Software Maintainability Measurement

Akshay Kumar Jilla
9508109015
akji16@student.bth.se

Sai Priyatham
940617-3337
sado16@student.bth.se

Kota Maheshwar
9212199070
mako16@student.bth.se

Saikumar Bodicherla
950710-0552
sabo16@student.bth.se

***Abstract*— The purpose of this paper is to perform a review on software maintainability measurement. This exercise consists of a review of published works on metrics and models for software maintainability and description of open source tools that can be used with source code to predict/measure maintainability.**

Keywords- Software maintainability; Software Metrics; Measurement; Software quality; Software prediction.

I. INTRODUCTION

Software maintenance is considered as one of the most crucial stages in the evolution of software engineering. Software maintainability can be defined as to how comfortably the software system adapts to the changes made in the environment and to the faults corrected [1]. From the past decades, maintenance of the software in the software industry has been the most tiresome and largest financial draining process because the code which contains the errors must be corrected and updated which is difficult in real time [2]. To measure the maintenance of software some metrics have been proposed called the Software maintainability metrics. Software metrics are the valuable indicators to quantitatively measure the software designs [2]. Metrics also help to identify the areas of the code where testing should be performed and where the errors exist [2].

This particular paper is divided into two parts. In the first part, we will sort out the different maintainability metrics, maintainability predictors, models used for predicting maintainability and the systems studied in maintainability from ten different Peer reviewed journals and conference papers are summarized into different sections.

Later in the second part two open source java tools are selected and briefly described. Comparison of the tools is performed and a comparative report on its advantages and disadvantages is presented.

II. SUMMARIES

- Article [1]:

  In the first article, the performance of fuzzy interface systems in predicting the software maintainability by anticipating the software changes in the future using two heterogeneous databases is evaluated. The experiment was performed using two datasets which are Quality Evaluation Systems (QUES) and User Interface Management Systems (UIMS). In the experiments conducted Mamdani-based model was used and each dataset was divided into training set and validation set in the ratio 2:1. The experiments were repeated many times to get accurate results.

  The mean value of the models accuracy of the results obtained from 10 different test subsets for both the datasets is presented using a figure. In the experiment performed using QUES dataset each run in the dataset consists of 40 epochs for training and testing.

  The author mentioned that the values of the prediction accuracy measures using Mamdani-based model played a significant role in predicting software maintainability for both the datasets.

- Article [2]:

  In the second article, the author described different software metrics. The use of specific implementation tools and techniques to decrease the effort and cost of maintenance in large-scale software projects are scrutinized. This methodology was applied in a commercial environment where over 6000 procedures of a specific project code were analyzed. The statistical correlations of metrics were calculated based on error data corresponding to the software system. This was used to validate the results.

  The correlations obtained indicates that the code metrics used were successful in detecting the errors in the source code. This proved the fact that software quality metrics play an important role in identifying

error-prone areas in source code where the level of complexity is very high.

- Article [3]:

  In the third article, empirical examination of the relationship of class level object oriented metrics is calculated along with maintainability as the parameter. Here, Neuro-Genetic algorithm was used as an approach for estimating the maintainability. To analyze the effectiveness of the metrics, they have been categorized into different groups (A1, A2, A3, and A4). QUES with 10-fold cross-validation and UIMS with 5-fold cross validation are the two software systems proposed for comparing the models.

  The effort estimation was calculated based on the median values of the hidden nodes in their respective folds. Graphical representation was given for the number of hidden nodes in each fold for both the software systems.
  The analysis concludes that Neuro-GA approach obtained better results when compared with the other approaches.

- Article [4]:

  This article performs a case study on a hypothesis that Dynamic metrics perform better than Static metrics in maintainability prediction. Required literature support was provided for supporting the hypothesis.

  The methodology was applied on an open source software system called "Hoduku". Static and dynamic metrics were collected from the same source code of this system. The collected metrics were tabulated neatly along with their descriptions. Machine learning techniques were applied to both metrics and the results are presented in a table.
  By observing and analyzing the empirical values of the result, the author concluded stating that Dynamic metrics were easy in predicting the maintainability when compared with Static metrics.

