SOFTWARE CODE MAINTAINABILITY: A LITERATURE REVIEW

Berna Seref and Ozgur Tanriover

Department of Computer Engineering, Ankara University, Ankara, Turkey

ABSTRACT

Software Maintainability is one of the most important quality attributes. To increase quality of a software, to manage software more efficient and to decrease cost of the software, maintainability, maintainability estimation and maintainability evaluation models have been proposed. However, the practical use of these models in software engineering tools and practice remained little due to their limitations or threats to validity. In this paper, results of our Literature Review about maintainability models, maintainability metrics and maintainability estimation are presented. Aim of this paper is providing a baseline for further searches and serving the needs of developers and customers.

KEYWORDS

Maintainability Model, Maintainability Metric, Maintainability Estimation, Maintainability Prediction

1. INTRODUCTION

Every software system needs to be modified in order to meet requirements of customers, users and new technologies. Addition and deletion of codes and adopting the system to a new operational platform are examples of the modification operations. ISO/IEC 9126 [1] defines software maintainability as "the capability of the software product to be modified." Another definition [2] for software maintainability is as "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." On the other hand, maintainability also depends on the extend of use of software constructs/patterns, programming paradigms/languages, application frameworks, programming skill of developers, coding rules, design patterns etc.

It has been observed that maintenance effort in the software life cycle ranges from 65% to 75% of total software development time [3]. As the majority of the time is spent in maintenance phase, effort spent to increase maintainability effects software cost in a negative or positive way [4]. Hence, one of the most important aim of software engineers is developing maintainable software. Maintainability feature of a software increases quality of it. With maintainable software, it is easy to modify parts of the software, meet user or customer requirements in a shorter time and manage the software efficiently.

Maintainability Index (MI) estimation calculation is one of the well-known maintainability estimation techniques. However, MI estimation is still a problem. It is not clear that how MI

DOI : 10.5121/ijsea.2016.7305

increases or decreases. There is sometimes no explanation of MI variation. For example, in Figure 1 [5], two versions of the same code are shown. They have different MI values.

Sample 1 (*MI score of 71*)

```
public static String Sha1(String plainText)
{
    using (SHA1Managed sha1 = new SHA1Managed())
    {
        Byte[] text = Encoding.Unicode.GetBytes(plainText);
        Byte[] hashBytes = sha1.ComputeHash(text);
        return Convert.ToBase64String(hashBytes);
    }
}
```

Sample 2 (*MI score of 73*)

```
public static String Sha1(String plainText)
{
    Byte[] text, hashBytes;
    using (SHA1Managed sha1 = new SHA1Managed())
    {
        text = Encoding.Unicode.GetBytes(plainText);
        hashBytes = sha1.ComputeHash(text);
    }
    return Convert.ToBase64String(hashBytes);
}
```

Figure 1. Sample codes

In Figure. 1., while MI value of Sample 1 is 71, MI value of Sample 2 is 73. According to the MI values, Sample 2 is more maintainable. On the other hand, this result is arguable. Count of line of codes is smaller in Sample 1 and it is more readable but Sample 2 is more maintainable based on MI values. There is a discussion about these samples at [5]. Contributions about this discussion are listed below:

- An idea is that declaring all variables at the top (as in Sample 2) is reasonable but minimizing an identifier's lifespan affects maintainability in a positive way. The counter idea to it is that location of variables does not affect metrics.

- An idea counter to the sample 2. This idea supports that distance between the declaration of the variable and where it is used is reduced, variable span is reduced. As a result, maintainability is increased. However, given samples and results are opposite of this idea.

- An idea supports that MI difference between two samples are because of number of used operator. Sample 2 has fewer operators that increase maintainability.

As it is seen, MI prediction and its accuracy are not clear. There are more different ideas that support or not support usefulness of it on maintainability estimation. We think that Sample 1 is more maintainable for the reason that it is more readable, understandable, effective and has a fewer line of code.

Another discussion [6] is about MI values, their meanings and if they are dependent on technology. In this discussion, MI values and their meanings are listed below:

- 85 and more: good maintainability
- 65-85: moderate maintainability
- 65 and below: difficult to maintain

In this discussion [6], common idea is it is not a good idea to use these numbers on different technologies such as C# and Java. We agree with this common idea and we strongly believe that these numbers can not show degree of maintainability on different technologies and can not compare maintainability of them. For example, the same MI value of C and Java technologies can not show that they have the same maintainability. C and Java codes have different syntax and different terminology. While C code has pointers, Java code has generics. It is meaningless to compare these two different technologies by using MI values.

A lot of researches have been carried out about maintainability, maintainability models, maintainability metrics and maintainability prediction to increase maintainability of the software and as a result, increase quality of the software and decrease software effort and cost. In these researches; new models are proposed to predict maintainability of software at early stage of its life cycle, factors which have positive or negative impact on maintainability are identified and maintainability of software is measured. At these researches, new algorithms are proposed or hybrid methods are tried to get more accurate results.

In this study, Literature Review about maintainability models, maintainability metrics and maintainability estimation are presented to provide a baseline for further researches and to serve the needs of developers and customers.

This paper is structured as follows: Section 2 illustrates review methodology. Results are given in Section 3. Section 4 includes summary and discussions. Section 5 lists identified future works in the literature. Section 6 illustrates limitation of this review. Finally, review is concluded in Section 7.

