Analyzing Software Quality and Maintainability in Object-Oriented Systems using Software Metrics

Submitted by

1. Faria Tasim ID: 24341110
2. Farib Md. Ferdoush ID: 24341120
3. Salequzzaman Khan ID: 20101330
4. Mahdi Islam ID: 20101326
5. Fatema Haque ID: 20101415

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
October 2024

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing a degree at Brac University.

2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.

3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.

4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Faria Tasnim

Faria Tasnim
24341110

[illegible]

Farib Md. Ferdoush
24341120

[illegible]

Salequzzaman Khan
20101330

Mahdi

Mahdi Islam
20101326

[illegible]

Fatema Haque
20101415

Approval

The thesis titled "Analyzing Software Quality and Maintainability in Object-Oriented Systems using Software Metrics" submitted by

1. Faria Tasim ID: 24341110
2. Farib Md. Ferdoush ID: 24341120
3. Salequzzaman Khan ID: 20101330
4. Mahdi Islam ID: 20101326
5. Fatema Haque ID: 20101415

Summer, 2024 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on October 17, 2024.

Examining Committee:
Supervisor:
(Member)

Md. Aquib Azmain

Mr. Md. Aquib Azmain
Lecturer
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam, PhD
Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Ethics

We formally announce the outcomes of our research to be in favor of this concept. This study, on its whole, is devoid of plagiarism. The source material includes citations for any additional information sources. To award a degree, this thesis has not been submitted, in whole or in part, to any other institution or organization.

Abstract

Effective evaluation of software quality and maintainability is compulsory for successful object-oriented system development, and the potential of software metrics in achieving these goals are investigated in this research. To evaluate the quality of software, this research employs software metrics to identify potential errors and weaknesses in object-oriented systems. This analysis has been conducted by us in the Python programming language. We have applied machine learning techniques to different software metrics to analyze the issues consistently, which has evaluated the effectiveness and long-term feasibility of the system. Lastly, this study e stablishes a foundation for future advancements in software quality assurance, demonstrating the significant benefits of integrating machine learning with traditional quality measurements to enhance the predictability and reliability of object-oriented systems.

Keywords: LOC (Lines of Code), Comment Percentage, Cyclomatic Complexity (CC), Weighted Methods per Class (WMC), Access to Foreign Data (ATFD), Response For a Class (RFC)

Dedication

Every challenging endeavor demands our elders, especially the ones closest to our hearts, to put forth personal effort and support. In addition to all of the exceptional academics we encountered and learned from while pursuing our bachelor's degrees, and especially our cherished supervisor, Dr. Md. Aquib Azmain, we dedicate our efforts to our loving parents, whose love, devotion, motivation, and nightly prayers have made us deserving of this achievement and honor.

Acknowledgement

Firstly, all praise to the Great Allah for whom our thesis have been completed without any major interruption. Secondly, to our supervisor Md. Aquib Azmain sir for his kind support and advice in our work. He helped us whenever we needed help. Finally, to our parents without their throughout support it may not be possible. With their kind support and prayer we are now on the verge of our graduation.

Contents

List of Figures

Chapter 1

Introduction

Evaluating software quality and maintainability is really important for object-oriented systems to succeed in the long term. The implementation of structured object-oriented design and programming has becoming the most widely used paradigm in today's software systems [6]. As systems become more complex, manual assessment becomes increasingly challenging. This highlights the necessity for automated methods using software metrics to identify issues that can undermine quality over time. One essential aspect of quality analysis involves detecting code smells, which are warning signs of deeper design problems that can complicate code comprehension and maintenance. This research focuses on utilizing metrics to automatically detect code smells in Python programs.

We focus our study on object-oriented Python projects, as Python has gained tremendous popularity in recent years. Numerous static and dynamic metrics will be collected, including lines of code, weighted methods per class, response for Class etc. Our current project focuses on making our analysis more comprehensive by including metrics that consider dynamic aspects. We aim to improve the detection of code smells by integrating these metrics with machine-learning models. Enhanced automated analysis will help developers keep code clean, understandable, and maintainable as complexity increases over time.

1.1 Motivation

Software development aimed at quality, maintenance, and being standard is what the developers are expected to pursue. By being able to peer into the heartland of software metrics, it is there that we find the breathtaking landscape of object-oriented systems, containing all the information that we need to develop, build, and maintain software in a different way. By identifying weaknesses, indicating places of viable growth, and at the same time creating more comprehensive systems, critical analysis and evaluations can be done. This process not only adds credibility and increases the performance of our product but also changes us by generating solutions that are scalable, long-lasting, and reliable. The trouble with the application of software metrics in the assessment and sustainability of software quality and maintenance is more than what we do; it is the experience that transforms our living into a superior state provided by the information technology sector within the field of software engineering.

Software technology everywhere covers infrastructure and end-user applications as well as many other fields of technological life with a continual development pace. The quality of the software can be rather determined by a customer satisfaction experience, reliability, and performance. First-class software must be reliable, right, and secure, which means that operations always be accurate and effective. Software quality analysis management offers reduce on maintenance costs, performance improvement and identifying (actual) problems at the early level of programming.

Another key factor is maintainability, which is also a standalone characteristic of high-quality software. It implies that ongoing maintenance and modification processes in software make it an easy and affordable task for a software system administrator to correct flaws, add new features, or receive environmental change. Every frequency as software systems grow in complexity, maintenance costs rise, too. Through utilization of understanding and improvement on maintainability, companies would eventually realize cost savings over the long-term period.

1.2 Problem Statement

The problem revolves around assessing software quality and maintainability effectively in complex object-oriented systems, where manual evaluations are impractical due to project complexity. Neglecting these factors can result in a decrease in system dependability and an increase in technical debt. The long-term health of the system depends on human assessors being able to recognize code smells, which are subtle signs of possible issues. The major problem is that software quality and maintainability assessments and systematic code smell identification require automated methods. Due to their complex class hierarchies and encapsulation, object-oriented systems are particularly vulnerable to this issue. Lines of code and cyclomatic complexity are two conventional metrics that could not offer a complete picture of program quality. The study becomes more difficult when dynamic measurements are included and runtime behavior is taken into account. To solve this, we provide a methodical software metric-gathering approach that includes both static and dynamic measurements.Static metrics, such as code complexity, cohesion, coupling, and modularity, provide insights into the structure and interdependencies within the code. Dynamic metrics consider runtime behaviour and performance, offering a more comprehensive understanding of how the software operates in real-world conditions. By using machine learning,it ensures consistent detection and practical relevance. Comparing performance to industry benchmarks makes it easier to assess performance, pinpoint problem areas, and suggest best practices. Practical relevance is ensured by real-world validation using metrics on existing object-oriented systems. Last but not least, we want to incorporate new knowledge into how software is developed, encouraging best practices and raising the standard of software and its capacity to be maintained in object-oriented systems.

1.3 Research Objectives

Evaluate Software Quality using Metrics: The essential goal is to utilize programming to evaluate and quantify software quality in object-oriented systems. This includes examining metrics connected with code complexity, cohesion, coupling, and other important markers to acquire bits of knowledge about the nature of the product.

Assess Maintainability using Metrics: To assess the drawn-out maintainability of article-based systems, different software maintainability will be utilized. This evaluation incorporates analyzing metrics connected with code modularity, reusability, readability, and maintainability to distinguish regions for development.

Identify Correlations between Metrics and Quality Attributes: This includes laying out connections between software metrics and explicit quality attributes dependability, effectiveness). By relating metrics with noticed quality attributes, we mean to comprehend what programming plan decisions mean for software quality and maintainability.

Benchmarking and Comparative Analysis: Lead a benchmarking investigation by looking at the metrics obtained from object-oriented systems with established industry benchmarks. This takes into consideration a relative analysis of the software's quality and maintainability against industry standards, enabling a better understanding of its performance.

Recommendations for Improvement: In light of the analysis and findings, propose recommendations and best practices for further developing software quality and viability in object-oriented programming. These proposals will be drawn from the recognized connections among metrics and quality attributes.

validation through Case Studies Approve the proposed metrics and analysis approach through real-world object-oriented systems. This objective ensures the relevance and adequacy of the distinguished metrics in real-world scenarios.

Integrate Findings into Software Development Practices: Give rules on how the insights obtained from the analysis can be into software development processes into software development processes. This coordination intends to further develop the improvement works, leading to better software quality and maintainability.

Chapter 2

Literature Review

2.1 Application of machine learning algorithms for code smell prediction using object-oriented software metrics

Agnihotri et al. (2020) [1] identified the value of utilizing machine learning to recognize code smells in Java systems. The initial purpose is to facilitate software quality improvements and to reduce the effort required for software maintenance by detecting code smells, which are potential defects in the code. The study employs an extensive range of software metrics and machine-learning approaches to assist developers in comprehending source code quality. One of the most important issues is the thorough description of the particular code smell types including the detailed analysis and prediction results and rules of the God Class and Feature Envy code smell types [1]. This comes in very handy to provide better clarity towards quality defects in Java projects.

As far as both are related to the ongoing research there are several obvious parallels. The two studies reaffirmed the criticality of software metrics in determining software quality and its maintainability. They also align with their belief that the more complex software systems are, it is necessary to automate the detection of code smells since manual analysis fails to meet the required standards. Moreover, both studies aim to use outcomes from machine learning to facilitate code quality analysis.

