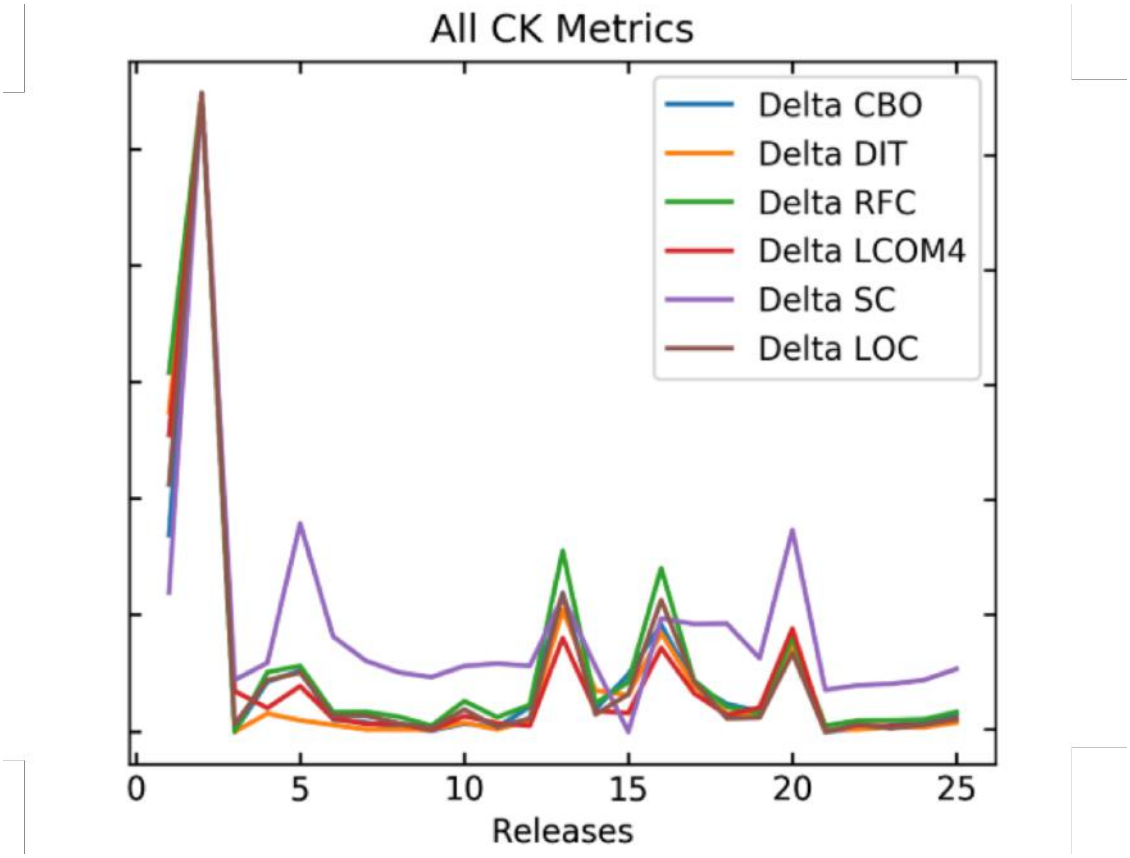
CHALMERS
UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG



The Relation Between Documentation and Internal Quality of Software

HUMBERTO LINERO

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

The Relation Between Documentation and Internal Quality of Software

HUMBERTO LINERO

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

The Relation Between Documentation and Internal Quality of Software
HUMBERTO LINERO

Supervisors: Michel R.V. Chaudron, Truong Ho Quang
Examiner: Robert Feldt

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

The Relation Between Documentation and Internal Quality of Software
HUMBERTO LINERO
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Regardless of the software development process used, there are many factors that take place during such process. Those factors may affect positively or negative the internal quality of the resulting software product. In the context of open source software, the success of a project depends on having a community in which people can contribute to expand and improve the project. How modular and easy-to-modify a system is, is one of many factors that developers take into consideration before contributing to a project. Therefore, creators of open source projects need to ensure that their system has certain level of internal quality and modularity in order to make it easier for contributors around the world to modify and extends the system. Therefore, understanding the factors that affect the quality of software product is the first step towards developing software of higher internal quality.

This study investigates the effect that factors such as documentation quality, size of a system, and documentation size have on a specific internal quality metric: *Coupling Between Objects* or *CBO*. Open Source Software projects were selected for this study. Their internal quality over time was studied as well as the quality and content of their documentation. Finally, a Spearman's correlation analysis revealed the correlation between documentation quality, size of a system, and documentation size with the metric *CBO*.

Our results suggest a strong positive correlation between the CBO metric and factors such as lines of code, number of modules, and documentation size. Such results indicate that as the size of the system increases (size expressed as lines of code or number of modules), the CBO of the system increases as well. The same is true for amount of documentation files. As it increases, so does the CBO of the system. This study further explains such results and discusses the possible causes of the correlations.

Keywords: Unified Modeling Language, internal software quality, software documentation, Chidamber & Kemerer metrics, object-oriented metrics, Spearman's correlation

Acknowledgements

My profound gratitude to my supervisors Michel R.V. Chaudron and Truong Ho Quang for providing me with unfailing support, guidance, and feedback through the process of researching and writing this thesis. Additional thanks to professors Richard Torkar and Lucas Gren for guidance and feedback in the analysis of the data.

Humberto Linero
Gothenburg, June 2018

Contents

1
Introduction

The advances in computational technology have enabled the creation of highly complex software systems. These software systems may contain thousands, even millions, of lines of code. For example, the Windows Vista operating system has approximately 50 million lines of code; the Mac OSX Tiger operating system has nearly 85 million lines of code, and the software system of a modern car has approximately 100 million lines of code[1].

Maintaining and evolving software system as complex and large as the ones mentioned above is not an easy task. Therefore, it is imperative for software architects to design systems that have an acceptable level of internal quality so that their testing and maintenance do not become a costly and time-consuming task.

As part of the process for designing the architecture of a system, designers typically use modeling languages and modeling tools to represent and communicate better the details of the architectural design. The Unified Modeling Language (UML) is an example of a general-purpose modeling language that software architects use for visualizing the architectural design of a system [28]. These models not only serve as a way of communicating the structure of the system but also as a way of documenting it. Although UML diagrams are widely used in the industry, it is not the only form of documentation developers use for understanding and maintaining a software system.

This section will provide the reader with general, background information about the concepts of internal quality and software documentation; specifically, UML diagrams.

1.1 Software Documentation

Artifacts that are considered to be documentation varies from project to project. For example, software documentation could be defined as an artifact whose purpose is to communicate information about a system to the project stakeholders. Those stakeholders may include managers, project leaders, developers, or customers. Some examples of documentation include source code, inline comments, or specification documents. [10].

An Architectural model is an example of documentation used for communicating the architecture of a system and the relationship between software components. Such models are commonly created using a modeling language such as the Unified Modeling Language or UML. Due to the special attention this thesis gives to UML

[1] `https://informationisbeautiful.net/visualizations/million-lines-of-code/`

diagram, the next section provides an background information about such modeling tool.

1.1.1 Unified Modeling Language

Modeling is an activity architects use to create an abstraction of a system. In the context of software development, models allow architects to understand and communicate better the complexity of an architecture. This is accomplished by modeling the various artifacts that make up the system. The role of models has become more relevant as a result of the appearance of methodologies such as model-driven design and model-driven architecture. As a consequence, the Unified Modeling Language or UML has become a central tool in model-based engineering [28].

The Unified Modeling Language or UML is a general-purpose modeling language introduced in 1997, which has now become the de facto standard for modeling systems. Its usage has reached domains beyond software. Some of those domains include business and hardware design [28].

Various empirical studies have demonstrated the benefits of using UML diagrams in the software development process. Some of those benefits are associated with the reduction of efforts required during the maintenance phase [19].

1.2 Software Quality

It is necessary to define what software quality is in order to comprehend its importance and how it is measured. There are many philosophies that define quality. Crosby's, Deming's, and Ishikawa's are some examples. Although they have their own view, in general, quality can be defined in terms of [11]:

1. **Conformance to specification:** In this view, the degree of quality of a product depends on the extent that it conforms to the requirements specifications of the product.

2. **Satisfying customer needs:** In this view, the degree of quality of a product depends on the extent that it satisfies customer's needs.

As shown in the previous two views, quality is not a binary value, i.e., either the product has quality or not. Instead, it is a degree, i.e., a product can have a higher degree of quality than another product.

1.2.1 Measuring Quality

It is necessary to measure quality in order to improve it. With the purpose of understanding and measuring quality, researchers have created models illustrating how quality characteristics are related. The *McCall's Quality Model* is among the earliest models of this type. In his model, McCall defines software quality as a function

of various factors, criteria, and metrics. For example, in McCall's model the maintainability of a software depends on factors such as modularity, self-descriptiveness, simplicity, and conciseness. [5].

The ISO 9126 is another model used for defining quality. Such model indicates that the quality of the process influences the internal quality of the product. At the same time, the internal quality of the product influences the external quality of the product, which in turn influences quality in use. Researchers have developed various metrics to measure the internal quality of software. Cyclomatic complexity, fan-in, fan-out, total lines of code, and CK-Metrics are some examples of metrics used for measuring the internal quality of software. To gather these and more metrics, software engineers typically use static code analysis tools. Analizo, Source Monitor, Sonarqube are some tools used for calculating internal quality metrics of software.

1.3 Problem Statement

So far the reader has been introduced to the concept of software documentation, UML as a tool for modeling the architecture of a system, and the concept of software quality along with metrics for measuring it. But how is software documentation related to software quality? What effect does quality and size of documentation have on the internal quality of software? What other factors affect internal quality of software and by how much? This study will explore the effect that various factors have on the internal quality of software.

1.4 Thesis Structure

This section provided general information about the concepts of software documentation and internal quality. In section II, the reader will be presented with a literature review analysis discussing the current state of knowledge in the topics of internal software quality and software documentation. Section III presents the research questions, the methods, and steps taken to accomplish the purpose of this study. Section IV and V present the results and their implication; respectively. Finally, section VI presents the limitations and threats of the study.

2

Literature Review

2.1 Software Documentation

Just like source code and tests, documentation is an artifact that has a role in the software development process. Depending on the development method used (i.e., Agile, Waterfall, etc.), the importance of documentation may vary. However, in order to comprehend the role that documentation plays in software development it is important to understand how software engineers use such artifact.

Lethbridge et al. [13] conducted a study to more accurately comprehend and model the use of documentation, its usefulness, and maintenance. The results of the study confirm the widely held belief that documentation is not completely up-to-date nor updated in a timely manner. However, their results also suggest that out-of-date documentation could remain useful in certain circumstances. The same study also reveals some general attitudes software engineers have about documentation. Some of those include the following:

1. Inline comments are good enough for assisting the maintenance work.
2. Systems often have too much documentation and such documentation is often poorly written.
3. Creating documentation could be time-consuming tasks that outweigh the benefits.
4. Trying to find useful content in documentation may be a challenging task.
5. A considerable portion of the documentation is not trustworthy.

Forward et al.[10] did a similar study in which they not only study the perceived relevance of documentation but also the relevance of the tools and technologies for creating, maintaining, and verifying documentation. Their results indicate that software developers value technologies and tools that automate documentation maintenance. Their results also indicate that participants consider that test code contains a lot of useful data that should be automatically extracted to generate documentation. Their results also support the idea that software systems have a large amount of documentation, which is hardly organized, understandable, and maintainable. An important conclusion obtained from the research suggests that documentation is a tool for communication. Therefore, technologies that automatically generate documentation should be efficient at communicating ideas instead of providing rules for validating and verifying facts.

As suggested by the research studies previously mentioned, developers consider

that creating and maintaining documentation is a time-consuming task. This typically results in documentation being missing or out-of-date. As a result, in a study by DeSouza et al. [17] the authors tried to investigate how much documentation is enough and the types of documentation that are most useful during maintenance efforts. Their results indicate that source code, inline comments, data models, and requirements are considered to be the important type of documentation for maintaining a system; with source code and inline comments being the most important. Interestingly, architectural models are not considered to be very important. The authors argue that this could be the case because architectural documentation is used once for getting a global understanding of the system and not consulted afterward; however, this does not take away its importance. But what role does documentation play in the internal quality of software? Does the quality of documentation affect internal quality of software?

Although all types of documentation are considered in this study, special attention is given to UML diagrams. As a result, the following section will provide the current state of knowledge regarding the usage and role of UML diagrams in software development.

2.1.1 UML Diagrams

Before the introduction of methodologies such as model-driven engineering, model-driven design, and model-driven architecture, source code was considered the primary artifact in software development, while models were secondary artifacts used for supporting the communication and understanding of the source code. However, practitioners of a model-driven engineering approach consider models to be the primary artifact in the development process. The Unified Modeling Language or UML is the de facto tool used for creating models [26]. With its increased usage in the industry, it is no surprise that researchers in the field of software engineering have tried to better understand and explore how UML diagrams are used, its impact on the development process, and the expectations developers have as a result of its usage.

Tilley et al. [15] performed a qualitative study to asses the efficiency of UML diagrams as a documentation tool for aiding program understanding. The preliminary results suggest that UML diagrams can help engineers understand large system. The same study also indicated that the efficacy of the diagram is affected by factors such as syntax, semantics, and layout of the diagram, and by how much domain knowledge of the system the developer had. In the context of maintainability, an experiment perform by Arishholm et al. [19] focused on determining if UML models helped developers make changes quicker and better to existing systems. Their results indicated that UML diagrams does help developers making changes to code faster. However, the time saved is lost whenever modification of the diagram is required. Additionally, functional correctness of changes as well as quality of design is positively impacted whenever UML diagrams are available. Nevertheless, this only applies for tasks that were considered to be complex.

Since the cost and effort required to modify software systems increases as the project progresses, engineers are interested in applying techniques that allow them

to predict the quality of a system early in the development process. With that motivation, researchers have explored the usage of UML diagrams as a tool for predicting the quality of systems. One technique used for early assessment of quality requirements is to transform software models into a mathematical notation that is suitable for validation [8]. Other techniques have been explored as well. For example, Cortellessa et al.[8] analyzed UML diagrams and used Bayesian analysis for making predictions regarding reliability of the system. In their study, UML diagrams are annotated with attributes associated to reliability of component and connector failure rates. Those attributes are then used for making predictions of the reliability of the system.

Using UML diagrams assets as tool for predicting source code quality in early stages of development is important. However, determining the quality of the UML diagrams is also an important area of research. Genero et al.[12] proposed a set of metrics that served as class diagram maintainability indicators. Those metrics include: *understandability time*, *modifiability correctness*, and *modifiability completeness*. In there study they concluded that those measures are affected by the structural complexity of the class diagram.

As the size of a software system increases, usually its complexity increases as well. Therefore, it is important for engineers to find effective ways of communicating such complexity in an abstract and understandable manner. Cherubini et al.[25] studied how and why software developers used diagrams. The results indicated that diagrams are mainly used for supporting face-to-face communication. Additionally, the study also suggested that current tools were not effective at aiding developers externalize their mental models of the code.

It is important to also understand developer's perceived impact of UML usage in productivity and internal quality of software. Nugroho et al. [28] made a study to understand the impact that UML modeling styles had on both productivity and quality. The results suggest that developers perceive that using UML is most influential in improving software quality attributes such as understandability and modularity. In the context of productivity, the study indicated that UML is perceived to be most helpful at the stages of design, analysis, and implementation.

Most of the studies associated to UML usage are within the context of the industry. Nevertheless, Hebig et al. [31] studied how UML diagrams were used in open source software. They studied a total of 1.24 million projects from GitHub in order to understand how UML diagrams were used. Their results suggest that 26% of the projects investigated updated UML files at least ones. It also suggest that most projects introduce UML diagrams at the beginning stages of development and it is at such stages where engineers work with the UML diagrams.

In essence, models are an important asset in model-driven activities. As a result, UML diagrams have become the de facto standard for building such models. Due to the importance of those assets, it is necessary to understand how such models are used and their impact on the development process. Although research suggests that UML diagrams have a positive impact on external quality attributes such as maintainability and understandability, does using UML diagrams influence internal quality of software? If so, what internal quality attributes does it influence?

2.2 Internal Quality of Open Source Software

The role of open source software, in both the industry and economy, has increased over the years. The success of many open source systems is surprising given that such systems are developed by volunteer programmers that are dispersed through the globe and communicate in an informal or loose manner [20]. Although users are allowed to freely access and modify the source code of an open source software, the impact this type of software has in both the economy and industry is high. For example, in the year 2006 the European Commission's Directorate General for Enterprise and Industry financed a study to identify the economic impact of open source software in the information and communication technology sector in the European Union [22]. The study suggested that open source software can be found in markets such as web and email servers, operating systems, web browsers, and other Information and Communication Technology infrastructure systems. The same study also reveled that the volunteer work of the programmers represents 800 million Euros each year.

Some examples of successful open source systems include the Linux operating system, which represents 38% share of the operating systems market, the Apache web server, which accounts for 70% market share, and the FireFox web browser, which has been able to obtain 5% market share from Microsoft's Internet Explorer web browser [20]. Given the important role that open source software is having in the economy and mission-critical application, it is important to ensure that such system attain certain level of quality and security. Therefore, extensive research has been done to understand better the quality of open source systems. This section explores selected research on the area of software quality in the context of open source systems.

The benefit of open source projects is that researchers have the opportunity to access freely a large set of software development data. Such data could then be used to study, among other things, quality of open source projects. Nevertheless, as Yu et al. [18] demonstrate in their study, having a large set of open source maintenance data freely accessible does not guarantee that it would be useful for measuring maintainability of open source software. In such study, the authors studied various maintenance data such as defects from defect-tracking systems, change logs, source codes, average lag time to fix a defect, and more. They concluded that the reason why such data sets were not necessarily useful for measuring maintainability is that either, they were incomplete, out-of-date, inaccurate, lacked information regarding the origin of a defect, and lacked construct validity.

Although Yu et al.[18] mentions that source code is impractical for measuring maintainability, Bakar et al. [30] used the Chidamber and Kemerer (CK) object-oriented metrics to analyze the source code of two open source software and thus determine their internal quality. In such study, they wanted to investigate which quality factors had the biggest influence on class size as well as understand the correlation between various quality factors with class size. Their results indicate that coupling and complexity influences class size. Additionally, their correlation analysis results indicate that class size, expressed as lines of code, is significant in predicting coupling and complexity.

There are many factors that contribute to the success of an open source software. Aberdour [24] suggests in his study that having a large community of contributors was one of the most important factors that determined success. The study also suggests that software with high code modularity is a factor that motivates programmers to contribute to a project. Given that contributors are usually dispersed around the globe, been able to contribute to a system without knowing the architecture of the entire system is a benefit. But how can the core team of an open source system ensure their project obtains a certain degree of modularity? What role, if any, does UML diagrams play when creating architectures of open source systems? Yu et al.[18] tried measuring maintainability using data from defect-tracking systems, change logs, average lag time, etc. but what about measuring the internal quality metrics of the software as a method to understand maintainability? Bakar et al. [30] studied the relation between various metrics, but how are those metrics influenced whenever developers use modeling techniques in the development process of software?

2.3 Thesis Contribution

In order to create a software system, developers engage in a software development process. In such process, there are various factors that affect the quality of the final product; i.e., the internal quality of software. As shown in **Figure 2.1**, some of those factors include the software documentation used, project complexity, the skills of programmers and their experience, and the number of people involved in the project. The main contribution of this thesis is to study the relation that usage, quality, and size of documentation, and system size have on the internal quality of software.

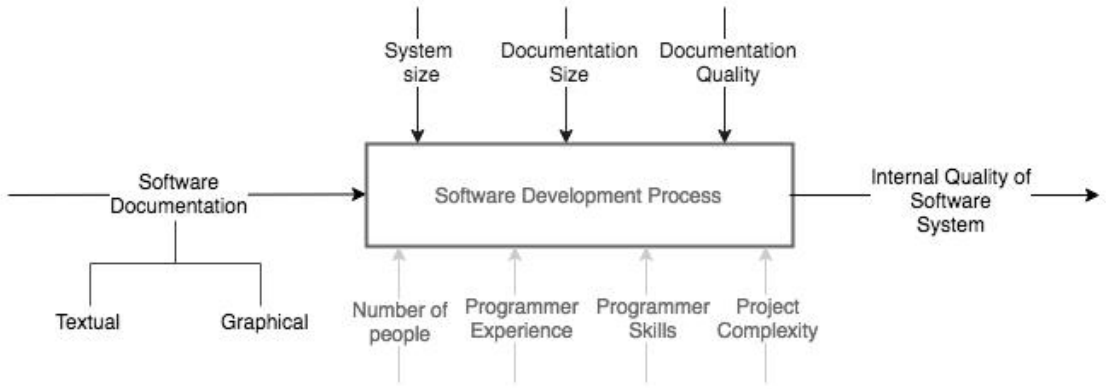


Figure 2.1: Factors that influence the development process and affect the internal quality of software.

The end goal of software development is to create system that are capable of satisfying the needs of a given market. In order to keep a software system relevant, it is essential to evolve it and maintain it. The cost and time involved in maintenance efforts depends on how good the internal quality of the software is. Therefore,

identifying the factors that influence the internal quality of software is an important step towards the creation of systems that are easily maintainable.

In the context of open-source software, being able to understand better how various factors influence the internal quality of open source software will allow the developers of the open source communities to develop software with higher internal quality and thus attract more volunteers to their community. Future sections will discuss in more details the research questions, hypothesis, and methods used for this thesis.

3
Methods

Recall that the purpose of this thesis is to study the relation that usage, quality, and size of documentation and system size have on the internal quality of software. **Figure 3.1** illustrates a high-level overview of the steps that were taken to accomplish such purpose.

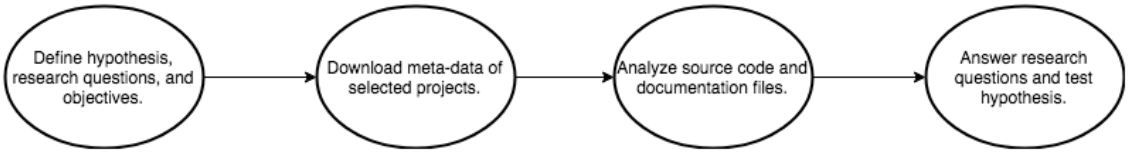


Figure 3.1

This chapter will provide a description of each step presented in **Figure 3.1** as well as a detailed explanation of how each step was be executed.

3.1 Hypothesis, Research Questions, & Objectives

The main research question of this study is the following:

- What is the correlation between quality of documentation and the internal quality of software?

Additional research questions of the study include the following:

RQ 1. What is the change of the software internal quality over time of the investigated projects?
RQ 2. How frequent is the documentation of open source software updated?
RQ 3. What is the quality of design-related documentation of the investigated projects?
RQ 4. What is the content of the documentation of the investigated projects?

As shown in **Figure 3.2** the answer to RQ 1 and RQ 2 serve as input for answering the main RQ. The reason for such relationship is because, in order to answer the main research question, it is necessary to first understand the quality of documentation and the quality of software internal quality of the selected projects separately. In this study software internal quality over time is studied in order to obtain a broader knowledge of how systems evolved over time and thus a better understanding of the projects analyzed. Although RQ 3 and RQ 4 do not serve as input for answering the main RQ, they are important for better understanding of the documentation.

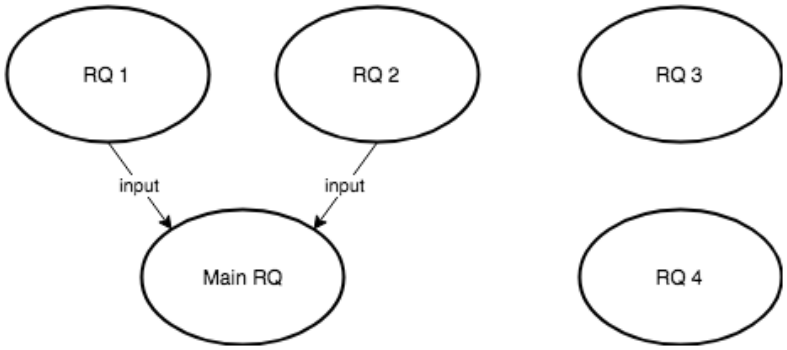


Figure 3.2: Relationship between proposed research questions.

Since the main research question of this study is investigate the correlation between documentation quality and internal quality of software. The hypothesis of this study are the following:

- **H_0:** $\rho = 0 \rightarrow$ There is no correlation between documentation quality and internal quality of open source software.
- **H_1:** $\rho \neq 0 \rightarrow$ There is a correlation between documentation quality and internal quality of open source software.

The significance level used in the hypothesis testing is 0.05

3.2 Collecting the Data

The second step shown in **Figure 3.1** is to download meta-data of selected project. Therefore, the purpose of this section is the following:

1. Describe the criteria used for selecting projects.
2. Describe the meta-data collected for each project.
3. Describe the method used for collecting the meta-data.

3.2.1 Selection Criteria

The GitHub platform was the source for downloading the projects to be studied. Projects could be part of this study as long as they satisfied the following requirements:

1. The project shall have more than one release [1]
2. The project shall be written in Java, C, or C++ [2]
3. The project shall contain UML diagrams

Millions of open source projects are available in the GitHub platform, but not all of those projects contain UML models and, most importantly, not all of those

[1] The reason for this requirement is explained in the *Analyzing the Data* section.
[2] The reason for this requirement is explained in the *Limitations* section.

projects have a size and complexity similar to projects commonly used in the market. For that reason, this research studies a subset of the projects studied by Hebig et al. [31]. In their study, Hebig et al. created a semi-automated approach for collecting UML models from projects in GitHub. Among their contributions is a list of 3,295 GitHub projects that include UML diagrams. The reason for using projects from such data set is because it facilitated the process of selecting projects that met the requirements of this study.

In order to compare how the internal quality of software that uses UML as part of their documentation differs from the internal quality of software that does not use UML diagrams as part of their documentation, this study includes a certain number of projects that do not use UML diagrams as part of their documentation.

3.2.2 Downloaded Meta-Data

For each selected project, the following meta-data was collected:

1. **Complete file directory for each release:** By downloading the complete directory for each release, we had access to the source code and documentation files used at each release. Having the source code of each release, allowed us to study the internal quality of each project over time. Having access to documentation file allowed us to study, its quality and patterns in usage. It is important to mention that the internal quality over time of documentation is not studied in this project. Instead, the study focused on analyzing internal quality for the latest release of documentation files.
2. **All the commits messages of the project:** The content of each commit messages were analyzed in order identify what files from the project directory was documentation.
3. **The date each release was published:** This information served as an index for organizing releases by date.

Additional meta-data collected includes number of contributors, a link for downloading source codes, the date each commit was submitted, and the default branch for each project. Although this information is considered secondary because it did not have a direct impact on the main purpose of this study; it helped provide different perspectives when analyzing the data.

3.2.3 Data Collection Method

So far the reader has been presented with the criteria used for selecting projects and the meta-data downloaded for each project. Now the reader will be presented with the methods used for collecting such meta-data.

3.2.3.1 Collecting Meta-Data of Selected Projects

The complete directory of each GitHub projects was automatically collected with the use of Python scripts that queried GitHub using the GitHub API [3]. For each

[3] To view the scripts used for data collection visit `https://bitbucket.org/hlthesis/`

project, the scripts downloaded all the meta-data mentioned previously. Although automation facilitated the collection of data, only 17 projects were part of this study. The reason for the inability to analyze more project was due to the limitations of the tools used for analyzing the internal quality of software [4]. From those 17 projects, 14 projects contained UML diagrams as part of its documentation, while 3 projects did not. **Appendix L** contains the name and description of the projects used in this study.

3.2.3.2 Identifying Documentation Files

In this study the term *documentation* refers to any file whose purpose is to explain or describe the architecture of the system or how the system works. As mentioned previously, commit messages were used for identifying what files in the directory were documentation. Identifying documentation files was accomplished as follows:

- A Python script searched for specific keywords that are associated to documentation in the message of each commit. The keywords used included the following:
  1. documentation
  2. uml
  3. diagram
  4. manual
  5. sad

  Regular expressions were used to identify any combination in which such keywords could appear in the commit message. If the script identified a commit message contained any of those keywords, then the commit was classified as a *documentation commit*. Each *documentation commit* was then linked to a project release by analyzing the date such commit was published. Additionally, all of the files associated with the *documentation commit* were downloaded using a python script and GitHub API.

  After downloading all the documentation files for each commit, all unique files were identified. Finally, the number of times each unique file was modified was calculated. After identifying all the unique files, the extension was analyzed in order to categorize the file as either:

  1. **Text Documentation File:** A file that contains information about the architecture of the system, how the system works, or how it is configured. Such information is provided in textual format.
  2. **Graphical Documentation File:** Just as textual documentation, it provides information about the architecture of the system, how the system works, or how it is configured but such information is provided in a graphical format using UML models or non-UML models.

  Appendix A contains the extensions that belong to each category.

[4] Details about the limitation will be presented in the *Limitations* section

3.3 Analyzing the Data

The third step in **Figure 3.1** is to analyze the meta-data. **Figure 3.3** shows the analysis that was made to the source code and documentation files of each project.

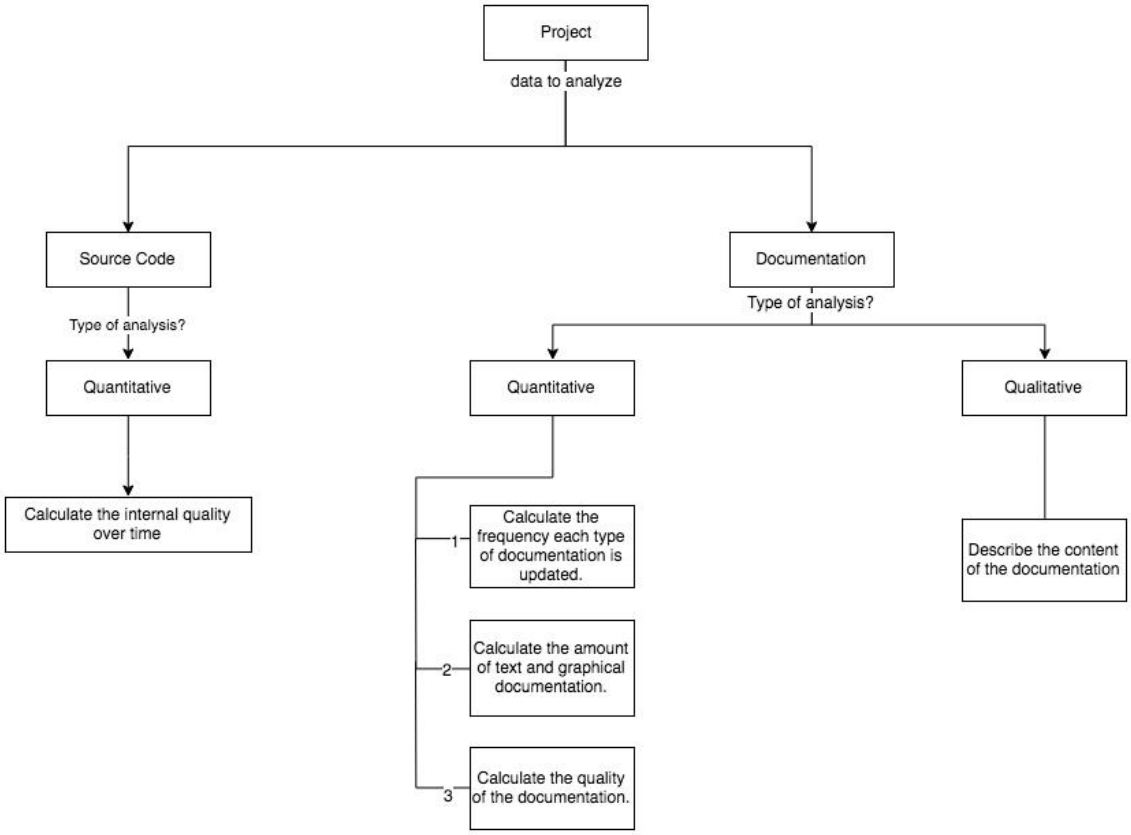


Figure 3.3: Strategy for analyzing data of the projects.

