Cover Letter

Respected Professor Yang,

Our team would like to submit the enclosed manuscript entitled "Correlation Exploration Between Diverse Software Metrics". In this paper, we did correlation analysis among the following Five Software metrics: branch coverage, statement coverage, mutation score, McCabe complexity, backlog management index and Defect density. On the bases of four open-source projects to utilize as our data pool.

The followed table is the information about our team members:

Name	**Student Number**	**E-Mail**
Ali Zafar Iqbal	40076897	alizafar9361@gmail.com
Micheal Hanna	40075977	michael_baligh@yahoo.com
Mashhad Saghaleini	40058409	Mahshad.saghaleini@gmail.com
Han Gao	40020549	rebeccahangao@gmail.com
Arjun Singh	40076630	arjun.thakur565@gmail.com

Following is a link to the replication package in GitHub:

https://github.com/HannaMichael/SOEN6611

Thank you very much for your undivided attention.

Sincerely,
Group B

Correlation Exploration Between Diverse Software Metrics

Ali Zafar Iqbal
Gina Cody Department of Engineering,
Concordia University, Canada
alizafar9361@gmail.com

Micheal Hanna
Gina Cody Department of Engineering,
Concordia University, Canada
michael_baligh@yahoo.com

Mahshad Saghaleini
Gina Cody Department of Engineering,
Concordia University, Canada
Mahshad.saghaleini@gmail.con

Han Gao
Gina Cody Department of Engineering,
Concordia University, Canad
rebeccahangao@gmail.com

Arjun singh Thakur
Gina Cody Department of Engineering,
Concordia University, Canada
arjun.thakur565@gmail.com

Abstract— **Correlation between software metrics is the measure of whether there is a relationship between two different variables within the structure of a software In this paper, we focus on several common internal software metrics: statement coverage, branch coverage, mutation score, McCabe complexity, backlog management index and Defect Density and. We find relations between these software metrics by analyzing analytical data.**

We Have selected four different open source projects from which we collected our collective data from Apache Commons Collection, Apache commons CLI, Apache commons Configurations and Apache commons math. To calculate correlation, we used the Pearson Correlation and the Spearman Correlation.

In conclusion, Branch coverage and McCabe complexity are negative and the strength of the association is good but not very strong; branch coverage, statement coverage, and metric mutation score are negative and very weak when considering Pearson; branch coverage and statement coverage and metric 6 were very small; backlog management index and Defect density were positively correlated and moderately strong.

Keywords—metric, correlation, software measurement, test

I. INTRODUCTION

With the development of science and technology, the software industry has been growing at an accelerated rate for the past 30 years [7]. Excellent software can improve work efficiency, while software with some bugs may have a bad impact on work.. Accordingly, programming estimation has become noteworthy lately. Being developed, programming engineers and task supervisors need to consistently assess programming tasks and study relations and connections, everything being equal and factors that can affect emphatically or adversely their created programming items. In this paper, we tend to analyze the correlation between different metrics. According to the development experience, we selected four open-source projects and six software measurement metrics.

II. RELATED WORK

Software metrics are often supposed to give in-depth information for the development of software. Some researchers already analyze correlation matrices with C andC++ programs [8].
According to some researcher's analysis, we got followed conclusions:

(1) there is a very strong correlation between Lines of Code and Halstead Volume;[1]

(2) there is a stronger correlation between Lines of Code and McCabe's Cyclomatic Complexity [1]

(3) none of the internal software metrics makes it possible to discern correct programs from incorrect ones;

However, most of these correlations are related to complexity. We are going to research correlation metrics in other aspects as well.

III. PROJECT DESCRIPTION

To make our analysis more convincing, we're choosing 4 java open source projects, in which 2 of them are greater than 100K LOC. Moreover, for each project, we choose 5 different subversions to collect the data. So we are able to collect the difference during the version evolution period. In addition, all of the projects that we choose have an issue-tracking system, such as Jira or apache which we used for collecting the data for maintenance relevant metrics.

1- Apache Commons Collections *(72.9K lines of code)*
Source Code: https://github.com/apache/commons-collections
Issue Tracking:
https://issues.apache.org/jira/projects/COLLECTIONS/issues/C

OLLECTIONS-756?filter=allissues

The Java Collections Framework was a major addition in JDK 1.2. It added many powerful data structures that accelerate the development of the most significant Java applications. Since that time it has become the recognized standard for collection handling in Java.