2. METHOD

In this study, three stages which are defining research questions, designing the search strategy and selecting the studies are followed.

2.1. Research Questions

In this literature review, we are trying to answer following research questions:

RQ1: Which models are proposed to evaluate maintainability?

Aim of RQ1 is identifying proposed models that evaluate and measure maintainability.

RQ2: Which methods or algorithms are carried out to propose models? At which rate they are effective?

Aim of RQ2 is focusing on methods or algorithms that are used to propose a model and discuss effectiveness of this model.

RQ3: Which metrics and methods used for maintainability estimation?

Aim of RQ3 is extracting metrics used for maintainability estimation and focusing methods to estimate maintainability.

RQ4: Which metrics used to improve software maintainability?

Aim of RQ4 is researching usage of metrics to improve maintainability and effect of metrics on maintainability.

2.2. Search Strategy

To limit the search to the most relevant search term, the listed steps are followed:

1. Major distinct terms are extracted from our research questions.
2. Our search term is updated with keywords.
3. Relevant papers are selected with the condition of having all keywords.

In step 1, candidate search terms are defined: maintainability, maintainability models, maintainability evaluation, model effectiveness, maintainability estimation, maintainability metrics, maintainability methods, improvement of maintainability. Then, keywords of the search term are selected as "maintainability model", "maintainability metric" and "maintainability estimation." Different spelling is accepted such as "maintainability metrics", "model of system maintainability", "estimation of software maintainability" etc.

In step 3, papers which have all keywords in the same or different spelling are accepted for the search.

Four electronic databases are included to search our review studies: *IEEE – Xplore, Springer Link, Science – Direct,* and *Wiley Online Library*. Advanced search features of the each database are used to get relevant literature.

Our search includes the period 2005 to 2015. Searching is carried out for conference papers and journal papers separately for each database. Candidate papers are ordered from most relevant to least relevant.

2.3. Study Selection

After getting candidate papers, it is aimed to select papers from the top five candidate papers if they provide conditions to be selected. While selecting papers, for some searches, all of the top five candidate papers are selected because of providing selection condition. On the other hand, in some searches, we are not able to select any papers because of not providing selection condition. Inclusion and exclusion criteria to select papers are listed below:

Inclusion criteria:

- The paper must take place at the first five candidate papers which are ordered from the most relevant to the least relevant between the period of 2005 to 2015.

- The paper title must have search term. For example, in this study, search term is constructed with two words. The paper's title must include both of the words.

- Besides providing all conditions, abstract of the paper must be focused on "software" and searching keywords.

Exclusion criteria:

- Studies does not take part at the first five candidate papers that are proposed by searching engine.

- Studies does not have all keywords at their titles.

- Studies whose abstracts are not directly related to search area.

- Studies which are selected by us for this search before. (to prevent duplication)

After applying inclusion and exclusion criteria, relevant papers are narrowed to 20 and listed in Table I with the features of ID, Authors, Addressed Research Questions and References No.

Table 1. Selected studies

ID	Authors	Addressed Research Questions				Ref.
S1	I. Heitlager et al.	1	2			[7]
S2	F. Ye et al.	1	2		4	[8]
S3	O. Turetken	1	2			[9]
S4	A. Kaur et al.	1	2			[10]
S5	A. Kaur et al.	1	2		4	[11]
S6	M.A. Ahmed and H.A. Al-Jamimi	1	2	3		[12]
S7	A. Sheshasaayee and R. Jose	1	2		4	[13]
S8	S. MISRA	1	2		4	[14]
S9	K.T. Al-Sarayreh et al.	1	2			[15]
S10	H. Washizaki et al.			3	4	[16]
S11	L. Kumar et al.	1	2	3		[17]
S12	Y. Zhou and B. Xu			3	4	[18]
S13	A. Pratap et al.			3	4	[19]
S14	H.S. Chae et al.			3	4	[20]
S15	N. Yoshida et al.				4	[21]
S16	F. Zhang et al.				4	[22]
S17	S. Counsell et al.				4	[23]
S18	D. Baski and S. Misra				4	[24]
S19	J.M. Conejero et al.				4	[25]
S20	J. de A.G.Saraiva et al.				4	[26]

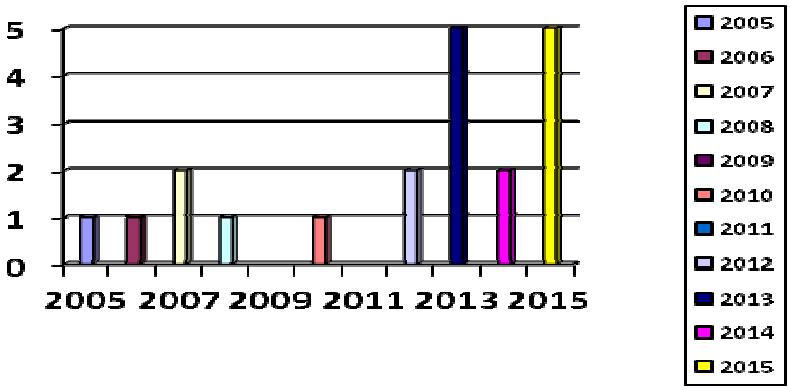

Figure 2. Distribution of selected papers

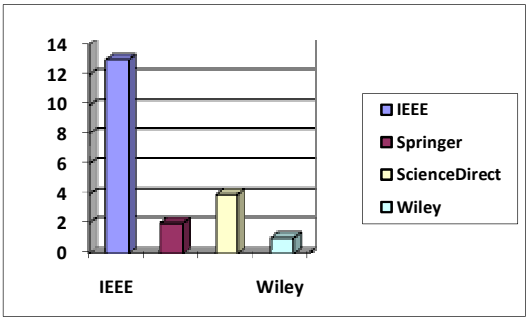

Figure 3. Distribution of selected studies based on electronic databases

We aimed to select up to date papers to be able to see the last level of maintainability researches. Figure 2 gives information about distribution of selected studies per year. Figure 3 gives information about number of selected studies based on electronic databases.

