# Team Information

Team B

|  |  |  |
| --- | --- | --- |
| Name | Student ID | Email |
| Nirav Ashvinkumar Patel | 40081268 | niravjdn@gmail.com |
| Himansi patel | 40072262 | himansipatel1994@gmail.com |
| Darwin Anirudh Godavari | 40093368 | d\_godava@encs.concordia.ca |
| Jayaprakash Kumar | 40083709 | jayaprakash6034@gmail.com |
| Krishnan krishnamoorthy | 40089054 | krish27794@gmail.com |

# Selected Metrics and Correlation analysis

**Metric 1&2**​: Two test coverage metrics, i.e., statement coverage and branch coverage. Test coverage metrics are measured from existing developers’ test cases

We will use EclEmma and Jacoco for test coverage[1].

For Test Coverage, We will use following procedure.

* (A) the total lines of code in the piece of software you are testing, and
* (B) the number of lines of code all test cases currently execute, and
* Find (B divided by A) multiplied by 100 – this will be your test coverage %

By using Branch coverage method, we can also measure the fraction of independent code segments. It also helps us to find out which is sections of code don't have any branches. [2][3]

Branch Coverage = Branch Executed / Total No of Branch

**Metric 3**​: Test suite effectiveness,

Test Suite Effectiveness can be measured using mutation metric.

The normalized effectiveness measurement is the number of mutants a test suite detected divided by the number of non-equivalent mutants it covers. A test suite covers a mutant if the mutant was made by altering a line of code that is executed by the test suite, implying that the test suite can potentially detect the mutant.[7]

We will use PIT as a helper tool for mutation testing, consequently for calculating Mutation Score.

**Metric 4**​: One complexity metric (e.g., McCabe, Halstead)

McCabe Complexity Metric will be calculated by Jacoco, a plugin of eclipse.[4][5]

**Metric 5**​:

Maintainability of a software can be predicted using MOOD and CK Metrics. MOOD helps in determining system level maintainability while the CK metrics at class level.

CK Metric clearly shows that by keeping low values of CK metrics, the system developers can invariably improve the maintainability of software systems for a qualitative utility in this important field.[9]

To measure maintenance effort at system level, we’ll use MOOD Metric to measure system internal structure, hence measuring system maintainability. As a part of MOOD measure, we’ll measure Methods Hiding Factor,Attributes Hiding Factor, Methods Inheritance Factor, Attributes Inheritance Factor, Polymorphism Factor, Coupling Factor.

[10]

Our hypothesis is that, MIF, CF and PF better correlate with defect density.

**Metric 6**​:

As a software quality metric, we’ll calculate Post Release Defect Density Metric. Our hypothesis is that code coverage has an insignificant correlation to the number of bugs as well as to other metrics such as number of bugs/LOC and number of bugs/complexity found after the release of the software.[8]

To calculate Post release defect density, we will use issue tracker of the respective

project, and will calculate it using this formula.

Post Release Defect Density = Defect count(No of bugs)/size of the release

For ​**Metric 5**​ and ​**6**​, you need to properly define and describe them: algorithm (how to calculate the metric from software artifacts), and/or text descriptions. Also, you should properly cite related work if the definition or a similar definition is defined by current literature.

|  |  |
| --- | --- |
|  | **Hypothesis** |
| 1 | Higher Test Coverage might show better test suite effectiveness |
| 2 | Higher Mccabe Complexity will have lower test coverage |
| 3 | Code coverage has an insignificant correlation to the number of bugs as well as to other metrics such as defect density. |
| 4 | CF has high positive correlation with defect density.PF has moderate to high negative correlation with defect density. MIF has a moderate negative correlation with fault density. |

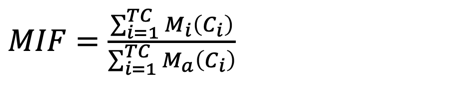
# Related Work

The Paper compares different tools for calculating code coverage. It also states which tools provide statement and branch coverage both. It summarize that JCover as best tool for calculating coverage. However, we analyzed other tools on our own and decided to use jacoco.[6]

The Paper States that there is a low to moderate correlation between coverage and effectiveness when the number of test cases in the suite is controlled for. In addition, It says that stronger forms of coverage do not provide greater insight into the effectiveness of the suite. Their results suggest that coverage, while useful for identifying under-tested parts of a program, should not be used as a quality target because it is not a good indicator of test suite effectiveness.[7]

