Evaluating the maintainability of Freemind software: A case study

Software metrics
Assignment-II

Akshay Kumar Jilla, Maheshwar Kota, Sai Priyatham Dongoor, Sai Kumar Bodicherla, Chandan Srivatsav Ganuga

WORK DISTRIBUTION TABLE

S.No	Name of the person	Contribution in %
1.	Akshay Kumar Jilla	20%
2.	Sai Priyatham Dongoor	20%
3.	Maheshwar Kota	20%
4.	Sai Kumar Bodicherla	20%
5.	Chandan Srivatsav Ganuga	20%

I. INTRODUCTION

Software maintainability is an important attribute of software quality which defines the comfort with which modifications can be adapted into the existing system and maintained [1]. This phase is proven to be the most costliest and effort-consuming phases among all other phases in the SDLC [2]. The present paper is an assignment of the subject Software metrics. The main purpose of this assignment is to evaluate the maintainability of modules of a mind-mapping software called "Freemind" and to determine the best and worst maintainable modules.

This goal is accomplished using a series of steps. Initially, in the second section a Goal-Question-Metric (GQM) tree is formulated using appropriate questions and metrics used to achieve the measurement goal. In the third section, the relevant entities, attributes and metrics are tabulated. The relevant internal attributes, metrics and their scale types are identified and tabulated. In the fourth section, a measure used to calculate software maintainability is calculated and the relationship between various internal attributes and maintainability is described. Later, a suitable study type is selected where an open source tool used to measure the metrics is selected and described with proper justification. The results of measuring the metrics is tabulated. From the results, the best and worst maintainable modules among the modules is displayed.

II. GQM TREE

Goal-Question-Metric is goal centric measurement framework used for deriving measures from the defined organizational or business goals [3]. This framework developed by Basili and Weiss was proven to be the most successful framework as it is adaptable to many organizations and environments [4]. The goal which we have taken for this scenario is to evaluate the maintainability of Freemind software. The goal definition is as follows.

- **Purpose**:
  To evaluate the maintainability of a mind-mapping tool "Freemind" for determining the problematic modules that may require special attention.
- **Perspective**:
  - To select suitable maintainability measure and tools for this evaluation from the perspective of the manager
  - To measure the various metrics, Maintainability Index (MI) and determine the maintainability of the software from the perspective of the developer.

- **Environment and constraints:**
  - "Freemind" is a mind mapping tool developed using programming in JAVA.
  - Five student developers with knowledge of the selected open source tool are working in this project.
  - Tools selected for measuring the metrics must be an open source tool.

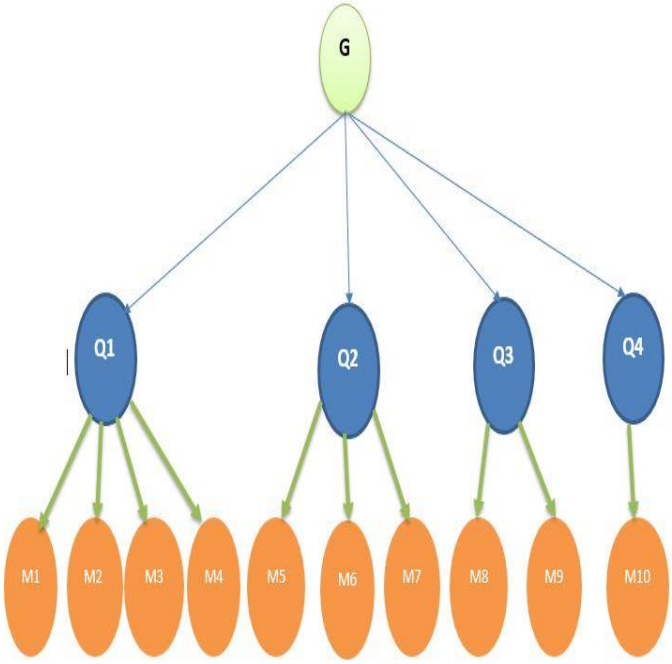

Figure 1: GQM tree

	GOAL
G	To evaluate the maintainability of Free mind
	QUESTIONS
Q1	How are the dependencies between the modules inside the software?
Q2	What are the sizes of the modules inside the software?
Q3	How difficult is it to maintain the code in each module?
Q4	What is the strength of relationship between the elements inside the module?
	METRICS
M1	Coupling between object class (CBO)
M2	Response for class (RFC)
M3	Fan-in
M4	Fan-out
M5	Lines of Code (LOC)
M6	Number of Methods (NOM)
M7	Number of classes (NOC)
M8	Halstead Volume (HV)
M9	McCabe cyclomatic complexity
M10	Lack of cohesion of methods (LCOM)