3. RESULTS

Firstly, it is observed that there are relatively few relevant studies matching our search terms between 2005 and 2012. When Figure 2 is analyzed, it seen that number of studies about the search term increases year by year and it becomes more active in 2013 and 2015. When Figure 3 is analyzed, it is observed that a large number of the selected studies is belong to *IEEE* while the smallest part is belong to *Wiley Online Library.*

In addition, results show that 35% of the selected studies (7) are journal while the remaining 65% of them are conference papers (13).

Review results for each research questions are given in the following subsections.

3.1. Proposed Maintainability Models

RQ1: Which models are proposed to evaluate maintainability?

I. Heitlager et al. (S1) discuss problems of Maintainability Index such as root cause analysis, average complexity, computability comment, understandability and control. Then, they propose a new model to measure maintainability. In this model, by using ISO 9126, source code measures are mapped onto the sub characteristics of maintainability and a relationship between source code metrics and ISO 9126 quality characteristics is provided.

F. Ye et al. (S2) propose a model based on multiple classifiers combination which has three parts: attribute selection, model training and model interpretation.

O. Turetken (S3) discusses applicability of a software maintenance model and choses Software Improvement Group Maintainability Model (SIG Model).

A. Kaur et al. (S4) discuss Oman and Hagemeister Maintainability Index model, show ineffectiveness of it. After that, propose two regressions models built by stepwise selection and backward elimination.

A. Kaur et al. (S5) propose "extensibility" as a sub character and evaluate aspect oriented software maintainability quality using Analytic Hierarchy Process (AHP).

A.A. Moataz and A.H. Al-Jamimi (S6) propose fuzzy logic-based transparent prediction model.
A. Sheshasaayee and R. Jose (S7) propose aspect oriented maintainability model which is based on static metrics for aspect oriented systems.

S. MISRA (S8) focuses on relationship between some software metrics and MI to increase maintainability of a system.

K.T. Al-Sarayreh et al. (S9) propose a reference model of Software FUR (Functional User Requirements) for identification of system maintainability requirements.

3.2. Methods/Algorithms and Effectiveness of Proposed Models

RQ2: Which methods or algorithms are carried out to propose models? At which rate they are effective?

S1 decreases problems of MI with proposed model.

S2 uses Genetic Algorithm to select attributes (metrics) and then train the system with Weka tool. After that, to interpret results of proposed model, uses rule extracting algorithm based on decision tree. Dataset for this study is 300 classes of open source C++ software system downloaded from internet. As a result, it is observed that the model trained by selected properties is faster than the model trained by all properties. In addition, this model has a benefit of enabling developers and researchers to select best maintainable object oriented software system.

S3 enables business process designers and practitioners to understand deficiencies and pinpoint parts of the possible improvement places and predict efficiency of maintenance tasks. Doesn't

taking into account cognitive physiology on the understandability of process models is one of the limitation of this model.

S4 uses different versions of Luceno search engine which are 2.0.0 and 2.1.0 as a dataset and calculates metrics of them by using CKJM and IntelliJ IDEA tool. Proposed regression models are built by using Stepwise Selection and Backward Elimination. As a result, it is observed that proposed models shows better performance when compared to Oman and Hagemeister (O&H) Maintainability Index (MI) model based on correlation coefficient and Change between models. Dataset used in this study is written in Java and it is seen as a limitation. As a future work, it is planned to replicate the study on other software.

S5 evaluates aspect oriented software maintainability quality using Analytic Hierarchy Process (AHP). The model consists of six attributes which are extensibility, reusability, modifiability, analyzability, testability and modularity. Value of pair wise relative weights for the characteristics are defined according to the survey that is carried out with 8 participants by taking mean of survey results. Then, Eigenvector is computed to get relative ranking of the attributes in relation to maintainability. As a result, it is observed that the order is: extensibility, reusability, modifiability, analyzability, testability and modularity. In addition, it is observed that this model can estimate maintainability of Aspect Oriented projects. In the future, it is planned to compare maintainability of different Aspect Oriented projects with the weights which are gotten in this study.

S6 predicts software maintainability using Mamdani fuzzy inference engine. QUES and UIMS are selected as datasets. The model is compared with T-S-based, SVM (Support Vector Machine), PNN (Probabilistic Neural Network), RBF (Radial Basis Function), BN (Bayesian Network) and MARS (Multivariate Adaptive Regression Splines) models. As a result, it is observed that Mamdani based model offers the best accuracy. Data scarcity is seen as a limitation for this study. In the future, it is planned to try this model on different datasets.