Despite there are certain similarities between the two studies there are a number of differences between them which should be mentioned. The result of the previous research only provides information related to Java projects. However, the current study is done on object-oriented Python projects that make it easy for the developers to understand, as Python has gained popularity in the last few years. Furthermore, the above article has provided an extensive prediction of each category of code smells in Java projects and rules that can be applied to identify each one. However, our work adopts a broader perspective in that we do not focus on specific code smell types for Python development projects. Differences between these programming languages discussed and differences between the quality of code analysis.

2.2 Survey on Impact of Software Metrics on Software Quality

Software metrics are specifically important during the lifecycle of a software project as a measurement for development and requirement documents, designs, programs, and tests as Mrinal et al.(2012) [19] identified in their research work. They assist in managing the quality of the software produced and are logics that are easy to understand and are well defined and documented. Software metrics are classified into three types: operations management (OM), operations strategy (OS), and operations performance. Process metrics are concerned with the way how the software is developed whereas project metrics deal with the development progress and the improvement of the project charter. Product metrics are concerned with definitional parameters of the software product and facilitating the efficient running of software projects. There is no such thing as a generally accepted definition of the software quality paradigm; however, it can be described in terms of the user, manufacturing, product, and value perspectives. It is necessary to understand that it is subjective and the definition should be situation-based therefore no generic definition of software quality can be provided. As goals are time-bound, it becomes extremely important for them to be monitored to guarantee that they are accomplished within the defined time and conditions. The Boeing 777 project was a big leap in software quality management, as engineers were implementing, the same practices in all parts of the project and in regime reporting and monitoring of project progress and software quality.

As far as the classification into three categories of software metrics is concerned, however, the metrics for software quality and reliability can be considered in terms of software quality metrics as well as measurement of the outcome of specific software projects and there exists LOC(Lines of code) for such metrics. LOC is a software metric that is calculated based on the amount of source code to determine the size of a program. In particular, LOC is easy to measure and has the aspect of automation counting. But, it might have dead com code which is not useful. Metric problems arise because the software industry does not have any standardization on how to measure things. Source code metrics methodology, function point analysis and other object-oriented modeling techniques provide the same functionality of reusability and modifications but cannot be applied in other contexts and lack conversion rules. But in our current study software metrics' importance is expected to exhibit a greater increase since the industry leaders will start to increase the stringency of approaches for monitoring and enhancing the systems. Other avenues for further research lie in enhancing existing metrics.

2.3 Comparative Study of the Software Metrics for the Complexity and Maintainability of Software Development

Dr. Sonal Chawla and Gagandeep Kaur [4] (2013) explains that OO plays a critical role in modern systems as it helps to maintain software and solve the most common issues. It is noteworthy to mention that OO design includes all characteristics or qualities of software which means that the system objects can afford the given properties. This classifying approach has many positive aspects like reliability, whether you are developing a large system or a complex application, or a separate component, or whether you are designing a system or decomposing a substantial problem into separated objects. Quality engineering metrics are a vital tool in the planning stage, process improvement, quality control, reliability estimation, and customer satisfaction estimation. Software metrics are of two types namely static and object-oriented- used to analyze the code statically without actually running the software which provides a better insight into the security problem and detects deficiencies within the software. Actual Code (AC) and SLOC are used to measure program size, while CP and CC enhance comprehensibility and maintainability. Halstead Metrics are applied on various parameters such as operands and operators while Cyclomatic Metrics define a number of free paths using source code [4]. Dynamic metrics are created based on the results of objects during the execution, oriented towards object-oriented programs. The CK metrics suite is designed for measuring object-oriented programs and includes six metrics: Weighted Method Per Class (Wmc), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Object Class (CBC), Response of A Class (CA), etc. – Lack of Cohesion Of Methods (LCOM) and Methods for Object-Oriented Design (MOOD) [Chawla 2013,

p. 445]. How quality is measured has become more challenging in the software industry due to the transformation changes that the software industry has undergone. Surveys and studies need to be conducted in order to best choose the right metrics, how to ensure usefulness, how to use the metric and published value thresholds. This drives several benefits for the software and gives rise to high-quality software, higher reuse and reduction in the cost of maintenance.

Our paper has observed that Comment Percentage (CP) enhances multidimensional software quality and maintainability of the program while SLOC is a measure of the size of the program. The characteristics of the program are calculated through the metrics as stated in Halstead where the numbers of free paths are calculated using the methods in source code as in the cyclomatic complexity. Finally, for measuring Object Oriented program, we use CK metrics including the following Wmc- Weighted Method Per Class, Rfc- Response Of A Class and so on.

2.4 Effectiveness of software metrics for object-oriented system

Yeresime Suresh, Jayadeep Pati and Santanu Ku Rath (2012) – Department of Science and Humanities [23] focuses on the current research on the area of software metrics and their influence on evaluating the quality of the software systems, as well as the complexity and reliability of the O.O.S with the help of both the conventional and O.O.S metrics. Metrics are regarded as indispensable approaches employed at every step of the software engineering process by the author. Software metrics enable the analyst, designer, coder, tester, and manager by providing quantitative means to determine a specific characteristic of the system, component, or process characteristics that have been determined. This thorough investigation divides metrics into two major categories: Functional metrics; logical metrics; structural/size metrics; complexity metrics; design complexity metrics; conventional metrics; and; object-oriented metrics. These three journal papers are similar in terms of their focus on well-established metrics such as Cyclomatic Complexity, Size, and Comment Percentage [23] while the latter is mainly concerned with the Chidamber and Kemerer metrics suite, also used for evaluation of system reliability. To help understand the complexity and reliability of the system, the article used some of the above metrics to analyze a real and typical ATM software application. Thus, one of the major contributions of the paper is the interpretation of the collected metric values. This profound work provides a keen analysis of the consequences of elevating each metric value and encourages feasible actions for enhancing software design. For example, high Cyclomatic Complexity values indicate that there is a lot of potential flow through the program and therefore a need for a large number of test cases. The need to have to limit the depth of inheritance while developing is often indicated by a very high Depth of Inheritance Tree which may help in understanding how a system operates. As such, object-oriented metrics like coupling between objects (CBO), response for a class (RFC), and other measures on class quality and testing complexity are valuable. The research also captures how helpful they are in predicting system stability and failure likelihoods. It highlights the significance of determining fault proneness utilizing measurements like LCOM, LOC, and WMC, which have proven successful in spotting possible problems. In addition, it addresses how these measures may be used, including automated test case development and the use of neural networks to forecast system reliability. In summary, this study makes a substantial contribution to the body of knowledge by expanding on the critical function of software metrics in software development. It emphasizes how metrics can be used to improve software quality, gauge the complexity of designs, and foretell system stability.The guidelines and insights provided within this study empower software professionals with valuable tools to ensure the delivery of high-quality software products.

2.5 The Effects of Software Size on Development Effort and Software Quality

Zhizhong Jiang, Peter Naudé, Binghua Jiang's (2007) research article 'The Effects of Software Size on Development Effort and Software Quality' [11] carries out an in-depth exploration of this central area of software development economics and quality. It sums up the key ideas of the concept and stresses the strategic significance of the perspective, which revolves around viewing software development through the prism of economic and quality factors. It shows that software development is a complex economic process of production and the necessity to estimate software time and cost-effectively in order to achieve the desired project management results. It is able to differentiate the different effort estimation techniques and outlines their strengths and weaknesses and making the case for simpler, yet more accurately based models. It is evident that FP helps address concerns about size estimation, although there are several criticisms associated with these ideas, and the study is therefore crucial for addressing concerns about the quality of software systems as they get bigger. Using the ISBSG repository this paper discovers a major finding- that size and defects resonate positively and sound a message for better debugging and testing.[11] To summarize, this research provides a significant contribution to the software engineering field – it presents the multi-factorial nature of software size-effort-quality models in detail, highlighting the impact on practitioners and academics in the industry.

2.6 Maintainability of Object-Oriented Software Metrics with Analyzability

Satya et al. (2015) discusses the role of software maintainability as one of the factors in the development of excellent software.[18] The authors emphasize that, based on the standard ISI-9126, maintainability accounts for a significant percentage of software product quality. It stresses that properties such as Understandability, Analyzability, Reusability, Modifiability, Complexity, Durability, and Expandability can be used to measure maintainability. The paper is primarily aimed at developing models for measuring maintainability with special emphasis on Understandability, Modifiability and Analyzability. These characteristics are considered to be crucial for software maintainability in OOD. The authors note that critical choices for these parameters of maintainability are determined by design size and structural complexity metrics.

The authors of this research use 3 size metrics with 8 structural complexity metrics to prove their relation to modifiability, understandability, and analyzability. To better understand class diagrams they also introduce levels Trivial and Consequential to the sample data for the Maintainability factors. They fit Ln models for Understandability, Modifiability, Analyzability, and Maintainability and use the two-tailed t-test to identify significant metrics.[18] The significance of their suggested models is reiterated in the conclusion section of the paper as well as the high correlations of their models with actual values. It also has recommendations for further research on model enhancement, other factors and the evaluation of external quality attributes of software to demonstrate its primary concern with enhancing the maintainability and quality of software.