3.3.1 Source Code Analysis

As shown in **Figure 3.3** the source code of each release was analyzed in order study the internal quality over time for each project. In this subsection the following concepts will be discussed:

1. Describe what internal quality metrics are.
2. Describe the method used for calculating internal quality over time.

3.3.1.1 Internal Quality Metrics

For many years researchers have introduced many object-oriented metrics for measuring internal quality of software [2], [4], [6], [7], [16]. Nevertheless, many of those metrics have not been validated theoretically or empirically. Additionally, it is also common that such metrics are insufficiently generalized, too dependent on technology, or too computationally expensive to collect [2]. In this study, the Chidamber and Kemerer (CK) metrics were used for calculating the internal quality of open

source software [2]. The reason for using them is because there is research indicating their validity and usefulness for measuring internal quality of object-oriented software [1], [3], [9], [14].

In this study, all CK metrics [5] were calculated. Those metrics include:

- Weighted Methods per Class (WMC)
- Depth of Inheritance Tree (DIT)
- Number of Children (NOC)
- Coupling Between Objects (CBO)
- Response for a Class (RFC)
- Lack of Cohesion in Method (LCOM)

Besides the CK metrics, other metrics that were calculated include total lines of code, the total number of modules, and structural complexity.

3.3.1.2 Calculating Internal Quality Over time

The open source software *Analizo* [6] was used for calculating the metrics mentioned previously. Analizo is the result of a study done by Terceiro et al. [29] and its purpose is to calculate an extensive set of metrics from source code written in Java, C, or C++. Analizo was selected for this project because it satisfies all the following requirements [7]:

1. The software shall be open source.
2. The software shall belong to an active community.
3. The software shall have a consistent history of releases.
4. The software shall support automation.
5. The software shall not require source code to be compiled in order to generate the metrics.

Other tools were also explored [27]; nevertheless, those tools did not satisfy one or more of the requirements mentioned above. The rationale behind the above requirements is that the researchers of this study had the objective to embrace automation for data collection at the lowest monetary cost possible and in the most reliable way. Finding a tool that satisfied the requirements above was essential for accomplishing the goal of this study.

As mentioned previously, 17 projects were analyzed and each of those projects had a certain number of releases. Analizo was used to calculate the CK metrics of each release for each project. For example, if project X had 33 releases, then Analizo calculated the CK metric for each of the 33 releases. This approach enabled the understanding of how each project evolved over time; specifically, how each CK metrics changed from one release to another.

[5] The study by Chidamber et al.[2] provides a full description of each metric.

[6] `www.analizo.org`

[7] The rationale for these requirements is provided in the *Limitations* section.

3.3.2 Documentation Analysis

Unlike source code, the evolution of documentation overtime was not analyzed. Instead, only the latest release of the documentation files was analyzed. The purpose of this subsection is to describe how each documentation analysis, shown in **Figure 3.3**, was accomplished.

3.3.2.1 Quantity of Documentation and Update Frequency

As explained *previously*, documentation files were categorized as either source code, textual documentation, or graphical documentation. Therefore, counting how many documentation files in each project were textual and graphical was a trivial task of counting how many non-repeated files were in each category.

Additionally, since it was already known how many times each documentation file was modified, calculating the update frequency or the average number of times each file type was modified was a trivial task of calculating a mean. This study does not analyze how much information changed in a document after each update. It only focuses on determining how frequently documentation files are changed without taking into account the amount or type of changes made.

3.3.2.2 Documentation Content and Quality

Documentation was manually analyzed and it involved the tasks of reading each file and determining what aspects of the system they described. After understanding the content of each file, a set of guidelines were followed in order to grade the quality of documentation. **Appendix B** contains the set of guidelines used for measuring quality of documentation.

Recall that in this study two types of documentation are studied: *Graphical* and *Textual* [8]. In the case of *Graphical* documentation, the quality of only UML diagrams was analyzed. In the case of *Textual* documentation, the quality of only the files whose content described UML models or architecture of the system was analyzed.

It is common for developers to consider source code comments as a form of documentation. In many cases, source code comments are used to describe the purpose of modules, methods, and possibly, the way algorithms are implemented. Nevertheless, in this study source code comments are not analyzed. The reason was because of time constraints and the inability to find a tool capable of automatically parsing and filtering comments that could be considered documentation from those that were not documentation.

3.4 Answer Questions & Test Hypothesis

The fourth and final step shown in **Figure 3.1** is to answer the research questions and test the hypothesis. Research questions 1, 2, 3, and 4 were answered by

[8] Section *Identifying Documentation Files* explains the difference between the types of documentation.

interpreting the meaning of the results obtained in the third step.

To answer the main research question and test the hypothesis, a correlation analysis was performed. The result of the correlation analysis provided information regarding magnitude and direction of the correlation between selected factors and internal quality of software.

3.4.1 Selecting Type of Correlation Analysis

Pearson's and Spearman's correlation are two types of analysis used for calculating the correlation between variables. To determine if Pearson's correlation was an appropriate analysis, our data was tested to determine if it complied will all of Pearson's requirements. Those requirements include the following [9]:

1. Variables must be measurements of type ratio o intervals.
2. The data of the variable should be normally distributed.
3. There should be a linear relationship between the variables.
4. The data should have little to no outliers.
5. There is homoscedasticity in the data.

In order to determine if our data complied with such requirements, the following tests were done:

1. **Shapiro-Wilk Test:** Used for testing normality of the data.
2. **Levene's Test:** Testing homoscedasticity
3. **Boxplot:** To determine if the data had outliers.
4. **Normality Plot**: To visualize how normal the data is.

The results of these tests are presented in the *Results* section.

[9] `https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php`

4

Results

Recall from **Figure 3.2** that the answers to research questions 1, 2, and 3 served as input for the answer to the main research question. Therefore, this section begins by answering research questions 1, 2, 3, and 4, followed by the answers to the main research question.

4.1 RQ 1: Internal Quality of Software Over Time

RQ 1 asked *What is the change of the software internal quality over time of the investigated projects?* As mentioned previously, a total of 17 open source software projects were part of this study and for each project, the internal quality of each release was analyzed. **Figure 4.1** illustrates the change in each CK metric for the project with ID **9c699c33**.

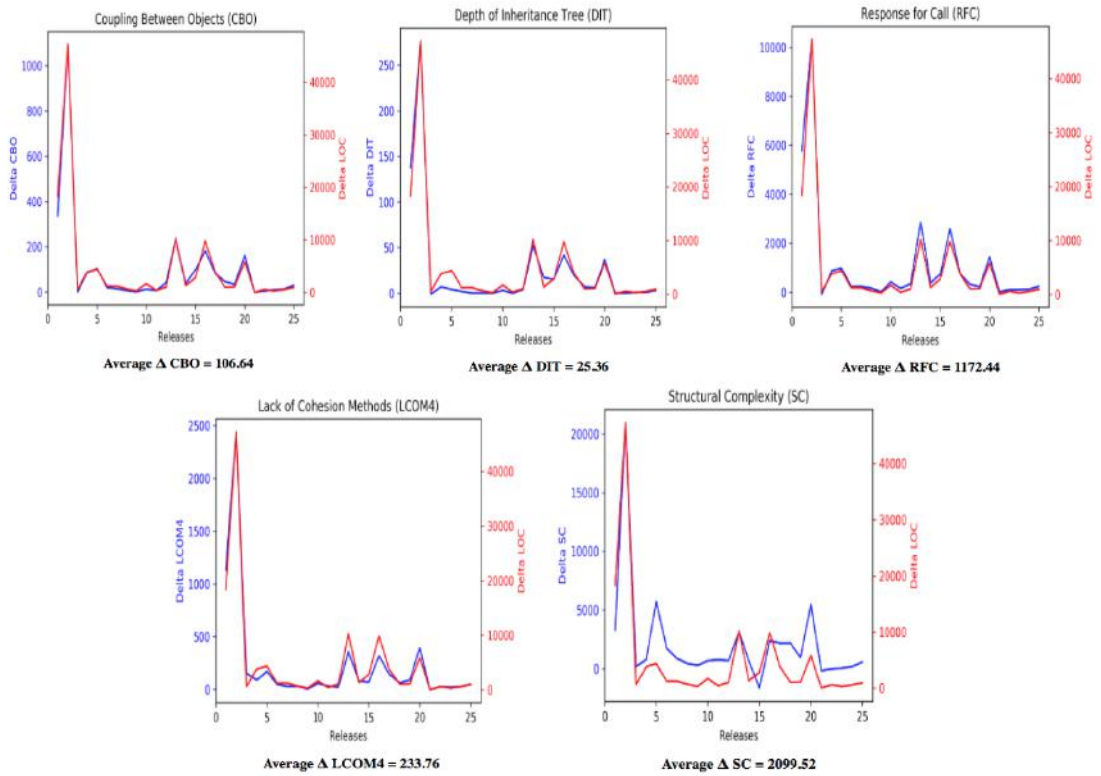


Figure 4.1: Change in CK Metrics for project with ID 9c699c33

Since RQ 1 is focused on understanding *change* of software internal quality over time, the *delta* value or change from release X to release X+1 of each CK metric was calculated. To illustrate how the *delta* value is calculated let's imagine that Project X has a CBO for release 1 of 20 and a CBO for release 2 of 15. Then the change from release 1 to release 2 is -5. The benefit of using the *delta* value is that such value indicates the magnitude and direction of change for a given internal quality metric. In the given example there is a decrease in CBO of 5 units.

Results for RQ 1 - Part 1

The direction of change for each metric is similar to the direction of change of LOC. In other words, both the change in LOC and the change in any CK Metric have the tendency to move in the same direction from one release to another. For example, the change in LOC from release 1 to release 2 was positive and so was the change in the CK metrics. More interestingly, this behavior is also present in the other projects.

In order to better visualize the results obtained, **Figure 4.2** illustrates the behavior of all CK metrics in a single graph.

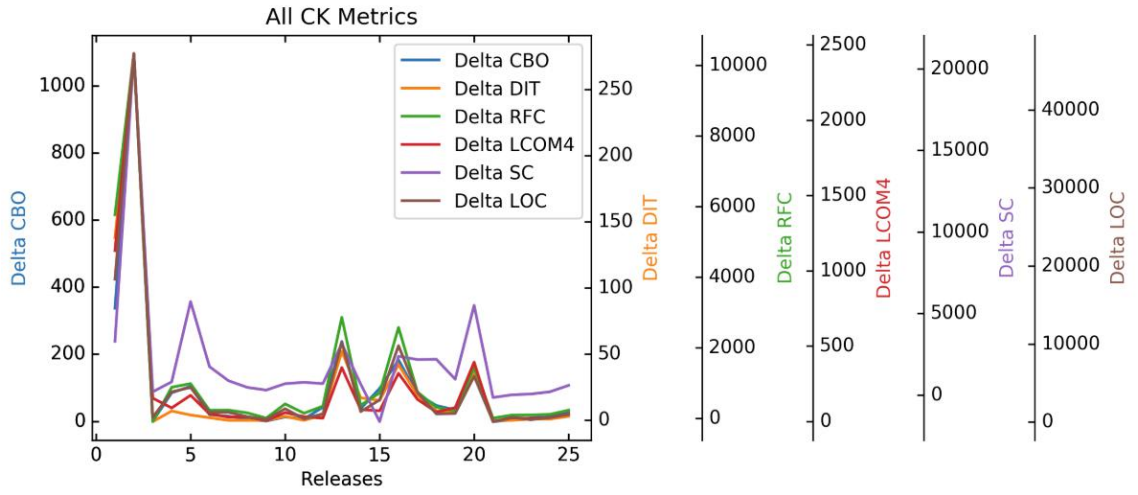


Figure 4.2: The change in each CK Metrics for project with ID 9c699c33

Results for RQ 1 - Part 2

The change in all CK Metrics tend to move in the same direction from one release to another. More interestingly, the same behavior is present in the other projects.

Appendix C contains the same set of graphs as shown in **Figure 4.1** and **Figure 4.2** but for all projects.

The data shown so far regarding CK metrics is associated to the *change* of each metric from one release to another. Additionally, results indicate that there is a tendency for all CK metrics to follow a similar direction of change from one release to another. Therefore, in order to better explore the internal quality of each

project, the CBO and LOC for each release of each project was explored. **Figure 4.3** illustrates the CBO and LOC for all releases of selected projects (Note that the CBO and LOC is graphed and not the *change* in CBO neither the *change* in LOC, as done previously).

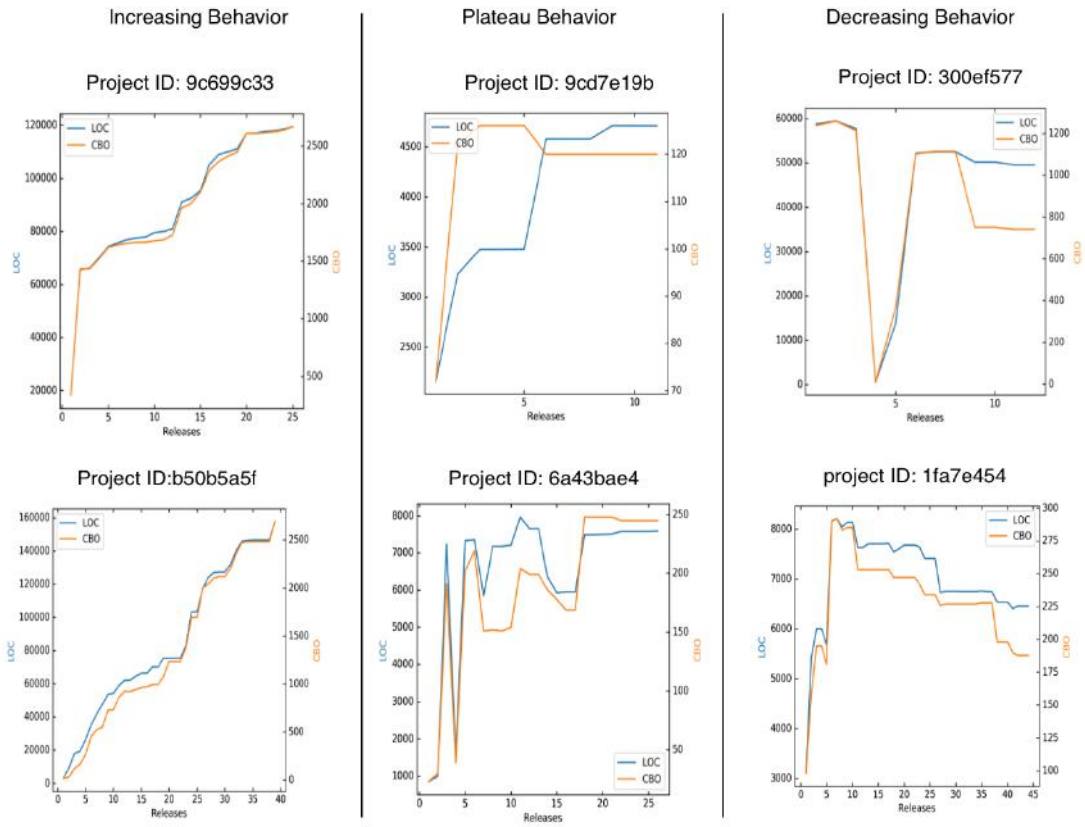


Figure 4.3: CBO and LOC of the releases of selected projects

As observed in **Figure 4.3**, both CBO and LOC tend to follow the same magnitude from one release to another. This behavior is also present on the other projects studied. Additionally, it may be observed that CBO and LOC tend to follow one of the following behaviors:

Behaviors	Description
Increasing Behavior	The CBO and LOC tend to increase after each release but not necessarily at the same rate.
Plateau Behavior	The CBO and LOC plateaus at a certain point and either does not change or the change is small.
Decreasing Behavior	The CBO and LOC tend to decrease after each release but not necessarily at the same rate.

From the 17 projects studied, 12 had an increasing behavior, 2 had a plateau behavior, and 3 a decreasing behavior.

Results for RQ 1 - Part 3

Both LOC and CBO tend to follow the same direction at each release. Additionally, it is common for such metrics to increase after each new release. Additionally, since all the metrics are closely related in behavior, it is likely that other CK metrics also increase with each new release.

Part of understanding the internal quality of software involves measuring the quality of source code structure. In order to calculate such measure, the magnitude of CBO at release X is divided by the number of lines of code at release X. The quality of the source code structure is inversely proportional to the result of the division. Therefore, the lower the result of the division, the higher the quality. **Figure 4.4** illustrates the various behaviors of the source code structure quality encountered in the study.

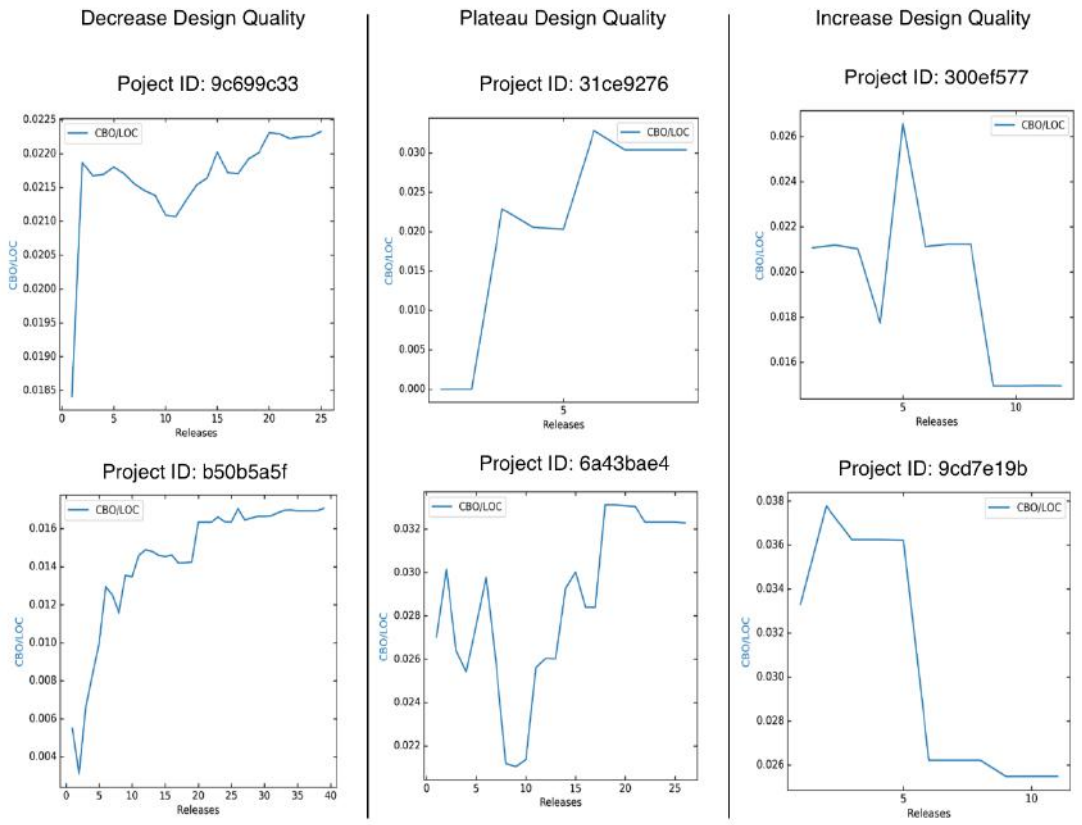


Figure 4.4: CBO of the releases of selected projects

Similarly to the behavior of CBO and LOC, the quality of the source code structure also present the increasing, plateau, or decreasing behavior previously explained. In the case of quality of source code structure, eight projects presented an increasing behavior; i.e., the quality of source code structure tended to deteriorate. Seven projects presented a decreasing behavior; i.e., the quality of the source code

structure tended to improved, and two projects had a plateau behavior; the quality of the source code structure tended to stay approximately the same.

Results for RQ 1 - Part 4

Approximate the same amount of projects presented an increasing and decreasing source code structure quality. For those projects that presented a decreasing behavior, the rate at which the size of the project increased was higher than the rate at which the CBO increased. The opposite occurred for projects with an increasing behavior.

Appendix D contains the same set of graphs as shown in **Figure 4.3** and **Figure 4.4** but for all projects.

4.2 RQ 2: Frequency of Documentation Update

RQ 2 asked *How frequent is the documentation of open source software updated?* As mentioned previously, all the commit messages of each project were analyzed and filtered in order to identify documentation files from source code files. As explained in the *Methodology* section, in this study, documentation is classified as either *Textual* or *Graphical*. **Figure 4.5** illustrates the five distributions of documentation types encountered in the projects studied.

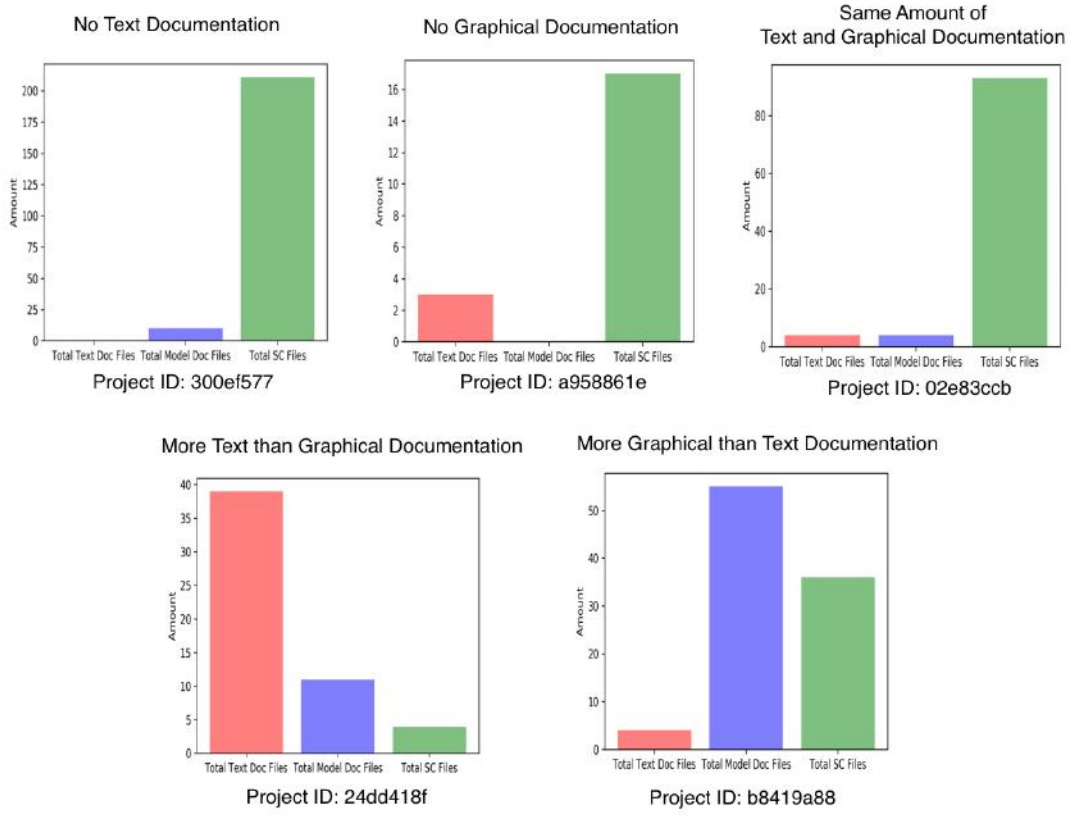


Figure 4.5: Documentation Distribution

Based on the projects studied, documentation of was distributed as follows:

Documentation Distribution	Description
No Text Documentation	One project presented this behavior.
No Graphical Documentation	Two projects presented this behavior.
More Text than Graphical	Nine projects presented this behavior.
More Graphical than Text	Four projects presented this behavior.
Same Amount of Text and Graphical	One projects presented this behavior.

Results for RQ 2 - Part 1

It is common for projects to provide documentation; however, such documentation is mainly presented as textual rather than graphical. Projects contained an average of 24 files associated to textual documentation; while an average of 12 files were associated to graphical documentation.

Through the lifetime of a project, besides changes to the source code of the system, changes to its documentation are also made. However, not all documentation of a project is updated with the same frequency. **Figure 4.6** illustrates the possible behaviors regarding documentation update based on selected projects.

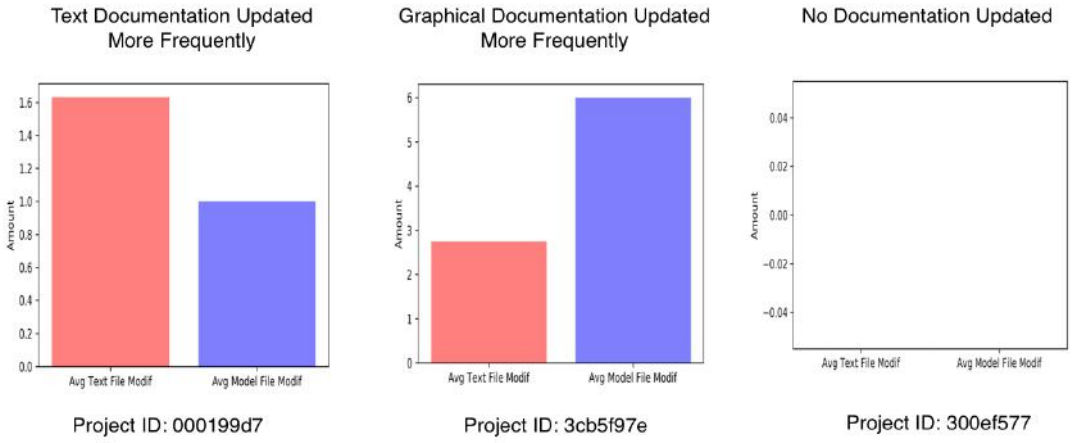


Figure 4.6: Cases for Documentation Update Frequency

Based on the projects studied, the update frequency of documentation could follow one of the following behaviors:

Documentation Update Behaviors	Description
Text documentation updated more frequently	Nine projects presented this behavior.
Graphical documentation updated more frequently	Seven projects presented this behavior.
No documentation update	One projects presented this behavior.

Moreover, in 14 projects, the documentation type that was most frequently updated was the one most available in the project. Except in 3 projects in which the opposite occurred.

Results for RQ 2 - Part 2

Since textual documentation was the prominent type of documentation present, it was the type of documentation most frequently updated, however, but not by a large difference. On average, textual documentation was modified 2.49 times; while, graphical documentation was modified 2.33 times.

Appendix E contains the same graphs as in **Figure 4.5** and **Figure 4.6** but for all projects.

4.3 RQ 3: Documentation Quality

RQ 3 asked *What is the quality of design-related documentation of the investigated projects?* As explained previously, in order to answer such question, the quality of both UML diagrams and SAD files was analyzed. **Figure 4.7** illustrates the quality of the UML documentation for selected projects.

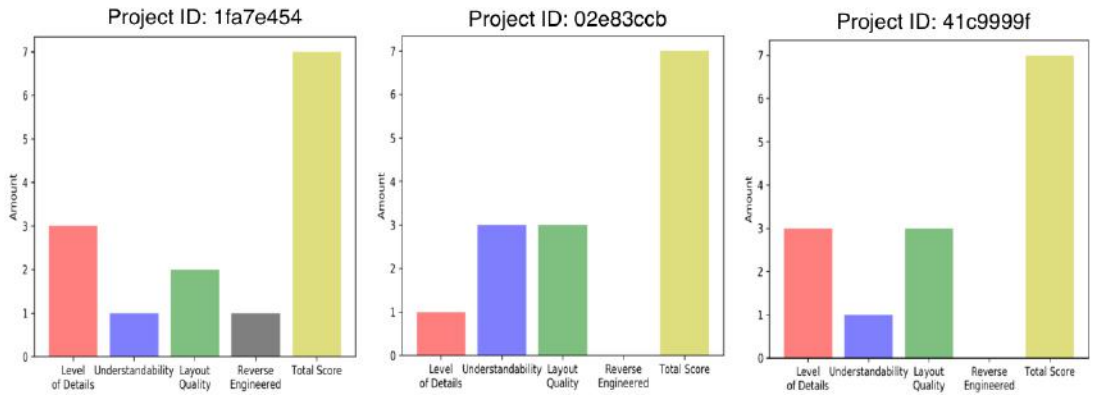


Figure 4.7: Result of UML documentation quality for selected projects.

From a total of seventeen projects, only fourteen contained UML diagrams as part of their *graphical* documentation. Based on those fourteen projects, the average for each quality attribute was the following:

Quality Attribute	Average Score
Level of Detail	2.25 out of 3.0
Understandability	2.41 out of 3.0
Layout Quality	2.91 out of 3.0
Usage of Reverse Engineering tool	0.16 out of 1

Results for RQ 3 - Part 1

Projects that contained UML diagrams have the tendency to have high scores for all the quality attributes; with *Layout Quality* having the highest average score. Results also indicate that models are, generally, constructed manually. Finally, the average total quality of UML documentation was 7.75 out of 10 or 77.5 out of 100.

After studying the quality of UML documentation, the quality of SAD documentation was studied as well. **Figure 4.8** illustrates the quality of SAD documentation for selected projects.

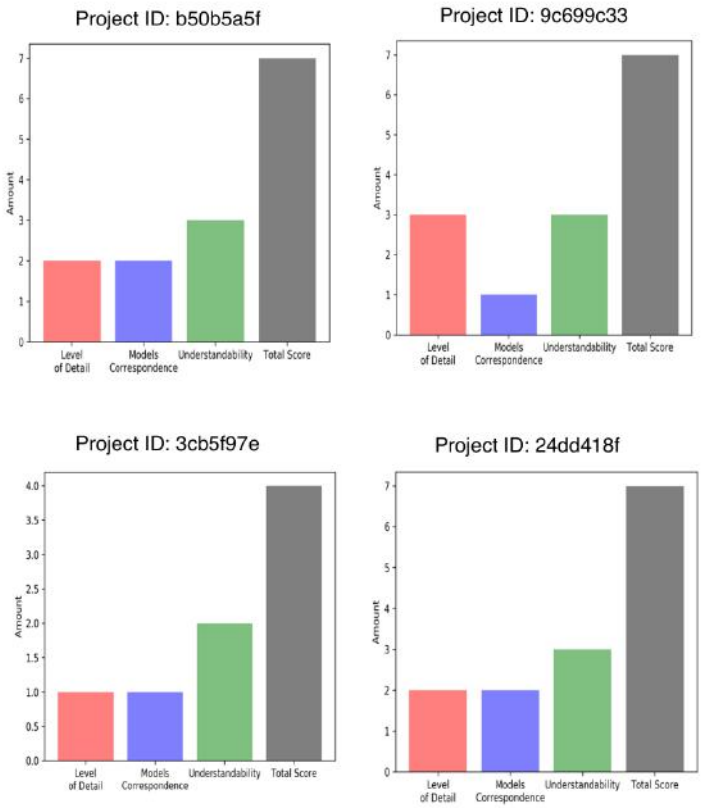


Figure 4.8: Result of SAD documentation quality for selected projects.

From a total of seventeen projects, only four contained SAD files as part of their *textual* documentation. Based on those four projects, the average score for each quality attribute was the following:

Quality Attribute	Average Score
Level of Detail	2.0 out of 3.0
Correspondence to model	1.5 out of 3.0
Understandability	2.75 out of 3.0

Results for RQ 3 - Part 2
Projects that contained SAD files have the tendency to have high scores for attributes such as *level of detail* and *understandability*. Finally, the average total quality of SAD files was 6.25 out of 9 or 69.4 out of 100.

Appendix H contains the same graphs as in **Figure 4.7** and **Figure 4.8** but for all projects.

In order to calculate the overall documentation quality of the documentation of each project, the quality result of UML documentation and SAD files were added together. Since a large number of projects lacked SAD documentation, the average score of overall documentation quality was low.

Results for RQ 3 - Part 3
The average score for overall documentation quality was 8.57 out of 19 or 45 out of 100. This low score is a result of a significant number of projects not containing SAD documentation.

4.4 RQ 4: Documentation Content

RQ 4 asked *What is the content of the documentation of the investigated projects?* As expected, the content of the documentation varied from project to project and depends on the type of documentation as well. Nevertheless, there was a common pattern regarding the content of documentation. In the case of projects that had *Graphical* documentation, the content could be either mainly UML diagrams or mainly non-UML diagram. **Figure 4.9** illustrates how *Graphical* documentation was distributed in selected projects.