S7 proposes Aspect Oriented maintainability model which is based on WOM (Weighted Operations in Module), RFM (Response for a Module), CAE (Coupling on Advice Execution),

CDA (Crosscutting Degree of Aspect), CIM (Coupling on Intercepted Module), CFA (Coupling on Field Access), CBM (Coupling between Modules), LCCO (Lack of Cohesion in Operation) and LOCC (Lines of Class Code) metrics. In this study, aim is providing quality attributes to predict parameters like maintainability, changeability etc. On the other hand, this model is theoretical.

S8 focuses effect of 20 design/code measures on maintainability. These measures are extracted using Krakatau metric professional measurement tool from the 50 projects written in C++ language. As a result, it is observed that there is a strong dependence between MI and SLOC (Source Line of Code) metrics. There is a positive relationship between MI and AHF (Attribute Hiding Factor), AIF (Attribute Inheritance Factor), AVPATHS (Average Depth of Paths), COF (Coupling Factor), DIT (Depth of Inheritance Tree), MIF (Method Inheritance Factor) and RFC (Response for Classes) metrics. There is a strong effect of ACLOC (Average Class Size), AMLOC (Average Method Size), AVPATHS (Average Depth of Paths), CDENS (Control Density), COF (Coupling Factor), DIT (Depth of Inheritance Tree), n (Program Vocabulary), N (Program Length), PPPC (Percentage Public/Protected Members), and WMC (Weighted Methods

in Classes) metrics on MI. However, the results are gotten using 50 systems so results are not universally general for all systems. In the future, it is planned to carry out further theoretical and experimental studies with the aim of validating the obtained results in this study.

S9 aims to provide a model to identify system maintainability requirements and proposes Software FUR (Functional User Requirements) model to achieve this aim. As a result, with this study, budget overruns and missed deadlines are prevented, input requirements are determined earlier in the project life cycle. However, in this study, only software requirements are allocated, and the study is not expanded to hardware and manual system requirements.

3.3. Metrics and Methods for Maintainability Estimation

RQ3: Which metrics and methods used for maintainability estimation?

H. Washizaki (S10) proposes CCOF (Component Coupling Factor) metric which takes into consideration of characteristics of remote components to measure coupling-based complexity of Component Based Software (CBS) system. As a result, it is observed that CCOF is a useful and effective index for maintainability. In the future, it is planned to expand number of samples.
L. Kumar et al. (S11) predict maintainability of Quality Evaluation System and User Interface System using Neuro Genetic Algorithm. Genetic Algorithm is used to find optimum weights in the Neural Network. Metrics are inputs for the network that will be trained. Mean Absolute Error (MAE), Mean Absolute Relative Error (MARE), Root Mean Square Error (RMSE) and Standard Error of the Mean (SEM) are performance evaluation criterions. As a result, it is observed that this method shows better performance when compared to the last studies. On the other hand, this model is created using object oriented systems so the model is likely to be valid for the systems which are developed using object oriented programming languages. In the future, it is planned to entegrate other techniques such as Particle Swarm Optimization, Fuzzy Logic, Clonal Selection Algorithm to the network to increase accuracy rate of estimation.

Z.Yuming and X. Baowen (S12) study relationship between a number of metrics and maintainability, and prediction ability of these metrics for software maintainability. 148 Java open source software collected from the websites http://sourceforge.net/ and http://java-source.net

and used as dataset. 15 design metrics are extracted from the dataset and prediction ability of these metrics are reported when used together. As a result, it is seen that average control flow complexity per method (OSAVG) is the most important maintainability factor while cohesion and coupling are less. In addition, multivariate prediction model is showed a good accuracy.

A. Pratap et al. (S13) predict software maintainability using Fuzzy Logic on Matlab Platform. In this study, while Adaptability (AD), Complexity (CLX), Understandability (USD), Documentation Quality (DocQ) and Readability (RD) metrics are used as input, maintainability is used as an output. Triangular Membership Function, Trapezoidal Membership Function and Gaussian Membership function are chosen membership functions. As a result, it is observed that this model can be used to predict software maintainability. It is advised that to increase maintainability of the system, Adaptability (AD), Understandability (USD), Documentation Quality (DocQ) and Readability (RD) metrics should be high while Complexity (CLX) should be low. As a future work, it is planned to propose new metrics and new approaches for software maintainability.

H.S. Chae et al. (S14) study relationship between LOC (Line of Code), LCOM (Lack of Cohesion in Methods), RFC (Response for Classes), DAC (Data Abstraction Coupling) metrics and software maintenance effort. Dataset is constructed with 4 developers by recording the maintenance time needed for each class of maintenance action. Correlation analysis is carried out to observe relationship between the selected metrics and maintenance effort. As a result, it is seen that selected metrics can not show a good performance for Web-based applications to predict maintenance effort. In the future, it is planned to propose new metrics that take into account characteristics of design patterns of Web based applications.

3.4. Metrics to Improve Software Maintainability

RQ4: Which metrics used to improve software maintainability?

N. Yoshida et al. (S15) propose dividing source code into functional segments using cohesion metric to improve software maintainability. As a result, start and end points of each functional segment are defined; understandability of the source code is increased. In this study, $NCOCP_2$ (Normalized with number of LCOM (Lack of Cohesion in Methods)) metric is calculated and a threshold value is set to $NCOCP_2$. Then, functional segments are extracted based on this cohesion metric. In the future, it is planned to validate the cohesion metric theoretically and propose a method which add feature names to extracted functionalities.

