MEASUREMENT METRICS

TEAM I

CONTENT

- Test subjects
- Metrics
- Analysis
- Correlation

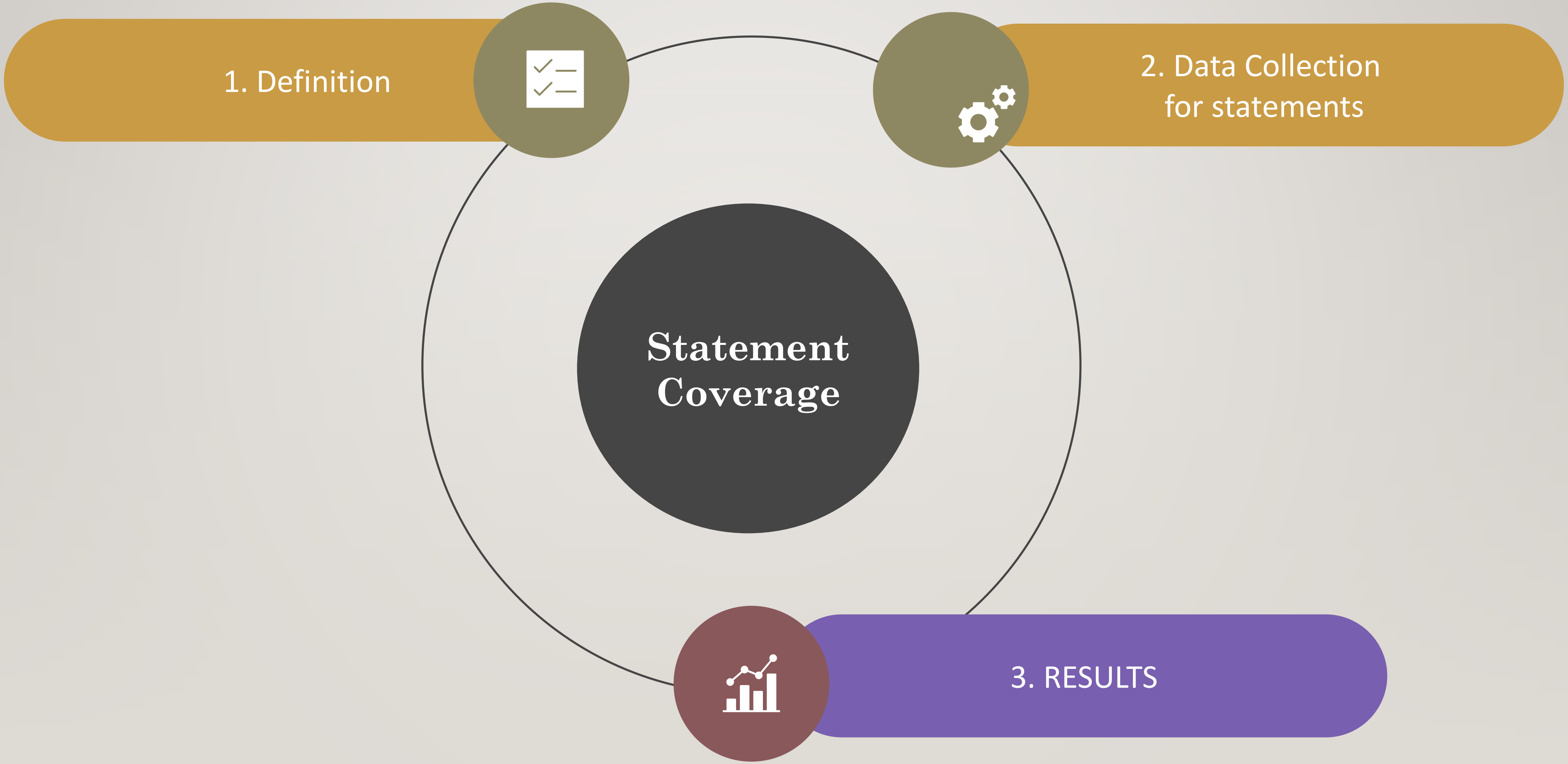
TEST SUBJECTS

Project Name	Size	Version Control System	Issue Tracking System
Commons Net	63k	Git	Jira
Commons Math	388k	Git	Jira
Commons Collections	127k	Git	Jira
jFreeChart	297k	Git	Github

METRICS

- Metric 1 : (Test Coverage Metric) Statement Coverage
- Metric 2 : (Test Coverage Metric) Branch Coverage
- Metric 3 : (Test Suit Effectiveness) Mutation Testing
- Metric 4 : (Complexity Metric) McCabe's Cyclomatic Complexity
- Metric 5 : (Software Maintenance Metric) Adaptive Maintenance Effort
- Metric 6 : (Software Quality Metric) Post Release Defect Density

Metric 1



1. Definition

2. Data Collection
for statements

Statement
Coverage

3. RESULTS

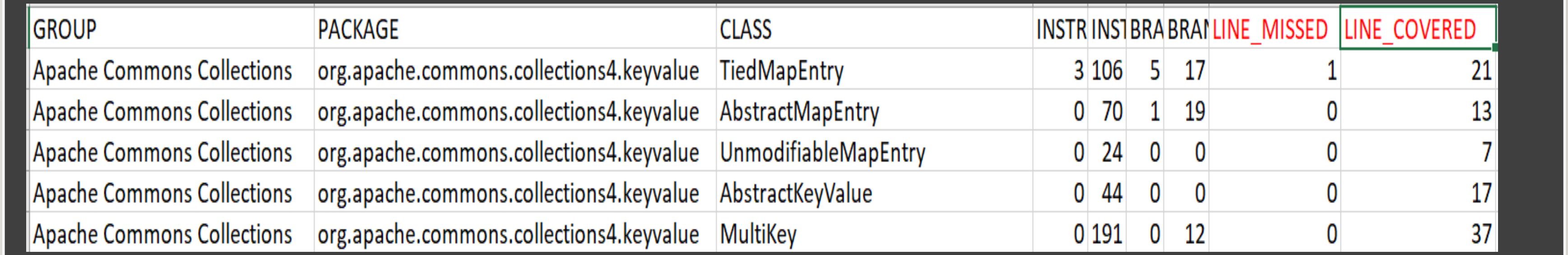
DEFINITION

- SC = (Number of statements executed/Total number of statements) *100
- It is based on the number of statements executed
- It is applied at the **class level**
- The goal of statement coverage technique is to cover all the possible **executing statements** and **path lines** in the code.

Jacoco tool is used for the calculation. Plugin for Jacoco is added to the pom file in the maven project and the project is run as maven test to get the jacoco.csv and index.html file containing class level results for the number of lines missed and lines covered. Result for the percentage is calculated considering the covered lines.

