Software Measurement (SOEN-6611-2191-A)

Summer 2019

Team R

Analysis and Correlation of Metrics

Professor: Jinqiu Yang

Student Name	Student ID	Email address
Anusha Keralapura Thandavamurthy	40102962	kt.anusha21@gmail.com
Jaswanth Banavathu	40080737	jaswanthbanavathu@gmail.com
Arvind Korchibettu Adiga	40105178	adiga1993@gmail.com
Kranthi Kumar Sankuru	40102793	kranthikumar9922@gmail.com
Manushree Mallaraju	40082236	manushreemallaraju@gmail.com

Link to the replication package : https://github.com/Anushakt/Software_Measurement_TeamR

Anusha Keralapura Thandavamurthy
Dept. of Software Engineering
Concordia University
Montreal, Canada
kt.anusha21@gmail.com

Jaswanth Banavathu
Dept. of Software Engineering
Concordia University
Montreal, Canada
jaswanthbanavathu@gmail.com

Aravind Korchibettu Adiga
Dept. of Software Engineering
Concordia University
Montreal, Canada
adiga1993@gmail.com

Kranthi Kumar Sankuru
Dept. of Software Engineering
Concordia University
Montreal, Canada
kranthikumar9922@gmail.com

Manushree Mallaraju
Dept. of Software Engineering
Concordia University
Montreal, Canada
manushreemallaraju@gmail.com

Abstract: **By considering some qualitative metrics we can measure the quality and effectiveness of particular software. In this present work, we considered six different metrics and to determine the relation in projects we correlated them. We took 5 open source java projects, namely, Apache Sling, Apache Commons Lang, Apache Commons Collections, Apache Commons Math and JFreeChart. From the results obtained, we have estimated the impact of each metric on a software system. The correlation among the variables is measured using Spearman Correlation Coefficient.**

Keywords- Code Coverage, Branch Coverage, Test suite effectiveness (Mutation Testing), McCabe Cyclomatic Complexity, Backlog Management Index (BMI) and Post-release Defect Density

I. INTRODUCTION

In the development phase of a project, Software Quality is considered as one of the major factors. It can be measured by using various software metrics based on their usage and requirements. The software development team is analyzing the improvements which are required to process, by documenting all the changes in the previous versions of the software. Software quality aids in determining the correctness of the system and also improves all the activities associated with the software in SDLC. In this research work, we have considered 5 open source projects which are developed using Java programming language. In the Initial state, we calculated statement and branch coverage by knowing the percentage of branches and statements that were executed by some definite test suites. Later, in order to know the mutation coverage of the system we calculated the mutation score which is carried out in mutation testing. EclEmma JaCoCo library, a free code coverage library in Java is used for calculating both statement and branch coverage. For mutation testing, the mutation score is directly obtained by executing the project in IntelliJ IDE. Our Fourth metric was McCabe Cyclomatic Complexity, which was calculated by using the reports that were generated for first and second metric. This code complexity is done for the non- abstract methods in the project. Metric five was about fixing the backlog and backlog management index. This metric is the count of reported problems in a single project that remain at the end of a certain time period (e.g., a week, a month, a year). All the backlog of open and unresolved problems is managed by this metric. Our last metric was Post Release Defect Density, which is the code quality per unit, identified after a version is released. For knowing the defect count of a release, Jira reports were used.

After calculating these six metrics, we performed Spearman Correlation Coefficient for determining the correlation analysis between them. We performed correlation analysis between Metric 1&2 and 3, Metric 1&2 and 4, Metric 1&2 and 6 and Metric 5 and 6.