2.7 Study on Software Quality Factors and Metrics to Enhance Software Quality Assurance

Nigussu Bitew Kassie and Jagannath Singh (2020) discusses the use of SQA in scientific and medical fields. [12] Current research aims at identifying and mining the best metrics and parameters that contribute to software quality assurance. In order to determine these characteristics and metrics this research is based on: literature reviews and examining a large number of research works. The paper analyses the metrics for assessment and the essential areas of quality: functionality, dependability, usability, portability, maintainability, and efficiency. Some of these indicators are mean time to failure, defect density, mean client problems, and client satisfaction. There are also a number of Software Quality Models (SQM) that are also discussed in the study like McCall's Model, Boehm Model, Drome Model and the ISO 9126 Quality Model. It has a further discussion of goal question metrics (GQM), process metrics, and product metrics. The final section thus articulates the significance of SQA in different industries and calls for further research towards improving the measurement of software quality and factors to realize specific outcomes.

2.8 Predicting Code Smells and Analysis of Predictions: Using Machine Learning Techniques and Software Metrics.

Mhawish and co-authors (2020) suggests a machine learning and software metric-based technique for code smell prediction. The authors create 4 binary label datasets for 4 code smells including Data Class, God Class, Long Method, and Feature Envy as well as multi-label data containing multiple smells [15]. They use 6 machine learning algorithms in their experiment to examine the performance and they find that tree-based algorithms such as Random Forest and Gradient Boosted Trees give the best accuracy. The authors also use the improved feature selection method that employs genetic algorithms to enhance the accuracy of the model. They explain the models' predictions by using the LIME algorithm to find out the most important software metrics supporting the prediction results. The main conclusions are that predicting the occurrence of code smells has high potential with machine learning algorithms and that the size, complexity, coupling, and encapsulation metrics are important predictors of code smells. In summary, this paradigm proves to be effective in using machine learning for automated code smell detection. This paper describes the prediction of code smells using machine-learning algorithms and software metrics. The authors create binary labeling datasets of 4 code smells (Data Class, God Class, Long Method, Feature Envy) and multi-label datasets that include combinations of the smells. They also applied 6 machine learning algorithms and discovered that tree-based algorithms like Random Forest and Gradient Boosted Trees are the most appropriate and accurate algorithms They also used feature selection using a genetic algorithm and it significantly increased the performance of the models. The researchers employ the LIME algorithm to explain what the models predict and what their top software metrics are in supporting the predictions. The key findings for the paper are that machine learning has great potential for code smell prediction and adequately explained factors – size, complexity, cohesiveness, coupling and encapsulation. As a whole, this approach successfully incorporates machine learning into automatic code smell identification. It should be noted that both the research focuses on code smell prediction approach using machine learning. Both of them employ sophisticated machine-learning techniques to arrive at the best accuracy. There are also a few differences between the two studies. The above project author used binary–based datasets for 4 code smells to achieve accuracy and we used some metrics for quantification of a specific project like SLOC(Source Lines Of Code), and CP(Comment Percentage). Source Lines of Code (SLOC) is a measure used by computer programmers to estimate the size or complexity of software development. It's not directly applied to differentiate code smells though it can assist with identifying potential problems that may be associated with code smells in a codebase. We have used Comment percentage Because: It will make sure that the algorithm provides an accuracy of the algorithm.

2.9 Object-Oriented Software Quality Metrics

In the research, the authors made a critical review of conventional software metrics with a focus on the weaknesses of the metrics in the evaluation of the complexity of the object-oriented design, including the need for new measures directly dedicated to this field [10]. Despite their importance, traditional metrics, like Cyclomatic Complexity and Source Lines of Code are based on procedural characteristics and do not reflect object-orientation artifacts like inheritance and polymorphism. In an effort to address these shortcomings, this paper evaluates subject-object metrics, particularly the CK set of metrics, and examines metrics such as Weighted Methods per Class and Depth of Inheritance Tree in order to gain greater programme quality control. The authors suggest the need of additional metrics, Total Function Calls and Reuse Ratios, as a means to assess OO software with a particular overview of inherited methods and attributes. Experimental use of these new metrics to evaluate important quality characteristics such as understandability, maintainability and reusability of object-oriented systems has shown that they are a reliable measure of these essential quality criteria. As a result, the use of OOQ not only focuses on the object-oriented characteristics of a system but also utilizes some conventional metrics to develop a comprehensive method for quality evaluation.

2.10 Comparison of Related Studies

Metrics: Both static metrics (cyclomatic complexity, LOC) and object-oriented metrics (ATFD, WMC, RFC) are widely used to assess software quality and maintainability. However, metrics like LOC are mostly insufficient for identifying deeper issues in complex systems [19, 4]. On the other hand, object-oriented metrics offer better insights into modularity and class-level dependencies [1, 23].
Methodologies: Studies progressively implementing machine learning models (Random Forest, Gradient Boosting) for maintainability predictions [15, 1]. Although the ML techniques enhance predictive accuracy, most models are trained on Java datasets, limiting their universality to other languages.
Programming Languages: Most of the research focuses on Java-based projects while few studies are focused on other popular languages like Python [1]. This leaves a gap in understanding how object-oriented metrics apply to Python's growing ecosystem.
Limitations: It is widely observed that majority researches depends heavily on static metrics that fail to observe dynamic behaviors of software at runtime[23]. Furthermore, ML models trained on only one language datasets risk limited cross-platform applicability [11].

Chapter 3

Detailed Description of the Developed Tool or Python Script

It is a further developed script that helps in the downloading and analysis of the required GitHub repositories; extracting information and data from them, and generating excell-based reports with all the relevant and essential metrics etc. The script itself is depicted in detail in the figure below and the name and functions of all libraries used and a detail of each section are provided.

3.1 Libraries used

Python libraries help to ease many crucial operations including data analysis and data visualization, web scraping, image detecting or processing, machine learning model creation, textual information processing, etc. [9]. Python libraries are an array of fundamental sets of necessary skills that make it easier for the user not to write new codes [13].

1. Requests: The requests library is a human-friendly tool for making HTTP requests in Python. It makes it possible that a Python shell send an HTTP/1. Two requests using both GET and POST operations. This library enables us to make HTTP requests of any kind – and all while dealing with the responses in a natural and intuitive manner since it hides the complexity of HTTP request handling behind a simple yet powerful API. Furthermore, it supports SSL, performs cookies and sessions, implements data verification, etc., which makes it a crucial utility for working with web services and API. This is used in our research in order to download the GitHub repository when

an HTTP request is made to the GitHub API.

2. OS: The OS module in Python gives a way for making communication between the OS and the program. Through the os module, it remains very simple to manipulate or operate on the operating system and this also makes the code portable [5]. Entry and exit of the file system and managing the processes are taken care of by the module; os through operating system-dependent facilities like handling of paths and reading from or writing to the file system. It has a few mechanisms to interact with the underlying operating system such as renaming file paths or getting the OS environment variables or creating removing or moving directories. This module is required for hosting platform-independent system-level jobs pertaining to the use of the file system and OS-related operations.

3. Zipfile: The zip file module can be used for creating, reading, writing and extracting zip file archives. It provides an algorithm to support compression and decompression of files and can be utilized to support ZIP archives in Python. Large volumes of files and archives can be read or written as ZIP files if the ZIP64 feature is supported by the module you are using. It is also very useful for creating archives that contain files that will be distributed or stored efficiently in a group of files.

4. Openpyxl: openpyxl is a Python library for reading and writing Excel xlsx/xlsm/xltx/xltm based on xml standards. You can employ it for data manipulation that involves the creation, editing, and removing of data on an Excel file. With respect to the creation of compound spreadsheets, the library provides methods and data types appropriately dealing with formatting, charts and graphs as well as calculating formulas. The ability to automate Excel files is well-suited for jobs in data analysis or report generation.

5. Re: Regular expressions are a technique or tool for conducting text processing [2]. Regular expressions used with the Python programming language are the re-module. It is an easy-to-learn and use tool that has applications for searching by using regex patterns, matching of regex patterns, and manipulating strings with the regex patterns. The regular expressions are very useful for matching text patterns and the re-module can be used for complex pattern definitions such as character classes and quantifiers and so on. This module is relevant for text processing tasks, where string splitting, pattern matching, and substitution are performed.

6. Ast: Tools for working with Python abstract syntax trees are included in the module ast. This tool facilitates the programmatic parsing, processing, and modification of Python code by transforming source code structuration enabling an attacker to transform rce code to a tree-like structure for the code's syntax. This module is useful for the expression of the structure of

Python code in the form of an abstract syntax tree (AST), code generation, static code analysis, and many other tasks. It provides the tools to explore and to modify the tree, allowing for complex code navigation and code change operations.

7. Collections with counter: Hasable objects are counted using the class dict subclassed to Counter. It represents a simple method of determining the number of item appearances and is a subset of the collections module. This class is very useful when adding elements in a collection or iterating through an iterable and the number of each element needs to be counted or the number of elements that match some element in a given array needs to be found out. It gives arithmetics for the counters, frequency mode for specifying the most frequent elements, and increment mode for adding counts.