JACOCO OUTPUT EXAMPLE

The data collection using Jacoco is illustrated as below:



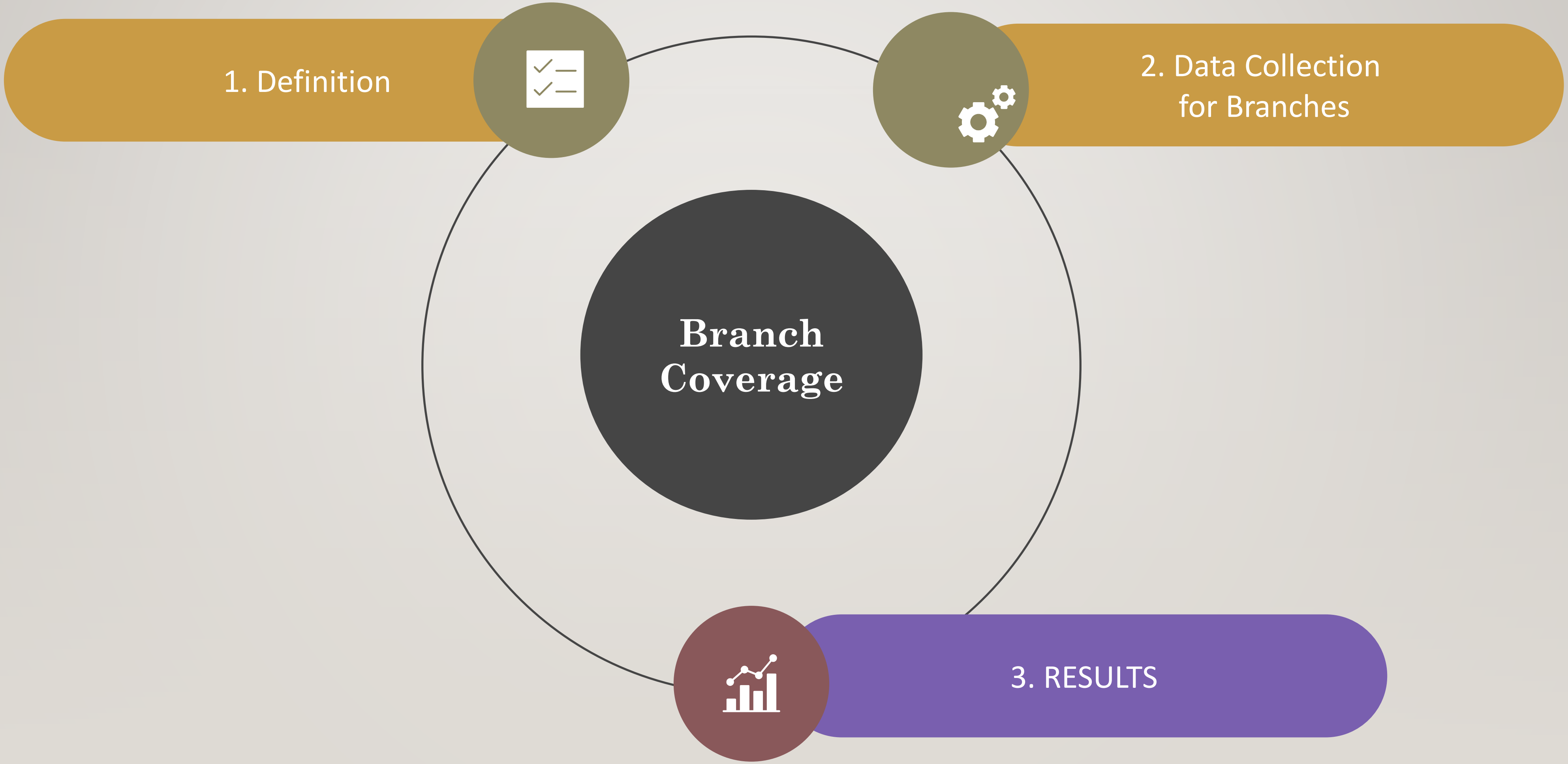
GROUP	PACKAGE	CLASS	INSTR	INST	BRA	BRAN	LINE_MISSED	LINE_COVERED
Apache Commons Collections	org.apache.commons.collections4.keyvalue	TiedMapEntry	3	106	5	17	1	21
Apache Commons Collections	org.apache.commons.collections4.keyvalue	AbstractMapEntry	0	70	1	19	0	13
Apache Commons Collections	org.apache.commons.collections4.keyvalue	UnmodifiableMapEntry	0	24	0	0	0	7
Apache Commons Collections	org.apache.commons.collections4.keyvalue	AbstractKeyValue	0	44	0	0	0	17
Apache Commons Collections	org.apache.commons.collections4.keyvalue	MultiKey	0	191	0	12	0	37

RESULTS

- Below Table shows Statement Coverage calculated for our selected Projects.

Metric	Commons Net	Commons Collections	Commons Math	jFreeChart
SC	59%	89%	92%	69%

Metric 2



1. Definition

2. Data Collection for Branches

Branch Coverage

3. RESULTS

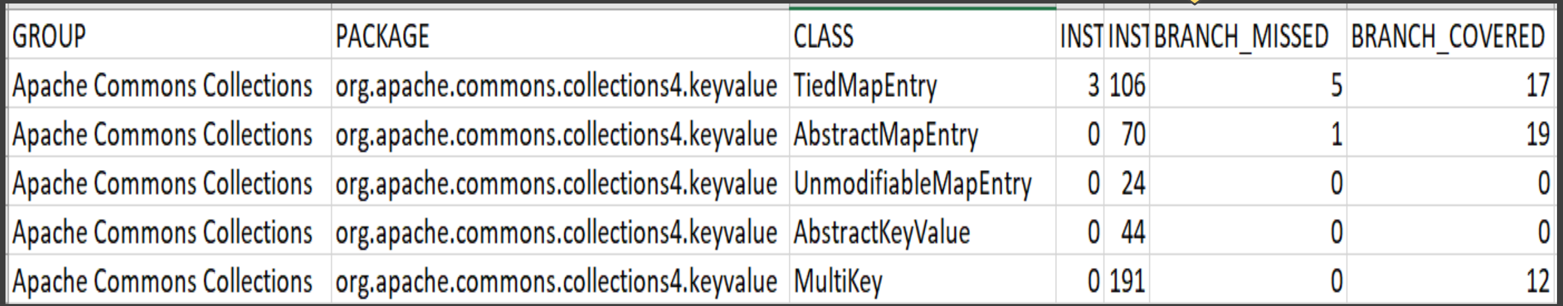
DEFINITION

- BC = (Number of decision outcomes tested/Total number of decisions) *100
- It is based on the number of branches executed
- It is applied at **class level**.
- Branch coverage is a testing method, which aims to ensure that each one of the possible branch from each decision point is executed at least once and thereby ensuring that all reachable code is executed.

Jacoco tool is used for the calculation. Plugin for Jacoco is added to the pom file in the maven project and the project is run as maven test to get the jacoco.csv and index.html file containing class level results for the number of branches missed and branches covered. Result for coverage is calculated considering branches covered.

JACOCO OUTPUT EXAMPLE

The data collection using Jacoco is illustrated as below:



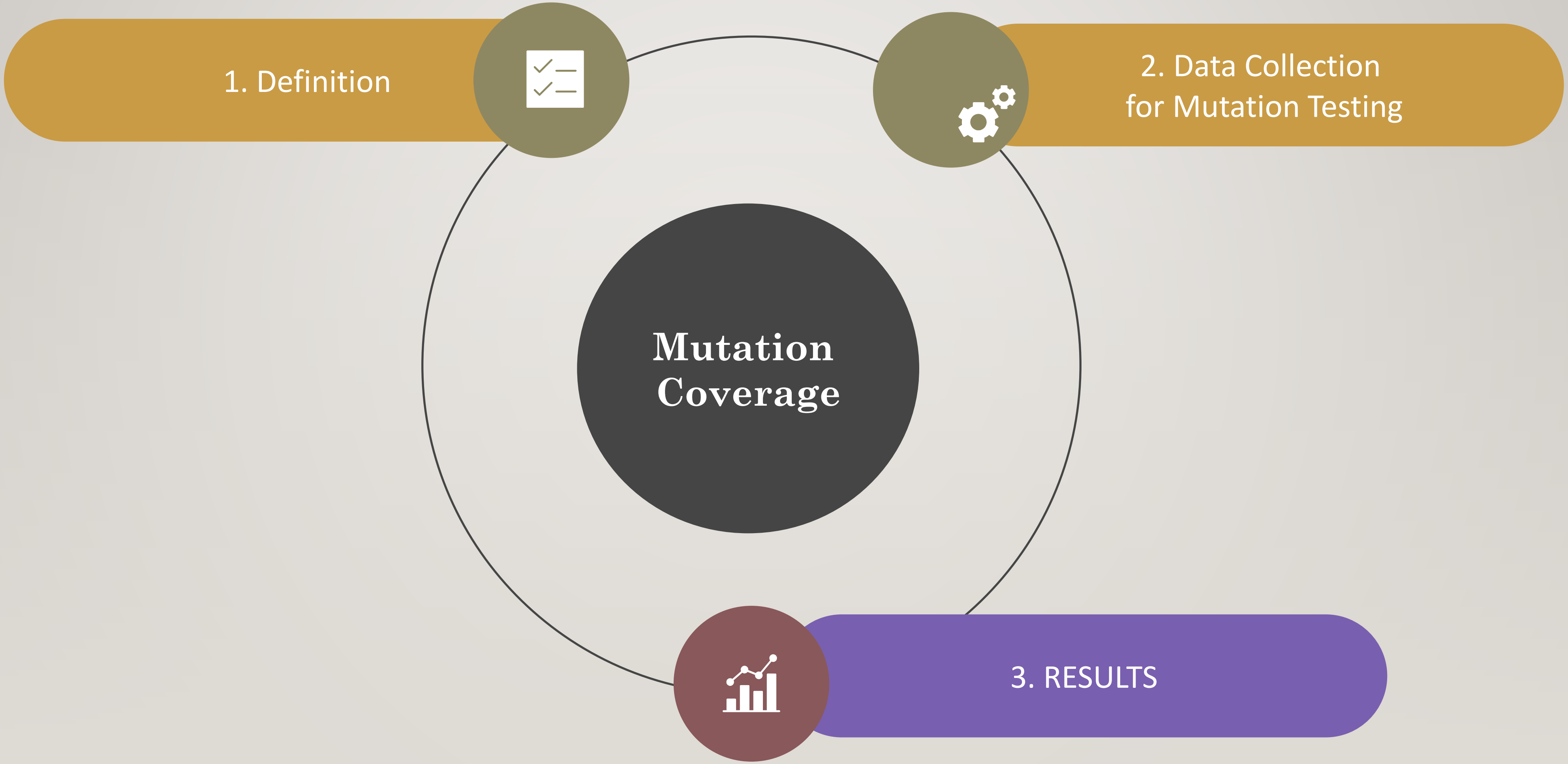
GROUP	PACKAGE	CLASS	INST	INST	BRANCH_MISSED	BRANCH_COVERED
Apache Commons Collections	org.apache.commons.collections4.keyvalue	TiedMapEntry	3	106	5	17
Apache Commons Collections	org.apache.commons.collections4.keyvalue	AbstractMapEntry	0	70	1	19
Apache Commons Collections	org.apache.commons.collections4.keyvalue	UnmodifiableMapEntry	0	24	0	0
Apache Commons Collections	org.apache.commons.collections4.keyvalue	AbstractKeyValue	0	44	0	0
Apache Commons Collections	org.apache.commons.collections4.keyvalue	MultiKey	0	191	0	12

RESULTS

- Below Table shows Branch Coverage calculated for our selected Projects.

Metric	Commons Net	Commons Collections	Commons Math	jFreeChart
BC	26%	81%	85%	46%

Metric 3



1. Definition

2. Data Collection for Mutation Testing

Mutation Coverage

3. RESULTS

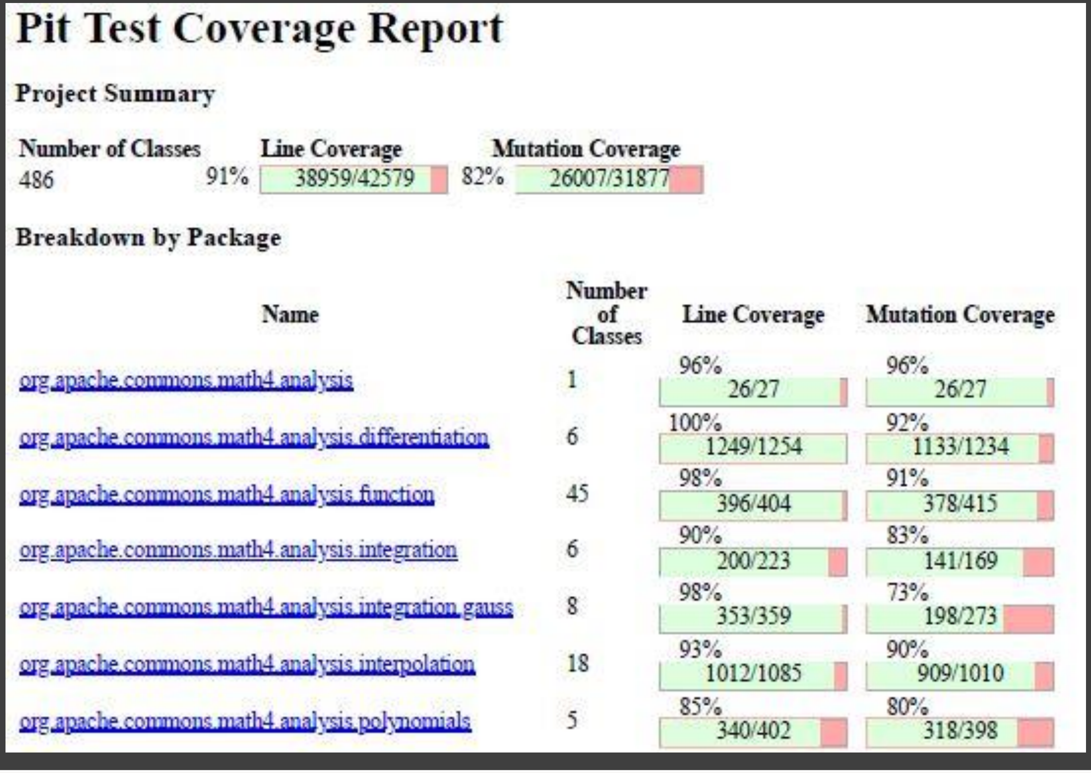
DEFINITION

- Mutation Testing is a type of software testing where we mutate (change) 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.
- It is based on the mutation score.
- It is applied at package level.
- MS = (Number of mutants killed/Total number of mutants) * 100

PIT Test tool is used for the calculation. Pitclipse is a plugin that runs the pitest mutation test tool against your unit tests. Pitclipse is added to Eclipse and the project is run as PIT configuration to get the index.html file containing package level results for the line and mutation coverage.