*2- **Apache commons CLI*** *(9.29K lines of code)*
Source Code:
https://github.com/apache/commons-cli.git
Issue Tracking:
https://issues.apache.org/jira/projects/CLI/issues/CLI-244?filter=allopenissues

The Apache Commons CLI are the components of the Apache Commons which are derived from Java API and provide an API to parse command-line arguments/options passed to the programs. This API also enables print help related to options available.

3- Apache Commons Configuration *(847K lines of code)*
Sourcecode:
https://github.com/apache/commons-configuration
Issue tracking:
https://issues.apache.org/jira/projects/CONFIGURATION/issues/CONFIGURATION-786?filter=allissues

The Commons Configuration software library provides a generic configuration interface which enables a Java application to read configuration data from a variety of sources.

The configuration is a large apache project, which contains several active versions as well as a continuous bug-tracking system, which lists out all the issues and its detailed description, solving status and timestamp. It makes our data collection work for metrics 5 easier. We monitor mostly post-release versions 5 to be exact

4- Apache Commons Math *(166k lines of code)*
Sourcecode:
https://github.com/apache/commons-math.git
Issue tracking:
https://issues.apache.org/jira/projects/MATH/issues/MATH-1528?filter=allissues

Apache Commons Math is the biggest open-source library of mathematical functions and utilities for Java.consists of mathematical functions (erf for instance), structures representing mathematical concepts (like complex numbers, polynomials, vectors, etc.), and algorithms that can be used with these structures (root finding, optimization, curve fitting, computation of intersections of geometrical figures, etc.).

We checked with the loc count to be consistent bi two projects, therefor two of our projects have a total size count of greater than 100k+ lines of code whilst the other two have less than 100K, this was done In order to keep data consistent we encompassed the collection from 5 different post-release versions of the open-source systems. And showcase the correlation between large and smaller project sizes

IV. METRICS DESCRIPTION

In order to better measure selected projects, six different software measurement metrics are used. These five metrics belong to different aspects of software measurement. The details will be given as follow:

Metric 1: Statement Coverage

Statement coverage is an irreplaceable metric of software quality testing, it allows the investigation of thoroughness during testing. In the industry, test coverage is often measured as statement coverage [1]. According to the actual development experience, statement coverage is suitable for software measurement in project analysis. In layman's terms Statement coverage counts is how many program statements are executed at least once during the test and thereby the more coverage percent it shows, which therefore increases the opportunity to find the existing bugs [2].

Metric 2: Branch Coverage

Though statement coverage is essential, it also has some defects. For example, statement coverage only considers the executed statements and ignores the combinations of branches. However, branch tests are available to detect cryptic errors in code. As a result, branch coverage was chosen. Branch coverage is how many branches from each decision point is executed at least once thereby the more coverage percent it shows, the more opportunity to find the existing bug [2].

Metric 3: Mutation Score

To find the weakness of code, the mutation score is a useful measurement metric. A mutation score could be obtained through mutation testing. Mutation testing is a means of creating more effective test cases. Mutation testing is primarily used as a program-based technique. It uses mutation operations to mutate the program and generate program mutants. The goal in mutation testing is killing the generated mutants by causing the mutant to have different behavior from the original program on the same input data [3]. The way to calculate the mutation score as follow:

$$MutationScore = \frac{KilledMutants}{TotalnumberofMutants} * 100$$

Metric 4: McCabe Metric (Cyclomatic Complexity)

Cyclomatic complexity is a measure of a module's structural complexity. Studies show a correlation between a program's Cyclomatic Complexity and its maintainability and testability[1][2],

Complexity is vital to software measurement. In the analysis, McCabe's complexity (Cyclomatic Complexity) was selected. Cyclomatic Complexity is used as indicators for program modularization, revising specifications, and test coverage. In addition, it has been used in software quality prediction models, whose purposes include predicting fault numbers through multivariate regression analysis and identification of error-prone modules based on discriminant analysis [4]. For calculating McCabe complexity, followed elements should be counted:

• E = the number of edges in CFG
• N = the number of nodes in CFG
• P = the number of connected components in CFG
• D = the number of control predicate (or decision) statements
• For a single method or function, P is equal to 1
• Cyclomatic Complexity = E – N + 2P
(Or Cyclomatic Complexity = D + 1)

Metric 5: Fix Backlog and Backlog Management Index (BMI)

In software measurement, software maintenance effort should not be ignored. The backlog and backlog management index is related to software maintenance effort and it is a metric to manage the backlog of open, unresolved tasks. If the BMI is less than 100, then the backlog increases. With enough data points, the techniques of control charting can be used to calculate the backlog management capability of the maintenance process. More investigation and analysis should be triggered when the value of BMI exceeds the control limits. A BMI trend chart or control chart should be examined together with trend charts of defect arrivals, defects fixed (closed), and the number of problems in the backlog [4]. The formula as following shows:

Matric 6 : Defect Density

Defect Density is the number of confirmed defects detected in software/component during a **defined period** of development/operation divided by the size of the software/component.:

$$\text{Defect Density} = \frac{\text{Number of Defects}}{\text{Size}}$$

V.DATA RETRIEVAL PROCESS

To track the progress of software build systems, we define and analyze build system metrics for individual release snapshots of a software project. An overview of our approach is shown in the steps mentioned below. We now explain each step of our approach. Our methodology is split into several sections

S1 Selecting Projects

S2 Building projects

S3 Adding Jacoco Plugin

S4 Adding Pit Plugin

S5 Selecting subversions and active months for collection of data from issue tracking systems

S6 *Consolidation of information for final analysis*

Step 1: *Selecting projects*

There were several factors that were accounted for whilst choosing sample projects for our research, some of the standards are as followed

1. It's an open-source project with retrievable source, programmed in the Java language

2. The project is managed or built by either Maven or ant

3. There should be an issue tracking system with the bug reports and feature request available per release

4. There are several subversions to allow us variance in our collected data

Our projects were retrieved corresponding to official project releases. These releases were downloaded from the official release archives such as GitHub and SourceForge in tandem,

We enforced these standards on ourselves to provide consistency in our research's investigation and allow us to cut the loose off time in the initialization of proprietary tools for matrice measurement

Step 2 : *Building Projects*

Upon selecting the projects, we test build them to ensure that there aren't any missing dependencies, fixing the test suites and the concerned projects are fully buildable without errors. This is a crucial step as this will allow the integration of future plugins without errors.

Step 3 : *Adding Jacoco plugin*

As one of our covering matrices was for statement coverage and branch coverage in coordination with coverage of the complexity of our target project, therefore we configured each of our projects including their subversions to generate the Jacoco reports for them. Implementation examples of the Jacoco plugin in maven and the output report can be seen in **Fig 1** and **Fig 2** respectively.
During our testing process, we saw that some of the test cases were not passing and causing build errors for our programs, therefore some of these tests were taken out to resolve the build issues.

Step 4: *Adding Pit plugin*

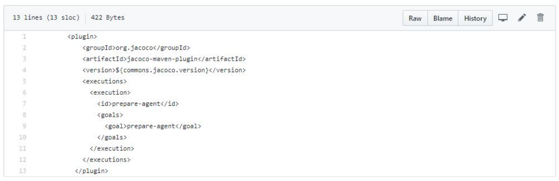
In order to calculate our mutator scores, we are using the pit plugins to generate the report, in the configuration, it allowed us to choose specific mutator, target rudimentary java classes and target test cases an example of our configuration setup and the output report are illustrated **Fig 3 and Fig 4** respectively**.**

Step 5: *Selecting subversions and active months for the collection of data from issue tracking systems*

For our data variance target, we decided on collecting the test data from 5 active months for each subversion of our projects, this allowed us to extend our data pool for the calculation of our Matric 5 Fix Backlog and Backlog Management Index while selecting the subversions, we were careful to select the versions with the most amount of reported bugs and hence calculate an average value of these five months.
The Issue tracking systems were utilized for Matric 6 as well,

Step 6: *Consolidation of information for final analysis*

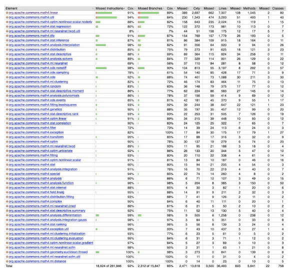
After the collection of all our data from the different plugins and issue tracking system, the collected data was consolidated in CSV files for final calculations and graph generation to showcase the file comparison and correlation between matrices.



13 lines (13 sloc) | 422 Bytes

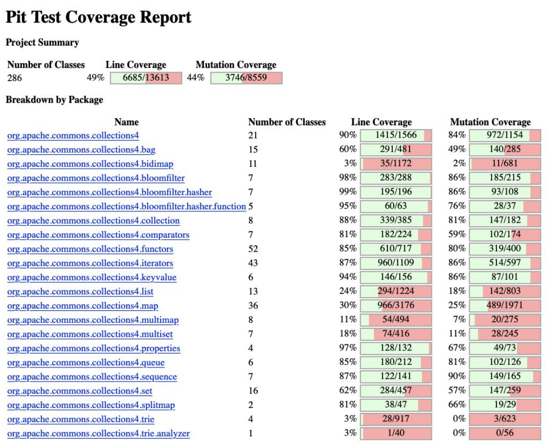
```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>${commons.jacoco.version}</version>
    <executions>
      <execution>
        <id>prepare-agent</id>
        <goals>
          <goal>prepare-agent</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
```

