**Project preparation/proposal**

**SOEN 6611**

**Dr. Rodrigo Morales Alvarado**

**Table of Contents**

Content Page Number

[1. Team Formation](https://docs.google.com/document/d/1dHZ5ffOKQlTGUIi3EbvEex-oVYzGT05Et3G8PyFNtwM/edit#heading=h.hfejj3qloa44) 2

[2. Type of the study](https://docs.google.com/document/d/1dHZ5ffOKQlTGUIi3EbvEex-oVYzGT05Et3G8PyFNtwM/edit#heading=h.n6o18bl2wey7) 3

3. Related Studies 4

[4. Projects](https://docs.google.com/document/d/1dHZ5ffOKQlTGUIi3EbvEex-oVYzGT05Et3G8PyFNtwM/edit#heading=h.milc18ay7b0c) 6

[5. Metrics](https://docs.google.com/document/d/1dHZ5ffOKQlTGUIi3EbvEex-oVYzGT05Et3G8PyFNtwM/edit#heading=h.w4wrqzlkgxxn) 8

[6. Resource planning](https://docs.google.com/document/d/1dHZ5ffOKQlTGUIi3EbvEex-oVYzGT05Et3G8PyFNtwM/edit#heading=h.ngzwoae67mi3) 10

[7. References11](https://docs.google.com/document/d/1dHZ5ffOKQlTGUIi3EbvEex-oVYzGT05Et3G8PyFNtwM/edit#heading=h.d91j733reqxp)

**1. Team Formation**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Student Name** | **Student ID** | **Data Collection** | **Source Code Development** | **Statistical Analysis** | **Report Writing** | **Presentation Preparation** | **Final Review/ Verification** |
| Kiranmayie Bethi | 40092284 | \* | \* | \* |  | \* | \* |
| Koshaben Patel | 40094385 | \* | \* | \* |  | \* | \* |
| Sanjana Udar | 40094029 | \* | \* | \* |  | \* | \* |
| Elham Zamani | 40059941 | \* |  | \* | \* | \* | \* |
| Marjan Vahed | 40068762 | \* |  | \* | \* | \* | \* |
| Vida Abdollahi | 40039052 | \* | \* | \* | \* |  | \* |

**2. Type of the study**

One of the standard practices in the software industry is code commenting and the reason behind writing code comments by developers is to explain code, document specifications, communicate with other developers, ease future maintenance operations and etc. Since comments are key elements to understand and maintain the programs due to containing a rich amount of information, the importance of them should not be ignored. However, alot of developers do not pay enough attention to this issue and take writing comments as the least priority in the development process. In this proposal we focus on comments described in java programs and conduct an analysis about correlation between the amount and quality of comments and their effects to fault-proneness, understandability and readability of the program. Furthermore, we will be studying at least five open source software (based on type 1) in order to prove our hypothesis as follows:

Bug proneness, maintainability, usability and readability of the system (user comments, we are not considering auto-generated comments) based on:

1. Comment Density
2. Comment Quality

**3. Related Studies**

1- Aman.H et.al. in “Lines  of comments as a noteworthy metric for analyzing fault-proneness in methods” Focused on the comments to predict fault-prone programs. They believed comments can enhance the understandability of a program but also they can be used to mask the lack of readability in a program. They use LCM metric (lines of comments) for analyzing fault-proneness in java methods and they found out LCM can be a useful factor in fault-prone method prediction model and also more commented methods are almost two times more likely to be faulty than the others.[3]

2- Hiroshia Aman, in “An empirical analysis of fault-proneness of well-commented modules” focused on the comments of lines written in modules and mentioned the possibility of being faulty in well commented modules is almost two to eight times more than the modules with no comments at all. More precisely, understanding and comprehending the modules that are well-commented is hard when the comments are not well-written causing complexity and potential faults. Using the LCM metric and considering a regression model-based method, it was figured out that well-commented modules are most likely to be fault-prone and further investigations are needed to find the clear reason for that [5].