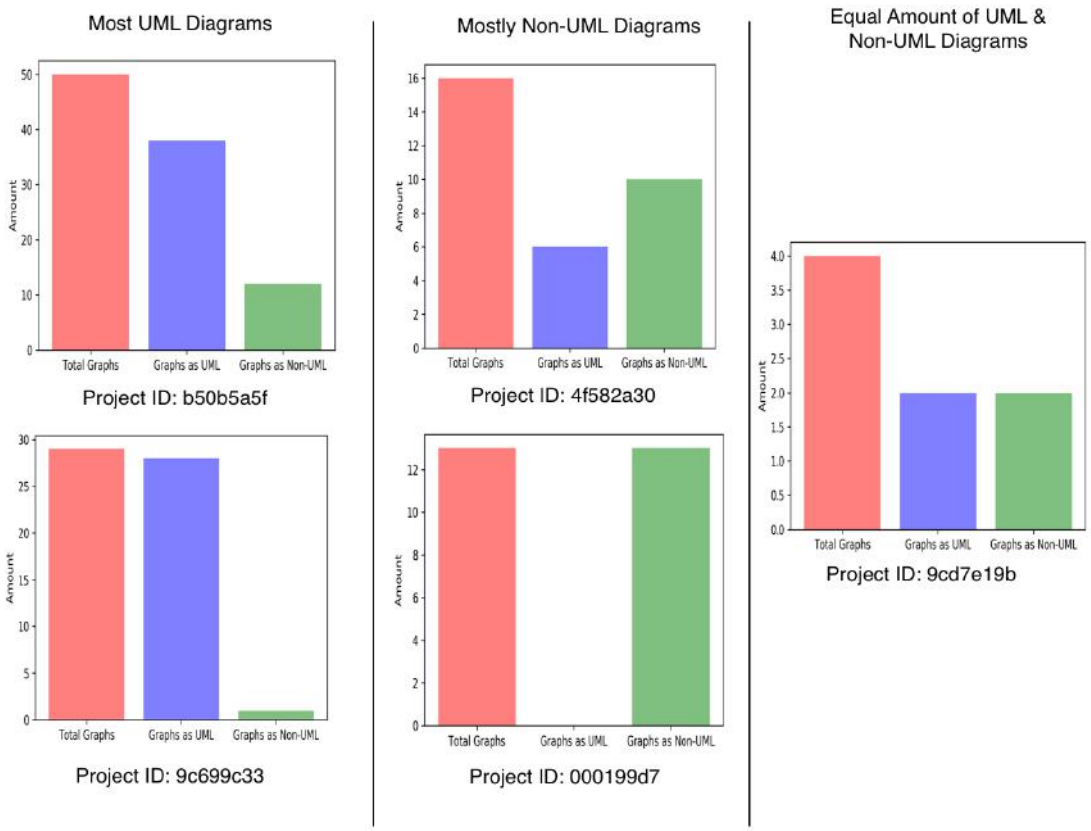


Figure 4.9: Graphical Documentation Content

Based on the projects who contained *Graphical* documentation, its content could contain:

Content Type	Description
Mainly UML diagrams	Twelve projects presented this behavior.
Mainly Non-UML Diagrams	Two projects presented this behavior.
Equal amount of UML and Non-UML	One project presented this behavior.

In general, for projects that used UML diagrams, the diagram most used was class diagrams; followed by sequence diagrams. On the other hand, Non-UML diagrams were either screenshot of a software or boxes and arrows connections that do not follow the UML standard.

UML diagrams may be automatically created or manually created. However, Regardless of the method used for creating them, a UML diagram file may contain an image of the diagram (UML as Image) or XML (UML as Text), which is then compiled in order to generate the diagram. In the case of projects who had UML diagrams, projects used a combination of text or image to express their UML diagrams. **Figure 4.10** illustrate how UML diagrams were expressed.

Results for RQ 4 - Part 1

For project that contain *graphical* documentation, it is common for such documentation to be presented as UML diagrams. Projects had an average of 12 graphical documentation files. From those 12 files, an average of 10 files were UML diagrams and 2 files were non-uml diagrams. Additionally, class diagram was the most common type of UML diagram used, followed by sequence diagram.

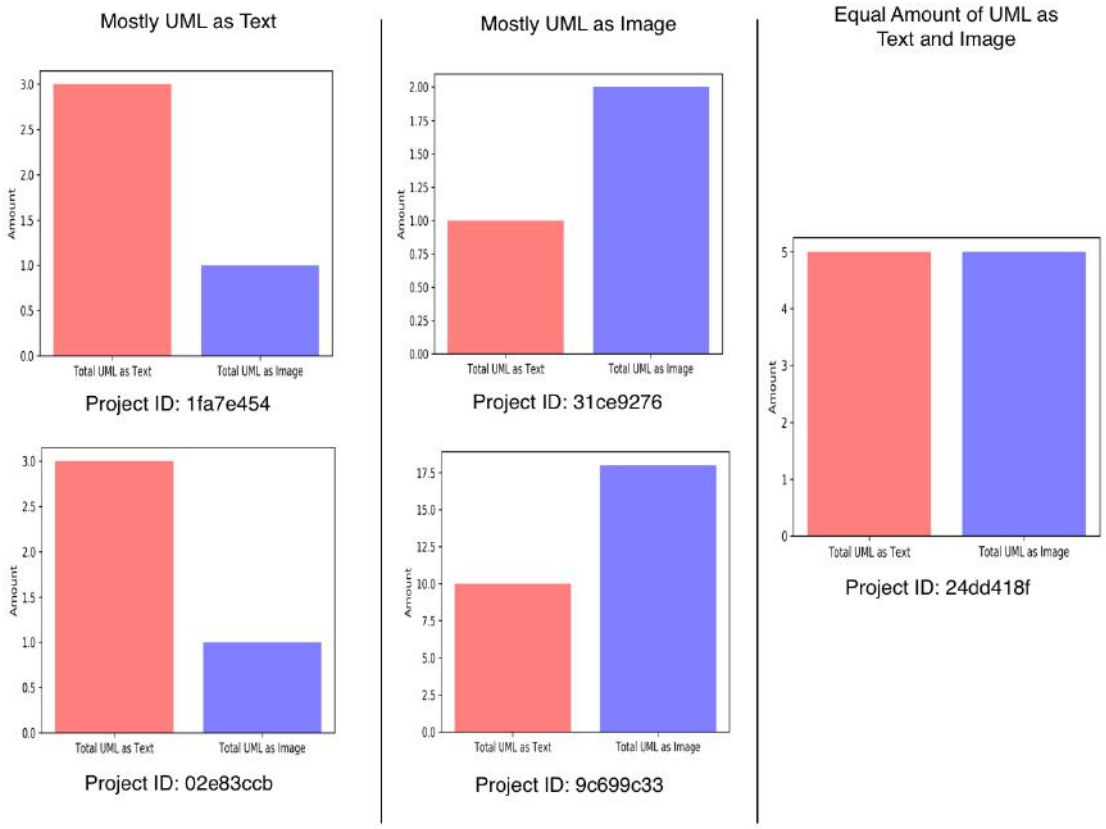


Figure 4.10: Ways of Expressing UML Diagrams

Based on the projects that contained UML diagrams, such diagrams were expressed in one of the following forms:

UML Format	Description
UML as Text	Five projects expressed most, if not all, of their UML diagrams as text; i.e., the file contained XML code that needed to be compiled in order to create the diagram.
UML as Image	Eight projects expressed most, if not all, of their UML diagrams as an image; i.e., the file is an image format and not XML format.

Additionally, one project presented all of its diagrams in both text and image format. **Appendix F** contains the same graphs as in **Figure 4.9** and **Figure 4.10** but for all projects.

Results for RQ 4 - Part 2

The UML diagrams are expressed in text format files with extension such as .xmi, .uml, and .puml. However, it is more frequent that diagrams are expressed as image files such as .png, .svg, and .jpg. From all the UML diagrams provided, on average 7 of those diagrams were presented as images, while 3 of those diagrams were presented as text format.

So far, the results for RQ 4 have provided information regarding the content of *graphical* documentation. However, *textual* documentation can also be classified based on its content. In this study, files that are considered *textual* documentation can be classified as either Software Architecture Documentation (SAD) or Non-SAD. **Figure 4.11** illustrates how *textual* documentation is categorized for selected projects.

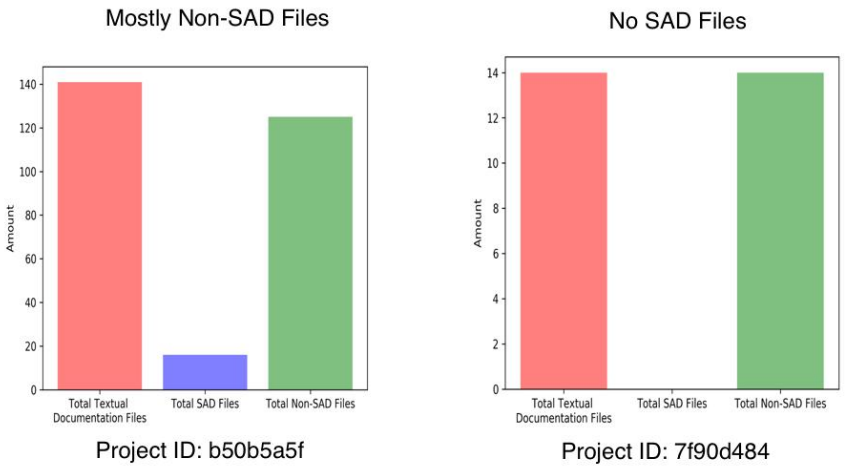


Figure 4.11: Textual Documentation Content

After analyzing the *textual* documentation of all projects, it was concluded that the analyzed projects followed one of the following behaviors:

Behavior	Description
Little to no SAD Files	This means the project contained little to no SAD documentation. Only four projects had SAD documentation.
No SAD Files	This means that projects contained no SAD files at all. A total of twelve projects presented this behavior.
No Documentation	Only one file presented this pattern.

Appendix G contains the same graphs as in **Figure 4.11** but for all projects.

Results for RQ 4 - Part 3

Projects do not commonly contain SAD files. However, when they do, such files contain explanations about the models and architecture of the system. On the other hand, non-SAD files contain non-architectural information such as building instructions, how to execute tests, how to use certain modules, contribution guidelines, and release history. Projects had an average of 24 textual documentation files. From those 24, an average of 23 was non-SAD documentation and 1 was SAD documentation.

4.5 Main RQ: Correlation Analysis

Recall that the main research question asked: *What is the correlation between quality of documentation and the internal quality of software?* Although, a significant amount of project meta-data was collected not all of such data was part of the correlation analysis. The variables that were taken into account for the correlation analysis were the following:

- Documentation Files to Source Code Files commit ratio
- Number of Documentation Files
- Number of Source code files
- Total Documentation Quality
- Total number of LOC
- Total number of Modules
- CBO Average

Appendix I contains the complete dataset used in the correlation analysis. All 17 were part of the correlation analysis.

Notice that *CBO* is the only CK metric that is part of the analysis. The reason for excluding the other CK metrics and other metadata was in order to keep the number of variables to a minimum and thus, simplify the analysis. This simplification was achieved by selecting the most relevant variables that would allow answering the main research question. The reason for choosing CBO as part of the analysis and not another CK metric is because of the researcher's interest in exploring the aspect of architecture-design quality of systems. Moreover, previous results demonstrate the high positive correlation between the CK metrics; therefore, it is expected that results would have been very similar regardless of the CK metric chosen.

It is important to mention that *CBO Average* is not a metric provided by Analizo. Instead, it is a metric that resulted from dividing *CBO Sum* (the value provided by Analizo) by the total number of modules. The reason for performing such division is because *CBO Sum* was a sum of the CBO value of each module in a project. Therefore, to obtain a value that provided a better representation of the overall design of the system, the CBO Sum was normalized by dividing it by the number of modules in the system [23].

4.5.1 Correlation Analysis Results

Recall that to determine if Pearson's correlation was the appropriate correlation analysis method to use with our data, it was necessary to ensure that such data complied with various conditions.

Figure 4.12 illustrates the normality Plot for selected variables. As shown in the image, many variables do not follow a normal distribution. The Shapiro - Wilk Test confirmed that multiple variables did not have a normal distribution. To be more specific, 5 out of 7 variables did not have a normal distribution.

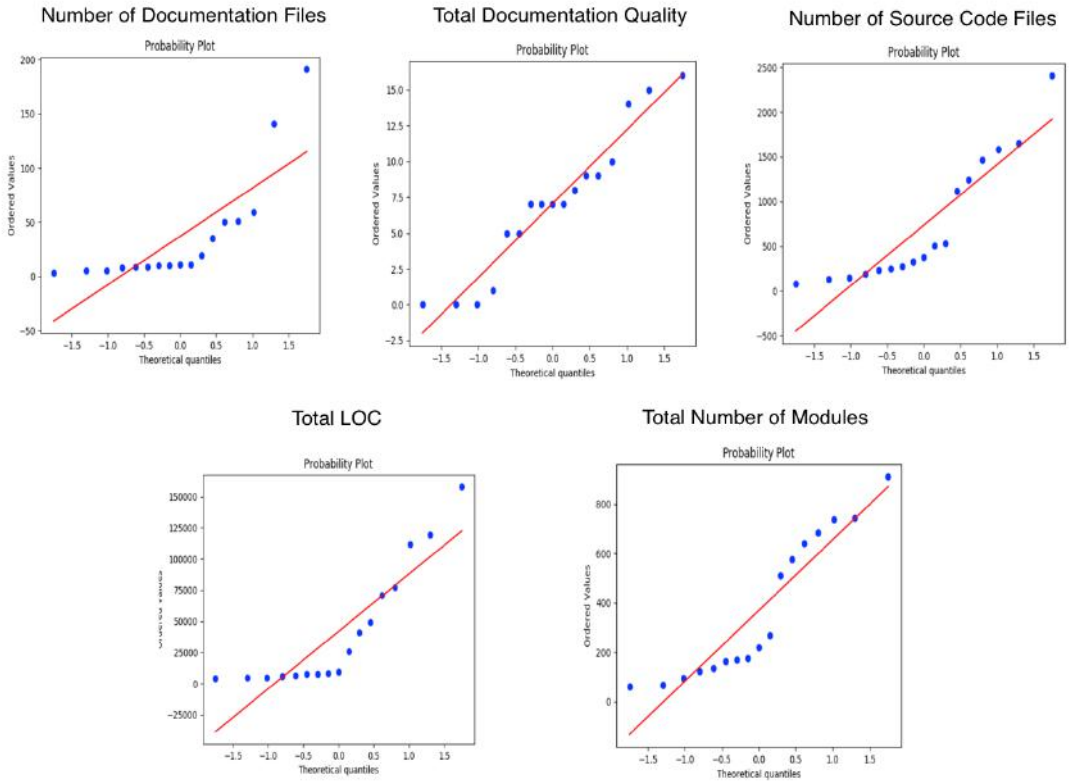


Figure 4.12: Normality Plot for selected variables

Another condition that must be met for using Pearson's correlation is that there should be no outliers in the dataset. **Figure 4.13** shows the box plot for selected variables. As shown in the image, there are various variables that do contain outliers. The results indicate that 2 out of 7 variables contain outliers.

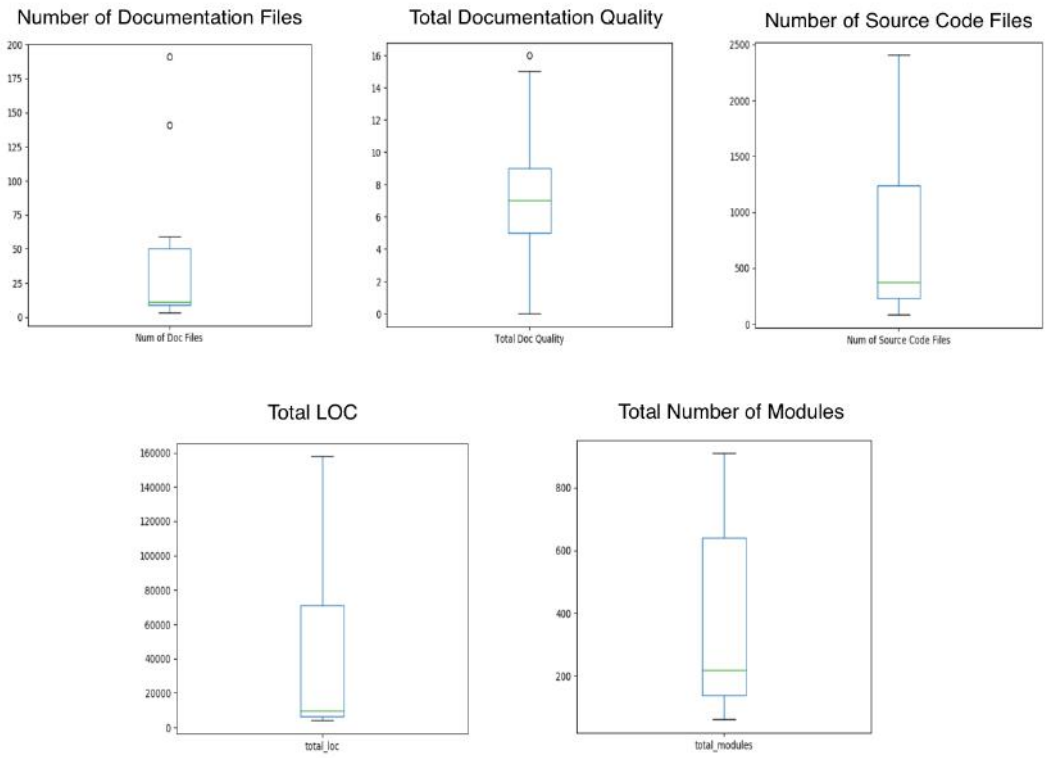


Figure 4.13: Box Plot for selected variables

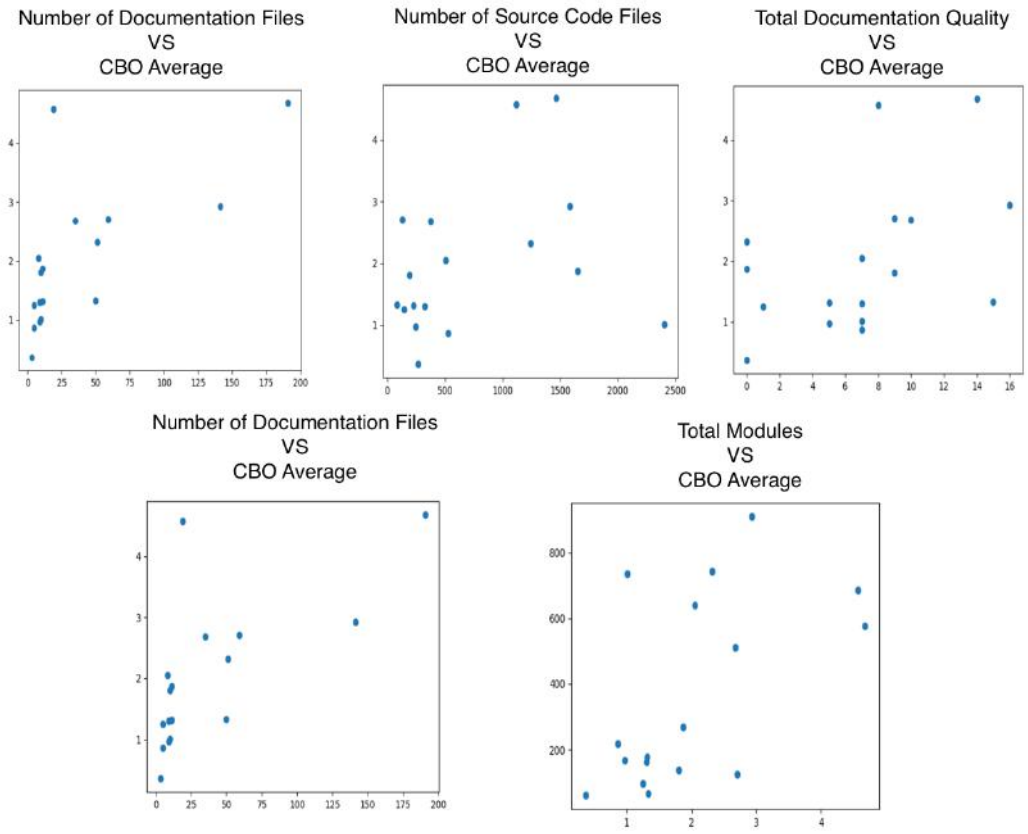


Figure 4.14: Correlation graphs between CBO Average and other variables

Finally, to observe the homoscedasticity or equality of variance of the data, a correlation plot between *CBO Average* and all other metrics were created. **Figure 4.14** illustrates the results of the correlation graphs. As shown in the image, the data tends to not have equality of variance. The results from the Levene's test confirms that all graphs lack homoscedasticity.

Appendix J contains normality plot and box plot for all variables and **Appendix K** contains the correlation graphs.

Results for Main RQ - Part 1

Because variables from the input data of the correlation analysis has outliers, is not normally distributed, and lacks homoscedasticity, a Pearson's correlation is not an appropriate correlation method. Instead, Spearman's correlation was the analysis method used.

After determining that Spearman's correlation was the appropriate analysis method to use, a correlation matrix was generated with the use of SPSS software. **Figure 4.15** shows the resulting correlation matrix.

Correlation Matrix

		DoctoSCCommitratio	NumofDoc Files	Num of Source CodeFiles	TotalDoc Quality	total_loc	total_modules	cbo_avg
DoctoSCCommitratio	Correlation Coefficient	1.000	-0.058	-0.473	0.222	-0.221	-0.297	-0.010
	Sig. (2-tailed)		0.826	0.055	0.393	0.395	0.248	0.970
NumofDocFiles	Correlation Coefficient	-0.058	1.000	0.201	.584*	.494*	0.419	.828**
	Sig. (2-tailed)	0.826		0.438	0.014	0.044	0.094	0.000
NumofSourceCodeFiles	Correlation Coefficient	-0.473	0.201	1.000	-0.068	.787**	.836**	0.260
	Sig. (2-tailed)	0.055	0.438		0.795	0.000	0.000	0.314
TotalDocQuality	Correlation Coefficient	0.222	.584*	-0.068	1.000	0.188	0.145	.552*
	Sig. (2-tailed)	0.393	0.014	0.795		0.470	0.579	0.022
total_loc	Correlation Coefficient	-0.221	.494*	.787**	0.188	1.000	.922**	.635**
	Sig. (2-tailed)	0.395	0.044	0.000	0.470		0.000	0.006
total_modules	Correlation Coefficient	-0.297	0.419	.836**	0.145	.922**	1.000	.510*
	Sig. (2-tailed)	0.248	0.094	0.000	0.579	0.000		0.037
cbo_avg	Correlation Coefficient	-0.010	.828**	0.260	.552*	.635**	.510*	1.000
	Sig. (2-tailed)	0.970	0.000	0.314	0.022	0.006	0.037	

*. Correlation is significant at the 0.05 level (2-tailed).
**. Correlation is significant at the 0.01 level (2-tailed).

Figure 4.15: Correlation matrix

Results for Main RQ - Part 2

The main insights obtained from the significant results of the correlation matrix are the following:

1. There is an indication of a strong positive correlation between *CBO Average* and size of the system. However, the correlation between *CBO Average* and LOC is stronger than the correlation between *CBO Average* and the number of modules.
2. There is an indication of a strong positive correlation between *CBO Average* and documentation size.
3. There is an indication of a strong positive correlation between *CBO Average* and documentation quality.
4. From all the variables in presented in **Figure 4.15**, the two with highest correlation with *CBO Average* are *number of documentation files* and *total_loc*.

Appendix M contains information regarding categorizing the value of correlation coefficients into a given strength scale.

4.6 Results of Hypothesis Testing

The correlation matrix in **Figure 4.15** shows various correlation values that are significant; however, recall that the hypothesis of the study are the following:

- **H_0:** $\rho = 0 \rightarrow$ There is no correlation between documentation quality and internal quality of open source software.
- **H_1:** $\rho \neq 0 \rightarrow$ There is a correlation between documentation quality and internal quality of open source software.

Therefore, in order to determine if the null hypothesis is rejected or failed to be rejected, it is necessary to examine the correlation coefficient for the correlation between documentation quality and *CBO Average* and its significance.

Results of Hypothesis Testing

The correlation coefficient between *documentation quality* and *CBO Average* is 0.552 with a p-value of 0.022. Therefore, this results indicate that the null hypothesis can be reject. Rejecting the null hypothesis indicates that there is evidence to suggest there is a correlation between *documentation quality* and *CBO Average*. Since the correlation coefficient is 0.552 such correlation is positive.

5

Discussions

The data presented in the *Results* section answered questions associated to internal quality of software over time, the relation between the CK metrics, the frequency that documentation is updated, documentation content, and documentation quality. Recall that the purpose of this research is to study the relation between documentation quality and internal quality of software. However, before identifying this relation, if any, it is essential to study the factors of software internal quality and documentation quality separately. Therefore, the purpose of this section is to discuss the results of studying internal software quality, documentation content, and documentation quality.

5.1 Internal Quality of Software Over time

The results presented in *RQ 1 - Part 3* suggested that software tends to have an increasing behavior of LOC over time. That is, for each new release, software tends to increase in size (measured in LOC). Additionally, the results for *RQ 1 - Part 1, 2, 3* suggest that there is a strong positive correlation between the CK metrics and LOC. Both results are important because they suggest that it is natural for CK metrics to increase as the size of the system increases. Since more classes and relations are introduced as software increases, it is expected that coupling, complexity, and other metrics will increase as well. But by how much? In other words, if the software increases by 1000 LOC by how much does each CK metrics increases? The answer to such question depends on other factors such as the quality of architecture design. For example, introducing 1000 new LOC containing a bad architectural design may produce a much higher value of CK metrics than introducing 1000 new LOC containing a good architectural design. Therefore, although the strong positive correlation between LOC and CK metrics exists, the analysis does not explain how much of the LOC effect plays a role in the internal quality of software.

Besides suggesting a correlation between CK metrics and LOC, the results obtained also demonstrate a correlation between the metrics. In other words, the magnitude and direction of change in each metric from one release to another were similar. However, the idea of a correlation between LOC and CBO is not new. Bakar et al.[30] presented results also suggesting a strong positive correlation between those two factors.

5.2 Documentation Updates and Content

As expressed by De Souza et al. [17] documentation is an artifact that plays an important role in development and maintenance of software. However, it is an artifact that is often either missing or out-of-date. As mentioned before, from all projects analyzed, only one lacked documentation. Most documentation was expressed as textual rather than graphical and they were not updated frequently. Moreover, most of the textual documentation was not architecture related. Instead, it was related to installation or configuration of the system. Since the evolution of documentation was not explored in this study, it is no possible to make any conclusions regarding how up-to-date was the documentation with respect to each release. Based on existing knowledge and the frequency that documentation was updated, it would not be surprising if at certain stages of development the documentation was outdated with respect to the source code.

Most of the graphical documentation was expressed as UML diagrams. Such diagrams were presented mostly as .xmi, .uml, or .png formats. This result is consistent with the results obtained by Hebig et al.[31] in which they reported that from all the UML diagrams they explored, 44.9% were expressed as .uml, 29.9% as .png, and 3.4% as .xmi. Keeping documentation up-to-date is not an easy task and it can become more tedious and time consuming if not done appropriately. Providing a UML diagram as image file without providing the source used for generating it, creates difficulties at the moment of modification. Instead of having to modify an existing diagram, developers would have to create a new one for each modification. For models generated manually, projects should not only contain the model as an image but also the source file used for generating such image. Providing the source file used for generating UML diagrams accelerates the modification process and makes the task less time consuming and tedious. Additionally, developers should integrate tools that automatically generate documentation so that the documentation is updated with the updates in the source code.

Our results indicated that the most commonly used type of diagrams were class diagrams followed by sequence diagrams. These results are consistent with the finding of Dobing at al.[21] which also mentioned that class diagrams and sequence diagrams are among the top three types of diagrams developers use. Not all class diagrams contained detailed information but all sequence diagrams did contain detailed information.

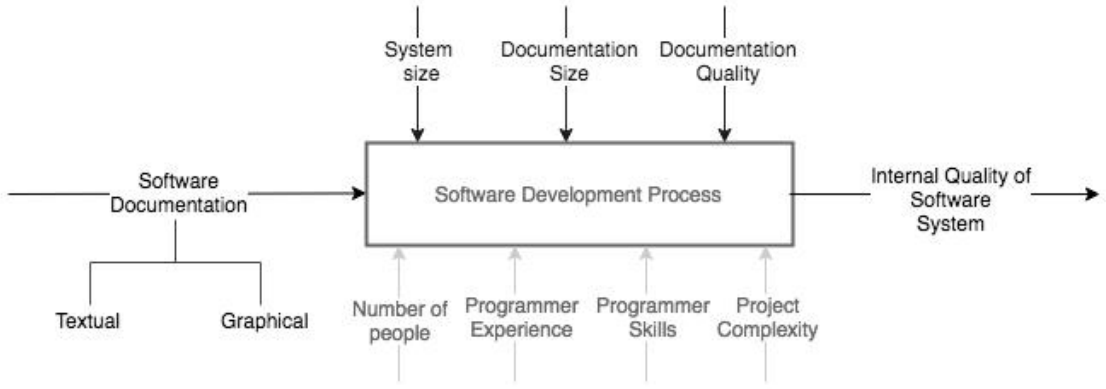
Regarding textual documentation, it was surprising that a large number of projects did not contain SAD documentation. For the project who did have SAD documentation, the process of understanding models and the overall architecture of the system was faster. The success of an open source software project depends on how large the community of contributors is [24]. There are many factors that affect how attractive an open source software is and the ability to easily modify the system is one of those. The lack of SAD documentation may create difficulties for new contributors to understand the parts of the system and thus successfully contribute to the project.

5.3 Documentation Quality

As specified in the *Results* section, the average quality of UML documentation was 77.5 out of 100, which could be considered high. Moreover, the average quality of SAD documentation was 69.4 out of 100, which can also be considered high. However, when exploring the average quality of overall documentation, the results was 45 out of 100, which is low; how is it possible? The reason is that when calculating the average quality of UML documentation, only the projects with UML were taken into account; i.e., fourteen projects. The same process was taken when calculating average quality of SAD documentation; i.e., four projects. However, when calculating the average quality of overall documentation, both scores were added and every single project was taken into account. As a result of a high number of projects lacking SAD documentation, the overall documentation quality of each project resulted in a low value.

When analyzing models for attributes such as *Level of Detail*, *Understandability*, and *Layout Quality*, the scoring scale was from 1 to 3. On the other hand, the scoring scale for *Reverse Engineering* was 0 to 1 and projects who used automation for generating UML diagrams received a value of 1. The reason for rewarding automatically generated models is based on the research by Forward et al.[10]. The results obtained in such study indicate that software professional value technologies that enable automation of documentation because it aids in documentation maintenance. However, on average, UML diagrams were generated manually. One reason for the lack-of-use of tools that automate documentation creation and update could be the lack of reliable tools for automating such process. If it is the case that there is a lack of tools that automate the creation and maintenance of documentation, then it means that there is an area for developers and researchers to explore. That is, the creation of tools that automate documentation creation and maintenance.

5.4 Correlation Analysis



Recall that **Figure 2.1**, shown above, displays the various factors that affect the development process and have an influence on the internal quality of the resulting

software. Since CBO is the only CK metrics taken into account in the correlation analysis, the results of such analysis suggest that from all the factors shown in **Figure 2.1**, documentation size and system size have the strongest positive correlation with the average CBO of the system.

A strong positive correlation between documentation size and average CBO indicates that as one variable increases, so does the other one. Although the correlation analysis does not provide any indication of causation, there are possible explanations for this pattern. One explanation is that as the size and complexity increases, so does the coupling; therefore, developers increase the amount of documentation with the expectation that it would help them during the maintenance phase or training newcomers. Small and non-complex systems may not require as much documentation as a system with more complexity and larger size. Another way of interpreting such results is that increasing the documentation of a system produces an increase in average CBO. However, it is unlikely that just adding more documentation files increases average CBO because the quality of the documentation and its content do play a role in the resulting value of average CBO.

The role of documentation quality in influencing average CBO is supported by the results indicating a strong positive correlation between documentation quality and average CBO. Just as before, this result can be interpreted that as the average CBO of a system increases, developers tend to create documentation of higher quality. A reason for this phenomenon is that the complexity of a system can be better expressed and understood if high-quality and well-written documentation is available. The opposite scenario is unlikely because just increasing the quality of a documentation file, without changing the overall architecture, does not lead to an effect on internal quality of a system.

It is natural that as the size of a system increases, more modules are created, and thus, more dependencies tend to emerge between the modules. Therefore, it was no surprise to observe results indicating a strong positive correlation between system size and average CBO. Additionally, such correlation emerged regardless if system size was expressed in terms of LOC or number of modules. Bakar et al. [30] obtained similar results indicating a correlation between system size and coupling. One way of interpreting such results is that as the size of the system increases, so does the average CBO of the system. As mentioned before, this is a likely scenario due to the natural tendency of an increase in coupling as the system size increases.