Fig.1. *Jacoco configuration (Commons Math)*



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.apache.commons.math4.linear		91%		89%	389	2,697	662	7,307	108	1,045	2	80
org.apache.commons.math4.util		94%		84%	230	1,343	474	3,293	51	493	1	40
org.apache.commons.math4.optim.nonlinear.scalar.noderiv		88%		82%	158	643	235	2,024	13	119	1	15
org.apache.commons.math4.stat.regression		80%		72%	122	372	173	981	10	113	0	7
org.apache.commons.math4.ml.neuralnet.twod.util		8%		7%	44	51	158	175	12	17	5	7
org.apache.commons.math4.dfp		91%		87%	154	769	157	1,779	26	193	0	5
org.apache.commons.math4.stat.inference		87%		81%	86	384	109	915	18	164	0	11
org.apache.commons.math4.analysis.interpolation		98%		82%	91	356	64	920	9	94	0	26
org.apache.commons.math4.distribution		85%		75%	79	273	91	625	18	121	3	23
org.apache.commons.math4.stat.descriptive		88%		69%	89	364	115	824	34	258	0	11
org.apache.commons.math4.analysis.solvers		89%		84%	77	329	114	901	26	129	0	22
org.apache.commons.math4.ml.neuralnet		69%		56%	37	110	84	281	8	58	0	12
org.apache.commons.math4.ode.nonstiff		98%		90%	104	813	55	3,197	8	292	0	57
org.apache.commons.math4.ode.sampling		78%		61%	54	140	85	428	17	71	3	9
org.apache.commons.math4.ode		92%		88%	74	401	73	1,088	30	211	3	30
org.apache.commons.math4.ml.clustering		82%		72%	46	174	83	464	11	75	0	13
org.apache.commons.math4.random		83%		83%	36	148	79	375	17	77	0	12
org.apache.commons.math4.stat.descriptive.moment		86%		77%	62	224	86	583	27	145	0	14
org.apache.commons.math4.analysis.polynomials		86%		78%	37	156	59	414	9	69	0	12
org.apache.commons.math4.ode.events		85%		81%	42	181	45	370	9	55	1	17
org.apache.commons.math4.fitting.leastsquares		93%		92%	39	244	38	647	22	121	1	23
org.apache.commons.math4.genetics		90%		88%	35	197	35	457	12	99	0	20
org.apache.commons.math4.stat.descriptive.rank		93%		93%	22	253	31	505	7	126	0	19
org.apache.commons.math4.optim.linear		91%		89%	34	215	39	448	10	93	0	12
org.apache.commons.math4.stat.correlation		90%		83%	24	125	25	322	7	65	0	7
org.apache.commons.math4.filter		72%		73%	14	39	24	113	6	24	0	3
org.apache.commons.math4.exception		82%		100%	17	84	30	175	17	79	1	27
org.apache.commons.math4.transform		95%		85%	17	88	17	316	3	32	0	9
org.apache.commons.math4.optim		90%		78%	30	137	19	279	6	74	0	20
org.apache.commons.math4.ml.neuralnet.twod		89%		93%	11	95	21	188	3	18	0	4
org.apache.commons.math4.optim.univariate		92%		86%	26	133	20	344	5	47	0	10
org.apache.commons.math4.fitting		93%		80%	20	110	22	338	4	61	0	14
org.apache.commons.math4.optim.nonlinear.scalar		91%		81%	13	84	10	187	0	46	0	16
org.apache.commons.math4.stat		90%		80%	15	94	21	205	4	54	0	3
org.apache.commons.math4.analysis.integration		91%		76%	16	70	22	219	4	40	0	7
org.apache.commons.math4.special		94%		85%	22	74	14	260	4	8	0	2
org.apache.commons.math4.analysis		90%		86%	6	71	12	107	3	49	0	15
org.apache.commons.math4.analysis.function		96%		98%	5	204	8	385	3	150	0	50
org.apache.commons.math4.stat.interval		88%		85%	4	30	3	82	1	20	0	6
org.apache.commons.math4.field.linalg		95%		88%	14	91	9	211	2	32	0	3
org.apache.commons.math4.distribution.fitting		93%		88%	7	41	9	126	2	11	0	2
org.apache.commons.math4.complex		90%		84%	6	40	11	111	0	20	0	1
org.apache.commons.math4.ml.neuralnet.oned		86%		81%	6	21	5	50	2	10	0	2
org.apache.commons.math4.stat.descriptive.summary		92%		75%	11	52	13	130	5	40	0	4
org.apache.commons.math4.analysis.differentiation		99%		98%	9	505	6	1,258	0	238	0	12
org.apache.commons.math4.analysis.integration.gauss		98%		97%	3	84	5	351	0	33	0	8
org.apache.commons.math4.stat.ranking		96%		96%	5	56	8	139	3	24	0	4
org.apache.commons.math4.exception.util		99%		93%	7	43	10	437	5	27	1	4
org.apache.commons.math4.ml.clustering.initialization		93%		77%	6	23	5	61	0	5	0	2
org.apache.commons.math4.ml.clustering.evaluation		95%		84%	6	31	3	71	1	12	0	2
org.apache.commons.math4.optim.nonlinear.scalar.gradient		97%		84%	5	27	3	69	0	10	0	3
org.apache.commons.math4.ml.neuralnet.sofm		98%		95%	2	31	1	63	1	21	0	8
org.apache.commons.math4.geometry.euclidean.threed		100%		98%	3	273	0	841	0	139	0	6
org.apache.commons.math4.ml.neuralnet.sofm.util		100%		100%	0	11	0	31	0	4	0	2
org.apache.commons.math4.ml.distance		100%		100%	0	14	0	23	0	10	0	5
Total	18,624 of 261,946	92%	2,312 of 15,847	85%	2,471	13,618	3,500	36,493	603	5,641	22	758