PITCLIPSE OUTPUT EXAMPLE

The data collection for **commons-math** using Pitclipse is illustrated as below:



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage
486	91% 38959/42579	82% 26007/31877

Breakdown by Package

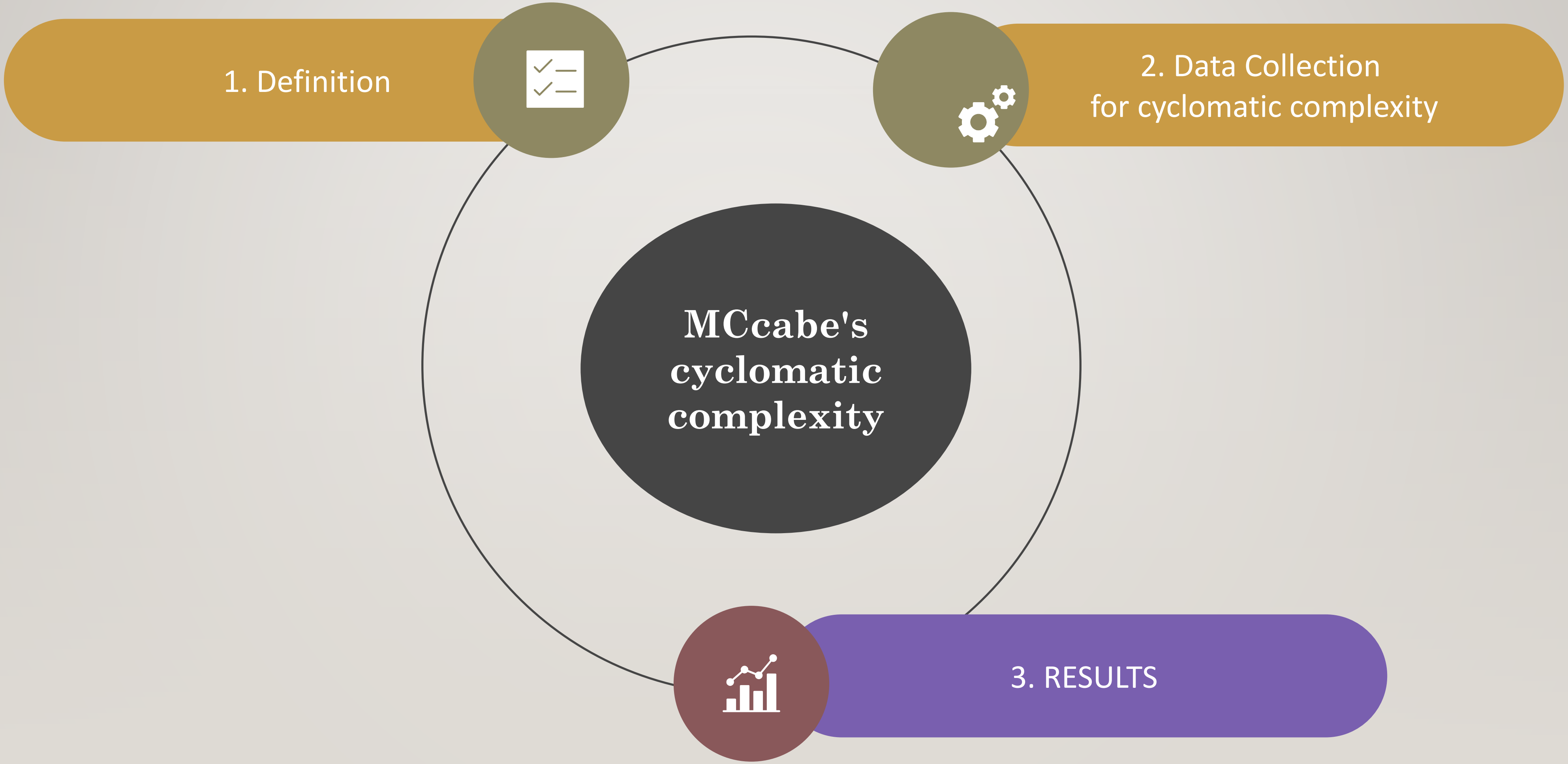
Name	Number of Classes	Line Coverage	Mutation Coverage
org.apache.commons.math4.analysis	1	96% 26/27	96% 26/27
org.apache.commons.math4.analysis.differentiation	6	100% 1249/1254	92% 1133/1234
org.apache.commons.math4.analysis.function	45	98% 396/404	91% 378/415
org.apache.commons.math4.analysis.integration	6	90% 200/223	83% 141/169
org.apache.commons.math4.analysis.integration.gauss	8	98% 353/359	73% 198/273
org.apache.commons.math4.analysis.interpolation	18	93% 1012/1085	90% 909/1010
org.apache.commons.math4.analysis.polynomials	5	85% 340/402	80% 318/398

RESULTS

- Below Table shows Mutation Coverage calculated for our selected Projects.

Metric	Commons Net	Commons Collections	Commons Math	jFreeChart
MC	26%	38%	82%	32%

Metric 4



MCcabe's cyclomatic complexity

1. Definition

2. Data Collection
for cyclomatic complexity

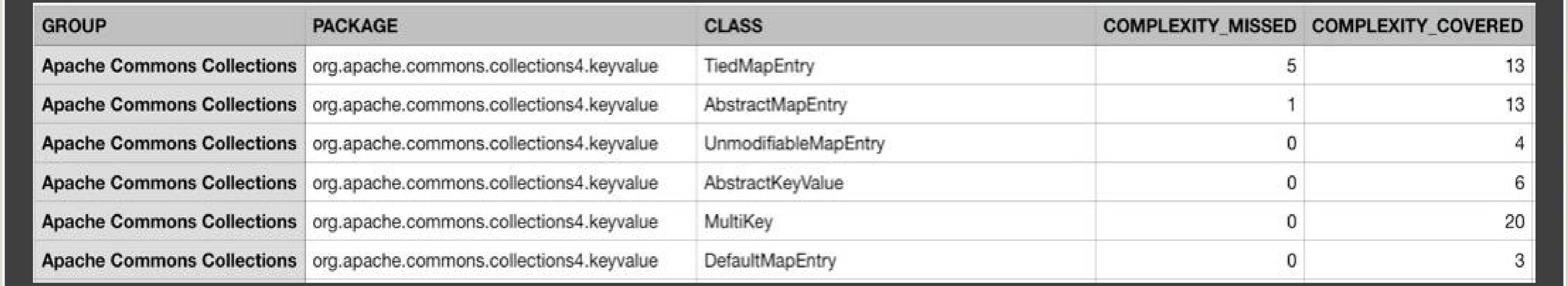
3. RESULTS

DEFINITION
-MCCABE'S CYCLOMATIC COMPLEXITY

- $CC = E - N + 2P$ or $CC = D + 1$
- Where, **E**-Number of edges in the graph. **N**-Number of Nodes in the graph. **P**-Number of connected components. **D**-Control Predicates in graph
- It is based on number of linearly independent paths
- It is applied at class level

Jacoco tool is used for the calculation. Plugin for Jacoco is added to the pom file in the maven project and the project is run as maven test to get the jacoco.csv and index.html file containing class level results for the number of complexity missed and complexity covered. Result for complexity is calculated considering complexity covered and missed.

The data collection using Jacoco is illustrated as below:



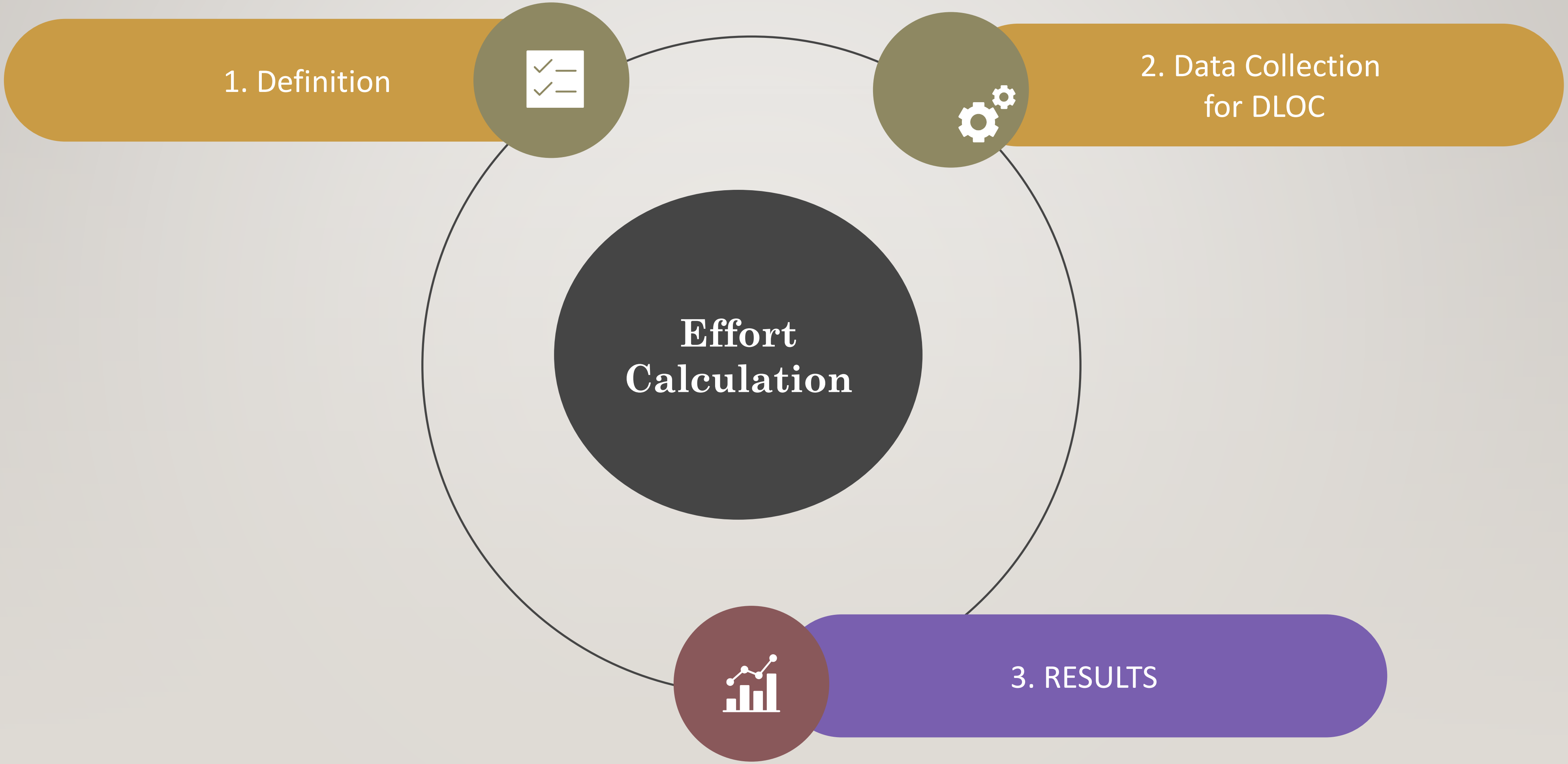
GROUP	PACKAGE	CLASS	COMPLEXITY_MISSED	COMPLEXITY_COVERED
Apache Commons Collections	org.apache.commons.collections4.keyvalue	TiedMapEntry	5	13
Apache Commons Collections	org.apache.commons.collections4.keyvalue	AbstractMapEntry	1	13
Apache Commons Collections	org.apache.commons.collections4.keyvalue	UnmodifiableMapEntry	0	4
Apache Commons Collections	org.apache.commons.collections4.keyvalue	AbstractKeyValue	0	6
Apache Commons Collections	org.apache.commons.collections4.keyvalue	MultiKey	0	20
Apache Commons Collections	org.apache.commons.collections4.keyvalue	DefaultMapEntry	0	3

RESULTS

- Below Table shows Cyclomatic Complexity calculated for our selected Projects.

Metric	Commons Net	Commons Collections	Commons Math	jFreeChart
MC	3695	7250	18887	19120

Metric 5



Effort Calculation

1. Definition

2. Data Collection for DLOC

3. RESULTS

DEFINITION

- A software continues goes under the enhancement which requires team effort to adapt the new functionality and keep the original product working. Unfortunately, developers and managers underestimate the planning and estimation required to make the required changes.
- $E = -40 + 6.56DLOC$

Where E is the effort and DLOC is the difference in lines of code.

- It is applied at package level.

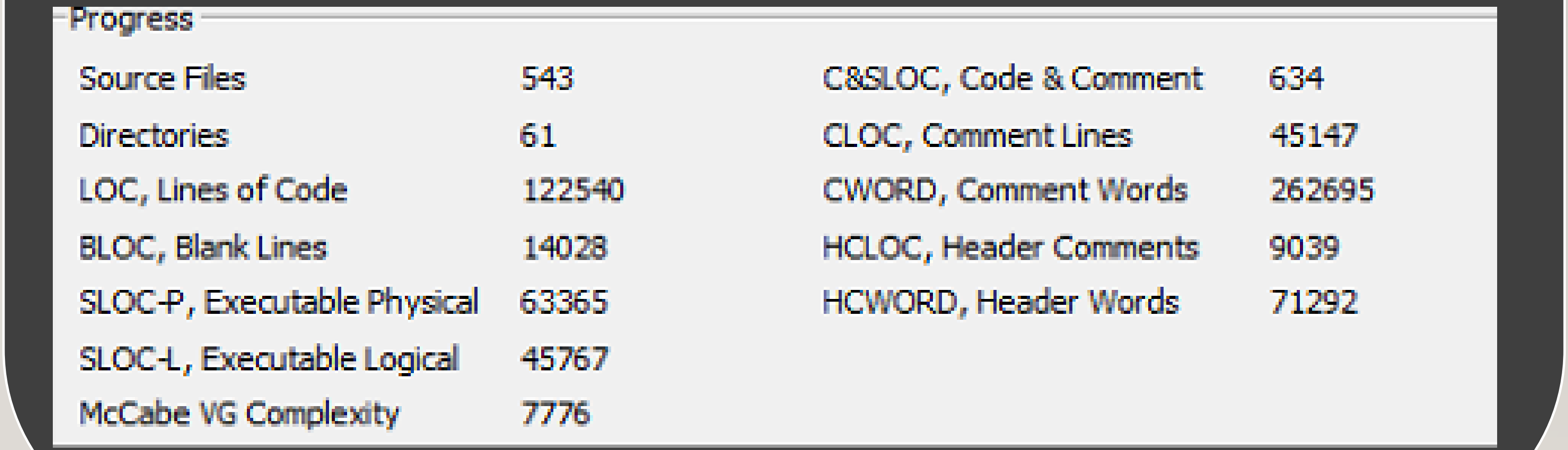
DEFINITION

- There are so many effort estimation models namely adaptive maintenance effort model which estimates the cost of software release this includes effort and time required to make changes.
- This model calculates the effort in person-hour.
- Few models prefer to use function points and/or algorithmic cost as discussed in the lecture.
- The adaptive maintenance effort model can also be used for regression analysis. Such analysis can be helpful to predict the future effort in machine learning model.

LocMetric tool is used to get the source lines of code. We have used SLOC-P and SLOC-L and calculated the sum (SLOC) of them to get the source lines of code of the project. Similarly, this operation is performed on all the versions of the product. Further, the DLOC is calculated by subtracting the SLOC of two consecutive versions of the product. Finally Effort is calculated from DLOC.

