Concordia University
Engineering and Computer Science

SOEN 6611
Software Measurement

1. Team Information

Team C

SN	Full Name	Student ID	Email
1	Shivam Nautiyal	40090841	shivam.nautiyal20@gmail.com
2	Nikunj Arora	40104832	nikunjarora333@gmail.com
3	Karan Sharma	40080005	95sharma.karan@gmail.com
4	Sardar Mutesham Ali	40094168	sardarms9@gmail.com
5	Saikiran Alagatham	40103833	saikiran.ask007@gmail.com

2. Selected Metrics and Correlation analysis

2.1 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.

Formula

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

Tools

Jacoco or Cobertura

2.2 Branch Coverage

- **Branch coverage** is a testing method, which aims to ensure that each one of the possible branches from each decision point is executed at least once and thereby ensuring that all reachable code is executed.

- By using the Branch coverage method, you can also measure the fraction of independent code segments. It also helps you to find out which sections of code don't have any branches.

Formula

Branch Coverage=No of executed branches/Total no of branches

Tools

Jacoco or Cobertura

2.3 Mutation Score

- **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.

- Each mutated version is called a mutant and tests detect and reject mutants by causing the behavior of the original version to differ from the mutant.

- The number of mutants depends on the definition of mutation operators and the syntax/structure of the software 100% mutation score means killing all mutants (or a random sample).

Formula

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

Tool

PITest Library

2.4 Cyclomatic complexity (McCabe complexity)

- **McCabe complexity** is a software metric used to measure the complexity of a program. It is a quantitative measure of independent paths in the source code of the program.

- An independent path is defined as a path that has at least one edge which has not been traversed before in any other path. Cyclomatic complexity can be calculated with respect to functions, modules, methods or classes within a program.

Formulas

a) **Cyclomatic Complexity = E – N + 2P,** where

E = the number of edges in CFG
N = the number of nodes in CFG
P = the number of connected components in CFG
D = is the number of control predicate (or decision) statements

b) **D + 1,** where

D = is the number of control predicate (or decision) statements

Tool

JaCoCo

2.5 Maintainability index

- Maintainability Index is a software metric that measures how maintainable (easy to support and change) the source code is. The maintainability index is calculated as a factored formula consisting of Lines Of Code, Cyclomatic Complexity and Halstead volume.

- The maintainability index is a value between 1 and 100. It represents the relative ease of maintaining the code. A high value means better maintainability.

Formula

MI = 171 - 5.2 * ln(V) - 0.23 * (G) - 16.2 * ln (LOC) , where

V = Halstead Volume
G = Cyclomatic Complexity
LOC = count of Source Lines of Code (SLOC)

Tool

Jira

2.6 Code Smells

- Code smells are a set of common signs which indicate that code is not good enough and it needs refactoring to finally have a clean code.

- There are various code smells like duplicated code, Long classes, Inappropriate Intimacy, Dead Code, etc.

Tool

SonarLint, SonarQube, Jdeodorant

3) Correlation Analysis

3.1) Correlation between Metric 1 & 2 and Metric 3

The rationale is if a program has a high branch as well as statement coverage it will be able to provide a better number of test suites. Thus, resulting in finding more bugs in a project.

3.2) Correlation between Metric 4 and Metric 1 & 2

The rationale behind this is if a program is having high complexity then it will have less branch as well as statement coverage because of the high number of executable paths.

3.3) Correlation between Metric 1 & 2 and Metric 6

Code smells have an impact on the developers' effort for certain kinds of activities related to software maintenance like higher "Feature Envy count" affect searching effort while high "Data Clumps" affects editing effort negatively which could lead to poor assessment of Statement or Branch coverage.

3.4) Correlation between Metric 5 and Metric 6

The rationale behind these metrics is that code smells are clear signs of design flaws that can affect the maintainability of the code. Low maintainability index indicates the higher number of code smells in code and vice versa is also true because more number of code smells means more unorganized and maintainable code.