Fig.2. *Jacoco report (Commons math)*



Pit Test Coverage Report

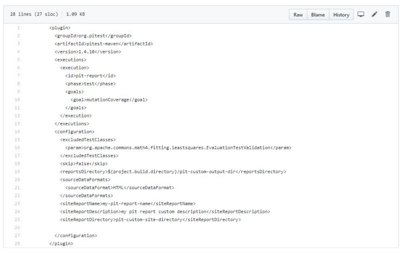
Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
286	49%	6685/13613	44%	3746/8559

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
org.apache.commons.collections4	21	90%	1415/1566	84%	972/1154
org.apache.commons.collections4.bag	15	60%	291/481	49%	140/285
org.apache.commons.collections4.bidimap	11	3%	35/1172	2%	11/681
org.apache.commons.collections4.bloomfilter	7	98%	283/288	86%	185/215
org.apache.commons.collections4.bloomfilter.hasher	7	99%	195/196	86%	93/108
org.apache.commons.collections4.bloomfilter.hasher.function	5	95%	60/63	76%	28/37
org.apache.commons.collections4.collection	8	88%	339/385	81%	147/182
org.apache.commons.collections4.comparators	7	81%	182/224	59%	102/174
org.apache.commons.collections4.functors	52	85%	610/717	80%	319/400
org.apache.commons.collections4.iterators	43	87%	960/1109	86%	514/597
org.apache.commons.collections4.keyvalue	6	94%	146/156	86%	87/101
org.apache.commons.collections4.list	13	24%	294/1224	18%	142/803
org.apache.commons.collections4.map	36	30%	966/3176	25%	489/1971
org.apache.commons.collections4.multimap	8	11%	54/494	7%	20/275
org.apache.commons.collections4.multiset	7	18%	74/416	11%	28/245
org.apache.commons.collections4.properties	4	97%	128/132	67%	49/73
org.apache.commons.collections4.queue	6	85%	180/212	81%	102/126
org.apache.commons.collections4.sequence	7	87%	122/141	90%	149/165
org.apache.commons.collections4.set	16	62%	284/457	57%	147/259
org.apache.commons.collections4.splitmap	2	81%	38/47	66%	19/29
org.apache.commons.collections4.trie	4	3%	28/917	0%	3/623
org.apache.commons.collections4.trie.analyzer	1	3%	1/40	0%	0/56

Fig.3. *Pit report (Commons Math)*



28 lines (27 sloc) | 1.09 KB

```
<plugin>
  <groupId>org.pitest</groupId>
  <artifactId>pitest-maven</artifactId>
  <version>1.4.10</version>
  <executions>
    <execution>
      <id>pit-report</id>
      <phase>test</phase>
      <goals>
        <goal>mutationCoverage</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <excludedTestClasses>
      <param>org.apache.commons.math4.fitting.leastsquares.EvaluationTestValidation</param>
    </excludedTestClasses>
    <skip>false</skip>
    <reportsDirectory>${project.build.directory}/pit-custom-output-dir</reportsDirectory>
    <sourceDataFormats>
      <sourceDataFormat>HTML</sourceDataFormat>
    </sourceDataFormats>
    <siteReportName>my-pit-report-name</siteReportName>
    <siteReportDescription>my pit report custom description</siteReportDescription>
    <siteReportDirectory>pit-custom-site-directory</siteReportDirectory>

  </configuration>
</plugin>
```

Fig.4. *Pit configuration (Commons Math)*

VI.DATA ANALYSIS PROCESS

We utilized the Pearson correlation coefficient and Spearman correlation coefficient for the analysis of our matrice correlation.