This result is a call-to-action to motivate developers to create architectures that better handle dependencies between modules. Well-architectured systems are imperative in order to avoid a rapid increase in average CBO as the size increases.

6
Limitations

The success of this study depends not only the method, process, and techniques used to solve the research questions but only depends on being aware of the limitation of such methods, processes, and techniques used. The purpose of this section is to present the limitations encountered in this study, the effect they could have on the study, and the approach taken to address them.

6.1 Tool Limitations

Although Analizo is a powerful and useful tool for calculating internal quality metrics of software, such tool has certain limitations that constrained the study. Those limitations, and the way each influenced the study are the following:

1. **Only source code written in C, C++, or Java could be analyzed:** As a result of this limitation, the population of projects that could be analyzed was reduced.
2. **Analysis time could last many hours:** In the worst case scenario, Analizo could take up to ten hours to finish calculating the metrics for a single project. Some factors that affected calculation time include the number of releases that needed to be analyzed and the size of each release. The latter having the most impact. As a result of this limitation as well as the time constraint for finalizing this study, it was not possible to analyze more projects.
3. **Analizo was unable to analyze every C, C++, or Java project:** In many occasions, Analizo would abruptly stop calculating metrics as a result of faults in its source code. In the best case scenario, the abrupt stop would occur early in the calculation process. In the worst case scenario, it would stop calculating after many hours of analysis. Both cases, and especially the latter, resulted in wasted time. As a result of this limitation, a significant amount of time was wasted during the stage of calculating the internal quality metrics; thus, affecting the total number of projects analyzed in the study. Whenever Analizo was unable to analyze a given project then such project was disregarded from the analysis and another was taken.
4. **Tool Reliability:** Since the results of the study are based on the metrics Analizo generated, it means the validity of the results presented in this study depend to some extent on Analizo's reliability.

As a result of the limitations from Analizo, in many cases, projects to be analyzed were conveniently selected. That is, in many cases, a given project was not

necessarily randomly selected; instead, it was selected because it worked for Analizo. The result of conveniently selecting projects may have introduced selection bias into the study.

6.2 Sample Limitations

As mentioned in the *Methods* section, the set of projects analyzed in this study is a subset of the set of projects generated by Hebig et al. in their study [31]. Although this approach facilitated the process of selecting projects for this study, it also limited the randomness of the sample. The reasons for the lack of randomness are the following:

1. The set of projects by Hebig et al. only considered open source software from GitHub platform; nevertheless, GitHub is not the only platform with open source software.
2. The set of projects by Hebig at al. was the result of studying a percentage of all GitHub projects. However, it is not clear how randomly selected were the projects in the percentage they analyzed.

Finally, the sample size of the study is small compared to the population size. Therefore, it is not clear how generalizable are the results of this study.

6.3 Method Limitations

Recall that the method used for identifying documentation files was based on analyzing the message of commits. Although this method served its purpose, it is not the most effective way of identifying documentation because it generated many false positive. That is, the algorithm considered files to be documentation-related when in fact they were not. Although researchers tried their best to identify and filter out the false positives, it is possible that a number of them were not identified.

The opposite problem is also true for the method used. That is, if the commit message did not contain the keywords of interest it is possible that documentation files were not considered. Correcting this problem is harder since it would involve manually checking every files associated with every commit.

There is extensive research suggesting metrics for measuring internal quality of software. However, not all of those metrics have been validated empirically or theoretically. Given the extensive research supporting the validity of CK metrics [1], [3], [9], [14], the authors of this study decided that using them was the best available method for measuring internal quality of software. After analyzing the results, it was concluded that the CK metrics, specifically CBO, was a good indicator of coupling level in the system.

If the study would have been designed using other metrics, then it is possible that other results regarding internal quality would have been obtained.

6.4 Documentation Analysis

When grading overall documentation quality only UML models quality and SAD documentation quality was taken into consideration. Although this provided a good understanding of documentation quality, it might not have been enough for grading documentation quality. The reason being that many projects consider source code comments as documentation. Therefore, since source code comments were not analyzed, it is possible that certain project could have had a higher overall documentation quality score if their source code comments would have been considered in the analysis.

Recall that a set of attributes were taken into consideration when measuring the quality of SAD documents and UML models. Although such measures are not bad for measuring quality, it is not clear or proven that they are sufficient for measuring the quality of SAD documents and models, respectively. If it is the case that the selected attributes are not sufficient for measuring quality, then it means that we are providing a partial representation of documentation quality.

Finally, the process of measuring quality was a subjective process based on the opinion and experience of the researchers. As a results, other researchers could generate different quality scores for the same documentation.

7
Conclusion

The development and growth of open source software depend on the voluntary contribution of developers. Therefore, it is important to ensure that open source software has a certain level of internal quality to attract more contributors. Factors such as system size, system complexity, documentation size, documentation quality, programmers skills, and programmer experience affect the development process; and thus, have an effect on the internal quality of software. Understanding the effect that those factors have on the development process is the first step towards the improvement of the internal quality of software.

The purpose of this study is to explore the correlation between documentation quality and internal quality of software. To accomplish such goal, other aspects of software development were explored such as: how the internal quality of software changed over time, how frequently documentation is updated, the type of documentation found in projects, and the quality of documentation.

In this study, the Chidamber & Kemerer (CK) metrics were used to measure the internal quality of 17 software projects. However, focus was given to the metric *Coupling between Objects* or *CBO*. Additionally, Analizo was the tool used for calculating all the CK metrics. Moreover, to normalize the results obtained from Analizo, a metric called *CBO Average* was used. This metric resulted from dividing *CBO Sum* by the total number of modules in the system. Finally, a Spearman's correlation analysis was performed to find the correlation coefficient between the variables of interest.

The results of this study suggest there is a positive correlation between documentation quality and *CBO Average*. Additionally, documentation size and system size also have a positive correlation with *CBO Average*. The results of this study also indicate that it is not typical for projects to include SAD files as part of their documentation, UML class diagrams and sequence diagrams are the most common type of UML diagrams used, all the CK metrics tend to have a similar direction of change from one release to another, and the UML diagrams provided tend to have high quality.

For practitioners, the results show the importance of keeping the source code as slim as possible. In other words, not adding unnecessary modules or duplicated code. As shown in the results, as the size of the system increases, so does CBO Average. It also demonstrates to them that writing more and better documentation is a natural, and in many cases necessary, processes as the size and complexity of the system increases. On the other hand to researchers, the results open opportunity for future work. As the results demonstrate, there is no perfect correlation between the variables, therefore, there are other factors that take place in the correlation

analysis. Uncovering those factors as well as the influence they play is critical to further understand what affects internal quality of software and how to improve it.

In summary, as the size and complexity of the project increases so does the amount of documentation written and its quality. Possible explanations of this phenomenon is that the increase in size and complexity of the system motivates developers to write more and better documentation.

In order to ensure the validity of this study, it was essential to identify the limitations that could negatively affect the results. Some limitations encountered in this study include the small sample size analyzed, the reliability of Analizo tool, the accuracy of the algorithm used for detecting documentation files, and high-subjectivity of the method used for grading documentation quality.

7.0.1 Future Work

Although the research questions of this study were answered, there are other questions and possible directions that could be explored as a result of this study.

One direction for future work is to collect more data in order to study a larger sample and investigate if the results obtained in this study remain consistent with a larger sample. Additionally, the data obtain could be analyzed using a Bayesian data analysis and investigate if the results obtained from such analysis are consistent with the results obtained in this study.

An alternative direction is to further explore the results obtain and solve unanswered questions such as what is the role that developer's experience and skills have on the internal quality of software? What are the reasons for projects not including as much SAD documentation as other types of documentation? What are the correlation between documentation quality, documentation size, and system size and the other CK metrics?

Finally, the result of the Spearman's analysis done in this study provides information about correlation and not causation. Therefore, in order to better understand the correlations obtained, a future area of work is to study their causation.

Bibliography

[1] W. Li and S. Henry, "Object-oriented metrics that predict maintainability", *Journal of systems and software*, vol. 23, no. 2, pp. 111–122, 1993.

[2] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994, ISSN: 00985589. DOI: 10.1109/32.295895. arXiv: 1011.1669.

[3] V. R. Basili, W. L. Melo, and L. C. Briand, "A validation of object-oriented design metrics as qualityindicators", *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, 1996, ISSN: 0098-5589. DOI: 10.1109/32.544352. [Online]. Available: http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=544352.

[4] F. Brito e Abreu and W. Melo, "Evaluating the impact of object-oriented design on software quality", *Proceedings of the 3rd International Software Metrics Symposium*, pp. 90–99, 1996. DOI: 10.1109/METRIC.1996.492446. [Online]. Available: http://ieeexplore.ieee.org/document/492446/.

[5] B. Kitchenham and S. L. Pfleeger, "Software quality: the elusive target [special issues section]", *IEEE software*, vol. 13, no. 1, pp. 12–21, 1996.

[6] P. Nesi and T. Querci, "Effort estimation and prediction of object-oriented systems", *Journal of Systems and Software*, vol. 42, no. 1, pp. 89–102, 1998, ISSN: 01641212. DOI: 10.1016/S0164-1212(97)10021-8. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0164121297100218.

[7] J. Bansiya and C. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment", *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 4–17, 2002.

[8] V. Cortellessa, H. Singh, and B. Cukic, "Early reliability assessment of UML based software models", *Proceedings of the third international workshop on Software and performance - WOSP '02*, p. 302, 2002. DOI: 10.1145/584369.584415. [Online]. Available: http://portal.acm.org/citation.cfm?doid=584369.584415.

[9] K. El-Emam, "Object-oriented metrics: A review of theory and practice", in *Advances in software engineering*, Springer, 2002, pp. 23–50.

[10] A. Forward and T. C. Lethbridge, "The relevance of software documentation, tools and technologies: a survey", *DocEng 02 Proceedings of the 2002 ACM symposium on Document engineering*, pp. 26–33, 2002. DOI: 10.1145/585058.585065. [Online]. Available: http://portal.acm.org/citation.cfm?doid=585058.585065.

[11] S. H. Kan, *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[12] M. Genero, M. Piattini, E. Manso, G. Cantone, I. C. Society, and S. Ieee Computer, "Building UML class diagram maintainability prediction models based on early metrics", *Ninth International Software Metrics Symposium, Proceedings*, pp. 263–275, 2003, ISSN: 1530-1435. DOI: 10.1109/METRIC.2003.1232473.

[13] T. C. Lethbridge, J. Singer, A. Forward, and D. Consulting, "How Software EngineeUse Documentation : The State of the Practice Documentation :" *IEEE Computer Society*, pp. 35–39, 2003.

[14] R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects", *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, 2003, ISSN: 00985589. DOI: 10.1109/TSE.2003.1191795.

[15] S. Tilley and S. Huang, "A Qualitative Assessment of the Efficacy of UML Diagrams as a Form of Graphical Documentation in Aiding Program Understanding", *Proceedings of the 21st annual international conference on Documentation - SIGDOC '03*, p. 184, 2003. DOI: 10.1145/944905.944908. [Online]. Available: http://portal.acm.org/citation.cfm?doid=944868.944908.

[16] L. H. Etzkorn, S. E. Gholston, J. L. Fortune, C. E. Stein, D. Utley, P. A. Farrington, and G. W. Cox, "A comparison of cohesion metrics for object-oriented systems", *Information and Software Technology*, vol. 46, no. 10, pp. 677–687, 2004, ISSN: 09505849. DOI: 10.1016/j.infsof.2003.12.002.

[17] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, "A study of the documentation essential to software maintenance", *Proceedings of the 23rd annual international conference on Design of communication documenting & designing for pervasive information - SIGDOC '05*, p. 68, 2005. DOI: 10.1145/1085313.1085331. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1085313.1085331.

[18] L. Yu, S. R. Schach, and K. Chen, "Measuring the maintainability of open-source software", *2005 International Symposium on Empirical Software Engineering, ISESE 2005*, vol. 00, no. c, pp. 297–303, 2005. DOI: 10.1109/ISESE.2005.1541838.

[19] E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche, "The impact of UML documentation on software maintenance: An experimental evaluation", *IEEE Transactions on Software Engineering*, vol. 32, no. 6, pp. 365–381, 2006, ISSN: 00985589. DOI: 10.1109/TSE.2006.59.

[20] J. Bitzer and P. J. H. Schröder, "The economics of open source software development: An introduction", *The Economics of Open Source Software Development*, pp. 1–13, 2006. DOI: 10.1016/B978-044452769-1/50001-9.

[21] B. Dobing and J. Parsons, "How UML Is used", vol. 49, no. 5, pp. 109–114, 2006.

[22] R. A. Ghosh, "Study on the Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU Draft final report Subcontractors : Draft final report", p. 4, 2006. [Online]. Available: http://stuermer.ch/blog/documents/FLOSSImpactOnEU.pdf%7B%5C%%7D5Cnhttps://ec.europa.eu/

digital-single-market/en/news/call-tender-economic-impact-open-source-software-innovation-and-competitiveness-ict-sector.

[23] A. MacCormack, J. Rusnak, and C. Baldwin, "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code", *Management Science*, vol. 52, no. 7, pp. 1015–1030, 2006, ISSN: 0025-1909. DOI: 10.1287/mnsc.1060.0552. [Online]. Available: http://dx.doi.org/10.1287/mnsc.1060.0552.

[24] M. Aberdour, "Achieving quality in open source software", *IEEE software*, no. September, pp. 58–64, 2007, ISSN: 18727727. DOI: 10.1016/j.ejrad.2010.05.004. [Online]. Available: http://www.computer.org/portal/web/csdl/doi/10.1109/MS.2007.2.

[25] M. Cherubini, G. Venolia, R. Deline, and A. J. Ko, "Let ' s Go to the Whiteboard: How and Why Software Developers Use Drawings", *CHI 2007 Proceedings*, pp. 557–566, 2007, ISSN: 10629432. DOI: 10.1145/1240624.1240714.

[26] C. Lange, *Assessing and Improving the Quality of Modeling: a Series of Empirical Studies about the {UML}*. 2007, ISBN: 9789038611075. DOI: 10.6100/IR629604.

[27] R. Lincke, J. Lundberg, and W. Löwe, "Comparing software metrics tools", *Proceedings of the 2008 international symposium on Software testing and analysis - ISSTA '08*, p. 131, 2008. DOI: 10.1145/1390630.1390648. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1390630.1390648.

[28] A. Nugroho and M. R. Chaudron, "A survey into the rigor of UML use and its perceived impact on quality and productivity", *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '08*, p. 90, 2008. DOI: 10.1145/1414004.1414020. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1414004.1414020.

[29] A. Terceiro, J. Costa, and J. Miranda, "Analizo: an extensible multi-language source code analysis and visualization toolkit", *Brazilian Conference on Software: Theory and Practice (CBSoft)–Tools*, pp. 1–6, 2010. [Online]. Available: http://www.researchgate.net/publication/228414689%7B%5C_%7DAnalizo%7B%5C_%7Dan%7B%5C_%7DExtensible%7B%5C_%7DMulti-Language%7B%5C_%7DSource%7B%5C_%7DCode%7B%5C_%7DAnalysis%7B%5C_%7Dand%7B%5C_%7DVisualization%7B%5C_%7DToolkit/file/72e7e52569416d51d8.pdf.

[30] N. S. A. A. Bakar and N. Arsat, "Investigating the factors that influence the quality of open source systems", *Information and Communication Technology for The Muslim World (ICT4M), 2014 The 5th International Conference on*, pp. 1–6, 2014. DOI: 10.1109/ICT4M.2014.7020589.

[31] R. Hebig, T. H. Quang, M. R. V. Chaudron, G. Robles, and M. A. Fernandez, "The quest for open source projects that use UML", *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems - MODELS '16*, pp. 173–183, 2016. DOI: 10.1145/2976767.2976778. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2976767.2976778.

A

Types of File Extensions

The following table provides a list of file extensions and the category such files belong to. It is important to mention that this is not an exhaustive list of possible extensions. Additionally, it is possible that certain extensions fall into multiple categories.

Source Code	Text Documentation	Graphical Documentation
.java	.rst	.puml
.c	.md	.png
.cpp	.me	.uml
.h	.txt	.svg
.sql	.doc	.umlcd
.html	.docx	.eps
.xml	.odt	.uxf
.dist	.docbook	.jpg
.service		.zargo
.css		.xmi
.sh		.nomnoml
.xhtml		.pdf

B

Guidelines for Measuring Documentation Quality

The purpose of this appendix is to describe the guidelines for measuring quality of graphical and textual documentation. The guidelines are based on experience of the researchers of this study. It is not a standard way of measuring documentation quality.

B.1 Measuring Quality of Textual Documentation

Attributes	Scoring System	Description
Understandability	High(3), Medium(2), Low(1)	A measure of how easy it is to understand the content of the documentation.
Correspondence to models	High(3), Medium(2), Low(1)	A measure of the degree to which the explanation and model provided relate to each other.[1]
Level of Detail	High(3), Medium(2), Low(1)	A measure of how much information the document provides.

B.2 Measuring Quality of UML Models

Attributes	Scoring System	Description
Level of Detail	High(3), Medium(2), Low(1)	A measure of how much information a diagram provides.
Understandability	High(3), Medium(2), Low(1)	A measure of how easy it is to understand the content of the diagram.
Layout	High(3), Medium(2), Low(1)	A measure of how well organized the parts of the diagram are.
Reversed Engineered	Yes(1), No(0)	Was the diagram built manually or generated automatically?

[1] In all projects it was possible to trace textual to graphical documentation.

B.3 Calculating Total Quality

The scoring system shown above is based on points. Therefore, a score of *High* represents 3 points, *Medium* represents 2 points, and *Low* 1 point. In the case of the *Reverse Engineered* attribute, a value of *yes* represents 1 point and 0 otherwise. It was decided to use this three-point scale system because it makes it easier to evaluate. A scale with greater granularity (a scale from 1 to 10, for example) would have made the grading more complicated.

To calculate the total quality of textual documentation, the score for each attribute was added up. Therefore, the maximum quality score textual documentation could obtain was 9, while 3 is the lowest value. The same approach was taken to calculate the total quality score of graphical documentation. However, the maximum score that graphical documentation could obtain was 10, while 3 is the lowest value. Finally, to calculate the total quality score of documentation, the overall quality score of textual and graphical documentation were added. Therefore, the maximum quality score the documentation of a project could obtain was 19, while 6 is the smallest value.

Since the quality scale for textual and graphical documentation differ, to compare quality between them, the following normalization was done:

- If the total score for textual documentation of a project was 5 out of 9 then 5 was divided by 9 and multiplied by 100% giving a result of 56%. The same procedure was done with the quality score of graphical documentation. The result is that both are on the same scale based on 100 and it was possible to make comparisons as needed.

C

Change in CK Metrics for Each Project

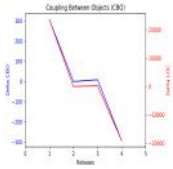
C. Change in CK Metrics for Each Project

CK Metric for each Project

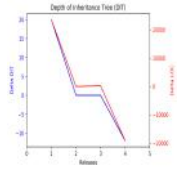
Legend

- Each of the graphs below illustrates how the value of a given CK metric *changes* from one release to another. To put the CK metrics into perspective, each graph also contains the *change* in LOC from one release to another.

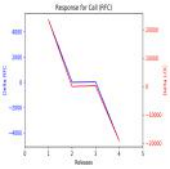
a958861e



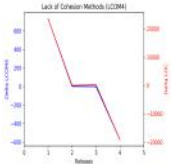
Average Δ CBO = 5.5



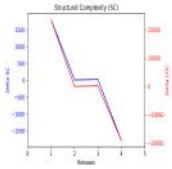
Average Δ DIT = 2.0



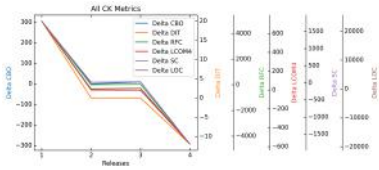
Average Δ RFC = 93.75



Average Δ LCOM4 = 39.5

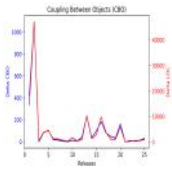


Average Δ SC = 12.75

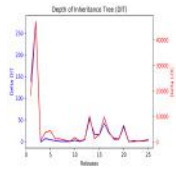


All CK Metric of the project plotted in a single graph.

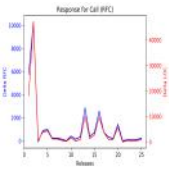
9c699c33



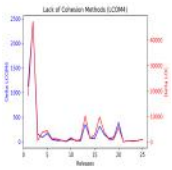
Average Δ CBO = 106.64



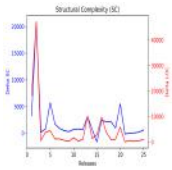
Average Δ DIT = 25.36



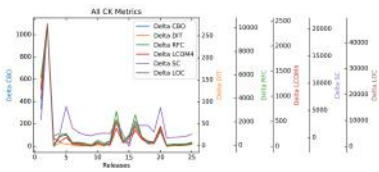
Average Δ RFC = 1172.44



Average Δ LCOM4 = 233.76

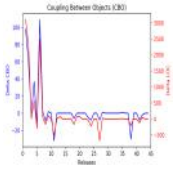


Average Δ SC = 2099.52

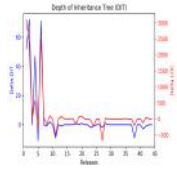


All CK Metric of the project plotted in a single graph.

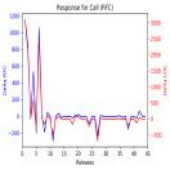
1fa7c454



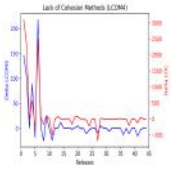
Average Δ CBO = 4.27



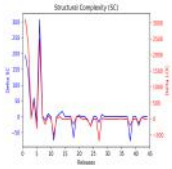
Average Δ DIT = 4.7



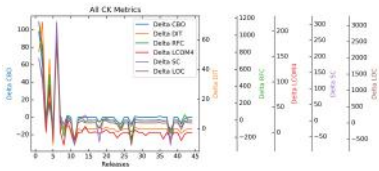
Average Δ RFC = 56.82



Average Δ LCOM4 = 10.57

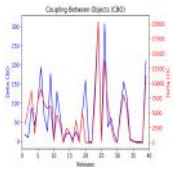


Average Δ SC = 9.43

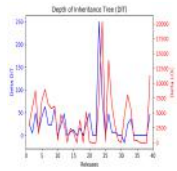


All CK Metric of the project plotted in a single graph.

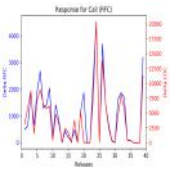
b50b5a5f



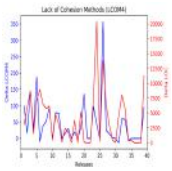
Average Δ CBO = 69.08



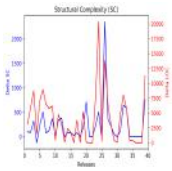
Average Δ DIT = 24.85



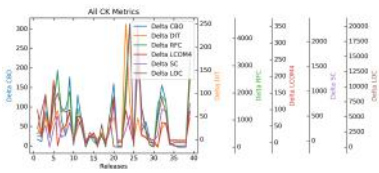
Average Δ RFC = 1044.51



Average Δ LCOM4 = 45.49

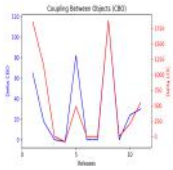


Average Δ SC = 241.1

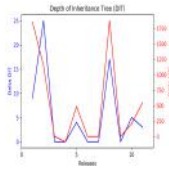


All CK Metric of the project plotted in a single graph.

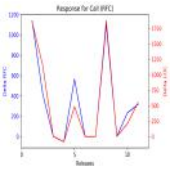
b8419a88



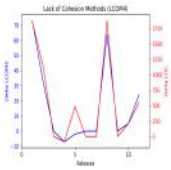
Average Δ CBO = 30.27



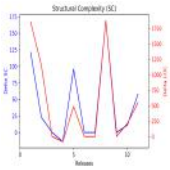
Average Δ DIT = 5.73



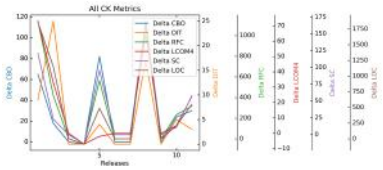
Average Δ RFC = 338.27



Average Δ LCOM4 = 17.27

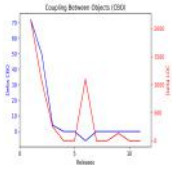


Average Δ SC = 42.18

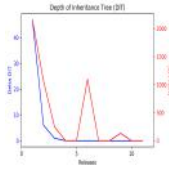


All CK Metric of the project plotted in a single graph.

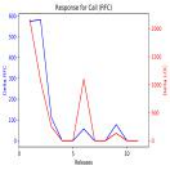
9cd7e19b



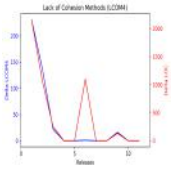
Average Δ CBO = 10.91



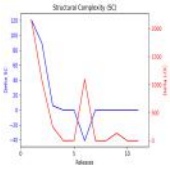
Average Δ DIT = 4.91



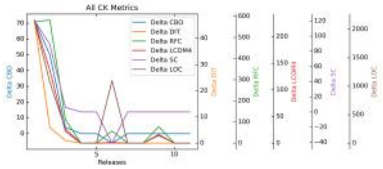
Average Δ RFC = 127.45



Average Δ LCOM4 = 37.0

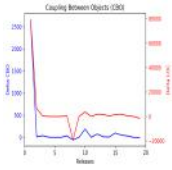


Average Δ SC = 15.91

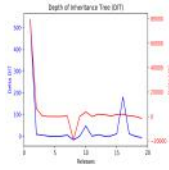


All CK Metric of the project plotted in a single graph.

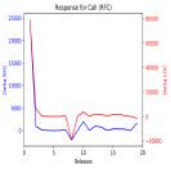
7f90d484



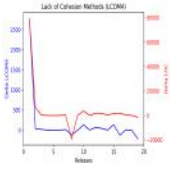
Average Δ CBO = 165.0



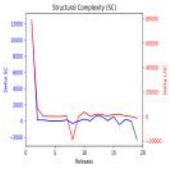
Average Δ DIT = 41.53



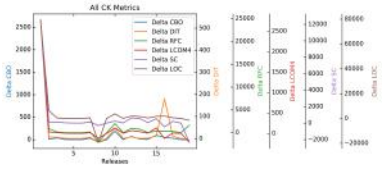
Average Δ RFC = 1589.79



Average Δ LCOM4 = 146.63

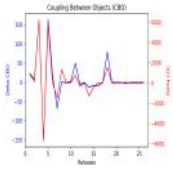


Average Δ SC = 635.63

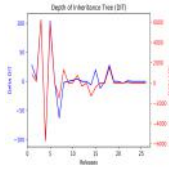


All CK Metric of the project plotted in a single graph.

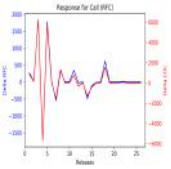
6a43bae4



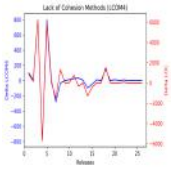
Average Δ CBO = 9.42



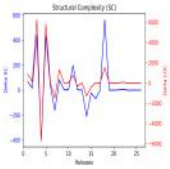
Average Δ DIT = 4.19



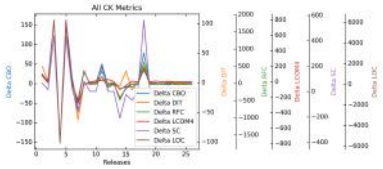
Average Δ RFC = 90.58



Average Δ LCOM4 = 27.54

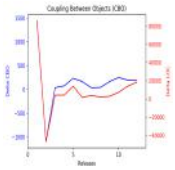
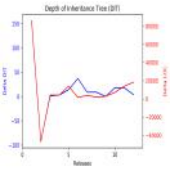
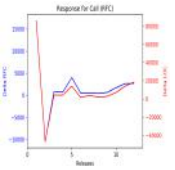
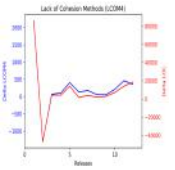
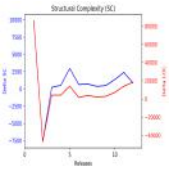
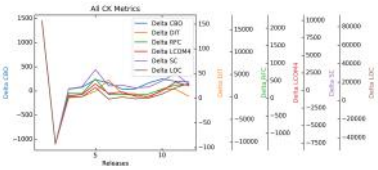


Average Δ SC = 38.04



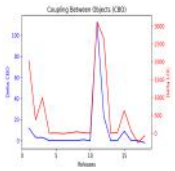
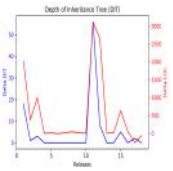
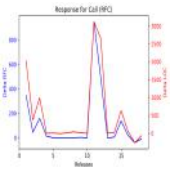
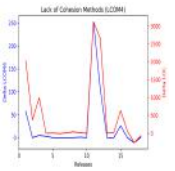
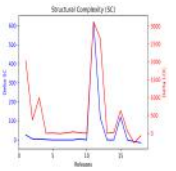
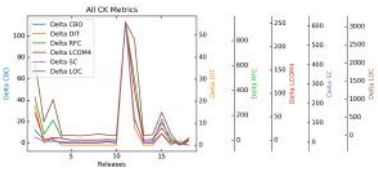
All CK Metric of the project plotted in a single graph.

000199d7

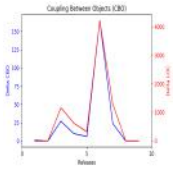
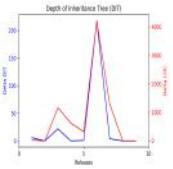
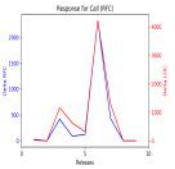
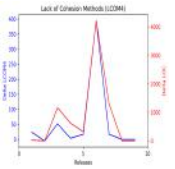
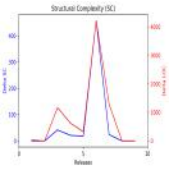
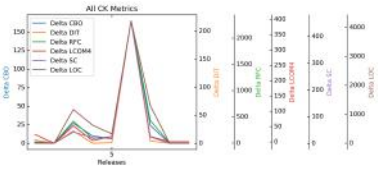
Average Δ CBO = 143.92 | Average Δ DIT = 14.67 | Average Δ RFC = 1764.83 | Average Δ LCOM4 = 243.33 | Average Δ SC = 1062.08 | All CK Metric of the project plotted in a single graph.

3cb5f97e

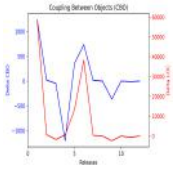
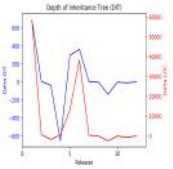
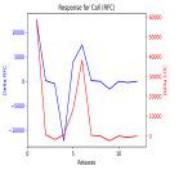
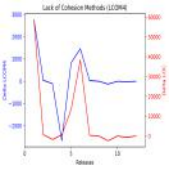
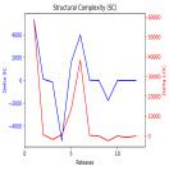
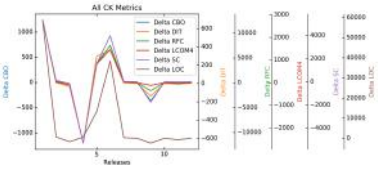
Average Δ CBO = 9.0 | Average Δ DIT = 5.17 | Average Δ RFC = 118.06 | Average Δ LCOM4 = 24.56 | Average Δ SC = 48.56 | All CK Metric of the project plotted in a single graph.

31ce9276

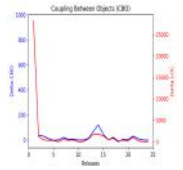
Average Δ CBO = 25.78 | Average Δ DIT = 27.89 | Average Δ RFC = 377.23 | Average Δ LCOM4 = 56.0 | Average Δ SC = 62.56 | All CK Metric of the project plotted in a single graph.

