Evolution of the Maintainability in Open Source Office Collaboration Tools

Dhruv Gusain*, Emmanuel Uhegbu*, Sidharth Sharma*, Uchechukwu Iroegbu*, Guneet Saini*, Deepika Sharma*
*Dept. of Software Engineering,
Concordia University, Montreal, Canada

***Abstract*—Maintainability is one of the quality characteristics of a software system which is impacted by code readability or complexity as well as modularization. This paper, describes the evolution of maintainability across several versions of two software systems using an improved maintainability model which considers different metrics such as duplication of codes, unit size and volume. We apply this model on two open source office collaboration tools with large code bases to see if we are able to gain important information enough to influence an improvement to their maintainability or to verify the reason as to why this quality characteristic may have improved or declined over time in these systems. Finally, we then verify that this proposed model, as stated by its authors, address the shortcomings of the maintainability index when applied to various systems.**

I. INTRODUCTION

Evolution of Software becomes inevitable as its requirements and its working environment changes. The maintenance of systems during its evolution becomes necessary as the design of the system may need to be changed by fixing bugs to ensure its correctness, adapted to fit in new operational environments by adding new or removing certain features. Since maintenance is an important factor in the use, management and operation of software systems, we decided to study the evolution of maintenance in software system looking into why there may be an increase or decrease in its measure for maintainability.

A popular measure for maintainability that has widely been used in industry is the Maintainability Index. This metric is calculated using a fitting function that considers the volume, Cyclomatic complexity, lines of code and in some variants, consider the percentage lines of comment that constitute the software system. However, we found that using a simple index value resulting from a mathematical equation may not be sufficient enough to inform the maintainers of systems certain details required to effect certain decisions that may be necessary to influence of evolution of a system's' maintainability.

In this paper, we have implemented a simplified version of a more sophisticated model used by engineers at the software improvement group (SIG) in the Netherlands during their consultancy practice. This model claims to take a more pragmatic approach to mapping and ranking well-chosen source code measures on the sub-characteristics of maintainability as suggested by the ISO 9126 standard. In the end, we hope that this model provides an increased degree of precision and detail in the study of the evolution of the software systems under our analysis. We also hope that this work would motivate the implementation of this model across several other types of systems to further test how helpful this suggested model is to every stakeholder involved with their maintenance. The rest of the paper is organized as follows. Section 2 discusses the works related to the study of the effects and results of analyzing the maintainability of systems using metrics. In section 3, we outline the metrics used for measuring maintainability according to the model we have chosen to implement and the methods used for data collection and implementation is discussed. We then report the results of our study and provide a brief description of the results in sections 5 and 6 respectively. Finally, in sections 7 and 8, we discuss the validity of our findings and then provide conclusions to the work done in this paper.

II. RELATED WORK

Different studies have been performed on the measurement of the maintainability of systems. In this section, we present a summary of three contributions that are related to our work.

G.K. Gill and C.F. Kemerer [1] proposed the use of Cyclomatic Complexity metric to improve the software maintenance process by tweaking the complexity metric. The authors have used McCabe's metric, CYCMAX to find Cyclomatic complexity and have used estimated relations of three statistics of the pre-maintenance software with the associated maintenance productivity, three statistics examined were CMAXDENS and its components, KNCSLOC and CYCMAX.

The complexity density ratio proposed by this research could prove to be a simple, but practically useful measure of complexity. It incorporates the McCabe Cyclomatic complexity metric, a well-known and well-understood measure of complexity. One of the most significant results of the paper turned out to be the correlation of the CYCMID AND CYCMIN to McCabe's metric CYCMAX. These results also indicate that the complexity and length of the premaintenance code (CYCMAX and KNCSLOC) are not very useful predictors of the productivity of the maintenance project as is the complexity density. Also, the results suggest that maintenance productivity declines with increasing complexity density.

The research [2] aim to define metrics for measuring the maintainability of a software system and the combination of these metrics to formulate a single index for maintainability. The author's have defined the various attributes that affect the maintainability of software systems. These definitions of these attributes were then combined into a hierarchical structure that serves as a taxonomic definition for software maintainability compatible with the 35 publications upon which it is based. The authors aimed at building an automated maintainability assessment system and have on this paper, defined the maintainability attributes within a target software system using defined metrics. All of these definitions were then combined into a single value that can be used to as an index of the software system's maintainability. To achieve this, a method useful for quantifying trees was presented and used to present a formula that represented the maintainability of the software subsystem.

Ash, D., Yao, L., Oman, P. and Lowthert, B. [3] have proposed the use of automated software maintainability models that monitors code health to control software entropy. Their proposed methodology would enable maintainers receive an early warning and enable early fix to avoid maintenance problems getting out of hand. In order to track code health two methods for quantifying software maintainability from software metrics were implemented and explained. First, Hierarchical Multidimensional Assessment where software maintainability is modeled as a hierarchical structure of attributes of the source code [9]. The approach was tailored towards calculating the pre-post analysis of maintenance changes, where an existing Hewlett-Packard subsystem was analyzed prior to perfective maintenance modification. Once modification was complete, the new system was again analyzed sing this method and results were compared, if there was any detectable change in the maintainability of the system. Another approach was the use of Polynomial Regression Models that use regression analysis as a tool to explore the relationship between software maintainability and software metrics [1]. Here the approach to get software maintainability is to create a polynomial equation where maintainability is a function of the associated metric attributes. The proposed concepts and metrics provided in this were successfully used by a set of tools to measure the maintainability of software. Tests on the two models explained in the paper indicated that they both compute reasonably accurate maintainability scores from calculations based on simple metrics.

III. STUDY DEFINITION

In our proposed project, we will be examining the evolution of metrics by mining the software repositories of open source office productivity software. The metrics that we study, will help to describe the evolution of the maintainability in these types of software and as a result, we will be able to gain insight as to why changes may have occurred in our quality attribute of concern.

To achieve this, we will be using the widely used maintainability index proposed by Paul Oman and Jack Hagemeister together with the maintainability model proposed by Ilja Heitlager, Tobias Kuipers, and Joost Visser from the software improvement group as outlined in the related studies section above.

We will be analyzing 10 versions of two open source projects, Apache OpenOffice andLibreOffice. These projects provide a large-scale implementation with a number of versions which gives us the freedom to analyze and compare the evolution of a software by calculating metrics for each of its releases and comparing the metrics calculated for each release. Our type of study requires that the project should provide us with a suitable number of factors for determining coupling, Cyclomatic complexity and other metrics required for our quality attribute.

Listed below are the version of these systems under our analysis.

Apache OpenOffice: AOO416, AOO416-RC1, AOO415, AOO414 , AOO413, AOO4121, AOO412, AOO411, AOO410, AOO401.

LibreOffice: libreoffice-6.1.5.2, libreoffice-6.2.0.3 , libreoffice-6.1.5.1, libreoffice-6.2.0.2, libreoffice-6.2.0.1, libreoffice-6.1.4.2, libreoffice-6.1.4.1, libreoffice-6.0.7.3, libreoffice-6.1.3.2, libreoffice-6.0.7.2.

METRICS

In our study, we implemented and used several metrics to gain insight into the maintainability the systems under our analysis. In this subsection, we will explain the metrics we have used and the assumption that have been made during their implementation.