The procedure for our data analysis is as follows:

S1. Identifying and determining the two matrices which are suitable for comparative analysis and correlation hence, extraction of data for the defined projects from the collected data.

S2. Importing the collected data into Excel/MATLAB to calculate their correlation using the Pearson correlation coefficient and spearman's rank coefficient. And generating the final graphs and diagrams for showcase

S3. Final Comparison of the results of the specific metric correlation coefficients of the five projects and foretell our conclusion

VII. RESULTS

To preserve readability, the results for our Correlation analysis are divided out in the following sections.

Section A - Correlation between *Metric Statement, Branch and Mutation Score*

Section B - Correlation between *Statement, Branch,* and Complexity

Section C - Correlation between *Statement, Branch* and Defect Density

Section D - Correlation between *Backlog* maintainability *Index and defect density.*

Section A - Correlation between Metric (M1), Statement Branch Coverage (M2) and Mutation Score (M3)

The correlation analysis of metric 1 and 2 and metric 4 was carried out using the Spearman's rank correlation coefficient *r(s)* as well as Pearson. (refer to databank for further details)

The results for the analysis conducted for the correlation of metric 1, 2, 3. Branch, statement and mutation scores can be viewed below.

TABLE 1. R(Spearmen) Correlation for *(Apache Collections)*

	M1 Statement	**M2 Branch**	**M3 Mutation**
M1	1	0.856735588	0.142347144
M2	0.856735588	1	0.135639761
M6	0.142347144	0.135639761	1

TABLE 2. R(Spearmen) Correlation for (Apache CLI)

	M1 Statement	**M2 Branch**	**M3 Mutation**
M1	1	0.950531676	0.821433029
M2	0.950531676	1	0.669638111
M6	0.821433029	0.669638111	1

TABLE 3. R(Spearmen) Correlation for *(Apache Configuration)*

	M1 Statement	**M2 Branch**	**M3 Mutation**
M1	1	0.886769235	0.212873189
M2	0.936628645	1	0.168724012
M6	0.212873189	0.168724012	1

TABLE 4. R(Spearmen) Correlation for *(Apache Math)*

	M1 Statement	**M2 Branch**	**M3 Mutation**
M1	1	0.881994865	-0.424014601
M2	0.881994865	1	-0.469562855
M3	-0.4240146	-0.469562855	1

For this correlation we facilitated to achieve a cumulative sum of the class level data for all the observed projects, this allowed us to carry out a more accurate correlation with the mutation scores for each of the projects.

There was overall a small correlation between mutation statement and branch was observed, however, an anomaly was discovered in CLI when Spearman correlation was utilized instead of a Pearson when we observed an 82% correlation between Mutations and branch coverage. We believe that this might be due to some mutators being written specifically for missed statements.

We utilized Spearmen for this correlation as a test, which gave us more non zero results in comparisons to Pearson as seen in the sample **Table 5** below. One reason why Pearson might be trying to linearize the end sum values, whereas Spearman looks at the change in general, which causes a higher value correlation in comparison to Pearson.

TABLE 5. Sample R(Pearson) Correlation for *(Apache CLI)*

	M1 Statement	**M2 Branch**	**M4 Complexity**
M1	1	0.966404035	0.027337456
M2	0.966404035	1	0.110382735
M6	0.027337456	0.110382735	1

Section B - Correlation between Metric 1 & 2 and Metric4

The correlation analysis of metrics 1,2 and 4 was carried out using the Pearson cumulative coefficient *r*(p).