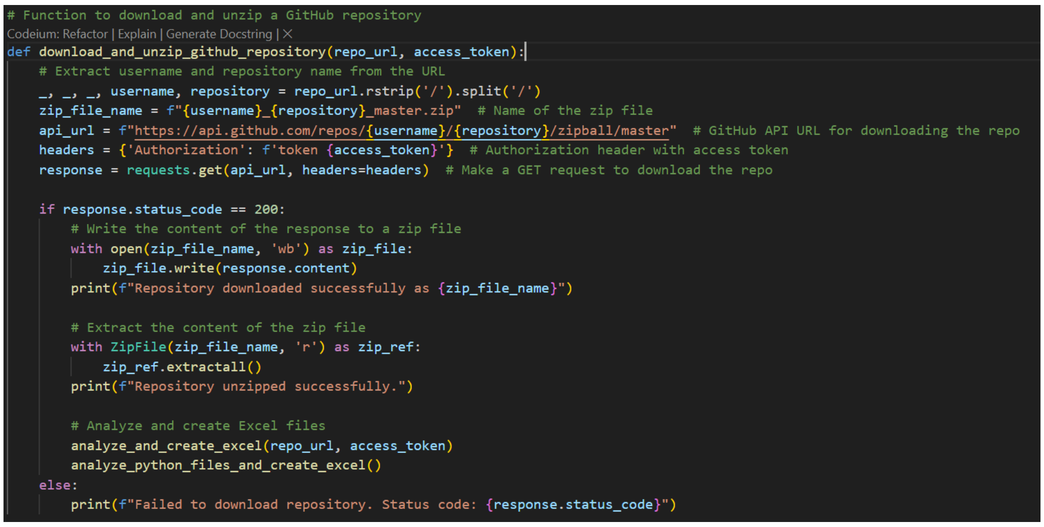
8. Pandas: It is the most used and the most common set of tools used for data computations in the data science field and matplotlib in Python. It contains approximately over 17 thousand comments on GitHub and has an engaged community of about 1200 members, It is widely used for data analysis and data cleaning tasks [20]. Pandas is an open-source library for Python programming tool which is highly powerful and flexible and easy to learn. When it comes to managing structured data it has data structures like a dataframe and series. Pandas – data analysis tool which was intended for practical use in real world. This is based on another Python library called Numpy. It is an important asset and it acts to deliver tools for data cleaning, joining, transforming, and visualization of data to data scientists and analysts.

3.2 Breaking down the script to achieve the desired outputs for the research

These functions have been used to make the tool and analyze the results of the research.

1. Download and unzip GitHub repository Function: It is this function that is left with the role of downloading and unzipping a GitHub repository. To download the repository as a ZIP file, it first decodes the repository name and username from the given URL of the repository on the GitHub website to form the GitHub API URL. The ZIP file is retrieved by the requests library, downloaded to the user's local folder and compressed using the zip file module. The first one, analyze and create excel and analyze Python files and create excel after extracting the repository, is to analyze and generate Excel files for analysis results. Further, the function ensures error handling

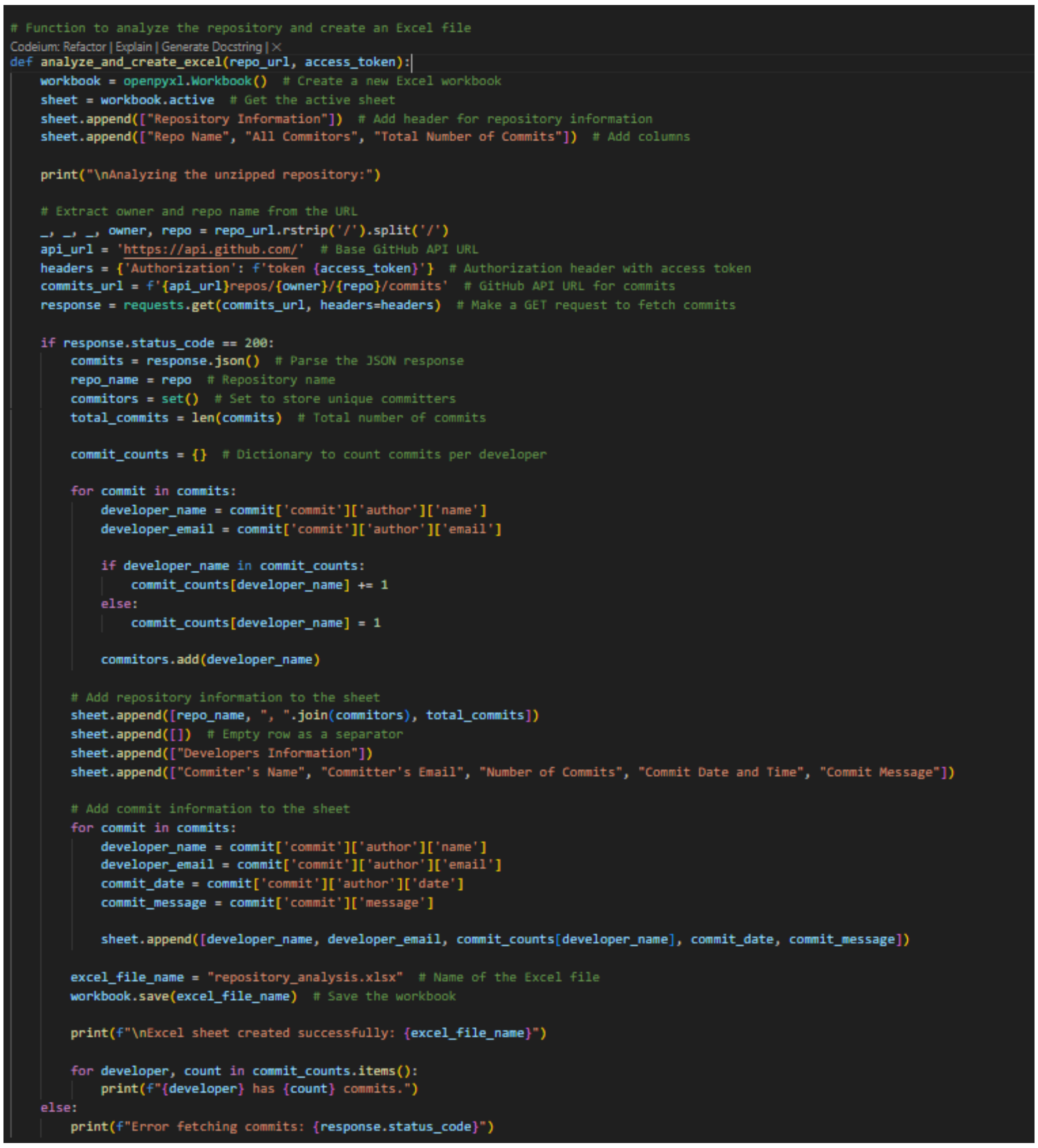
by checking that the HTTP response has a status of 200 implying a successful operation, and displays the appropriate messages accordingly.



```python
# Function to download and unzip a GitHub repository
def download_and_unzip_github_repository(repo_url, access_token):
    # Extract username and repository name from the URL
    _, _, _, username, repository = repo_url.rstrip('/').split('/')
    zip_file_name = f"{username}_{repository}_master.zip"  # Name of the zip file
    api_url = f"https://api.github.com/repos/{username}/{repository}/zipball/master"  # GitHub API URL for downloading the repo
    headers = {'Authorization': f'token {access_token}'}  # Authorization header with access token
    response = requests.get(api_url, headers=headers)  # Make a GET request to download the repo

    if response.status_code == 200:
        # Write the content of the response to a zip file
        with open(zip_file_name, 'wb') as zip_file:
            zip_file.write(response.content)
        print(f"Repository downloaded successfully as {zip_file_name}")

        # Extract the content of the zip file
        with ZipFile(zip_file_name, 'r') as zip_ref:
            zip_ref.extractall()
        print(f"Repository unzipped successfully.")

        # Analyze and create Excel files
        analyze_and_create_excel(repo_url, access_token)
        analyze_python_files_and_create_excel()
    else:
        print(f"Failed to download repository. Status code: {response.status_code}")
```

Figure 3.1: Download and unzip GitHub repository

2. Analyze and create excel Function: This process produces an Excel file with the repository content that the script analyzed based on the information it downloaded from the GitHub repository. New packs for Excel workbook and sheets are then created using openpyxl. The actual data is then extracted from Git Hub by using the API and passing the access token and specifying the repository URL. The function obtains data from the GitHub API is a number of commits per some committer, commit dates, commit messages, data on some repository, etc. Headers are created and new rows are inserted for each piece of information on an Excel spreadsheet. The function then proceeds to save the compiled data into an excel file called repository analysis. xlsx. A complete analysis of contributors and the changes made to the repository by each contributor of the repository is provided by this function.