II. SUBJECT PROJECT

a) Apache Sling:

Link: https://projects.apache.org/project.html?sling

Description: We have worked on different versions of this project. Base version (2.20.0) that we have chosen has 3.5 million LOC. Apache Sling is a framework for RESTful web applications based on extensible content tree. It maps the HTTP request URLs to content resources based on the request's path, extension and selectors

b) Apache Commons Collections:

Link:
https://projects.apache.org/project.html?commons-collections

Description: We have chosen 4.3 as its base version which has 132k LOC. It became a standard for collection handling in Java because it has many powerful data structures that enhance the development of many significant Java application.

c) JFreeChart:

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

Description: Base version (1.50) that we have selected had 317k LOC. It is a free Java chart library that supports bar charts (horizontal, regular, stacked and vertical), line charts, scatter plots, pie charts (2D and 3D), Gantt charts, combined plots and more. It can be used in client-side as well as in server-side applications.

d) Apache Commons Lang:

Link:https://projects.apache.org/project.html?commons-lang

Description: Base version 3.8 that we have considered had 80.7k LOC. This project provides the methods required for manipulation of its core classes which the standard Java libraries fail to provide. The methods include basic numerical methods, object reflection and String manipulation methods**.**

e) Apache Commons Math:

Link:https://projects.apache.org/project.html?commons-math

Description: Base Version (3.6.1) that we have selected had 186k LOC. This project is a library of lightweight, self-contained mathematics and statistics components addressing the most common practical problems not immediately available in the Java programming language.

III. METRICS

1. Code Coverage:

Code coverage, also known as statement coverage, is used to measure the total number of lines that have been successfully executed in the source code. It is a white box testing technique, which ensures the quality in source code by executing a statement at least once. In this metric, all test cases are written to ensure that all the statements in the program are executed. It covers only true conditions and is recorded as 100 % if every statement in the program is executed at least once.

$$Code\ Coverage = \frac{Total\ no.of\ statements\ Exercised}{Total\ no.of\ Executable\ Statements}$$

2. Branch Coverage:

In order to know the test suite effectiveness of subject programs, we used mutation coverage metric. In mutation testing, we create a mutant in a program and verify that the particular mutant is killed by the test cases or not. If the mutation score is 1 i.e. 100%, then we can deduce that all the mutants are killed by test suite and vice versa. The testers and developers can improve their test suite by considering the percentage obtained in the mutation testing, hence improving the overall quality of the software system.

$$Mutation\ Score = \frac{No.\ of\ Killed\ Mutants}{Total\ no.of\ Mutants}$$

3. McCabe Cyclomatic Complexity:

McCabe 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. All the selection statements i.e. if, if else and switch cases, all conditional statements i.e. for, while, do while and all the conditional expressions i.e. && and || are calculated in this metric.

4. Backlog Management Index (BMI):

BMI serves as the one of the metrics needed for maintenance analysis. In BMI, we consider the defect arrival rate and availability of fixed problems i.e. count of reported problems that remain at the end of a certain period (e.g. a month) for a single project. This metric is used to manage the backlog of open and unresolved problems of a project. The formula is

$$BMI = \frac{No.of\ defects\ closed\ during\ the\ month}{No.of\ defects\ arrived\ during\ the\ month}$$

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

5. Post- release Defect Density:

Software quality acts as an important factor in the development of software. The calculation of defect density is always challenging because it is calculated for multiple releases of a developed project. 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. The defects which are identified are fixed as soon as possible or at least by the next version. Defect Density is the ratio between number of defects found in the software during a specific period of operation or development to the size of the software. It is calculated per thousand lines of code.

$$Defect\ Density = \frac{No.of\ defects\ in\ a\ version}{Size\ of\ release\ (LOC)}$$

IV. DATA COLLECTION

Metric 1 and 2 Collection:

To Compute both the metrics 1 & 2, we have installed Jacoco Plugin in the POM file, which is a best tool for statement coverage and branch coverage to show results in accuracy. The test cases were already built in the apache projects itself that we have chosen. So, it was easy to execute Junit and generate the results. Then, we exported our results in the formats of CSV, HTML and more. And we can export the code coverage details in all levels ie Class, Package and Project levels.

