1. Related Work

For each related work you identified in Milestone 1, write a paragraph with the following content:

1. Summary

a. Research goal (What is the problem being solved/investigated?)

b. Motivation (Why is the problem important?)

c. Approach/Methodology (How did they solve/investigate the problem?)

d. Empirical Findings/Experimental Results (What is the outcome of the study?)

e. Conclusions (What did we learn from the study?)

2. Comparison with the study you proposed.

a. What are the similarities with your study?

b. What are the differences with your study?

c. What are the limitations of this related work?

d. What are the improvements you are planning to do in your study?

**1- “Lines of comments as a noteworthy metric for analyzing fault-proneness in methods”**

**1) Summary:**

**a.Research goal (What is the problem being solved/investigated?)**

Hirohisa Aman 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.

**b.Motivation (Why is the problem important?)**

They said for performing quality management in a code, size metrics and complexity metrics are usually used because normally larger programs have less quality, but we should never forget to assess human factors as well like the comments written by a programmer. 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, as comments can’t effect a software’s functionality or performance but they can help to assess the program’s quality.

**c.Approach/Methodology (How did they solve/investigate the problem?)**

Their approach was to consider a method in a class to be a module and they focused on comments written in a software module. They use LCM metric (lines of comments) for analyzing fault-proneness in java methods. LCM is the number of lines of comments in a method body, except for commented-out code.

**d.Empirical Findings/Experimental Results (What is the outcome of the study?)**

Their hypothesis were: can LCM discriminate fault-prone methods from others? Can LCM enhance the fitness of logistic regression model for detecting faulty methods? They worked on 4 open source software: Eclipse checkstyle plugin, Hibernate ORM, PMD, Squirrel SQL client. Reasons to choose: 1-They were in Java, the source files are maintained with GIT and they were popular OSS. In previous works, the researchers didn’t consider LCM metric alongside with LOC (Lines of Code) and CC (Cyclomatic Complexity) which are well-known metrics for predicting fault-prone programs. They worked on two different cases, one with no data of faults and then with data of faults and used the logistic regression model

**e.Conclusions (What did we learn from the study?)**

They found out LCM can be a useful factor in fault-prone method prediction model and also they concluded that more commented methods are almost two times more likely to be faulty than the others.

**2) Comparison:**

**a.What are the similarities with your study?**

In this project, we try to find the correlation between comments and fault-proneness in Java programs, we will also consider the number of comments and their relation to fault-proneness and also we will consider some level of quality of comments in our studies and this article gave us a really clear idea.

**b. What are the differences with your study?**

We will work on number of comments and also quality of comments and the metrics we will use will be more than this related work.

**c.What are the limitations of this related work?**

In this paper, only comments written inside method’s body had been considered and all of the comments outside a method’s body had been ignored because they are usually a description of how to use the program. This paper gave a very clear idea about this relation. Usually more comments show some kind of technical debt in a code but it is important to consider the difference between types of comments.

They only worked on the number of comments and not on the content of them, also how to count the number of lines of comments are important because a programmer can write a comment in one line while the other programmer might write it in a couple of lines. So LCM and also LOC might vary for different programmers for same description. Preprocessing like normalization of program’s style might be needed for these concerns.

In this paper, the fault collection was based on keyword matching in commitment logs so there was a possibility to miss some of true faulty methods. In this kind of work, most of the studies used a keyword matching approach but there are suggestions of using bug tracking system.

This paper didn’t count file renaming event and only focused on events related to file creation and changes(modifications). For accuracy, we can use Git’s function for tracing file renaming events or using Historage for Java.

**d.What are the improvements you are planning to do in your study?**

We plan to work on more open source programs and consider quality in our job as well, we also calculate more metrics.

**2- “An Empirical Analysis on Fault-proneness of Well-Commented Modules”**

**1. Summary**

**a. Research goal (What is the problem being solved/investigated?)**

Hirohisa Aman in “An Empirical Analysis on Fault-proneness of Well-Commented Modules” studies whether Comment statements are always useful to enhance the readability and/or understandability of software modules.

**b. Motivation (Why is the problem important?)**

Writing readable program modules is the most basic quality-assurance activity in software development. To enhance the readability of source code, comments are widely known as effective entities , and programmers are encouraged to write helpful comments into their program modules. But too many comments may indicate code fragments that are complex and hard to understand, i.e., bad code.

