Contributors to Reduce Maintainability Cost at the Software Implementation Phase

Mohammed Abdullah H. Al-Hagery
Faculty of Computer/Department of Computer Science,
Qassim University, Boridah, KSA

Abstract

Software maintenance is important and difficult to measure. The cost of maintenance is the most ever during the phases of software development. One of the most critical processes in software development is the reduction of software maintainability cost based on the quality of source code during design step, however, a lack of quality models and measures can help asses the quality attributes of software maintainability process. Software maintainability suffers from a number of challenges such as lack source code understanding, quality of software code, and adherence to programming standards in maintenance. This work describes model based-factors to assess the software maintenance, explains the steps followed to obtain and validate them. Such a method can be used to eliminate the software maintenance cost. The research results will enhance the quality of the source code. It will increase software understandability, eliminate maintenance time, cost, and give confidence for software reusability.

Keywords: Maintainability Time, Software Maintenance, Standard Code, Quality of lines of Code, Understandability, Maintainability Factors.

1. INTRODUCTION

Software maintenance is an important phase in the software life cycle. It focuses on keeping the software fully functional and up to date. Maintenance engineers used different approaches and methods to gain understanding of software systems so maintenance tasks can be performed effectively. A lot of efforts have been put into finding a way to measure maintainability of software [1].

Maintainability cannot be seen as an attribute of the software system alone, because it depends a great deal on who maintains it, a team that has a lot of experience with a particular system will maintain it more easily. Both the software and the team have internal attributes that influence maintainability, for example, structural complexity of the software and skill of the team members. We want to survey the factors that lead to low or high maintainability [2].

A change request can be due to a failure, changing requirements, prevention or any other reason. The activities by the maintenance team include actually performing the change, but also documenting, testing, and reporting, depending on the maintenance procedures. When a system is changed extensively a new team is formed to implement the changes that are not regarded as a change. Such a situation is more like a new system being developed [2]. There are many factors that influence maintainability can be assembled and adapted from [3], [4], [5], [6], [7], [8], [9]. Measuring the maintainability of source code revisions presents some challenges [10].

This work concentrates on quality of source code rather than code defects. Code defects are defects attributable to coding errors such as branching to a wrong location. These defects are found throughout the coding process as well as in final test of changes and enhancements to an application.

1.2 Survey of Related Works

The largest cost associated with any software product over its life-cycle is the software maintenance cost. One approach to controlling maintenance costs was to utilize software metrics during the development phase [11]. A number of studies is examining the link between Object Oriented software metrics and maintainability have found that in general these metrics can be used as predictors of maintenance effort [12],[11],[13],[14], and [15], which can be measured in working hours.

Yuming and Hareton, presented an empirical study that sought to build object-oriented software maintainability prediction models using a novel exploratory modeling technique, MARS. To build the MARS models, they made use of the Li and Henry's data sets, UIMS and QUES, obtained from two different object-oriented systems. The prediction performances of the MARS models were assessed and compared with those of the multivariate linear regression models. These models are the artificial neural network models, the regression tree models, and the support vector models, but their focus was not on the implementation phase and data set used was not enough to prove the suggested model [16].

Mari et al. introduces the framework of maintainability and the techniques that promote maintainability in three abstraction levels; system, architecture and component. In system dimension, the maintainability requirement is considered from a business related point of view. In architecture, maintainability means a set of quality attributes e.g. extensibility and flexibility. At the component level, maintainability focuses on modifiability, reusability, integration, and testability [17]. Ardimento et al. in [18] reports the results of their empirical study aimed at understanding how characterizations of components affect the maintenance effort of the system components. They have made the assessment that:

(i) Functionality of each component should be as concentrated as possible over a single aspect of the application domain,
(ii) The training time offered by the component's producer usually indicates the complexity of understanding it and if a component is difficult to understand, then it is difficult to maintain; and
(iii) A deep knowledge of the component is necessary for the organization before its adoption.

