Software Measurement (SOEN-6611-2841-D)

Winter 2019

Team J

Analysis and Correlation of Metrics

Professor: Jinqiu Yang

Student Name	Student ID	Email address
Sravan Kumar Thumati	40070088	tumati.sravan@gmail.com
Naren Morabagal Somasekhar	40082567	narenms96@gmail.com
Adarsh Aravind	40082585	arvindadarsh891@gmail.com
Kundana Gangam	40085658	kundanagangam@gmail.com
Emmanuel Ambele	40050295	etambele@gmail.com

Link to the replication package in GitHub: https://github.com/AdarshArvind/SOEN6611-2184-D

Sravan Kumar Thumati
Dept. of Software Engineering
Concordia University
Montreal, Canada
tumatisravan@gmail.com

Naren Morabagal Somasekhar
Dept. of Software Engineering
Concordia University
Montreal, Canada
narenms96@gmail.com

Adarsh Aravind
Dept. of Software Engineering
Concordia University
Montreal, Canada
arvindadarsh891@gmail.com

Kundana Gangam
Dept. of Software Engineering
Concordia University
Montreal, Canada
kundanagangam@gmail.com

Emmanuel Ambele
Dept. of Software Engineering
Concordia University
Montreal, Canada
etambele@gmail.com

Abstract: **In order to measure the quality and effectiveness of the software, we need to consider some qualitative metrics. In this work, we considered six different metrics and correlated them to determine how related the projects are. We chose 5 open source java projects as subject programs, namely, Apache Commons Collections, Apache Commons Configuration, Apache Commons IO, Apache Commons FileUpload and JFreeChart. We have estimated the impact of metric on a software system with the results obtained for this study. Spearman Correlation Coefficient is used to evaluate the correlation between the variables.**

Keywords- Statement Coverage, Branch Coverage, Mutation Testing, Cyclomatic Complexity, Backlog Management Index (BMI), Defect Density.

I. INTRODUCTION

Software quality is one of the major factors 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. In this research work, we have considered 5 large open-source projects which are developed using Java programming language. At first, we measure statement and branch coverage, to know what percentage of the statements, and branches are being executed by the defined test suits. Next, we perform mutation testing to know the mutation coverage of the system. For both statement and branch coverage, we used EclEmma JaCoCo library, which is free code coverage library for Java. For mutation testing, we used an eclipse plugin called Pitclipse. Our fourth metric was McCabe Code Complexity, which was calculated using MetricsReloaded plugin in IntelliJ. It calculates the cyclomatic complexity of each non-abstract method. Metric five was about Fix backlog and backlog management index, which is a count of reported problems that remain at the end of a certain period (e.g. a week, a month). This metric helps to manage the backlog of open and unresolved problems. Last metric was to calculate the defect density in open-source projects which we had considered. Post Release Defect Density is the measure of code quality per unit, identified after a particular version is released. Jira reports were very helpful in determining the defect count of a particular release. And also, we used SciTools to calculated LOC for each version of a project.

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 4 and 1&2, Metric 1&2 and 6, Metric 5 and 6.

II. SUBJECT PROJECT

a) Apache Commons Configuration:

Link:https://commons.apache.org/proper/commons-configuration/

Description: We have worked on different versions of this project. Base version (2.4) 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

b) Apache Commons Collections:

Link:https://commons.apache.org/proper/commons-collections/

Description: Base version (4.4.2) that we have selected had 132 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.

c) JFreeChart:

Link:https://sourceforge.net/projects/jfreechart/files/1.%20JFreeChart/

Description: Base version (1.5.0) that we have selected had 228 KLOC. A free Java chart library. JFreeChart supports pie charts (2D and 3D), bar charts (horizontal and vertical, regular and stacked), line charts, scatter plots, time series charts, high-low-open-close charts, candlestick plots, Gantt charts, combined plots, thermometers, dials and more. JFreeChart can be used in client-side and server-side applications.

d) Apache Commons IO:

Link:https://commons.apache.org/proper/commons-io/

Description: We have worked on different versions of this project. Base version (2.6) that we have selected has 50 KLOC. Commons IO is a library of utilities to assist with developing IO functionality.

e) Apache Commons FileUpload:

Link:https://commons.apache.org/proper/commons-fileupload/

Description: We have worked on different versions of this project. Base version (1.4) that we have selected has 7.7 KLOC. The Commons FileUpload package makes it easy to add robust, high-performance, file upload capability to your servlets and web applications.