300ef577

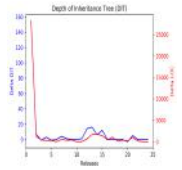







Average Δ CBO = 61.75 | Average Δ DIT = 45.08 | Average Δ RFC = 825.58 | Average Δ LCOM4 = 184.17 | Average Δ SC = 317.83 | All CK Metric of the project plotted in a single graph.

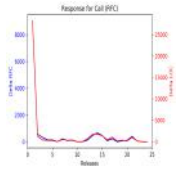
4f582a30



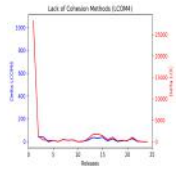
Average Δ CBO = 57.0



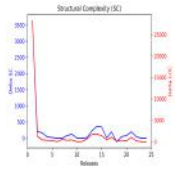
Average Δ DIT = 9.12



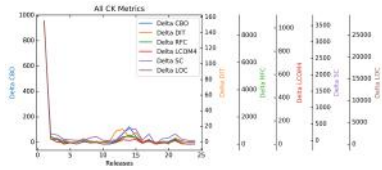
Average Δ RFC = 552.5



Average Δ LCOM4 = 58.29

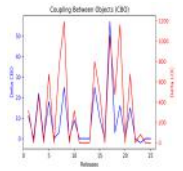


Average Δ SC = 236.08

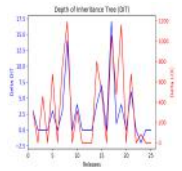


All CK Metric of the project plotted in a single graph.

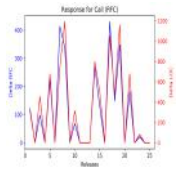
41c9999f



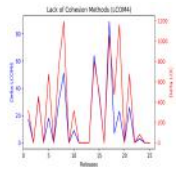
Average Δ CBO = 8.48



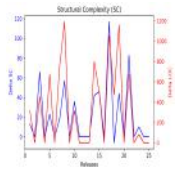
Average Δ DIT = 2.56



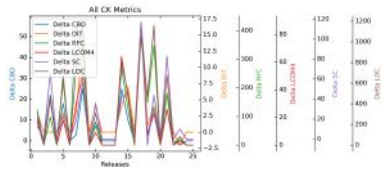
Average Δ RFC = 111.52



Average Δ LCOM4 = 16.2

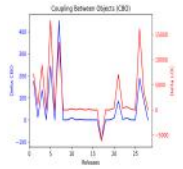


Average Δ SC = 21.96

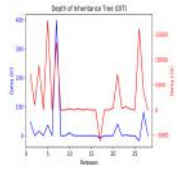


All CK Metric of the project plotted in a single graph.

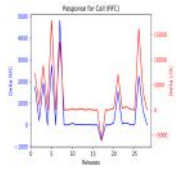
02e83ccb



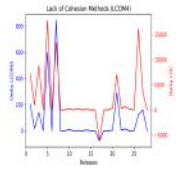
Average Δ CBO = 47.07



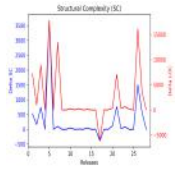
Average Δ DIT = 21.64



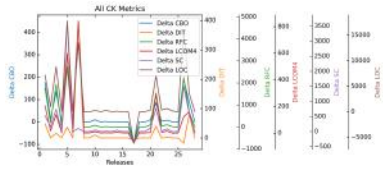
Average Δ RFC = 553.21



Average Δ LCOM4 = 83.18

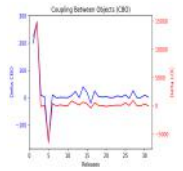


Average Δ SC = 284.89

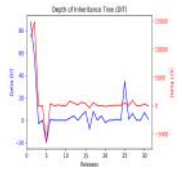


All CK Metric of the project plotted in a single graph.

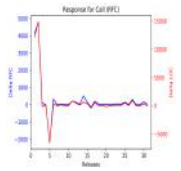
7753c60f



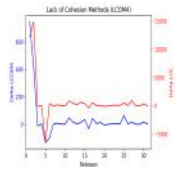
Average Δ CBO = 16.16



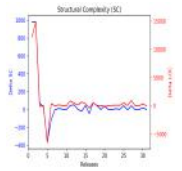
Average Δ DIT = 6.23



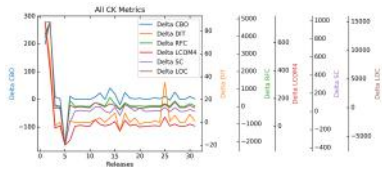
Average Δ RFC = 296.87



Average Δ LCOM4 = 37.26

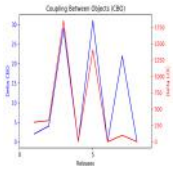


Average Δ SC = 63.35

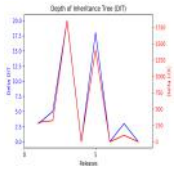


All CK Metric of the project plotted in a single graph.

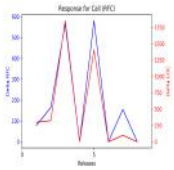
24dd418f



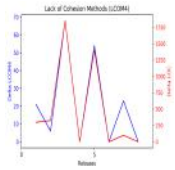
Average Δ CBO = 11.0



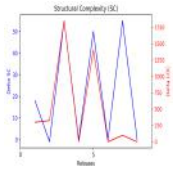
Average Δ DIT = 6.12



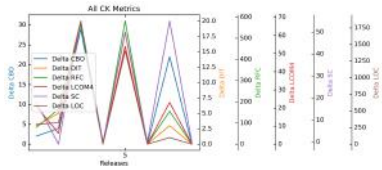
Average Δ RFC = 193.0



Average Δ LCOM4 = 21.5



Average Δ SC = 22.12



All CK Metric of the project plotted in a single graph.

D

CBO Based Analysis for Each Project

CBO Based Analysis

Legend

- **Graph A**

  Illustrates the CBO value for each release.
- **Graph B**

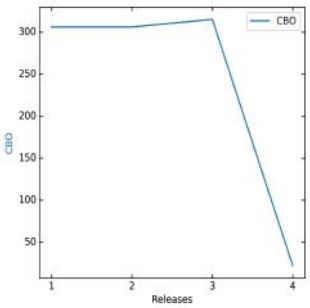
  Illustrates the LOC value for each release.
- **Graph C**

  Illustrates the CBO and LOC value for each release.
- **Graph D**

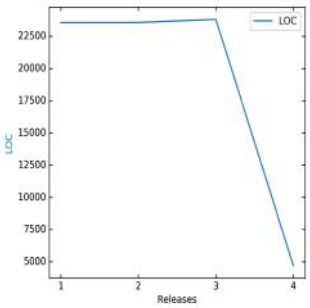
  Illustrates the CBO/LOC value for each release.

a958861e

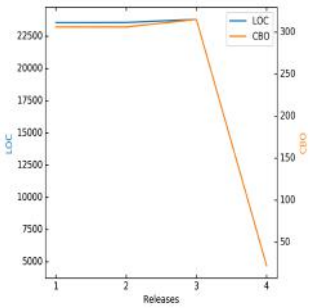
Graph A



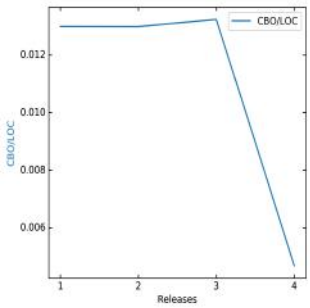
Graph B



Graph C

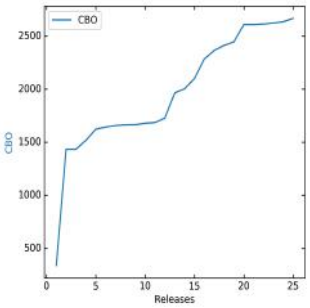


Graph D

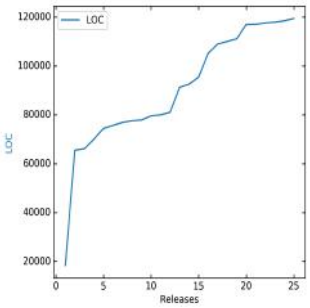


9c699c33

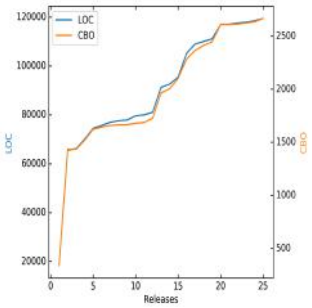
Graph A



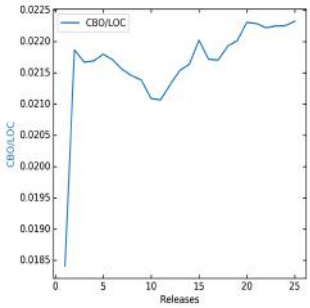
Graph B



Graph C



Graph D



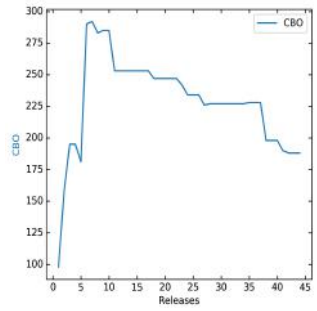
1fa7e454

Graph A

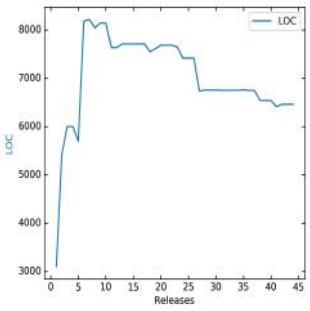
Graph B

Graph C

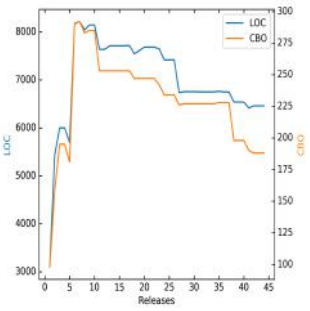
Graph D



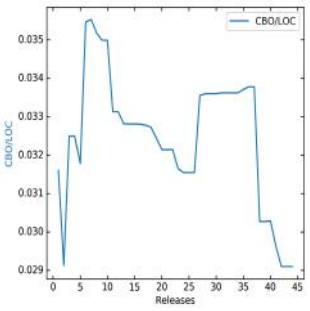
Graph A



Graph B

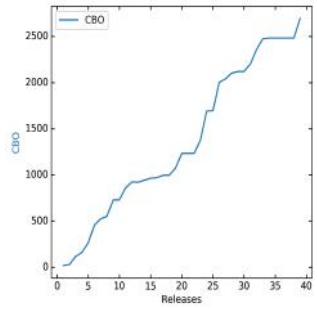


Graph C

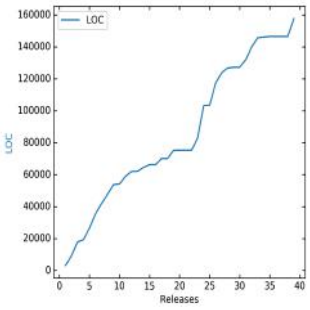


Graph D

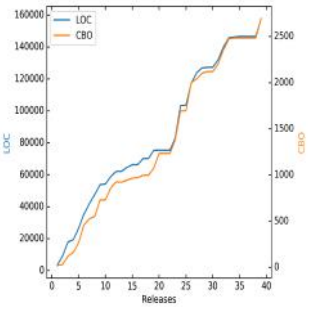
b50b5a5f



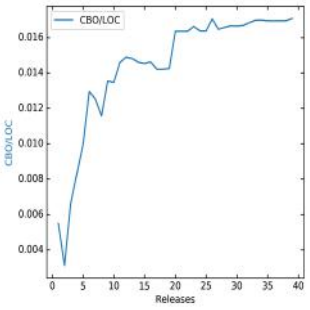
Graph A



Graph B

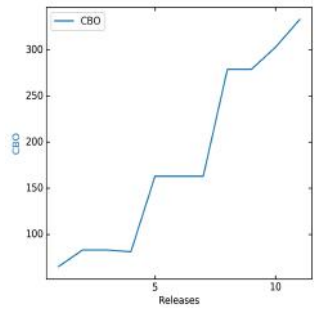


Graph C

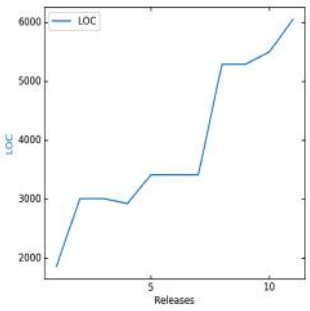


Graph D

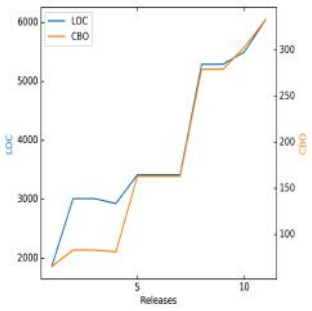
b8419a88



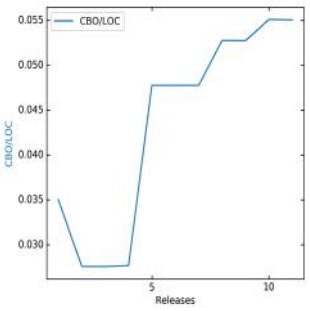
Graph A



Graph B

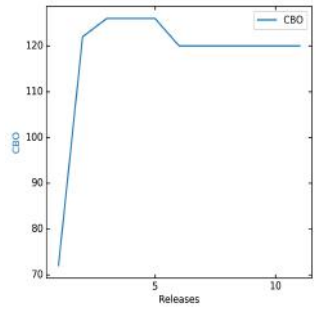


Graph C

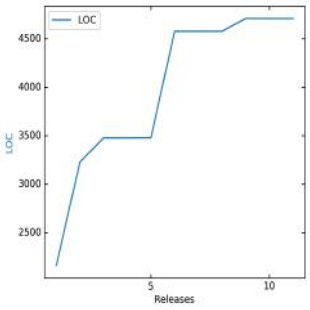


Graph D

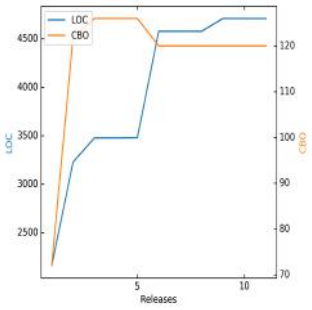
9cd7e19b



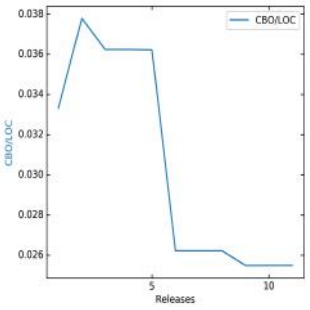
Graph A



Graph B

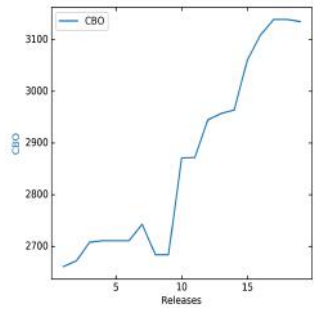


Graph C

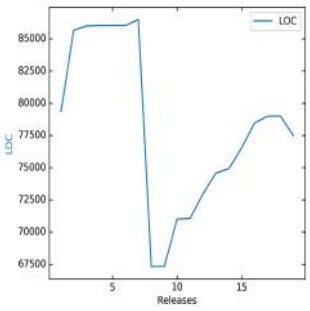


Graph D

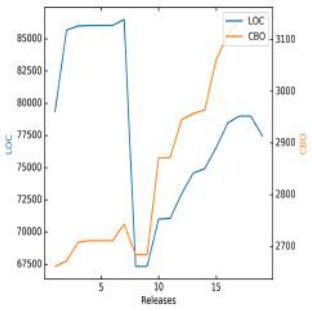
7f90d484



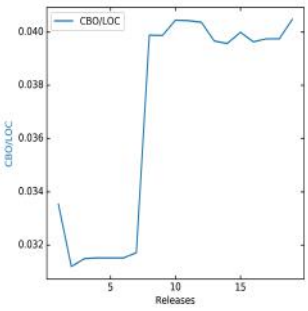
Graph A



Graph B

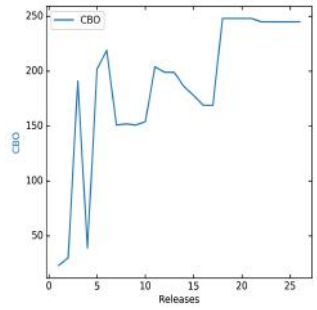


Graph C

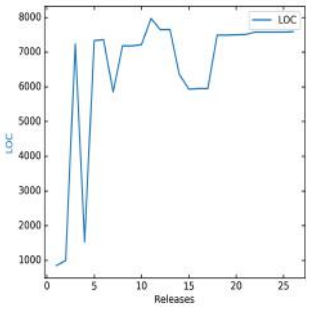


Graph D

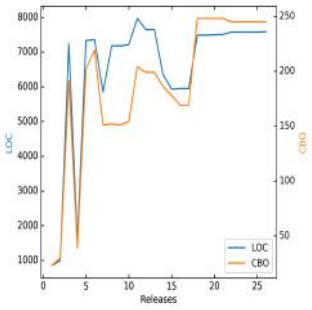
6a43bae4



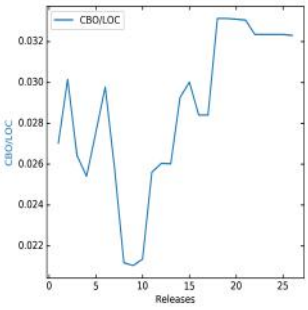
Graph A



Graph B

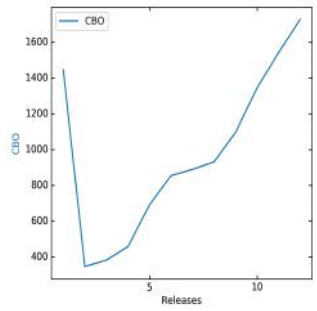


Graph C

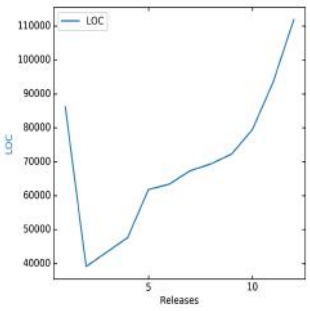


Graph D

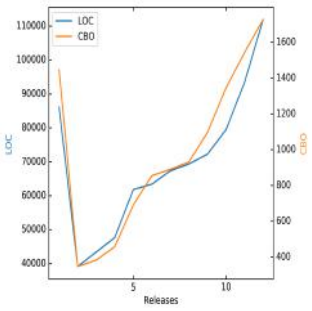
000199d7



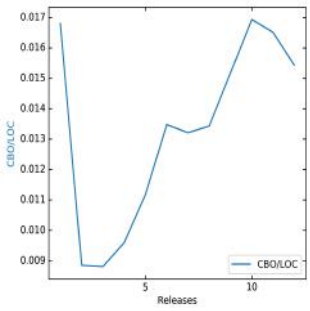
Graph A



Graph B

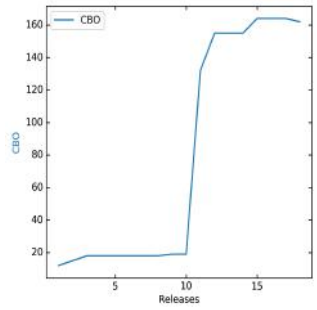


Graph C

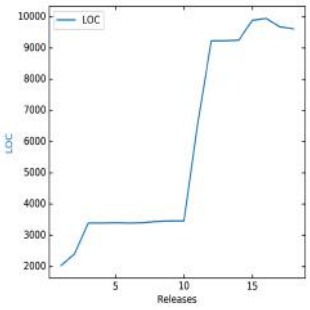


Graph D

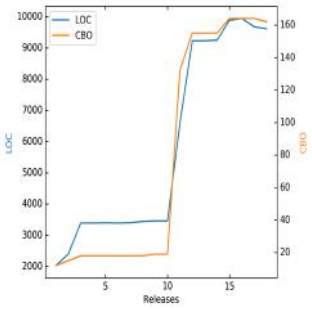
3cb5f97e



Graph A

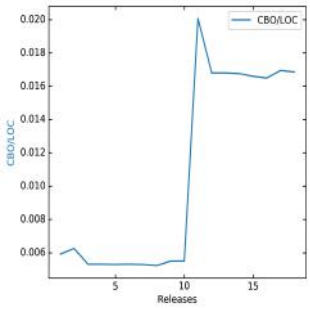


Graph B

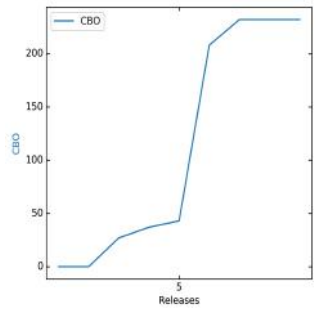


Graph C

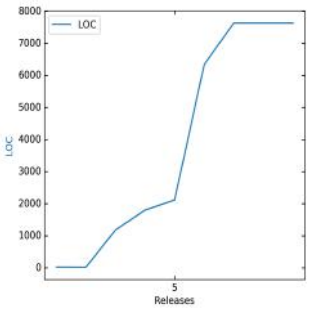
31ce9276



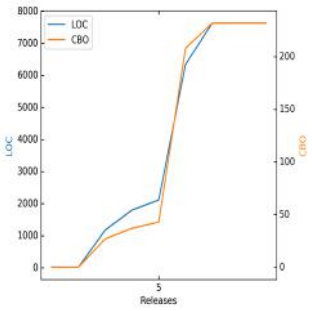
Graph D



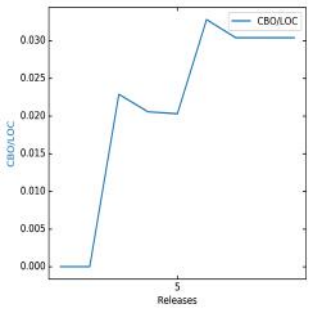
Graph A



Graph B

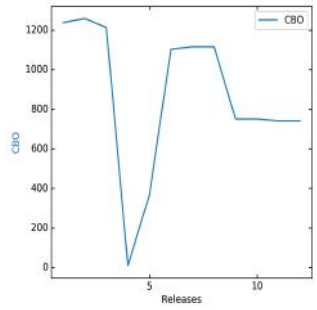


Graph C

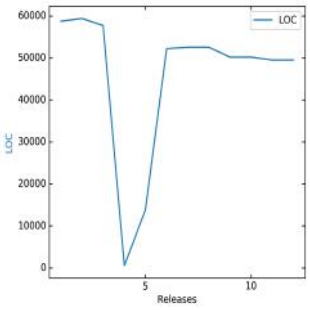


Graph D

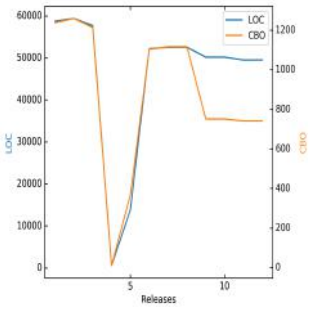
300ef577



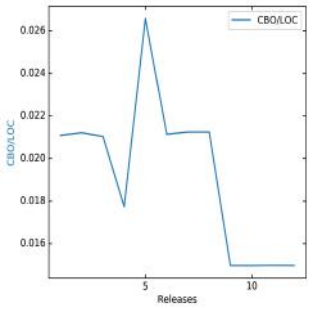
Graph A



Graph B

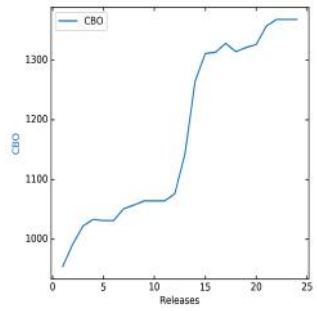


Graph C

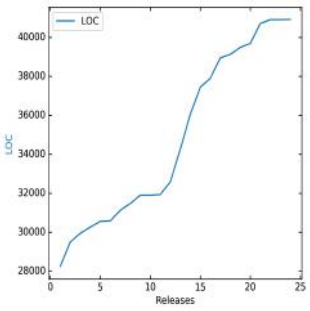


Graph D

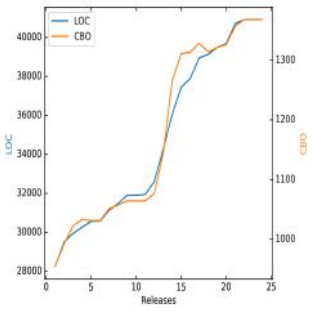
4f582a30



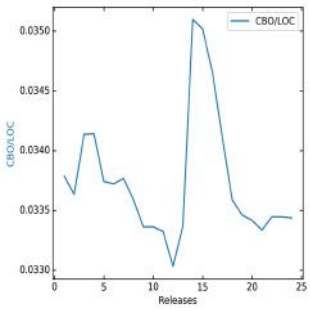
Graph A



Graph B

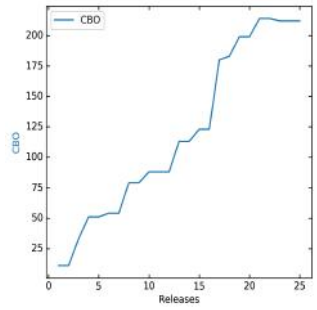


Graph C

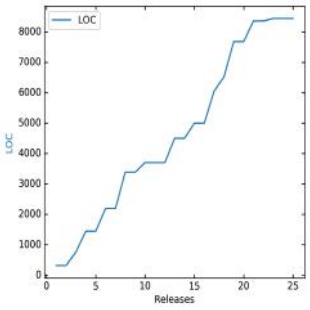


Graph D

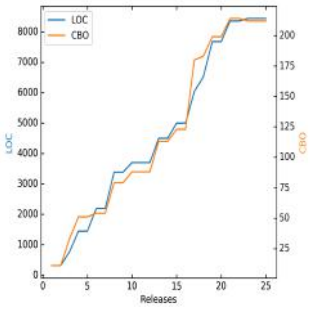
41c9999f



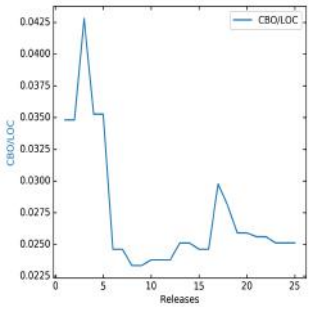
Graph A



Graph B

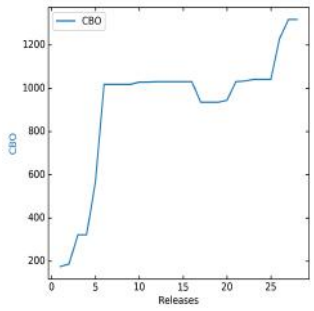


Graph C

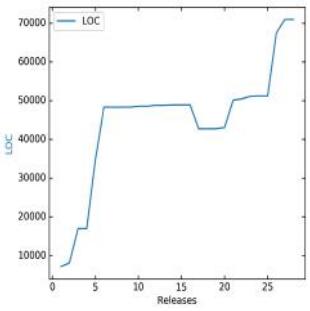


Graph D

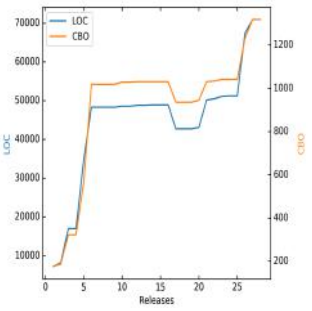
02e83ccb



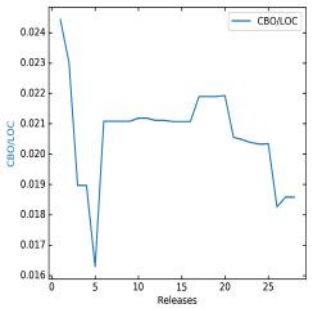
Graph A



Graph B

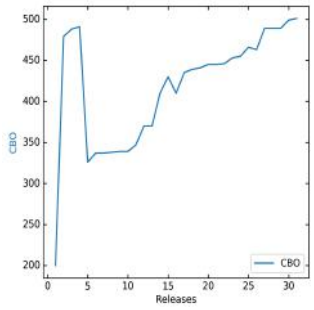


Graph C

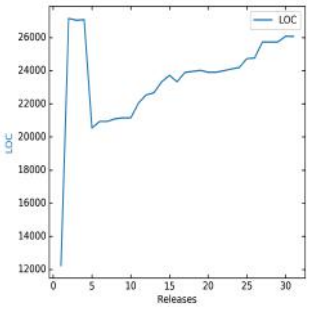


Graph D

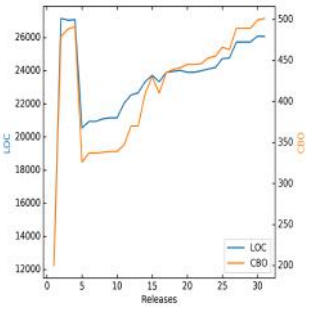
7753c60f



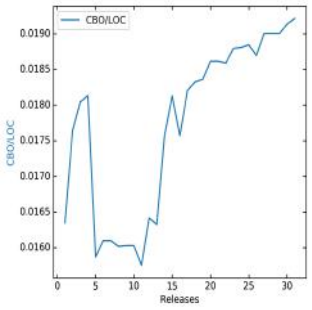
Graph A



Graph B

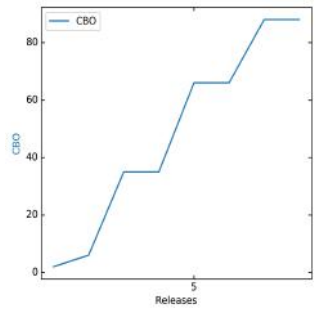


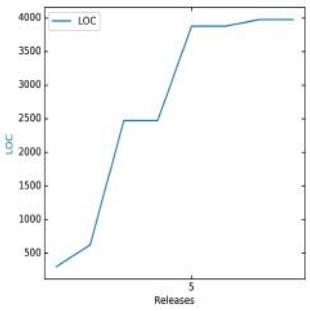
Graph C

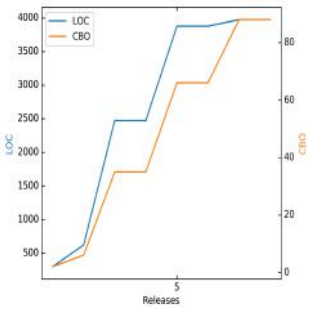


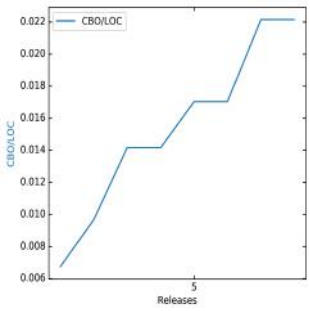
Graph D

24dd418f


CBO
Releases


LOC
Releases


LOC
CBO
Releases


CBO/LOC
Releases

E

Documentation Distribution

File Analysis

Legend

- **Graph A**

  Illustrates the total number of documentation commits in the given project and the total number of unique files associated to those commits.

- **Graph B**

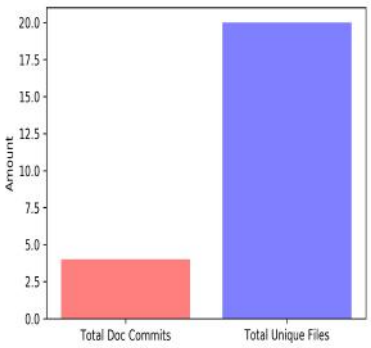
  Illustrates the distributions of the unique files in each category. For example, if Graph A indicated there are 10 unique files, then Graph B illustrates how many of those 10 files are of type *Source Code*, *Text Documentation*, and *Graphical Documentation*.