Table 1: Goal, Questions and Metrics

Description of questions:

- Q1 is selected to identify the dependencies between the modules which refers to the internal attribute of coupling. Coupling is indirectly proportional to maintainability. Here four metrics are selected to measure coupling (M1, M2, M3, M4). Hence, lower the amount of coupling, difficult the maintainability.

- Q2 is selected to determine the sizes of the modules. Size is also indirectly proportional to maintainability. Bigger size modules are difficult to maintain. Hence, to determine the size of modules three metrics have been selected (M5, M6, M7).

- Q3 is selected to find out the complexities in code in each module. Complexity is also indirectly proportional to maintainability. More complex modules are less likely to be maintainable. Hence, to calculate complexity two metrics have been selected (M8, M9).

- Q4 is selected to determine the strength of relationship between the elements inside the module. This relates to the internal attribute of cohesion. Cohesion can determine the system's stability and testability. Here, one package level metric LCOM is used to calculate cohesion.

Justification for metrics:

Motivation for selecting specific metrics for questions formulated to attain our goal is discussed as per literature:

CBO:

Coupling between objects (CBO) metrics is the count of a class coupled with number of classes. If one class uses methods or variables defined in other class, then those two classes are said to be coupled. High CBO is disadvantageous as it indicates lack of reuse potential, high maintenance effort and high effort to test the class [5]. This metric is used to measure coupling attribute.

RFC:
Response for a class (RFC) metric is, execution of number of methods as a response to a message received by an object of a specific class. With larger number of methods elicited as response to a message then testing and debugging off the class becomes complicated and complexity of the class increases [6]. This metric is used to measure coupling attribute.

Fan-in:
Module under consideration is called by number of modules. i.e. "The number of flows of information into a procedure plus the number of global data structures from which a procedure retrieves information" [7]. High fan-in indicates high reuse, reduces redundancy in coding and makes maintenance easier. This metric is used to measure coupling attribute.

Fan-out:
Number of module calls is called the fan-out. i.e. "The number of flows of information from a procedure plus the number of global data structures which the procedure updates" [7]. Higher fan-out value indicates poor system design, as it is the indication of degree of class interdependency. This metric is used to measure coupling attribute.

Lines of code (LOC):
The main purpose of selecting this metric is because it measures the number of lines of code written in each module of the software which is used to predict the size of the module. This metric is used for counting lines of code written. It may include dead code, blanks and comments which should be avoided [8].

Number of methods (NOM):
Average count of all class operations per class is calculated by Number of Method (NOM) which is used to measure size of the module. Higher the NOM value, higher is the complexity and maintainability effort of that class [9].

Number of classes (NOC):
It is the measure of number of packages in a class. This measure is used to measure size of this module. Higher value of this measure requires high maintainance of the module [10].

HV:
Halstead Volume(HV) is used to measure complexity of the module. This measurement is based on counts of operators and operands within the body of code. Halstead volume is calculated with the formula:

V= N*log n.
N= number of operators+ number of operands.
n= number of distinct operators+ number of distinct operands.

Cyclomatic Complexity:
It measures complexity of module by counting independent logical paths through a procedure using control flow graph. Higher cyclomatic complexity represents low quality. Maintainability is inversely proportional to cyclomatic complexity [11]. Cyclomatic complexity is measured by:

V= E-N+2.
E= number of edges in graph.
N= number of nodes in graph.

LCOM:
Lack of cohesion in methods (LCOM) is the measure of relation between the methods and local instance variable of the class. This metric is used to measure cohesion of the modules. Higher LCOM value indicates easy maintainance [12].

III. ENTITIES, ATTRIBUTES, METRICS AND SCALE TYPES

The main goal of this study is to measure the maintainability of Freemind software. To accomplish the goal, we have identified four internal attributes. For each internal attribute, the corresponding metrics and their entities are mapped and tabulate below. Later the metrics and their corresponding scale maps are determined.