**c. Approach/Methodology (How did they solve/investigate the problem?)**

First, this paper introduced the metric, lines of comments (LCM), to quantify the amount of comments. Then, the paper proposed a regression model-based method to automatically pick out “well-commented” modules by considering the distribution of LCM over the lines of code (LOC). After that, an empirical analysis with three major open source software was performed.

**d. Empirical Findings/Experimental Results (What is the outcome of the study?)**

The results showed that the well commented modules have about 2 to 8 times greater risks of being faulty than non-commented ones. Hence, well commented modules are more likely to be fault-prone , and they should be reviewed preferentially.

**e. Conclusions (What did we learn from the study?)**

This paper does not claim that the comments reduce the code quality. In other words, they do not deny the existence of comments, and do not hope developers to omit to write comments, since comments do not cause any faults. Comments are harmless. However, well-written comments can be signs of potential faults because problematic code may need more explanatory comments: if a developer faces a code fragment to which he/she wants to write some elaborate comments, the developer should be more careful with the code fragment and ask himself/herself, “Is there any room for improvement in the code?”.

**2. Comparison with the study you proposed.**

**a. What are the similarities with your study?**

As we try to study the bug proneness, maintainability, usability and readability of the system with respect to comment density and comment quality, we see that this paper gives us a good perspective regarding well commented code. That though it does increase the readability of the program it may indicate bad and faulty code.

**b. What are the differences with your study?**

Here they have focused on comments that are elaborate while we look forward to work on various categories of comments to gain a much wider perspective on which type comments are affecting the bug proneness, maintainability, usability and readability of the system and how.

**c. What are the limitations of this related work?**

In order to make a criterion for deciding well-commented modules, they proposed to use the regression line on the scatter diagram between LCM and LOC. Since their proposal is made under the assumption that there is a certain level of correlation (may be weak) between LCM and LOC from their experience, their discriminant method will be poor if they have a strong correlation. In such case, categorization will be worthless.

This work used only the comments written inside the method bodies.

The studied software are only three open source software. Their results may not be generalized to other open source software and/or commercial software.

**d. What are the improvements you are planning to do in your study?**

We intend to work on more open source programs and categorize the comments covering more areas of comments.

**3- “The Comment Density of Open Source Software Code”**

**1)Summary:**

**a.Research goal (What is the problem being solved/investigated?)**

Riehle and Arafat in “The Comment Density of Open Source Software Code “described the differences between closed source and open source projects and the high quality of open source codes. They worked on a quality indicator called the density of comments in open source software code. Their research goal was to improve the understanding of open source software development processes and to transfer appropriate practices into corporate software development. This has become particularly important, because the traditional life-cycle model or the more recent agile methods either don’t scale to large project sizes or have problems in coping with changing requirements.

**b.Motivation (Why is the problem important?)**

They assumed comment density is a good predictor of maintainability and hence survival of a software project and they wanted to prove it.

**c.Approach/Methodology (How did they solve/investigate the problem?)**

They used the database of the open source analytics firm Ohloh, Inc. and worked with a database snapshot of March 2008. The database contains detailed data from about 10,000 open source projects which all were active. They defined an active project filter to let a project pass only if by the end of 2007 which finally cut the related projects to 5000.

They had metrics such as SLoC(source line of code), which is a physical line in a source file that contains source code. CL (comment line) which is a physical line in a source file that represents a comment, LOC (Line of Code) which is either a source line of code or a comment line.

The commit size of a commit is the number of lines of code affected by a commit, whether added, removed, or changed and finally the comment density of a file or a group of files or the whole source code base of a project is defined as the number of comment lines divided by the number of lines of code of the same code body.

**d.Empirical Findings/Experimental Results (What is the outcome of the study?)**

They tested 5000 projects (almost 30% of all active open source projects in 2009). They used a tool chain that consists of the original database in a PostgreSQL RDBMS instance, intermediate processing using SQL queries and Java code, and their final processing was by R project and Excel.

**e. Conclusions (What did we learn from the study?)**

They concluded the comment density is independent of team and project size but it is related to project age and declines with an aging project. This decline is statistically significant but it is small and it practically limited implications.

