SOEN-6611 Software Measurements

Instructor: Dr. Jinqiu Yang

Group: Team J

Team Members:

Name	Student ID	E-mail ID
Suruthi Raju	40084709	suruthi77@gmail.com
Pranoti Mulay	40129435	opranoti@gmail.com
Avinash Damodaran	40078258	avinashdamu323@gmail.com
Shalin Rohitkumar Patel	40088004	shalinpatel610@gmail.com
Niralkumar Hemantkumar Lad	40080612	niralhlad.concordia@gmail.com

Replication Package: https://github.com/pranotimulay/soen6611-measurement-metrics

Date of submission: April 10, 2020

A Study of Correlation Analysis between various Software Measurement Metrics

Pranoti Mulay
Department of Computer Science & Software Engineering,
Concordia University,
Montreal, Canada
opranoti@gmail.com

Shalin Rohitkumar Patel
Department of Computer Science & Software Engineering,
Concordia University,
Montreal, Canada
shalinpatel610@gmail.com

Niralkumar Lad
Department of Computer Science & Software Engineering,
Concordia University,
Montreal, Canada
niralhlad.concordia@gmail.com

Avinash Damodaran
Department of Computer Science & Software Engineering,
Concordia University,
Montreal, Canada
avinashdamu323@gmail.com

Suruthi Raju
Department of Computer Science & Software Engineering,
Concordia University,
Montreal, Canada
suruthi77@gmail.com

***Abstract-* This paper illustrates a study of the correlation between various software metrics, namely, Statement Coverage, Branch Coverage, Mutation Score, Software Defect Density, Maintenance Effort Estimation Model, and Cyclomatic Complexity, applied on various open-source projects by Apache Software. Data visualization and Pearson Correlation Coefficient have been used to determine the correlation between these metrics.**

Keywords-CLOC, JaCoCo, AMEffMo, PIT, McCabe, Complexity, Defect Density, Coverage, DLOC, Mutation Testing.

I. INTRODUCTION

A software metric is a measure of software characteristics which are measurable or countable.Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.[8] This paper calculates various metrics like Statement coverage, Branch coverage, Cyclomatic complexity, Mutation score, AMEffMo and Software defect density using different tools such as EclEmma, PitClipse, and CLOC. The common metrics that you might see mentioned in your coverage reports include: Function coverage: gives how many of the functions defined have been called. Statement Coverage: gives how many of the statements in the program have been executed.Line Coverage: gives how many lines of source code have been tested. Here we are using Branch and Statement Coverage. Cyclomatic complexity is a software metric used to indicate the complexity of a program. Software maintenance effort involves a broad spectrum of activities like - optimization, error correction, deletion of discarded features and enhancement of existing features. Defect density is the number of defects detected in a software component during a defined period of development/operation divided by the size of the software
This paper also covers the correlation between different metrics. Correlation analysis is a statistical method used to evaluate the strength of relationship between two quantitative variables. A high correlation means that two or more variables have a strong relationship with each other, while a weak correlation means that the variables are hardly related[1]. We analyse the correlation between 1) Branch and statement Coverage with Software defect density. 2) Branch and Statement Coverage with Cyclomatic complexity 3) Maintenance Effort Model and Software Defect Density 4) Branch and Statement Coverage with Mutation Score.
Rest of the paper is divided into following sections: Section II describes the project and metric identification, Section III describes methodology which consists of tools are used, how data was collected & analyzed, Section IV gives a detailed view on the the correlation between different metrics that has been studied above Section V and Section VI focuses related work and conclusion of this paper. Section VII gives the references for this paper.

II. PROJECT AND METRIC IDENTIFICATION

This section describes the open source project used for the assignment and the corresponding metrics involved for identifying software measurement and Quality analysis.

A. PROJECT IDENTIFICATION

Various Open source projects are taken for our course study to perform metrics calculation and correlation analysis for different metrics. In our course study we took four apache projects to conduct a detailed analysis on each and every metrics and their relations between various metrics.

1) Apache Commons Logging

Apache Logging package is an ultra-thin bridge between different logging implementations. A library that uses the commons-logging API can be used with any logging implementation at runtime. Commons-logging comes with support for a number of popular logging implementations, and writing adapters for others is a reasonably simple task.

