establishes the relation between maintenance processes and defect management system and measures its effects on maintenance processes [11]. But it lacks in establishing the relation between maintenance processes and version management systems and also lacks in measuring its effects on maintenance processes. The study in this paper establishes the relation between versioning and maintenance by taking a case of open source software. The study concludes that versioning provides a better environment for maintenance of open source software.

Khondhu et. al in 2013 calculated the maintainability index of a population of open source projects. The study tries to establish a relation between the size growth over a period of time and the maintainability index of open source components. It finds that the maintainability index of open source software components increases with increase in size of components. That is, as the number of lines increases in software components over a period of time the maintainability index also increases [10]. These results support the results obtained in this paper of study. Thus, the study by khondhu et. al in 2013 helps to validate this piece of study as it involves a large population of open source software components.

VIII. CONCLUSION AND FUTURE WORK

The study states a method for maintenance of reusable software components. The method is intended to improve the maintainability of reusable software components by creating a new version. It determines the factors that affect the maintainability of reusable software components. It makes use of metrics to calculate the maintainability index. It validates the concept through a case study. In a case study, it determines the effect of factors on maintainability of reusable software components. Through case study, it has been validated that the maintainability index of reusable software components improves with versioning. A comparison of the study is also made with other related studies in order to exhibit its contribution.

In this study, the effects of factors on maintainability of reusable software components mentioned in section 4 have been measured. The concept stated in this study can be extended to reusability for future work. That is, the impact of maintainability on reusability of reusable software components should be measured. It should be checked that how the maintainability affects the reusability of reusable software components.

IX. REFERENCES

[1] Bhakthavatsalam, N., Jayaraaman, A., D'Souza, G., Raghavachar, P., Gopal, P., Gosselin, J., Garrison, J., Cloutier, L. (2015). Managing Compliance. In *Oracle Enterprise Manager Lifecycle Management Administrator's Guide* (pp. 45-1-45-86). Oracle. Retrieved November 20, 2016, from https://docs.oracle.com/cd/E24628_01/em.121/e27046/compliance_lcm.htm#EMLCM93381.

[2] Birth, H., Durrmann, V., & Strohmaier D. (2014). *Difference between Closed and Open Source Software Maintenance.* Retrieved June 12, 2016, from https://wiki.oulu.fi/download/attachments/45090047/ossd_2014_birth_durrman_strohmaier.pdf?version=1&modificationDate=1416734248000&api=v2

[3] C. (2007, November 20). Maintainability Index Range and Meaning. Retrieved December 6, 2016, from https://blogs.msdn.microsoft.com/codeanalysis/2007/11/20/maintainability-index-range-and-meaning

[4] Cyclomatic complexity and its Applications. Retrieved December 3, 2016, from http://www.guru99.com/cyclomatic-complexity.html

[5] Dart, S., Christie, A. M., & Brown, A. W. (1993). *A Case Study in Software Maintenance* (pp. 1-58, Tech. No. CMU/SEI-93-TR-8). Pittsburgh, Pennsylvania: SEI-CMU. doi:http://www.sei.cmu.edu/reports/93tr008.pdf (NTIS No. ESC-TR-93-185)

[6] Farago, C., Hegedus, P., Ladanyi, G., & Ferenc, R. (2015). Impact of Version History Metrics on Maintainability. *2015 8th International Conference on Advanced Software Engineering & Its Applications (ASEA),* 30-35. doi:10.1109/asea.2015.14

[7] Heitlager, I., Kuipers, T., & Visser, J. (2007). A Practical Model for Measuring Maintainability. *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007).* doi:10.1109/quatic.2007.8

[8] Hristov, D., Hummel, O., Huq, M., & Janjic, W. (2012). Structuring Software Reusability Metrics for Component-Based Software Development. *ICSEA 2012: The Seventh International Conference on Software Engineering Advances*, 421-429. ISBN: 978-1-61208-230-1

[9] IEEE Standards Association. (2014). IEEE Standards Definition Database Search-S. Retrieved September 9, 2015, from http://standards.iee

[10] Khondhu, J., Capiluppi, A., and Stol, K. (2013) Is It All Lost? A Study of Inactive Open Source Projects, In: Proceedings of the 9th International Conference on Open Source Systems.