- **Graph C**

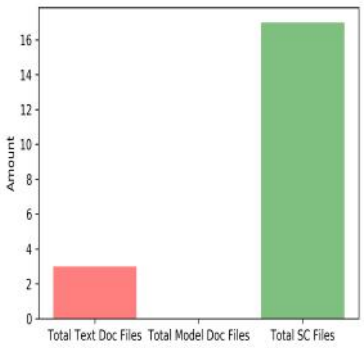
  Illustrates the average frequency that each documentation file type is updated.

a958861e

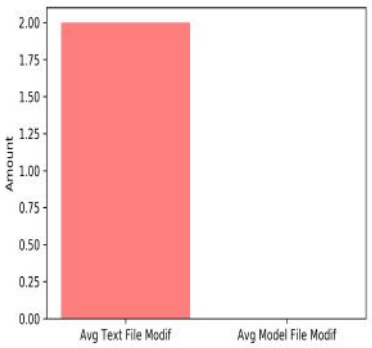
Graph A



Graph B



Graph C

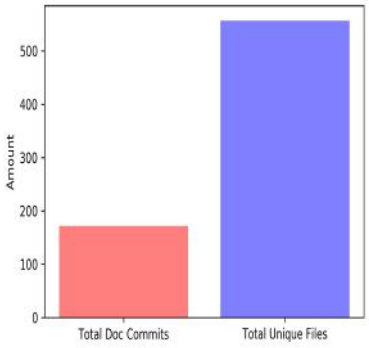


9c699c33

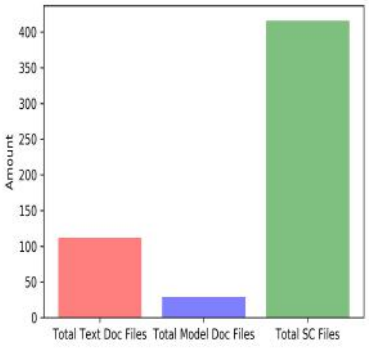
Graph A

Graph B

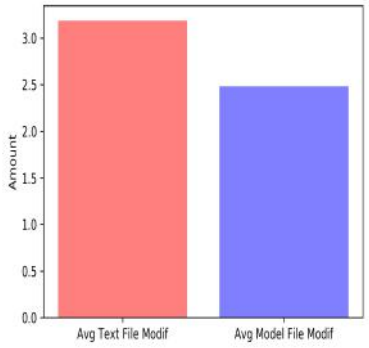
Graph C



Graph A

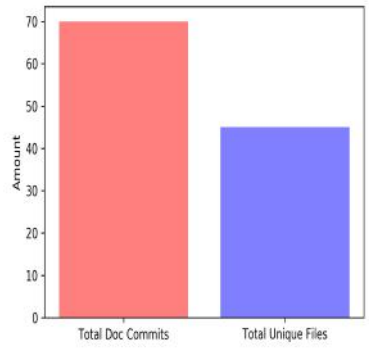


Graph B

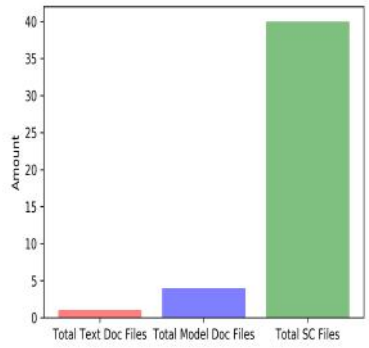


Graph C

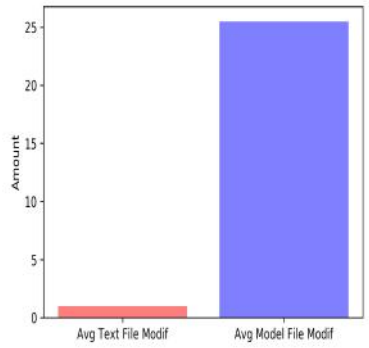
1fa7e454



Graph A

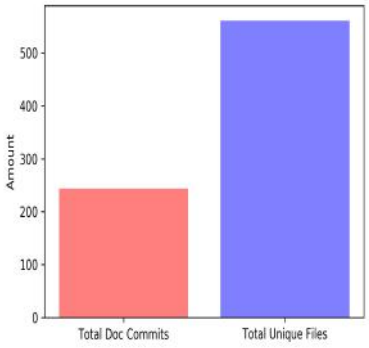


Graph B

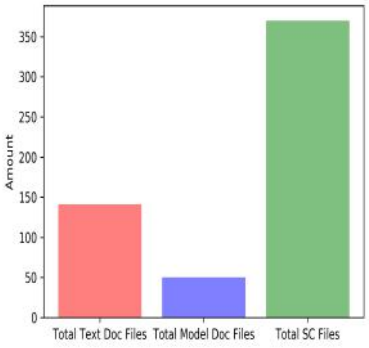


Graph C

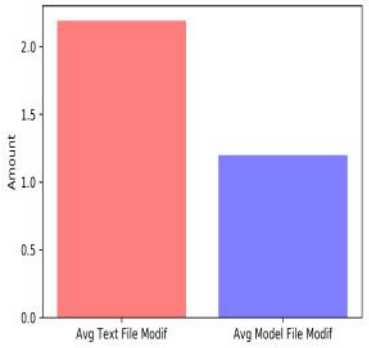
b50b5a5f



Graph A

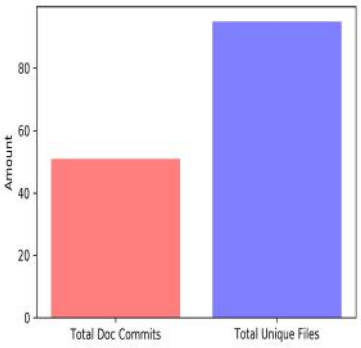


Graph B

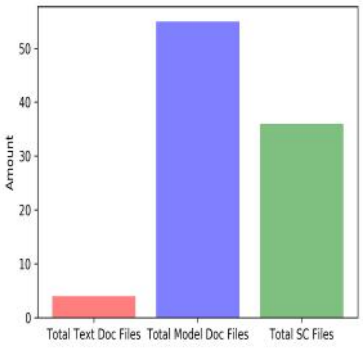


Graph C

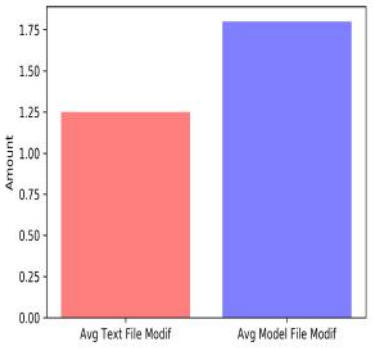
b8419a88



Graph A

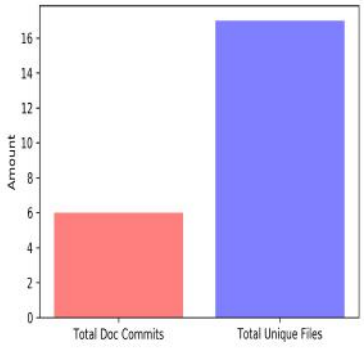


Graph B

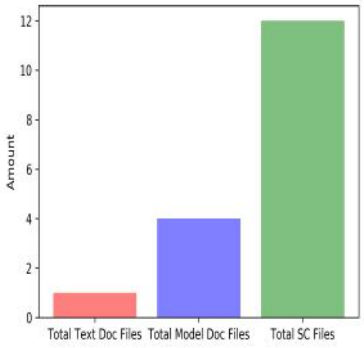


Graph C

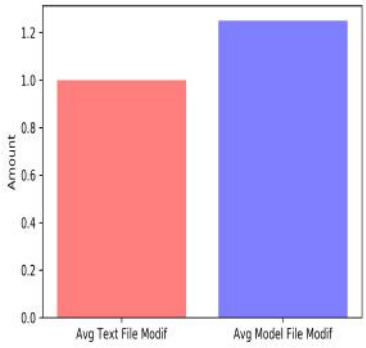
9cd7e19b



Graph A

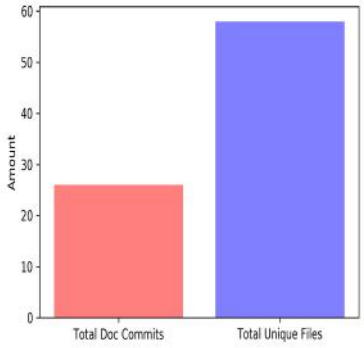


Graph B

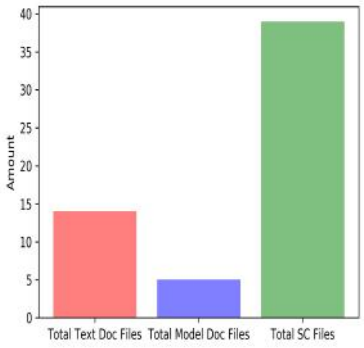


Graph C

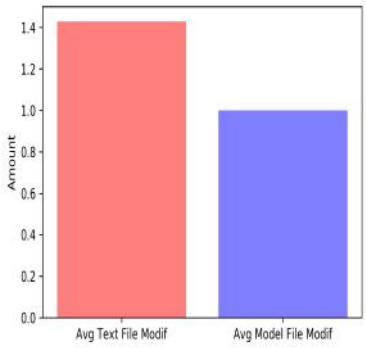
7f90d484



Graph A

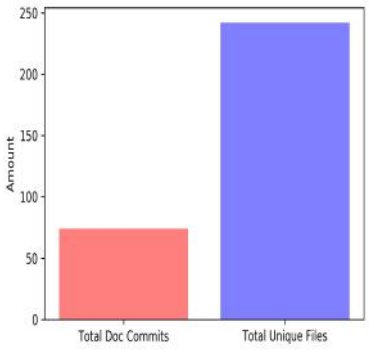


Graph B

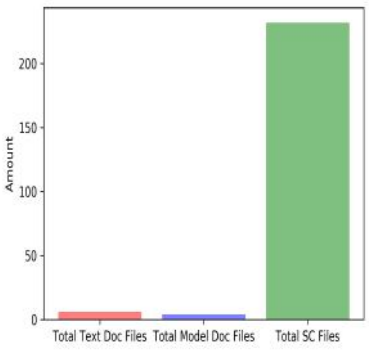


Graph C

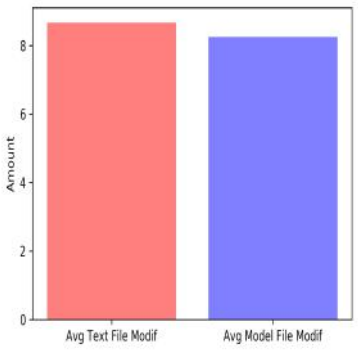
6a43bae4



Graph A

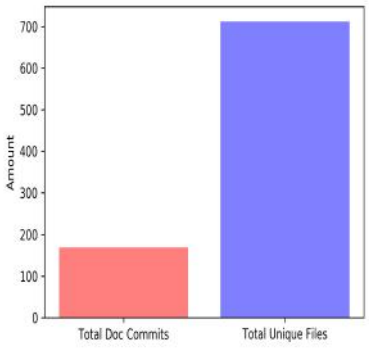


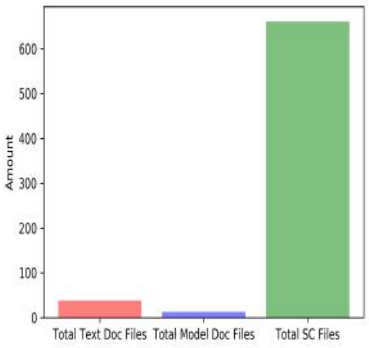
Graph B

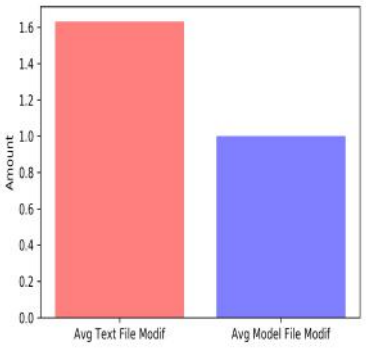


Graph C

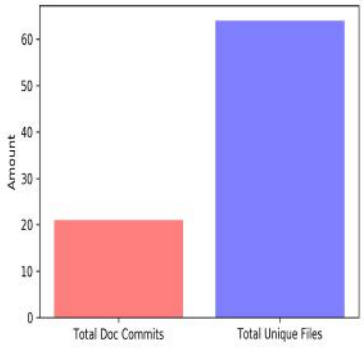
000199d7



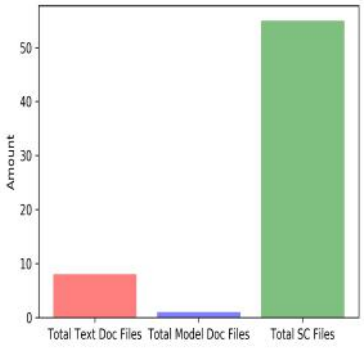




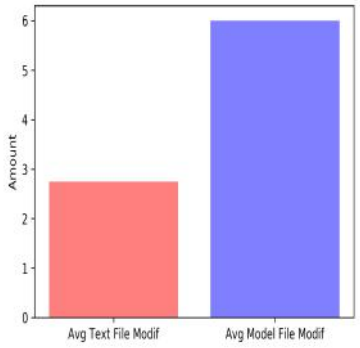
3cb5f97e **Graph A** **Graph B** **Graph C**



Graph A

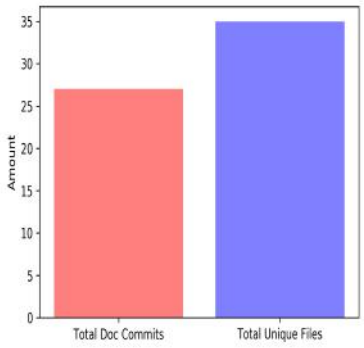


Graph B

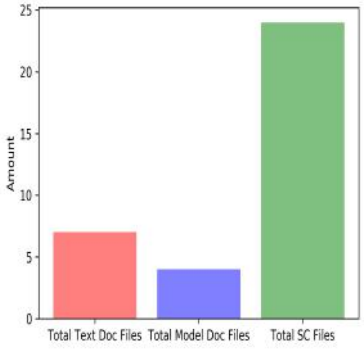


Graph C

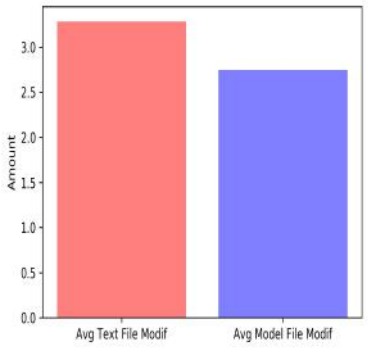
31ce9276



Graph A

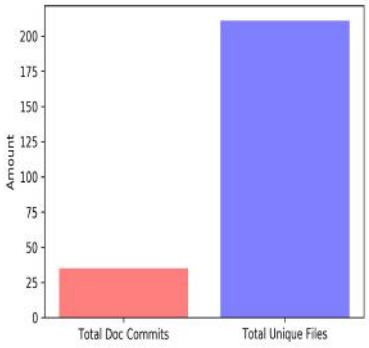


Graph B

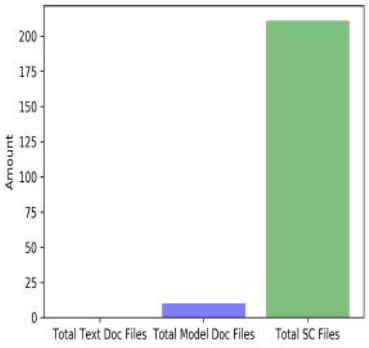


Graph C

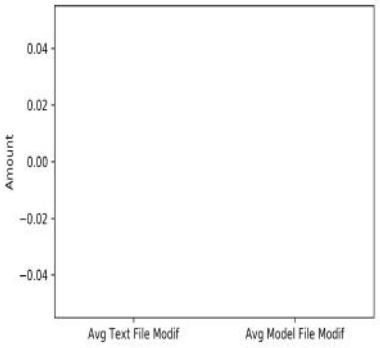
300ef577



Graph A

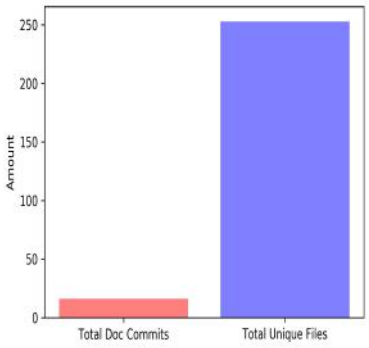


Graph B

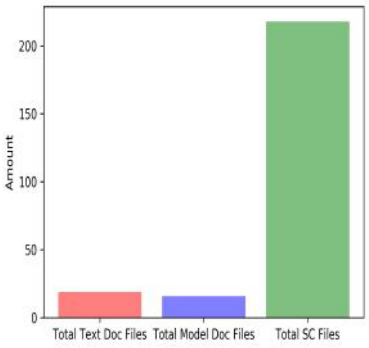


Graph C

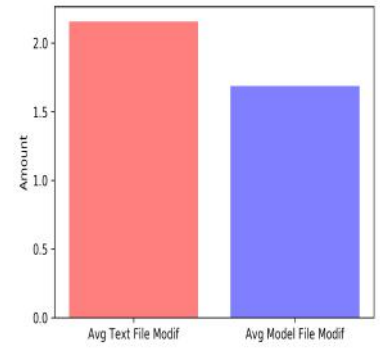
4f582a30



Graph A

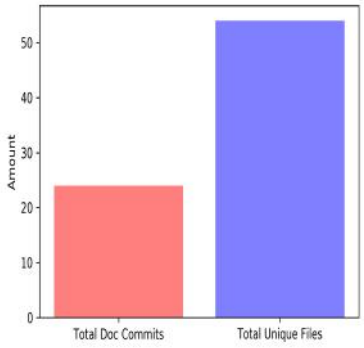


Graph B

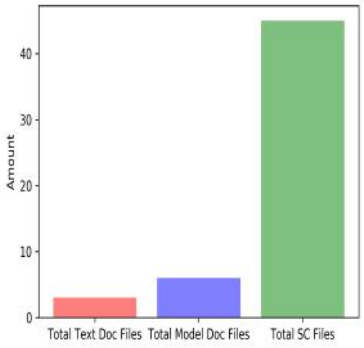


Graph C

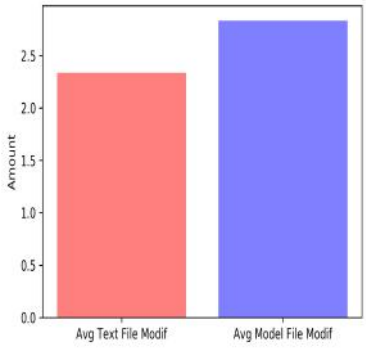
41c9999f



Graph A

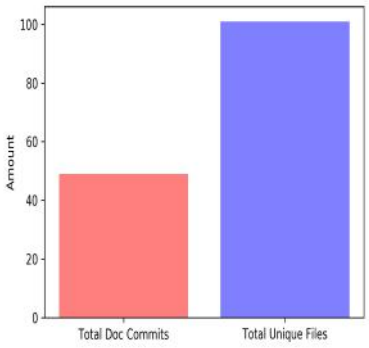


Graph B

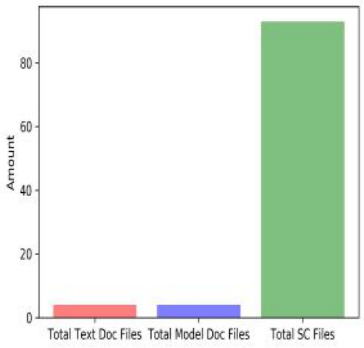


Graph C

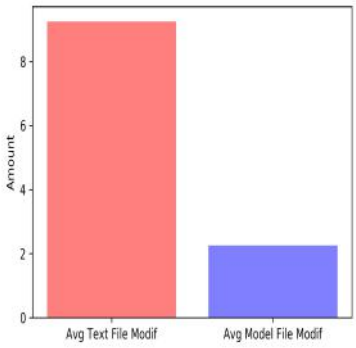
02e83ccb



Graph A

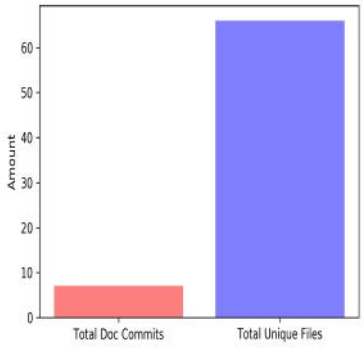


Graph B

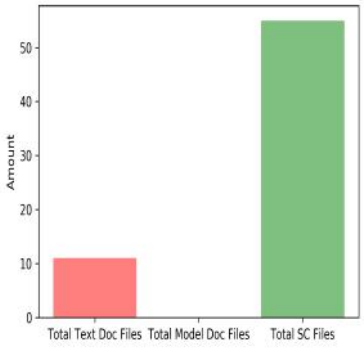


Graph C

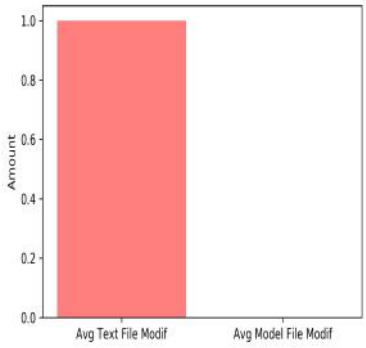
7753c60f



Graph A

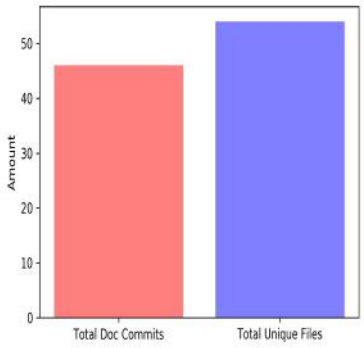


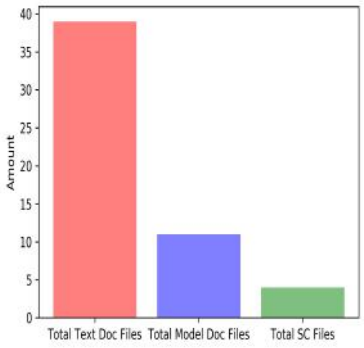
Graph B

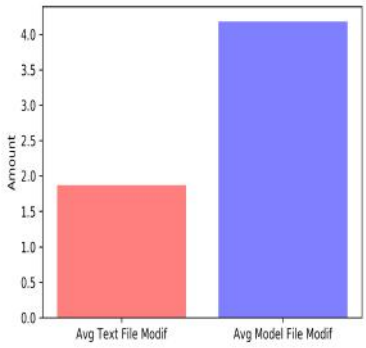


Graph C

24dd418f







Project ID	Total Doc Commits	Total Files	Total Model Doc Files	Total SC Files	Total Text Doc Files	Avg Model File Modification	Avg Text File Modification
a958861e	4	20	0	17	3	0.000	2.000
9c699c33	172	557	29	416	112	2.483	3.188
1fa7e454	70	45	4	40	1	25.500	1.000
b50b5a5f	244	561	50	370	141	1.200	2.191
b8419a88	51	95	55	36	4	1.800	1.250
9cd7e19b	6	17	4	12	1	1.250	1.000
7f90d484	26	58	5	39	14	1.000	1.429
27847def	16	253	16	218	19	1.688	2.158
6a43bae4	74	242	4	232	6	8.250	8.667
000199d7	169	712	13	661	38	1.000	1.632
3cb5f97e	21	64	1	55	8	6.000	2.750
31ce9276	27	35	4	24	7	2.750	3.286
300ef577	35	211	10	211	0	0.000	0.000
4f582a30	16	253	16	218	19	1.688	2.158
41c9999f	24	54	6	45	3	2.833	2.333
02e83ccb	49	101	4	93	4	2.250	9.250
7753c60f	7	66	0	55	11	0.000	1.000
24dd418f	46	54	11	4	39	4.182	1.872

F

Classification of Graphical Documentation

Distribution of Models

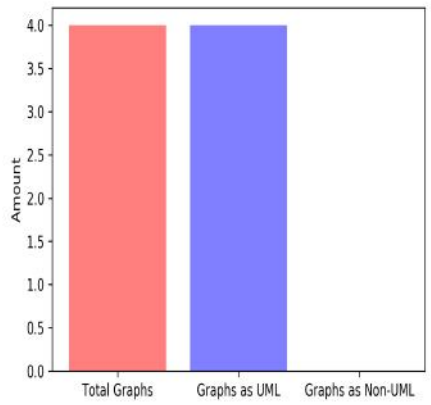
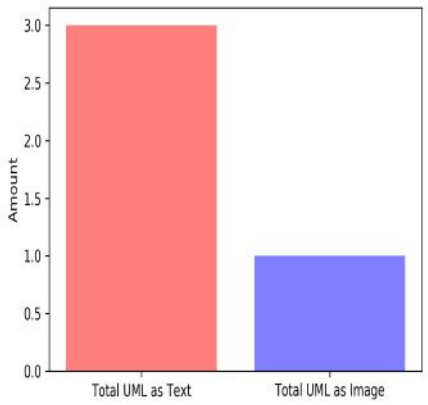
Legend

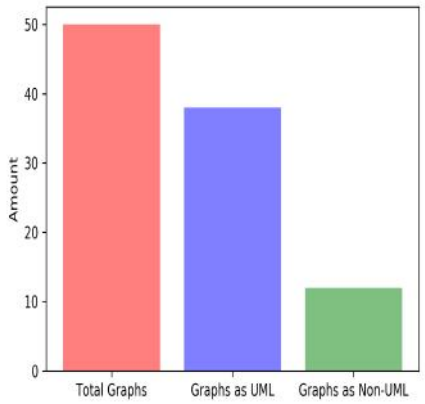
- **Graph A**

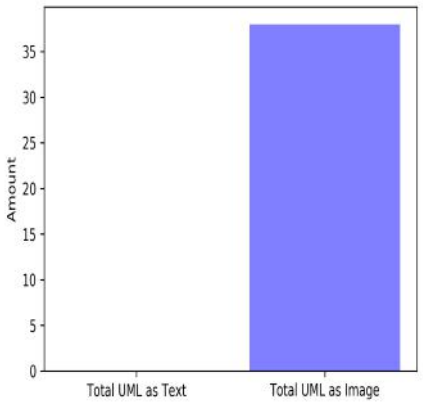
  The column *Total Graphs* illustrates the total number of graphs available in the given project. From the total number of graphs available, the columns *Graphs as UML* and *Graphs as Non-UML* indicate how many are UML and non-UML, respectively.

- **Graph B**

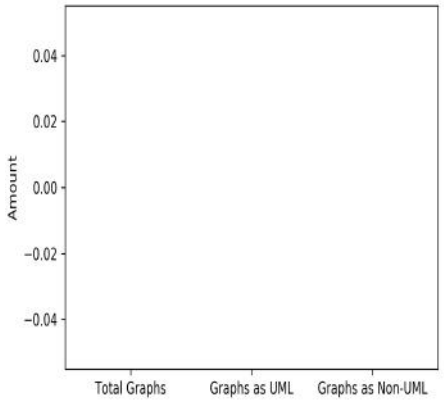
  UML diagrams can be provided in a graphical file such as png or jpg or as source code such as .uml. Therefore, from the total number of graphs that were UML, the column *Total UML as Text* and *Total UML as Image* indicate how many of those UML models were provided as text or image file, respectively.

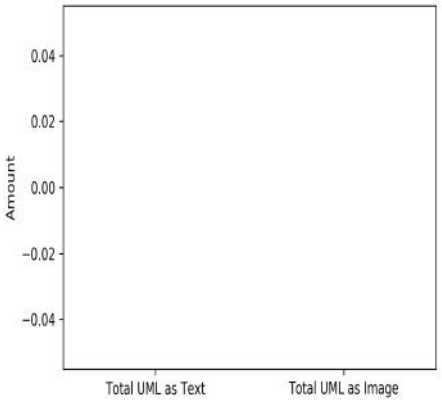
	Graph A	Graph B
1fa7e454		
b50b5a5f	**Graph A**	**Graph B**

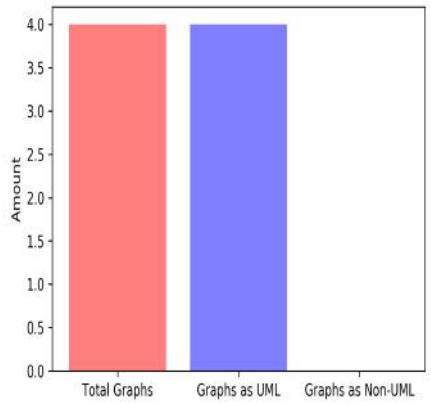
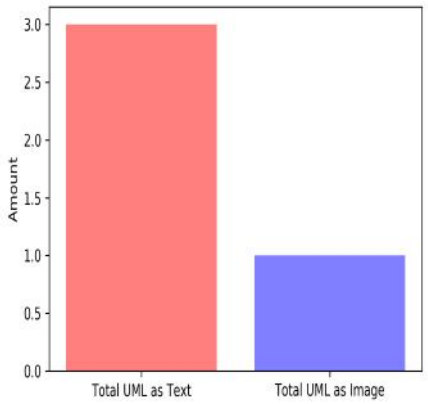


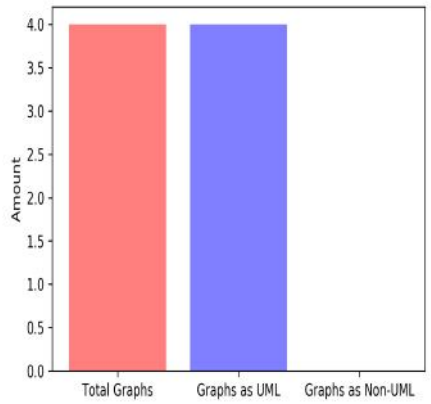


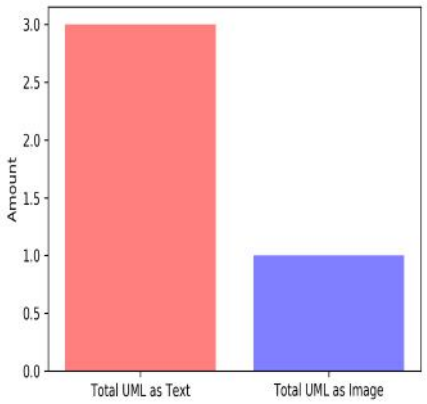
7753c60f **Graph A** **Graph B**



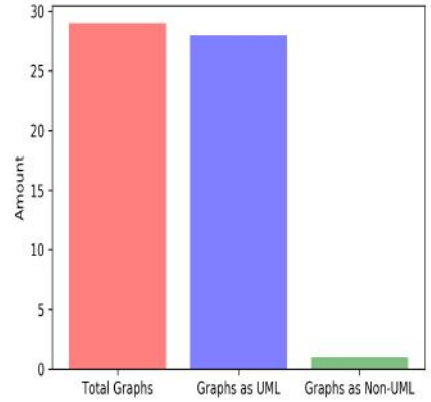


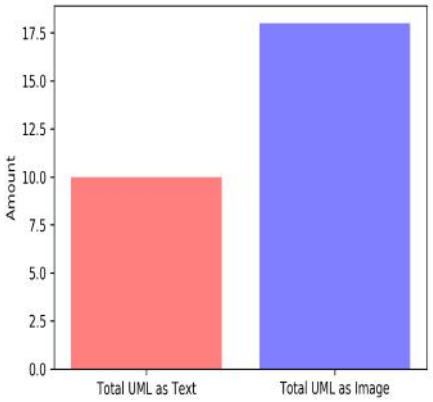
	Graph A	Graph B
02e83ccb		
6a43bae4	**Graph A**	**Graph B**

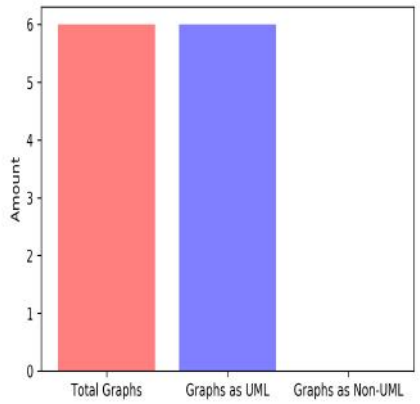
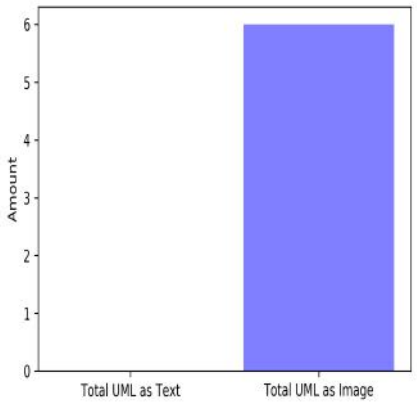