III. METRICS

1) Statement Coverage:

Statement coverage is used to quantify the total number of lines in the source code that have been successfully executed. Statement coverage is a white box testing technique, 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. Through statement coverage we can identify the statements executed and where the code is not executed because of blockage.

$A = \text{Total no. of Statements Excercised}$

$B = \text{Total no. of Executable Statements in a Program}$

$$Statement\ Coverage = \frac{A}{B} * 100$$

2) Branch Coverage

Branch Coverage is also called as all-edges coverage. It 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.

$A = \text{No. of Decision Statements Exercised}$

$B = \text{Total No. Decision Outcomes}$

$$Branch\ Coverage = \frac{A}{B} * 100$$

3) Test Suite effectiveness:

We have used mutation coverage metric which is a code coverage technique to define the test suite effectiveness of subject programs. 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. Based on the percentage obtained by the tools of mutation testing, developers and testers can improve their test suite which in turn improve the quality of the software system.

$A = \text{Dead Mutants}$

$B = \text{Total no. of Non} - \text{Equivalent Mutants}$

$$Mutation\ Score = \frac{A}{B} * 100$$

4) Cyclomatic Complexity

Cyclomatic complexity is a measure of the number of distinct execution paths through each method. This can also be considered as the minimal number of tests necessary to completely exercise a method's control flow. In practice, this is 1 + the number of if's, while's, for's, do's, switch cases, catches, conditional expressions, &&'s and ||'s in the method.

5) Fix Backlog and Backlog Management Index

This metric is used commonly for maintenance analysis. The calculation of the Backlog Management Index (BMI) of a project utilizes the defect arrival rate and availability of fixed problems rate for that project which can be said to be the count of reported problems that remain at the end of a certain period (e.g. a month). This metric is used to manage the backlog of open and unresolved problems of a project and this can be visualised using graphs or charts (e.g. Trend Chart). BMI is calculated using the formula mentioned below,

$A = \text{No. of defects closed during the month}$

$B = \text{Total no. of defects arrived during the month}$

$$BMI = \frac{A}{B} * 100$$

If BMI is larger than 100, it means the backlog is reduced. If BMI is less than 100, then the backlog increased. The aim is to achieve a backlog larger than 100.

6) Post Release Defect Density:

Quality of the software project is the most important from the stakeholder's perspective (e.g. Users, Researchers). If the project is developed from the scratch it's easy to calculate the defect density, but for the multiple releases and incremental developed project, it's always challenging.

Post release defect density measures the code quality per unit identified after a version is released, it measures the defects relative to the software size expressed as lines of code or function point. It requires waiting for the field

defect to be deployed. The defects that are identified are fixed as soon as possible or at least by the next version.

Defect Density is defined as the ratio between number of defects found in the software during a specific period of operation or development to the size of the software. Defect density is calculated per thousand lines of code.

$A = \text{Defect Count}$

$B = \text{Size of release (KLOC)}$

$$Defect\ Density = \frac{A}{B}$$

IV. DATA COLLECTION

Metric 1 and 2 Collection:

To calculate metric 1 and 2, we have used JaCoCo plugin which is the best available tool for code coverage, as it provides accurate results. The projects which we selected had test cases already built into it. So, it was easy to run JUnit and get the results. We can export result in several formats such as HTML, CSV, and more. We can export the coverage details in class, package and project levels.

Metric 3 Collection:

To estimate the effectiveness of test suite, we use Mutation score as quality attribute. To calculate the mutation score

we have used Pitclipse plugin in eclipse. PIT takes the byte code which is generated when the code is compiled and applies several configurable set of mutation operators. PITclipse generates the result in HTML format.