Van Koten and Gray, make the first use of the BBNs in building software maintainability prediction models. They use a special type of Bayesian networks called Naïve–Bayes classifier, which assumes no expert knowledge about the prior probability distribution but learns it from data by batch learning. The results show that the prediction accuracy of the BBN model is more accurate than regression-based models for one system but is less accurate than regression-based models for another system. Accurate software metrics-based maintainability prediction is desirable first because it reduces future maintenance efforts by enabling developers to better identify the determinants of software quality and thereby improve design or coding, and second because it provides managers with information for more effectively planning the use of valuable resources. Although a number of maintainability prediction models have been developed in last decade, they have low prediction accuracies according to the criteria suggested in [15], [19].

Maintainability metrics are commonly language dependent, and computing them requires tools that typically assume access to the full definitions of the software entities [10]. It was found that a number of metrics such as the lines of code changed, and the number of operators changed are strongly correlated to maintenance efforts [1]. Heitlager et al. discussed several problems with the maintainability index (MI), and they identified a number of requirements to be fulfilled by a maintainability model to be usable in practice. they sketched a maintainability model that alleviates most of these problems, and discussed their experiences with using such as system for IT management consultancy activities [20].

Bertoa et al. have been reported that they presented a set of measures to assess the maintainability of software components. Furthermore, they described the process followed to obtain and validate them. Such a process can be maintained for defining and validating measures for other quality characteristics [21]. Wu et al. proposed a technique for maintaining evolving component based system by utilizing a static analysis to identify the interfaces, events and dependence relationship that would be affected by the modification in the maintenance activity [22], [23]. The maintainability of a software system can be measured in different ways. Currently and in past studies, maintainability has been defined as ''time required to make changes'' and ''time to understand, develop, and implement modification''[24]. As well as, Yuming and Hareton measured the maintainability of a software system as the number of changes made to code during a maintenance period. They employed a novel exploratory modeling technique, multiple adaptive regression splines (MARS), for building maintainability prediction models using the metric data collected from two different object-oriented systems [16].

1.3 Motivations and Objective

One of the most critical processes in software development is the reduction of software maintainability cost accordingly the quality of code design, however, a lack of quality models and metrics can help asses the software maintainability process. Software maintainability suffers from many challenges such as lack of source code quality, and source code understanding, adherence to programming standards in maintenance. The main objective of this work is to define and establish a Criteria-Based-Model that can be used to assess S/W quality characteristics, and that can assist in implementation phase. Such criteria could reduce the maintenance cost; these criteria will be created as three or one group. This objective can be detailed in the following points:

1. Create a group of criteria that support writing a standard software programs(proposed criteria)
2. Construction of a mathematical model for applying the proposed criteria to reduce the final S/W cost.
3. Increase S/W understandability, readability and flexibility.
4. Participation of undergraduate students in the research work through the formation of work groups to study the code standardization, to write some programs and then execute software maintenance on several software programs. These programs help to ensure acceptance of the model and the proposed factors or criteria.

2. SOFTWARE MEASUREMENT

Software measures can be classified into three types; derived measures, base measures, and indicators. Base measures do not depend upon any other measure (e.g., the number of tables in the manuals). A derived measure is derived from other base or derived measures (e.g., the ratio of methods per interface). An indicator is a measure that is derived from other measures using an analysis model according to decision criteria. The objective of that is to obtain a measurement result that satisfies an information need (e.g., the size of a sub-system is “medium” if it has more than 30 assemblies, provides more than 45 interfaces, and its manuals have more than 7,000 Line of Code (LOC).

Measures relate a defined measurement approach and a measurement scale. A measurement approach is the logical sequence of operations, described generally, used in quantifying an attribute with respect to a specified scale [25]. A measure is expressed in units, and can be defined for more than one attribute. Examples of measures for software component attributes include the number of provided interfaces, the ratio of methods per required interface, or the throughput of video frames emitted per input video frame (they correspond, respectively, to possible measures for the aforementioned attributes size, interface complexity, and performance)[21].

Accurate software metrics-based maintainability prediction can not only enable developers to better identify the determinants of software quality and thus help them improve design or coding, it can also provide managers with useful information to help them plan the use of valuable resources[16].
The act of measuring software is a measurement, which can be defined as the set of operations that aims at determining a value of a measurement result, for a given attribute of an entity, using a measurement approach [21].