Overall we observe the correlation is positive and the strength of the association is good and is very strong. We can conclude that classes with higher Cyclomatic Complexity show a higher level of branch coverage,

specifically in this project, we observed that there was a high correlation between branch and complexity rather than statement and complexity this shows that for specific, there is a low correlation in between Statement and branch, which affected the correlation in between complexity and statement

TABLE 6. R(Pearson) Correlation for *(Apache Collections)*

	M1 Statement	**M2 Branch**	**M4 Complexity**
M1	1.00000	0.96662	0.96998
M2	0.96662	1.00000	0.94132
M4	0.96998	0.94132	1.00000

TABLE 7. R(Pearson) Correlation for *(Apache CLI)*

	M1 Statement	**M2 Branch**	**M4 Complexity**
M1	1.00000	0.96640	0.98077
M2	0.96640	1.00000	0.95583
M4	0.98077	0.95583	1.00000

TABLE 8. R(Pearson) Correlation for *(Apache Configuration)*

	M1 Statement	**M2 Branch**	**M4 Complexity**
M1	1.00000	0.93663	0.95413
M2	0.93663	1.00000	0.89532
M4	0.95413	0.89532	1.00000

TABLE 9. R(Pearson) Correlation for *(Apache Math)*

	M1 Statement	**M2 Branch**	**M4 Complexity**
M1	1.00000	0.37952	0.37625
M2	0.37952	1.00000	0.96948
M4	0.37625	0.96948	1.00000

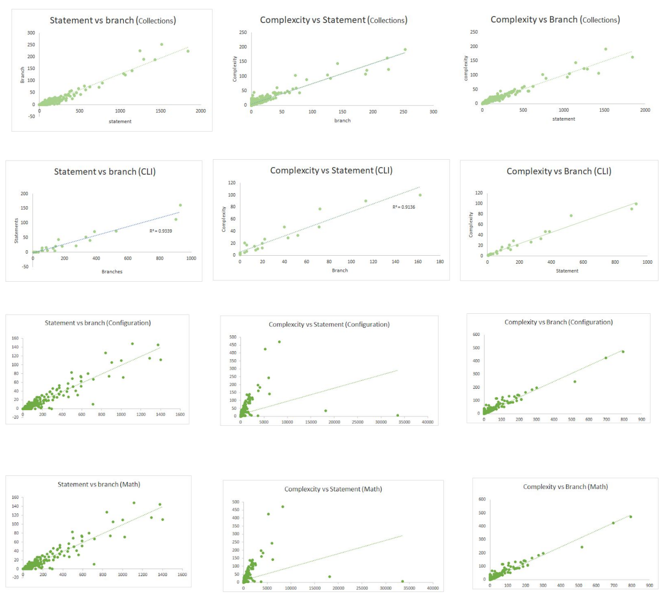


Fig.5. *Correlation Graphs for Metrics 1,2 and 4*

Section C *Correlation between Metric 1 & 2 and Metric 6*

The correlation analysis of Metric 1&2 and Metric 6 was carried out using the Pearson cumulative coefficient *r(p)*.

There was a negative correlation observed between M1 and M2 and M6 as seen on **TABLE 10** This might be due to good project management (Proper CRs, Regression testing, etc)It is also noted that in similar Research area, low correlation in between defect density and M1

Since we had conducted the correlation on class bases before

TABLE 10. R(Pearson) Correlation for *M1, M2, and M6*

Project	**M1 Statement**	**M2 Branch**	**M6 Defect Density**
Collections	50767	4938	0.000137174
CLI	4938	678	0.001291712
Configuration	42136	3856	0.000001181
Math	243558	13576	0.000421687

	M1 Statement	**M2 Branch**
M6 Defect Density	-0.224766735	-0.353265962

Section D Correlation between Metric 5 and Metric 6

The Pearson correlation coefficient *R(P)* is calculated from 5 subversions of the projects and the cumulative average value of the correlation value of each of the projects is shown below. It shows that the positive low correlation between metric 5 and metric 6 is varied by the project.

TABLE 11. Correlation between Metric 5 & 6 *(Apache Collections)*

	Average of BMI	**Defect Density**
Average of BMI	1	-0.04254908
Defect Density	-0.04254908	1

TABLE 12. Correlation between Metric 5 & 6 *(Apache CLI)*

	Average of BMI	**Defect Density**
Average of BMI	1	0.027093002
Defect Density	0.027093002	1

TABLE 13. Correlation between Metric 5 & 6 *(Apache Configuration)*

	Average of BMI	**Defect Density**
Average of BMI	1	0.762106928
Defect Density	0.762106928	1

TABLE 14. Correlation between Metric 5 & 6 *(Apache Math)*

	Average of BMI	**Defect Density**
Average of BMI	1	0.887554523
Defect Density	0.887554523	1

A low amount correlation was observed between the Cumulative BMI and Defect density of the subversions of Apache Collections and CLI whereas high correlation was observed within the Apache Math and configuration.