This paper describes the results of a study where the impact of Object-Oriented design on software quality characteristics is experimentally evaluated. A suite of metrics for OO design called MOOD was adopted to measure the use of OO design mechanisms. Data collected on the development of eight small-sized information management systems based on identical requirements was used to assess the referred impact. The results obtained in this experiment show how OO design mechanisms like inheritance, polymorphism, information hiding and coupling influence quality characteristics such as defect density and rework. Predictive models based on OO design metrics built in this study are also presented.[10]

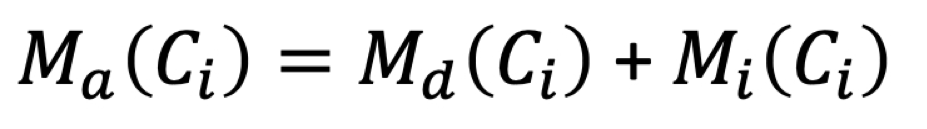
For the calculation of metric 5, We will calculate following metrics.

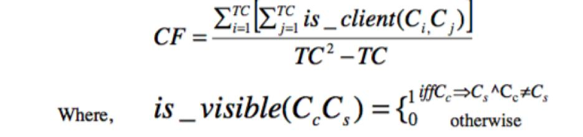
**Method Inheritance Factor (MIF)**:

Where,

Mi(Ci) is the number of methods inherited (and not overridden) in class Ci.

Md(Ci) is the number of methods actually declared in class Ci

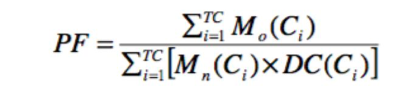


**Coupling Factor (CF):** ​The Coupling Factor (CF) metric is a measure of coupling and proposed as a measure of coupling between classes excluding coupling due to inheritance. The coupling factor metric has been defined as: 

And the C​c => C​s r​epresents the relationship between a client class C​c​ and a supplier class C​s

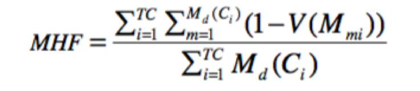
**Polymorphism Factor (PF)**: ​The Polymorphism Factor (PF) metric is proposed for a

measure of polymorphism in object-oriented environment. The PF metric is formally defined as:

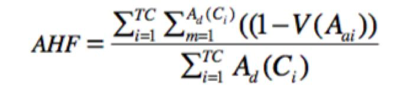


M​d(​C​i)​ =M​n(​C​i)​ +M​o(​C​i)​ and TC is the total number of classes and M​n(​C​i​) is the number of new methods and M​o(​ C​i)​ is the number of overriding methods and D​C(​ C​i)​ is the descendants count.

**Method Hiding Factor (MHF):** ​The Method Hiding Factor (MHF) metric is a measure of encapsulation and is formally defined as:

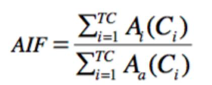


Where M​d(​C​i) ​is the number of methods declared in a class C​i a​nd TCi s the total number of classes.

**Attribute Hiding Factor (AHF):** ​The Attribute Hiding Factor (AHF) metric is to measure the attribute hiding and is formally defined as:

Where A​d(​C​i) ​is the number of attributes declared in a class C​ia​nd TC is the total number of classes.

**Attribute Inheritance Factor (AIF)**: ​The Attribute Inheritance Factor (AIF) is a measure of inheritance properties metric and it is formally defined as:



Where A​a(​ C​i)​ = A​d(​ C​i)​ + A​i(​ C​i​) and TC is total number of classes.

This paper consist of analytics of 100 large open-source Java projects and measure the code coverage of the test cases that come along with these projects. The authors collect real bugs logged in the issue tracking system after the release of the software and analyse the correlations between code coverage and these bugs. They also collect other metrics such as cyclomatic complexity and lines of code, which are used to normalize the number of bugs and coverage to correlate with other metrics as well as use these metrics in regression analysis. Their results show that coverage has an insignificant correlation with the number of bugs that are found after the release of the software at the project level, and no such correlation at the file level.[8]

We will use number of different tools as we come to know the requirements eventually such as JDeodorant, Jacoco, JCover, etc.

# Selected Open-Source Systems

There are the five open source software systems we are going to analyze.