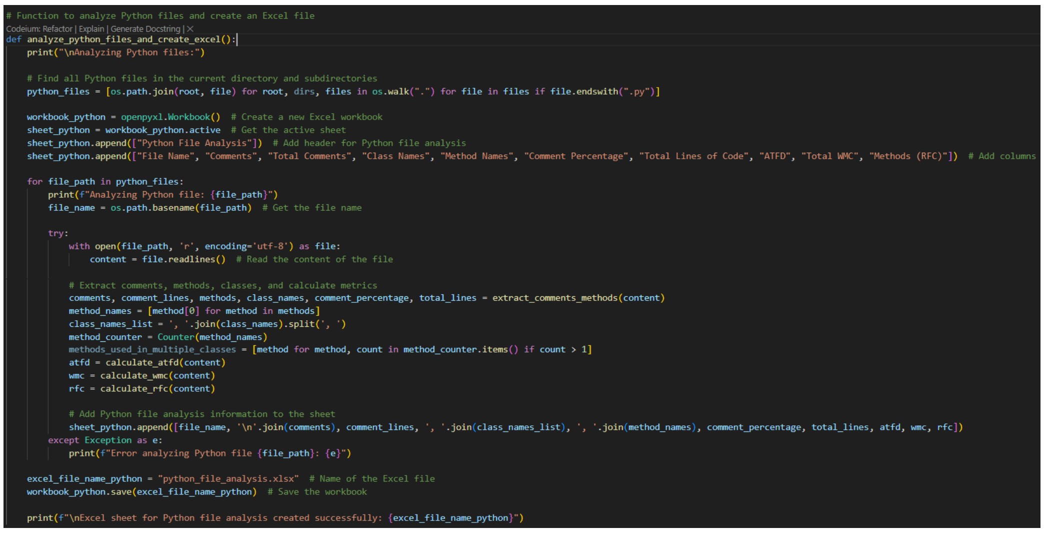


```python
# Function to analyze the repository and create an Excel file
Codeium: Refactor | Explain | Generate Docstring | ×
def analyze_and_create_excel(repo_url, access_token):
    workbook = openpyxl.Workbook()  # Create a new Excel workbook
    sheet = workbook.active  # Get the active sheet
    sheet.append(["Repository Information"])  # Add header for repository information
    sheet.append(["Repo Name", "All Commitors", "Total Number of Commits"])  # Add columns

    print("\nAnalyzing the unzipped repository:")

    # Extract owner and repo name from the URL
    _, _, _, owner, repo = repo_url.rstrip('/').split('/')
    api_url = 'https://api.github.com/'  # Base GitHub API URL
    headers = {'Authorization': f'token {access_token}'}  # Authorization header with access token
    commits_url = f'{api_url}repos/{owner}/{repo}/commits'  # GitHub API URL for commits
    response = requests.get(commits_url, headers=headers)  # Make a GET request to fetch commits

    if response.status_code == 200:
        commits = response.json()  # Parse the JSON response
        repo_name = repo  # Repository name
        commitors = set()  # Set to store unique committers
        total_commits = len(commits)  # Total number of commits

        commit_counts = {}  # Dictionary to count commits per developer

        for commit in commits:
            developer_name = commit['commit']['author']['name']
            developer_email = commit['commit']['author']['email']

            if developer_name in commit_counts:
                commit_counts[developer_name] += 1
            else:
                commit_counts[developer_name] = 1

            commitors.add(developer_name)

        # Add repository information to the sheet
        sheet.append([repo_name, ", ".join(commitors), total_commits])
        sheet.append([])  # Empty row as a separator
        sheet.append(["Developers Information"])
        sheet.append(["Commiter's Name", "Committer's Email", "Number of Commits", "Commit Date and Time", "Commit Message"])

        # Add commit information to the sheet
        for commit in commits:
            developer_name = commit['commit']['author']['name']
            developer_email = commit['commit']['author']['email']
            commit_date = commit['commit']['author']['date']
            commit_message = commit['commit']['message']

            sheet.append([developer_name, developer_email, commit_counts[developer_name], commit_date, commit_message])

        excel_file_name = "repository_analysis.xlsx"  # Name of the Excel file
        workbook.save(excel_file_name)  # Save the workbook

        print(f"\nExcel sheet created successfully: {excel_file_name}")

        for developer, count in commit_counts.items():
            print(f"{developer} has {count} commits.")
    else:
        print(f"Error fetching commits: {response.status_code}")
```

Figure 3.2: Analyze and create Excel Function

3. Analyze Python files and create an excel function: This function produces an Excel file in an aggregated form once it has finished analyzing the Python files in the downloaded repository. It starts by using the os and this is done by trying to obtain the output from the os. system function. module then uses the 'walk' function to find every Python file in the repository. Then it imports the openpyxl package and uses the workbook and sheet functions to create a new Excel workbook and sheet. The function calculates the number of various metrics, including ATFD (Access To Foreign Data),

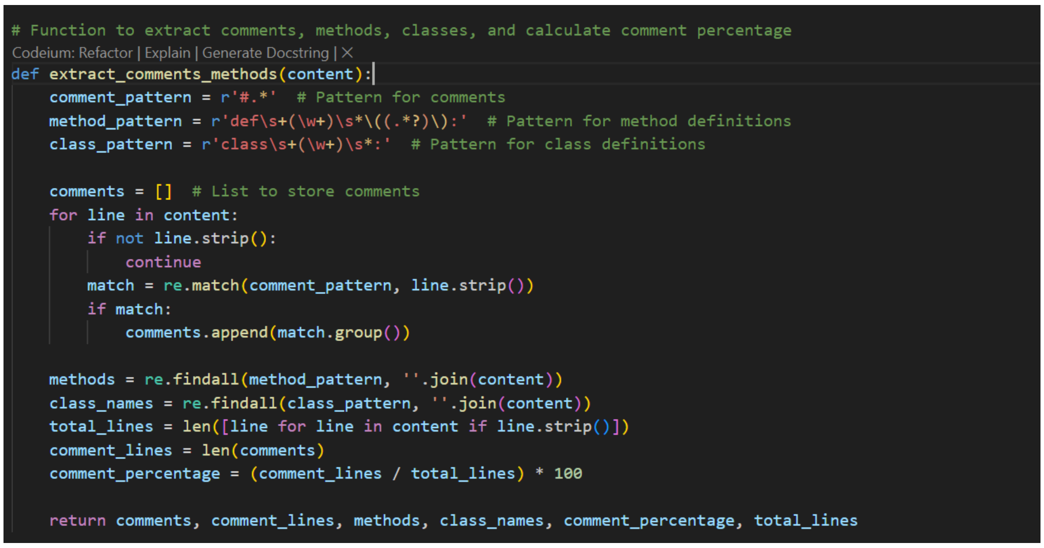
WMC (Weighted Methods per Class), and RFC (Response For a Class) for every Python file that contains extracted comments, methods, and classes. In order to determine these metrics, functions such as extract comments method, calculate atfd, calculate wmc, and calculate rfc are applied. The function places the headers and rows for each of the metrics as it summarizes together the analysis into the excel sheet. Finally, all the data is stored in a file named python file analysis in Excel. xlsx. This function explains the structure and coherency of the code on the use of Python code in the repository.



```
# Function to analyze Python files and create an Excel file
Codeium: Refactor | Explain | Generate Docstring | ×
def analyze_python_files_and_create_excel():
    print("\nAnalyzing Python files:")

    # Find all Python files in the current directory and subdirectories
    python_files = [os.path.join(root, file) for root, dirs, files in os.walk(".") for file in files if file.endswith(".py")]

    workbook_python = openpyxl.Workbook()  # Create a new Excel workbook
    sheet_python = workbook_python.active  # Get the active sheet
    sheet_python.append(["Python File Analysis"])  # Add header for Python file analysis
    sheet_python.append(["File Name", "Comments", "Total Comments", "Class Names", "Method Names", "Comment Percentage", "Total Lines of Code", "ATFD", "Total WMC", "Methods (RFC)"])  # Add columns

    for file_path in python_files:
        print(f"Analyzing Python file: {file_path}")
        file_name = os.path.basename(file_path)  # Get the file name

        try:
            with open(file_path, 'r', encoding='utf-8') as file:
                content = file.readlines()  # Read the content of the file

            # Extract comments, methods, classes, and calculate metrics
            comments, comment_lines, methods, class_names, comment_percentage, total_lines = extract_comments_methods(content)
            method_names = [method[0] for method in methods]
            class_names_list = ', '.join(class_names).split(', ')
            method_counter = Counter(method_names)
            methods_used_in_multiple_classes = [method for method, count in method_counter.items() if count > 1]
            atfd = calculate_atfd(content)
            wmc = calculate_wmc(content)
            rfc = calculate_rfc(content)

            # Add Python file analysis information to the sheet
            sheet_python.append([file_name, '\n'.join(comments), comment_lines, ', '.join(class_names_list), ', '.join(method_names), comment_percentage, total_lines, atfd, wmc, rfc])
        except Exception as e:
            print(f"Error analyzing Python file {file_path}: {e}")

    excel_file_name_python = "python_file_analysis.xlsx"  # Name of the Excel file
    workbook_python.save(excel_file_name_python)  # Save the workbook

    print(f"\nExcel sheet for Python file analysis created successfully: {excel_file_name_python}")
```