- Article [5]:

  This specific paper focuses on the software product health status as to finding a way to know the possible rate of the software product being deteriorated. Maintenance in this particular paper was categorized into different sections such as corrective maintenance, adaptive maintenance, perfective maintenance and preventive maintenance. These were used as key factors affecting software maintainability. Tabular forms were provided for the result analysis of the metrics implemented and percentage of distribution for types of maintenance.

  HMM algorithm was applied for calculating the software maintainability and the different states of the calculation were elaborated. This was successful in the course.

- Article [6]:

  This article emphasizes to propose a new generic method called the reference procedural method for maintaining the software products. Three metadata models for maintainability requirements using ISO standards were also proposed. Maintainability in this particular context was divided into different sections such as analyzability, changeability, stability, testability.

  Pictographic representations were given for Generic of FSM model for different maintainability categories.

  Formulas for the total functional size internally and externally for all the above-mentioned maintainability sections were provided.

- Article [7]:

  In this article, a new automate tool was proposed called the SPMLerner was proposed for predicting the maintainability of future software releases and effectiveness of SPMLerner on the amount of training data. SPMLerner uses 4 level hierarchical code metrics which consists of 44 metrics in total. It has been evaluated using 24- machine learning algorithms on a large dataset.

  Tabular representations were given for metrics at different levels of maintainability such as application level, class level and file level. The correlation between these metrics and average maintenance effort was also tabulated. SRCC values between actual and predicted maintainability of 24 machine-learning based models were also presented using a table.

  The models obtained through SPMLerner were more accurate in predicting the maintainability of future software releases when compared with other traditional measures.

- Article [8]:

  The main focus of this article was to conduct analyzes on how metrics based software maintainability models used in 11 industrial software systems help to make decisions in software process?

  Comparative analysis was performed on five maintainability models. From these two models were selected called the Hierarchical multidimensional assessment (HPMAS) and Polynomial regression model. Each of these models was applied to the industrial systems provided by Hewlett Packard and Defense Department contractors.

The results of the comparison of these models prior and post to their application were neatly tabulated. Feedback was taken from the maintainability engineers for assessing the results. The results showed that the maintainability analysis for the system after the implementation of both the models had a positive impact. Based on the statistics, the author stated that prior to the application of polynomial regression model roughly 50 percent of the system fall below the quality-cutoff index and was difficult to modify and maintain.

- Article [9]:

The ninth article focuses on the evaluation of effects of code change by using software maintainability metrics in large industrial software systems.

Maintenance was divided into four activities namely unused code removal, compiler warning removal, code restructuring and integrating new features. The Maintainability Index software metric was calculated using a four metric polynomial model and was applied on each of the four activities. The results for prior and post application of the metric were compared and tabulated neatly.

The author stated that in two of the analyzes made MI fails as a single measure of maintainability whereas the other two studies stated that MI can be used to gauge the impacts of rebuilding code and adding new elements to code. This suggests that a good maintainability assessment tool should not only give a shortsighted record of maintainability but also provide the information important to translate and comprehend that file by means of crude measures and significant diagnostics.

- Article [10]:

The purpose of this article is to develop an automated maintainability model using design level metrics. This is divided into four stages. The primary stage is to evaluate and select a set of metrics which show desired properties such as low connection with one another, do not rely upon any subjective appraisal provided by the maintainers and relate with various design properties of the framework.

The second stage points on the gathering of subjective maintainability information from the product engineers. This gathering of information will frame the center against which the metric values will be contrasted to build up the prediction model.

The third step includes the determination of the metrics which are the strongest maintainability predictors. This can be accomplished by analyzing the relationship between each metric and the subjective appraisals gave by the product engineers. The process of the fourth step is first to decide a single maintainability prediction value by joining a minimal set of metrics. This maintainability prediction value is called as the Maintainability Index.

Based on the observations, the author stated in the conclusion that this model may not be perfect in all situations but it illustrates the utility of growing such models for specific application domains.