Metric 3 Collection:

To estimate Test suite effectiveness, we use Mutation Testing as a quality attribute because it is fast, actively developed, actively supported and easy to use. We performed mutation testing to run all unit tests against all automatically modified versions on all selected apache projects and calculated the mutation score. For that, we added some dependencies in the POM.xml file i.e. we used the pit test plugin in the POM file, and we run the projects and got the results in the formats of HTML, CSV and more.

Metric 4 Collection:

To calculate Code Complexity, we have used Jacoco plugin. This is a neat plugin that gives a good collection of metrics about our projects. Once executed, it gives the output in both HTML format and CSV file. We have calculated the average code complexity based on the data which we got in CSV file.

Metric 5 Collection:

For this metric, the use of number of reported defects of the project is essential to calculate BMI on the basis of time

frames, which is calendar months. For this, we used JIRA issue tracking and Github and using all the kinds of filters, we generated the number of closed, created, updated issues in the respective CSV file. This information is used to calculate every bug report for each month to calculate BMI.

Metric 6 Collection:

To Calculate Post release defect density, we need number of defects in the project and LOC. we have calculated the number of defect count by using Jira issue tracking and Github. We have used small queries to get the data from the site and exported the results to CSV files.

We can get the reports by using them and therefore, we can count the number of confirmed and closed defects. To calculate the size of the release (KLOC), we have used Understand (Sci- tools) software, which combines an Eclipse with an array of static analysis tools. Statistical information about the project and other entities like lines of code complexity can be obtained metric reports. Sci-tools provides several ways to gather the metric data such as project metric report, class metric report, class OO metric report. We have used class metric report we get the LOC of each class and we have done the addition of those to get the total LOC of the entire project. Baes on the collected data we have calculated the defect density of the project.

We have computed KLOC as below:

$$KLOC = \text{Lines of Code} / 1000$$

Table 1: Metric Data of Subject Projects

Projects	Statement Coverage (%)	Branch Coverage (%)	Test Suite Effectiveness (%)	Complexity	BMI	Number of Defects	KLOC	Post Release Defect Density
JfreeChart 1.5.0	54	46	33	35	0	307	317	0.9684
Apache Commons Math v3.6.1	87	52	79	18	66.67	40	186	0.215
Apache Commons Lang 3.8	94	90	86	33	32.43	12	80.7	0.1486
Apache Commons Sling 2.20.0	44	54	48	11	91.39	10	3500	0.0028
Apache Commons Collections 4.4.3	86	81	42	15	58.33	5	132	0.037

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

Apache Commons Collections	Statement Coverage (%)	Branch Coverage (%)	Test Suite Effectiveness (%)	Complexity	BMI	Number of Defects	KLOC	Post Release Defect Density
Apache Commons Collections 3.2.2	82	70	40	15	56.54	8	226	0.035
Apache Commons Collections 4.0	84	77	40	14	48.23	5	206	0.024
Apache Commons Collections 4.1	84	78	43	15	55	7	277	0.025
Apache Commons Collections 4.2	86	81	42	15	58.33	5	132	0.037
Apache Commons Collections 4.4.3	86	81	42	15	58.33	5	132	0.037

V. CORRELATION BETWEEN METRICS - I (INCLUDING RESULTS AND ANALYSIS)

In this section we will be correlating between different metrics and analyze the results. We have used Spearman coefficient to evaluate the correlation. This value provides a positive/negative and strong/weak correlation result. We have used a script in R language, using the tool RStudio, which takes input from CSV file. The first part shows correlation results between different projects and the second part contains correlation results between different versions of the same project.

The statistical dependency of two variables is measured by Spearman's coefficient. The correlation of matrices was determined by using Spearman's coefficient. the value of the coefficient falls between -1 and 1, where -1 defines strong and negative correlation and 1 defines strong and positives and the values close to 0 defines that the correlation is weak.

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

d = Difference between the ranks of two observations

n = number of observations

R script was used to measure the correlation between the metrices which requires an input in csv and the spearman correlation coefficient is generated leading to a scatter plot.