Figure 3.3: Analyze Python files and create an excel function

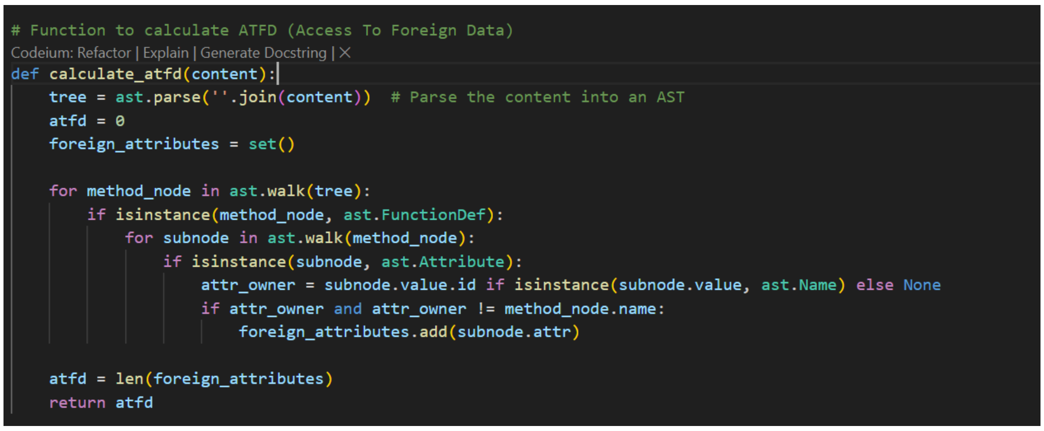
4. Extract comments methods function: This function takes any Python file input data and returns comments, methods, classes, and the percentage of comments associated with each one. It scans through the file content using regular expressions to grep comments, extract method definitions, or extract class definitions from it. This function uses the regular expression pattern to parse through each line of the file content and extract comments before coerced into a list. It also employs a different regular expression pattern to locate class declarations and method definitions. It tells the function how many lines the file has overall and the number of comments in there and provides a drop of how much of the file is made out of comments. With a comprehensive analysis of the file structure and documentation, it produces the extracted comments, comment lines, methods, class names, comment percentages, and total lines.



```
# Function to extract comments, methods, classes, and calculate comment percentage
Codeium: Refactor | Explain | Generate Docstring | X
def extract_comments_methods(content):
    comment_pattern = r'#.*'  # Pattern for comments
    method_pattern = r'def\s+(\w+)\s*\((.*?)\):'  # Pattern for method definitions
    class_pattern = r'class\s+(\w+)\s*:'  # Pattern for class definitions

    comments = []  # List to store comments
    for line in content:
        if not line.strip():
            continue
        match = re.match(comment_pattern, line.strip())
        if match:
            comments.append(match.group())

    methods = re.findall(method_pattern, ''.join(content))
    class_names = re.findall(class_pattern, ''.join(content))
    total_lines = len([line for line in content if line.strip()])
    comment_lines = len(comments)
    comment_percentage = (comment_lines / total_lines) * 100

    return comments, comment_lines, methods, class_names, comment_percentage, total_lines
```

Figure 3.4: Extract comments methods function

5. Calculate atfd function: The ATFD metric is a direct implementation of this formula, which measures the number of attributes in other classes directly accessed. Using the ast. to the parse function which parses the content of the file into an abstract syntax tree (AST). Next the function has to look for attribute accesses in the AST of methods in which the attributes are components of other classes. Quantifies these counts after putting all of these foreign attributes into a set. This is the number of different attribute clones used in the file and is the final ATFD score. It helps in establishing the coherence of classes in the code.

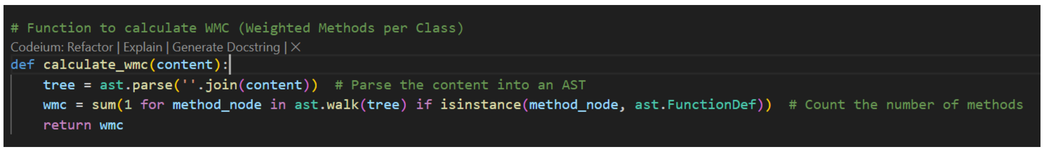


```
# Function to calculate ATFD (Access To Foreign Data)
Codeium: Refactor | Explain | Generate Docstring | X
def calculate_atfd(content):
    tree = ast.parse(''.join(content))  # Parse the content into an AST
    atfd = 0
    foreign_attributes = set()

    for method_node in ast.walk(tree):
        if isinstance(method_node, ast.FunctionDef):
            for subnode in ast.walk(method_node):
                if isinstance(subnode, ast.Attribute):
                    attr_owner = subnode.value.id if isinstance(subnode.value, ast.Name) else None
                    if attr_owner and attr_owner != method_node.name:
                        foreign_attributes.add(subnode.attr)

    atfd = len(foreign_attributes)
    return atfd
```

Figure 3.5: Calculate atfd function

6. Calculate wmc function:WMC is added into the suite as it previously

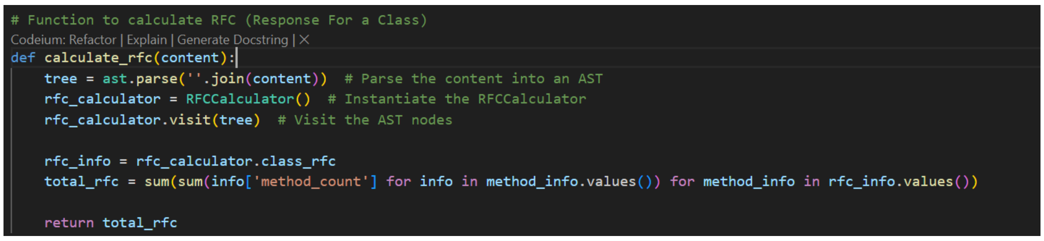
showed good performance in accurately predicting maintenance and testing efforts. This function determines WMC (Weighted Methods per Class), that is, the weight of a class is to a certain extent dependent on the number of methods it includes. Using the ast. tokenize function whereby treats the content of the file as a token stream and from this performs a translation to the abstract syntax tree. This means that the content of the file is converted into the abstract syntax tree and then the tree is traversed to count the number of method definitions. The function does this by finding the nodes that are objects that represent method definitions such as ‘ast. FunctionDef’ that can be found in every node in the AST using ast. walk. The overall total of method definitions in the file is the last WMC count. This metric measures an idea of the complexity of a class and it’s maintainability.



```
# Function to calculate WMC (Weighted Methods per Class)
Codeium: Refactor | Explain | Generate Docstring | ×
def calculate_wmc(content):
    tree = ast.parse(''.join(content))  # Parse the content into an AST
    wmc = sum(1 for method_node in ast.walk(tree) if isinstance(method_node, ast.FunctionDef))  # Count the number of methods
    return wmc
```

Figure 3.6: Calculate wmc function

7. Calculate rfc function: This function uses the RFC Calculator class to calculate the RFC Response For a Class (RFC) metric. Using the ast. The content of the file is run through the parse function the first time the content is parsed into an abstract syntax tree (AST). Then the function creates an instance of the RFC Calculator class and passes it to the AST nodes. The RFC metric for every class is calculated by the RFC Calculator metric which begins by walking through every node in the AST to get information about the methods encoded within each class as well as the method calls that exist within these classes. RFC index for the file is calculated by summing the RFC values for each class. This metric helps in understanding the possible complexity of a class interface and its relationship to other techniques.



```
# Function to calculate RFC (Response For a Class)
Codeium: Refactor | Explain | Generate Docstring | ×
def calculate_rfc(content):
    tree = ast.parse(''.join(content))  # Parse the content into an AST
    rfc_calculator = RFCCalculator()  # Instantiate the RFCCalculator
    rfc_calculator.visit(tree)  # Visit the AST nodes

    rfc_info = rfc_calculator.class_rfc
    total_rfc = sum(sum(info['method_count'] for info in method_info.values()) for method_info in rfc_info.values())

    return total_rfc
```

Figure 3.7: Calculate rfc function

3.3 Script Execution Process

The following query is started by asking the script to ask the user to enter GitHub repository URL and the GitHub access token. It is important that this step is interactive so that the script is able to access and download the target repository and to ensure that the script is not missing the required data it needs to run. The access token ensures the script is authenticated and helps the script bypass access restrictions or limits set by GitHub. Repository name is identified by the particular URL.

After having entered the information required by the script the download and unzip GitHub file executes the function download and unzip GitHub repository. In this function, the needed API endpoint is created and the repository is downloaded as a ZIP file by processing the URL to obtain the right username and repository name. Next, it uses the requests library to perform an HTTP GET request to the endpoint and add the access token in the headers. This function reads from the ZIP archive and writes it to the local file system and then extracts the ZIP using the zipfile module if the status code it is 200 indicating a successful response. The terminal commands and file and directory structure of the repository are replicated on the local machine to perform further analysis.

When the repository is extracted the function calls to analyze and create excel performs two important analysis procedures and the analyze Python and create excel to start the two analysis procedures. This function is responsible for analyzing the information in the repository metadata such as commit history and contributor details, and creates it as an excel report. Some of the key features include using the openpyxl library to collect the commit information in an excel based workbook after retrieving the commit data from the GitHub API and manipulating the information collected to obtain the author names, their emails, dates, and messages of the last 10 commits. This analysis can also be used to identify the total commits and contributors in each repository, as well as to provide information about how many commits and from whom the commits came from the repository. The functionality of this is achieved from saving the report in an excel file called repository analysis. xlsx.