The term metric is not present in the measurement terminology of any other engineering disciplines, at least with the meaning it is commonly used in software measurement. Therefore, the use of the term “software metric” seems to be imprecise, while the term “software measure” `seems to be more appropriate to represent this concept. Accordingly, in the following the term measure will be used. This is also consistent with ISO/IEC and IEEE Computer Society positions which, in order to ensure both consensus and consistency with other fields of sciences, made a decision in the year 2002 to align their terminologies on measurement with the internationally accepted standards in this field. In particular, ISO-JTC1-SC7 is trying to follow as much as possible the ISO international vocabulary of basic and general terms on metrology [26]. A number of software metrics measuring maintainability has been proposed by means of theoretical and empirical studies. However, component based system presents a unique maintenance challenges. Unlike the traditional software systems, one cannot be done by viewing or changing the source codes of the component, but are restricted to reconfiguring and reintegrating components [27].

3. MAINTAINABILITY

Maintainability [28] is "The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment". The seminal research work by Basili and Turne in 1975 has identified different characteristics of software system that effect software maintainability. Effective maintenance involves detailed observations of the behavior of a system and is driven by software complexity [29]. Voas in 1998 provided an overview of the maintenance challenges raised by Component Based Software Development by identifying reasons including frozen functionality, incompatible upgrades, unreliable components and complex middleware [27].

The "understandability" of a source code is related directly to the maintainability, because understandability is one of the dominant factors affecting software maintainability [30]. For example, let us assume a perfect source code that does not have any faults or logical errors. Nevertheless, if a source code is difficult to understand, an increase of costs and/or of failure potential during maintenance is then inevitable. Several factors such as complex logic, the many variables included in a code and lengthy codes could interfere with the understanding of the program context by maintenance personnel [31].

3.1 Maintainability Attributes

The software maintainability affects by a number of criteria such as: understandability, reusability, learnability, readability, and operability. It can be defined as follow:

- Understandability: the capability of the component to enable the user to understand whether the component is suitable, how it can be used for particular tasks and conditions of use. System developers should be able to select a component suitable for their intended use, for example, component elements (e.g. interfaces, operations) should be easy to understand [21].
- Reusability: the capability of the software to enable the developer or the maintainer to modify its functions easily.
- Readability: the ability of the software to enable the developer or maintainer to understand the software functions by reading its lines of source code.
- Learnability: the capability of the software component to enable the user or system developer to learn its application. For example, the user manual and the help system should be completed, the help should be context sensitive and explain how to perform common tasks, etc.
- Operability: the capability of the software component to enable the user (system developer) to operate and control it. An Operability measure should be able to assess whether system developers can easily operate and control the component. Operability measures can be categorized by the dialogue principles described in ISO/IEC-9241-10 [21]. Figure 1 illustrates the relation between maintainability and source of code.

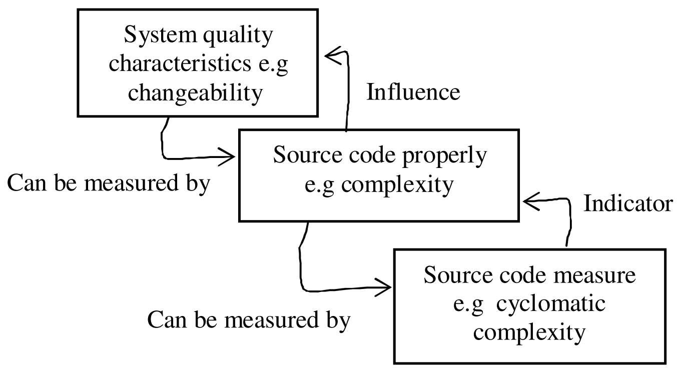
System quality characteristics e.g changeability
Influence
Source code properly e.g complexity
Can be measured by
Indicator
Source code measure e.g cyclomatic complexity
Can be measured by

