PROJECT REPORT

TEAM-P SOEN 6611

Sehajpreet Singh Gill Vrind Gupta Kartik Nagpal Pritpal Kaur
Koteswara Rao Panchumarthy Karamveer Kaur Sangha

Department of Computer Science and Software Engineering
Concordia University
Montreal,Quebec,Canada

ABSTRACT

Our aim in this study is to estimate six software metrics and to analyse their correlation with each other with some predefined hypothesis. The metrics that we are considering are two test coverage metrics, one test suite effectiveness, one complexity metric, one software maintenance and one software quality metrics. To understand the field of software measurement and the above metrics, we have chosen 5 open source projects , three projects have more than 100 KSLOC, namely Apache Commons Collection, Apache Commons Math, Apache Commons Configuration, and two have less than 100 KSLOC, namely Apache Commons Logging and Apache Commons File Upload

. Keywords: Statement Coverage, Branch Coverage, Mutation Testing, Cyclomatic Complexity, Code Churn and Defect Density.

INTRODUCTION

Software quality is one of the key factors to the efficiency in the development activity. Software quality can be measured using various software metrics based on the need and requirement. As the software evolves, documenting the changes with respect to the previous versions helps the development team analyse what improvements need to be made. And also, measuring the software quality can improve all the activities involved in SDLC directly or indirectly. Software quality helps to determine the correctness of the system. The main objective of this study is to analyze the five open source projects for finding six software measurement metrics and the correlation between them.We have selected five projects that are totally developed in java programming language.The 1st and 2nd metric includes test coverage metrics i.e statement coverage and branch coverage. The 3rd metric is a test suite effectiveness metric ,we used mutation score for observing the effectiveness of projects.For to find the mutation testing we used an eclipse plugin called pitclipse. The 4th metric is destined to find the complexity of code i.e McCabe complexity , also known as cyclomatic complexity. The 5th metric includes software maintenance effort, we estimate code churn for this purpose .It mainly measures two separate churn metrics: the number of added lines of code, and the number of deleted lines. Finally ,6th metric is about software quality i.e post-release defect density. It is used to find the software quality by calculating the number of bugs for each version .The number of bugs will be calculated through the Jira reports.Later we used Loc Metrics,a windows tool to calculate the Ksloc the and at last but one of the important correlations is conducted.

After calculating these five metrics, we had to perform correlation analysis between them. We used Spearman Correlation Coefficient for the same. We performed correlation analysis between Metric 1&2 and 3, Metric 1&2 and 4, Metric 1&2 and 6, Metric 5 and 6.

1.METHODOLOGY

1.1 SUBJECTS PROJECTS

a) Apache Commons Collections

Description: Base version (4.3) that we have selected had 126.68 KLOC. The Java Collections Framework was a major addition in JDK 1.2. It added many powerful data structures that accelerate development of most significant Java applications. Since that time, it has become the recognized standard for collection handling in Java

Link: https://github.com/apache/commons-collections.git

b)Apache Commons Logging

Link: https://github.com/apache/commons-logging.git

Description: A thin adapter allowing configurable bridging

to other, well-known logging systems.
We have worked on different versions of this project. Base version (1.2.1 Snapshot) that we have selected has 9.63 KLOC. The Commons Configuration software library provides a generic configuration interface which enables a Java application to read configuration data from a variety of sources. Commons Configuration provides typed access to single, and multi-valued configuration parameters

c)Apache Commons Configuration

Description: We have worked on different versions of this project. Base version (2.6) that we have selected has 105 KLOC. The Commons Configuration software library provides a generic configuration interface which enables a Java application to read configuration data from a variety of sources. Commons Configuration provides typed access to single, and multi-valued configuration parameters.
Link:https://github.com/apache/commons-configuration

d)Apache Commons File Upload

Description: Provides a simple and flexible means of adding support for multipart file functionality servlets and applications.
We have worked on base versions of this project. Base version (2.0 Snapshot) that we have selected has 14.56 KLOC. The Commons FileUpload package makes it easy to add robust, high-performance, file upload capability to your servlets and web applications.

Link: http://commons.apache.org/fileupload/

e)Apache Commons Math

Description: a library of lightweight, self-contained mathematics and statistics components addressing the most common practical problems not immediately available in the Java programming language or commons-lang. We worked on the version 4.0 Snapshot, with the source lines of code to be 16 kLOC.
Link: https://github.com/apache/commons-math.git

2.METRICS
1.Statement Coverage: Statement coverage is a white box testing technique, used to quantify the total number of lines in the source code that have been successfully executed. It ensures quality by determining whether each statement in the source code is executed at least once. In statement coverage, test cases are written trying to ensure the execution of all statements of a program at least once.
Statement coverage is said to be 100% if every statement in the program is executed at least once. It covers only the true conditions.