III. MAINTAINABILITY PREDICTORS

Maintainability predictors are the means through which estimations are made for the maintenance process to predict the measurable value of the quality attributes of the Software product [11]. A software system prediction model helps the organization to predict the maintainability of their software systems [1].

Article ID	**Types of metrics**	**Names of metrics in each category**
1.	Design level Complexity Metrics Source code Metrics	McCabe's Complexity, Regularity metric, Halstead Effort metric. Source Code Metrics: - DIT, NOC, RFC, DAC , LCOM
2.	Design Level, Complexity Metrics, Code Metrics	McCabe's Cyclomatic complexity, Halstead Software, Fan-in & Fan-out, Lines of code, Effort.
3.	Source code metrics Complexity Metrics	NOC, DIT, WMC, CBO, RFC.
4.	Structural Metrics, Design Metrics, Complexity Metrics	LOC added, Modified & Deleted. IC, CBM, LOC, NPM & DAM
5.	Structural & Design Level Complexity Metrics	LOC, Cyclomatic Complexity and Coupling Between Objects,
7.	Design Level, Complexity Metrics	ELOC, Number of Comments.
9.	Structural & Complexity Metrics	Halstead's complexity(volume), Lines of code, percent of comment lines and Extended version of McCabe's Cyclomatic Complexity.
10.	Design Level, Complexity Metrics	McCabe's Cyclomatic complexity, McClure's cyclomatic complexity, Fan-out, Data complexity metric and Design Stability metric and other 18 types

Table1: Maintainability predictors used in Articles

IV. MAINTAINABILITY MEASUREMENT

Maintainability measurement is a process of measuring the quality attribute of the software using some metrics called as software metrics. Software quality metrics are divided into three types namely code metrics, structure metrics and hybrid metrics [2].

Metric Name	Metric Definition	Article Used it
BKLOC	Bugs in 1k lines of code	[5]
MI	Maintainability Index	[7],[8],[9]
Chidamber and Kemerer metrics	DIT, NOC, RFC, LCOM and WMC	[1], [3]
Li and Henry metrics	MPC, DAC, NOM, SIZE1, SIZE2	[1]
CHANGE	No of lines changed in a class	[1],[3]

Table 2: Metrics that are empirically proven in the article

V. MODELS/TECHNICHES FOR PREDICTING MAINTAINABILITY

Here, we are going to describe the different models and techniques used in the ten articles. They are Regression models, Halstead Models, Bayesian Network Model, a Hierarchical Multidimensional Assessment Model (HPMAS), four metric polynomial maintainability assessment model, Hidden Markov Model (HMM), Static code analysis models, SMPLerner, Reference procedural method, FSM model, Neuro-genetic algorithm, and Mamdani-based model.

Article ID	Model Summary in Steps
1.	Comparative analysis of fuzzy, Neuro fuzzy and fuzzy-GA was performed. Fuzzy logic is used to solve complex quantitative problems. Mamdani based model provided a significant performance.
2.	A methodology is described to integrate maintainability by using software metrics.
3.	Neuro-genetic algorithm is used to predict maintainability. It is a hybrid approach of neural-network and genetic algorithm.
4.	Four machine learning algorithms Linear Regression, Multilayer Perceptron, Gaussian Process and SMOreg were used in developing the prediction model.
5.	HMM (Hidden Markov Model) is used to simulate the maintenance behaviors shown as their possible occurrence probabilities.
6.	Proposed a generic model for locating and identifying maintainability requirements.
7.	SMPLerner is an automated learning based approach for training maintainability predictors
8.	Comparison of the five models HPMAS, polynomial regression model, An aggregate complexity measure, Principal components analysis, Factor analysis was performed.
9.	MI model was used to quickly and simply evaluate the source code maintainability
10.	An automated working maintainability model was proposed using design level metrics.

Table 3: Models/Techniques discussed in the article

VI. SYSTEMS STUDIED IN MAINTAINABILITY STUDIES

The systems on which the models were applied in these articles are QUES (Quality Evaluation System) UIMS (User Interface System), Fuzzy interface system, Industrial systems provides by Hewlett Packard and Defense department contractors, Hoduku system and large scale software systems.