Volume:

Formal Definition: - The total size of system is very important when it comes to the measure of maintainability. Larger system requires larger maintenance effort. Referring to paper, higher the volume the lower the analyzability.

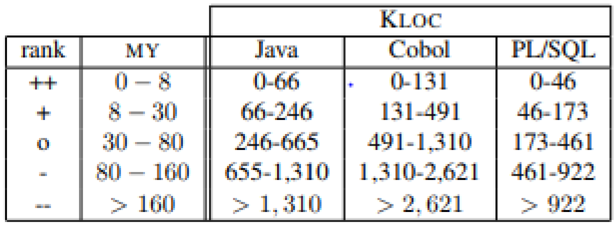
Metric: CountLineCodeDecl + CountLineCodeExe

Definition and Assumptions: To measure volume, lines of code metric can be used but with the exception of shunting out blank lines and comment. To achieve this through Understand tool, two metrics are first calculated, that are:

CountLineCodeDecl: - Number of lines containing declarative source code. Note that a line can be declarative and executable - int I = 0; for instance.

CountLineCodeExe: Number of lines containing executable source code.

Application in the Project: - Post their calculation, the addition of these two metrics for all the files yield us Volume for each version. In understand tool, CountLineCode metric gives us a result inclusive of blank lines and comment. Henceforth, to achieve uniqueness referring to our paper, the above technique was used, where in calculation was a result of sum of two metrics. The below table is finally used to categorize the result for each version, which will further help us in plotting the final model.



		KLOC		
rank	MY	Java	Cobol	PL/SQL
++	$0-8$	0-66	0-131	0-46
+	$8-30$	66-246	131-491	46-173
o	$30-80$	246-665	491-1,310	173-461
-	$80-160$	655-1,310	1,310-2,621	461-922
--	> 160	$> 1,310$	$> 2,621$	> 922

Fig. 1

Complexity Per unit:

Formal Definition: -The complexity property of source code refers to the degree of internal intricacy of the source code units from which it is composed. Complex units are difficult to understand (analyze) and difficult to test, i.e. complexity of a unit negatively impacts the analyzability and testability of the system.

Metric: AvgCyclomatic

Definition and Assumptions: - To measure this metric, we have simply calculated the average cyclomatic complexity for each unit through the referred tool.

AvgCyclomatic: Average cyclomatic complexity for all nested functions or methods.

Post the calculations, specific to each unit, complexity is then plotted for further analysis as per the below model.

To arrive at a more meaningful aggregation, we take the following categorization of units by complexity, provided by the Software Engineering Institute, into account:

cc	Risk evaluation
1-10	simple, without much risk
11-20	more complex, moderate risk
21-50	complex, high risk
> 50	untestable, very high risk

Fig. 2

We now perform aggregation of complexities per unit by counting for each risk level what percentage of lines of code falls within units categorized at that level.

Unit size:

Formal Definition: - The size of each unit throws light on its maintainability. It has been observed to form up a strong statistical relationship between size and Cyclomatic complexity. Larger unit are difficult to maintain not only because of lower analyzability but lower testability as well.

Metric: (CountLineCodeDecl + CountLineCodeExe)

Definition and Assumptions: - To measure unit size, we again use the same metric as used in Volume category but this time it is restricted to specific unit. The risk categories and scoring guidelines are similar to those for complexity per unit, except that the particular threshold values are different.

Duplication:

Formal Definition: - Excessive amounts of duplication of source code fragments are detrimental to a program's main- tainability, in particular to the characteristics of analyzability and changeability. In the SIG model, code duplication is calculated as the percentage of all code that occurs more than one in equal code blocks at least 6 lines. Leading spaces are ignored when calculating duplication and the following rating scheme is used.

Metric: Duplicated Lines of Code

Definition and Assumptions: - As per the paper, code duplication is to be focused for every six lines of code and we are assuming that every six lines contain 40 tokens, which are then given as input to our tool, that is, PMD. It is an open source static source code analyzer that reports on issues found within application code. PMD includes built-in rule sets and supports the ability to write custom rules.

Application in the Project: - Once the count of duplicate code is observed and finally the percentage is determined in terms of the whole source code, the graph is plotted as per the below model for all the respective versions.

rank	duplication
++	0-3%
+	3-5%
o	5-10%
-	10-20%
--	20-100%

Fig. 3

The authors of this model state that other causes such as lack of development skills or supporting tools and other factors could be reasons for the problems associated with duplication in a system.

Challenges faced during metrics implementation:

Duplication:

We faced the major challenge in implementing the code duplication. There was a plugin suggested by Understand tool and it did work but only limited to certain size of files. Large source files went into sleep mode and the tool failed to recognize any input.

To overcome this, we came up with the PMD tool, which helped us count the number of duplicated lines of code in a million source code lines. An Amazon EC2 instance with 32gb of RAM was deployed to allow an increased heap size of 12 Gigabytes to enable the calculation of this metrics across the entire 20 versions under consideration.

Volume:

As mentioned previously, we were stuck in a situation as to how to exclude the comments and blank lines. As the removal of these two aspects yielded a deep impact on the plotting. To overcome this difficulty, we calculated the mentioned volume through the calculation of two separate metrics and then adding them up to find the real result.

Finally, this can be plotted against the mentioned or derived table, which will further help us in analysis through the final maintainability model.

Unit Size:

The only difference between unit size and volume is the unit. In volume the whole source code is into the picture whereas in unit size, individual unit is taken into consideration. We faced the similar issue here as well. That is the exception of not including blank and comments.

This was again achieved by calculating individual metrics at the unit level and then summing them up.

Testing the correctness of the collected metrics

Manual Testing

This was achieved through manually picking the file under consideration which is then checked corresponding to particular metric. The calculated number or result was then saved or kept as a highlighted text. Post that the same file was then manually processed by the team member to cross verify the result. Example: - For Unit Size metric, we first calculated the Lines of Code as per the research paper's definition using the addition of two metrics that are (CountLineCodeDecl + CountLineCodeExe). The result was then saved and the same file was then processed to check the Lines of code for that particular unit. The result matched in all the cases and thereby proved the competency of the metrics.

Automation Testing

As part of the automated testing we have written a junit test case in Java to verify the Cyclomatic complexity of any unit/class present in any of the projects analyzed.

IV. ANALYSIS METHOD

The Maintainability Index has been proposed to objectively determine the maintainability of software systems based on source code of the system being studied. The complete fitting function is as follows:

MI = 171 5.2ln (HV) 0.23CC 16.2ln (LOC)+
50*sin (sqrt (2.4*COM))

where HV is the Halstead Volume (HV) metric, CC is the Cyclomatic Complexity metric, LOC being the average number of lines of code per module, and COM is the percentage of comment lines per module which is an **optional** metric.