A = Total no. of Statements Exercised

B = Total no. of Executable Statements in a Program

$Statement\ Coverage = (A/B) * 100$

2.Branch Coverage: Branch Coverage measures the proportion of branches in the control structure (Control Predicates) that are executed by the test suite. Achieving full branch coverage will protect against errors in which some requirements are not met in a certain branch. Test coverage criteria requires maximum test cases such that each condition in a decision takes on all possible outcomes at least once, and each point of entry to a program or subroutine is invoked at least once. That is, every branch (decision) taken each way, true and false. It helps in validating all the branches in the code making sure that no branch leads to abnormal behaviour of the application.

A = No. of Decision Statements Exercised

B = Total No. Decision Outcomes

$Branch\ Coverage = A/B * 100$

expressed as Lines of Code(LOC) or Function Point(FP).
Defect density= Defect count/KSLOC
Defect density= number of post release defects
KSLOC= total number of lines SLOC

3.Mutation Score: It is a metric which is used for estimating the effectiveness of test suites by generating mutants in code and check whether the test cases are able to find the errors or not.If it finds the error then the mutants is killed otherwise not. Basically, mutation testing is all about creating a mutant in a program and checking if that particular mutant is being killed by the test cases provided. If the mutation score is 1 (100%), all the mutants are killed by test suite and vice versa.

Mutation Score=Killed mutants/Total no. of mutants

4.McCabe Cyclomatic Complexity: It is used to estimate the complexity of the program by measuring the linearly independent paths of the source code
McCabe complexity= No.of Edges- No. of Vertices +2 connected components*

or

McCabe Complexity= No. of control predicates +1

5.Code Churn: Code churn is a measure that tells you the rate at which your code evolves. Code churn has several usages:
1.Visualize your development process.
2.Reason about delivery risks
3. Track trends by task
CodeScene measures two separate churn metrics: the number of added lines of code, and the number of deleted lines.

6.Post-release Defect Density: The post -release Defect Density is the number of identified defects found during the operational phase per 1000 source line of code.
It measures the defects relative to the software size

3.DATA COLLECTION
We have used different tools to calculate the values of all of the above metrics. The process of collecting and tools used is as follows:

1.Metric 1 & 2: Statement coverage and Branch coverage
Tools Used: JaCoCo

Procedure: Statement coverage and Branch coverage is calculated using JaCoCo (Java Code Coverage) tool. JaCoCo tool is a widely used tool that can generate reports in html, csv and xml format.
JaCoCo tool can be downloaded automatically from maven repositories by including JaCoCo plugin in the pom.xml file of the project. The 5 projects that are selected have test cases built in. JaCoCo run these test cases can check the number of missed/covered branches and missed/covered statements.

2.Metric 3: Mutation Score
Tools Used: PiTest Tool

Procedure: For big projects, as we used, it is not possible to do it manually. It has to be automated and we used PiTest for this. It has plugins for major IDEs and we used Pitclipse for Eclipse IDE. To run this tool in Eclipse or any other IDE, it is essential to build the project, which is being tested, properly with all the test cases successfully running. After this, Pitclipse plugin can run and it will generate the PIT summary for the whole project down until the class level. It will even show the mutations done, active mutators and tests examined. At last, the total mutation score can be used to know how much effective the test suite of the project is. The major problem for this metric faced was the requirement that the project needs to be built properly with all the test cases running successfully and it took a lot of time and effort.

3.Metric 4: McCabe Complexity
Tools Used: JaCoCo

Procedure: Statement coverage and Branch coverage is calculated using JaCoCo (Java Code Coverage) tool. JaCoCo tool is a widely used tool that can generate reports in html, csv and xml format.
JaCoCo tool can be downloaded automatically from maven repositories by including JaCoCo plugin in the pom.xml file of the project. JaCoCo calculates linearly independent paths that a program's execution can take.
We used the Complexity data from the .csv file. This data is used to draw the correlation between Metric 1,2 and 4.
Average comexity is calculated as Total Complexity fivided by number of classes.