Correlation between 1,2 & 3:

Correlation shows that as Statement and Branch Coverage goes high, Mutation score also goes high, which shows that test suite effectiveness is high.

The spearman coefficient between Statement Coverage and Mutation score was found to be **0.57** and the value between Branch Coverage and Mutation score was found to be **0.67**. This indicates that there is a strong positive correlation between code coverage and mutation score.

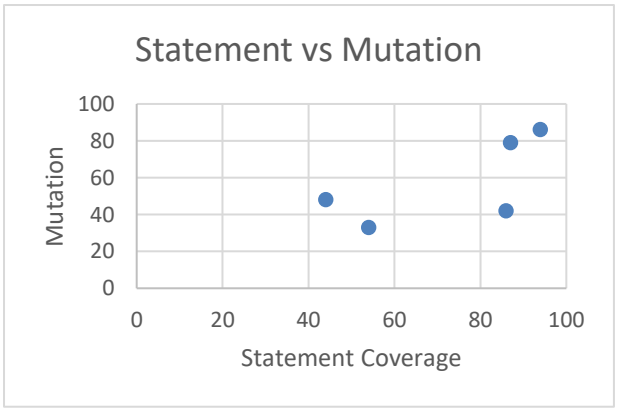

Figure 1: Correlation between Statement coverage and Mutation coverage

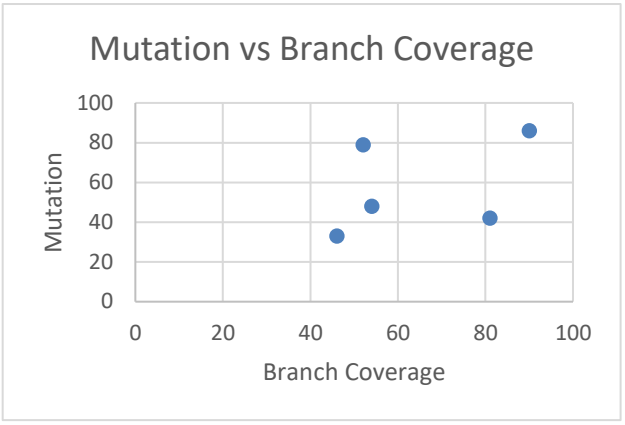

Figure 2: Correlation between Branch coverage and Mutation coverage

Correlation between 1,2 and 4:

Complexity refers to the number of independent paths that a code can take while execution. A higher complexity is associated with higher maintenance and testing, therefore higher branch and statement coverage. The Spearman coefficient between Statement Coverage and Complexity is **0.83** & the coefficient value between Branch Coverage and Complexity is **0.7**, which indicates a strong positive correlation between the metrics.