Project Link: https://commons.apache.org/proper/commons-logging/[2]

Source Code: https://github.com/apache/commons-logging

2) Apache Commons Lang

The Apache Commons Lang is used as a helper class for the utilities in the java.lang API.. The library acts as a host for providing various functionalities such as String manipulation methods, basic numerical methods, object reflection, concurrency, creation and serialization, and System properties .[3]

Project Link: http://commons.apache.org/proper/commons-lang/

Source Code: https://github.com/apache/commons-lang

3) Apache Commons Math

The Commons Math is a library of lightweight, self-contained mathematics and statistics components addressing the most common problems not available in the Java programming language or Commons Lang. The Guiding principles are Real-world application use cases determine development priority and This package emphasizes small, easily integrated components rather than large libraries with complex dependencies and configurations.[4]

Project Link: http://commons.apache.org/proper/commons-math/

Source Code: https://github.com/apache/commons-math

4) Apache Commons Net

Apache Commons Net library implements the client side of many basic Internet protocols. The purpose of the library is to provide fundamental protocol access, not higher-level abstractions. Therefore, some of the design violates object-oriented design principles. Our philosophy is to make the global functionality of a protocol accessible when possible, but also provide access to the fundamental protocols where applicable so that the programmer may construct his own custom implementations[4]

Project Link: http://commons.apache.org/proper/commons-net/

Source Code: https://github.com/apache/commons-net

B. METRIC IDENTIFICATION

Metrics are the data elements that are collected regarding the software program, measurement, qualified checks and outcomes. This section demonstrates various metrics which have been identified and used for the analysis and their working. These metrics are calculated for various versions and projects.

1) Branch Coverage

Branch coverage is a requirement that, for each branch in the program (e.g., if statements, loops), each branch has been executed at least once during testing. That leads to every branch can lead to either tTrue or False which ensures that there is no abnormal behaviour.

Branch Testing = (Number of decisions outcomes tested / Total Number of decision Outcomes) x 100%

2) Statement Coverage

Statement coverage is a white box testing technique, which involves the execution of all the statements at least once in the source code. It is a metric, which is used to calculate and measure the number of statements in the source code which have been executed. It can also be used to check the quality of the code and the flow of different paths in the program.

Statement coverage = No of statements Executed/Total no of statements in the source code x 100

3) Mutation Score

Mutation Testing is a type of software testing where we mutate certain statements in the source code and check if the test cases are able to find the errors. It is a type of White Box Testing which is mainly used for Unit Testing. The changes in the mutant program are kept extremely small, so it does not affect the overall objective of the program.[5]

The goal of Mutation Testing is to assess the quality of the test cases which should be robust enough to fail mutant code. This method is also called a Fault-based testing strategy as it involves creating a fault in the program.

Mutation Score = (Killed Mutants / Total number of Mutants) * 100

If mutation score= 0% the test cases are not written correctly on the contrary mutation score=100% means that all the faults are recognized completely.

4) McCabe Cyclomatic complexity

Cyclomatic complexity is a quantitative measure of the number of linearly independent paths through the program’s source code. Cyclomatic complexity is used as a benchmark to compare two different source code. The program with high cyclomatic complexity is more error-prone and requires more understanding of testing. It also helps us in determining the number of test cases that will be required for complete branch coverage.

Cyclomatic complexity is calculated with the help of the number of edges(E), the number of nodes(N) and the number of connected points (P).

Cyclomatic Complexity = E – N + 2P

Cyclomatic complexity can also be determined with the help of number of control predicate (D):

Cyclomatic Complexity = D + 1

In general, a low McCabe complexity is good to have, a higher complexity (>10) indicates that the method is complex.

5) Adaptive Maintenance Effort Model (AMEffMo)

This metric is used to build a model for estimating adaptive maintenance effort. As per paper[reference 9], a model called Adaptive Maintenance Effort Model (AMEffMo) is to predict adaptive maintenance effort in terms of person-hours

“A typical estimation model is derived using regression analysis on data collected from past software projects”. They used 70% of data to build the model and the remaining 30% to validate it. There are two approaches: Multiple Regression and Simple Regression Here we use Simple Regression. We calculate

Maintenance Effort by following steps Step 1: Identify metrics which affect estimation effort Step 2: Perform simple regression. Step 3: In Regression, use datapoint collected from data source and use least square method to produce the following model: E = 78 + .01DLOC [6]

6) Software Defect Density:

Defect Density is defined as the number of defects per size of the software or application area of the software. [7]

Defect Density = Total number of defects / Total Number of Volume

Or

Defect Density = Total no. of defects/KLOC

The average number of defects in a section or per KLOC of a software application is bug or defect density. The higher the bug density, the poorer the Quality.

III. METHODOLOGY

A. TOOL IDENTIFICATION

B. DATA COLLECTION

This section will demonstrate how tools are used for data generation for a project. Every tool has different dependencies, and tools are run for data generation.

1) EclEmma (JaCoCo)

EclEmma is a free Java code coverage tool for Eclipse and is based on the JaCoCo code coverage library. This tool helps to generate the statement coverage, branch coverage and cyclomatic complexity of the project. EclEmma plugin can be installed from the Eclipse marketplace. Once the tool is integrated, we can run the tool to get the above metric data and can be exported as CSV format.