Article ID	System Used
1.	UIMS and QUES are the two commercial datasets designed in classical-Ada were used to perform the experiments.
2.	Medium-sized software systems were used to perform the study.
3.	UIMS and QUES are the two case-studies on which the technique is applied.
4.	An open source software system Hodoku was used to statistically test and evaluate the hypothesis.
5.	A possible threshold is set to the empirical data provided by the ISO/IEC 14764:2006
6.	Modification of the existing software product while preserving its integrity.

7.	Eight open source software systems VirtualBox, QuickFix, VCF, Disk Cleaner, Nginx AJP module, Ecere SDK, Chaos, nessDB were used.
8.	11 Software industrial systems were used For fact-finding and process selection decisions.
9.	Industrial systems provided by Hewlett-Packard.
10.	Can be used to estimate the maintainability of large and medium scale industrial systems.

Table 4: Software Systems discussed in the articles

VII. Tools Supporting Software Maintainability Measurement and Prediction

Software metrics are the prime factors to improve quality and enhance the process of development. Apart from planning the development of the software process, aim of developing a software measurement, helps in probing different characteristics of the software and will also determine schedules, costs of the projects and quality.

Several metrics of the software are classified such as Static and Dynamic metrics. Metrics that deal with the features that are associated with the structure and effort estimation, in developing and maintaining the code are known as *Static metrics*. With that being said, *dynamic metrics*, generically, useful in the later stages of software development life cycle.

Programs like java are used to enhance the features such as portability, availability, platform independence and compatibility Hence programs like java are used in tools that calculate software engineering metrics.

While CCCC and JDepend are available for languages such as C and Pascal, tools like Ckjm, JCcn and Refractorit are available for Java programs.

CKJM TOOL: The Object Oriented (O-O) Design metrics which were proposed by Chidamber and Kemerer emphasized more on the design of the class and measures the complexity in it with a strong base of theoretical concepts. Various aspects of CKJM have been explored in the [12]. The findings are reflected below and we have extracted the prime features from the literature that are required for this assignment.

Using CKJM 1.7, eight metrics were calculated. The reflection of the metrics that the tool calculates is given below:

WMC: The total sum of the complexities present in the methods in class for which it is being calculated is known as *Weighted Methods per class*

DIT: When programming in java, all the classes must inherit the object. This metric describes the measure of the level of inheritance from the top of the object hierarchy for each class. This is known to be *Depth of Inheritance Tree.* The least value of DIT is 1.

- The Multiple Inheritance levels are relatively low for DIT.
- Standard Deviation is less CBO, LCOM and NPM.

NOC: The immediate descendants of a class are also to be measured. Hence *Number of Children* is used as a metric to measure the children.

- Minimum mean and median values are seen in NOC.
- Inheritance measures are analogous to DIT.

CBO: Metric that presents the measures of number of classes that are couples to a given class is known as *Coupling between object classes.*

- Factors such as calls to methods, field accesses function arguments, function return types, inheritance and thrown exceptions are considered to make the coupling take place.
- When it comes to standard deviation, it is larger than NOC and DIT.

RFC: *Response for a class,* investigates the number of unique methods that exist and can be executed on receiving a message from an object of that particular class. Minimum standard deviations are observed in RFC.

- When compared against platforms, Java provides a void to enhance the quality in this metric.

LCOM: The metric that is used for measuring the sets of methods that exist in the class and are not connected via sharing of several classes' fields is known as *Lack of cohesion in methods.*

- LCOM is inversely proportional to cohesiveness.
- As projects with less cohesion result in better projects, LCOM enhances the quality.

Ca: The number of other classes using a specific class is determined by *afferent couplings.*

NPM: Determining the methods declared public in the class have to be measured by a certain metric. Hence *Number of Public Methods* provide a void to measure so. Measuring the Application Program Interface's size is also provided by this metric in some package.

