1. 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	darwinanirudh@gmail.com
Jayaprakash Kumar	40083709	jayaprakash6034@gmail.com
Krishnan Krishnamoorthy	40089054	krish27794@gmail.com

2. 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:

We'll measure a maintenance effort as

Churned LOC / Total LOC

Churned LOC = (added + changed) LOC

We expect the larger the proportion of churned (added + changed) code to the LOC of the new binary, the larger the magnitude of the defect density for that binary will be.

We'll use git functionalities for calculating manipulations.

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(statement and branch)
3	Code coverage has an insignificant correlation to the number of bugs as well as to other metrics such as defect density.
4	We expect the larger the proportion of churned (added + changed) code to the LOC of the new binary, the larger the magnitude of the defect density[9]

3. 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 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]

This paper states that Code churn, which measures the changes made to a component over a period of time, quantifies the extent of this change and present a technique for prediction of system defect density using a set of relative code churn measures that relate the amount of churn to other variables such as component size and the temporal extent of churn. [9]

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

4. Selected Open-Source Systems

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

No	Name	Version	SLOC	VCS	Issue Tracker
1	Apache Common Collections	4.2	132k	Yes	Yes
2	Apache Common Configuration	2.4	847k	Yes	Yes
3	Apache Common Math	3.6.1	186k	Yes	Yes
4	Apache Common Lang	3.8.1	80k	Yes	Yes
5	Apache Common IO	2.6	35k	Yes	Yes

5. 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

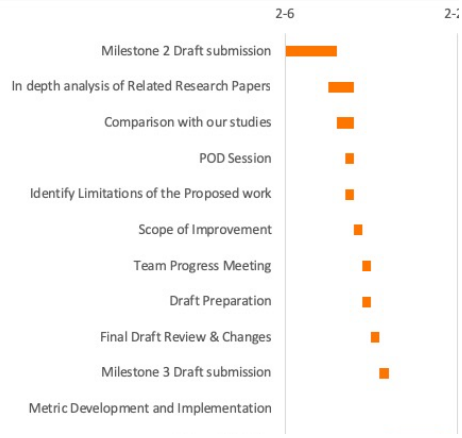
Darwin	Y			Y		Y	Y
Krishnan	Y	Y			Y		Y

T1 – Literature review
T2 – Scheduling
T3 – Supporting Documentation
T4 – External tools
T5- Metric Study
T6 – Install and Build
T7 – Compiling and Documentation

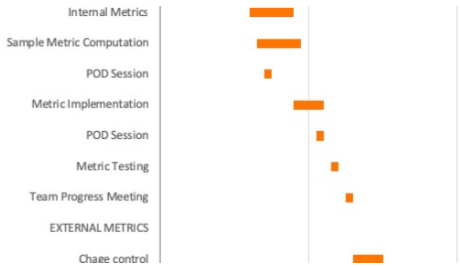
Nirav and Darwin will also be dealing with any automations(e.g selenium scripts to capture details from repository or issue tracking system).

Below is the Gantt Chart that covers detailed tasks.

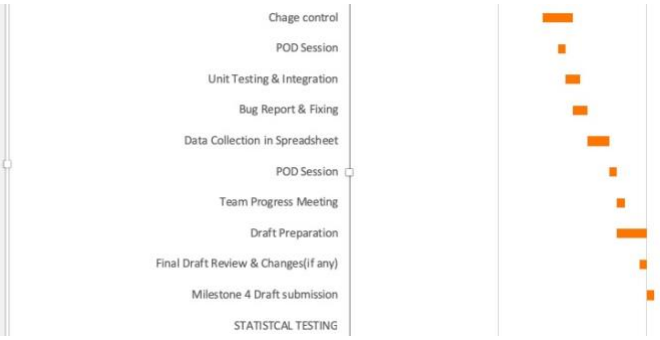
SOEN6611 - Project Proposed Timeline			DURATION (days)
START DATE	END DATE	DESCRIPTION	
2-6-19	2-11-19	Milestone 2 Draft submission	6
2-11-19	2-13-19	In depth analysis of Related Research Papers	3
2-12-19	2-13-19	Comparison with our studies	2
2-13-19	2-13-19	POD Session	1
2-13-19	2-13-19	Identify Limitations of the Proposed work	1
2-14-19	2-14-19	Scope of Improvement	1
2-15-19	2-15-19	Team Progress Meeting	1
2-15-19	2-15-19	Draft Preparation	1
2-16-19	2-16-19	Final Draft Review & Changes	1
2-17-19	2-17-19	Milestone 3 Draft submission	1



		Metric Development and Implementation	
2-18-19	2-23-19	Internal Metrics	6
2-19-19	2-24-19	Sample Metric Computation	6
2-20-19	2-20-19	POD Session	1
2-24-19	2-28-19	Metric Implementation	4
2-27-19	2-27-19	POD Session	1
3-1-19	3-1-19	Metric Testing	1
3-3-19	3-3-19	Team Progress Meeting	1
		EXTERNAL METRICS	



		EXTERNAL METRICS	
3-4-19	3-6-19	Chage control	4
3-6-19	3-6-19	POD Session	1
3-7-19	3-7-19	Unit Testing & Integration	2
3-8-19	3-9-19	Bug Report & Fixing	2
3-10-19	3-12-19	Data Collection in Spreadsheet	3
3-13-19	3-13-19	POD Session	1
3-14-19	3-14-19	Team Progress Meeting	1
3-14-19	3-16-19	Draft Preparation	4
3-17-19	3-17-19	Final Draft Review & Changes(if any)	1
3-18-19	3-18-19	Milestone 4 Draft submission	1



FINAL PHASE	
Prepare Presentation	5
POD Session	1
Research Paper Writing	4
POD Session	1
Initial draft of Research Paper	2
Paper review POD(Possibility)	1
Milestone 5 Final report & Presentation	

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] *A Multipurpose Code Coverage Tool for Java - IEEE Conference Publication*. (2019). *Ieeexplore.ieee.org*. Accessed on: 01 February 2019, Available on: https://ieeexplore.ieee.org/document/4076910

[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] *Code churn: a measure for estimating the impact of code change - IEEE Conference Publication*. (2019). *Ieeexplore.ieee.org*. Accessed on: 11 February 2019, Available on https://ieeexplore.ieee.org/document/738486

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

IEEE style guide · Citing · Help & how-to · Concordia University Library. (2019). Library.concordia.ca. Accessed on: 11 February 2019, Available on: https://library.concordia.ca/help/citing/ieee.php

https://library.concordia.ca/help/citing/ieee.php