****

**Project Proposal**

**SOEN 6611 – Software Measurement**

**1. Team Information:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. no | Student ID | Full Name | E-mail |
| 1. | 40098439 | Harsh Mehta | harshmehta493@gmail.com |
| 2. | 40079362 | Manisha Jalota | manisha.jalota223@gmail.com |
| 3. | 40043821 | Navroop Virk | virknavroop@gmail.com |
| 4. | 40067532 | Raghav Dutta | raghav.dutta29@gmail.com |
| 5. | 40085663 | Yash Chandreshkumar Golwala | golwalayash@gmail.com |

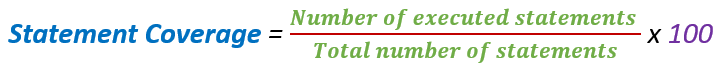
**2. Selected Metrics and Correlation analysis**

|  |  |
| --- | --- |
| **Metrics** | **Name** |
| Metrics 1 | Statement Coverage |
| Metrics 2 | Branch Coverage |
| Metrics 3 | Mutation Score |
| Metrics 4 | McCabe Cyclomatic Complexity |
| Metrics 5 | Adaptive Maintenance Effort Model |
| Metrics 6 | Software Defect Density (DD) |

**2.1 Detailed Rationalization of Metrics:**

**2.1.1 Statement Coverage and Branch Coverage:[10]**

Statement coverage is a white box test design technique which involves the execution of all the executable statements in the source code at least once.



**Branch Coverage-**

In Branch coverage, every outcome is tested . It makes sure that every branch is executed at least once. It also helps to determine the code which is independent and does not have any branches,

Branch Coverage = Number of Executed Branches / Total Number of Branches

**2.1.2 Mutation Score:[10]**

This is the kind of testing in which we can change the code and we will verify whether the introduced errors are detected by test cases. It measures the quality of the test cases. It is also called Fault based Testing Strategy .

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

**2.1.3 Cyclomatic Complexity (McCabe):**

Cyclomatic complexity is a quantitative measure of the number of linearly independent paths through program’s source code. Cyclomatic complexity is used as a benchmark to compare two different source code.[11] The program with high cyclomatic complexity is more error prone and require more understanding for testing. It also helps us in determining the number of test cases that will be required for complete branch coverage.[11]

Cyclomatic complexity is calculate with the help of number of edges(E), number of nodes(N) and number of connected point(P).[11]

* Cyclomatic Complexity =E – N + 2P

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

* Cyclomatic Complexity = D + 1

**2.1.4 Adaptive Maintenance Effort Model:**[1]

The correlation between some maintenance measures and maintenance effort. Among several measures empirically considered and found that the number of revised LOC is strongly correlated to maintenance effort measured in units of time.

**2.1.5 Software Defect Density (DD):**[10]

“Software defect density is defined as number of defects during the period of software development by the size of the software.”

DEFECT DENSITY=( Total No. of defects / size of the software);

Algorithm-

**Step 1:** Calculate the number of defects arising in different modules(release/built/cycle).

**Step 2:** Calculate the size of the software that can be measured in KLOC.

**2.2 Correlation Analysis:**

**Metrics 1&2 and metrics 3:**[10]

* Mutation score is used to design the new tests and evaluate the quality of existing software tests. It can be deducted from the test suite coverage as the branch testing and statement testing calculates all the possible test cases and paths which forms a test suite coverage and tests can be deducted through that only.
* Higher the test suite coverage shows better the test suite effectiveness.
* Correlation between one complexity metric (Metric 4) and each coverage metric

(Metric 1&2). The rationale is that classes with higher complexity are less likely to

have high coverage test suites.

**Metric 4 and Metric 1&2:**[10]

* McCabe Cyclomatic complexity is calculated by calculating the number of execution paths in a program.
* Each Coverage metrics calculates the number of execution paths in a program. Hence, the cyclomatic complexity is calculated using the branch coverage metrics only.

**Metric 6 and Metric 1&2:**[10]

* Software defect density is calculated by using the coverage metric only as it calculates branch testing and statement testing calculates the number of test cases considering each branch and each statement respectively.
* Higher the number of test cases, lesser the bugs found in the system
* Coverage testing has a positive effect on metric 6.

**Metric 5 and Metric 6:**

* Defect density helps us to compare small and large file by normalizing number of defects to the amount of code review. Variation in defect density has lot to do with file’s “risk” in the system. With the help of Metric 5, we can calculate person-time which help us to calculate inspection rate and defect rate.[9]

Inspection rate= (Lines of code reviewed) / (Total person-time)

* Manager might insist on a slower inspection rate especially on stable branch, close to product release or core module, when everyone wants to be more careful with code change. Also, inspection rate helps us to predict amount of time required to change code.[9]

Defect Rate= (Number of defects) / (Total Person-Time)

* Defect rate helps us to identify at which rate, reviewer uncover defects in code.[9]

**3. Selected Open-Source Systems**

Below is the list of projects that will be considered for our Analysis purpose.

|  |  |  |
| --- | --- | --- |
| Projects | Versions | Lines of Code (LOC) |
| Apache Commons Pool | 2.6.2 | 14.3K |
| Apache Commons Math | 3.5.1 | 174K |
| Apache Commons FileUpload | 1.4 | 100K |
| Apache Commons IO | 2.6 | 58.4K |
| Apache Commons Collections | 4.4 | 270K |

These projects have high number of users and commits. Many Developers have worked on these projects which gives us a good chance to start our analysis on different metrics which we have studied. Above mentioned projects have several bugs and features reported which provide vision to understand the metrics. Good number of system releases are there which can aids us to understand system better with stable builds.