LocMetric OUTPUT EXAMPLE

The data collection using LocMetric is illustrated as below:



Progress

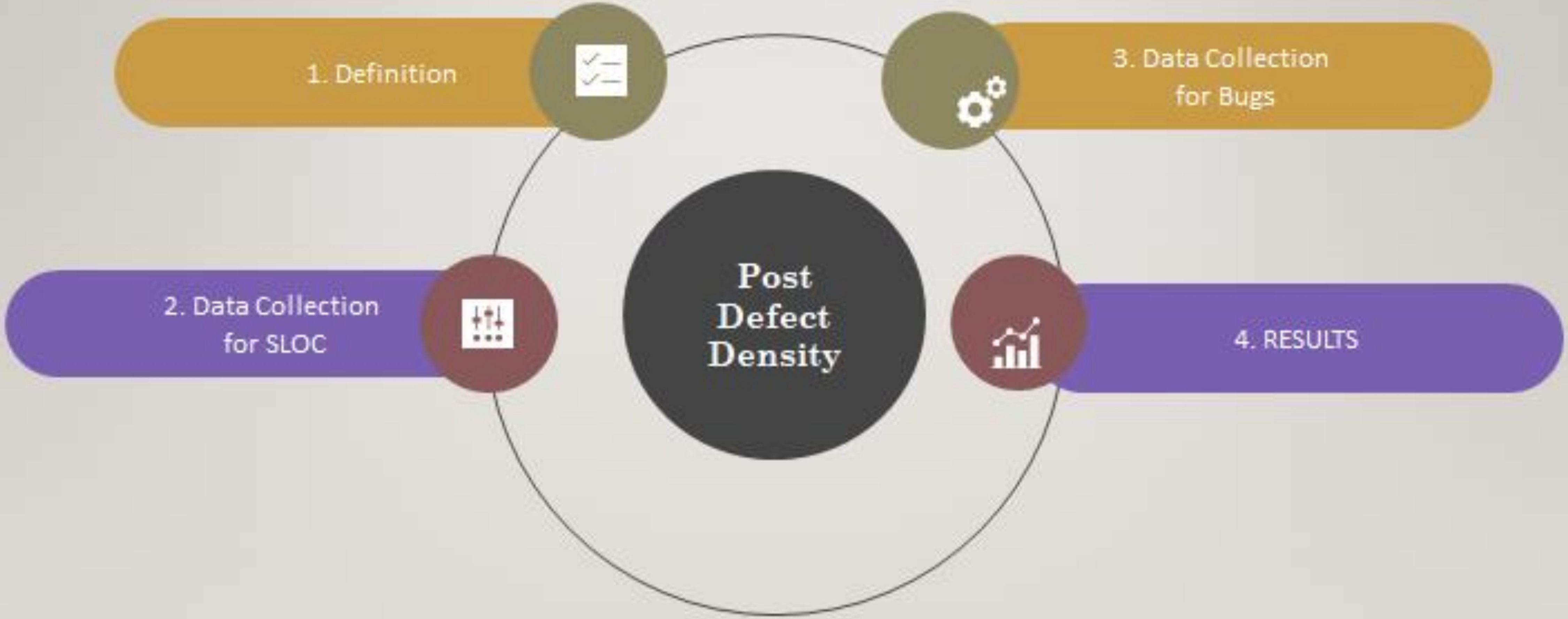
Source Files	543	C&SLOC, Code & Comment	634
Directories	61	CLOC, Comment Lines	45147
LOC, Lines of Code	122540	CWORD, Comment Words	262695
BLOC, Blank Lines	14028	HCLOC, Header Comments	9039
SLOC-P, Executable Physical	63365	HCWORD, Header Words	71292
SLOC-L, Executable Logical	45767		
McCabe VG Complexity	7776		

RESULTS

- Below Table shows Effort calculated for our selected Projects.

Metric	commons collections	commons-math	jfreechart	commons-net-NET
E	161604.96	736303.60	207183.84	12184.56

Metric 6



Post Defect Density

1. Definition

2. Data Collection for SLOC

3. Data Collection for Bugs

4. RESULTS

DEFINITION

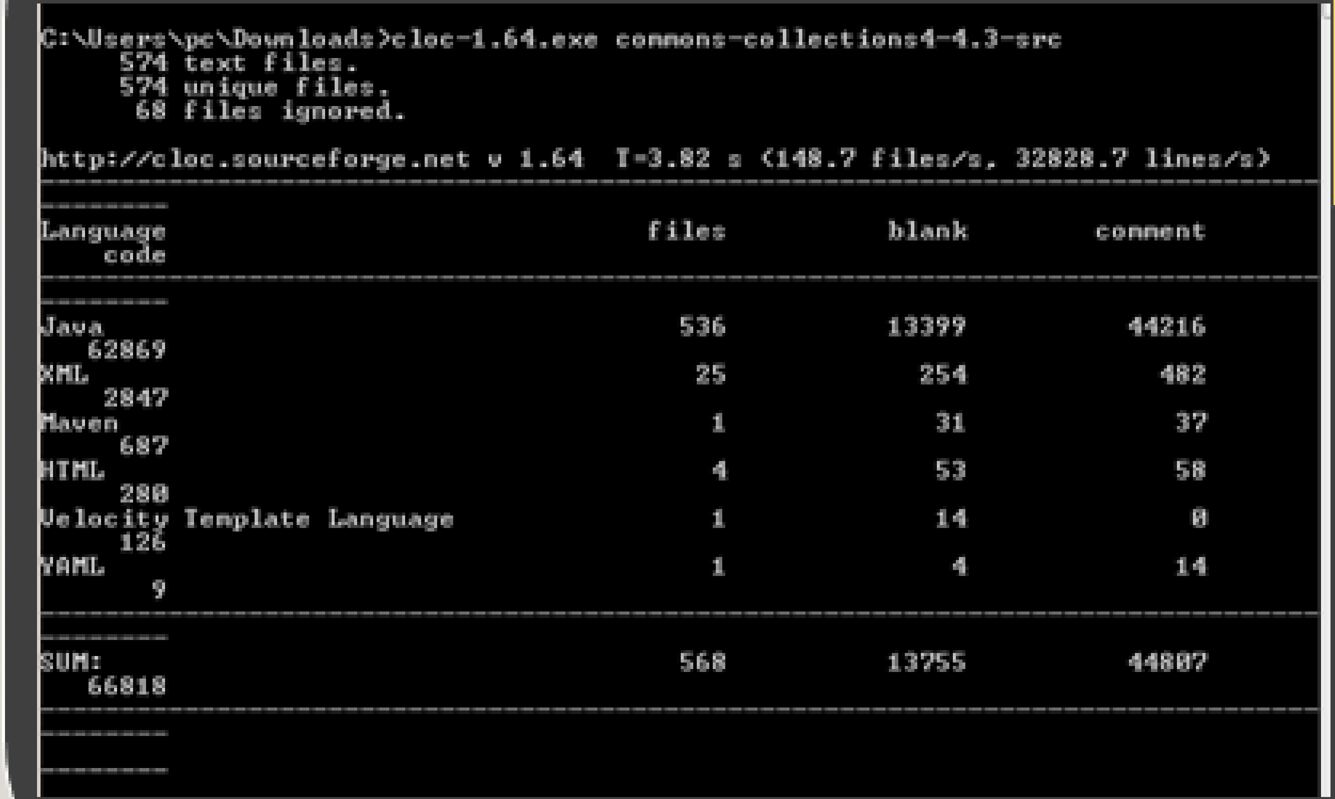
- Defect is what is the deviation observed from an expected behavior.Once the software is released to the customer and then if defects are encountered,is referred as post-release defects.
- It is defined as the number of defects confirmed in software/module **during a specific** period of **operation or** development divided by the size of the software/module.
- It is counted per thousand lines of code also known as KLOC.
- Post-release Defect Density can be used to improve the Quality of the software, which can be done during testing and development phase by improving the code quality..

Data Collection For SLOC

We need the source lines of code to calculate the size of the system. There are many tools available to gather this data. One such tool that we used is cloc (https://github.com/AlDanial/cloc/). To **run** **cloc** on **Windows** computers, one must first open up a command (aka DOS) window and invoke **cloc**.exe from the command line there.