Question No.	Attribute	Type of Attribute	Entity	Metrics
Q1	Coupling	Internal	Package	CBO, RFC, Fan In, Fan Out.
Q2	Size	Internal	Module	LOC, NOM, NOC.
Q3	Complexity	Internal	Code	HV, McCabe Cyclomatic complexity.
Q4	Cohesion	Internal	System	LCOM.

Table 2: Relationship between attributes, entities and metrics.

Rationale for selecting internal attributes:

Coupling:
Coupling gives the degree of interdependency between the modules of software system [13]. The metrics which are used to calculate coupling in this case are Coupling between objects (CBO), Response for class (RFC), Fan-in and Fan-out. Modules with low coupling are easy to maintain.

Size:
The internal attribute size is used to determine the extent of magnitude of the modules. The metrics used to measure size in this case are LOC, NOM, NOC. Modules with small size can be deduced as easily maintainable modules.

Complexity:
Complexity can be defined as the degree of difficulty in maintaining and comprehending codes [14]. Complexity can be measured by measuring the control flow of modules [14]. The metrics used to measure complexity in our case are McCabe cyclomatic complexity, Halsted volume (HV). Modules with more complexity are less likely to be maintainable.

Cohesion:
Cohesion can be defined as the degree of relative closeness of the elements inside the module [15]. It can be measured by counting the number of internal calls within the module. The metrics used to calculate cohesion in our assignment is LCOM (Lack of cohesion of methods).

No.	Attributes	Metrics	Scale types
1.	Coupling	CBO	Ratio
		RFC	Ratio
		Fan-in	Absolute
		Fan-out	Absolute
2.	Size	LOC	Ratio
		NOM	Absolute
		NOC	Absolute
3.	Complexity	HV	Absolute
		McCabe cyclomatic complexity	Absolute
4.	Cohesion	LCOM	Absolute

Table: Mapping of metrics to their corresponding

Scale types:
The metrics which were mentioned above are of two scale types:

Ratio:
Ratio is a type of scale type which preserves the order and size of intervals between the units [16]. In ratio scale mapping of measurement must start at zero and the interval size between the units must be equal. In the metrics that we have selected CBO, RFC and LOC come under ratio scale type.

Absolute:
Absolute scale type is used for a metric which can only be computed by counting the number of entities [16]. The only way of mapping the measurement is by actual count in this scale type. In the metrics that we have selected, Fan-in, Fan-out, NOM, NOC, HV, Mccabe's cyclomatic complexity, LCOM use absolute scale type.

IV. MAINTAINABILITY INDEX

Maintainability index was first introduced by Oman and Hagemeister as a composite metric that incorporates many traditional source code metrics into one number which indicates relative maintainability [17]. Maintainability can be defined as the ease with which software system can modify or correct the faults. This in other way can also be called as adapting to the changed environment [18].

There are two different polynomial equations for assessing the maintainability index. One which considers the comments and one which does not consider the comments. The following are the modified 3-metric and 4-metric polynomial equations for maintainability index (MI) [17].

3-Metric: MI = 171 -5.2ln(aveV) -0.23aveV(g') – 16.2ln(aveLOC)

where aveV = Average Halstead Volume per module.
aveV(g') = Average extended cyclomatic complexity

per module.
aveLOC = Average lines of code per module.

4-Metric: MI = 171 -5.2ln(aveV) -0.23aveV(g') – 16.2ln(aveLOC) +50.0sin (sqrt (2.46 per CM)

where aveV = Average Halstead volume per module.
aveV(g') = Average extended cyclomatic complexity per module
aveLOC = Average lines of code per module.

Per CM = Average percent of lines of code per Module.

From the above two equations, the usage of appropriate is based on the decision of a particular individual. If the individual feels that the comments made in the software are appropriate, then the 4-metric polynomial is used. If he decides not to include the comments, then the 3-metric polynomial equation is used.