Running the tests:

PIT runs your unit tests against the mutated code automatically. Before running the tests, PIT performs a traditional line coverage analysis for the tests, then uses this data along with the timings of the tests to pick a set of test cases targeted at the mutated code.

This approach makes PIT much faster than previous mutation testing systems such as Jester and Jumble, and enables PIT to test entire code bases, rather than single classes at a time.

Metric 4 Collection:

To calculate this, we have used MetricsReloaded plugin in Intellij IDE. This plugin provides a good collection of metrics about our project. Once installed, the metrics can be run for the current file, a specific file or module, or for an entire project making it very flexible in terms of what is to be analysed. Also, it allows us to create custom metrics and also to define threshold values.

Metric 5 Collection:

This metric requires the use of reported project defects to calculate the BMI based on a time frame, for this project the time frame used was calendar months. These defects were collected manually on Jira issue tracking and Github. Every issue reported had the date which it was created, updated, and closed, hence it was possible to know which month a bug was created and closed if it's applied. This information was used to manually collect every bug report for each month used for the calculation of the BMI. The table below shows the BMI for the projects in the report.

Metric 6 Collection:

We have calculated the defect count based on number of confirmed and closed issues mentioned in the Jira reports. We obtained the defect count from issue tracking systems using Jira reports. We calculated the size of release (KLOC) using Understand (Sci-tools) software. It is a customizable integrated development environment which permit static code analysis using metric tools. Once installed, several metrics can be calculated for the project. We compute KLOC as,

$$KLOC = \frac{LOC}{1000}$$

Projects	Metric 1	Metric 2	Metric 3	Metric 4	Metric 5	Metric 6		
	Statement Coverage (%)	Branch Coverage (%)	Test Suite Effectiveness %	Complexity	BMI	Number of Defects	KLOC	Post - Release Defect Density
Commons Collections v4-4.2	89	78	43	11,666	21.35	15	107.272	0.14
Commons Configuration v2.4	94	87	85	8,568	0	5	105.01	0.05
Commons FileUpload v1.4	82	74	70	767	22.925	4	7.77	0.51
Commons IO v2.6	89	81	59	4,851	0	30	50.185	0.59
JFreeChart v1.5.0	72	46	33	21,227	12.9125	22	228.112	0.096

Table 1: Metric Data of Subject Projects

Apache Commons IO Versions	Statement Coverage	Branch Coverage	Mutation Coverage	Cyclomatic Complexity	BMI	Defect Density
2.0	88	79	82	2935	0	0.64
2.1	89	79	83	3035	0	0.57
2.2	89	81	85	3287	0	0.14
2.3	89	81	85	3349	0	0.29
2.4	90	82	85	3426	0	2.09
2.5	87	79	86	2860	0	0.82
2.6	90	81	83	3986	0	0.59

Table 2: Metric Data of different versions of Apache Commons IO

V. 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.

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

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 Backlog Management Index based on selected version of the projects within the time period of 16 months. Generally, BMI and Defect density should be positively correlated.

VI. CORRELATION RESULTS AND ANALYSIS

Software metrics play major role in analysing and improving the quality of the software project. The main intention of calculating the metrics is to monitor and control the software processes which helps us in decision making and for project maintenance. The tables below will summarize the metrics discussed.

All metric shows some impact (negative or positive) on another metric either directly or indirectly. In this section we are going to correlate different metrics and elaborate the results. To calculate correlation, we used spearman correlation coefficient. We have written python script that calculates the Spearman coefficient which takes input as CSV files. The results are explained in two ways. Firstly, we correlate the data of base version of all subject projects. Secondly, correlation of different version project Apache Commons IO.

Correlation between 1,2 and 3:

Spearman coefficient between statement coverage and mutation score is **0.66** and **0.7** for branch coverage and mutation score. We noticed that there is strong and positive correlation between code coverage and mutation score.