FIGURE 1: The relation between maintainability and source of code

3.2 Factors Affecting Maintainability

In [32], four main factors for software maintainability are included in ISO/IEC 9126 such as analyzability, changeability, stability, and testability. These factors are defined clearly in [33]. First, analyzability is attributes of software that bear on the effort needed to diagnose of deficiencies or causes of failure and to identify parts to be modified. Second, changeability is some attributes of software that bear on the effort needed to make modifications, eliminate faults or change the system in response to environmental change. Third, stability that can be represented by attributes of software that bear on the risks associated with unexpected effects of modifications. Finally, Testability attributes of software that bear on the effort needed to validate modifications.

Studies that take the application development view of software seem to address maintenance as an afterthought of development rather than a critical and expensive part of the total system life-cycle. For example, Dekleva [34] evaluates how the choice of development approach will influence maintenance. Perry in [35] addresses maintenance quality in the context of development quality. The maintenance phase of the life-cycle is a natural and necessary part of the system operation [36]. Software evolves over time primarily due to changes in requirements and technologies. As a result, Information systems development is typically acknowledged as an expensive and lengthy process, often producing code that is of uneven quality and difficult to maintain. Software reuse has been advocated as a means of revolutionizing this process. The claimed benefits from software reuse are reduction in development cost and time, improvement in software quality, increase in programmer productivity, and improvement in maintainability [37]. Prasanth et al., proposed a model for improving software maintainability based on risk analysis, they identified a set of metrics that affects the external and internal complexity [38].

4. QUALITY OF SOURCE CODE

There are two main types of software quality, Quality of process and quality of products. In general, there is a lack of consensus about how to define and categorize software quality characteristics. Quality of system documentation includes quality of external documentation and quality of internal documentation [39].

The development of high-quality software must satisfy both the users' requirements and the software firm's budget [40]. Program restructuring is a key method for improving the quality of ill-structured programs, thereby increasing the understandability and reducing the maintenance cost [41]. Our concentration is on some important rules of code design. Quality is one of the most sought after dimensions of the business software applications that organizations depend on today. Despite this high demand for quality, very few studies have been done that evaluate the ongoing quality of software applications during the maintenance portion of the system life-cycle [42]. Quality is also measured objectively as number of failures and defects per month [42] and also quality can be supported by a standard implementation of code which, will result in quality software maintenance.

5. METHODOLOGY STEPS

The research methodology includes; establishment of some criteria related to standard code design, construction of a suitable model for measuring the values of the proposed criteria, maintain of construction groups (BSc students team), and results comparison. The following steps are representing the research methodology in details

1- Construction of documentation criteria and evaluation formula as shown in Table 1 and Formula 1.

2- Preparation of code segments (two sets, each one contains 18 programs) by two ways

a) Undocumented code, denoted by g1

b) Documented code, denoted by g2

3- Execute a short training course in the international documentation standards, to train two groups of code maintainers (four Bsc students)

4- Apply the criteria of Table 1 on g1 and g2 separately, the calculated results are shown in Table 2.

5- Calculate the total satisfaction for each set.

6- Maintain the software code (g1 & g2) depends on adaptive maintenance, then calculate the maintainability time for each program in g1 and g2.

7- Results comparison

5.1 Coding Factors

The proposed factors selected depend on three groups [43], these factors increase the code understandability; this will reduce the maintainability time of software. The proposed factors are thirteen factors, can be classified in three groups; first associated with general code, second associated with methods, third associated with classes. Each factor can be assigned to any of the following values {0,1,2,3,4}. Where, 0 indicates that the factor effect is absent, 1 means factor satisfaction is low, 2 means factor satisfaction is medium, then 3 is high and 4 means factor is completely satisfied (very high), kindly see Formula (1), that was created by Al-Hagery [43], the values of any factor FR in Table 1 can be estimated by Formula (1).

$$FR\_measure = \begin{array}{l} 0 : iff\ satisfaction \geq o\ \&\ < 10\% \\ 1 : iff\ satisfaction > 10\%\ \&\ \leq 25\% \\ 2 : iff\ satisfaction > 25\%\ \&\ \leq 50\% \\ 3 : iff\ satisfaction > 50\% \leq 75\% \\ 4 : iff\ satisfaction > 75\% \leq 100\% \end{array} \qquad (1)$$

The proposed factors were extracted from three groups of factors implemented in [43]. These factors produce a high quality code to reduce the maintainability cost. These factors are shown in Table 1.