Over the years this method that has been used to calculate Maintainability of a system throughout the industry has been debated thoroughly owing to certain limitations listed by the authors of our implemented model as follows:

i. The probability to accept a metric increases greatly when it can be determined what change in a system resulted in a change in the metric. When the MI function produces a value indicating low maintainability, it is not clear what can be done in order to increase the maintainability.

ii. Using average Cyclomatic Complexity as a metric poses a problem as, especially, for systems built using object-oriented languages as the use of averaging on individual system parts tends to mask the presence of high-risk parts.

iii. Counting the number of lines of comment as a metric has no relation with maintainability as in most cases, a comment might be a line of code that has been commented out and documentation for a particular piece of code may have been added, precisely

because it is more complex, hence more difficult to maintain. These reasons act as motivators for the creators of the Maintainability index to make COM optional.

iv. When trying to make the stakeholders of the project (e.g. clients) understand the significance of the formula being used to determine the maintainability of the system, it becomes difficult to justify the use of this formula to invoke empirical experimentation.

v. The Halstead Volume metric is difficult to compute as there is no definition for an operator or an operand in a language such as Java or C# upon which a universal consensus has been achieved.

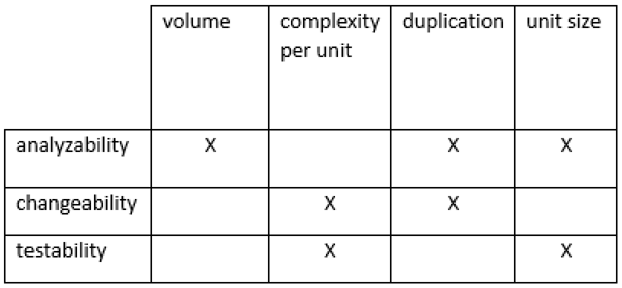
vi. The use of MI results in problems for the management as well as the developers. We find that the lack of control the developers feel they have over the value of MI makes them disregard its significance for quality assessment purposes which puts an impact on the acceptability of this metric by the management.

After assessing the cons of using the Maintainability Index approach we need adopted a new approach that could result infinding a clear answer as to what changes are responsible for adecrease or increase in the maintainability of the system. For determining this we used the maintainability characteristic presented as a part of the quality model of ISO 9126. It is subdivided into characteristics that help us in getting to the root of the problem that causes changes in the maintainability:

Analyzability: The level of difficulty in finding the sys- tem's deficiencies or in recognizing the modules that need modification.

Changeability: The level of difficulty in making modifi- cations to the system.

Testability: The level of difficulty to test the system after the changes have been made.



	volume	complexity per unit	duplication	unit size
analyzability	X		X	X
changeability		X	X	
testability		X		X

Fig. 4

The rows in this matrix represent the 4 maintainability characteristics according to ISO 9126. The columns represent code-level properties, such as volume, complexity, duplication. When a particular property is deemed to have a strong influence on a particular characteristic, a cross is drawn in the. corresponding cell.

We have omitted the stability sub-characteristic and its corresponding property unit testing owing to limitations in calculating the Unit test coverage for all methods in the source code as the implementation is both C++ and Java based.

We used R as our language for analyzing the collected metrics to calculate the maintainability of the projects we were studying. We wrote an R-script to scans the csv file containing the metrics data and filters data by considering a Class file as a unit. The resultant value is then obtained if it satisfies the set criteria. The reason for using R scripts was that the csv's generated were quite large and rather than viewing them in desktop applications like Excel, which was resulting in a long loading time, a better option was to straightaway calculate the required values.

V. STUDYRESULT

OpenOffice Analysis Results

We did an analysis of each version of OpenOffice. We will compare the values calculated for each metric with the versions we chose.The versions are listed in a descending order, that is, it begins from the most recent versions

Volume: It is quite clear, that the values for all the versions are almost the same.

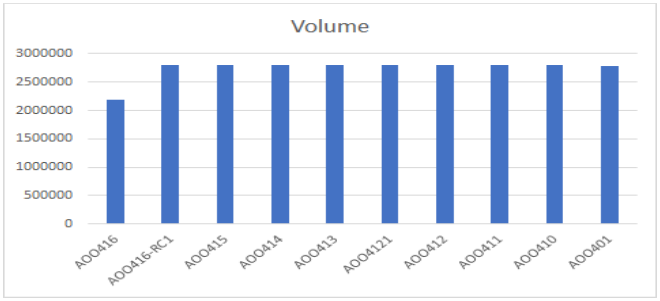


Fig. 5

Complexity: The values obtained are again almost the same for all the versions.

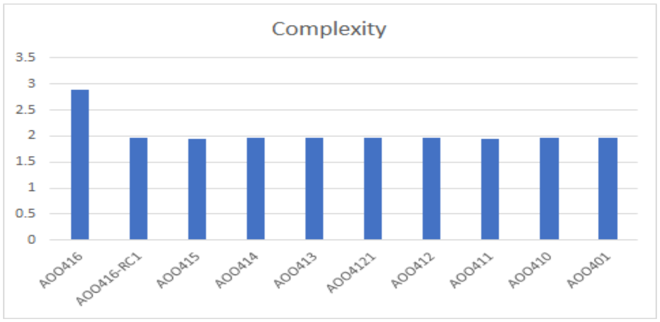


Fig. 6

Duplication: This result is quite interesting as we observe that we have no duplication at all for the earlier versions but it increases with the more recent versions but again remains almost the same.

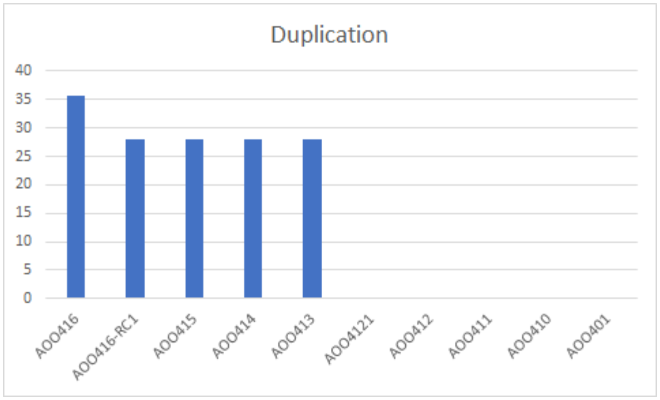


Fig. 7

Unit Size: The unit size metric yields a similar result where we can see identical results across all versions except a dip in the most recent version.

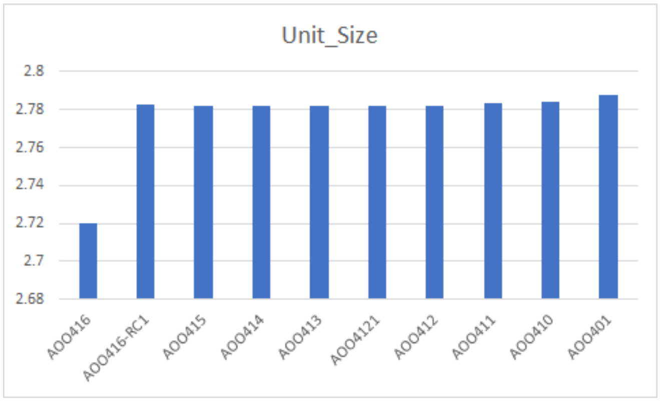


Fig. 8

LibreOffice Analysis Results

Now we analyze each version of LibreOffice. We will compare the values calculated for each metric with the versions we chose.

First, we analyze the Volume of each subsequent release. The versions are listed in a descending order, that is, it begins from the most recent versions. As, we can see the variations in the values are major in only 3 versions but for the rest of the versions any major changes aren't visible.