Excel involves creating spreadsheets for the repository files and analyzing the given Python files at the same time. It used OS to interact with the directory structure. subprocess. walk function to locate all files with .py extensions. It scans the content of each file and parses the regular expressions to obtain the comments, the method definitions, and the class definitions. Moreover, it analyses the code after translating the files into an abstract syntax tree (AST) and using the information about the code structure to calculate the WMC (Weighted Methods per Class), RFC (Response For a Class), and ATFD (Access To Foreign Data). These metrics provide meaningful information about the level of complexity, maintenance, and quality of the code. The outputs are stored as a Python file called as analysis. One is to save the file in xlsx and combine it with another Excel file.

Two things that the script is designed to do that would benefit the user and give feedback on the process; one is to attempt to check and make sure that the script detects and fixes any errors in the process and another is, that the script should be able to tell the user what part of the processes is currently being performed. For example, it displays messages like downloading the repository successful and unsuccessful download of the repository and the HTTP status code on responses. It might also catch some exceptions that may arise when this script is in the process of reading from or writing to files so that the process may proceed in a kind of civilized way.

To summarize, the execution of the script consists of several interrelated steps: the compilation of all the feedback from the user, the download and extraction of the repository, the processing of the contents and metadata of the file, and the preparation and generation of a comprehensive report. The process is entirely automated using libraries for accurate identification in addition to repetitive tests of the GitHub repository using methodical approach. The result of this includes two excel docs that not only provide a summary of info for the activity and quality of the code in the repository but also provide more detailed description of the respective subject.

Chapter 4

Comparison of Our Custom Python Tool with Open source Python libraries Online (Pylint and Radon)

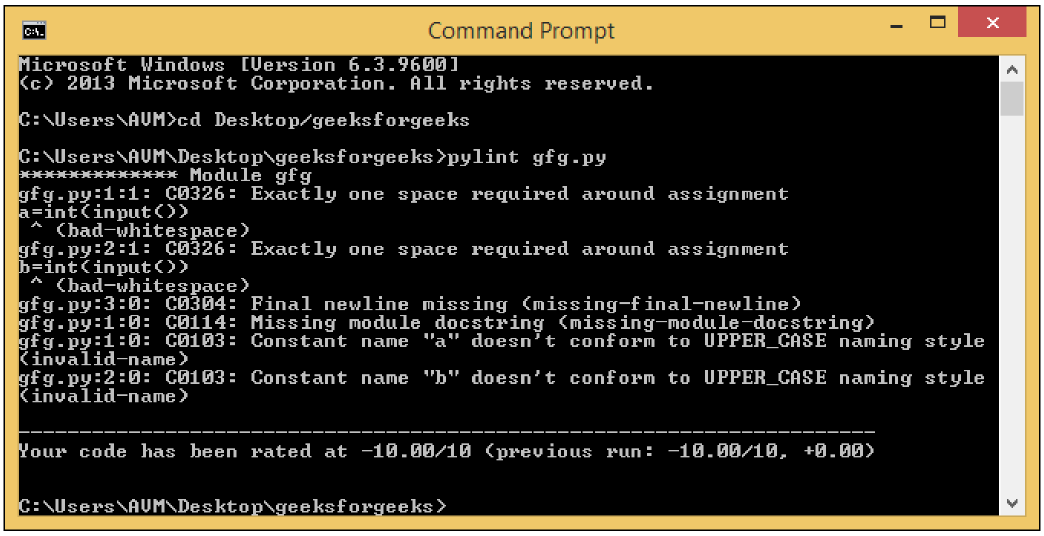
In this chapter, we present a comparison between our custom designed Python tool and two open-source Python-based code analysis tools: Pylint and Radon that have risen to great popularity. While Pylint and Radon are recognized in the field as tools designed to help developers in maintaining proper standards of code quality and in performing maintenance, which encompasses a range of tools geared at error detection, code complexity analysis, and code improvement suggestions, our tool is restricted to a particular scope of research. This tool is somewhat similar to these two but also has additional features. In this chapter, we wish to present the points of development and variation of the three solutions in question.

4.1 Pylint

Pylint is a code analysis tool that aims to assist programmers and help them conform to coding standards of the Python language, as well as to find and fix bugs, and enhance the overall quality of the written code[7].The detailed study of implementation suggests that by making the code quality checks automatic, maintaining uniform standards, and providing feedback instantly, Pylint can contain the developers' context-gathering and code reviews to a minimum extent[24]. It is more of a tool which enforces PEP 8 coding standard compliance and suggests how the code can be restructured.

Major characteristics include bug detection, code quality rating,controllable through config files and creates Pylint plugins. The same reasons bring popularity when it comes to Pylint's use in combined development environments and continuous integration / continuous deployment (CI/CD) processes.

In the pylint 2.4.4 version, pylint generates a report as shown below. Messages might change depending on the version[8].



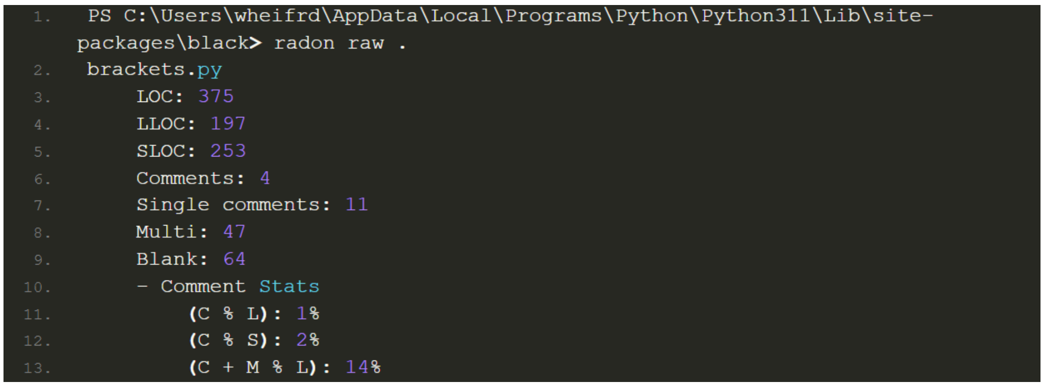
```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\AVM>cd Desktop/geeksforgeeks

C:\Users\AVM\Desktop\geeksforgeeks>pylint gfg.py
************** Module gfg
gfg.py:1:1: C0326: Exactly one space required around assignment
a=int(input())
 ^ (bad-whitespace)
gfg.py:2:1: C0326: Exactly one space required around assignment
b=int(input())
 ^ (bad-whitespace)
gfg.py:3:0: C0304: Final newline missing (missing-final-newline)
gfg.py:1:0: C0114: Missing module docstring (missing-module-docstring)
gfg.py:1:0: C0103: Constant name "a" doesn't conform to UPPER_CASE naming style
(invalid-name)
gfg.py:2:0: C0103: Constant name "b" doesn't conform to UPPER_CASE naming style
(invalid-name)

------------------------------------------------------------------------
Your code has been rated at -10.00/10 (previous run: -10.00/10, +0.00)


C:\Users\AVM\Desktop\geeksforgeeks>
```

Figure 4.1: Pytint

4.2 Radon

The opposite of this, is that Radon is a complexity measurement tool written in Python[3]. It provides several metrics calculated for the maintainer's convenience to spot development code that is complicated enough so as to be unmaintainable or full of bugs.Radon is a popular utility for the Python programming language that examines and evaluates systems of codes with the intention of gaining several code metrics such as: raw ones, Cyclomatic complexity, Halstead measures, maintanence measurements, etc[21]. Among Radon's functions are computations of cyclomatic complexity and Halstead's metrics, as well as the Maintainability index, and also basic counts like lines of code and comments. It is flexible with regard to the types of outputs offered including JSON and XML among others and also serves as a command-line tool and a library.

Here is a sample output of radon for a Python file[16]:



```
PS C:\Users\wheifrd\AppData\Local\Programs\Python\Python311\Lib\site-
packages\black> radon raw .
brackets.py
    LOC: 375
    LLOC: 197
    SLOC: 253
    Comments: 4
    Single comments: 11
    Multi: 47
    Blank: 64
    - Comment Stats
        (C % L): 1%
        (C % S): 2%
        (C + M % L): 14%
```

Figure 4.2: Radon

4.3 Our Custom Python Tool

The offered solution stands out as a GitHub repository analysis tool capable of doing much more than Pylint or Radon. It lets you download and analyze the contents of Python files located in GitHub repositories and carries out the extraction of certain parameters in the form of a report in Excel sheets. In these regards, it estimates such indexes as ATFD, WMC, RFC, which is the most informative when it comes to the analysis of the structure of Python files, especially the methods, classes, comments, etc.

4.4 Comparative Analysis

This section highlights the similarities and differences of the solutions.

Similarities

1. Code Analysis: Our tool, Pylint and Radon are capable of performing the static code analysis of Python files. The common aim of all the tools is facilitations of high-quality code by finding problems and giving suggestions on how to overcome the problems concerning complexities maintainability or standards.

2. Metrics Calculation: The tool, Pylint and Radon used in the study measures a range of metrics for code quality assessment. Although the met-

rics are different, the purpose is still to offer metrics that will assist users to write better code.

3. Extensibility: All of the tools can be used as a part of a bigger workflow, whether it is a CI or a research project, hence they can be employed in different coding contexts and for different purposes.