F. Zhang et al. (S16) focus the effects of six context factors on distribution of software maintainability metrics. These context factors are application domain, programing language, age, life span, number of changes and number of downloads. As a dataset, 320 nontrivial software systems from Source Forge is used. Kruskal Wallis test, Mann-Whitney U test and Cliff's δ effect size are selected as statistical methods to analyze 320 software systems. As a result, it is observed that all context factors affect 39 calculated maintainability metrics. But the most are: application domain, programming language and number of changes. In the future, it is planned to use more software system from SourceForge, GoogleCode, and GitHub.

S. Counsell et al. (S17) study relationship between MI and coupling, defects and size features. Three releases of two Eclipse projects are used as datasets. Class-based metrics are extracted with JHawk tool. As a result, it is observed that there is a relationship between the features and MI.
D. Baski and S. Misra (S18) propose data weight of a web service description language, distinct message ratio metric, message entropy metric and message repetition scale metric to evaluate maintainability of XML Web Service. All metrics are evaluated using Weyuker's properties and Caner's framework. As a result, usefulness of the metrics are proved. In the future, it is planned to develop an automated tool to compute metrics and make researches to assign right values for upper and lower boundaries of the complexity values for proposed metrics.

J.M. Conejero et al. (S19) try to find an answer for the question that if a certain crosscutting characteristics affect software maintainability. Correlation between crosscutting properties and changeability and stability attributes is focused. As a result, it is observed that certain crosscutting properties affect changeability and stability in a negative way. In the future, it is planned to carry out some empirical studies to compare obtained results.

J. de A.G. Saraiva et al. (S20) propose metrics' categorization to define which metrics can be used in the experiments to increase rate of maintainability. 7 categories and 17 subcategories are presented. These are tested using Wilcoxon Test and survey which includes 47 expert opinion

about the proposed catalog. As a result, it is observed that proposed approach is useful and has a positive effect on maintainability. In the future, it is planned to improve catalog generalization and entegrate GQM (Goal-Question Metric) Model to the study.

4. SUMMARY AND DISCUSSIONS

To achieve maintainability evaluation and estimation, in general, databases and software metrics are needed. Databases are obtained in two ways: constructing a new databases for the study or using ready databases. Databases and tools to extract metrics used in the literature are listed in Table II [8,10,14,17,18,22,23].

Table 2. Databases and tools

Database	**Tools**
300 classes of open source C++ software system downloaded from internet	Weka
Luceno search engine	CKJM and IntelliJ IDEA tool
User Interface Management System (UIMS) contains 39 classes and Quality Evaluation System (QUES) contains 71 classes	-
50 projects written in C++ language downloaded from several websites	static analysis tool
148 Java open source software collected from the websites http://sourceforge.net/ and http://java-source.net	-
320 nontrivial software systems from Source Forge	https://bitbucket.org/serap/contextstudy http://www.scitools.com
Three releases of two Eclipse projects	JHawk tool

In Table 2, it is observed that projects that is used as dataset are written in Java or C++. It can be said that researchers preferred using Java or C++ projects instead of C# or C projects.
As it is seen in the literature, some studies are focused on problems of MI and tried to decrease these problems with proposing new models and methods and analyzing relationship between MI and software metrics.

MI is a measure that define how maintainable a system is. Higher values of MI means more maintainable system. MI was introduced in 1992 by Paul Oman and Jack Hagemeister and presented at the International Conference on Software Maintenance ICSM 1992 and formulated in Equation (1) [27].

$$171-5.2In(HV)-0.23CC-16.2In(LOC)+50.0sin\sqrt{2.46*COM} \quad (1)$$

In Equation (1), *HV* is Halstead's Volume, *CC* is McCabe's cyclomatic complexity, *LOC* is line of code, *COM* is percentage of comments.

Problems of MI can be listed as follows:

- It is not known that why *sin* operation is used, why *CC* is multiplied with 0.23 value or how we can get *171*. There are some understandability questions about MI formula but there is no clear explanation.
- In MI Formula, *COM* takes place. However, it is one of the arguable points if comments must take part in the formula.
- MI Formula can mask the presence of high-risk parts especially for object oriented systems because of power low distribution of complexity and as a result, giving low average complexity. It is not advised to use MI Formula as a maintainability measure for object oriented systems.
- *HV, CC, LOC* and *COM* are used in MI Formula according to the statistical correlation. However, there is no clear explanation and strong evidence that there is a causal relation between the metrics. Because of it, it is hard to tune maintainability index by changing used metrics in the Formula.
-

To increase performance of MI, some metrics are proposed and relationship between MI and metrics are analyzed. Relationship between MI and metrics are shown in Table 3 [14,23].