4.Metric 5: Code Churn
Tools Used: CLOC

*Procedure:*First, after selecting the project releases are checked on the Github.and then different releases in a zip format are downloaded.After unzipping the file , installation of a tool called CLOC is done.Finally we run the 5 versions of each project i.e. in total 25 versions using the command.
cloc - - diff versionname1 versionname2 - - out = metric5versionname1_versionname2.txt.
It will generate the text file stating the difference in lines of code in two versions of a project. Then finally from the output we checked for the lines of code that are changed, modified, added or deleted and calculated the code churn using formula:

changed lines of code/total lines of code.

5.Metric 6: Post-release Defect Density
Tools Used: JIRA, LocMetrics
Procedure: Post Release Defect Density is mainly used to find the number of bugs present in the software project for each version. To find the number of bugs for each version we need to export the JIRA reports from the issue tracking system by applying certain filters for each version. The filters used are Type and Version Number. By exporting the Jira reports, we can find the number of bugs

Table:3 Data collection for all six projects

Project	Size (LOC)	Statement coverage	Branch coverage	Mutation score	Complexity metric	Code churn	Post-release defect density
Apache commons Collection	126.68K	86%	81%	36%	12	0.013	0.233365
Apache commons Logging	9.63K	70%	75%	58%	7	0.016	2.659086
Apache commons Configuration	105K	87%	83%	85%	10	0.0118	0.093412
Apache Commons File Upload	14.56K	81%	77%	49%	6	0.804	2.480683
Apache commons Math	186K	92%	85%	79%	11	0.781	1.102035

5. CORRELATION BETWEEN METRICS

We have used spearman coefficient to estimate the correlation between metrics. Spearman's coefficient is a measure of statistical dependency between rankings of two variables. The range of this co-efficient is from -1 to 1, where -1 means strong and negative correlation, 1 means strong and positive correlation and if the value is near to zero the correlation is weak.

$$r_s = 1 - \frac{6\sum D^2}{n\left(n^2 - 1\right)}$$

Where 'd' is the difference between ranks of two observation and 'n' is number of observations.

To calculate the correlation between the metrics at class level we wrote a python script which will take input in csv format and generates the spearman correlation coefficient and scatter plot.

Correlation between 1,2 and 3:

The correlation between 1,2 and 3 is that, if the statement coverage and branch coverage is high, the mutation score also should be high which in turn shows that test suite is more effective and vice versa.

Correlation between 4 and 1,2:

Generally, if the Complexity is high it means that the project is difficult to maintain and test due to many linearly independent paths. This situation might lead to a need for Increased statement and branch coverage.

Correlation between 1,2 and 6:

The classes with less statement coverage and branch coverage are likely to have higher defect density because there are many chances to ignore some of the inputs which the code might produce erroneous behaviour. This effect might be noticed only after the release.

Correlation between 5 and 6:

This correlation between Metrics 5 and 6 is the average of density defect and code churn. Our basic hypothesis is that code that changes many times prerelease will likely have more post-release defects than code that changes less over the same period of time. Increase in relative code churn measures is accompanied by an increase in system defect density and more precisely we can also state that using relative values of code churn predictors is better than using absolute values to explain the system defect density.

6. CORRELATION RESULTS AND ANALYSIS

The purpose of metric analysis is to determine the current quality of the product or system, improve quality and predict nature immediately after the complete software development program

In this part of our report, we correlate the different metrics and provide detailed information about their results.

Co-relation Between Metric 1 (Statement Coverage) and Metric 4 (Complexity Code Coverage):

The Spearman Coefficients for all the projects (at class level) are written in the .xlsx file and the spearman Coefficient for all 5 projects at the project level is -0.7 which shows the string negative correlation between Metric 1 and 4.

Co-relation Between Metric 2 (Branch Coverage) and Metric 4 (Complexity Code Coverage):

The Spearman Coefficients for all the projects (at class level) are written in the .xlsx file and the spearman Coefficient for all 5 projects at the project level is -0.7 which shows the string, negative co-relation between Metric 2 and 4.