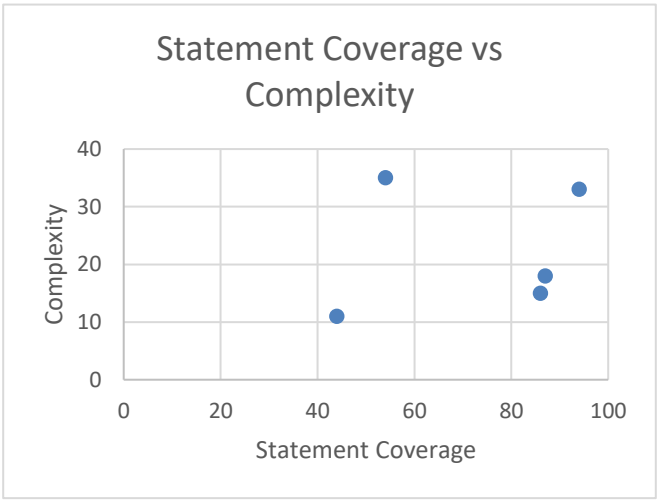

Figure 3: Correlation between Statement coverage and Complexity

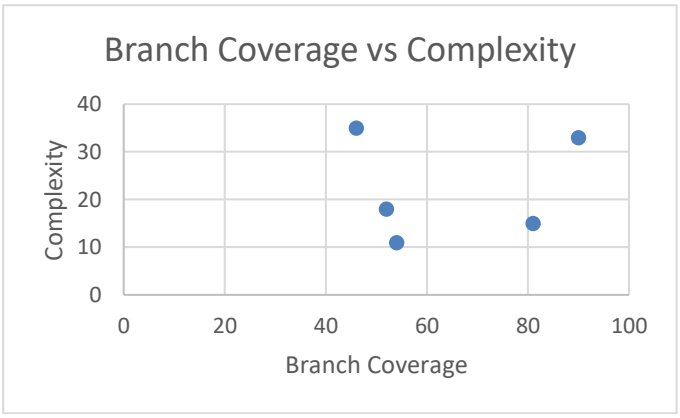

Figure 4: Correlation between Branch coverage and Complexity

Correlation between 1,2 and 6:

Defect density refers to the presence of bugs after a release. Higher defect density is associated with lower statement and branch coverage since some inputs may be missed out and may produce unexpected results.

The spearman coefficient between Statement Coverage and Post-release Defect Density is **-0.4** and the value between Branch Coverage and Defect Density is **-0.06**. This indicates a weak/medium and negative correlation between the two metrics, coverage and defect density.

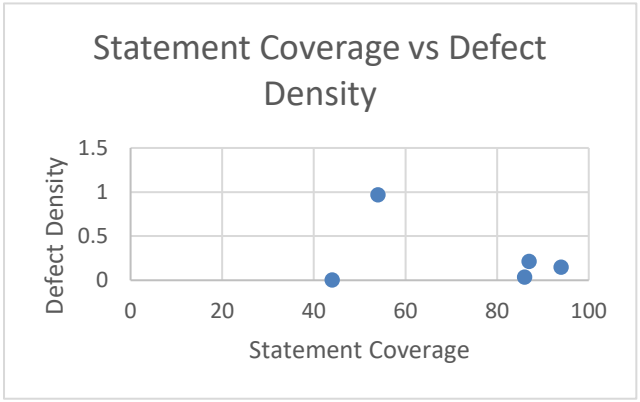

Figure 5: Correlation between Statement coverage and Defect Density

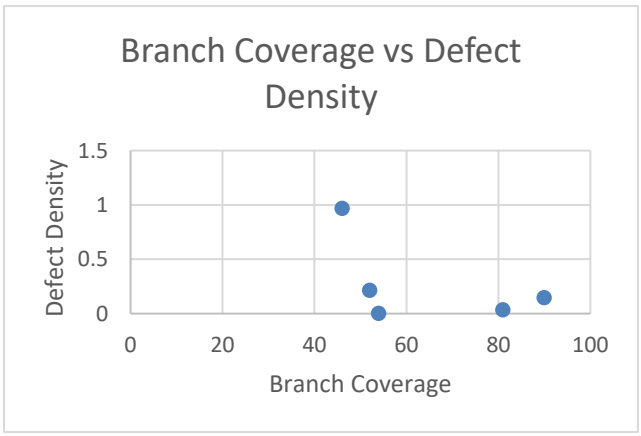

Figure 6: Correlation between Branch coverage and Defect Density

Correlation between 5 and 6:

The correlation between Backlog Management Index (BMI) and Defect density is based on the data collected over a period of 6 months. The correlation is the average of BMI and defect density and is correlated positively.

The Spearman coefficient between BMI and defect density was found to be **0.34**, which shows a medium positive correlation.