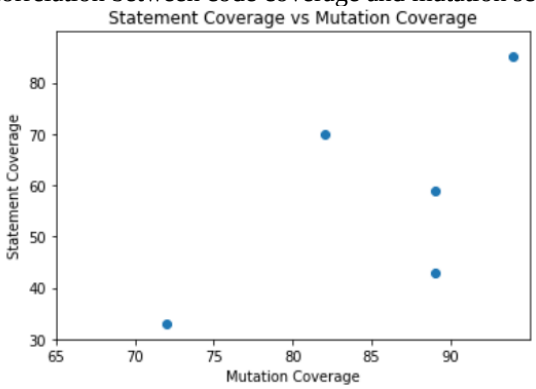

Figure 1: Correlation between Statement coverage and Mutation Coverage

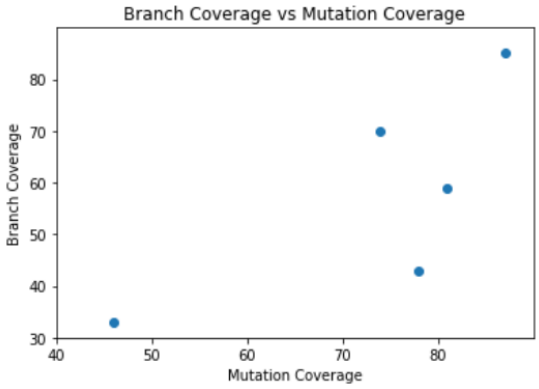

Figure 2: Correlation between Branch coverage and Mutation Coverage

Correlation between 4 and 1,2:

Spearman coefficient between 4 and 1 is **0.86**. The correlation coefficient between 4 and 2 is **0.76**. The values show that there is strong and positive correlation between 4 and 1,2.

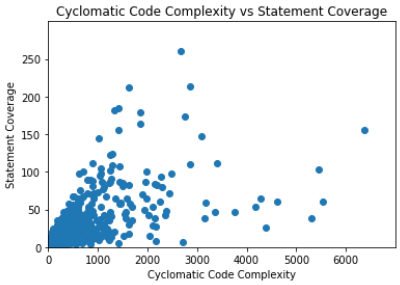

Figure 3: Correlation between Complexity and Statement coverage

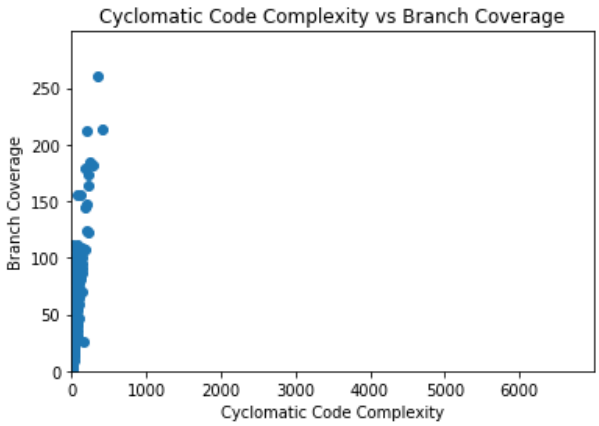

Figure 4: Correlation between Complexity and Branch Coverage

Correlation Between 1,2 and 6:

Spearman coefficient between statement coverage and Defect Density is **-0.2** and **-0.09** for branch coverage and Defect Density. We noticed that there is weak and negative correlation between code coverage and defect density.