1. Apache Common Collections (132k LOC) - (4.2)
2. Apache Commons Configuration (847k LOC) - (2.4)
3. Apache Commons Math (186k LOC) - (3.6.1)
4. Apache Common Lang (80k LOC) - (3.8.1)
5. Apache Commons IO (35k LOC) - (2.6)

# Resource Planning

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **T1** | **T2** | **T3** | **T4** | **T5** | **T6** | **T7** |
| Nirav Patel | Y |  | Y | Y |  | Y | Y |
| jayaPrakash | Y | Y |  |  | Y |  | Y |
| Himansi | Y |  | Y |  | Y |  | Y |
| Darwin | Y |  |  | Y |  | Y | Y |
| Krishnan | Y | Y |  |  | Y |  | Y |

T1 – Literature review

T2 – Scheduling

T3 – Supporting Documentation

T4 – External tools

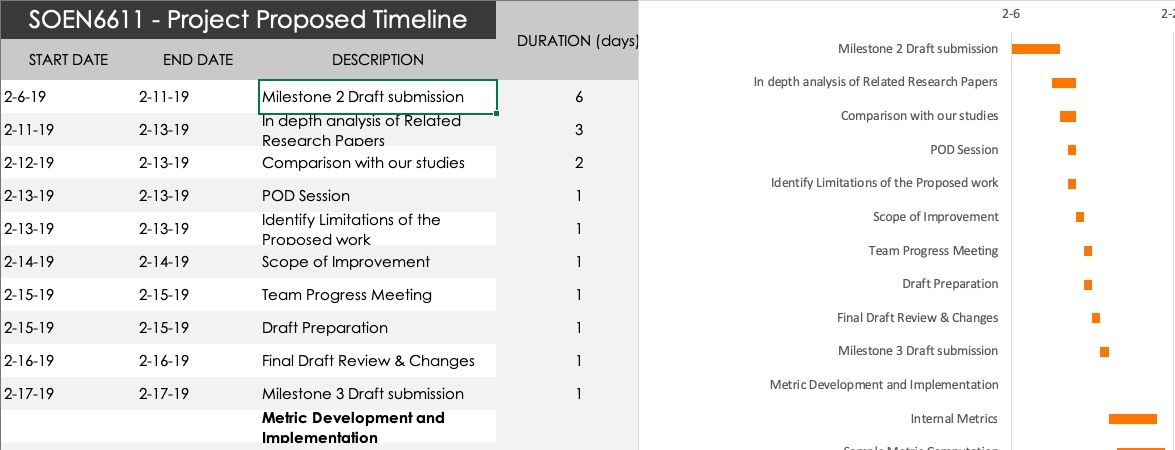
T5- Metric Study

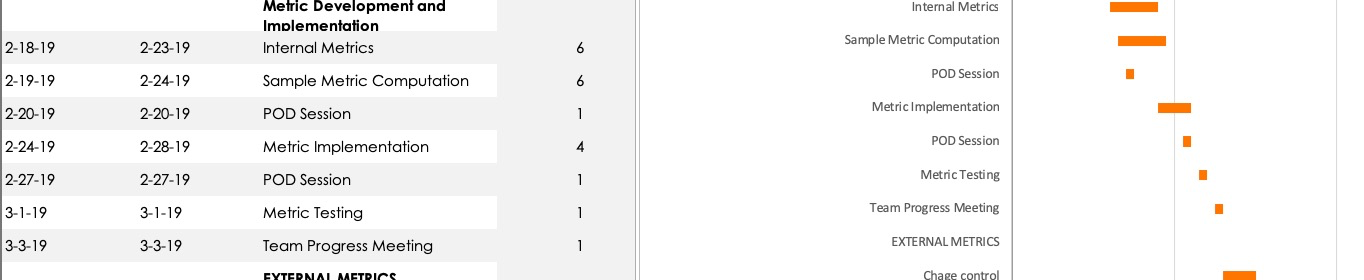
T6 – Install and Build

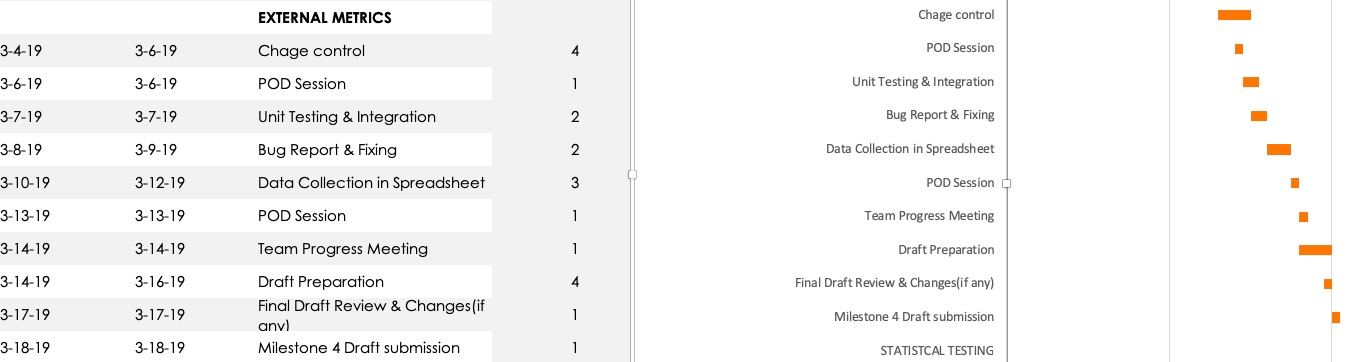
T7 – Compiling and Documentation

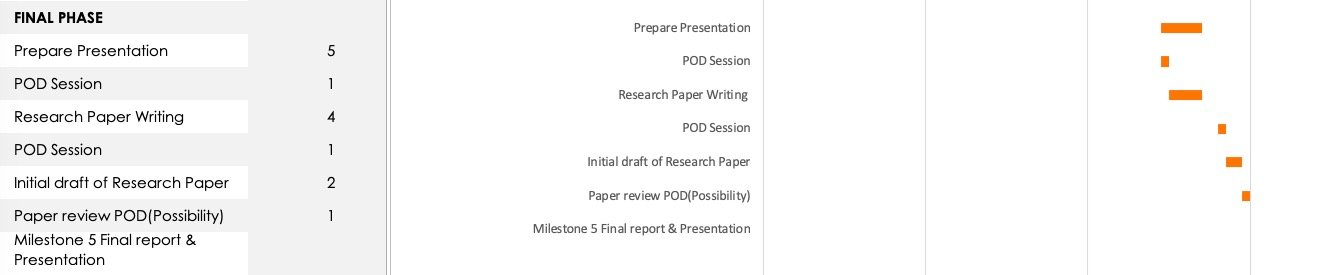
Nirav and Darwin will also be dealing with any automations(e.g selenium scripts to capture details from repository or issue tracking system).

Below is the Gantt Chart that covers detailed tasks.









# References

[1] Java Code Coverage for Eclipse Available: <https://www.eclemma.org/> Accessed on: 02/01/2019

[2] Code Coverage Metrics

Available: <https://www.sealights.io/test-metrics/code-coverage-metrics/> Accessed on: 02/01/2019

[3] Code Coverage Tutorial: Branch, Statement, Decision, FSM Available: <https://www.guru99.com/code-coverage.html#6> Accessed on: 01/31/2019

[4] How to calculate McCabe cyclomatic complexity in Java Available: <https://www.theserverside.com/feature/How-to-calculate-McCabe-cyclomatic-complexity-in-Java>