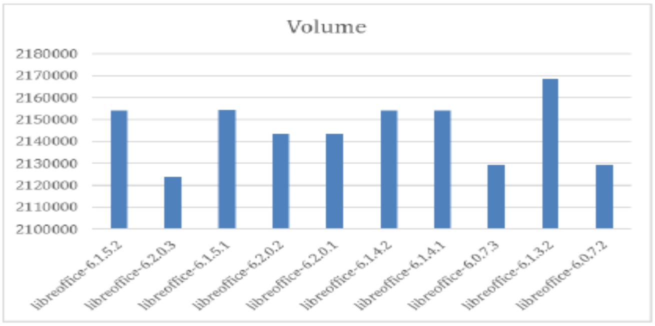


Fig. 9

Complexity: As, we can see there are no significant variations as the values lie in the 2.87-3.0 range.

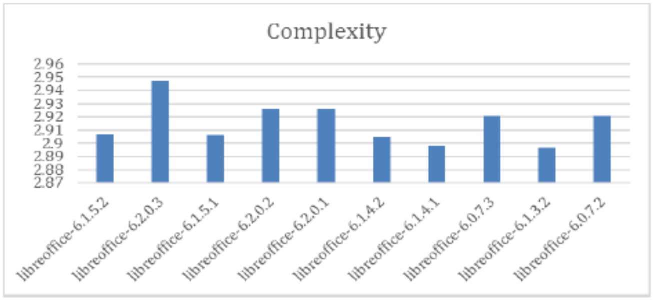


Fig. 10

Duplication: We can observe that there are no significant variations as the majority of the values lie in the 35.6-36.6 range.

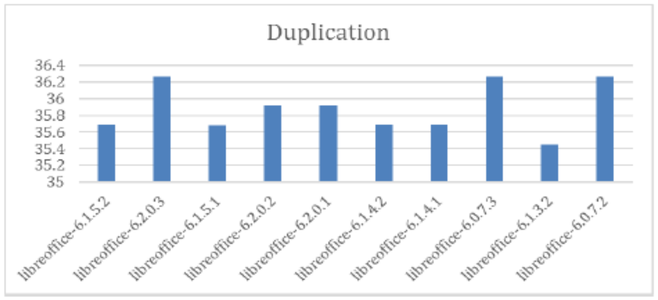


Fig. 11

Unit Size: Almost all the values lie in the 2.85-2.95 range representing minor variations.

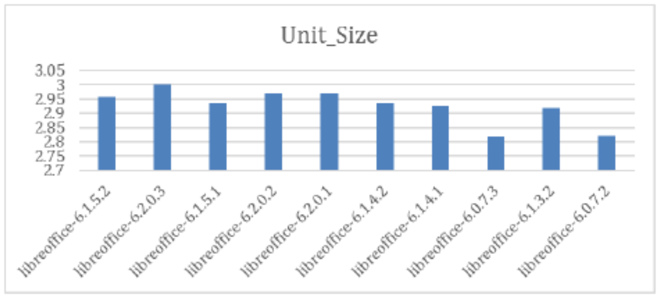


Fig. 12

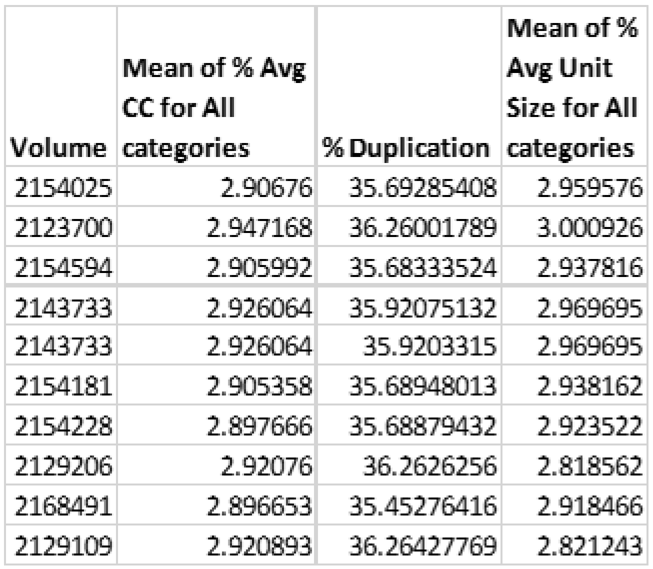
VI. DISCUSSION

In this section we will get deeper insight into the results depicted in the previous section. One can pinpoint, that how changes in individual metrics can yield observable results when combined with other metrics. Interdependency among metrics have helped to draw attention to certain factors, which in turn affect the maintainability. The point of value is, that rather than quantifying it, we were able to infer the broader aspect of the maintainability. It can be seen that consistent maintainability among both systems was proven to be accurate, when checked with the existing comments. If this model is applied and results are discussed, developers can end up knowing the real picture behind the maintainability, which in turn can motivate them to make changes in a direction which might grow the system in the positive direction. Not only will it boost the morale of developers or concerned maintainers of the system but will also help to engage them in regular maintenance of the system.

Below we figure out the individual results as per each system and draw some conclusive and notable points.

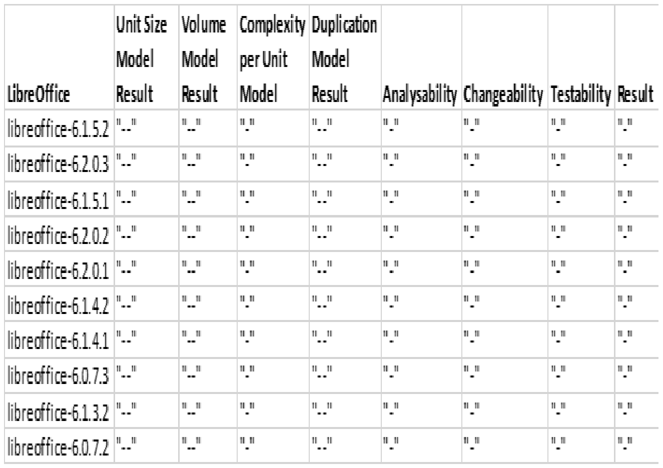
LibreOffice

Let us first start with Volume metric, which discusses Lines of Code and it was seen to show changed trends across different versions. This trend has shown to affect analyzability and must be seen as a bad indicator of maintaining a system due to high in number of Lines of code, there by affecting maintainability. When studying or analysing complexity of a unit, it was seen to be impacted too. The graphical representation shows the clear indication as of how the units were becoming complexed with size and more demand of change requirement. Not only did it affect testability but has also affected changeability. As clear from the model that these two factors are affected by complexity per unit, thereby affecting the overall maintain- ability due to heavy reliability of these. LibreOffice has shown some positive enhancements in the code duplication though, which started with high code percentage of duplication and ended up in lowering down with the subsequent versions. One can note down that analyzability and changeability in the recent versions were affected positively when it comes to code duplication, which do pose a optimistic trend of change and not forgetting the increased LOC at the same time. It becomes quite obvious that since the overall size of an individual version increases, the unit size will be affected as well. It was seen clearly in the analysis report that size of a unit has increased gradually among different versions and was seen to affect testability along with analysability which was left unaffected in the overall volume count. Testing a unit, undoubtedly becomes a heavy task and leads to com- plex testing process with an incremental of unit size. In a nutshell, we observed, that analysability and testability both were affected for a version under consideration if there is a increase in volume size. Not only increased LOC'S contribute to increased unit size as well but makes the system fall in a category of complex testing as well. On the other hand, changeability was seen to give positive trends which can be attributed to less use of predicate statements or more use of nested predicates. Finally, maintainability for libreoffice neither increased nor decreased across version, due to multi depen- dency of analysability, changeability and testability amongst different metics, which in the end yielded neither positivenor negative results. Below are the step by step observations across each version and finally concluding the maintainability which failed to show any positive or negative deviation from its normal flow. All this credit goes to inter-dependency amongst various metrics which finally reach towards the end plotting of all the quality attributes that are analyzability, changeability and testability. We concluded that for maintainability to be effective, there should be some positive or negative trends for all the attributes which we have studied so far and as mentioned in our previous talks.