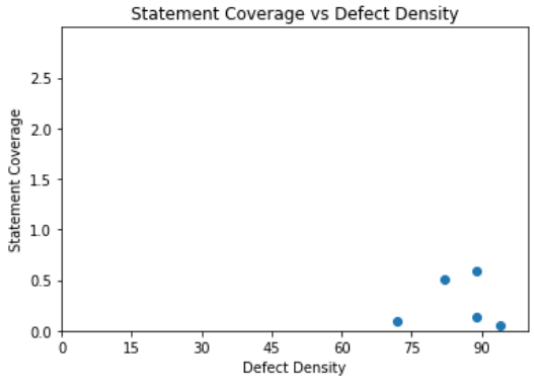

Figure 5: Correlation between statement Coverage and Defect Density

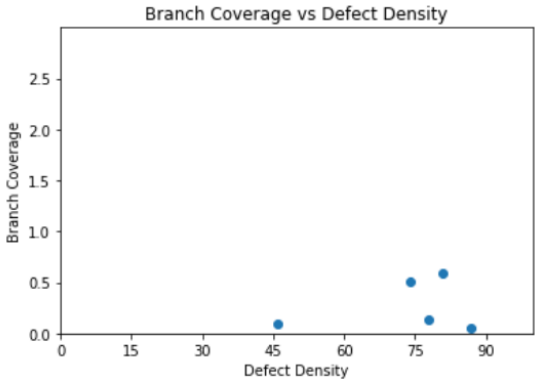

Figure 6: Correlation between Branch Coverage and Defect Density

Correlation between 5 and 6:

Spearman coefficient between 5 and 6 metrics is **0.20** which means there is week and positive correlation between the metrics.

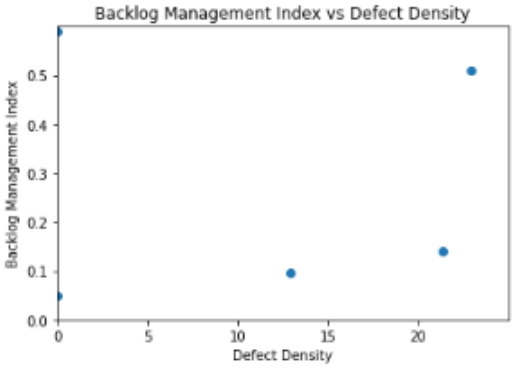

Figure 7: Correlation between Backlog Management Index and Defect Density.

Correlation between different versions of Apache Commons IO:

The data for all metrics is collected for seven different versions of Apache Commons IO and are correlated as shown below.

- **Correlation between 1,2 and 3:**

The spearman coefficient for statement coverage and mutation score is **-0.16**, which shows weak and negative correlation, spearman coefficient for branch coverage its **0.26** which is weak and positive correlation.

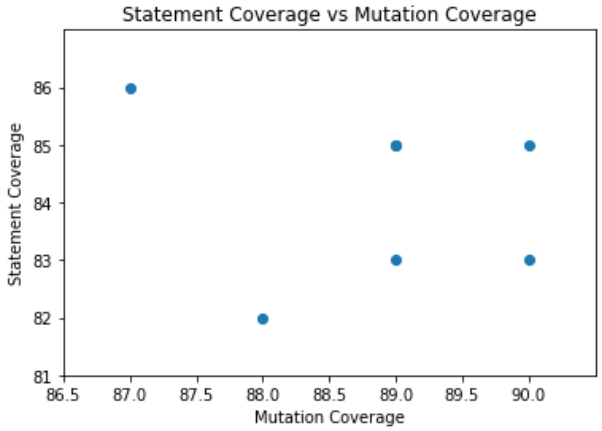

Figure 8: Correlation between statement Coverage and Mutation Coverage

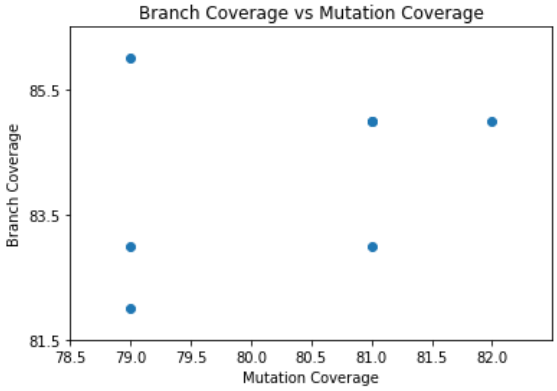

Figure 9: Correlation between Branch Coverage and Mutation Coverage

- **Correlation between 4 and 1,2:**

Spearman coefficient between cyclomatic complexity and statement coverage is **0.89**. The correlation coefficient between cyclomatic complexity and branch coverage is **0.75**. The Correlation is calculated at version level. The values show that there is strong and positive correlation between 4 and 1,2.