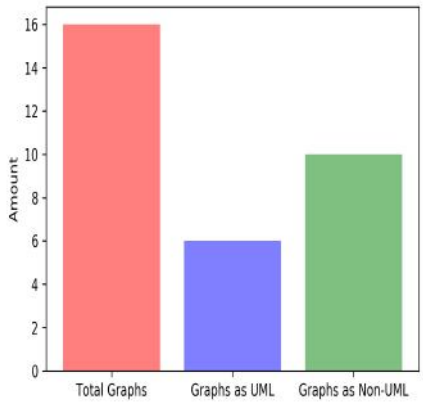


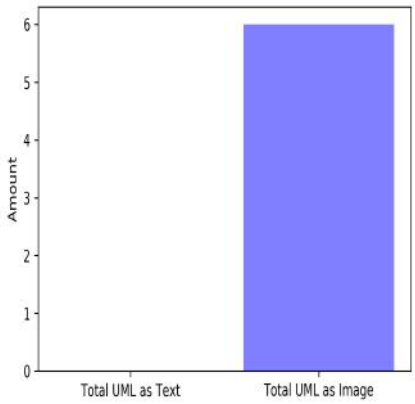
9c699c33 **Graph A** **Graph B**



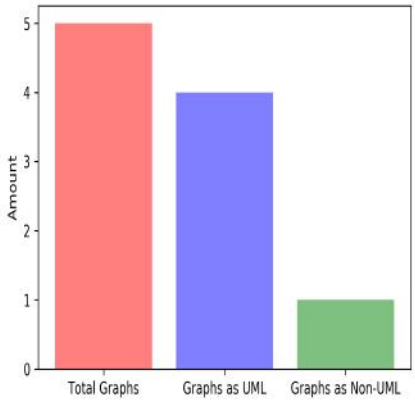


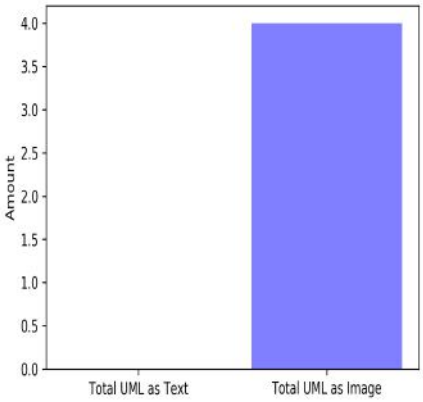
	Graph A	Graph B
41c9999f		
4f582a30	**Graph A**	**Graph B**

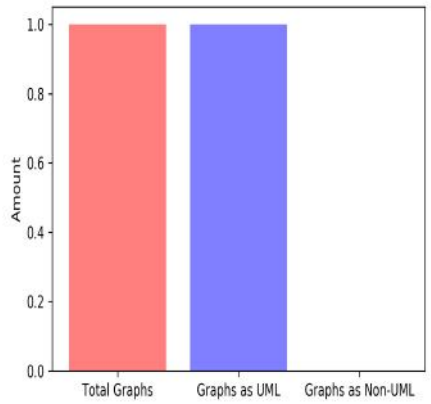
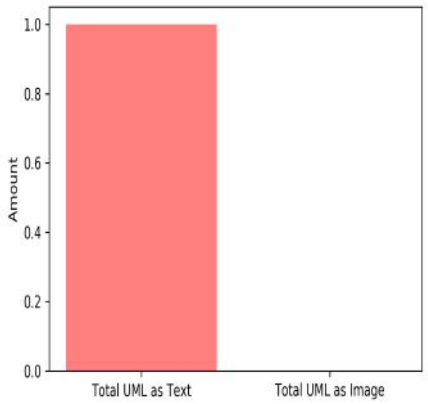


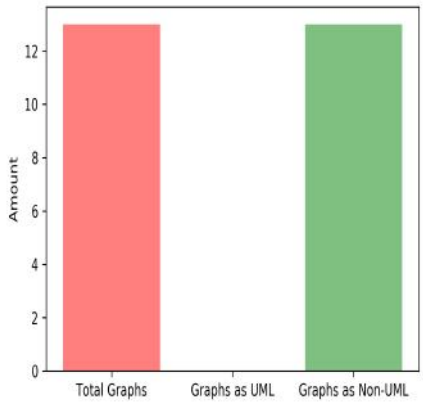


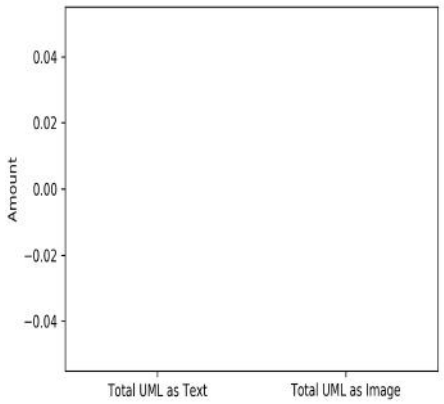
7f90d484 **Graph A** **Graph B**



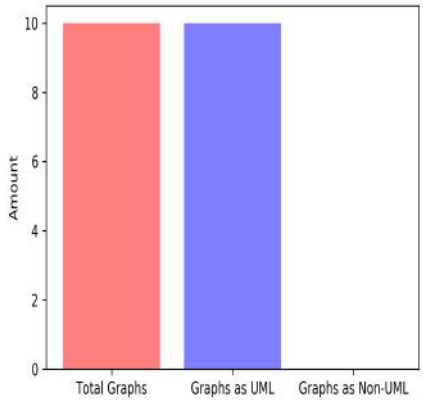


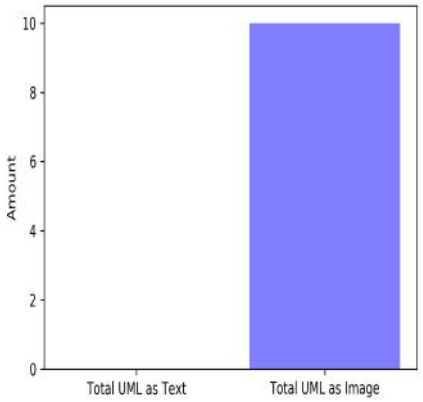
	Graph A	Graph B
3cb5f97e		
000199d7	**Graph A**	**Graph B**

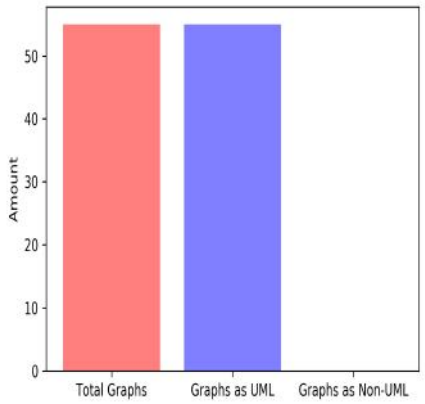
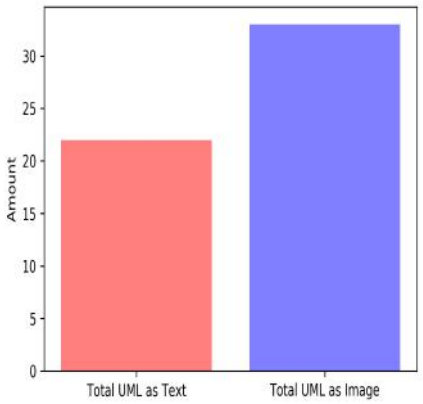


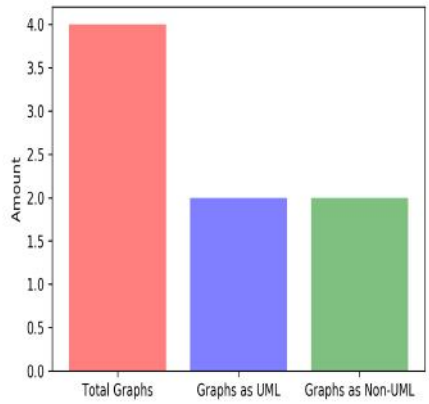


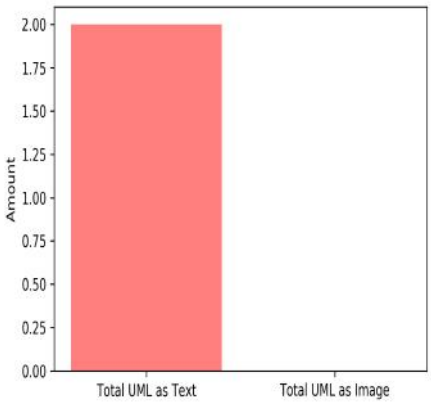
300ef577 **Graph A** **Graph B**



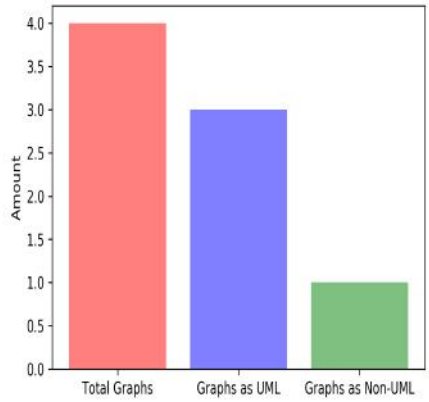


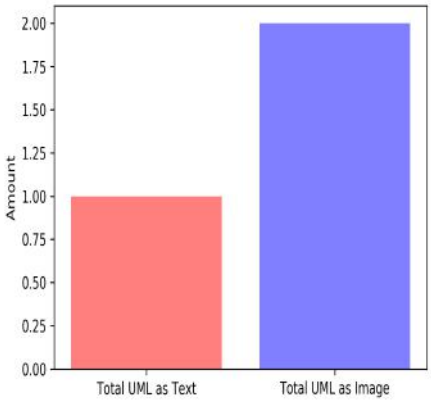
	Graph A	**Graph B**
b8419a88		
9cd7e19b	**Graph A**	**Graph B**

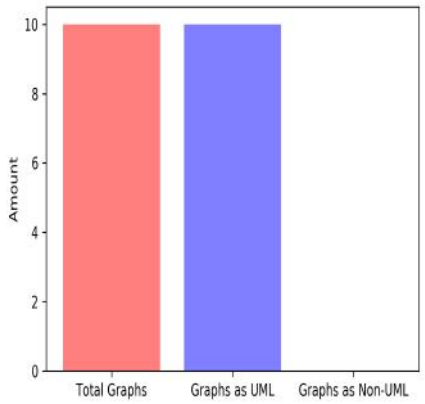
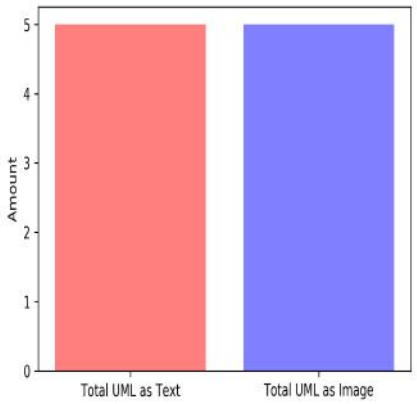


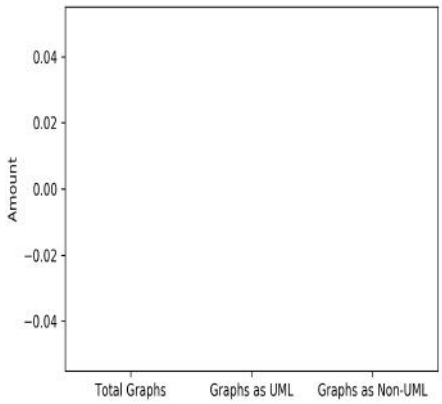


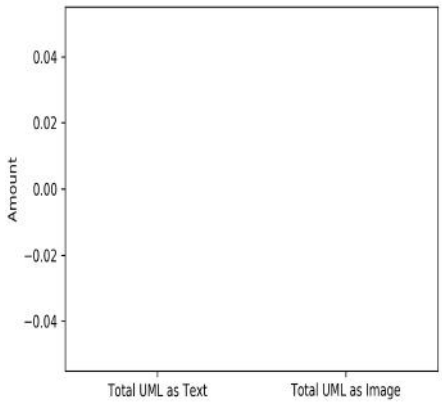
31ce9276 **Graph A** **Graph B**





	Graph A	Graph B
24dd418f		
a958861e	**Graph A**	**Graph B**


0.04
0.02
0.00
−0.02
−0.04
Amount
Total Graphs
Graphs as UML
Graphs as Non-UML


0.04
0.02
0.00
−0.02
−0.04
Amount
Total UML as Text
Total UML as Image

Project ID	UML Diagrams	Non-UML Diagram	UML as Text	UML as Image
1fa7e454	4	0	3	1
b50b5a5f	38	12	0	38
7753c60f	0	0	0	0
02e83ccb	4	0	3	1
6a43bae4	4	0	3	1
9c699c33	28	1	10	18
41c9999f	6	0	0	6
4f582a30	6	10	0	6
7f90d484	4	1	0	4
3cb5f97e	1	0	1	0
000199d7	0	13	0	0
300ef577	10	0	0	10
b8419a88	55	0	22	33
9cd7e19b	2	2	2	0
31ce9276	3	1	1	2
24dd418f	10	0	5	5
a958861e	0	0	0	0

G

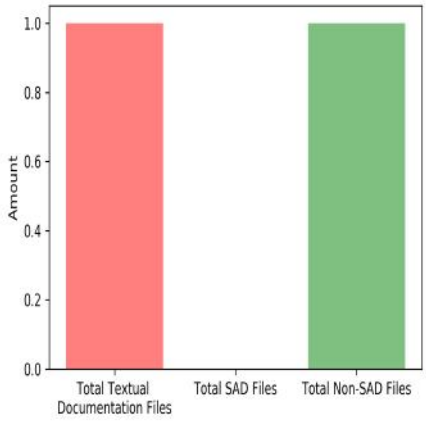
Classification of Textual Documentation

Classification of Textual Documentation

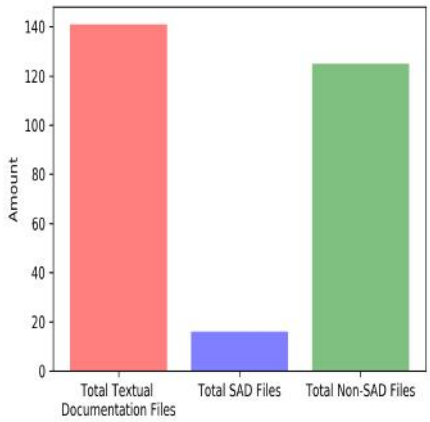
Legend

- Each project has a certain number of documentation files, whose content contains is mainly in textual format. In many cases, such documentation could be associated to the architecture or design of the system. In our study, any file whose purpose is to explain the architecture or design of a system is called a *SAD* file. Therefore, from the total number of textual documentation files available in each project, the graph below illustrates how many are SAD file and how many are not.

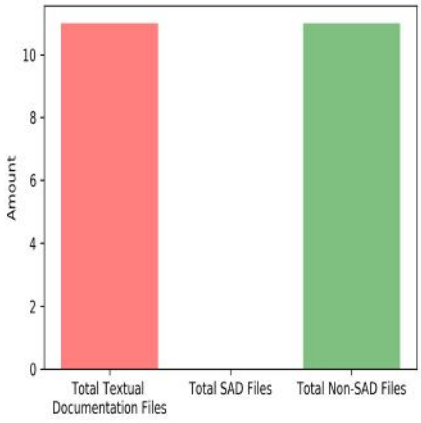
1fa7e454



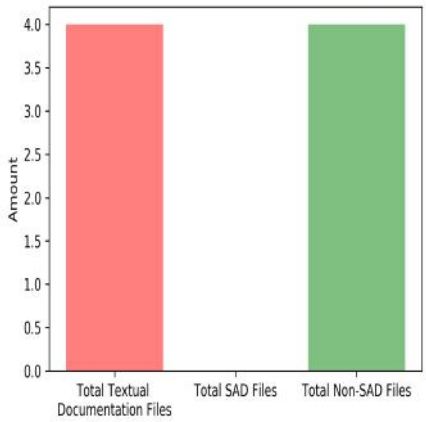
b50b5a5f


140
120
100
80
60
40
20
0
Amount
Total Textual Documentation Files
Total SAD Files
Total Non-SAD Files

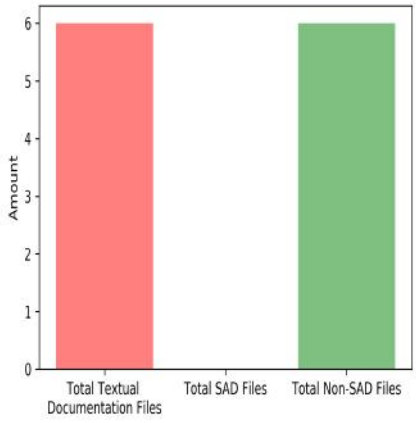
7753c60f


10
8
6
4
2
0
Amount
Total Textual Documentation Files
Total SAD Files
Total Non-SAD Files

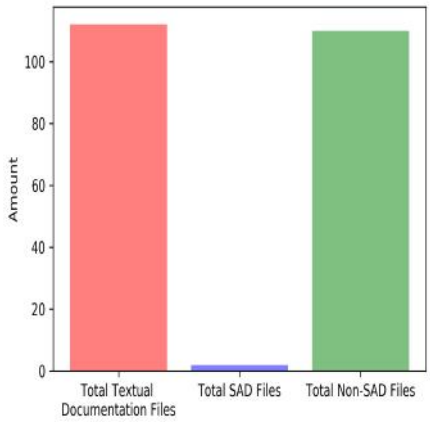
02e83ccb



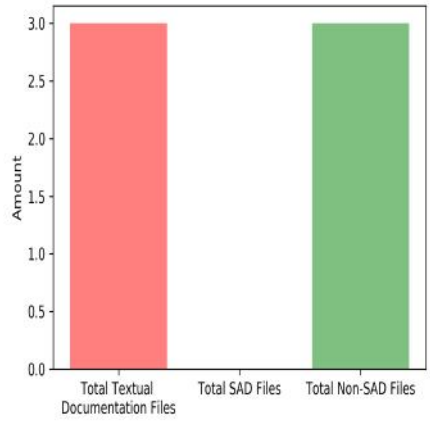
6a43bae4



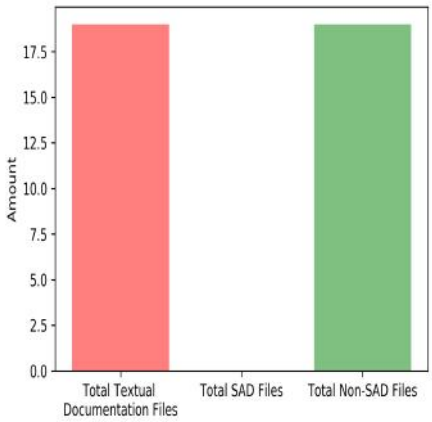
9c699c33



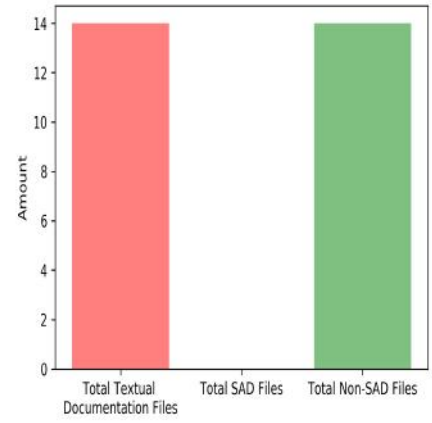
41c9999f



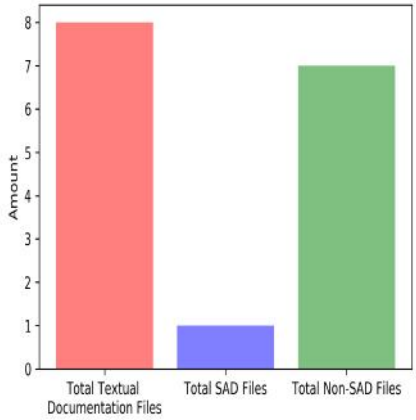
4f582a30



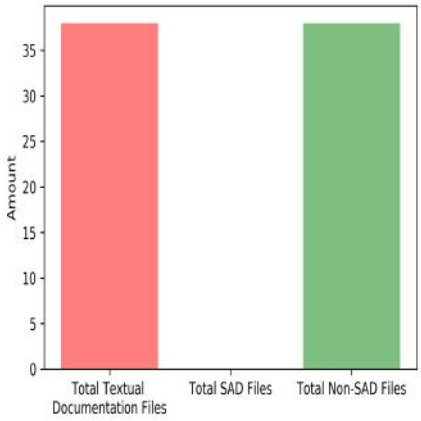
7f90d484



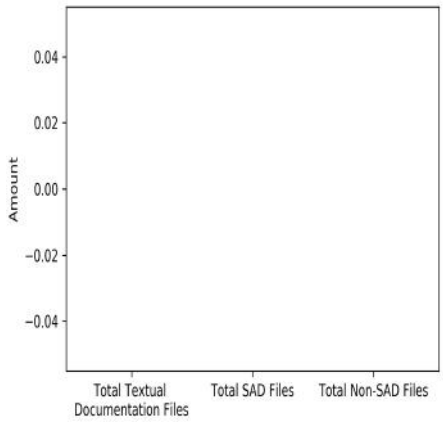
3cb5f97e



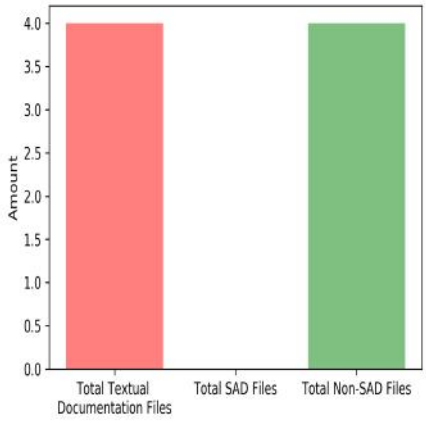
000199d7



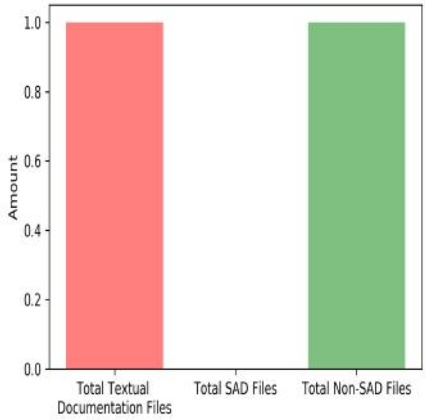
300ef577



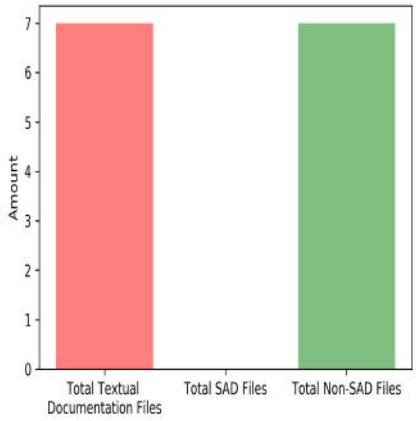
b8419a88



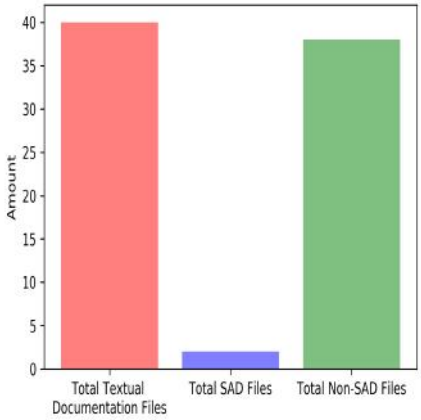
9cd7e19b



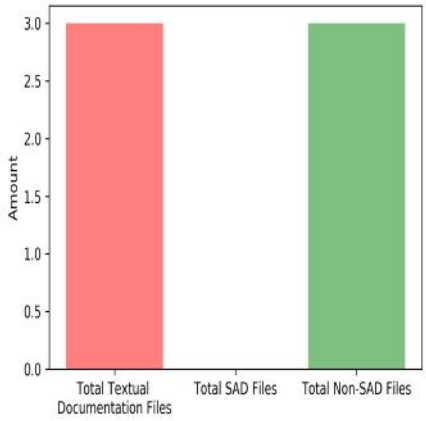
31ce9276



24dd418f



a958861e



Project ID	SAD	Non-SAD	Total
1fa7e454	0	1	1
b50b5a5f	16	125	141
7753c60f	0	11	11
02e83ccb	0	4	4
6a43bae4	0	6	6
9c699c33	2	110	112
41c9999f	0	3	3
4f582a30	0	19	19
7f90d484	0	14	14
3cb5f97e	1	7	8
000199d7	0	38	38
300ef577	0	0	0
b8419a88	0	4	4
9cd7e19b	0	1	1
31ce9276	0	7	7
24dd418f	2	38	40
a958861e	0	3	3

H

Documentation Quality

Documentation Quality

Legend

For each parameter a maximum score of 3 and a minimum of 0 can be obtained. A score of zero indicates that no UML model or documentation was provided. Note that the maximum score for the parameter *Reverse Engineered* is 1.

- **Quality of UML Documentation**

  This graph demonstrate the score for each parameter taken into account when grading model documentation.

- **Quality of SAD Documentation**

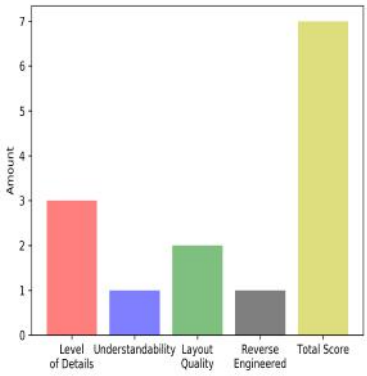
  This graph demonstrate the score for each parameter taken into account when grading SAD documentation.

- **Overall Documentation Quality**

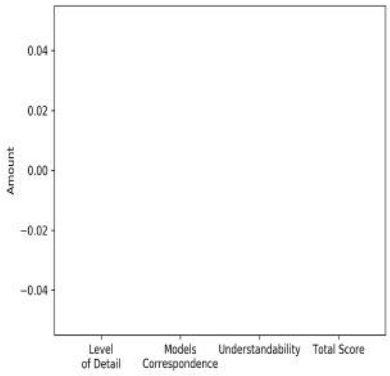
  The overall documentation quality is a sum for the total score obtained for UML documentation quality and SAD documentation quality.