CLASS	INSTRUCTION_MISSED	INSTRUCTION_COVERED
Range.ComparableComparator	0	19
ThreadUtils.NamePredicate	0	36
CharRange	2	235
CharSequenceUtils	11	282
StringEscapeUtils.CsvUnescaper	0	93
SerializationException	0	16
RegExUtils	3	95
StringUtils	9	7104

2) Pitclipse (Pitest)

Pitest is the mutation testing system and provides the test coverage for java and JVM. It has an eclipse plugin which is easy to integrate and run the mutation testing. Once the pit is integrated, various dependencies are to be added to the project to run the mutation testing.

Steps to integrate dependencies to the project:

- Add the plugin to pom.xml

```
<plugin>
    <groupId>org.pitest</groupId>
    <artifactId>pitest-maven</artifactId>
    <version>LATEST</version>
 </plugin>
```

If your project has JUnit 5 test cases you have to update the above dependency by

```
<plugin>
  <groupId>org.pitest</groupId>
  <artifactId>pitest-maven</artifactId>
  <version>1.5.0</version>
  <dependencies>
    <dependency>
      <groupId>org.pitest</groupId>
      <artifactId>pitest-junit5-plugin</a
      <version>0.12</version>
    </dependency>
  </dependencies>
</plugin>
```

- Once the dependency is added you can run the mutation testing by terminal or command line by going to the project folder and typing the following command

```
mvn org.pitest:pitest-maven:mutationCoverage
```

Mutation testing will start and the HTML report will be generated after the testing is finished. There are few tested cases which need to be ignored as they need configuration from the server level or system level.

3) CLOC

This tool does need any configurations at the project level. You can install the plugin depending on the operating system you are using. It's easy to install and use. After installing you can run the command bypassing the file or folder path.

Metric 6:

$ cloc “path to folder or file”

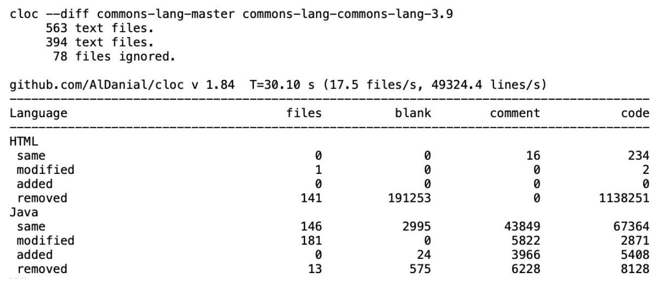
```
cloc commons-lang-master
     567 text files.
     560 unique files.
      46 files ignored.

github.com/AlDanial/cloc v 1.84  T=11.12 s (47.0 files/s, 133519.7 lines/s)
-------------------------------------------------------------------------------------------
Language                          files          blank        comment           code
-------------------------------------------------------------------------------------------
HTML                                142         191266             16        1138487
Java                                340          14422          55899          78363
XML                                  29            409            533           3813
Maven                                 1             33             38            910
CSS                                   1            127              0            436
Markdown                              2             36              0            186
Velocity Template Language            1             21             31             87
Groovy                                1             12             22             81
YAML                                  2              9             28             32
INI                                   3              0              0             15
Bourne Shell                          1              0              2              2
-------------------------------------------------------------------------------------------
SUM:                                523         206335          56569        1222412
-------------------------------------------------------------------------------------------
```

Metric 5:

$ cloc [options] <file(s)/dir(s)/git hash(es)> | <set 1> <set 2> | <report files>

--diff <set1> <set2>

--csv



```
cloc --diff commons-lang-master commons-lang-commons-lang-3.9
     563 text files.
     394 text files.
      78 files ignored.

github.com/AlDanial/cloc v 1.84  T=30.10 s (17.5 files/s, 49324.4 lines/s)
-------------------------------------------------------------------------------------------
Language                                 files          blank        comment           code
-------------------------------------------------------------------------------------------
HTML
 same                                        0              0             16            234
 modified                                    1              0              0              2
 added                                       0              0              0              0
 removed                                   141         191253              0        1138251
Java
 same                                      146           2995          43849          67364
 modified                                  181              0           5822           2871
 added                                       0             24           3966           5408
 removed                                    13            575           6228           8128
```

C. DATA ANALYSIS

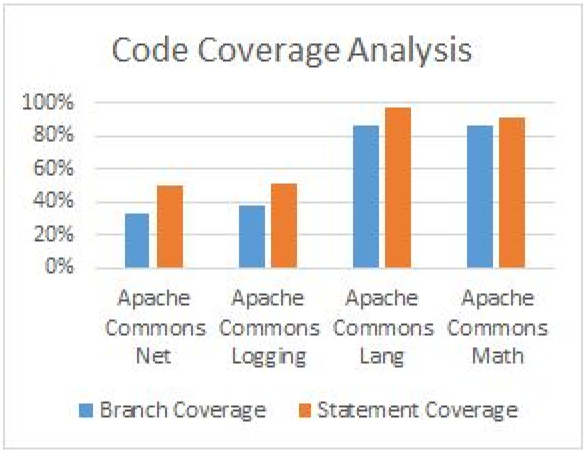
We perform a general analysis on the data obtained from the various metrics stated above. The data obtained is not seen to be normally distributed. Data Analysis is an important process in order to obtain a trend in the data values to visualize the results.