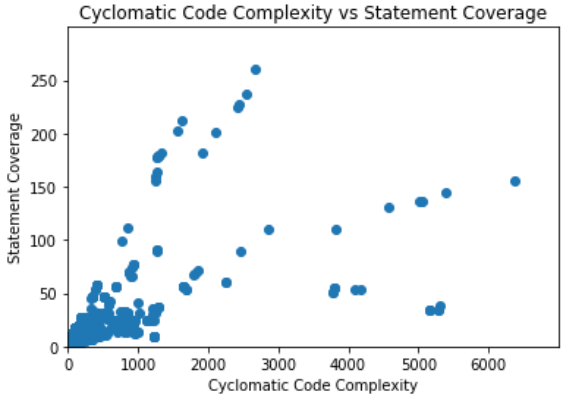

Figure 10: Correlation between Cyclomatic Complexity and Statement Coverage

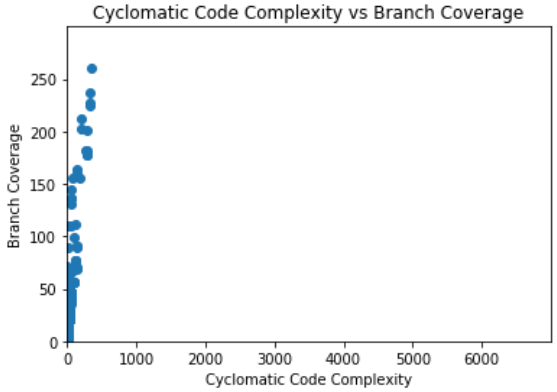

Figure 11: Correlation between Cyclomatic Complexity and Branch Coverage

- **Correlation Between 1,2 and 6**

Spearman coefficient between statement coverage and Defect Density is **-0.02** and **zero** for branch coverage and Defect Density. We noticed that there is weak and negative correlation between code coverage and defect density.

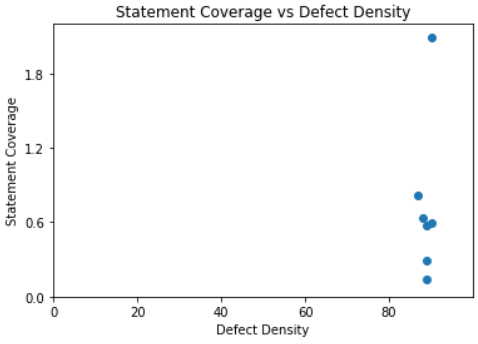

Figure 12: Correlation between Statement Coverage and Defect Density

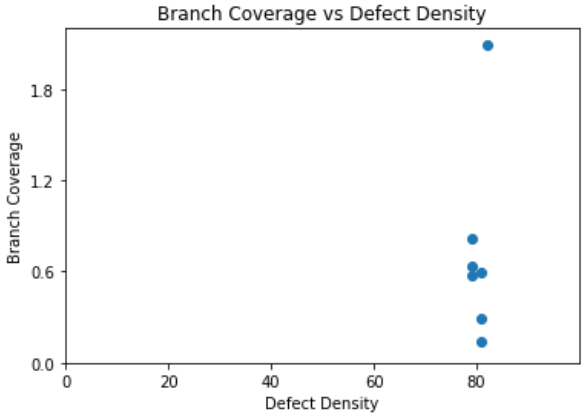

Figure 12: Correlation between Branch Coverage and Defect Density

- **Correlation between 5 and 6**

Spearman coefficient is zero since arrived defects are not closed during the specific time period (2 months). Since BMI value is zero, there exist no correlation between the variables.