Fig. Scatter-Plot for Statement Coverage (%age)/100 Vs Complexity Code Coverage (%age)/100

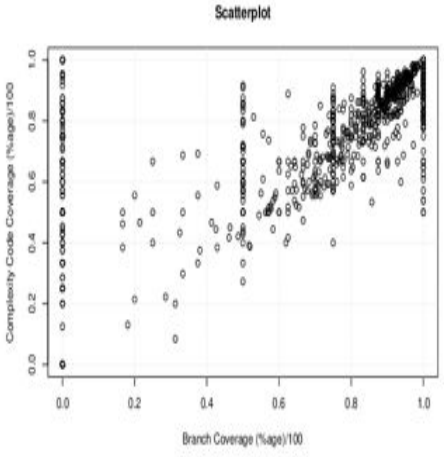
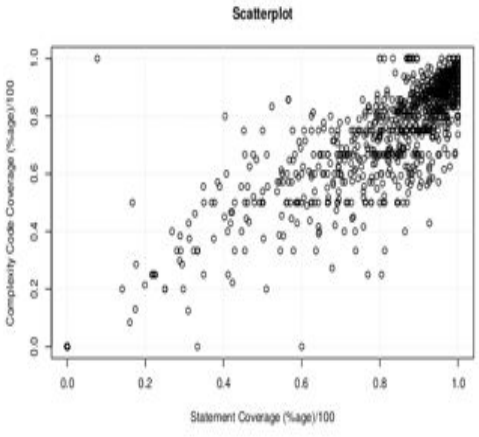


Fig. Scatter-Plot for Branch Coverage (%age)/100 Vs Complexity Code Coverage (%age)/100

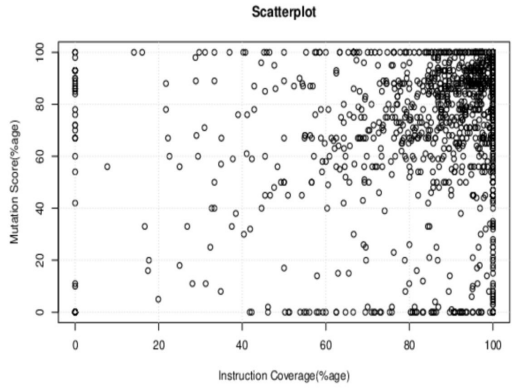


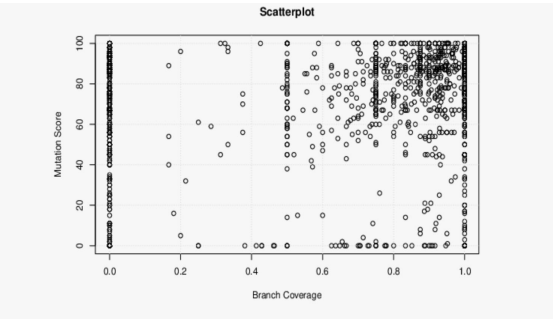
Below are the class level Spearman Coefficient for all the Projects:

Projects Name	**Spearman Coefficient for Metric 1 and 4**	**Spearman Coefficient for Metric 2 and 4**
Apache Commons Collections	0.920665321	-0.026222491
Apache Commons Logging	0.203338807	0.48358
Apache Commons Configuration	0.818194	-0.01075
Apache Commons File Upload	0.917260788	0.384615385
Apache Commons Math	0.807957484	0.058236547

Correlation between 1,2 and 3:

The Spearman Coefficients for all the projects (at class level) are written in the .xlsx file and the spearman Coefficient for all 5 projects at the project level is 0.28 which shows the weak, positive correlation between Metric 1 and 3 whereas the spearman Coefficient for all 5 projects at the project level is 0.06 which shows the weak, positive correlation between Metric 2 and 3 .