Step 1: Perform the data gathering. Gather data for all the metrics for chosen versions of the given open source projects using the specified tools. For metric 5 and 6, additional versions of the projects need to be included in the data collection step.

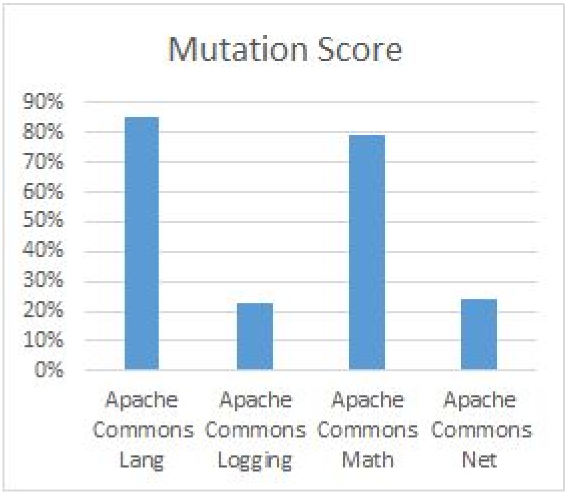
Step 2: Rationalize the data. For each metric, aggregate the data from all the projects. Aggregation mechanisms vary according to the type of data and logic applied.

Step 3: Plot graph of Project Vs the data values in Excel.

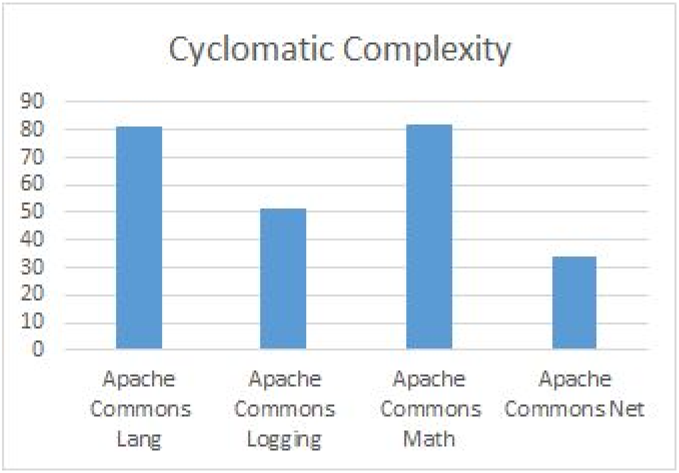
Below various types of graphs demonstrate the data population in each metric for all the given projects and provides a more acceptable and realizable form of obtained data.



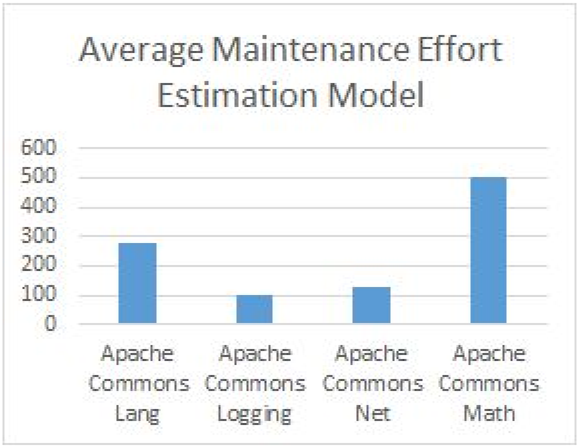
Graph 1: Percentage of Branch and Statement Coverage for the given projects



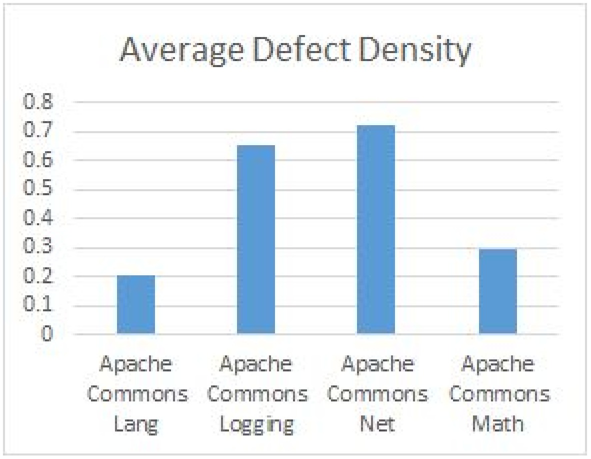
Graph 2: Mutation Score in percentage (extracted from tool) for the given projects



Graph 3: Average Cyclomatic Complexity calculated as (Complexity Covered / (Complexity Covered + Complexity Missed))