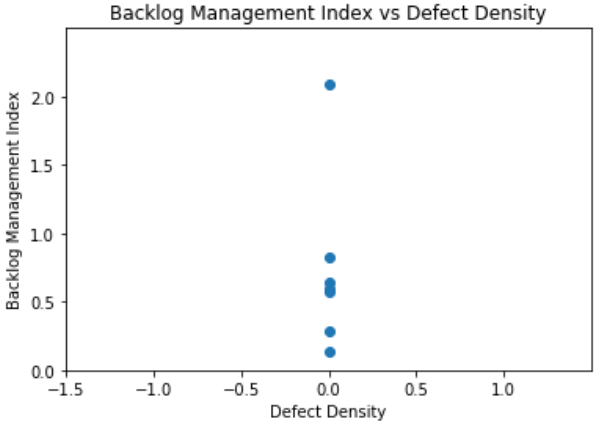

Figure 12: Correlation between Backlog Management Index and Defect Density

VII. RELATED WORK

We have selected Jacoco tool to get statement coverage and branch coverage metrics as it is the matured and easy to use when compared with 'cobertura' and 'Emma'[1][2]. PiTclipse is the famous mutation testing tool as it is targeting the open source projects and supports recent version of Junit unlike MuJava tool [3].

We have used MetricReloaded tool to get the Cyclomatic complexity. This tool provides with 250 different metrics, thus increasing the scope of future work. 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.,[4]

Fix Backlog and Backlog Management Index measures the amount of Backlog in a project at the end of each time frame, in other words, It is a simple count of reported problems that remain at the end of each time frame, the

time frame being each month or each week. Fix Backlog and Backlog Management Index has been used widely in academia and the industry to provide meaningful information for managing the maintenance process of a project. In the report we used the Fix Backlog and Backlog Management Index to get information about management process of the five projects indicated earlier. Every information needed to calculate the Fix Backlog and Backlog Management Index for each report was obtained on the issue tracking website-Jira, and GitHub.

Based on the perspective, we have different ways to calculate the size of release. Suppose if we consider design perspective, we must count number of function points, from requirements perspective number of use-case points are considered.

We chose Sci-tools to calculate size of the release because its effective way to calculate the number of lines [5].

VIII. CONCLUSION

Metrics are most important factors to manage and estimate most of the software activities like effort, productivity, cost and quality.

For analysing the dependences correlation 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 coverage is positively correlated with mutation coverage and code complexity unlike with post release defect density. Backlog management index is also positively but weakly correlated with Post release defect density.

The scenario of correlation when considering the versions for Apache Commons IO, we noticed an exception in correlation between code coverage and mutation coverage. Here, the correlation between projects is positive which doesn't hold good if we consider the same for different versions of Apache Commons IO.

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 Static Metrics https://pdfs.semanticscholar.org/f464/250c01a38411e58ba206b939a399534c506c.pdf?_ga=2.169420326.668200107.1555461533-921356720.1555461533

[4] Bo Wang - An Android Studio Plugin for Calculating and Measuring Code Complexity Metrics in Android Applications https://mdsoar.org/bitstream/handle/11603/2459/TF2015Wang_Redacted.pdf%3Fsequence%3D1

[5] 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

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

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

[8] 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

[9] 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_improvement_of_cyclomatic_ complexity_metric, International Journal of Software Engineering and its Applications. 7. 67-82.

[10] Stephen H. Kan, Metrics and models in software quality engineering. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ©2002 ISBN:0201729156

[11] 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

[12] DANA H HALABI1 & ADNAN SHAOUT (June'16) "MUTATION TESTING TOOLS FOR JAVA PROGRAMS – A SURVEY", Available at: https://www.researchgate.net/publication/303401556_MUTATION_TESTING_TOOLS_FOR_JAVA_PROGRAMS_-_A_SURVEY

[13] L. Vinke (June 2011) "Estimate the post-release Defect Density based on the Test Level Quality". Available at: https://research.infosupport.com/wp-content/uploads/2017/08/MasterThesis-LammertVinke-Final.pdf