**2) Comparison:**

**a.What are the similarities with your study?**

In this project, we will also consider comment density on some of open source projects. We try to consider large active projects and we consider quantity and quality of comments and their relation to fault-proneness of a program.

**b.What are the differences with your study?**

In our work, we will also consider comment quality as well and we try to find the correlation of comments and fault-proneness.

**c.What are the limitations of this related work?**

The biggest limitation of this was considering the equality size of all comment lines, whether they provide rich content or even were auto-generated stubs. (They used diff tool/parser to distinguish programming languages and recognize multi-line comments).

They also analyzed only active projects and have yet to determine to what extent a high comment density can be used as a predictor of project success or failure.

**d.What are the improvements you are planning to do in your study?**

We plan to consider comments quality and quantity and find their relation to fault-proneness of open source projects in java and see how comments are related and impact the program.

**4-** “**Code Comment Quality Analysis and Improvement-Recommendation: An Automated Approach”**

**1. Summary**

**a. Research goal(What is the problem being solved/investigated?)**

Xiaobing Sun and five other researchers in Code Comment Quality Analysis and Improvement- Recommendation: An Automated Approach, proposed an approach to evaluate code comment quality and provided some suggestions how to improve the quality of comments.

**b. Motivation (Why is the problem important?)**

It is mentioned due to the time pressure and poor coding style during development process, many developers ignore writing high quality comments leading to having difficulties of comprehending the code and understanding the functionality of methods and classes by developers while doing software maintenance in the future. So, this paper was proposed to deal with this problem.

**c.Approach/Methodology (How did they solve/investigate the problem?)**

To investigate if their automatic approach was effective, they did a user study to assess the quality of comments manually. They analyzed not only method comments but also header/class comments of both program elements with comments and those without any comments. Eventually, the quality of comments is analyzed based on the quality of header and quality of methods existing in a file. The SCF metric was used for quality assessment of all comments in a file in two java source programs including jdk8.0 and jEdit. Then existence-authorship and correlation-header class metrics were defined to evaluate a header comment authorship information and the relevance of the class name to its header comment respectively. Also, metric of correlation-method is defined to examine the relevance of the method to its comments.

**d. Empirical Findings/Experimental Results (What is the outcome of the study?)**

The results for existence of authorship metric shows the authorship information in class comments are important and necessary and their case study was effective to prove this matter. It was found out header comments contain useful information about the class, and their metric was helpful to evaluate their correlation. For comments with correlation-method, their metric was correct and provided relevant and fruitful information. Finally, it was concluded that results of their approach in comparison with the manual assessment by case study were reasonable and some of their recommendations to improve the quality of code comments were not as good as what they had thought.

**e. Conclusions (What did we learn from the study?)**

They only worked on quality analysis of comments that provided reasonable results of comment quality analysis. Also, the lexicon aspect of comments was considered not the semantics of them and the accuracy of former comment quality analysis can be improved through their approach.

**2. Comparison with the study you proposed.**

**a. What are the similarities with your study?**

**b. What are the differences with your study?**

**c. What are thelimitations of this related work?**

Since they used a case study by some participants, there was a variety of participators with various knowledge levels in programming ranging from university students to experts in industry. Hence, they were supposed to provide a structure, example and sample of their work and their expectations regarding doing the study and analyzing manually by these people to make sure their automatic approach was more effective or the manual analysis. Furthermore, the quality analysis was taken into account based on lexicon of the comments not the semantics of them.

**d. What are the improvements you are planning to do in your study?**

**5- How Good is your Comment?**

**A study of Comments in Java Programs**

**1. Summary**

**a. Research goal (What is the problem being  
solved/investigated?)**

Dorsaf Haouari, Houari Sahraoui, Philippe Langlais in How Good is your Comment?-A study of Comments in Java Programs presented that comments have been under investigation in several studies but the lack of analyzing commenting habits can be observed literally. So, a study of open source java projects by studying comments from the perspectives of quality and quantity through classifying the comments to conduct their analysis.

**b. Motivation (Why is the problem important?)**

It is believed very few comment contents are considered and targeted to understand and comprehend the program while they provide helpful pieces of information to take advantage of.