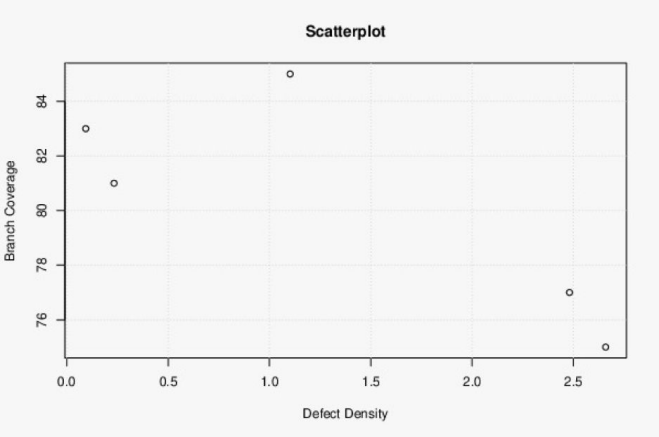


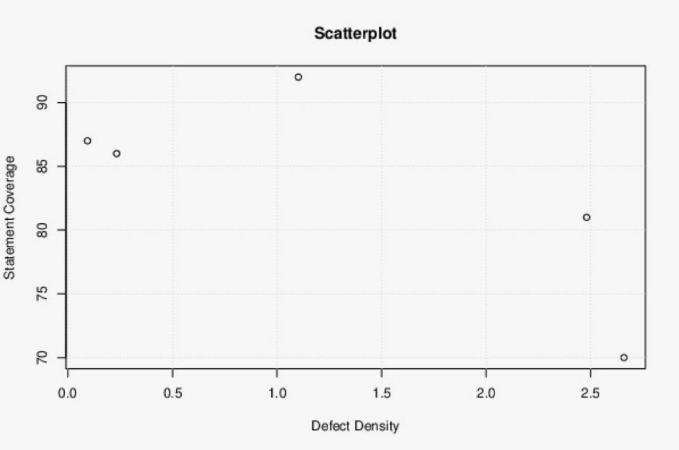
Co-relation Between Metric 2 (Branch Coverage) and Metric 6 (Defect Density):

The Spearman Coefficients for all the projects (at class level) are written in the .xlsx file and the spearman Coefficient for all 5 projects at the project level is -0.7 which shows the string, negative co-relation between Metric 2 and 4.

Co-relation Between Metric 1 (Statement Coverage) and Metric 4 (Complexity Code Coverage):

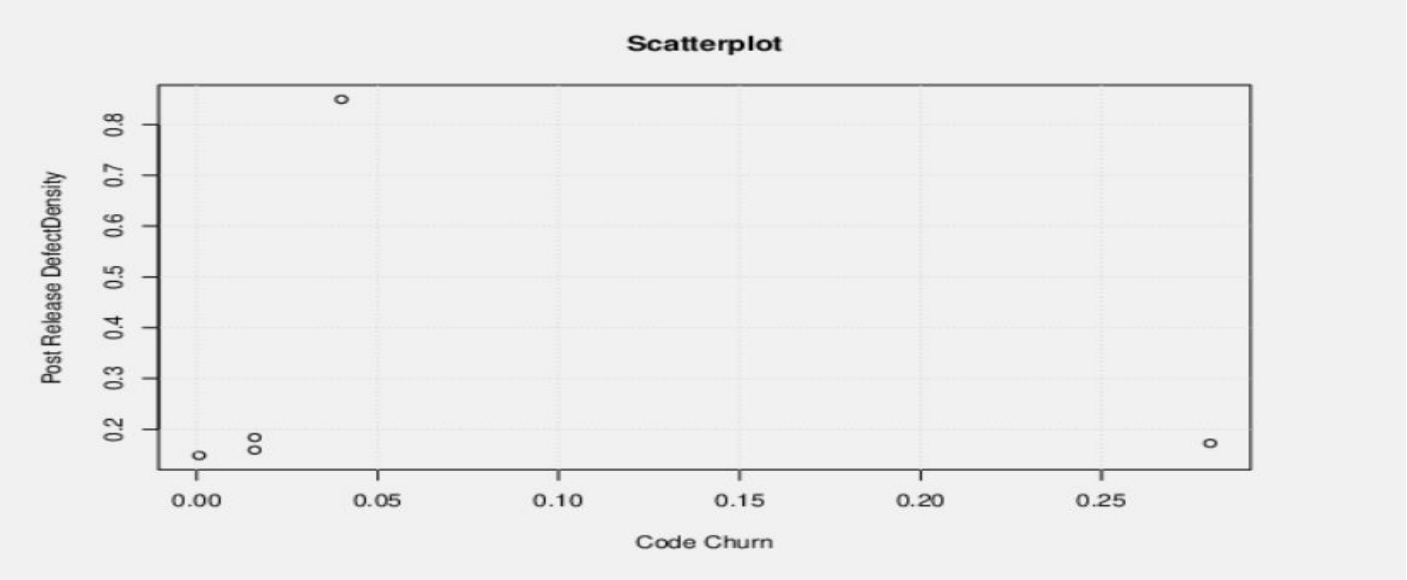
The Spearman Coefficients for all the projects (at class level) are written in the .xlsx file and the spearman Coefficient for all 5 projects at the project level is -0.7 which shows the string, negative co-relation between Metric 1 and 4.





Correlation between 5 and 6:

The Spearman Coefficients for all the projects (at class level) are written in the .xlsx file and the spearman Coefficient for all 5 projects at the project level is 0.61 which shows the string, positive correlation between Metric 5 and 6.