COIN TOOL: With [13] as the reference, we have managed to describe the COhesion INheritance tool as follows:

- Cohesion, inheritance, size metrics and maintainability factors of the class hierarchies in java projects are evaluated in this COIN (Cohesion, inheritance).

- Modifiability and understandability of classes and class hierarchies are measured through different metrics in COIN.
- Complexities of class hierarchies in terms of software maintenance are recognised at early design phase through this tool.
- Testability factors of classes and class hierarchies in the software projects are assessed in this tool.
- Design metrics such as MaxDIT, AID, NOCC and summary reports of maintainability factors of individual class hierarchies are measured in this tool.
- Cohesion, coupling, size, inheritance metrics and maintainability metrics are validated in this tool to predict software quality, maintainability, reliability, testability and reusability.
- Inheriting highly coupled systems is a complex process. *Verbatim reuse* is the process of instantiation of such structures.
- 13 Cohesion metrics, 2 coupling metrics, 19 inheritance metrics and 4 size metrics are covered in COIN along with values of modifiability, understandability and testability factors of classes and class hierarchies.
- Main features of COIN are:
  1. At class level and class hierarchy level, it delivers cohesion, inheritance coupling and inheritance metrics of a given Java project.
  2. Hierarchical structures of different class level hierarchies are displayed. A spread sheet shows the metric data for analysis.
  3. Maintainability factors of different class hierarchies are graphically delivered.
  4. Maintainability factors like modifiability, understandability and testability of all the class hierarchical values are given.

- COIN is the extension of Class IN tool, size metrics, cohesion, and coupling are the additional metrics in COIN.
- Software attributes are categorised into *internal and external attributes*.
  1. INTERNAL ATTRIBUTES: classes, cohesion, coupling, Inheritance and size metrics.
  2. EXTERNALATTRIBUTES: functionality, reliability, usability, efficiency, maintainability, portability.
- Internal attributes are directly measurable and external attributes are indirectly measured i.e. they are measured using internal attributes.

- Modifiability and understandability metrics at class level and class hierarchy level are proposed by Sheldon.
  1. TAssert (total assert methods in class), TNM (total no. of methods in test classes), TLOC (lines of code test class) are the units of class testability.
  2. Test cases(total no. of assert methods in hierarchy classes) and testing effort(no. of leaf test classes in an inheritance hierarchy)

Architecture:

- The java code is taken as an input for this architecture.
- The reflection API allows to inspect the number of classes and objects present in the code at run time.
- The structured document of the code can be extracted from Abstract Syntax Tree (AST API).
- It also provides an opening to record Javadoc and comments.
- This tool also yields out dataset for selected metrics and the same in excel files. It also displays class hierarchies in the projects and maintainability graphs.

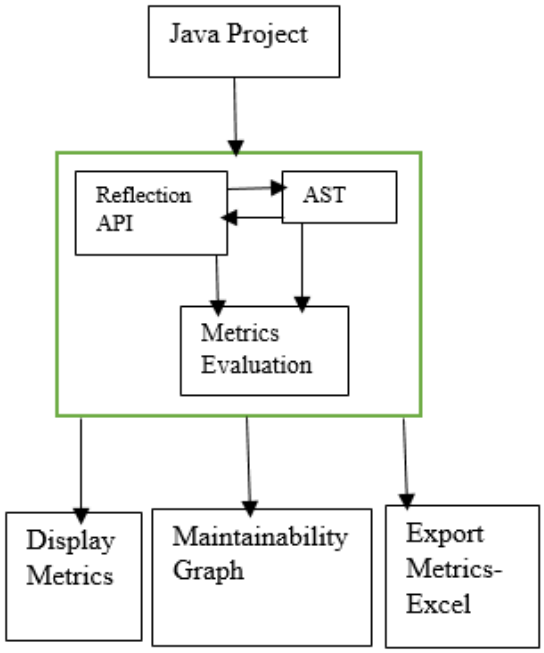

Fig: Architecture of COIN

STRENGTHS AND WEAKNESSES:

The lists of strengths and weaknesses of CKJM and COIN are given below. [14]

- CKJM cannot calculate features like Graphical User Interface, XML output and binding tools such as Ant and Eclipse.
- Fault proneness is calculated by cohesion, coupling and inheritance design characteristics of OO classes in CKJM.
- The files you want to measure cannot be recused automatically in directories in CKJM.
- However CKJM's performs efficiently on 1.6 GHz Pentium-M machine version 1.1 of the tool processed the 33MB of the eclipse 3.0 jar files in 95seconds.
- COIN documents the metrics results in an excel sheet and also in graphical view.