[11] Koponen, T. (2007). *Evaluation of maintenance processes in open source software projects through defect and version management systems* (Doctoral thesis, University of Kuopio, Finland, 2007) (pp. 1-92). University of Kuopio. doi:http://epublications.uef.fi/pub/urn_isbn_978-951-27-0107-0/urn_isbn_978-951-27-0107-0.pdf

[12] Koponen, T., & Hotti, V. (2005). Open source software maintenance process framework. *ACM SIGSOFT Software Engineering Notes,30*(4), 30-34. doi:10.1145/1082983.1083265

[13] Kwon, O.C. (1997). *A process model for maintenance with reuse: an investigation and an implementation abstract*. Durham theses, Durham University. Available at Durham E-Theses Online: http://etheses.dur.ac.uk/4724/

[14] Land, R. (2002). Measurements of Software Maintainability. Retrieved June 18, 2016, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.600.6609&rep=rep1&type=pdf

[15] Larsson, M., & Crnkovic, I. (2000), *Component Configuration Management*. Retrieved June 3, 2016, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.36.2394&rep=rep1&type=pdf

[16] Lee, Y., & Chang, K. (2007). *Automated source code measurement environment for software quality* (Doctoral thesis). Auburn University, Alabama. Retrieved June 19, 2016 from https://etd.auburn.edu/bitstream/handle/10415/189/Lee_Young_28.pdf?sequence=1

[17] McCabe Cyclomatic Complexity. Retrieved December 3, 2016, from http://www.chambers.com.au/glossary/mc_cabe_cyclomatic_complexity.php

[18] Michura, J., Capretz, M. A., & Wang, S. (2013). Extension of Object-Oriented Metrics Suite for Software Maintenance. *ISRN Software Engineering,2013*, 1-14. doi:10.1155/2013/276105

[19] Microsoft. (2015). Visual Studio 2015 (Version 2015) [Computer software]. Retrieved October 20, 2016, from https://www.visualstudio.com/downloads/

[20] Murta, L., Oliveira, H., Dantas, C., Lopes, L.G., & Werner C. (2004). *Towards Component-based Software Maintenance via Software Configuration Management Techniques*. Retrieved September 8, 2015, from http://reuse.cos.ufrj.br/prometeus/publicacoes/cbsm.pdf

[21] Naboulsi, Z. (2011, May 26). Code Metrics – Maintainability Index. Retrieved December 6, 2016, from https://blogs.msdn.microsoft.com/zainnab/2011/05/26/code-metrics-maintainability-index/

[22] O. (2015, July 28). Maintainability. Retrieved November 17, 2016, from https://en.wikipedia.org/wiki/Maintainability

[23] Pherson, S. M. (2004, September 24). Cyclomatic complexity. Retrieved December 3, 2016, from https://en.wikipedia.org/wiki/Cyclomatic_complexity

[24] Pool, C., Prouse, R., Busoli, S., & Colvin, N. (2015). NUnit (Version 3.4.1, 3.2.1, 3.2.0, 3.0.1, 3.0.0) [Computer software]. Retrieved November 19, 2016, from https://www.nunit.org/

[25] Rouse, M. (2007). Versioning. Retrieved November 11, 2016, from http://searchsoftwarequality.techtarget.com/definition/versioning

[26] Saraiva, J., Barreiros, E., Almeida, A., Lima, F., Alencar, A., Lima, G., . . . Castor, F. (2012). Aspect-oriented software maintenance metrics: a systematic mapping study. *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*. doi:10.1049/ic.2012.0033

[27] Sjøberg, D. I., Anda, B., & Mockus, A. (2012). Questioning software maintenance metrics. *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '12*, 107-110. doi:10.1145/2372251.2372269

[28] Smacchia, P. (2016, April 29). NDepend (Version 6.3) [Computer software]. Retrieved November 2, 2016, from http://www.ndepend.com/download

[29] Software Engineering Standards Committee. (1998). *1219-1998 IEEE Standard for Software Maintenance*. New York, USA: IEEE / Institute of Electrical and Electronics Engineers Incorporated. ISBN: 0-7381-0336-5

[30] Software maintenance. (2004, July 5). Retrieved November 17, 2016, from https://en.wikipedia.org/wiki/Software_maintenance

[31] U. (2007, May 21). Thread: Percentage Compliance. Retrieved December 10, 2016, from http://www.dbforums.com/showthread.php?1618548-Percentage-Compliance

[32] Vattumalli, N. B. (2010). *Panorama - a software maintenance tool* (Master's thesis, Iowa State University, 2010) (pp. 1-62). Iowa State University. doi:http://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=2882&context=etd

[33] Voas, J. (1998). Upgrading Software Maintenance for Components. Retrieved September 9, 2015, from https://www.cigital.com/papers/download/maintain.pdf

[34] Wheeler, D. (2003, August 12). Source lines of code. Retrieved December 3, 2016, from https://en.wikipedia.org/wiki/Source_lines_of_code