Index	Factor name	Factor rank (FR)				
		0	1	2	3	4
1	Variables scope and role are defined clearly			○		
2	Code describes what is being done		○			
3	Understand the code by reading the comments					○
4	Preface comments defined clearly		○			
5	Use nouns or noun phrases for naming					○
6	Use alignment to enhances readability			○		
7	End of lines comments				○	
8	The meaning of return values		○			
9	Use verbs for Function names, Get, Find, …					○
10	The purpose of each method/function		○			
11	Variables declarations should be left aligned				○	
12	Use correct spelling in names		○			
13	Avoid using names that differ only by letter			○		
	Total Satisfaction = 29	**0**	**5**	**3**	**2**	**3**

TABLE 1: Maintenance Based Factors

Our model-based factor (MBF) is proposed to find the degree of documentation based on some standard criteria, as shown in formula (2).

$$\text{MBF} = \sum_{i=1}^{n}(\text{Factor } i \times \text{Factor\_Rank}), \qquad n=13 \tag{2}$$

For the example, the value of MBF obtained in Table 1 is 29, this value gives an indicator of the documentation level, the minimum value of MBF is 0 and the maximum value is 52, so the value of this example classified as medium.

6. WORKING GROUPS

Two teams are selected for maintenance purpose, each team consists of two students, the development strategy used is the "extreme programming". Team members are a final year students at the Computer Science department. On the other hand, the teams studied and practiced the concepts of writing standard code and they created some documented code as a result of their training, but this is not included within the research data, because they were maintain a code written by another people.

7. RESEARCH DATA SETS

The maintenance task performed by using eighteen software programs designed in C & C++ programming languages. This software constitutes the research data set that was used to prove the research validity. This data set was prepared as two groups, the first group prepared as a documented code, its documentation level graduated from 66% to 12% as partially documented code. Second group is prepared as undocumented code as shown in Table 2 column 3.

8. EXPERIMENTAL RESULTS

Table 2 displays summary results in this research. It includes some important attributes such as Complexity level, level of documentation (g1 and g2), total time1 for group 1 and total time2 for group2 and indicators. All these attributes are selected to be used for results evaluation and interpretation. The table contents are organized in ascending order depends on the value of indicator of the last column. The indicator value is assigned as follows*:*

Time1 > Time2 → - (the results are negative)
Time1< Time2 → + (the results are positive)
Time1 equal Time2 → ≡ (the results are equal)

program no	Complexity level	Level of Documentation		Time1	Time2	Indications *
		g1	g2			
9	15	15	1	6	5	-
16	65	18	1	10	7	-
1	60	40	1	5	4	-
14	80	36	1	22	20	-
11	20	42	1	5	3	-
7	20	21	2	7	7	≡
3	50	26	1	2	2	≡
6	20	12	1	3	5	+
5	40	21	1	8	16	+
10	10	25	1	17	24	+
4	35	34	1	3	10	+
8	35	35	1	3	4	+
18	70	35	2	4	6	+
2	45	36	1	1	2	+
17	70	40	1	14	23	+
13	75	45	1	3	4	+
12	60	46	1	2	5	+
15	50	44	2	8	14	+
Average				**6.83**	**8.94**	

TABLE 2: Summary of Experimental results

9. RESULTS DISCUSSION

Based on the results shown above in Table 2, these results show the rate of time that was measured during the maintenance of 18 programs applied in this research. The maintenance time was measured in two separate cases. First case, contains programs classified as partially documented. The second case contains undocumented programs, In the first case and second case, the average rate of time for maintenance was equal to (6.38.3) and (8.94) units of time, respectively.