Table 3. Relationship between MI and software metrics

Metric	**Relationship**
AMLOC (Average method size)	non-linear correlation
SLOC (Source lines of code)	non-linear correlation
AHF (Attribute hiding)	positive relationship
AIF (Attribute inheritance Factor)	positive relationship
AVPATHS (Average depth of paths)	positive relationship
COF (Coupling)	positive relationship
DIT (Depth of inheritance tree)	positive relationship
MIF (Method inheritance)	positive relationship
RFC (Response for class)	positive relationship
FIN (Number of incoming couplings)	Correlation is not significant at the 5% level or below according to Spearman's and Pearson's coefficient
CBO (Coupling between objects)	Correlation is significant at the 1% or 5% level according to Spearman's and Pearson's coefficient
NOS (Number of java statements in a class)	Correlation is significant at the 1% or 5% level according to Spearman's and Pearson's coefficient

As it is seen in Table 3, there is a non-linear correlation between MI and AMLOC and SLOC. When AMLOC is increased, method size increases, method becomes more complex and rate of readability decreases, as a result, maintainability decreases. When SLOC is increased, program

size and complexity increases as a result, maintainability decreases. According to the coefficient of correlation analysis, when attribute hiding, attribute inheritance, method inheritance, average depth of paths, coupling, depth of inheritance tree and response for class are increased, maintainability increases. In addition, according to the experiment which is carried on with two releases of two projects, correlation rate between MI and CBO and NOS are significant at the 1% or 5% level according to Spearman's and Pearson's coefficient but not significant at the 5% level or below for FIN.

There are some metrics used to predict maintainability of the systems. These metrics, systems and performance of the metrics are listed in Table 4 [12,16,18-20].

Table 4. Metrics for maintainability estimation

Metric	**System**	**Performance**
CCOF(Component coupling factor)	Component Based System (CBS)	useful index for maintainability, nonredundant with existing metrics
DIT (Depth of the inheritance tree)	UIMS dataset QUES dataset Both dataset (Merging UIMS and QUES Dataset)	Strong Pearson's correlation coefficient (-0.43) with dependent variable CHANGE (counting the number of lines in the code which has been changed during a 3-year maintenance period) for UIMS Dataset.
NOC (Number of children)	UIMS dataset QUES dataset Both dataset (Merging UIMS and QUES Dataset)	Strong Pearson's correlation coefficient (0.56) with dependent variable CHANGE (counting the number of lines in the code which has been changed during a 3-year maintenance period) for UIMS Dataset.
MPC (Message-passing coupling)	QUES dataset UIMS dataset Both dataset (Merging UIMS and QUES Dataset)	Strong Pearson's correlation coefficient (0.46) with dependent variable CHANGE (counting the number of lines in the code which has been changed during a 3-year maintenance period) for QUES Dataset.
RFC (Response for a class)	UIMS dataset QUES dataset Both dataset (Merging UIMS and QUES Dataset)	Strong Pearson's correlation coefficient (0.64) with dependent variable CHANGE (counting the number of lines in the code which has been changed during a 3-year maintenance period) for UIMS Dataset.

LCOM (Lack of cohesion of methods)	UIMS dataset QUES dataset Both dataset (Merging UIMS and QUES Dataset)	Strong Pearson's correlation coefficient (0.57) with dependent variable CHANGE (counting the number of lines in the code which has been changed during a 3-year maintenance period) for UIMS Dataset.
DAC (Data abstraction coupling)	UIMS dataset QUES dataset Both dataset (Merging UIMS and QUES Dataset)	Strong Pearson's correlation coefficient (0.63) with dependent variable CHANGE (counting the number of lines in the code which has been changed during a 3-year maintenance period) for UIMS Dataset.
WMC (Weighted method per class)	Both dataset (Merging UIMS and QUES Dataset) UIMS dataset QUES dataset	Strong Pearson's correlation coefficient (0.67) with dependent variable CHANGE (counting the number of lines in the code which has been changed during a 3-year maintenance period) for both datasets.
NOM (Number of methods)	UIMS dataset QUES dataset Both dataset (Merging UIMS and QUES Dataset)	Strong Pearson's correlation coefficient (0.64) with dependent variable CHANGE (counting the number of lines in the code which has been changed during a 3-year maintenance period) for UIMS Dataset.
SIZE1 (Lines of code)	Both dataset (Merging UIMS and QUES Dataset) UIMS dataset QUES dataset	Strong Pearson's correlation coefficient (0.65) with dependent variable CHANGE (counting the number of lines in the code which has been changed during a 3-year maintenance period) for both dataset.
SIZE2 (Number of properties)	UIMS dataset QUES dataset Both dataset (Merging UIMS and QUES Dataset)	Strong Pearson's correlation coefficient (0.67) with dependent variable CHANGE (counting the number of lines in the code which has been changed during a 3-year maintenance period) for UIMS dataset
OSAVG (Average	148 Java open	OSAVG is the most predictive

complexity per method) CSO (Average number of methods per class) CSA (Average number of attributes per class) SNOC (Average number of children per class)	source software	metric for maintainability based on multivariate linear regression model. CSO and CSA follows it.
AD (Adaptability) CLX (Complexity) USD (Understandability) DocQ (Documentation Quality) RD (Readability)	Some values are assumed for the metrics	When these metrics are used as input and maintainability is used as output, maintainability can be predicted based on fuzzy logic. AD, USD, DocQ and RD should be high whereas CLX should be low to improve maintainability.
RFC (Response for classes) LCOM (Lack of cohesion in Methods) DAC (Data abstraction coupling) LOC (Line of code)	Web-based applications	These metrics are not able to relate with maintenance for Web-based applications.

In Table 4, metrics used for maintainability estimation, where they are used and their prediction performance are listed. It is observed that, some metrics affect maintainability in a positive way whereas some of them affect negative. In addition, according to this table, it is important to choose right metrics for the system. For example, in the table, there are some metrics used for maintainability estimation of Web-based applications but these metrics fail to predict.

Some approaches are tried to improve maintainability of the systems such as using metrics, analyzing effect of context factors on metrics etc. These approaches are listed with aim of them and results in Table 5 [21-22].