1fa7e454

Quality of UML Documentation



Quality of SAD Documentation



Overall Documentation Quality

7 of 19

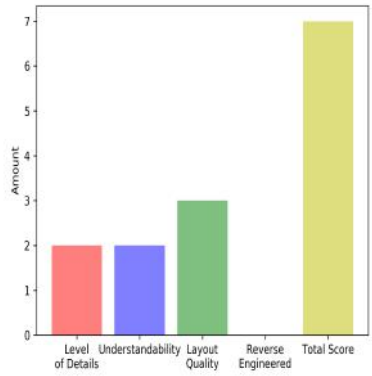
b50b5a5f

Quality of UML Documentation

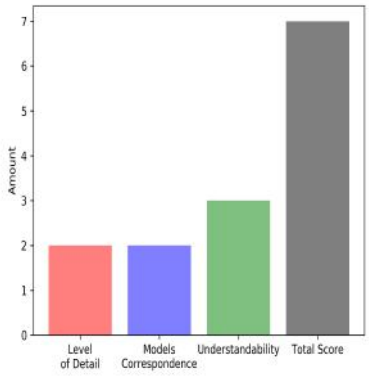
Quality of SAD Documentation

Overall Documentation Quality

14 of 19



Quality of UML Documentation

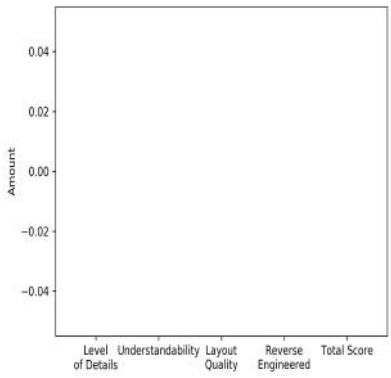


Quality of SAD Documentation

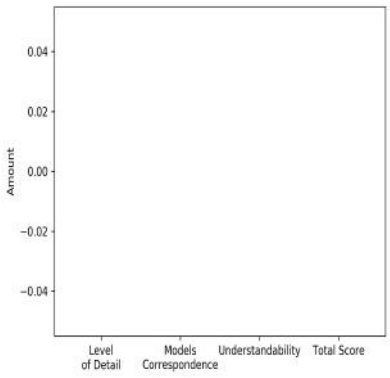
Overall Documentation Quality

0 of 19

7753c60f



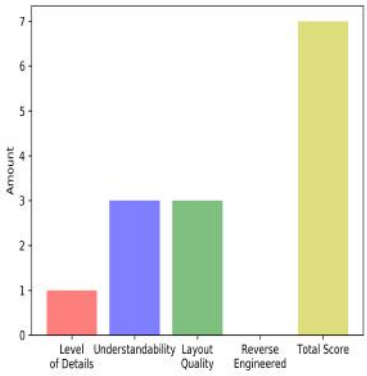
Quality of UML Documentation

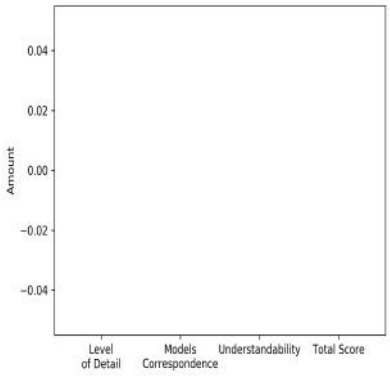


Quality of SAD Documentation

Overall Documentation Quality

02e83ccb





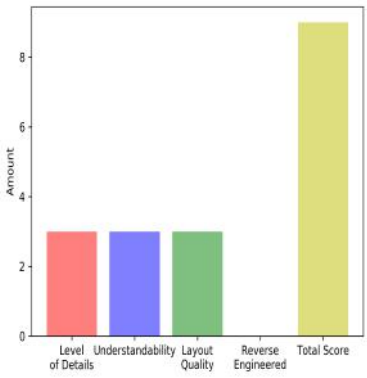
7 of 19

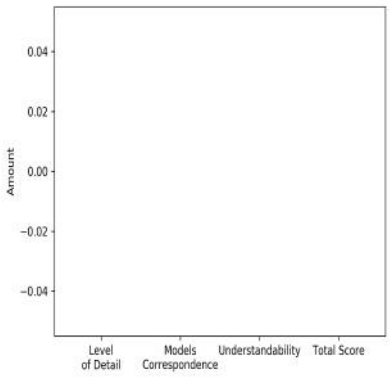
Quality of UML Documentation

Quality of SAD Documentation

Overall Documentation Quality

6a43bae4





9 of 19

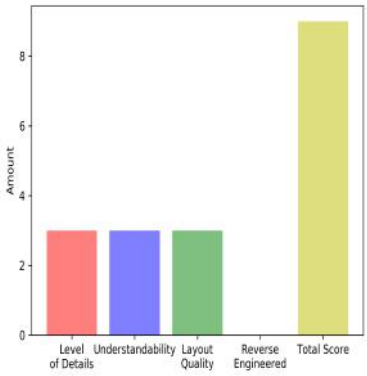
9c699c33

Quality of UML Documentation

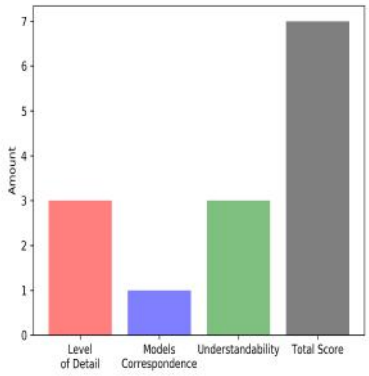
Quality of SAD Documentation

Overall Documentation Quality

16 of 19



Quality of UML Documentation

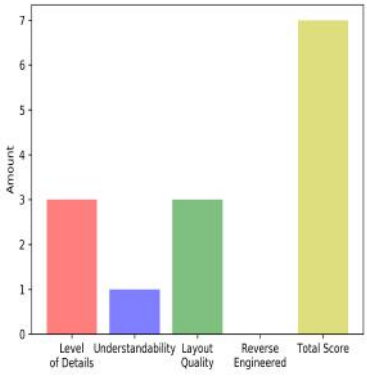


Quality of SAD Documentation

Overall Documentation Quality

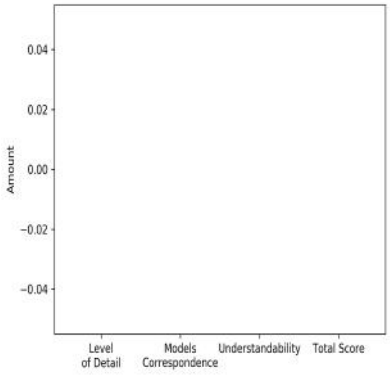
7 of 19

41c9999f



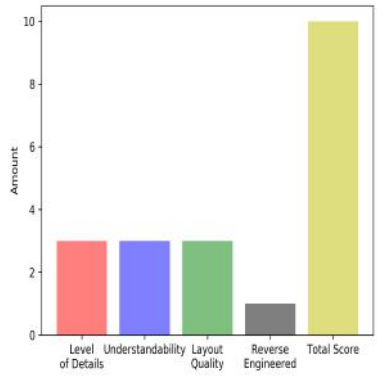
4f582a30

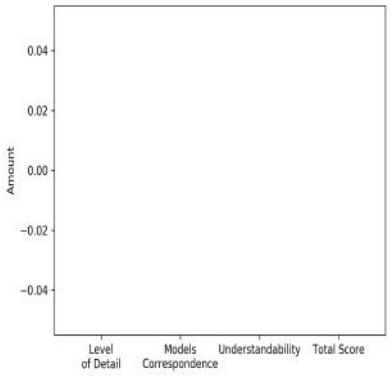
Quality of UML Documentation



Quality of SAD Documentation

Overall Documentation Quality





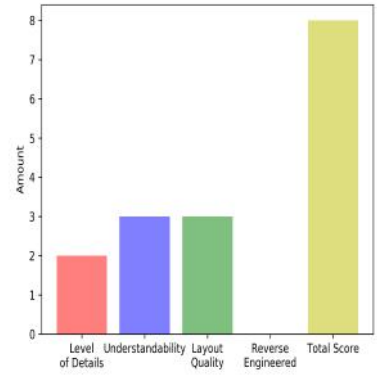
10 of 19

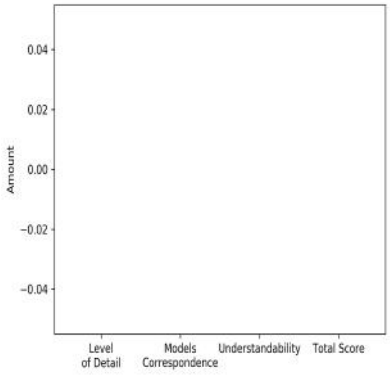
Quality of UML Documentation

Quality of SAD Documentation

Overall Documentation Quality

7f90d484





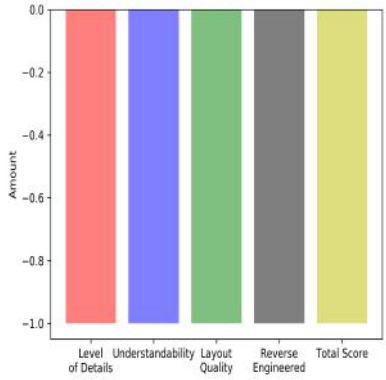
8 of 19

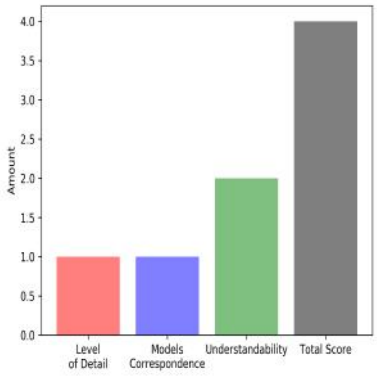
3cb5f97e

Quality of UML Documentation

Quality of SAD Documentation

Overall Documentation Quality





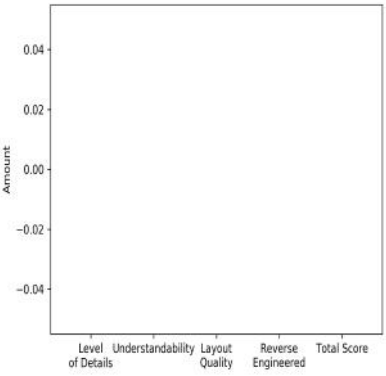
5 of 19

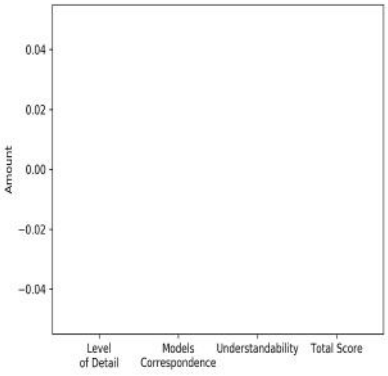
Quality of UML Documentation

Quality of SAD Documentation

Overall Documentation Quality

000199d7





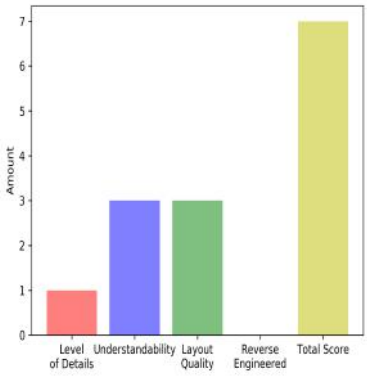
0 of 19

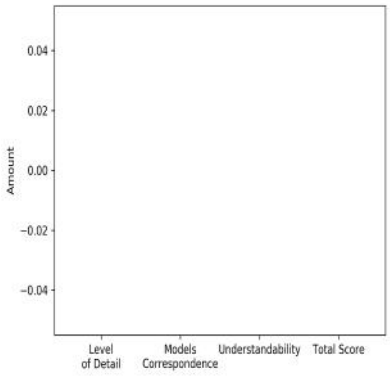
300ef577

Quality of UML Documentation

Quality of SAD Documentation

Overall Documentation Quality





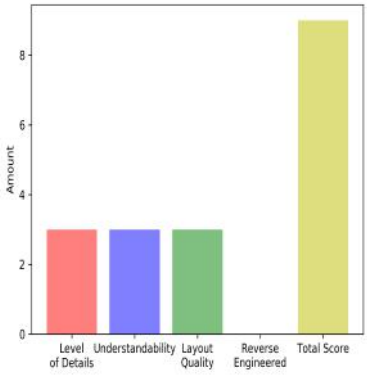
7 of 19

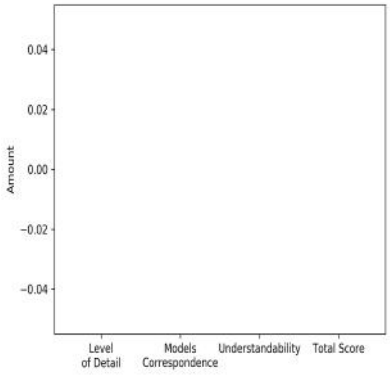
Quality of UML Documentation

Quality of SAD Documentation

Overall Documentation Quality

b8419a88





9 of 19

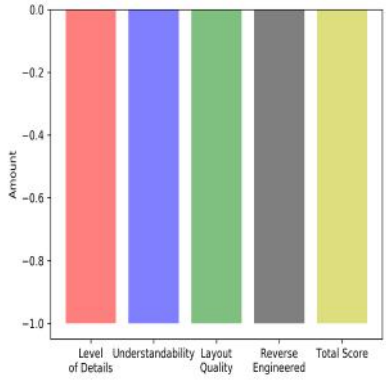
9cd7e19b

Quality of UML Documentation

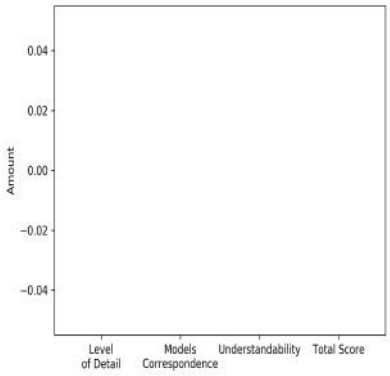
Quality of SAD Documentation

Overall Documentation Quality

1 of 19



Quality of UML Documentation

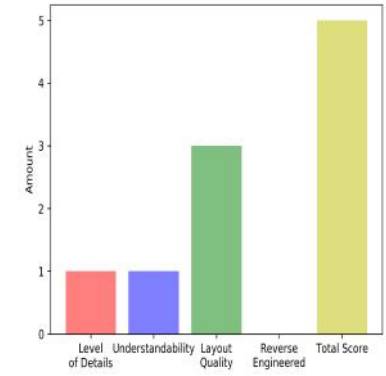


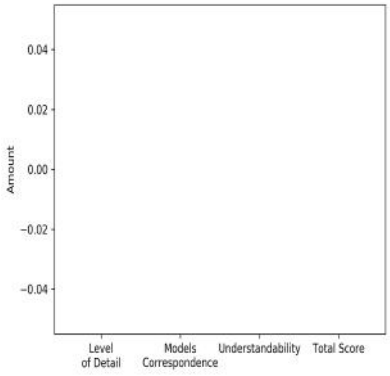
Quality of SAD Documentation

Overall Documentation Quality

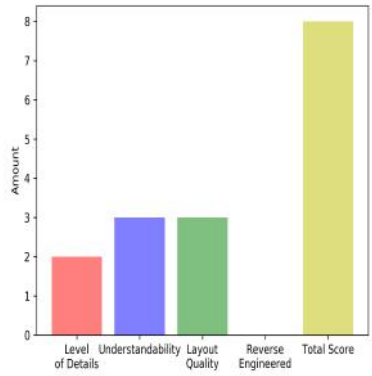
5 of 19

31ce9276

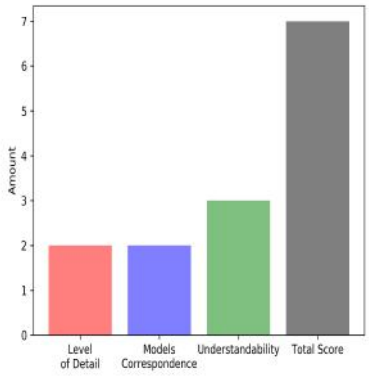




24dd418f **Quality of UML Documentation** **Quality of SAD Documentation** **Overall Documentation Quality**



Quality of UML Documentation

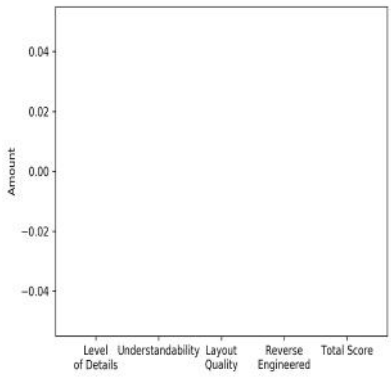


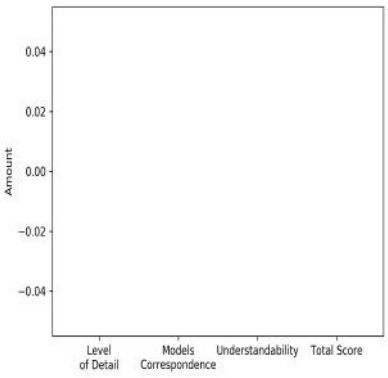
Quality of SAD Documentation

15 of 19

Overall Documentation Quality

a958861e





0 of 19

Documentation Quality

Project ID	Models Quality						SAD Quality					Full Total Score
	Total UML Models	Level of Details	Understandability	Layout Quality	Reversed Engineered	Total	Total SAD Files	Level of Detail	Correspondance to models	Understandability	Total	
1fa7e454	4	3	1	2	1	7	0	0	0	0	0	7
b50b5a5f	38	2	2	3	0	7	16	2	2	3	7	14
7753c60f	0	0	0	0	0	0	0	0	0	0	0	0
02e83ccb	4	1	3	3	0	7	0	0	0	0	0	7
6a43bae4	4	3	3	3	0	9	0	0	0	0	0	9
9c699c33	28	3	3	3	0	9	2	3	1	3	7	16
41c9999f	6	3	1	3	0	7	0	0	0	0	0	7
4f582a30	6	3	3	3	1	10	0	0	0	0	0	10
7f90d484	4	2	3	3	0	8	0	0	0	0	0	8
3cb5f97e	1	-1	-1	-1	-1	1	1	1	1	2	4	5
000199d7	0	0	0	0	0	0	0	0	0	0	0	0
300ef577	10	1	3	3	0	7	0	0	0	0	0	7
b8419a88	55	3	3	3	0	9	0	0	0	0	0	9
9cd7e19b	2	-1	-1	-1	-1	1	0	0	0	0	0	1
31ce9276	3	1	1	3	0	5	0	0	0	0	0	5
24dd418f	10	2	3	3	0	8	2	2	2	3	7	15
a958861e	0	0	0	0	0	0	0	0	0	0	0	0

Note: Value of -1 means that it was not possible to open the files associated to the models. The project was assigned a score of 1 for at least containing model documentation.

I

Input for Correlation Analysis

Doc to SC Commit ratio	Num of Doc Files	Num of Source Code Files	Total Doc Quality	cbo_new	total_loc	total_modules
0.014	3	271	0	0.367	4695	60
0.016	141	1584	16	2.930	119418	910
0.041	5	527	7	0.862	6461	218
0.035	191	1467	14	4.677	157798	576
0.043	59	132	9	2.707	6050	123
0.054	5	145	1	1.250	4708	96
0.009	19	1120	8	4.577	77478	685
0.060	10	192	9	1.801	7586	136
0.026	51	1242	0	2.324	111924	743
0.035	9	247	5	0.964	9613	168
0.021	11	229	5	1.311	7633	177
0.016	10	2406	7	1.007	49509	736
0.063	35	376	10	2.677	40912	511
0.041	9	328	7	1.309	8440	162
0.023	8	505	7	2.056	70926	641
0.005	11	1650	0	1.869	26074	268
0.017	50	81	15	1.333	3974	66

J

Normality Plot and Box Plot

Visual Statistics - Part 1

Legend

- **Box Plot**

  The purpose of this plot is to visualize determine if the data contains outliers.

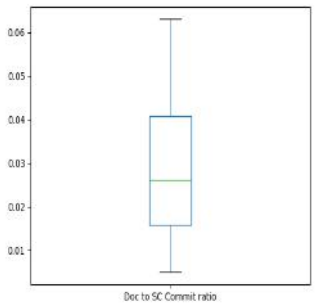
- **QQ Plot**

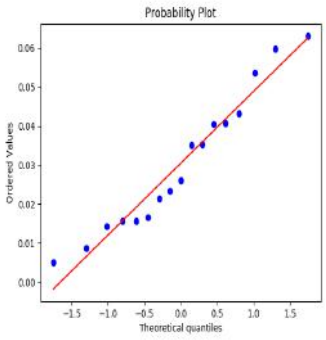
  The purpose of this plot is to determine if the data is normally distributed.

Box Plot

QQ Plot

Doc to SC Commit ratio





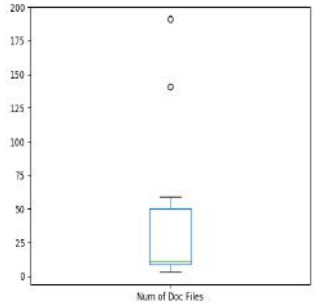
Shapiro-Wilk Test Result:

p value = 0.36070355772972107

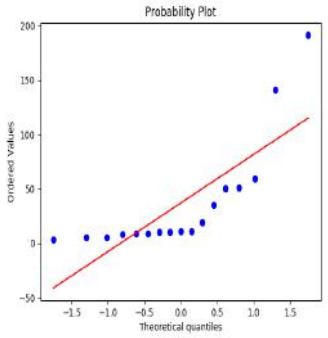
Num of Doc Files

Box Plot

QQ Plot



Box Plot

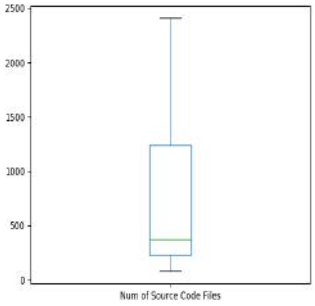


QQ Plot

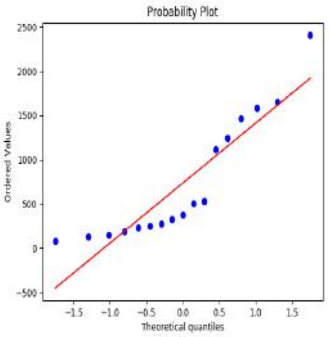
Shapiro-Wilk Test Result:

p value = 3.733437552000396e-05

Num of Source Code Files



Box Plot



QQ Plot

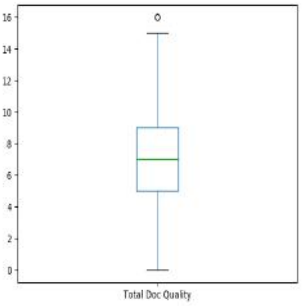
Shapiro-Wilk Test Result:

p value = 0.0051766578108072 28

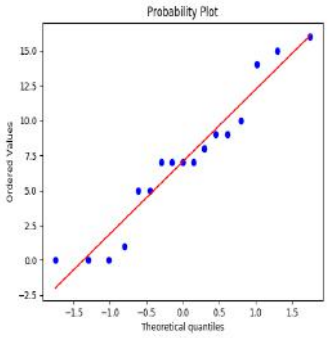
Total Doc Quality

Box Plot

QQ Plot



Box Plot

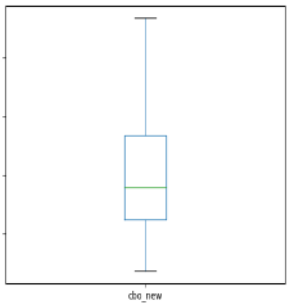


QQ Plot

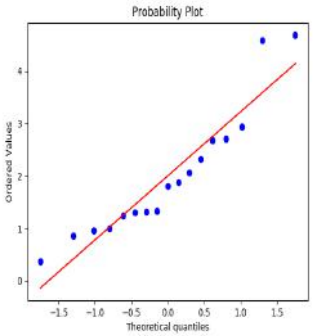
Shapiro-Wilk Test Result:

p value = 0.19283972680568695

cbo_new



Box Plot



QQ Plot

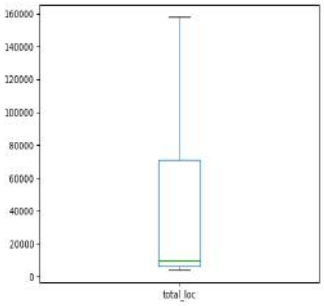
Shapiro-Wilk Test Result:

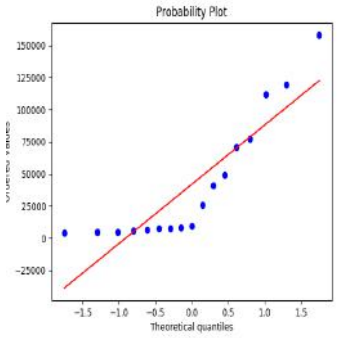
p value = 0.045170482248067856

total_loc

Box Plot

QQ Plot





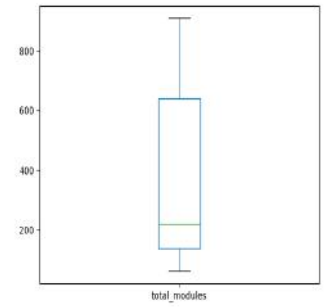
Shapiro-Wilk Test Result:

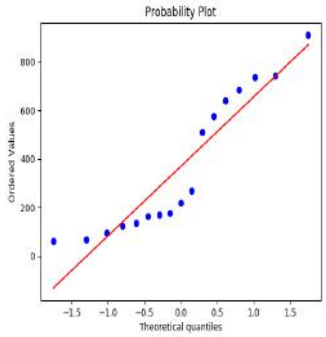
p value = 0.0013221652479842305

Box Plot

QQ Plot

total_modules





Shapiro-Wilk Test Result:

p value = 0.013696896843612194

K

Correlation Graphs

Visual Statistics - Part 2

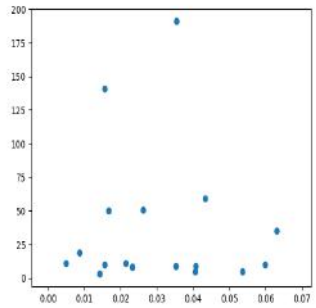
Legend

- **Correlation Graph**

  This graphs demonstrate the correlation between the specified items.

Doc to SC Commit ratio VS Num of Doc Files

Correlation Graph

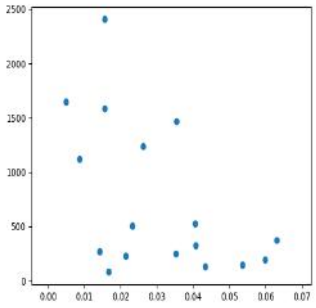


Levene's Test Result:

p value = 0.023076464440033712

Doc to SC Commit ratio VS Num of Source Code Files

Correlation Graph

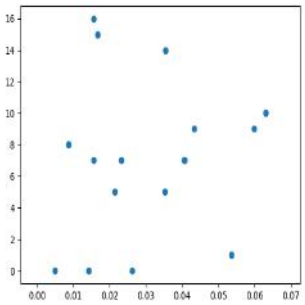


Levene's Test Result:

p value = 0.0007855210352787814

Doc to SC Commit ratio VS Total Doc Quality

Correlation Graph

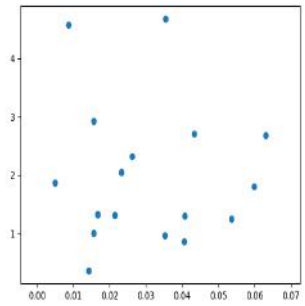


Levene's Test Result:

p value = 5.0921571657619 91e-05

Doc to SC Commit ratio VS cbo new

Correlation Graph

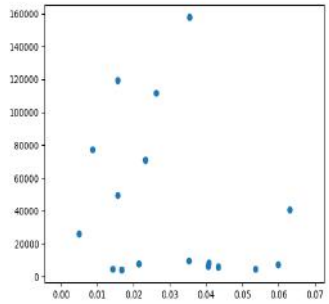


Levene's Test Result:

p value = 7.421011362270388e-05

Doc to SC Commit ratio VS total loc

Correlation Graph

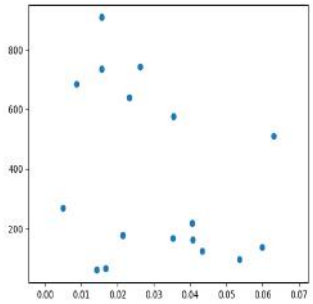


Levene's Test Result:

p value = 0.0033832325657260605

Correlation Graph

Doc to SC Commit ratio VS total modules

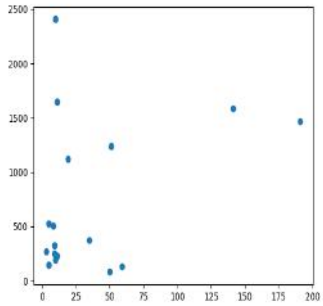


Levene's Test Result:

p value = 6.395780245332088e-05

Num of Doc Files VS Num of Source Code Files

Correlation Graph

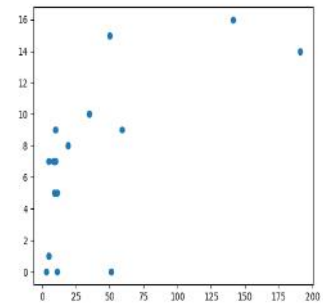


Levene's Test Result:

p value = 0.0014366826039991139

Correlation Graph

Num of Doc Files VS Total Doc Quality

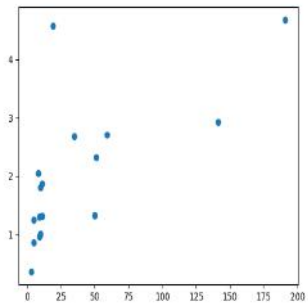


Levene's Test Result:

p value = 0.045461287322201976

Num of Doc Files VS cbo new

Correlation Graph

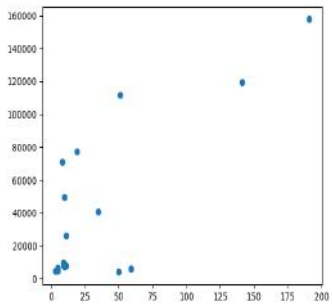


Levene's Test Result:

p value = 0.027256471333664337

Correlation Graph

Num of Doc Files VS total loc

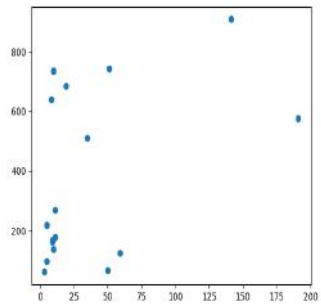


Levene's Test Result:

p value = 0.0034063334328217646

Num of Doc Files VS total modules

Correlation Graph

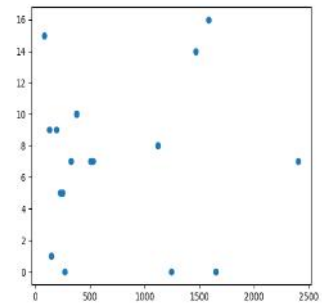


Levene's Test Result:

p value = 0.00042733798957598

Correlation Graph

Num of Source Code Files VS Total Doc Quality

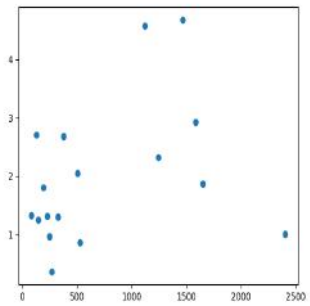


Levene's Test Result:

p value = 0.0008444297804990731

Num of Source Code Files VS cbo new

Correlation Graph

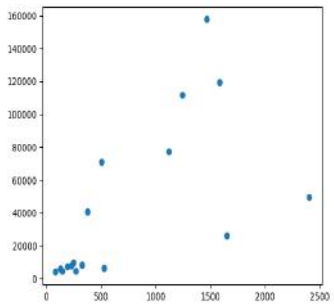


Levene's Test Result:

p value = 0.0007993573228403206

Correlation Graph

Num of Source Code Files VS total loc

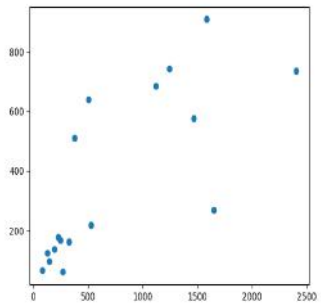


Levene's Test Result:

p value = 0.003820541750951273

Num of Source Code Files VS total modules

Correlation Graph

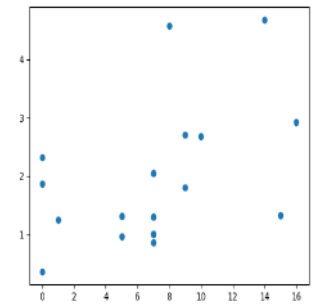


Levene's Test Result:

p value = 0.06944789898886793

Correlation Graph

Total Doc Quality VS cbo new

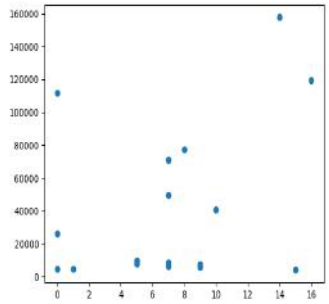


Levene's Test Result:

p value = 0.0016336538928209414

Total Doc Quality VS total loc

Correlation Graph

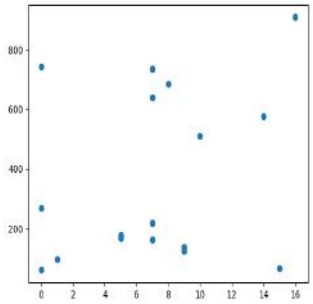


Levene's Test Result:

p value = 0.0033861345167250177

Correlation Graph

Total Doc Quality VS total modules

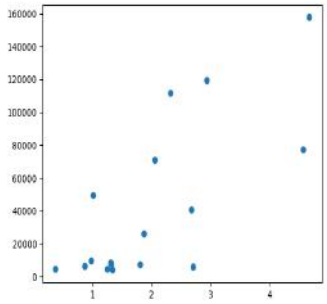


Levene's Test Result:

p value = 7.851129349475941e-05

cbo new VS total loc

Correlation Graph

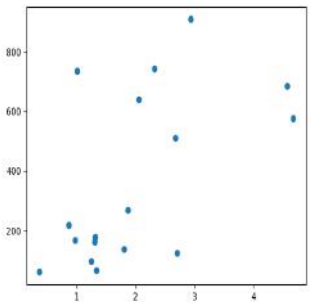


Levene's Test Result:

p value = 0.0033839336258097057

Correlation Graph

cbo new VS total modules

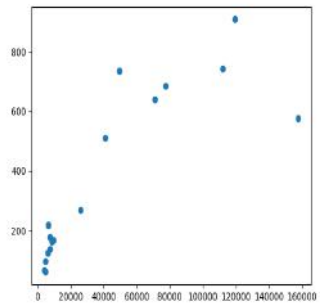


Levene's Test Result:

p value = 6.719115216211891e-05

total loc VS total modules

Correlation Graph



Levene's Test Result:

p value = 0.003577334088154543

L

Project Description

The following table provides the name and description of each Github project used in this study.

Project ID	Project Name	Description
1fa7e454	Qabel/qabel-core	Qabel is a free, published-source cryptography platform.
b50b5a5f	gammalib/gammalib	The GammaLib is a versatile toolbox for the high-level analysis of astronomical gamma-ray data.
7753c60f	aegif/NemakiWare	NemakiWare is an open source Enterprise Content Management system.
02e83ccb	adiknoth/ffado	The FFADO project aims to provide a free driver implemenation for FireWire (IEEE1394, iLink) based audio interfaces.
6a43bae4	FamilySearch/gedcomx	An open data model and an open serialization format for exchanging the genealogical data essential to the genealogical research process.
9c699c33	IQSS/dataverse	Dataverse is an open source web application for sharing, citing, analyzing, and preserving research data developed by the Data Science and Products team at the Institute for Quantitative Social Science and the Dataverse community.
41c9999f	AMOSTeam3/amos-ss15-proj3	This product maps changes to source code files with requirements.

4f582a30	animatedb/oovaide	An object oriented analysis and integrated development platform that automatically generates build, class, sequence, zone, portion, and component diagrams for C++, Objective C, and Java languages.
7f90d484	GluuFederation/oxAuth	OxAuth is an open source OpenID Connect Provider (OP) and UMA Authorization Server (AS).
3cb5f97e	KDE/wacomtablet	Wacomtablet implements a GUI for the Wacom Linux Drivers and extends it with profile support to handle different button / pen layouts per profile.
000199d7	fspindle/visp	Visp is a cross-platform library (Linux, Windows, Mac) that allows prototyping and developing applications using visual tracking and visual servoing technics at the heart of the researches done by Inria Lagadic team.
300ef577	apache/clerezza	Apache Clerezza is a set of Java libraries for management of semantically linked data.
b8419a88	Hjdskes/2048	Class project for course Software Engineering Methods.
9cd7e19b	joherma1/sia	SIA is an open Agricultural Information System.
31ce9276	apache/commons-rdf	Commons RDF aims to provide a common library for RDF 1.1 with implementations for common Java RDF frameworks like RDF4J, Apache Jena as well as for other libraries such as OWLAPI, Clerezza and other JVM languages.
24dd418f	abjugard/DAT255-EpiClock	Group09's project repository for course DAT255 at Chalmers Institute of Technology, faculty of Information Technology.

a958861e	jpmorganchase/perspective	A streaming data visualization engine for Javascript, Perspective makes it simple to build real-time and user configurable analytics entirely in the browser.

M

Correlation Strength

The result of a correlation analysis between variables X and Y is a correlation coefficient that indicates the magnitude and direction of the correlation between X and Y. The magnitude of a correlation coefficient ranges from -1 to 1, which represent the largest strength of correlation. In this study, the following scale was used for interpreting the strength of the correlation coefficients [1].

Strength of Association	**Positive**	**Negative**
Small	0.1 to 0.3	-0.1 to -0.3
Medium	0.3 to 0.5	-0.3 to -0.5
Large	0.5 to 1.0	-0.5 to -1.0

Table M.1: Strength Scale

It is important to mention **Table M.1** is one of many possible scales that could be used for interpreting the strength of a correlation coefficient magnitude. There is no special reason for using this scale other than is a scale commonly proposed.

[1] `https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php`