Relationship between various Internal Attributes of Maintainability:

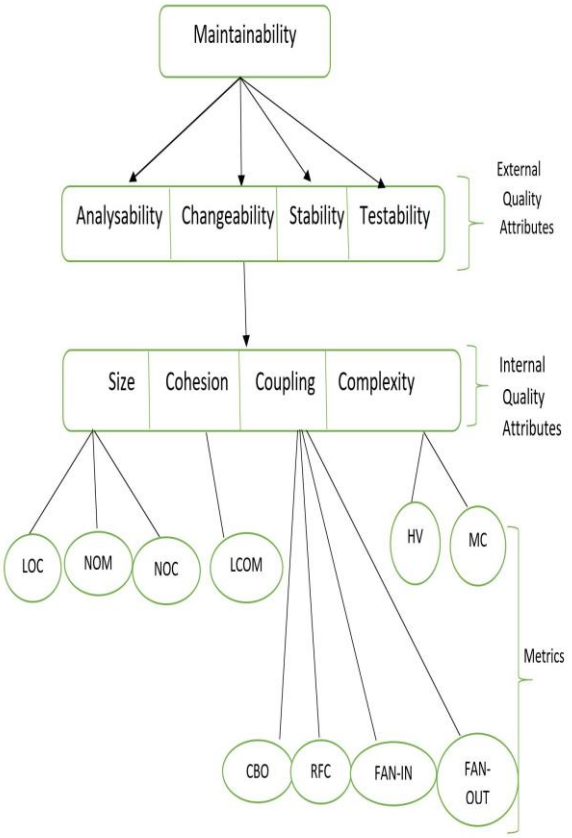

Fig : Relationship between various External and Internal attributes of Maintainability

Maintainability can be defined as the ability of the software system to be modified and how perfectly it adapts to the new modifications [19]. Maintainability is a quality attribute which consists of external and internal quality attributes. The external quality attributes are attributes which indicate the quality of the class and cannot be measured using only the knowledge of a class artifact [20]. This requires additional knowledge of how the process or product relates itself to the environment. The major external quality attributes of Maintainability are Analyzability, Changeability, Stability and Testability [21]. These attributes are generally measured late in the development process. Conversely, the internal quality attributes can be measured based only on the class artifact knowledge [20]. The internal quality attributes are used to predict the external quality attributes. The internal quality attributes used in this assignment are size, cohesion, coupling and complexity. These individual attributes can be easily quantified using various metrics mentioned in the figure above. For measuring the metrics identified above, tools selected are explained in the next sections.

The relationship between various internal attributes and maintainability can be explained in either inversely or directly proportional relation. Size of the modules is directly proportional to maintainability as modules with large size requires high maintenance and vice versa.

Cohesion is defined as the degree of relative closeness of elements within the modules [15]. Cohesion is inversely proportional to maintainability. Modules with low cohesion are difficult to maintain and vice versa.

Coupling is defined as the interdependency between the modules [13]. Coupling is directly proportional to maintainability. Modules with high coupling are difficult to maintain and vice versa.

Complexity can be defined as the degree of difficulty in maintaining and comprehending codes [14]. Complexity is directly proportional to maintainability. Highly complex modules are difficult to maintain and vice versa. These internal attributes are measured using various selected metrics for data collection and later the data is analyzed to identify the worst and best maintainable modules in the later sections.

V. Research Method

Survey, case study and experiment are three types of research methods. Survey is the systematic collection of data from a large sample of population who has experience in the domain of ongoing research, to pursue patterns of the data and generalize it to wider population.

In surveys, it is difficult to check if the obtained answer is genuine or not. Moreover, it has low response rate and has incomplete answers which disturbs the survey. Data acquired in survey is generalized data, it is not accurate. [22]. Hence survey is not selected as research method for this project as we require accurate attributes and entities to measure maintainability of Freemind software.

Experiment is the research based on planned and systematic investigation for collecting and analyzing data on an object to build a hypothesis in a controlled environment. Execution control, Measurement control, ease of replication of an experiment is very high, at the same time investigation cost of an experiment is also very high and low control over behavior of the product. As our research is assessing the Freemind software without changing its behavior and we do not have control over variables of Freemind, experiment is not chosen as our research method [23].

Case study uses multiple sources of evidence in real-life contexts to investigate a phenomenon or a software. A chain of evidences is constructed with traceable reasons and arguments [24]. A Case study is exploratory, qualitative and flexible. Complex concepts can be rationalized to great extent hence data collected is lot richer and has in-depth concepts, which is not possible with surveys or experiments. This is the reason for selecting case study over survey or experiment.