Table 5. Approaches to improve maintainability

Approach	**Aim**	**Result**
NCOCP2 (Normalized with Number of Lock of Cohesion in Methods) cohesion metric is used.	This metric is used to divide source code into functional segments with the aim of identifying start and end points of each functional segments.	Functional segments are identified. Understandability of source code is increased, as a result, maintainability is increased.
Effect of context factors (application domain, programming language, age, life span, number of changes, number of	Effect of six context factors on software maintainability metrics is analysed to improve maintainability.	It is observed that, application domain, programming language and number of changes effect software maintainability

downloads) on distribution of software maintainability metrics are analysed		metrics mostly.

To predict or evaluate maintainability of a system a lot of methods and correlation analysis are used. Fuzzy Logic, Mamdani Fuzzy Inference Engine, Genetic Algorithm, regression models built by stepwise selection and backward elimination, classical linear regressions, Kruskal Wallis test, Mann-Whitney U test, Cliff's δ effect size and statistical methods are one of the most used methods to predict or evaluate maintainability of a system.

Maintainability is an important point for software architecture and software design decisions. A lot of studies are carried out about maintainability of software architecture however; they are not able to analyze the optimal maintainability of a software architecture very well [28]. Design takes part in the software architecture development process. Architectural design decisions are taken at the beginning of project and these decisions affect software maintainability in a significant rate. For example; merging two components can affect maintainability in a negative way [29]. In the future, it is needed to focus on effect of design decisions on maintainability and assess maintainability to the software architecture to improve performance of the systems.

5. FUTURE WORKS IDENTIFIED IN REVIEWED PAPERS

Future works identified in reviewed papers are listed below:

- S1 aims to investigate if their rating schemas could be captured using Bayesian Belief Nets (BBN) and incorporate ISO 25000 series (SQuaRE) with their proposed maintainability model.
- S2 aims to take consideration of other object-oriented metrics and increase number of projects.
- S3 aims to take consideration of the role of cognitive physiology on the understandability of the models.
- S4 aims to apply the proposed model on other software and validate the model.
- S5 aims to compare maintainability of various Aspect Oriented projects using obtained weights in the study.
- S6 aims to take consideration of uncertainty which means while some internal attributes affect some projects in a positive or negative way, these attributes can not affect all of the projects.
- S8 aims to study the effects of several factors that analysis methods used in this study depend on. In addition, it is planned to validate the results obtained in this study by using further theoretical and experimental studies.
- S9 aims to evaluate proposed approach using a number of case studies.
- S10 aims to verify the experimental result and modify the component specification step of CCOF metric.
- S11 aims to entegrate other techniques such as Particle Swarm Optimization, Fuzzy Logic and Clonal Selection Algorithm to the neural network to increase accuracy rate of estimation.
- S13 aims to add new metrics to the system for software maintainability.

- S14 aims to apply the study to other web based applications and propose new metrics for Web-based applications.
- S15 aims to validate the cohesion metric theoretically and propose a method which add feature names to extracted functionalities.
- S16 aims to use more software system from SourceForge, GoogleCode and GitHub and derive the thresholds and ranges of metric values according to the results obtained in the study.
- S18 aims to develop an automated tool to compute metrics and make researches to assign right values for upper and lower boundaries of the complexity values for proposed metrics.
- S19 aims to carry out some empirical studies to compare obtained results.
- S20 aims to improve catalog generalization and entegrate GQM (Goal-Question Metric) Model to the study.

6. LIMITATIONS OF THIS REVIEW

In this study, 4 keywords are searched using 4 search engines. However, number of keywords and search engines can be increased. Also, different synonyms can be tried to increase number of papers. For example, in this study, "maintainability estimation" keyword is used to get papers which are about maintainability estimation. On the other hand, with this keyword, we are not able to get the papers with title "maintainability prediction" even if they have the same content. Deep searches can be carried out in the future.

7. CONCLUSION AND FUTURE WORK

In this study, results of a Literature Review about maintainability models, maintainability metrics and maintainability estimation are presented to provide a baseline for further researches and to serve the needs of developers and customers.

As a future work, it is planned to increase number of search engine and number of keywords to get more relevant journals and conferences, focus on effect of design decisions on maintainability and assess maintainability to the software architecture to improve maintainability.

REFERENCES

[1] ISO/IEC, ISO/IEC 9126. Software Engineering – Product quality 6.5.ISO/IEC, 2001.
[2] IEEE Std. 610.12-1990. 1993. Standard Glossary of Software Engineering Terminology, IEEE Computer Society Press, Los Alamitos, CA, 1993.
[3] S. Muthanna, K. Konotogiannis, K. Ponnambalam, and B. Stacey, "A maintainability model for industrial software systems using design level metrics," In Seventh Working Conference on Reverse Engineering, pages 248–256, November 2000.
[4] B. Kumar, "A Survey of Key Factors Affecting Software Maintainability," International Conference on Computing Sciences, DOI 10.1109/ICCS.2012.5.
[5] http://stackoverflow.com/questions/2749082/why-does-this-maintainability-index-increase (16.01.2016)
[6] http://stackoverflow.com/questions/592866/maintainability-index (16.01.2016)
[7] I. Heitlager, T. Kuipers and J. Visser,"A Practical Model for Measuring Maintainability", Sixth International Conference on the Quality of Information and Communications Technology, pp. 30-39.