Based on the comparisons between CKJM and COIN on grounds of comparisons such as metrics they are applied on, CKJM has better scope of measuring maintainability as per ours. Maximum metrics are accepted by CKJM tool only, so it is the suitable tool for analysing and measuring maintainability.

VIII. THREATS TO VALIDITY

In our document, as a part of the assignment, we consider the following as our threats to validity.

- Time has always been a constraint and it was not an exception for this assignment as well.
- Enough literature support for tools has been scouted but reliable articles were not found. Hence there is a void in the literature support that we present.
- Though, we are not experienced in this field, we surmise that this is the assignment has been done with the best of our abilities.

IX. REFERENCES

[1] H. A. Al-Jamimi and M. Ahmed, "Prediction of software maintainability using fuzzy logic," in *2012 IEEE International Conference on Computer Science and Automation Engineering*, 2012, pp. 702–705.

[2] J. Lewis and S. Henry, "A methodology for integrating maintainability using software metrics," in, *Conference on Software Maintenance, 1989. Proceedings*, 1989, pp. 32– 39.

[3] L. Kumar, D. K. Naik, and S. K. Rath, "Validating the Effectiveness of Object-Oriented Metrics for Predicting Maintainability," *Procedia Comput. Sci.*, vol. 57, pp. 798–806, 2015.

[4] H. Sharma and A. Chug, "Dynamic metrics are superior than static metrics in maintainability prediction: An empirical case study," in *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, 2015, pp. 1–6.

[5] L. Ping, "A Quantitative Approach to Software Maintainability Prediction," in *2010 International Forum on Information Technology and Applications (IFITA)*, 2010, vol. 1, pp. 105–108.

[6] K. T. Al-Sarayreh, A. Labadi, and K. Meridji, "A Generic Method for Identifying Maintainability Requirements Using ISO Standards," in *Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication*, New York, NY, USA, 2015, pp. 57:1–57:6.

[7] W. Zhang, L. Huang, V. Ng, and J. Ge, "SMPLearner: learning to predict software maintainability," *Autom. Softw. Eng.*, vol. 22, no. 1, pp. 111–141, Aug. 2014.

[8] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using metrics to evaluate software system maintainability," *Computer*, vol. 27, no. 8, pp. 44– 49, Aug. 1994.

[9] T. Pearse and P. Oman, "Maintainability measurements on industrial source code maintenance activities," in , *International Conference on Software Maintenance, 1995. Proceedings*, 1995, pp. 295–303.

[10] S. Muthanna, K. Kontogiannis, K. Ponnambalam, and B. Stacey, "A maintainability model for industrial software systems using design level metrics," in *Seventh Working Conference on Reverse Engineering, 2000. Proceedings*, 2000, pp. 248–256.

[11] M. Riaz, E. Mendes, and E. Tempero, "A systematic review of software maintainability prediction and metrics," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, pp. 367–377.

[12] B. Suri and S. Singhal, "Investigating the OO characteristics of software using CKJM metrics," in *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, 2015, pp. 1–6.

[13] B. R. Reddy, S. Khurana, and A. Ojha, "Software Maintainability Estimation Made Easy: A Comprehensive Tool COIN," in *Proceedings of the Sixth International Conference on Computer and Communication Technology 2015*, New York, NY, USA, 2015, pp. 68–72.

[14] S. Diomidis, "CJKM," 16-Jul-2007. [Online]. Available: http:www.spinellis.gr/sw/ckjm/doc/intro.html. [Accessed: 22-Apr-2016].