Volume	Mean of % Avg CC for All categories	% Duplication	Mean of % Avg Unit Size for All categories
2154025	2.90676	35.69285408	2.959576
2123700	2.947168	36.26001789	3.000926
2154594	2.905992	35.68333524	2.937816
2143733	2.926064	35.92075132	2.969695
2143733	2.926064	35.9203315	2.969695
2154181	2.905358	35.68948013	2.938162
2154228	2.897666	35.68879432	2.923522
2129206	2.92076	36.2626256	2.818562
2168491	2.896653	35.45276416	2.918466
2129109	2.920893	36.26427769	2.821243

Fig. 13 Values obtained for metrics



LibreOffice	Unit Size Model Result	Volume Model Result	Complexity per Unit Model	Duplication Model Result	Analysability	Changeability	Testability	Result
libreoffice-6.1.5.2	"--"	"--"	"-"	"--"	"-"	"-"	"-"	"-"
libreoffice-6.2.0.3	"--"	"--"	"-"	"--"	"-"	"-"	"-"	"-"
libreoffice-6.1.5.1	"--"	"--"	"-"	"--"	"-"	"-"	"-"	"-"
libreoffice-6.2.0.2	"--"	"--"	"-"	"--"	"-"	"-"	"-"	"-"
libreoffice-6.2.0.1	"--"	"--"	"-"	"--"	"-"	"-"	"-"	"-"
libreoffice-6.1.4.2	"--"	"--"	"-"	"--"	"-"	"-"	"-"	"-"
libreoffice-6.1.4.1	"--"	"--"	"-"	"--"	"-"	"-"	"-"	"-"
libreoffice-6.0.7.3	"--"	"--"	"-"	"--"	"-"	"-"	"-"	"-"
libreoffice-6.1.3.2	"--"	"--"	"-"	"--"	"-"	"-"	"-"	"-"
libreoffice-6.0.7.2	"--"	"--"	"-"	"--"	"-"	"-"	"-"	"-"

Fig. 14 Application of the values obtained on our Maintainability model

The numerical representation of the above are for '++' we have the value 4, for '+' we have the value 3, for '-' we have the value 2, or '--' we have the value 1.

We use these values to plot the following charts

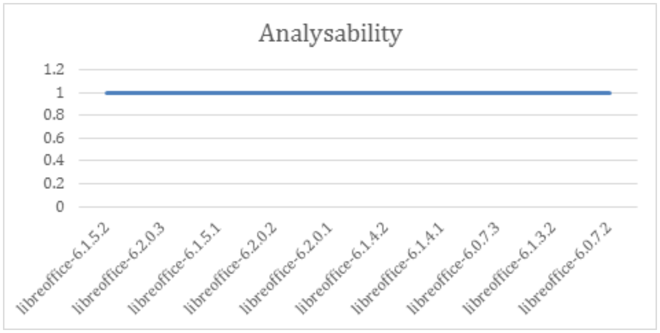


Fig. 15 Chart for Analyzability across all versions

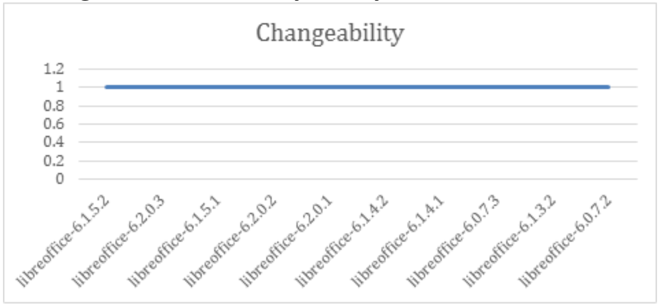


Fig. 16 Chart for Changeability across all versions

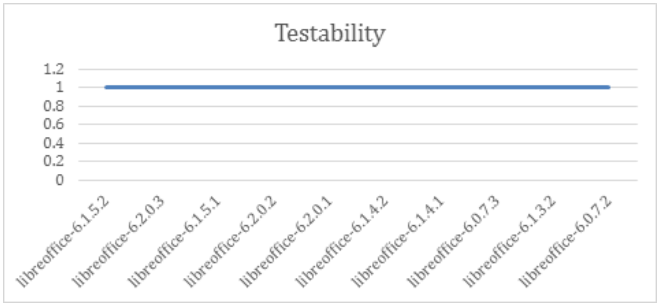


Fig. 17 Chart for Testability across all versions

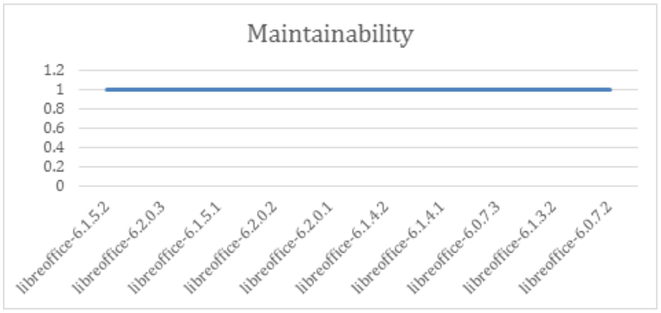
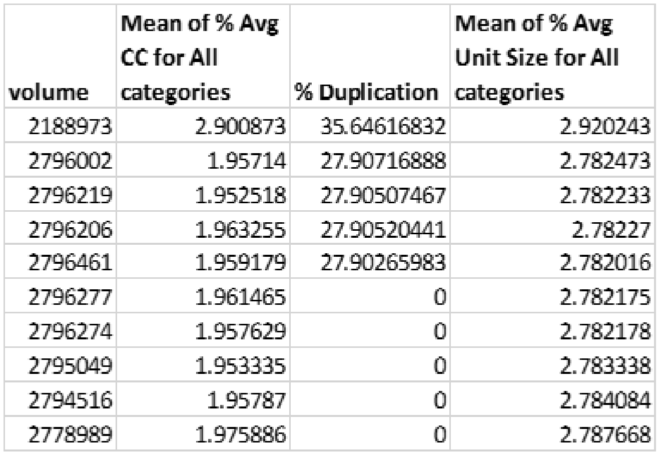


Fig. 18 Resultant Maintainability across all versions

The above charts and table show the system characteristics for the LibreOffice versions under
analysis. In this system, we have found a consistency in the internal characteristics of this system. It is then safe to conclude that the maintainability of this system was not necessarily improved overtime.