**c. Approach/Methodology (How did they solve/investigate the problem?)**

By analyzing some sample of comment contents randomly via defining a taxonomy, they figured out the most common trends in commenting by focusing on the most documented program construct types. Besides, it is found out comments does not always contain useful contents and they are not deliberately written for program understandability or documentation. NCOM-LOC metric was used to compute the ratio of comments per line of code. A parser generated from sbleCC was used to determine how comments were distributed and how they were used frequently in their java programs as an automatic analysis.

**d. Empirical Findings/Experimental Results (What is the  
outcome of the study?)**

The most prone comment constructs and the most extensively used category of comments named working comments by developers were represented through their work. The vast majority of comments belonged to the methods and the large portion of frequent constructs dedicated to the variable declarations and method invocations.

**e. Conclusions (What did we learn from the study?)**

From the quantitative point of view, the distribution of comments over the program constructs and the frequency of construct commenting were examined. Qualitative aspect concerned the comments objects, type, style and quality. It was concluded that some constructs like methods are highly prone to be commented regularly with purpose of explanatory reasons. And the most percentage of comment rate is belonged to communication between programmers or the ones that are about to be changed in the future.

**2. Comparison with the study you proposed.**

**a. What are the similarities with your study?**

**b. What are the differences with your study?**

**c. What are the limitations of this related work?**

There is a need for other replications with comment samples together with larger subjects are needed in regarding confirming and completing the assumption of comment location and their content in the code.

**d. What are the improvements you are planning to do in your study?**

**6- An Empirical Analysis of the Impact of Comment Statements on Fault-Proneness of Small-Size Module**

**1. Summary**

**a. Research goal (What is the problem being solved/investigated?)**

Hirohisa Aman in “An Empirical Analysis of the Impact of Comment Statements on Fault-Proneness of Small-Size Module” studies Small-size modules and works on the assumption that they should use metrics other than code size since all modules are small size. This paper focuses on “comments” written in the source code from a novel perspective of size-independent metrics; comments have not been drawn much attention in the field of faultprone module prediction. Thus a new method for small size modules is intended to be found.

**b. Motivation (Why is the problem important?)**

Code size metrics are commonly useful in predicting fault-prone modules, and the larger module tends to be more faulty. In other words, small-size modules are considered to have lower risks of fault. However, since the majority of modules in a software are often small-size, many “small but faulty” modules have been found in the real world. Hence, this paper intends to find another fault-prone module prediction method, intended for small-size module.

**c. Approach/Methodology (How did they solve/investigate the problem?)**

The empirical study collects 11*,* 512 small-size modules, whose LOC are less than the median, from three major open source software, and analyzes the relationship between the lines of comments and the fault-proneness in the set of small-size modules.

**d. Empirical Findings/Experimental Results (What is the outcome of the study?)**

The empirical results show the followings: 1) A module in which some comments are written is more likely to be faulty than non-commented ones; the fault rate of commented modules is about 1*.*8 – 3*.*5 times higher than that of non commented ones. 2) Writing one to four lines of comments would be thresholds of the above tendency.

**e. Conclusions (What did we learn from the study?)**

It is generally preferable that a small-size module is understandable even if it has no comment because of the smallness. We learn that a small-size module with some comments may contain some problematic code. Not that comments are bad entities but, while comments are good entities to enhance the code readability, they may cover up even problematic code’s complexity.

**2. Comparison with the study you proposed.**

**a. What are the similarities with your study?**

This article is closely related to the comment density aspect of our work. We want to study the bug proneness, maintainability, usability and readability of the system with respect to comment density and comment quality

**b. What are the differences with your study?**

This work does not cover the quality of the comments or which type of comments generate what faults which we intend to include in our work.

**c. What are the limitations of this related work?**

They used the data about whether a fault was found in the module or not, so they did not take into account the severity of faults and the number of faults in the modules. While the primary contribution of this paper is to introduce yet another perspective of fault-prone module prediction, such roughness of data may decrease usefulness of the empirical results. That is to say, the results might not be useful in finding problematic modules who have many critical Faults.

Moreover, they could not associate comments with faults, directly. LCM may correlate with LOC in many cases, i.e., LOC may be a confounding factor.