VI. TOOLS SELECTION AND JUSTIFICATION

Maintainability Index is a polynomial based on these measures that was suggested and validated as an indicator of Maintainability by OMAN [25]. Regardless of the intricated tool, measures adopted should assist the goals and questions of the specific measurement initiatives.

JHawk and IntelliJ Idea are the two tools selected to measure maintainability of FreeMind software's modules. JHawk tool measures Halstead metrics (halstead length, halstead volume), Fan-in and Fan-out, cyclomatic complexity, LCOM metrics among the metrics chosen to measure questions in our GQM. Then main advantage of this tool is that it generates a chart for the program and shows warning chart. Measuring looping and iterations which are present in input program are of high priority. It is a standalone eclipse plugin and command line versions [26]. Snapshots of code base can be recorded in JHawk itself or in Data viewer product available in JHawk. These snapshots are empowered by JHawk metrics interchange format. Graphical and textual overtime comparison of metrics is subsidized by JHawk Data Viewer [27].

IntelliJ Idea tool measures CBO, Number of Methods, Number of Classes, RFC, LOC metrics chosen to answer our GQM questions. It can also measure Halstead metrics. Functionality in IntelliJ Idea is provided as plugins. Many small IDE's are fragments of IntelliJ Idea focusing on arrangements of usefulness [28]. It is also possible to perform certain actions 'offline', without launching IDE, beyond working within IntelliJ Idea. This way we can inspect goals, view differences, open files, format files [29].

VII. DATA COLLECTION AND RESULTS

Freemind is a software which is written in JAVA language. There are seven main packages in the Freemind software which are common, controller, extension, main, modes, preferences and view. These packages are measured using various selected metrics with the help of two tools J hawk and IntelliJ Idea mentioned in the above section. The results of the metrics measured using the two tools are tabulated as follows.

Results of Size internal attribute:

The size internal attribute is measured using three metrics LOC, NOM, NOC using the tool IntelliJ Idea. The results of the metrics are tabulated in the following table

Packages	LOC	NOM	NOC
Freemind/ common	2250	64	22
Freemind/ controller	8896	483	117
Freemind/ extension	2164	76	15
Freemind/ main	10456	574	37
Freemind/ modes	33461	2256	266
Freemind/ Preferences	2564	56	14
Freemind/ View	9841	532	55

Fig: results of the metrics LOC, NOM, NOC.

Results of Coupling internal attribute:

The internal attribute coupling is measured based on the results from both the tools JHawk and IntelliJ Idea. The values of the metrics CBO, RFC, Fan-in, Fan-out are tabulated as follows.

Packages	Avg. CBO	Avg. RFC	Fan-in	Fan-out
Freemind/ common	8.14	17.80	0	9
Freemind/ controller	7.54	18.44	3	16
Freemind/ extension	12.60	13.62	0	5
Freemind/ main	19.35	38	0	12
Freemind/ modes	12.10	26.73	19	30
Freemind/ Preferences	6.17	19.37	3	11
Freemind/ View	11.42	24.57	2	12

Fig: Results of CBO, RFC, Fan-in and Fan-out.

Results of cohesion internal attribute:

The internal attribute cohesion is measured with the metric Avg. LCOM using the tool JHawk. The values of the metric LCOM is tabulate below.

Packages	Avg. LCOM
Freemind/ common	0.034
Freemind/ controller	0.124
Freemind/ extension	0.212
Freemind/ main	0
Freemind/ modes	0.07
Freemind/ Preferences	0.302
Freemind/ View	0.286

Fig: Values of the metric LCOM

Results of the complexity internal attribute:

The complexity internal attribute is measured using the metrics Halstead volume (HV) and Mc Cabe cyclomatic complexity. The metrics are measured using the tool JHawk and tabulated as follows.

Packages	HV	Avg. Mc Cabe cyclomatic complexity
Freemind/ common	6365.41	2.04
Freemind/ controller	42645.32	2.53
Freemind/ extension	11543.45	2.36
Freemind/ main	53742.12	3.80
Freemind/ modes	22534.60	1.93
Freemind/ Preferences	81.6	1.62
Freemind/ View	6243.45	1.63

Fig: Values of HV and Mc Cabe cyclomatic complexity Metrics

Results of Maintainability Index:

The Maintainability Index (MI) is calculated using the 4-metrics formula using the comments. The MI is calculated using the values of metrics HV, Ave v(g'), per CM (%) and LOC. If the MI value of the package is less than 65 then the package is difficult to maintain or requires high maintenance. If the value of MI is greater than 85, then the package is easy to maintain or requires low maintenance. The packages which have MI values in between 65 to 85 have moderate maintenance. The values of MI are tabulated below.