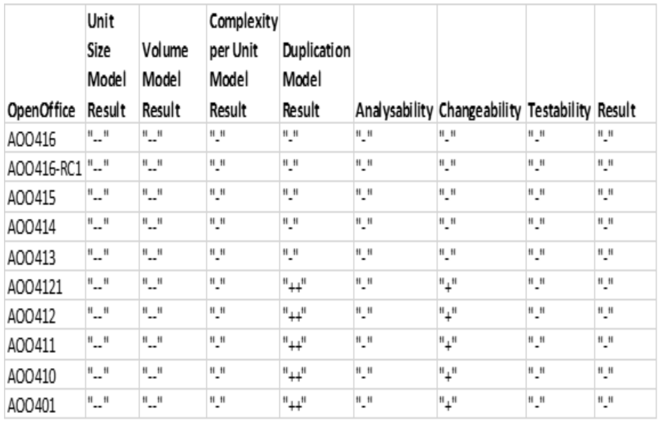
OpenOffice

We observed that with open office, change in volume across multiple versions remained same, until there was a gradual decline in the recent version. One can reach to a conclusion that analyzability remained same, which then showed a positive trend due to decrease in overall volume of the recent version under consideration. Whereas complexity lead to the high rise in the latest version which was analyzed, and remained same for the previous versions. Complexity per unit almost rose by almost one figure which amounts to a lot of change in changeability and testability both in negative trends, there by affecting the overall model. The most astonishing results arose in the duplication metric. It started off with zero percentage of duplication in the first five versions and then started increasing in the latest five versions. The point to be noted is that again the results show similar trends as compared to other metrics, wherein there were same duplication results for all the remaining versions as opposed to latest version. And suddenly it affected the analyzability and changeability in the last version, which rose in a negative trend. Now, moving on to other metric result, that is unit size, lead to same result for all the versions as expected and it was observed to lower down in the latest version which one can undoubtedly observe from the volume metric also. With this, testability got affected towards positive trend since unit's size decreased on an average. Now below are the graphical representations of the analyzability, changeability and testability. We observed that for all the versions except the changeability, all graphs had no change, it was a constant graph. Reason being, the changes when plotted, show such a minute change that observations are almost negligible on a plotted graph. But for changeability, there was a sudden deviation from normal flow due to zero duplication towards first five versions of open office. We also observed that when combined together and plotted through our final model, maintainability neither decreased nor increased and emphasized the fact that even though changes at certain level and in some aspect when made at the source code level but failed to manipulate maintainability towards any trend. One can conclude that for maintainability to be effective, there should be some positive or negative trends for all the attributes which we have studied that are analyzability, changeability and testability. Multi-dependency among various metrics and across various attributes show quite an effective and observable results in the final model. Below are the step by step observations.



volume	Mean of % Avg CC for All categories	% Duplication	Mean of % Avg Unit Size for All categories
2188973	2.900873	35.64616832	2.920243
2796002	1.95714	27.90716888	2.782473
2796219	1.952518	27.90507467	2.782233
2796206	1.963255	27.90520441	2.78227
2796461	1.959179	27.90265983	2.782016
2796277	1.961465	0	2.782175
2796274	1.957629	0	2.782178
2795049	1.953335	0	2.783338
2794516	1.95787	0	2.784084
2778989	1.975886	0	2.787668

Fig. 19 Values obtained for metrics



OpenOffice	Unit Size Model Result	Volume Model Result	Complexity per Unit Model Result	Duplication Model Result	Analysability	Changeability	Testability	Result
AOO416	"--"	"--"	"-"	"-"	"-"	"-"	"-"	"-"
AOO416-RC1	"--"	"--"	"-"	"-"	"-"	"-"	"-"	"-"
AOO415	"--"	"--"	"-"	"-"	"-"	"-"	"-"	"-"
AOO414	"--"	"--"	"-"	"-"	"-"	"-"	"-"	"-"
AOO413	"--"	"--"	"-"	"-"	"-"	"-"	"-"	"-"
AOO4121	"--"	"--"	"-"	"++"	"-"	"+"	"-"	"-"
AOO412	"--"	"--"	"-"	"++"	"-"	"+"	"-"	"-"
AOO411	"--"	"--"	"-"	"++"	"-"	"+"	"-"	"-"
AOO410	"--"	"--"	"-"	"++"	"-"	"+"	"-"	"-"
AOO401	"--"	"--"	"-"	"++"	"-"	"+"	"-"	"-"

Fig. 20 Application of the values obtained on our Maintainability model

The numerical representation of the above are for '++' we have the value 4, for '+' we have the value 3, for '-' we have the value 2, or '--' we have the value 1.

We use these values to plot the following charts

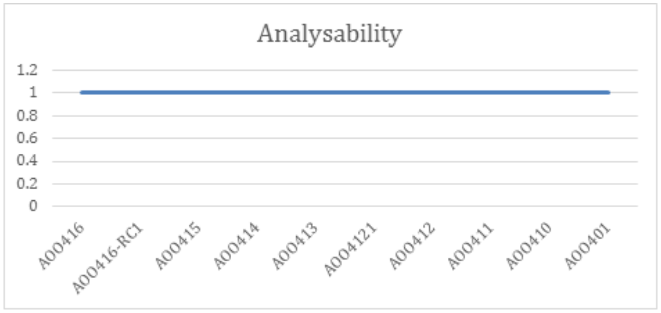


Fig. 21 Chart for Analyzability across all versions

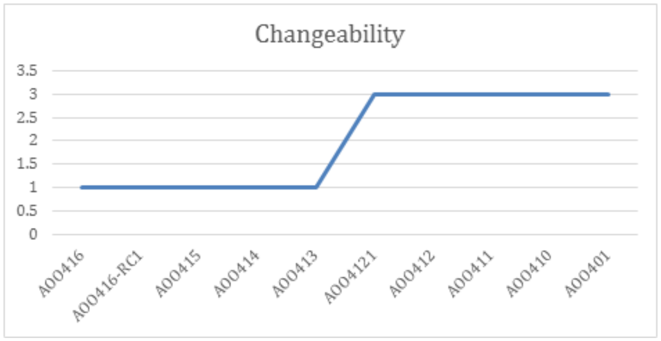


Fig. 22 Chart for Changeability across all versions

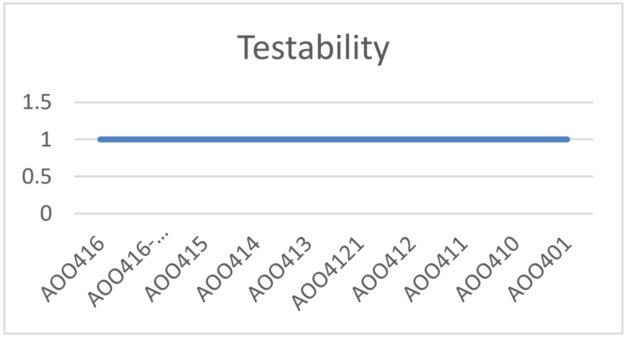


Fig. 23 Chart for Testability across all versions

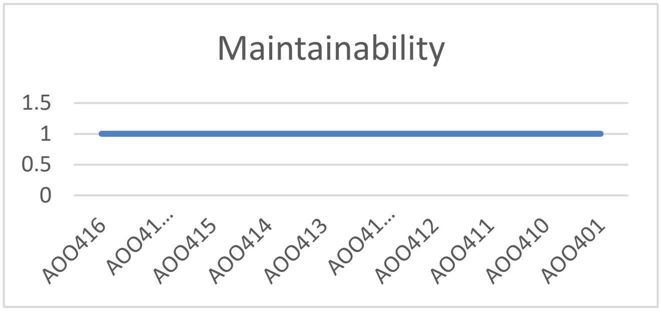


Fig. 24 Resultant Maintainability across all versions

The above charts and table show the system characteristics for the OpenOffice versions under analysis. Although changeability increased from the 5th version, we still find that the maintainability of this system did not evolve accordingly.