It is our conclusion that the correlation between BMI and Defect density increases with the number of Backlogs present in a project. As shown by the correlation growth based on the projects LOC count/Size.

VIII. Threats to validity

One of the first threats to the validity of our report would be that our evaluation uses only 4 subject programs, all written in Java. Other programs might have different characteristics in different languages. Moreover, all 4 subject programs are well-tested (see Figure 3). This may limit the applicability of the results to programs that are not well-tested or are written in a language other than Java.

Another threat would be our stance to emphasize the use of the Pearson correlation instead of the Spearman for most of our correlation analysis. However, it is our prognosis that the Pearson correlation provided an overall more accurate correlation analysis due to its overall desaturation in the final data presented in comparison to spearman's saturation of data nodes.

One of the last threats that we see in our report is that we utilized limited subversions or the post-release versions of our chosen open source projects, this could cause our analysis to have limited scope to the post-release versions of the project only. However, we believe that this threat is not justified as it allows us to measure the metrics of an already released project instead of for one that is in development.

IX. CONCLUSION

According to the results shown in **VII. Section A.**, It shows that the correlation between metric 1, 2 and 3 is positive and very strong. We can conclude that suites with higher statement or branch coverage can show a high mutation score. This conclusion is consistent with the rationale that test suites with higher coverage can show better test suite effectiveness.

According to the results shown in **VII. Section B.**, it shows that the correlation between metrics 1, 2 and 4 is positive and the strength of the association is good but not very strong. From that we observe the correlation is positive and the strength of the association is good and is very strong. we can conclude that classes with higher Cyclomatic Complexity show a higher level of branch coverage, However, a lower correlation score was observed in one the projects (Apache Math) as it was out to believe that this shows the project itself was highly complex This conclusion is consistent with the rationale that classes with higher complexity are less likely to have high coverage tests suites.

According to the results shown in **VII Section C.**, it describes that the Pearson correlation coefficients for metric 1, 2 and 6 were very small, even not greater than 0.1 in absolute value. We think that there is no or almost none correlation between statement/branch coverage and Defect density.

According to the results shown in **VII. Section D.,** it shows that the Pearson correlation coefficients of the metric 5 and 6 were positively correlated and moderately strong. We conclude that on the Subversion level, projects with higher Backlog Management Index might show higher defect density overall.

REFERENCES

[1] Ming-Chang Lee, "Software measurement and software metrics in software quality"https://www.researchgate.net/publication/260480820_Software_measurement_and_software_metrics_in_software_quality

[2] G. K. Gill and C. F. Kemerer, "Cyclomatic complexity density and software maintenance productivity," IEEE Trans. Software Eng., vol. 17, (12), pp. 1284-1288, 1991. Available: http://dx.doi.org/10.1109/32.106988.DOI:10.1109/32.106988.

[3] M. M. Suleman Sarwar, S. Shahzad, and I. Ahmad, "Cyclomatic complexity: The nesting problem," in 8th International Conference on Digital Information Management, ICDIM 2013, September 10, 2013 - September 12, 2013, Available: http://dx.doi.org/10.1109/ICDIM.2013.6693981. DOI: 10.1109/ICDIM.2013.6693981.

[4] R. Takahashi, "Software quality classification model based on McCabe's complexity measure," J. Syst. Software, vol. 38, (1), pp. 61-69, 1997. Available: http://dx.doi.org/10.1016/S0164-1212(97)00060-5. DOI: 10.1016/S0164-1212(97)00060-5.

[5] Y. H. Yang, "Software quality management and ISO 9000 implementation," Industrial Management + Data Systems, vol. 101, (7), pp. 329-38, 2001. Available: http://dx.doi.org/10.1108/EUM0000000005821. DOI: 10.1108/EUM0000000005821.

[6] Luca Cardelli, Serge Abiteboul.2015. Software Quality Management, p75-77. https://www.tutorialspoint.com/software_quality_management/index.htm

[7] M. H. Moghadam and S. M. Babamir, "Mutation score evaluation in terms of object-oriented metrics," in 4th International Conference on Computer and Knowledge Engineering, ICCKE 2014, October 29, 2014 - October 30, 2014, Available: http://dx.doi.org/10.1109/ICCKE.2014.6993419. DOI: 10.1109/ICCKE.2014.6993419.

[8] R. Hofman, "Behavioral economics in software quality engineering," Empirical Software Engineering, vol. 16, (2), pp. 278- 293, 2011. Available: http://dx.doi.org/10.1007/s10664 -010-9140-x. DOI: 10.1007/s10664-010- 9140-x.