Packages	Maintainability Index (MI)
Freemind/ common	76.476
Freemind/ controller	58.064
Freemind/ extension	61.567
Freemind/ main	45.467
Freemind/ modes	97.576
Freemind/ Preferences	113.780
Freemind/ View	100.645

Fig: Values of Maintainability Index

VIII. DATA ANALYSIS AND CONCLUSION

This section answers the questions we have identified in the first section of our GQM tree and deduce the best and worst maintainable modules among the seven modules based on the quantitative data we have presented in the previous section.

The first question we have considered is about the dependencies between the modules inside the software. By observing the average values of CBO and RFC and taking the mean values of both the metrics, we can deduce that modules *"main", "modes"* and *"view"* have more dependencies between the modules and are difficult to maintain. Modules *"common", "preferences", "controller" and "extension"* have less dependencies between the modules and likely requires less maintenance.

The second question we have considered is about the size of the modules inside the software. For answering this question, we have taken the mean of the values of metrics LOC, NOM, NOC. By observing the average values of the metrics, we can deduce that modules *"modes", "controller", "main" and "view"* have large size and thus are difficult to maintain. Modules *"preference", "common", "extension"* have small size and thus are easy to maintain.

The third question is regarding the difficulty in maintaining the code which is complexity. For answering the question, we have considered the values of metrics HV and McCabe cyclomatic complexity. The observation of the values derives that modules *"main", "controller", "modes", "extensions"* have more complex modules and require high maintenance. Modules *"preferences", "view", "common"* are less complex modules and are easy to maintain.

The fourth question is regarding the relationship between the elements inside the software which is cohesion. For answering this question, we have considered the values of the metric Avg LCOM. The modules having less LCOM values have less cohesion and are difficult to maintain. By observing the values, modules *"main", "modes", "controller"," common"* have less cohesion between the elements inside the module and require high maintenance. Modules *"preferences", "view" and "extension"* have more cohesion between the elements and are easy to maintain.

For answering the metrics regarding package level we have compared the values of Avg LCOM, MI and Avg McCabe cyclomatic complexity metrics and deduced the order of maintainability.

Packages	Avg LCOM	Avg McCabe Cyclomatic complexity	MI
Freemind/ common	0.034	2.04	76.476
Freemind/ controller	0.124	2.53	58.064
Freemind/ extension	0.212	2.36	61.567
Freemind/ main	0	3.80	45.467
Freemind/ modes	0.07	1.93	97.576
Freemind/ Preferences	0.302	1.62	113.780
Freemind/ View	0.286	1.63	100.645

By comparing all the values of the metrics and taking the mean values we can deduce that the best maintainable module is *"preference"* and the worst maintainable module is *"main"*. The order of maintainability of the modules from best maintainable modules to worst maintainable modules is "preferences" < "view" < "extension"< "common"< "modes"< "controller" < "main".

Threats to validity*:*

*Internal Validity**:***

- There might be small variations in the values of the metrics due to bugs in OS or tools we have selected.
- The values of metrics are corrected to three values after the point which may have small variations in the outcomes.

External validity:

- The results in the study are based only on few metrics which are selected in the beginning. There are many other many metrics for the same internal attributes which are not considered.
- The MI value is calculated based on only one formula (4-metrics polynomial equation) in this study, whereas it can be calculated in many other ways.

Construct validity:

- The persons who are working on the project are students and does not have practical knowledge in this field. This is mitigated using the best of our abilities.

Conclusion validity;

- There is no funding provided for the selection of tools for measuring the metrics. So, only open source tools are used for measurement.
- Two different tools were used in the study as we could not get the values for all the metrics using single tool.

REFERENCES

[1] 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.

[2] 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.

[3] P. Berander and P. Jönsson, "A Goal Question Metric Based Approach for Efficient Measurement Framework Definition," in *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, New York, NY, USA, 2006, pp. 316–325.