B. Possible Answers to research questions

As we have found through the trends on the model we have chosen in this project, we can clearly see the maintenance found that the maintainability of this system from across versions. As part of the verification of our model, we have decided to calculate the maintainability of several versions across our system verify consistency of the maintainability of our system.

In our bid to measure the MI of these versions, we have replaced the Hallstead's volume with Object vocabulary (OV). This measures the quantity of unique information contained by the programs' source code, similar to the traditional Halstead vocabulary with dynamic language compensation [1] due to the fact that different languages were used in the development of the projects under analysis.

$$MI = 171 - 5.2 * \log2(V) - 0.23 * G - 16.2 * \log2 (LOC) + 50 * \sin (\mathrm{sqrt} (2.4 * CM))$$

where V is the Halstead Volume (HV) metric, G is the Cyclomatic Complexity metric, LOC being the average number of lines of code per module, and CM is the percentage of comment lines per module which is an optional metric.

A Calculation for this index provided the following results for the first 3 versions of our source code under review. From the above table, we can conclude that the maintainability of these systems was neither increasing nor decreasing as shown by our model. However, we are not able to attribute these values to any direct reason or system characteristic.
After, analyzing the results we got from ProjectCodeMeter[11] we verified the results we got by checking whether the values we produced matched or not.

A Calculation for this index provided the following results for the first 3 versions of LibreOffice.

VII. THREATS TO VALIDITY

External validity

Threats to the externality validity of this implementation constitutes of how generalized are findings are. Since our study has been performed on two open source systems, Apache Open office and LibreOffice, there is need for this model to be applied on commercial software at various scales. As this model has been implemented on codebases of software systems developed using strongly typed programing languages, we may need to apply this model to systems that are a result of other types of Object-oriented programming languages. To generalize results, there is a need to measure the evolution of maintainability with systems in production environments.

Internal validity

The internal validity of this study refers to how the implementation in the study supports the models' claim providing insight to the evolution of maintainability of systems. Here we would like to point out that the consideration of duplication to be considered for blocks that constitute of repeated code of 6 lines. This may be a threat to internal validity as we do not know if this value allows for accurate results in systems of various scales.

Construct Validity

The degree to which this study test what it claims would heavily depend on the efficiency of the mapping that depends on external system attributes that are identified by the ISO 9126. Does ISO 9126 completely identify system characteristics that affect maintainability? We consider this a threat due to the fact that improved standards such as the ISO 25000 have been developed to solve the issues of the standard used in the model we have chosen in our study.

VIII. CONCLUSIONS

In our study, we calculate metrics (volume, duplication, unit size and Cyclomatic complexity) values for 10 versions of Apache OpenOffice and LibreOffice. We then follow a sketched practical maintainability model [2] to map these gathered metrics to system characteristics identified to affect maintainability by the ISO 9216 standard. This mapping allowed us to gain insight into the evolution of the software systems under analysis across their progressive versions.

Once these metrics that represent source code properties were collected using tools such as Scitools Understand, PMD CPD and projectcodemeter, they were then ranked according to the schemes identified by the authors of this model. Using these ranks, the mapping for the versions of these software systems were then plotted to gain insight to their overall maintainability.

In our bid to confirm that this proposed model did not provide results that were not necessarily false. We also calculated the Maintainability index of these versions under consideration. However, some modifications were then made to the formula for this index following from the reason that the conventional MI makes use of the Halstead volume. Due to the fact that there is no consensual definition of what constitutes an operator or operand in languages such as java or c# and a couple of other reasons. We replaced this measure in the formulae with Object vocabulary. This new measure considers the quantity of unique information contained by the programs' source code, similar to the traditional Halstead Vocabulary with dynamic language compensation. Once calculated and compared to the result from the model being implemented, we have found that there wasn't an overall increase in the system's maintainability and as anticipated, we found that the model provided information as to what aspects of the systems implementation constituted the almost insignificant increase or decrease of maintainability across versions.

Finally, we anticipate that future research opportunities lie within the boundaries of this work as we hope for a clear definition of the requirements to gather the metrics needed for this model to be proposed concretely. So much so that there would
be less dependence on expert opinions when applying this model to various other problem domains. As a consequence of this suggestion, we employ more researchers, developers and teams to implement this model within their production environments to further gain insights into the aspects of their codebase that would affect its overall maintainability.
Data: All the data and reported metrics gathered are online at the GitHub URL
https://github.com/drvg5/SoftwareMeasurement6611_winter2 019

I. REFERENCES

[1] P. Oman and J. Hagemeister, "Metrics for assessing a software system's maintainability", in Proceedings Conference on Software Maintenance 1992, Orlando, FL, USA, USA, 1992, pp. 337 - 344.
[2] I. Heitlager, T. Kuipers and J. Visser, "A Practical Model for Measuring Maintainability", in 6th International Conference on the Quality of Information and Communica-tions Technology (QUATIC 2007), Lisbon, Portugal, 2007.
[3] Ash, D., Yao, L., Oman, P. and Lowthert, B. (1994). Using software maintainability models to track code health. In: Proceedings 1994 International Conference on Software Maintenance. Victoria: IEEE.
[4] S. R. Chidamber , C. F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Transactions on

Software Engineering, vol. 20, no. 6, pp. 476-493, June 1994.
[5] GitHub - apache/openoffice: Mirror of Apache OpenOf- fice", GitHub, 2019. [Online]. Available: https://github.com/a pache/openoffice. [Accessed: 02- Feb- 2019].
[6] LibreOffice", GitHub, 2019. [Online]. Available: https:/ /github.com/LibreOffice. [Accessed: 01- Feb- 2019].
[7] PMD (software)", En.wikipedia.org, 2019. [On- line]. Available: https://en.wikipedia.org/wiki/PMD_(software). [Accessed: 14- Feb- 2019].
[8] Understand (software)", En.wikipedia.org, 2019. [On- line]. Available: https://en.wikipedia.org/wiki/Understand_(so ftware). [Accessed: 14- Feb- 2019].
[9] P. Oman, HP-MAS: A Tool for Software Maintainability Assessment, U.I. Software Engineering Test Lab Report #92- 07-ST, August 1992, 36 pages.
[10] Understand (software)", En.wikipedia.org, 2019. [On- line]. Available: https://en.wikipedia.org/wiki/Understand_(sof tware). [Accessed: 14- Feb- 2019].
[11] "ProjectCodeMeter User Manual - cost estimation and development productivity measurement software using automatic source code metrics analysis", Projectcodemeter.com, 2019. [Online]. Available:http://www.projectcodemeter.com/cost_estimation/images/files/PCMProManual.pdf. [Accessed: 18- Mar- 2019]

Peer evaluation form

Self: **Deepika Sharma**

Teammate #1: **Sidharth Sharma**

Teammate #2: **Dhruv Gusain**

Teammate #3: **Uhegbu Emmanuel**

Teammate #4: **Uchechukwu Iroegbu**

Teammate #5: **Guneet Saini**

The following is a list of statements to be answered for yourself and each of your team members. Think carefully about assigning rating values for each of the statements.