The data did not consider the timing of comments—when the comments are written. Some age-old comments might be no longer in use and have no impact on the code quality. Such tracking comments and their analysis would also be required.

**d. What are the improvements you are planning to do in your study?**

We intend to work on more open source programs and categorize the comments and consider quality of these comments in our job as well.

**7- Quality Analysis of Source Code Comments**

**1) Summary:**

**a.Research goal (What is the problem being solved/investigated?)**

Steidl et,al. in “Quality analysis of source code comments” presented a detailed approach for quality analysis of code comments. They worked and categorized different comments and used machine learning on Java and C++ programs.

**b.Motivation (Why is the problem important?)**

As a major part of source codes are comments, the authors decided to understand the quality of comments in a program because most of previous works ignored quality of comments and just worked on its quantity.

Even some evaluate the comment ratio of a system to measure documentation quality which it is not enough because copyrights and commented out codes should be ignored. In overall at that time a complete model of comment quality did not exist because comment analysis was a difficult task because of their nature.

**c.Approach/Methodology (How did they solve/investigate the problem?)**

They provide a semi-automatic approach for quantitative and qualitative evaluation of comment quality. They performed comment categorization for Java and C/C++ programs based on machine learning to differentiate between different comment types. By comment categorizing they detailed quantitative analysis of a system’s comment ratio and they considered qualitative analysis to suit each single category. For assessing quality attributes they used metrics detecting quality defects in comments of specific categories and evaluated the metrics’ validity and relevance separately. They evaluated validity by a survey among experienced software developers and evaluated relevance by a case study of five open source projects.

In this article, they differentiate between seven different types of comments:

\_ Copyright comments include information about the copyright or the license of the source code file. They are usually found at the beginning of each file.

\_ Header comments give an overview about the functionality of the class and provide information about.

\_ Member comments describe the functionality of a method/field, being located either before or in the same line as the member definition. They provide information for the developer and for a project’s API.

\_ Inline comments describe implementation decisions within a method body.

\_Section comments address several methods/fields together belonging to the same functional aspect.

\_ Code comments contain commented out code which is source code ignored by the compiler. Often code is temporarily commented out for debugging purposes or for potential later reuse.

\_ Task comments are a developer note containing a remaining to do, a note a about a bug that needs to be fixed, or a remark about an implementation hack.

**d.Empirical Findings/Experimental Results (What is the outcome of the study?)**

In this paper, the authors created two separate training sets (for Java and C++), then they used a set of features for machine learning. They used a J48 decision tree algorithm and for evaluation they used the standard five-fold cross validation method. The decision tree in most categories achieved a precision and recall above of 90 percent.

For quality model, their model resembles quality models in maintenance. It is based on entities (entities describe the concept whose quality is under evaluation like the comment categories), activities (Activities represent the developers’ intentions to comment code, for example to better understand implementation details or to know bugs/hacks), and criteria (Criteria represent quality aspects for different entities and show the impact of an entity on a specific activity).

Two metrics they used were: coherence between code and comment(called c\_coeff) and length of comments(metric counts the number of words in a comment).

**e.Conclusions (What did we learn from the study?)**

They used a machine-learning approach for comment classification to model comment quality which detailed quality attributes in terms of coherence, consistency, completeness and usefulness. Their case study showed comment classification provides better quantitative insights about the system documentation as the simple comment ratio metric. For quality analysis they used metrics reveal quality defects in code commenting; the length indicator which suggests to extract methods when inline comments with at most two words are used and the c\_coeff metric detects both comments with insufficient information and methods with a low-quality method identifier.

**2) Comparison:**

**a.What are the similarities with your study?**

In this project, we also work on the quantity and quality of comments in some of open source java projects.

**b. What are the differences with your study?**

We consider some different metrics and also works on the correlation between comments and fault-proneness. Also in this article, they considered both Java and C++ programs which we only will work on Java programs.

**c.What are the limitations of this related work?**

In this article, the authors worked on some aspects of automation of comment quality and they didn’t work on bugs related to comments.

**d.What are the improvements you are planning to do in your study?**

We plan to find the relation between number of comments and their quality and fault-proneness in some programs, so our work will have this attributes of quality and quantity together.