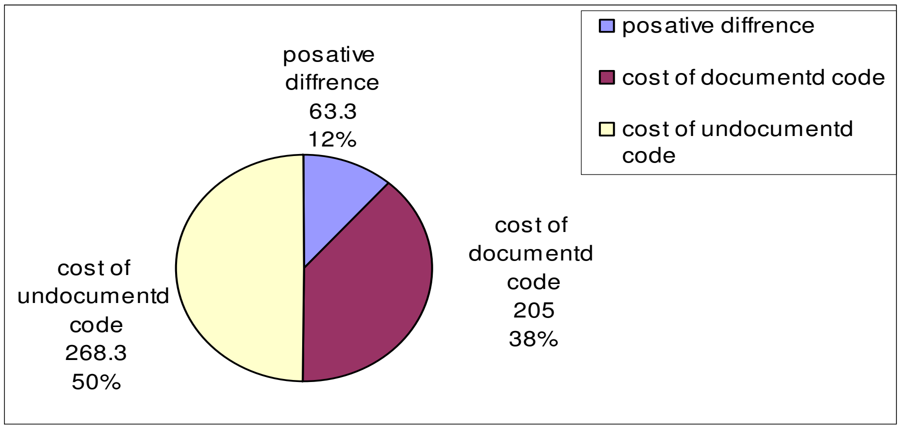

FIGURE 2: Maintenance cost results

Based on the previous values, found that the difference in time is equal to (2.11) unit of time, if we supposed that the cost of each maintenance hour is equal to 30 US$. So based on this value, the average cost of case 1 was 268US$ ≡ 50% of the total cost, and the average cost of case 2 was 205US$ ≡ 38% of the total cost, as well as the difference of the average in cost was 63.3 US $ ≡ 12% of the total cost, as presented in figure 2. Although there is a positive difference supports the principle of documented code which applied in this research. The total results showed three types of values, first gives a negative results, the second gives positive results, then the last gives a balanced results, as illustrated in figure 2.

Eleven programs of eighteen supports the principle of code documentation in a positive and shows important difference in the results, and based on this relative difference can present positive results for the proposed model, this obtained clearly how much cost can be reduced by building a complete documented code. Finally, it is important to mention that the average level of documentation of all applied programs was equal to 61% depends on both problem complexity and code size, and the average level of complexity of the used programs was equal to 46.

The more documentation process within large and complex programs, would contribute to the maintenance process required in the future, in addition to reducing the cost to do so. Also by comparing the results shown in Table 2, it is clear that small programs are not affected by documentation because its ideas is simple, easy, and the required time for maintenance is very short.

10. CONCLUSION

After discussing the results presented in this work, we found that applying the international quality standards on the code contents is very important to reduce its cost. In addition to that, it enables developers to reuse the source code. This code also will be more flexible, readable, easy to understand, and then S/W development organizations can do future development at a lower cost and better results depends on the results of this research. For programs that are small, simple, and well documented, they have negative results because the maintainers spend a lot of time and effort to understand the idea of the program by reading its documentation, although they can understand the idea directly without documentation of the Source code.

The presented results gave in general a positive effect of applying standard documentation process on software code, especially for long life software projects. The impact of this process is positive to support reducing the cost of software maintenance. By the proposed model we predicted that the medium level of software documentation reduces the cost of long-term maintenance by 12% and high level of software documentation (full documented code with complex programs) reduces the total maintenance cost by 24% at least, depending on the results comparisons presented above. This value is increasing with large, complex, and full documented projects/software. This also will encourage organizations to support the software quality by improving the developer's culture in this side, so any other S/W teams in future can enhance and improve documented legacy systems by adding new features or new functions.

11. FUTURE WORKS

There are some points can be taken into account to extend and modify this work from different points; firstly, increase the proposed factors to cover all quality factors. Secondly, improve the research results by increasing the number of maintenance teams. Thirdly, expanding the testing data to be more than 18 projects depends on big sizes, and complex projects that are completely documented.

12. REFERENCES

[1] M. Reformat, A. Kapoor, and N. J. Pizzi. “Software Maintenance: Similarity and Inclusion of Rules in Knowledge Extraction”, Proc of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), 2006, 723-731.