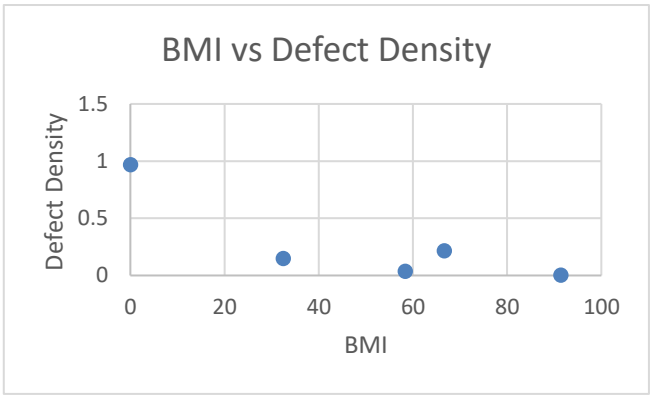

Figure 7: Correlation between BMI and Defect Density

CORRELATION BETWEEN METRICS – II (INCLUDING RESULTS AND ANALYSIS, FOR DIFFERENT VERSIONS OF APACHE COMMONS COLLECTIONS)

The data has been collected for 5 different versions of Apache Commons Collections.

Correlation between 1,2 and 3:

The spearman coefficient between statement coverage and mutation score is **-0.1**, which shows medium/weak and negative correlation and the value between branch coverage and mutation score **0.18** which shows a medium/weak and positive correlation.

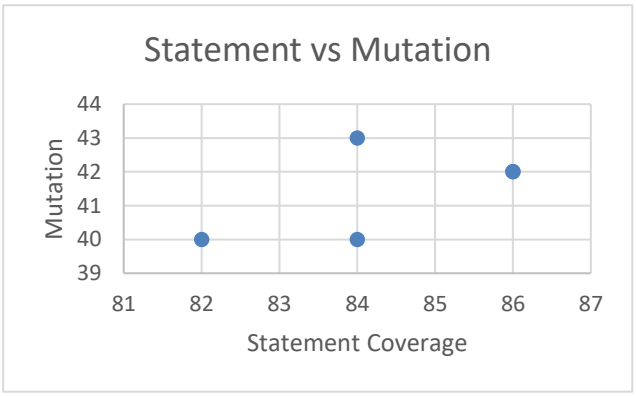

Figure 8: Correlation between Statement Coverage and Mutation Coverage

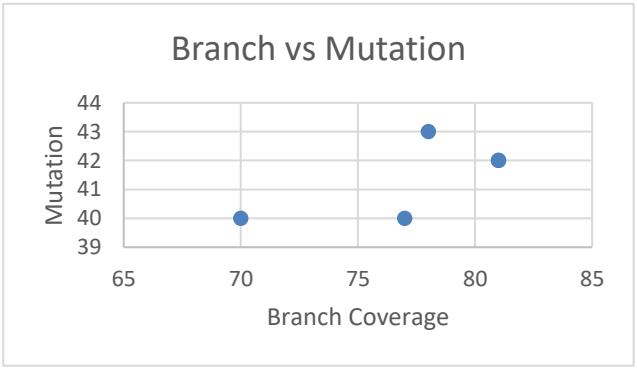

Figure 9: Correlation between Branch and Mutation Coverage

Correlation between 1,2 and 4:

The spearman coefficient between statement coverage and code complexity is **0.79** and the value between branch coverage and code complexity is **0.65**, which indicates a strong and positive correlation.