[4] R. Van Solingen, V. Basili, G. Caldiera, and H. D. Rombach, "Goal question metric (gqm) approach," *Encyclopedia of software engineering*, 2002.

[5] R. Harrison, S. Counsell, and R. Nithi, "Coupling metrics for object-oriented design," in *Proceedings Fifth International Software Metrics Symposium. Metrics (Cat. No.98TB100262)*, 1998, pp. 150–157.

[6] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, Jun. 1994.

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

[8] Jones, C. 2006. "Strengths and Weaknesses of Software Metrics". Version 5, Software Productivity Research.

[9] R. Lincke, J. Lundberg, and W. Löwe, "Comparing Software Metrics Tools," in *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, New York, NY, USA, 2008, pp. 131–142.

[10] S. Almugrin and A. Melton, "Estimation of Responsibility Metrics to Determine Package Maintainability and Testability," in *2015 Second International Conference on Trustworthy Systems and Their Applications*, 2015, pp. 100–109.

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

[12] A. D. Bakar, A. Sultan, H. Zulzalil, and J. Din, "Predicting maintainability of object-oriented software using metric threshold," *Information Technology Journal*, vol. 13, no. 8, p. 1540, 2014.

[13] J. Zhao. "Measuring coupling in aspect-oriented systems". In *Proc. of the 10th International Software Metrics Symposium (METRICS)*, Chicago, Illinois, USA, September 2004.

[14] K. Hashim, "A Software Maintainability Attributes Model," *MALAYSIAN JOURNAL OF COMPUTER SCIENCE*, vol. 9, no. 2, pp. 92–97, Dec. 1996.

[15] M. Paixao, M. Harman, Y. Zhang, and Y. Yu, "An Empirical Study of Cohesion and Coupling: Balancing Optimisation and Disruption," *IEEE Transactions on Evolutionary Computation*, vol. PP, no. 99, pp. 1–1, 2017.

[16] N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach, Third Edition*. CRC Press, 2014.

[17] K. D. Welker, “The software maintainability index revisited,” *CrossTalk*, vol. 14, pp. 18–21, 2001.

[18] J. Radatz, A. Geraci, and F. Katki, “IEEE standard glossary of software engineering terminology,” *IEEE Std*, vol. 610121990, no. 121990, p. 3, 1990.

[19] O. Alfonzo, K. Domínguez, L. Rivas, M. Pérez, L. Mendoza, and M. Ortega, “Quality Measurement Model for Analysis and Design Tools Based on FLOSS,” in *19th Australian Conference on Software Engineering (aswec 2008)*, 2008, pp. 258–268.

[20] J. Al Dallal, “Object-oriented class maintainability prediction using internal quality attributes,” *Information and Software Technology*, vol. 55, no. 11, pp. 2028–2048, Nov. 2013.

[21] I. Heitlager, T. Kuipers, and J. Visser, “A Practical Model for Measuring Maintainability,” in *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*, 2007, pp. 30–39.

[22] B. A. Kitchenham and S. L. Pfleeger, “Personal Opinion Surveys,” in *Guide to Advanced Empirical Software Engineering*, Springer, London, 2008, pp. 63–92.

[23] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer Science & Business Media, 2012.

[24] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons, 2012.

[25] H. Benestad, B. Anda, and E. Arisholm, “Assessing software product maintainability based on class-level structural measures,” *Product-Focused Software Process Improvement*, pp. 94–111, 2006.

[26] K. Kaur and H. Singh, “Determination of Maintainability Index for Object Oriented Systems,” *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 2, pp. 1–6, May 2011.

[27] P. E. Linda, V. M. Bashini, and S. Gomathi, “Metrics for Component based measurement Tools,” *International Journal of Scientific & Engineering Research*, vol. 2, no. 5, pp. 199–204, 2011.

[28] K. O. Elish and M. Alshayeb, “A classification of refactoring methods based on software quality attributes,” *Arabian Journal for Science and Engineering*, vol. 36, no. 7, pp. 1253–1267, 2011.

[29] T. Mens, S. Demeyer, B. Du Bois, H. Stenten, and P. Van Gorp, “Refactoring: Current Research and Future Trends,” *Electronic Notes in Theoretical Computer Science*, vol. 82, no. 3, pp. 483–499, Dec. 2003.