[8] F.Ye, X. Zhu, Y. Wang, "A New Software Maintainability Evaluation Model Based on Multiple Classifiers Combination," 2013 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering (QR2MSE), pp. 1588-1591.

[9] O. Turetken, "Towards a maintainability model for business processes: Adapting a software maintainability model (position paper)," 2013 IEEE 1st International Workshop on Communicating Business Process and Software Models Quality, Understandability, and Maintainability (CPSM), pp. 1 - 4.

[10] A. Kaur, K. Kaur, K. Pathak, "A proposed new model for maintainability index of open source software," 2014 3rd International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), pp. 1 - 6.

[11] A. Kaur, P.S. Grover, A. Dixit, "Quantitative evaluation of proposed maintainability model using AHP method," 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)", pp. 1367 – 1371, 2015.

[12] M.A. Ahmed, H.A. Al-Jamimi, "Machine learning approaches for predicting software maintainability: a fuzzy-based transparent model," Software, IET, Volume: 7, issue: 6, pp. 317 - 326, DOI: 10.1049/iet-sen.2013.0046.

[13] A. Sheshasaayee, R. Jose, "A Theoretical Framework for the Maintainability Model of Aspect Oriented Systems", The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015), Procedia Computer Science 62 (2015) 505 – 512.

[14] S. MISRA, "Modeling Design/Coding Factors That Drive Maintainability of Software Systems," Software Quality Journal, 13, 297–320, 2005.

[15] K. T. Al-Sarayreh, A. Abran and J. J. Cuadrado-Gallego, "A standards-based model of system maintainability requirements," J. Softw.: Evol. and Proc. 2013; 25:459–505.

[16] H.Washizaki, T. Nakagawa, Y. Saito, Y. Fukazawa, "A Coupling-based Complexity Metric for Remote Component-based Software Systems Toward Maintainability Estimation," 13th Asia Pacific Software Engineering Conference (ASPEC 2006), pp. 79 - 86.

[17] L. Kumar, D. Kumar Naik, S. Ku. Rath, "Validating the Effectiveness of Object-Oriented Metrics for Predicting Maintainability," Third International Conference on Recent Trends in Computing (ICRTC' 2015), Procedia Computer Science 57 (2015) 798 – 806.

[18] Y. Zhou and B. Xu, "Predicting the Maintainability of Open Source Software Using Design Metrics," Wuhan University Journal of Natural Sciences, Vol.13 No.1, 014-020.

[19] A. Pratap, R. Chaudhary, K. Yadav, "Estimation of software maintainability using fuzzy logic technique," 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), pp. 486 - 492.

[20] H.S. Chae, T.Y. Kim, W.S. Jung, J.S. Lee, "Using Metrics for Estimating Maintainability of Web Applications: An Empirical Study," pp. 1053 - 1059.

[21] N. Yoshida, M. Kinoshita, H. Iida, "A cohesion metric approach to dividing source code into functional segments to improve maintainability," 2012 16th European Conference on Software Maintenance and Reengineering (CSMR), Pages: 365 - 370.

[22] F. Zhang, A. Mockus, Y. Zou; F. Khomh, A.E. Hassan, "How Does Context Affect the Distribution of Software Maintainability Metrics?," 2013 29th IEEE International Conference on Software Maintenance (ICSM), pp. 350 - 359.

[23] S. Counsell X. Liu, S. Eldh, R. Tonelli, M. Marchesi, G. Concas, A. Murgia, "Re-visiting the 'Maintainability Index' Metric from an Object-Oriented Perspective," 2015 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 84 - 87.

[24] D. Baski and S. Misra, "Metrics suite for maintainability of eXtensible Markup Language web services," Software, IET, Volume: 5, issue: 3, pp. 320 - 341.

[25] J. M. Conejero, E. Figueiredo, A. Garcia, J. Hernandez, E. Jurado, "On the relationship of concern metrics and requirements maintainability," Information and Software Technology 54 (2012) 212–238.

[26] J.A.G. Saraiva, M.S. de França, S. C.B. Soares, F. J.C.L. Filho, R. M.C.R. de Souza, "Classifying metrics for assessing Object-Oriented Software Maintainability: A family of metrics'catalogs," The Journal of Systems and Software 103 (2015) 85–101.

[27] http://avandeursen.com/2014/08/29/think-twice-before-using-the-maintainability-index/ (26.12.2015)

[28] http://www.janbosch.com/Articles/OptimalSAMaintainabilityCSMR01.pdf (27.15.2015)

[29] F. Oquendo, B. Warboys, R. Morrison, Software Architecture: First European Workshop, EWSA 2004, St Andrews, UK, May 2004, Proceedings, B. Graaf, "Maintainability through Architecture Development" pp:206 -2010.

Authors

Berna Seref received the B.S. degree in computer engineering from Anadolu University, Eskisehir, Turkey, in 2012 and the M.S. degree in computer engineering from Kirikkale University, Kirikkale, Turkey, in 2015. She is currently pursuing the Ph.D. degree in computer engineering at Ankara University, Ankara, Turkey and working as a research assistant.

Ozgur Tanriover received the M.S. and PhD. degrees in information systems from Middle East Technical University, Ankara, Turkey in 2002 and in 2008 respectively. Since 2012, he has been working as an assistant professor in computer engineering department at Ankara University.