**4. Resource Planning**

|  |  |  |
| --- | --- | --- |
| Project | Metric | Analysis done by |
| Apache Commons Pool | Metric 1,5,6 | Harsh Mehta,  Yash Golwala |
| Apache Commons Math | Metric 2,5,6 | Harsh Mehta,  Yash Golwala |
| Apache Commons FileUpload | Metric 3,5,6 | Manisha Jalota,  Navroop Virk |
| Apache Commons IO | Metric 4,5,6 | Navroop Virk,  Manisha Jalota |
| Apache Commons Collections | Metric 5,6 | Raghav Dutta |

**SYSTEM 1**

**Name: Apache Commons Pool**

**Description:**

* The Apache Commons Pool open source software library provides an object-pooling API and a number of object pool implementations. [4]
* In addition to performance and scalability improvements, version 2 includes robust instance tracking and pool monitoring.[4]

**Version:** Apache Common Pool 3.9

**Rationale:**

* Apache Commons Pool has 14.3k lines of code.
* It’s an active project with large number of contributors and 1900 commits on Git. [5]

**SYSTEM 2**

**Name: Apache Commons Math**

**Description:** [7]

* The Apache Commons Math project is a library of lightweight, self-contained mathematics and statistics components addressing the most common practical problems not immediately available in the Java programming language or commons-lang.

**Guiding principles:**

1. Real-world application use cases determine development priority.
2. This package emphasizes small, easily integrated components rather than large libraries with complex dependencies and configurations.
3. All algorithms are fully documented and follow generally accepted best practices.
4. In situations where multiple standard algorithms exist, a Strategy pattern is used to support multiple implementations.
5. Limited dependencies. No external dependencies beyond Commons components and the core Java platform (at least Java 1.3 up to version 1.2 of the library, at least Java 5 starting with version 2.0 of the library).

**Version:** Apache Commons Math 3.6.1

**Rationale:**

* Apache Commons Math has 174.5k lines of code, 6408 commits by 31 Contributors.
* It has 0.52 Comment to Code ratio.
* Most recent commit is 2 days ago. [6]

**SYSTEM 3**

**Name: Apache JMeter**

**Description:**

* Apache JMeter is an open source software, 100% JAVA based application which is designed to load the test functional. [2]
* Apache JMeter is used to test performance for both static and dynamic resources, Web Dynamic applications. [2]
* It simulates the behavior of multiple users acting simultaneously on a Web Application. As it has developed, it has been extended to test other types of applications. [2]
* Its main features are: performance test and robustness test. [2]

**Version:** Apache JMeter 5.1.1

**Rationale:**

* Apache JMeter has 34.8k lines of code, 14 commits by 9 PMC members.
* Apache JMeter uses BugZilla to track the bugs.
* The last commit was done on 2019-03-13.
* The project status is still active.

**SYSTEM 4**

**Name: ImageJ**

**Description:**

ImageJ is an open source Java image processing program inspired by [NIH Image](https://imagej.net/NIH_Image) It runs, either as an online applet or as a downloadable application, on any computer with a Java 1.4 or later virtual machine. Downloadable distributions are available for Windows, Mac OS, Mac OS X and Linux. ImageJ was designed with an open architecture that provides extensibility via Java plugins.

Custom acquisition, analysis and processing plugins can be developed using ImageJ’s built in editor and Java compiler. User-written plugins make it possible to solve almost any image processing or analysis problem. [8]

**Version:** ImageJ2

**Rationale:**

➢ The project has had 9141 commits made by 15 contributors.

➢ It is mostly written in Java with a low number of source code comments.

➢ It has a well-established, mature codebase maintained by a small development team with

increasing Y-O-Y commits.

➢ Its most recent commit is 14 days before.

**SYSTEM 5**

**Name: Checkstyle**

**Description:**

Checkstyle is a development tool which helps programmers to write java code adhering to coding standard. It automates the process of checking java code and spare humans to the task. It is highly configurable according to the requirement for programmers. [3]

**Version:** 8.20

**Rationale:**

* Current version of project has 270k lines of code.
* Most recent commit is a day ago.
* Given project has had 8,348 commits which are made by 191 contributors.
* Given project has had around 488k total instructions.
* Licensed by Lesser GNU General Public License v2.1.

**5. References**

[1] J. Hayes, S. Patel and L. Zhao, "A metrics-based software maintenance effort model," *IEEE,* 2004.

[2] Foundation, A. S. (2019). *Apache JMeter.* Berlin: Apache Software Foundation.

[3] <http://checkstyle.sourceforge.net/>

[4] APACHECON. (2019). *Apache commons pool.* Las Vegas: Apache Software Foundation.

[5] https://github.com/apache/commons-pool

|  |  |
| --- | --- |
|  | [6] Gregory, G. (2006). *Apache commons Math* (Vol. 3.5). Gary Gregory.  [7] <https://commons.apache.org/proper/commons-math/>  [8] <https://imagej.nih.gov/ij/docs/intro.html>  [9] Smartbear. (2018). *Metrics Definitions.* Ireland: Smartbear.  [10] Guru99, "Mutation Testing in Software Testing: Mutant Score & Analysis Example," *https://www.guru99.com/mutation-testing.html,* 2019.  [11] M. M. S. Sarwar, S. Shahzad and I. Ahmad, "Cyclomatic complexity: The nesting problem," *IEEE,* 2014. |
|  |  |