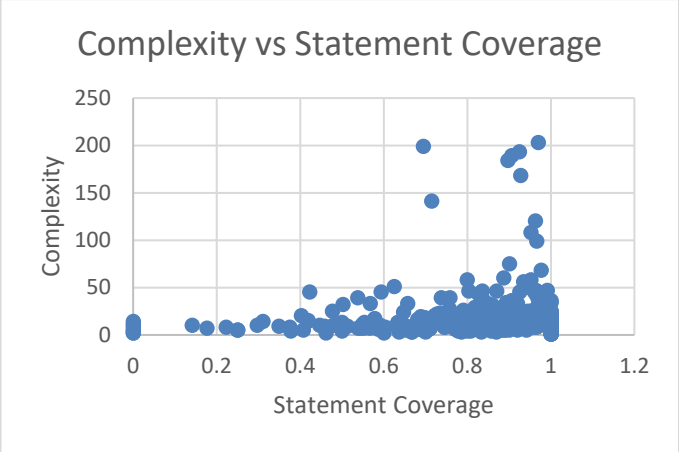

Figure 10: Correlation between Statement Coverage and Complexity

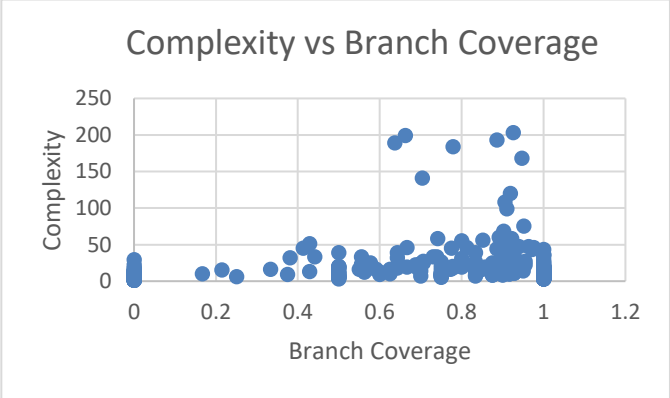

Figure 11: Correlation between Branch Coverage and Complexity

Correlation between 1,2 and 6:

The spearman coefficient value between statement coverage and defect density is **-0.04** and the value is **-0.02** between branch coverage and complexity. This shows a weak and negative correlation between the two metrics.

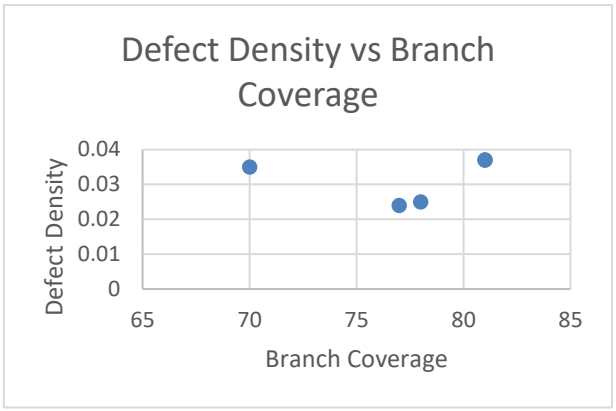

Figure 12: Correlation between Branch Coverage and Defect Density

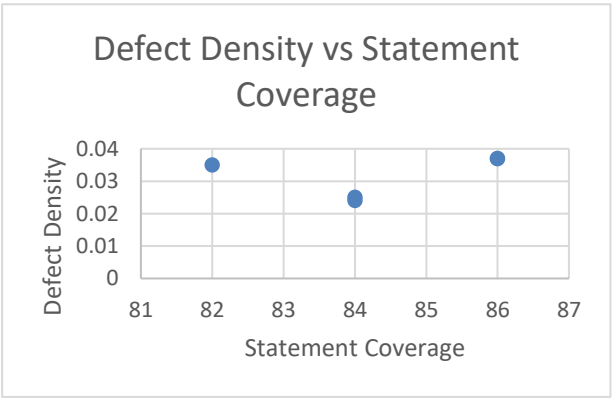

Figure 13: Correlation between Statement Coverage and Defect Density

Correlation between 5 & 6:

The spearman coefficient value between BMI and Defect Density is found to be **0.012**, which shows a medium positive correlation.