4) Related Work

4.1) Metric 1 & 2

Code coverage is part of a feedback loop in the development process. As tests are developed, code coverage highlights aspects of the code which may not be adequately tested and which require additional testing[1]. This loop will continue until coverage meets some specified target.

4.2) Metric 3

Mutation Testing techniques and tools are reaching a state of maturity and applicability, while the topic of Mutation Testing itself is the subject of increasing interest[2].

4.3) Metric 4

The McCabe Cyclomatic Complexity measure is useful in determining the complexity of the implementation mechanism at the architecture level which is useful for the maintainability risk assessment at the architecture level[3].

4.4) Metric 5

Maintainability index as explained before can be calculated using metrics such as Lines Of Code, Cyclomatic Complexity and Halstead volume that can be collected using metrics 1,2 and 3.

4.5) Metric 6

Developers and tool providers should be wary of the presence of code smells because they not only impact the change-and fault-proneness of classes [4] but also the developers' efforts during their different activities.

5) Selected Open Source Systems

5.1) Apache Struts (Project 1)

Apache Struts is a free, open-source, MVC framework for creating elegant, modern Java web applications. It favors convention over configuration, is extensible using a plugin architecture, and ships with plugins to support REST, AJAX and JSON[5].

SLOC- JAVA: 120K

5.2) Apache Commons Configuration (Project 2)

The Commons Configuration software library provides a generic configuration interface which enables a Java application to read configuration data from a variety of sources. Commons Configuration provides typed access to single, and multi-valued configuration parameters. The project is built by maven and consists of many documents to track the problems and help solve them during analysis[6].

SLOC- JAVA: 67K

5.3) Apache Commons Digester (Project 3)

Many projects read XML configuration files to provide initialization of various Java objects within the system. There are several ways of doing this, and the Digestercomponent was designed to provide a common implementation that can be used in many different projects. Basically, the Digester package lets you configure an XML -> Java object mapping module, which triggers certain actions called rules whenever a particular pattern of nested XML elements is recognized. A rich set of predefined rules is available for your use, or you can also create your own[7].

SLOC- JAVA: 27K

5.4) Apache Commons Math (Project 4)

Commons Math Provides a productive environment for aggregation, testing and support of efficient Java implementations of commonly used mathematical and statistical algorithms. It 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[8].

SLOC- JAVA: 160K

6) Resource Planning

Name	Task
Karan Sharma	• Selected Project 1 • Collecting data and statistics for metric 1&2 and code smell. • Report Documentation
Saikiran Alagatham	• Project 2 • Collecting data for different versions and backlog maintainability index • Report Documentation
Sardar Mutesham Ali	• Project 3 • Analyzing the project for metrics 1&2 and also the versioning repository for BMI. • Report Documentation
Shivam Nautiyal	• Project 4 • Data for metrics 5&6. • Report Documentation
Nikunj Arora	• Project 4 • Data Collection for metric 3. • Report Documentation

References

[1]https://confluence.atlassian.com/clover/about-code-coverage-71599496.html
[2]Yue Jia Student Member, IEEE, and Mark Harman Member, IEEE. An Analysis and Survey of the Development of Mutation Testing.
https://web.eecs.umich.edu/~weimerw/481/readings/mutation-testing.pdf
[3]Ronald Tombe et al, International Journal of Computer Science and Mobile Computing, Vol.3 Issue.11, November- 2014, pg. 89-101.
www.ijcsmc.com
[4]Soh, Zéphyrin & Yamashita, Aiko & Khomh, Foutse & Guéhéneuc, Yann-Gaël. (2016). Do CodeSmells Impact the Effort of Different Maintenance Programming Activities?. 393-402. 10.1109/SANER.2016.103.
[5] https://struts.apache.org/

[6]https://github.com/apache/commons-configuration
[7]https://github.com/apache/commons-digester
[8]https://github.com/apache/commons-math