Graph 4: Average Maintenance Effort Estimation Model for the given projects



Graph 5: Average Defect Density for the given projects

IV. RESULTS

For analysis, it is important to extrapolate the obtained data and carry out correlation to comprehend the effects of the given metrics on one another. The process to vindicate the data is crucial, as the data is irrational. It would have required a good deal of effort to rationalize the data, which would have deviated us from our prime objective of the project. We have used Pearson's Correlation Coefficient (PCC) measures the statistical relationship, or association, between two continuous variables. It is known as the best method of measuring the association between variables of interest because it is based on the method of covariance. It gives information about the magnitude of the association, or correlation, as well as the direction of the relationship. The formula to calculate PC manually is given as below,

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[n\Sigma x^2 - (\Sigma x)^2][n\Sigma y^2 - (\Sigma y)^2]}}$$

However, we have used readily available functions in Excel to obtain the coefficient value and the scatter plots.

Evaluation

Correlation depends upon the trend of the values of variable X being increased or decreased with correspondence to the respective increase or decrease in the other variable Y.

Degree of correlation in case of PCC, is evaluated as follow,

Perfect: If the value is ± 1, then it is said to be a perfect correlation.

High: If the coefficient value lies between ± 0.50 and ± 1, then it is said to be a strong correlation.

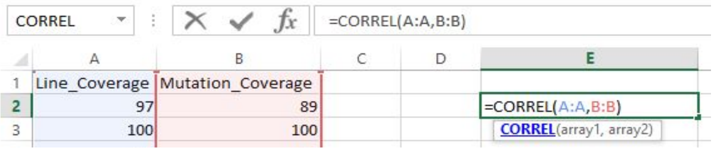
Moderate: If the value lies between ± 0.30 and ± 0.49, then it is said to be a medium correlation

Low: When the value lies below + .29, then it is said to be a small correlation.

No Correlation: When the value is zero.

A. Correlation between Metric 1, 2 and Metric 3

The correlation of Branch and Statement Coverage with the Mutation Score is calculated in Excel on the data obtained for all the projects.



CORREL =CORREL(A:A,B:B)

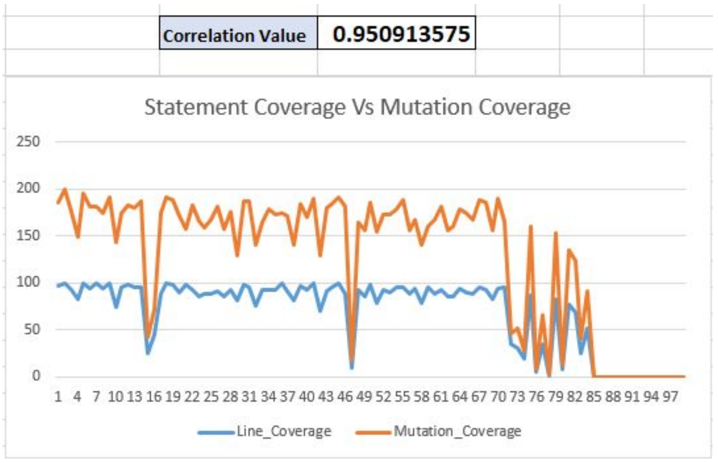
	A	B	C	D	E
1	Line_Coverage	Mutation_Coverage			
2	97	89			=CORREL(A:A,B:B)
3	100	100			CORREL(array1, array2)

Figure A.1: Calculation of Correlation in Excel

The correlation value, as shown in the below graph, comes out to be > 0.9, which indicates that there is a High correlation between Statement Coverage and Mutation Coverage.

Conclusion

Higher code coverage illustrates a higher mutation score.



Correlation Value	0.950913575

Figure A.2: A trend of Statement Coverage Vs Mutation Score

B. Correlation between Metric 4 and Metric 1,2

The correlation of Cyclomatic Complexity with Statement and Branch Coverage is given in the below table,

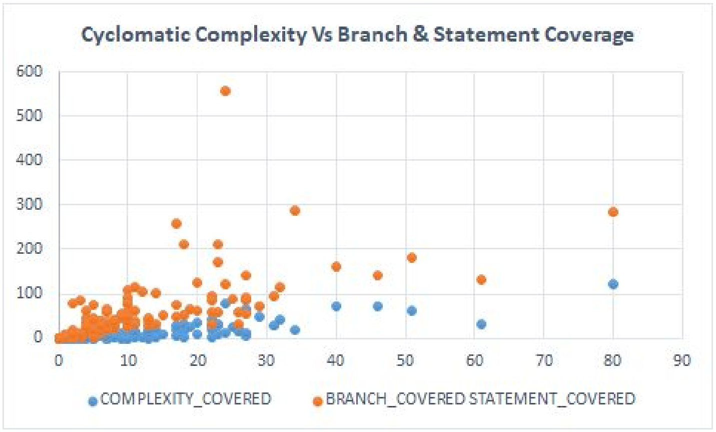
	COMPLEXITY_COVERED	BRANCH_COVERED	LINE_COVERED
COMPLEXITY_COVERED	1		
BRANCH_COVERED	0.894612464	1	
LINE_COVERED	0.691141802	0.512649171	1