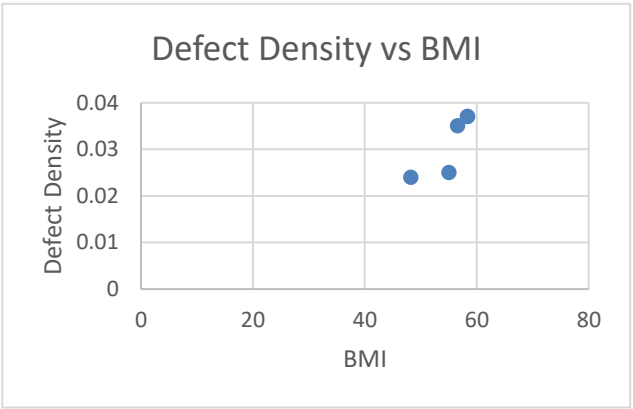

Figure 14: Correlation between BMI and Defect Density

VII. RELATED WORK:

We have used Jacoco tool to calculate coverage metrics. The advantage with this tool is that it generates the results in both html and csv formats from which we can easily calculate the metrics. To calculate the number of lines of code, we used Scitools. Average complexity was found by averaging the results of total complexity obtained in the Jacoco csv file.[3][6].

Test Suite effectiveness is determined by using PIT test in IntelliJ tool. This tool is convenient to use as it provides a mutation score on a class level.[3].

Backlog Management Index is associated with the backlogs present in a development environment during a particular time period. It is an important metric for determining maintenance issues for projects. The data required for BMI, like resolved problems and created issues during a time frame is obtained from GitHub and Jira tracker.[2].

Defect Density is a measure of number of defects in a project of a particular size. The information about number of defects in the project was determined using the Jira tracker.[1].

VIII. CONCLUSION :

In the development of particular software, Metrics play a crucial role in the management and estimation of all the software activities like productivity, effort, cost and quality.

For the analysis of dependencies between the metrics, correlation is preferred. Some of the correlation between the different metrics is that Branch coverage and Statement coverage are always positively and strongly correlated while the Code coverage bears a positive relation with code complexity and mutation coverage but not with defect density. The last two metrics i.e. BMI and Defect density has a positive value but a weak/medium correlation. Hence, the correlation analysis between these metrics gives us a definite understanding about the relationship between the metrics and within the projects.

REFERENCES:

[1] Rahmani, C. (2019). A Study on Defect Density of Open Source Software. [online] Available at: https://www.researchgate.net/profile/Deepak_Khazanchi/publication/221635599_A_Study_on_Defect_Density_of_Open_Source_Software/links/0c9605249a82ab8af0000000.pdf

[2] Ahamed, A. (2019). Backlog Clearance Planning Index - A Proactive Approach to Backlog Management. [online] Available at: https://independent.academia.edu/AshfaqAhamedMohammed

[3] THUNG, F., SINGH, P. and LO, D. (2019). Code Coverage and Test Suite Effectiveness: Empirical Study with Real Bugs in Large Systems. [online] Mysmu.edu. Available at: http://www.mysmu.edu/faculty/davidlo/papers/saner15-coverage.pdf

[4] Wiedmann, J., Wiedmann, J., & profile, V. (2011). The mess that is m2e connectors. Grumpyapache.blogspot.com. Retrieved 1 April 2019, from http://grumpyapache.blogspot.com/2011/08/mess-that-is-m2e-connectors.html

[5] objectledge/maven-extensions. (2019). GitHub. Retrieved 1 April 2019, from https://github.com/objectledge/maven-extensions

[6] EclEmma - JaCoCo Java Code Coverage Library. (2019). Eclemma.org. Retrieved 1 April 2019, from https://www.eclemma.org/jacoco

[7] Use of relative code churn measures to predict system defect density - IEEE Conference Publication. (2019). Ieeexplore.ieee.org. Retrieved 1 April 2019, from https://ieeexplore.ieee.org/document/1553571

[8] CLOC -- Count Lines of Code. (2019). Cloc.sourceforge.net. Retrieved 1 April 2019, from http://cloc.sourceforge.net/

[9] PIT Mutation Testing. (2019). Pitest.org. Retrieved 2 March 2019, from http://pitest.org/