3- In “Code Comment Quality Analysis and Improvement Recommendation-An Automated Approach” Xiaobing Sun, Qiang Geng, David Lo, Yucong Duan, Xiangyue Liu and Bin Li proposed an approach to assess the feature of comment code quality and provide recommendations in order to improve their quality. The comments relevant to header and member method of two source programs including jdk8.0 and jEdit were examined. The metrics used for header comments were consist of existence of authorship information and the correlation between the header comments and the class and for the member method comments they studied three metrics of the correlation between the method and its comments, the number of lines of code in a method body, and the number of method invocations in a method body. [14]

4- Tan et.al. in the article “iComment: bugs or bad comment” talked about the relation between comments and source codes, they believed comments and source codes provide independent information about a program’s semantic behavior. By evolving the software, bad comments and bugs grow as well. Bad comments are actually the inconsistent ones with correct code. Also bugs happen when source code doesn’t follow requirements specified by correct comments of program. Comments are usually written by natural language and they can’t be analyzed automatically for detecting inconsistency between them and source code. In this article, the writers introduced their solution called “icomment” which combines natural language processing (NLP), machine learning, statistics and program analysis techniques. They evaluated icomment on four large code bases and they extract comments with 90.8-100 percent accuracy.[15]

5- Riehle and Arafat in the “The Comment Density of Open Source Software Code” used a quality indicator called density of comments (Comment density is the percentage of comment lines in a given source code base) in open source software codes to realize how open source software creates high quality and how they can maintain it. They found that comment density is independent of team and project size but it is related to project age. They used the database of open source analytics firm Ohloh, Inc. In overall, they claimed that commenting source code is a consistent practice of active open source projects and the average comment density is independent from team size and project size but it declines with an aging project.[16]

6- [Daniela Steidl, Benjamin Hummel and Elmar Juergens](https://www.sciencedirect.com/science/article/pii/B9780124115194000173#!) in “Quality Analysis of Source Code Comments” suggested that most of comment analysis are quantitative while they lack the qualitative aspect of the comments. They provided an analysis of comment quality through a model based on a variety of comment categories. For the purpose of classification implementation, a machine learning technique in Java and C++ programming was presented and the two metrics were provided to reveal defects of high quality comments.[18]

**4. Projects**

We consider 6 case studies mentioned as following with open-source software projects in Java programming language to investigate the correlation between comments and other metrics (as we mentioned in section 5 of this proposal) and the probability of being fault-proneness.

These open source systems could be categorized into two major areas in terms of size and complexity. Analyzing large scale software will provide us with accessing to a large amount of lines of comment, as they have usually huge amount of contributors and there will be more data to be extracted. These softwares are carefully chosen after reviewing their source code using multiple tools. These softwares also have huge amount of starts and forks given by other Github users.

The second groups of these softwares are slightly different from the first group in terms of size and being well-documented, yet they still provide us with sufficient number of classes and methods which are useful for analyzing our hypothesis based on CK and QMOOD metrics. All of these systems are also among the most popular java projects on GitHub.

(Please apply Scitools and Finbugs to the following projects) and also http://metrics.sourceforge.net/

<https://github.com/jrtom/jung> **[Sanjana]**

<https://github.com/ReactiveX/RxJava> **[Marjan]**

<https://github.com/apache/wicket> **[Kiran]**

<https://github.com/apache/lucene-solr> **[Kay]**

<https://github.com/eclipse/paho.mqtt.java> **[Elham]**

(@Kay please verify that the following repos have javadoc)

for 1,2,3,4,5 project has no javadoc. Though, kiranmaiye says we can generate javadoc for each project. I tried to generate javadoc for projects, but I was unable to do. I suggest we talk about this.

1. <https://github.com/elastic/elasticsearch> **ElasticSearch**

* Version 7.0.0, release  February 14, 2019
* Last commit: Feb 21, 2019
* Number of Contributors: 1157

2. <https://github.com/SeleniumHQ/selenium> **Selenium -Elham**

* Version 3.1.14, release  Jan 31, 2019
* Last commit: Jan 31, 2019
* Number of Contributors: 420

3. <https://github.com/geotools/geotools> **GeoTools**

* Version 18.4, release June 18,2018
* Last commit: Feb 19, 2019
* Number of Contributors: 177

4. <https://github.com/spring-projects/spring-boot> **Spring-boot**

* Version 2.1.3, release February 16,2019
* Last commit: Feb 20, 2019
* Number of Contributors: 540

5. <https://github.com/apache/pulsar>  **Pulsar**

* Version 2.3.0, release February 21,2019
* Last commit: Feb 21, 2019
* Number of Contributors: 97

6. <https://github.com/vaadin/framework> **Vaadin - kay**

* Version 8.7.0, release February 1,2019
* Last commit: Feb 21, 2019
* Number of Contributors: 174

**Additional Tools**

As we are focusing on applications with java source code, so the tools for code analysis to examine the code and look for potential problems such as possible bugs, dead code, suboptimal code, over complicated expressions, and duplicated code are listed in the following with a brief description:

**FindBugs Plugin (Eclipse or IntelliJ):** Its purpose is looking for bugs in Java programs and detecting a variety of common coding mistakes, including thread synchronization problems, misuse of API methods, etc.