Table B.1: Pearson Correlation between Cyclomatic Complexity, Statement Coverage, and Branch Coverage

As seen in the above obtained Pearson correlation values, there is a High correlation between Cyclomatic Complexity and coverage. As the Cyclomatic Complexity of the code goes on increasing or decreasing, the required coverage on Branch and Statement also shows an equivalent increase or decrease.

Conclusion

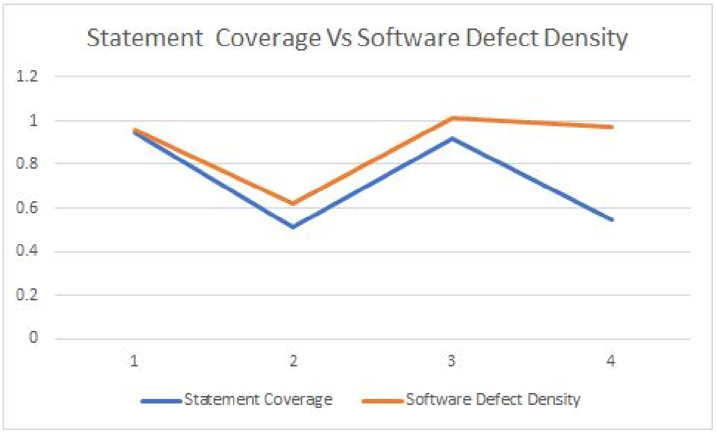
Higher Cyclomatic Complexity of a code requires higher code coverage.



Graph B.1: Cyclomatic Complexity Vs Code Coverage

C. Correlation between Metric 1,2 and Metric 6

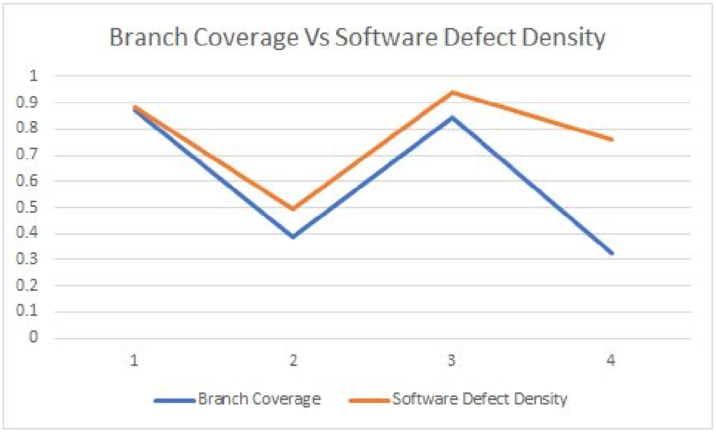
The correlation of code coverage and Software Defect Density is calculated in two parts as below. The correlation coefficient for both, comes out to be a negative value. This indicates that as the value of one variable increases, the value of another decreases and vice a versa. Two line graphs below, demonstrate the correlation between the stated variables.



Graph C.1: Statement Coverage Vs Software Defect Density

The value of correlation coefficient calculated is -0.6.
This indicates a high correlation between the two, and we can deduce the below statement,

Conclusion
Higher the statement coverage, lower the Software Defect Density.



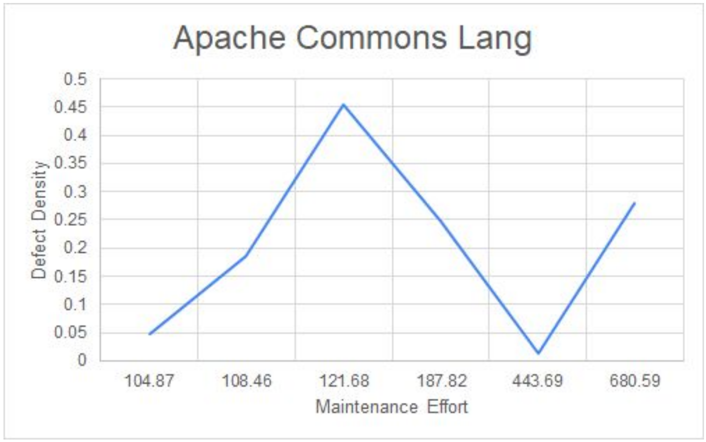
Graph C.2: Branch Coverage Vs Software Defect Density

The value of correlation coefficient calculated is -0.7.