[2] W. Hordijk, and R. Wieringa. "Surveying the Factors that Influence Maintainability", In: Proc of the 10th European software engineering conference held jointly with 13th ACM

SIGSOFT international symposium on Foundations of software engineering, 5-9 Sep. 2005, pp. 385-388.

[3] N. E. Fenton, and S. L. Pfleeger. "Software Metrics: A Rigorous and Practical Approach", PWS Publishing Co, 1998.

[4] M. A. CÔTÉ, W. Suryn, C. Y. Laporte, and R. A. Martin. "The evolution path for industrial software quality evaluation methods applying ISO/IEC 9126: quality model: Example of MITRE's SQAE method. Software Quality Journal", Elements of Software Science, vol. 13, pp. 17-30, 2005.

[5] Y. Ahn, J. Suh, S. Kim, and H. Kim. "The software maintenance project effort estimation model based on function points", Journal of Software Maintenance, vol. 15, Issue 2, pp. 71-85, March/April 2003.

[6] L. Bass, P. Clements, and R. Kazman, "Software Architecture in Practice", Addison-Wesley, 2nd edition, 2003.

[7] T. L. Graves, and A. Mockus. "Inferring change effort from configuration management databases", In METRICS '98: Proc of the 5th International Symposium on Software Metrics, IEEE Computer Society, 1998, pp 267-273.

[8] M. Lehman. "Laws of Software Evolution Revisited", Software Process Technology (EWSPT 96), 1996, vol. 1149, pp 108-124.

[9] A. Monden, D. Nakae, T. Kamiya, S. Sato, and K. Matsumoto. "Software quality analysis by code clones in industrial legacy software", In IEEE METRICS '02: Proceedings of the 8th International Symposium on Software Metrics, 2002, pp. 87-87.

[10] H. Abram, W. G. Michael, and C. H. Richard. "Reading beside the lines: Using indentation to rank revisions by complexity", journal of Science of Computer Programming,
Vol. 74, Issue 7, pp. 414-429, May 2009.

[11] R. K Bandi, V. K. Vaishnavi, and D. E. Turk. "Predicting maintenance performance using object-oriented design complexity metrics", IEEE Transactions on Software Engineering, vol. 29, no.1, pp.77-87, 2003.

[12] W. Li, and S. Henry. "Object-oriented metrics that predict maintainability", Journal of Systems and Software, vol. 23, no. 2, pp.111-122, 1993.

[13] S. C. Misra. "Modeling design/coding factors that drive maintainability of software systems", Software Quality Journal, vol. 13, no. 3, pp.297-320, 2005.

[14] M. T. Thwin, and T. S. Quah. "Application of neural networks for software quality prediction using object-oriented metrics", Journal of Systems and Software, vol. 76, no.2, pp.147-156, 2005.

[15] K. V. Coten, and A. Gray. "An application of Bayesian network for predicting object-oriented software maintainability", Information and Software Technology, vol. 48, no.1, pp. 59-67, 2005.

[16] Y. Zhou, and H. Leung. "Predicting object-oriented software maintainability using multivariate adaptive regression splines", The Journal of Systems and Software, vol. 80, pp. 1349-1361, 2007.

[17] M. Mari, and N. Eila. "The impact of maintainability on component-based software systems", In Proc of 29th Euromicro Conference, 2003, p. 25-32.

[18] P. Ardimento, A. Bianchi, and G. Visaggio. "Maintenance-oriented selection of software components", In Proc of Eighth European Conference on Software Maintenance and Reengineering, 2004, p. 115-124.

[19] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. "Software Engineering Metrics and Models", Benjamin-Cummings Publishing, Redwood City, CA, USA,1986.

[20] I. Heitlager, T. Kuipers, and J. Visser. “A Practical Model for Measuring Maintainability", Proc of the 6th International Conference on the Quality of Information and Communications Technology, IEEE, 2007, pp. 44-49.

[21] M. F. Bertoa, J. M. Troya, and A. Vallecillo. "Measuring the usability of software components", The Journal of Systems and Software, vol. 79, pp.427-439, 2006.