1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree

	Self	#1	#2	#3	#4	#5
Was attending group meetings (either physical or remote).	1	1	1	1	1	1
Willingly accepted assigned tasks.	1	1	1	1	1	1
Contributed positively to group discussions.	1	1	1	1	1	1
Completed work on time.	1	1	1	1	1	1
Helped others with their work when needed.	1	1	1	1	1	1
Did work accurately and completely.	1	1	1	1	1	1
Worked well with other group members.	1	1	1	1	1	1
Overall was a valuable member of the team.	1	1	1	1	1	1

Complete the next table with the percentage of work that you and your team members contributed for each task of the project. Note that the sum of percentages of all team members should not be over 100% for each task.

Source code development	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Data Collection	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Statistical analysis	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Report writing	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Presentation preparation	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Team organization and leadership	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%

Self: **Sidharth Sharma**

Teammate #1: **Dhruv Gusain**

Teammate #2: **Deepika Sharma**

Teammate #3: **Uhegbu Emmanuel**

Teammate #4: **Uchechukwu Iroegbu**

Teammate #5: **Guneet Saini**

The following is a list of statements to be answered for yourself and each of your team members. Think carefully about assigning rating values for each of the statements.

1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree

	Self	#1	#2	#3	#4	#5
Was attending group meetings (either physical or remote).	1	1	1	1	1	1
Willingly accepted assigned tasks.	1	1	1	1	1	1
Contributed positively to group discussions.	1	1	1	1	1	1
Completed work on time.	1	1	1	1	1	1
Helped others with their work when needed.	1	1	1	1	1	1
Did work accurately and completely.	1	1	1	1	1	1
Worked well with other group members.	1	1	1	1	1	1
Overall was a valuable member of the team.	1	1	1	1	1	1

Complete the next table with the percentage of work that you and your team members contributed for each task of the project. Note that the sum of percentages of all team members should not be over 100% for each task.

Source code development	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Data Collection	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Statistical analysis	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Report writing	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Presentation preparation	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Team organization and leadership	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%

Self: **Dhruv Gusain**

Teammate #1: **Deepika Sharma**

Teammate #2: **Sidharth Sharma**

Teammate #3: **Uhegbu Emmanuel**

Teammate #4: **Guneet Saini**

Teammate #5: **Uchechukwu Iroegbu**

The following is a list of statements to be answered for yourself and each of your team members. Think carefully about assigning rating values for each of the statements.

1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree

	Self	#1	#2	#3	#4	#5
Was attending group meetings (either physical or remote).	1	1	1	1	1	1
Willingly accepted assigned tasks.	1	1	1	1	1	1
Contributed positively to group discussions.	1	1	1	1	1	1
Completed work on time.	1	1	1	1	1	1
Helped others with their work when needed.	1	1	1	1	1	1
Did work accurately and completely.	1	1	1	1	1	1
Worked well with other group members.	1	1	1	1	1	1
Overall was a valuable member of the team.	1	1	1	1	1	1

Complete the next table with the percentage of work that you and your team members contributed for each task of the project. Note that the sum of percentages of all team members should not be over 100% for each task.

Source code development	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Data Collection	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Statistical analysis	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Report writing	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Presentation preparation	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Team organization and leadership	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%

Self: **Uhegbu Emmanuel**

Teammate #1: **Dhruv Gusain**

Teammate #2: **Sidharth Sharma**

Teammate #3: **Guneet Saini**

Teammate #4: **Uchechukwu Iroegbu**

Teammate #5: **Deepika Sharma**

The following is a list of statements to be answered for yourself and each of your team members. Think carefully about assigning rating values for each of the statements.

1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree

	Self	#1	#2	#3	#4	#5
Was attending group meetings (either physical or remote).	1	1	1	1	1	1
Willingly accepted assigned tasks.	1	1	1	1	1	1
Contributed positively to group discussions.	1	1	1	1	1	1
Completed work on time.	1	1	1	1	1	1
Helped others with their work when needed.	1	1	1	1	1	1
Did work accurately and completely.	1	1	1	1	1	1
Worked well with other group members.	1	1	1	1	1	1
Overall was a valuable member of the team.	1	1	1	1	1	1

Complete the next table with the percentage of work that you and your team members contributed for each task of the project. Note that the sum of percentages of all team members should not be over 100% for each task.

Source code development	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Data Collection	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Statistical analysis	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Report writing	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Presentation preparation	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Team organization and leadership	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%

Self: **Uchechukwu Iroegbu**

Teammate #1: **Dhruv Gusain**

Teammate #2: **Sidharth Sharma**

Teammate #3: **Uhegbu Emmanuel**

Teammate #4: **Deepika Sharma**

Teammate #5: **Guneet Saini**

The following is a list of statements to be answered for yourself and each of your team members. Think carefully about assigning rating values for each of the statements.

1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree

	Self	#1	#2	#3	#4	#5
Was attending group meetings (either physical or remote).	1	1	1	1	1	1
Willingly accepted assigned tasks.	1	1	1	1	1	1
Contributed positively to group discussions.	1	1	1	1	1	1
Completed work on time.	1	1	1	1	1	1
Helped others with their work when needed.	1	1	1	1	1	1
Did work accurately and completely.	1	1	1	1	1	1
Worked well with other group members.	1	1	1	1	1	1
Overall was a valuable member of the team.	1	1	1	1	1	1

Complete the next table with the percentage of work that you and your team members contributed for each task of the project. Note that the sum of percentages of all team members should not be over 100% for each task.

Source code development	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Data Collection	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Statistical analysis	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Report writing	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Presentation preparation	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Team organization and leadership	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%

Self: **Guneet Saini**

Teammate #1: **Deepika Sharma**

Teammate #2: **Sidharth Sharma**

Teammate #3: **Uhegbu Emmanuel**

Teammate #4: **Uchechukwu Iroegbu**

Teammate #5: **Dhruv Gusain**

The following is a list of statements to be answered for yourself and each of your team members. Think carefully about assigning rating values for each of the statements.

1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree

	Self	#1	#2	#3	#4	#5
Was attending group meetings (either physical or remote).	1	1	1	1	1	1
Willingly accepted assigned tasks.	1	1	1	1	1	1
Contributed positively to group discussions.	1	1	1	1	1	1
Completed work on time.	1	1	1	1	1	1
Helped others with their work when needed.	1	1	1	1	1	1
Did work accurately and completely.	1	1	1	1	1	1
Worked well with other group members.	1	1	1	1	1	1
Overall was a valuable member of the team.	1	1	1	1	1	1

Complete the next table with the percentage of work that you and your team members contributed for each task of the project. Note that the sum of percentages of all team members should not be over 100% for each task.

Source code development	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Data Collection	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Statistical analysis	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Report writing	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Presentation preparation	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Team organization and leadership	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%