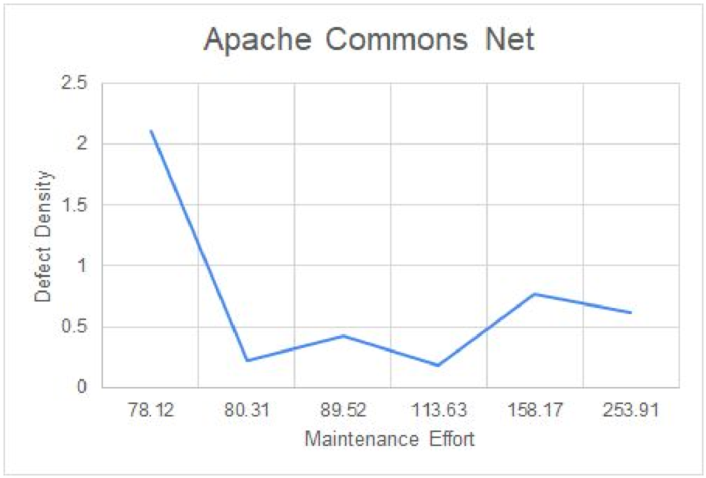
Conclusion
This indicates a high correlation between the two, and we can deduce the below statement,
Higher the branch coverage, lower the Software Defect Density.

D. Correlation between Metric 5 and Metric 6

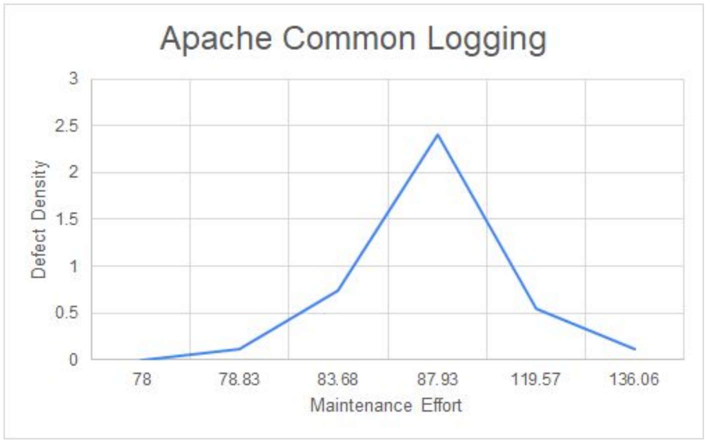
The correlation between Maintenance Effort and Defect Density is calculated for various versions of each project separately, as shown in various below graphs. As per the correlations deducted, there is a negligible or no correlation between the metrics for different versions of all the softwares studied.



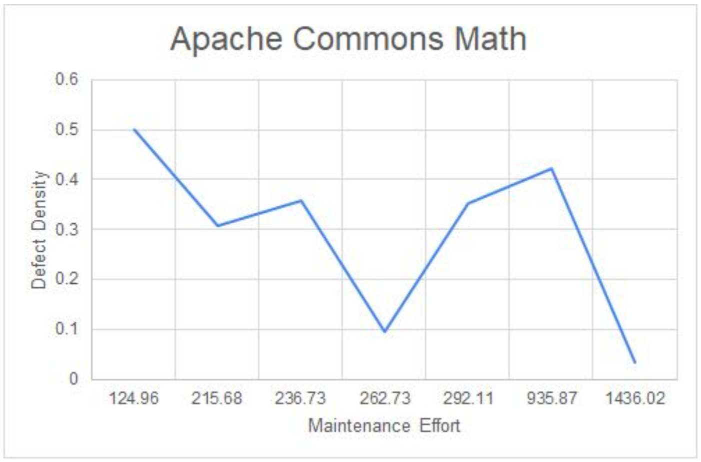
Graph D.1: Maintenance Effort Vs Defect Density for versions 2.6, 3.2, 3.3, 3.4, 3.5, and 3.1 of Apache Commons Lang



Graph D.2: Maintenance Effort Vs Defect Density for versions 1.4.1, 2.2, 3.3, 3.4, 3.5, and 3.6 of Apache Commons Net



Graph D.3: Maintenance Effort Vs Defect Density for versions 1.0.4, 1.1, 1.1.1, 1.1.2, 1.1.3, and 1.2 for Apache Commons Logging



Graph D.4: Maintenance Effort Vs Defect Density for versions 1, 1.1, 2, 2.1, 2.2, 3.4, and 3.6.1 of Apache Commons Math

Conclusion

All the Correlation Coefficient values are between 0.001 to 0.4, which states that there is no significant correlation between Metric 5 and Metric 6 for the given versions of projects.

V. RELATED WORK

The relation between the line of code and the quality of the software has various impacts on the software maintainability. Many researches have been carried out on this particular relation. Here are some of them defining the advantages of calculating various metrics.