[22] Y. Wu, and J. Offutt. "Maintaining evolving component-based software with UML", In Proc of Seventh European Conference on Software Maintenance and Reengineering, 2003, p. 133-142.

[23] Y. Wu, D. Pan, and M.H. Chen, "Techniques of maintaining evolving component based software", In Proc of International Conference on Software Maintenance, 2000, p. 236-246.

[24] L. S. Rising. "Information hiding metrics for modular programming languages", PhD dissertation, Arizona State University, 1992.

[25] ISO/IEC 15939, "Software Engineering-Software Measurement Process", 2002.

[26] ISO VIM, second ed. "International Vocabulary of Basic and General Terms in Metrology", International Standards Organization, Geneva, Switzerland, 1993.

[27] J. Voas. "Maintaining component based systems", IEEE Software, vol. 15, no. 4, pp. 22-27, 1998.

[28] IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE Std 610-1990, The Institute of Electrical and Electronics Engineers, New York, NY, 1990.

[29] G. T. Heineman, and W. T. Councill. "Component-Based Software Engineering: Putting the Pieces Together", Addison-Wesley, 2001, pp. 741-753.

[30]S. S. Yau, R. A. Nicholl, J. J. Tsai, and S.S. Liu. "An integrated life-cycle model for software maintenance", IEEE Transactions on Software Engineering, 1988, vol.14, no .8, pp.1128-1144.

[31] J. Park, W. Jung, and J. Ha. "Development of the step complexity measure for emergency operating procedures using entropy concepts", Journal of Reliability Engineering & System Safety, vol. 71, pp. 115-130, 2001.

[32] ISO/IEC 9126. "Software Engineering-Product Quality-Part 1: Quality Model", International Standards Organization, Geneva, Switzerland, 2001.

[33] C. Chen, C. Lin, C. Wang, and C. Chang. "Model for measuring quality of software in DVRS using the gap concept and fuzzy schemes with GA", Journal of Information and Software Technology vol. 48, pp.187-203, 2006.

[34] S. M. Dekleva. "The influence of the information systems development approach on maintenance", the journal of MIS Quarterly. Vol.16.issue.3, pp.353-372. 1992.

[35] W. E. Perry. "Quality concerns in software development", the challenge is consistency, Journal of Information Systems Management, vol. 9, Issue 3, pp. 48-50, 1992.

[36] M. A. Cusamano, and C. F. Kemerer. "A quantitative analysis of U.S. and Japanese practice and performance in software development", Journal of Management Science, vol. 36, issue 11, pp. 1384-1406, 1990.

[37] D. L. Nazareth, and M. A. Rothenberger. "Assessing the cost-effectiveness of software reuse: A model for planned reuse", The Journal of Systems and Software, vol. 73, pp. 245-255, 2004.

[38] P. Narayanan, S. P. Raja, X. Birla, K. Navaz, and S. A. Abdul Rahuman. "Improving Software Maintainability through Risk Analysis", International Journal of Recent Trends in Engineering, vol. 2, issue. 4, pp. 198-200,November 2009.

[39] J. A. Hoffer, J. F. George, and J. S. Valacich. "Modern Systems Analysis and Design", Third Edition, 2005.

[40] R. A. DeMillo, R. J. Liption, and A. J. Perlis. "Software Project Forecasting", Software Metrics, MIT Press, Cambridge, MA, p. 77, 1981.

[41] C. Lung, X. Xu, M. Zaman, and A. Srinivasan. "Program restructuring using clustering techniques", The Journal of Systems and Software, vol. 79, pp.1261-1279, 2006.

[42] M. Ghods, and K. M. Nelson. "Contributors to quality during software maintenance", Journal of Decision Support Systems, vol. 23, issues 4. pp. 361-369, 1998.

[43] M. A. Al-Hagery. "Model-based factors to extract quality Indications in software lines of code", International Journal of Computer Science & Information Technology (IJCSIT), vol. 3, issue 2, pp. 112-121, April 2011.