**Code Analyzer (Multi-Platform Java Code Analyzer):**  is a software source file metrics application.

**Java Metrics:** computes a number of useful metrics for Java program source code suites.

**Understand:** A multi-platform tool for code analysis and comprehension of large code bases.

**Eclipse/IntelliJ:** A leading Java IDE with built-in inspection and analysis.

**5. Metrics**

The use of software metrics for tracking and analyzing is to determine the quality of the current product or process, improve that quality and predict the quality once the software development project is complete and they areimportant components of quality assurance, management, debugging, performance, and estimating costs. Here is the list of all the possible metrics for the project we are planning to work on that may be narrow down into fewer ones during our estimations in the project.

* **C&K Metrics:**

- Weighted methods per class (WMC): It is the sum of all complexity of the class’s methods which it is being calculated [1].

- Depth of inheritance tree of a class (DIT): It measures the inheritance level from the top of object hierarchy for each class. [1]

- Number of Children (NOC): It measures the total number of immediate descendants or subclasses of a class. [1, 2]

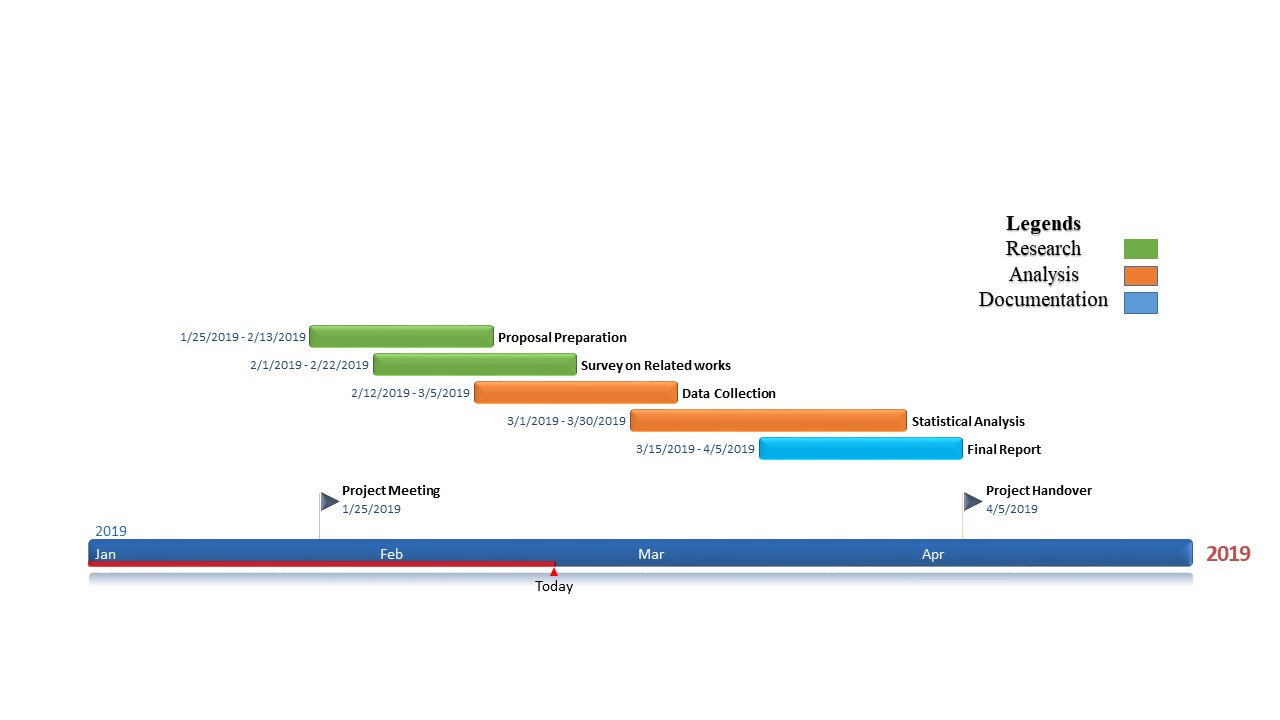
- Lack of Cohesion in Methods (LCOM): It represents the independence between member functions. [2]