Accessed on: 02/02/2019

[5] Eclemma.org. (2019). *JaCoCo - Coverage Counter*. [online] Available at: <https://www.eclemma.org/jacoco/trunk/doc/counters.html>. Accessed on: 7 February 2019

[6] Dubey, S., & Rana, A. (2011). Assessment of maintainability metrics for object-oriented software system. *ACM SIGSOFT Software Engineering Notes*, *36*(5), 1. doi:10.1145/2020976.2020983

[7] Laura Inozemtseva and Reid Holmes. Coverage Is Not Strongly Correlated with Test Suite Effectiveness (2019). *Linozemtseva.com*. Accessed on: 7 February 2019

Available : http://www.linozemtseva.com/research/2014/icse/coverage/coverage\_paper.pdf

[8] Pavneet Singh Kochhar, David Lo, Julia Lawall, Nachiappan Nagappan. Code Coverage and Postrelease Defects: A Large-Scale Study on Open Source Projects. IEEE Transactions on Reliability, Institute of Electrical and Electronics Engineers, 2017, 66 (4), pp.1213 - 1228. <10.1109/TR.2017.2727062>.

[9] Dubey, S., & Rana, A.. Assessment of maintainability metrics for object-oriented software system. *ACM SIGSOFT Software Engineering Notes*, *36*(5), 1. doi:10.1145/2020976.2020983

[10] F. Brito e Abreu, W. Melo Evaluating the impact of object-oriented design on software quality - IEEE Conference Publication. (2019). Ieeexplore.ieee.org. Retrieved 9 February 2019, Available on: https://ieeexplore.ieee.org/document/492446

Michura, J., Capretz, M., & Wang, S. (2013). Extension of Object-Oriented Metrics Suite for Software Maintenance. *ISRN Software Engineering*, *2013*, 1-14.