7. RELATED WORK

For the statement and Branch coverage, we have selected Jacoco tool to get both coverage metrics as it is the matured and easy to use when compared with 'cobertura' and 'Emma'[1][2].

PiTclipse is used to find the test suite effectiveness as it is the famous mutation testing tool and it is targeting the open source projects and supports recent version of Junit unlike MuJava tool [3].

We have used again the JaCoCo tool to get the Cyclomatic complexity. This tool provides detailed complexity in all levels which helps in correlation process. This plugin is usable in different platforms such as java, Android, etc.

For the fifth metric that is Code Churn, we compare the different versions of the project that were released in a given period of time. Then we generated the text file stating the difference in lines of code in two versions of a project. Then finally from the output we checked for the lines of code that are changed, modified, added or deleted and calculated the code churn using formula,

changed lines of code/total lines of code.

The last metric required a little amount of work to be done with computing the defect density for 5 versions of every project system that we used as the reference. The post -release Defect Density is the number of identified defects found during the operational phase per 1000 source line of code. It is calculated by the given formula.

8. CONCLUSION

Metrics play the most important key factor to manage and estimate most of the software activities like effort, productivity, cost and quality.

For analysing the dependencies correlation between the various quality / quantity attributes is recommended.

Statement coverage and branch coverage are always strongly and positively correlated. After analysing results for subject projects, we can observe that all the metrics are somehow correlated.

Code/Statement coverage is negatively correlated with mutation coverage and code complexity unlike with post release defect density with the exception of Apache Commons Logging.

The relation, that came after calculating between the metrics 1, 2 and 4, tell that the metrics are somehow weakly positively correlated.

When it come to Code churn and Post release Defect density, it can be seen that these are positively strongly correlated with each other according to our experiment work.

REFERENCES

[1]https://onlysoftware.wordpress.com/2012/12/19/code coverage-tools-jacoco-cobertura-emma-comparison-in-sonar/

[2] Michael Hilton,Jonathan Bell,Darko Marinov - A Large-Scale Study of Test Coverage Evolution http://www.cs.cmu.edu/~mhilton/docs/ase18coverage.pdf

[3] Paco van Beckhoven, Ana Oprescu, and Magiel Bruntink - Assessing Test Suite E↵ectiveness Using StaticMetrics https://pdfs.semanticscholar.org/f464/250c01a38411e58ba206b939a399534c506c.pdf?_ga=2.169420326.668200107.155546153 3-921356720.1555461533.

[4] Nachiappan Nagappan and Thomas Ball- Use of Relative Code Churn Measures to Predict System Defect Density https://www.st.cs.uni-saarland.de/edu/recommendation-systems/papers/ICSE05Churn.pdf

[5] Software reliability growth with test coverage.

https://ieeexplore.ieee.org/document/1044339

[] Sharma, R. (2014). TOOLS AND TECHNIQUES OF CODE COVERAGE TESTING. International Journal of Computer Engineering and Technology (IJCET), http://www.iaeme.com/MasterAdmin/UploadFolder/TOOLS%20AND%20TECHNIQUES%20OF%20CODE%20COVERAGE%20TESTING2/TOOLS%20AND%20TECHNIQUES%20OF%20CODE%20COVERAGE%20 TESTING2.pdf, [online] 5(9), pp.165-171.

[] Yang-Ming Zhu and David Faller- Defect-Density Assessment in Evolutionary Product Development: A Case Study in Medical Imaging https://ieeexplore.ieee.org/document/6253198

[] Relating Code Coverage, Mutation Score and Test Suite Reducibility to Defect Density https://ieeexplore.ieee.org/document/7528960

[] M. Moghadam and S. Babamir, "Mutation score evaluation in terms of object-oriented metrics", 2014 4th International Conference on Computer and Knowledge Engineering (ICCKE), 2014. Available: 10.1109/iccke.2014.6993419

[] Madi, A & Zein, O.K. & Kadry, Seifedine. (2013). On the improvement of cyclomatic complexity metric. Available at: https://www.researchgate.net/publication/288695710_On_the_i mprovement_of_cyclomatic_ complexity_metric, International Journal of Software Engineering and its Applications. 7. 67-82.

[] M. O. Elish and D. Rine. Design structural stability metrics and post-release defect density: An empirical study. In COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference, pages 1–8, Washington, DC, USA, 2006. IEEE Computer Society

[]https://www.tutorialspoint.com/software_testing_ dictionary/cyclomatic_complexity.htm