* **Additional Metrics:**

-Comment Percentage (CP): The CP is defined as a ratio of the number of comment lines to the number of non-blank LOC. Software development life cycle is normally long. In any stage of the life cycle, comments will help developers and maintainers to better understand the programs. Higher comment percentages will increase understandability and maintainability. It is suggested to maintain at least 8% on comment percentage to enhance the understandability. [12]

-LCM (Line of Comments): It is the number of lines of comments in a method body, except for commented-out code. [3]

**6. Resource planning**



**7. References**

[1] B. Suri, SH.Singhal, “Investigating the OO Characteristics of Software using CKJM Metrics“[2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)](https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7321997),Sep 2015.

[2] T.Honglei, S.Wei and Zh.Yanan, “The research on software metrics and software complexity metrics“, 2009 International forum on computer science-technology and application. 2009

[3] H. Aman, S. Amasaki, T. Sasaki and M.Kawahara, “Lines of comments as a noteworthy metric for analyzing fault-proneness in methods”, IEICE Trans.INF & SYST., Vol. E98-D, No. 12, Dec 2015.

[4] Beat Fluri, Michael Wursch and Harald C.Gall, “on the relation between source code and comment changes”, November 2007, IEEE Conference

[5] Hirohisa Aman, “An Empirical Analysis on Fault-Proneness of Well-Commented Modules”, 2012

[6] Ö. Özgür Tanrıöver, and Recep Eryigit, “An empirical analysis of early object-oriented design metrics in relation to code size”, Proceeding of 6th IEEE International Conference on Software Engineering and Service Science (ICSESS), pp. 62-65, 2015

[7] Jubair J. Al-Ja’afer, and Khair Eddin M. Sabri, “Metrics for Object-Oriented Design (MOOD) to Assess Java Programs”, International Arab Conference on Information Technology, 2004

[8] Puneet Kumar Goyal, and Gamini Joshi, “QMOOD metric sets to assess quality of Java Program”, 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), pp. 520-533, 2014

[9] Monika, Preeti Sharma, “Prediction of fault-proneness using CK metrics”, International journal of computer science and information technology research, vol. 4, September 2016

[10] B.Katoch, L.K.Shah, “ A systematic Analysis on MOOD and QMOOD metrics“, International journal of current engineering and technology, Vol.4, No.2, April 2014

[11] J. Bansiya, C.G.Davis, “A hierarchical model for object-oriented design quality assessment”, IEEE transactions on software engineering, Vol. 28, No. 1. January 2002.

[12] S.Chawla, G.Kaur, “Comparative study of the software metrics for the complexity and maintainability of software development”, IJACSA International journal of advanced computer science and applications, Vol.4, No. 9, 2013

[13] Luca Pascarella and Alberta Bacchelli, “Classifying code comments in Java open-source software systems”, ACM 14th International Conference on Mining Software Repositories (MSR), 2017

[14] Xiaobing Sun, Qiang Geng, David Lo, Yucong Duan, Xiangyue Liu and Bin Li, “Code Comment Quality Analysis and Improvement Recommendation-An Automated Approach”, International Journal of Software Engineering and Knowledge Engineering, Sep 2015

[15] Lin Tan, Ding Yuan, Gopal Krishna, and Yuanyuan Zhou. “Icomment: bugs or bad comments”, Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007.

[16] Dirk Riehle, Oliver Arafat, “The Comment Density of Open Source Software Code”, Conference Paper *in* Proceedings - International Conference on Software Engineering · May 2009.

[17] Y.Miyake, S.Amasaki, H.Aman and T.Yokogawa, “A replicated study on relationship between code quality and method comments”, Applied Computing and Information Technology, pp 17-30,Volume 695, March 2017.

[18] [Daniela Steidl, Benjamin Hummel and Elmar Juergens,](https://www.sciencedirect.com/science/article/pii/B9780124115194000173#!) “Quality Analysis of Source Code Comments”, [International Conference on Program Comprehension (ICPC)](https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6597032), 2013