The correlation study by Yahya, Mohammed and Bassam [18] concludes that there is a strong relation between Hallstead complexity and Cyclomatic while on the contrary it has weak relation with a number of errors. The reason proposed by them for this relationship is that the errors in the datasets used were less. Also, Hallstead complexity has a weak relation with the lines of code.

Another study by Hongyu Zhang[19], analysed the relationship between the static code attributes i.e. the LOC and defect density in the datasets. The outcome of the study concluded that LOC's ability to predict the number of defects can be improved by using the defect density values. Moreover, he used LOC and derived a defect prediction model.

VI. CONCLUSION AND FUTURE WORK

As seen, various combinations of correlations between metric 1 to metric 6 have been established. Correlations between metric 1 to metric 4 are highly visible, and those between metric 5 and metric 6 are negligible. After analyzing data of the metrics data obtained from the four projects, it is observed that all of the correlations are positive, except for metric 6, i.e. Software Defect Density, which happens to be negatively correlated with other metrics.

Even though the data collected is seen to be varied and unevenly distributed, it is observed that correlation values could be more reliable if studied on more projects from diverse domains. There is a future scope to run the analysis and the tests on large projects, with more than 500k lines of code, to obtain better results. One of tasks could be to study different domains to ensure coverage of different types of projects. The diversity of the projects could be the basis programming language of development, business need, or technologies used.

VII. REFERENCES

[1] "Correlation Analysis - Research-Methodology," *Research*. [Online]. Available: https://research-methodology.net/research-methods/quantitative-research/correlation-regression/. [Accessed: 10-Apr-2020].

[2] "Apache Commons Logging - Overview." [Online]. Available: https://commons.apache.org/logging. [Accessed: 10-Apr-2020].

[3] C. D. Team, *Lang – Home*. [Online]. Available: https://commons.apache.org/lang. [Accessed: 10-Apr-2020].

[4] *Math – Commons Math: The Apache Commons Mathematics Library*. [Online]. Available: http://commons.apache.org/math/. [Accessed: 10-Apr-2020].

[5] A. C. D. Team, *Apache Commons Net – Overview*. [Online]. Available: http://commons.apache.org/net/. [Accessed: 10-Apr-2020].

[6]"A metrics-based software maintenance Effort Model." [Online]. Available: https://www.researchgate.net/publication/4065418_A_metrics-based_software_maintenance_Effort_Model. [Accessed: 10-Apr-2020].

[7]"What is Defect Density? Formula to calculate with Example," *Guru99*. [Online]. Available: https://www.guru99.com/defect-density-software-testing-terminology.html. [Accessed: 10-Apr-2020].

[8]"Software Engineering: Software Metrics - javatpoint," *www.javatpoint.com*. [Online]. Available: https://www.javatpoint.com/software-engineering-software-metrics. [Accessed: 10-Apr-2020].

[9]"What is Defect Density? Formula to calculate with Example," *Guru99*. [Online]. Available: https://www.guru99.com/defect-density-software-testing-terminology.html. [Accessed: 10-Apr-2020].

[10]"Clover Support," *About Code Coverage - Atlassian Documentation*. [Online]. Available: https://confluence.atlassian.com/clover/about-code-coverage-71599496.html. [Accessed: 10-Apr-2020].

[11] "Metrics," *Metrics | Collaborator Documentation*. [Online]. Available: https://support.smartbear.com/collaborator/docs/reference/metrics.html. [Accessed: 10-Apr-2020].

[12] "Real world mutation testing," *PIT Mutation Testing*. [Online]. Available: http://pitest.org/. [Accessed: 10-Apr-2020].

[13] "Mutation Testing in Software Testing: Mutant Score & Analysis Example," *Guru99*. [Online]. Available: https://www.guru99.com/mutation-testing.html. [Accessed: 10-Apr-2020].

[14] "Overview," *EclEmma*, 28-Mar-2017. [Online]. Available: https://www.eclemma.org/. [Accessed: 10-Apr-2020].

[15] "(PDF) An Investigation of the Relationships between Lines ..." [Online]. Available: https://www.researchgate.net/publication/316922118_An_Investigation_of_the_Relationships_between_Lines_of_Code_and_Defects. [Accessed: 10-Apr-2020].

[16] "Pitclipse," *Eclipse Plugins, Bundles and Products - Eclipse Marketplace*. [Online]. Available: https://marketplace.eclipse.org/content/pitclipse. [Accessed: 10-Apr-2020].

[17] "Overview," *Mutation operators*. [Online]. Available: http://pitest.org/quickstart/mutators/. [Accessed: 10-Apr-2020].

[18] "The Correlation among Software Complexity Metrics with Case Study" [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1408/1408.4523.pdf [Accessed:10-Apr-2020]

[19] "An Investigation of the Relationships between Lines of Code and Defects" [Online]. Available: https://www.researchgate.net/publication/316922118_An_Investigation_of_the_Relationships_between_Lines_of_Code_and_Defects [Accessed: 10-Apr-2020]