SLOC Calculation Example

The data collection using this tool is illustrated as below:



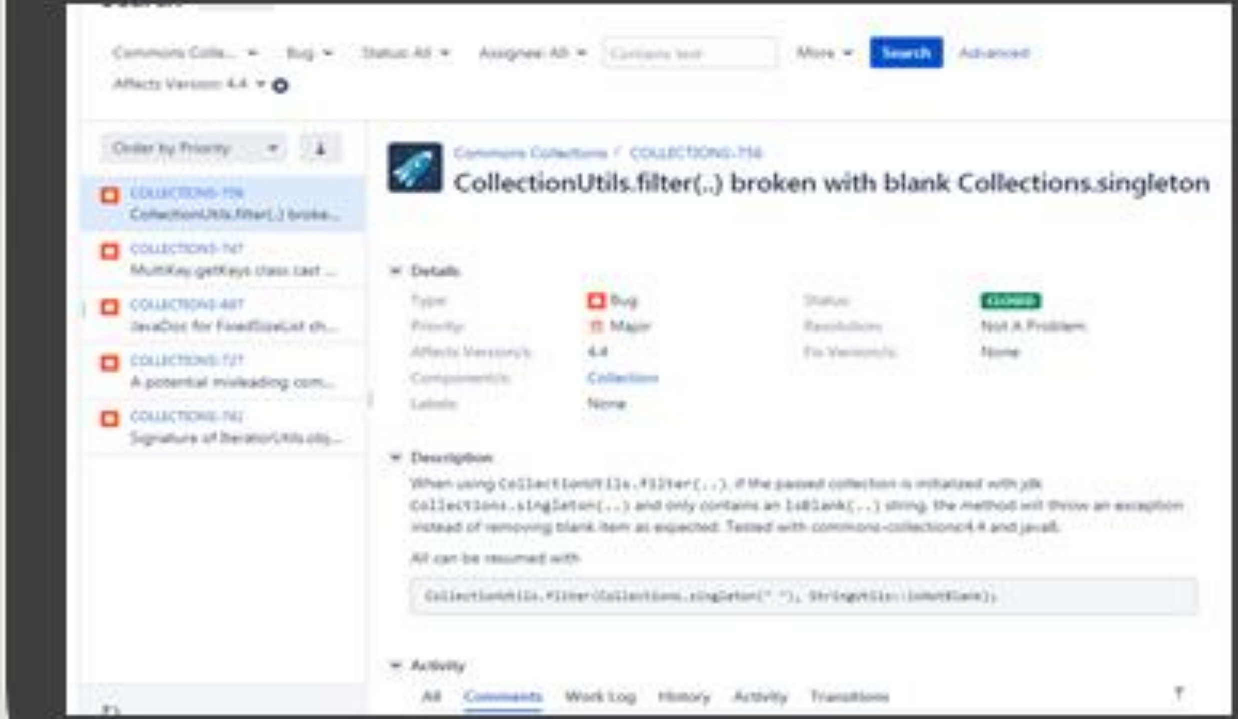
```
C:\Users\pc\Downloads>cloc-1.64.exe commons-collections4-4.3-src
     574 text files.
     574 unique files.
      68 files ignored.

http://cloc.sourceforge.net v 1.64  T=3.82 s (148.7 files/s, 32828.7 lines/s)
--------------------------------------------------------------------------------
--------
Language                                     files          blank        comment
    code
--------------------------------------------------------------------------------
--------
Java                                           536          13399          44216
   62869
XML                                             25            254            482
    2847
Maven                                            1             31             37
     687
HTML                                             4             53             58
     280
Velocity Template Language                       1             14              0
     126
YAML                                             1              4             14
       9
--------------------------------------------------------------------------------
--------
SUM:                                           568          13755          44807
   66818
--------------------------------------------------------------------------------
--------

--------
```

Data Collection For Defects

We can calculate the number of issues for different versions on Jira(Issue Tracker System) from the user interface and then export them as a CSV from the user interface itself. In order to achieve so, we first have to filter the issues on JIRA by the type“Bug” (the one with red icon). And we can apply another filter with the version number and then we can see the list.With the option on the top right corner,we can export the list to the CSV for further calculations

Jira Report Example

Data collected from the JIRA illustrated as below:



Commons Colle... ▾ Bug ▾ Status: All ▾ Assignee: All ▾ Contains text More ▾ Search Advanced

Affects Version: 4.4 ▾

Order by Priority ▾

- COLLECTIONS-756 CollectionUtils.filter(..) broke...
- COLLECTIONS-747 MultiKey.getKeys class cast ...
- COLLECTIONS-487 JavaDoc for FixedSizeList ch...
- COLLECTIONS-727 A potential misleading com...
- COLLECTIONS-762 Signature of IteratorUtils.obj...

Commons Collections / COLLECTIONS-756

CollectionUtils.filter(..) broken with blank Collections.singleton

▾ Details

Type:	Bug	Status:	CLOSED
Priority:	Major	Resolution:	Not A Problem
Affects Version/s:	4.4	Fix Version/s:	None
Component/s:	Collection		
Labels:	None		

▾ Description

When using CollectionUtils.filter(..), if the passed collection is initialized with jdk Collections.singleton(..) and only contains an isBlank(..) string, the method will throw an exception instead of removing blank item as expected. Tested with commons-collections4.4 and java8.

All can be resumed with:

```
CollectionUtils.filter(Collections.singleton(" "), StringUtils::isNotBlank);
```

▾ Activity

All Comments Work Log History Activity Transitions

DATASET FOR ALL THE TEST SUBJECTS

Project Name	Size	Version Control System	Issue Tracking System	Defects
Commons Net	63k	Git	Jira	27
Commons Math	388k	Git	Jira	48
Commons Collections	127k	Git	Jira	5
jFreeChart	297k	Git	Github	53

RESULTS

- Below Table shows Defect Density calculated for our selected Projects.

Metric	Commons Net	Commons Collections	Commons Math	jFreeChart
Post Defect Density	0.428571	0.039370079	0.12371134	0.16719

Correlations

As we have gathered data of all our metrics , we will calculate the correlation amongst them in order to determine how are they related.

We will be using both Spearman correlation and Pearson correlation to determine the relationship between our metrics.

How did we decide on the Coefficient?

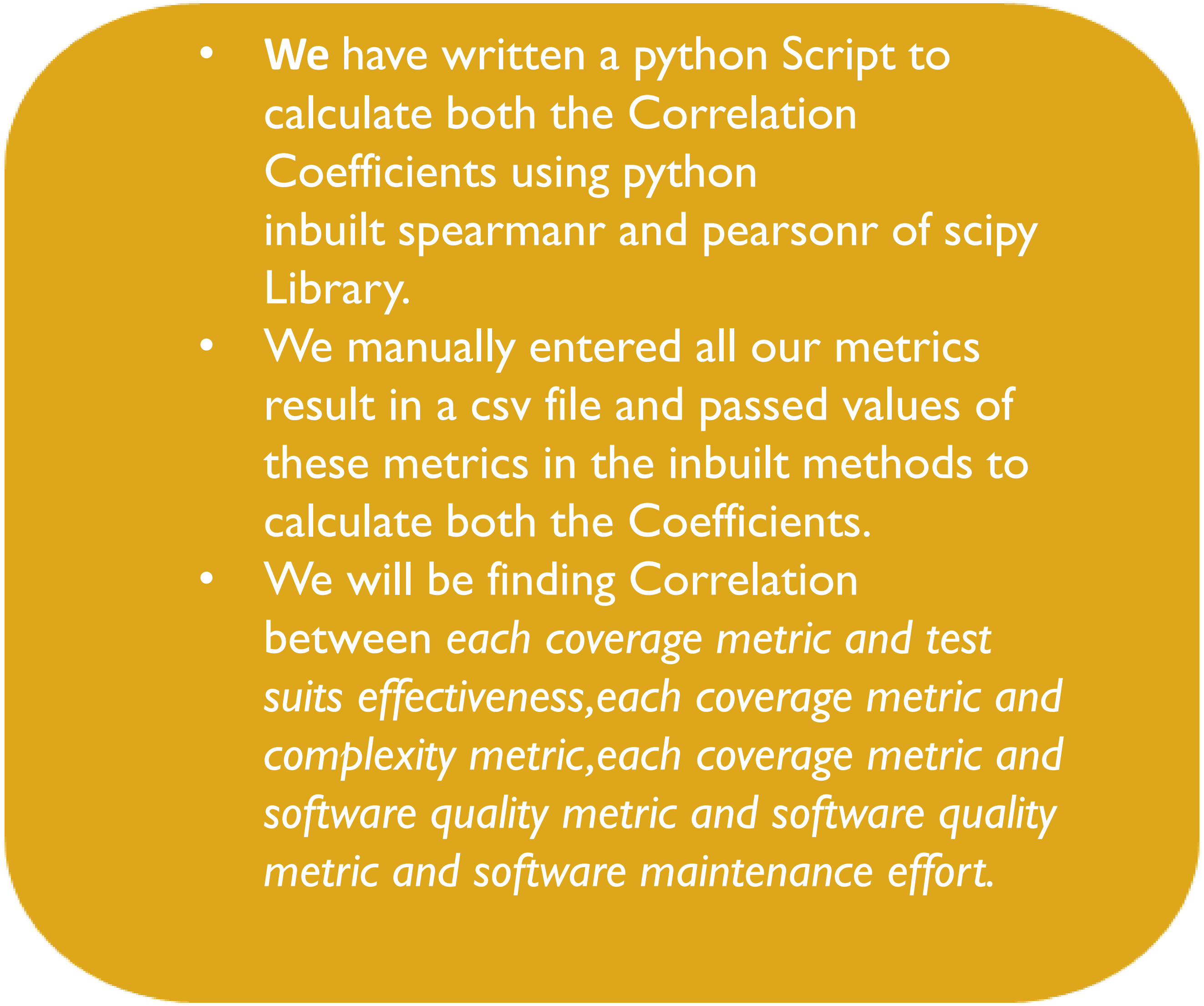
- Pearson

It 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.

Spearman

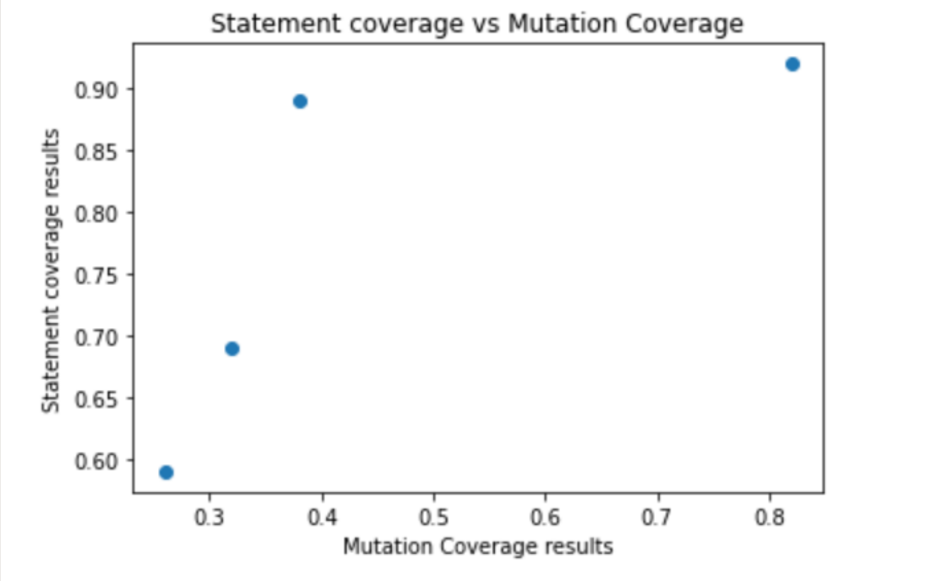
Spearman measures the monotonic relationship of the variables rather than the linear association in the Pearson setting. Thus, **Spearman's correlation coefficient** is more reliable with non-linear data compared to Pearson'

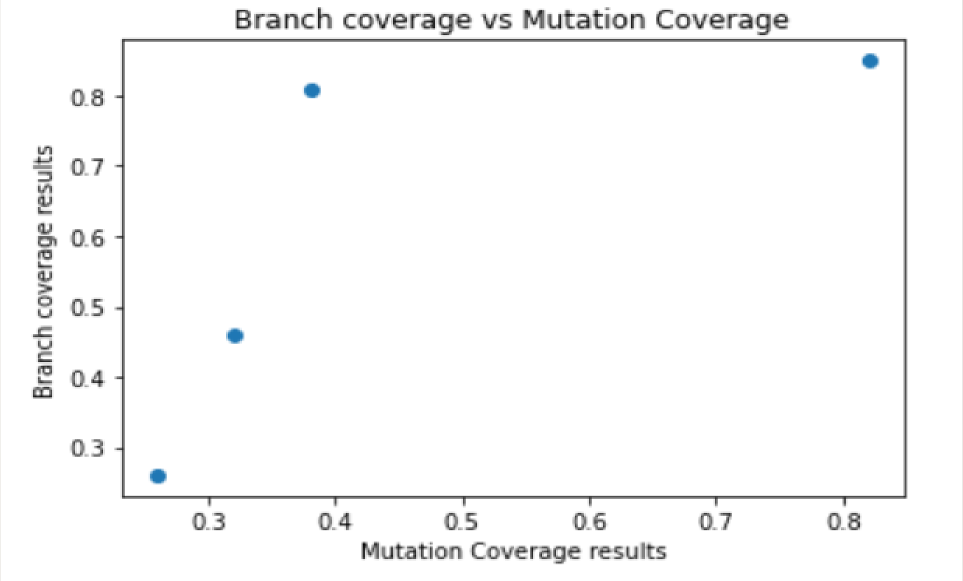
Our Approach



- **We** have written a python Script to calculate both the Correlation Coefficients using python inbuilt spearmanr and pearsonr of scipy Library.
- We manually entered all our metrics result in a csv file and passed values of these metrics in the inbuilt methods to calculate both the Coefficients.
- We will be finding Correlation between *each coverage metric and test suits effectiveness,each coverage metric and complexity metric,each coverage metric and software quality metric and software quality metric and software maintenance effort.*

RESULTS


Statement coverage vs Mutation Coverage
Statement coverage results
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.3
0.4
0.5
0.6
0.7
0.8
Mutation Coverage results


Branch coverage vs Mutation Coverage
Branch coverage results
0.3
0.4
0.5
0.6
0.7
0.8
0.3
0.4
0.5
0.6
0.7
0.8
Mutation Coverage results

THANK YOU!

Name	Student Id	Email
Iknoor Singh Arora	40082312	iknoorcan123@gmail.com
Sukhpreet Singh Bhatia	40083564	sukhpreetbhatia025@gmail.com
Aditi Bhayana	40083419	aditibhayana029@gmail.com
Aakash Ahuja	40082822	aakashahuja1993@gmail.com
Ashmeet Singh	40070369	ashu6811singh@gmail.com