4. Python specific: The tools in question are meant for analyzing code written with the Python programming language thus presenting tailored tools for Python developers.

Differences

1. Scope:

- Our tool: The main aim is conducting the analysis of GitHub repositories. This includes the option to grab the repositories and pull out more particulars related to the study, especially from the files written in Python.
- Pylint: The major concern of Pylint is detection of bugs, formatting the code, and following the rules of PEP 8.
- Radon: Radon is concerned with the application of eight Halstead metrics and cyclomatic complexity among other complexity metrics which makes it suitable for finding the coded areas of high complexity.

2. Formats of Output:

- Our Tool: Produces Excel reports that are easier to use for the stakeholders who may not be used to using code analysis tools.
- Pylint: By design, handles its results mainly by writing them to console or text files suitable for CI pipelines.
- Radon: Provides support for different output options such as JSON and XML which allows for some level of use in analysis or combining with other tools.

3. Specific Metrics:

- Our Tool: Provides advanced object oriented measures like ATFD, WMC and RFC, which are not included in Pylint or Radon.

- Pylint: Offers more arguments related to coding practices and bug finding than those aimed at complexity calculations.There are restrictions on some higher-level issues as well: for instance, the maximum number of lines in a function or maximum number of methods a class can have[25].
- Radon: Has its main function computing cyclomatic complexities, Halstead metrics, and Maintainability Index that seek to evaluate the maintainability of the code and sections of it which pose problem areas.

4. GitHub Integration:
   - Our Tool: Has the capability to download and examine repositories from Github internally.
   - Pylint and Radon: No direct integration with coded repositories id provided. These softwares perform local code analysis but can be incorporated with the github ecosystem with extra configuration.
5. Comment Analysis:
   - Our Tool: Supports comment extraction and analysis from code files in Python which is useful in many research situations where one seeks to comprehend the purpose of coding or the explanations within the codes themselves.
   - Pylint: Provides the ability to align comments with styles defined in PEP 8 but does not provide any in-depth analysis.
   - Radon: Offers only counts of comments present in the analyzed block of text without analyzing the intelligibility or structure of the comments.
6. Customization
   - Our Tool: Is an intermediate level tool that is meant more for specific research interests, especially those aimed at analyzing GitHub repositories and extracting precise measures.
   - Pylint: Provides extensive adjustment of the tool using external configuration files to facilitate adoption of varying coding practices.
   - Radon: Offers some of the customizations, up to altering the metric and its configuration.The primary benefit of using Pylint is its flexibility in terms of configuration and customization. the config file and one can go ahead to create complex configurations that ensures the tenets are followed all over the codebase[17].

Other Observations

1. **Specialization:** Our tool seems to be more of a research tool as it focuses on features such as pull and analysis of GitHub repositories, reporting on their contents and metrics. Pylint and Radon are very similar but these tools are more for performing code quality checks.

2. **Integration:** This tool being a one-stop shop for repository analysis and report generation is an advantage. However combining the tools Pylint and Radon can also be effective owing to the high trust they have gained in the Python domain.

3. **Unique Features:** The inclusion of the ability to pull and examine entire GitHub repositories, which is one of the useful features of our tool and could help researchers and developers working on massive projects in the long run.

4. **Reporting:** The reporting functionality of our tool based in Excel could be made more user-friendly to non-technical users since it demonstrates results of code analysis in an orderly form. On the other hand, Pylint and Radon are command line tools, which are more technical.

5. **Maintenance:** Owing to the fact that our tool is a bespoke one, it always needs maintenance and upgrades. However, Pylint and Radon are supported by users who regularly upgrade and fix the software and ensure that it works with any new releases of Python.

In conclusion, it can be seen that although these three instruments have a code analysis function, their scope and purposes are quite different from each other. Our custom built Python tool has superior capabilities in analyzing and reporting on GitHub repositories than Pylint and Radon which are more generic tools focused at assessing code quality. However, for certain research activities that entail extensive analysis of code repositories, our tool is useful. However, due to their vast customization possibilities and community support, Pylint and Radon are priceless in wider Python development contexts.

Chapter 5

Description Of The Models

The Python tool developed for this study is a comprehensive tool for analyzing object-oriented programming (OOP) based Python repositories on GitHub. It extracts valuable information from Python files in the repository, such as:

1. Number of commits 2. Developer information 3. Percentage of comments 4. Various code metrics: - Lines of Code (LOC) - Access to Foreign Data (ATFD) - Weighted Methods per Class (WMC) - Response for a Class (RFC)

The tool utilizes the abstract syntax tree (AST) module to parse Python code and employs regular expressions for extracting relevant information. It creates Excel sheets that summarize the distribution and database developer information, as well as an analysis of individual Python files. The analysis includes metrics like content percentage, ATFD, WMC, and RFC.

After creating the dataset from the Excel sheets, the researchers used regression models to predict indicators like "LOC" based on features like "ATFD", "WMC", and "RFC". The regression models employed include:

1. RandomForestRegressor 2. DecisionTreeRegressor 3. KNeighborsRegressor 4. GradientBoostingRegressor

The integration of these machine learning models enriches the analysis by providing a predictive dimension to understanding how various metrics interact in a Python database. This approach allows researchers to gain insights into the relationships between different code metrics and their impact on the overall project.

By leveraging the Python tool and machine learning techniques, the study aims to provide a comprehensive understanding of OOP-based Python repositories on GitHub. The extracted information and predictive models can help developers and researchers analyze code quality, identify potential issues, and make informed decisions during the development process.

Chapter 6

Description of the Data

Firstly, we have created a summary file with all repository information such as total developer number, number of classes and methods, averages and totals of comment percentage and software metrics like RFC, WMC, ATFD, and LOC of 20 projects.

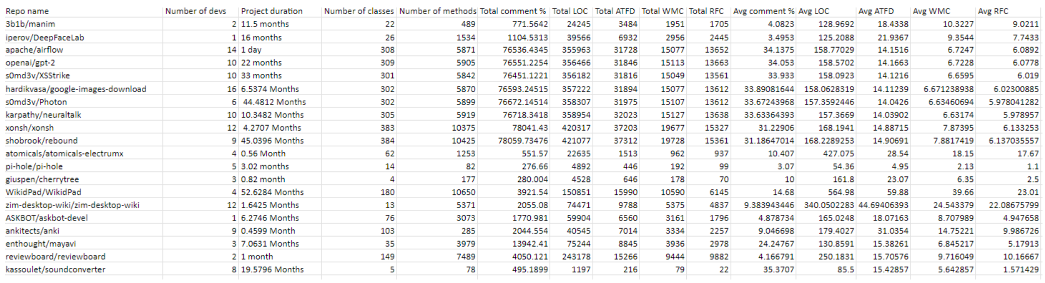
Secondly, we have created our main dataset by merging 20 Python file analysis sheets which we generated from the tool into a single dataset with 21,563 entries, offering insights into software projects. It focuses on individual Python files of each repository, providing insights into code metrics like total lines of code, ATFD, WMC, RFC, LOC, and comment percentages.

RFC (Response For a Class): An indicator of the number of methods that can be called in response to a message sent to an object of that class. This encompasses all the operations that are available in the class, including operators that are in the subclasses as well as the superclass. They defined that if the RFC value is high, the class would be more complex and may take more effort to comprehend and test.

WMC (Weighted Methods per Class): Sums up the value of the complexity of all the methods contained within a particular class. Complexity can also be measured based on the number of decision points within a method. A higher WMC implies that the class has more functionality but it could also be complex in maintenance and relatively harder to understand.

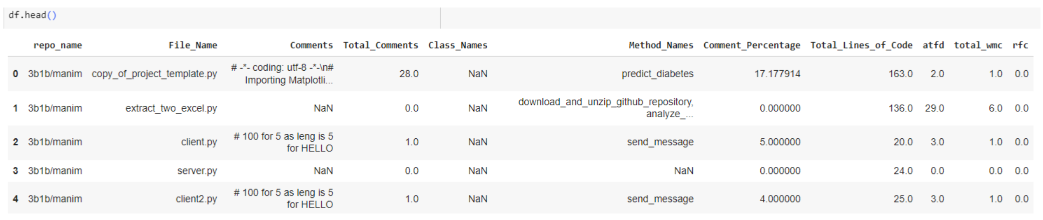
ATFD (Access to Foreign Data): Calculates the count of distinct attributes from other classes which are referenced directly or by the help of "getters" and "setters". It measures how many foreign attributes are employed by the class [14], indicating the extent of dependency on external data. High ATFD levels could indicate a greater interaction and a lack of encapsulation between the classes, and this hinders the modularity and might lead to the generation of more bugs during the maintenance phase.

LOC (Lines of Code): The simplest and most direct measure is to quantify only the number of lines of the actual code of the program. This metric is used to measure the extent of a software project or, to some extent the size of the project. As the complexity of the codebase increases, it becomes even more difficult to manage and as such may contain many more defects.



Repo name	Number of devs	Project duration	Number of classes	Number of methods	Total comment %	Total LOC	Total ATFD	Total WMC	Total RFC	Avg comment %	Avg LOC	Avg ATFD	Avg WMC	Avg RFC
3b1b/manim	2	11.5 months	22	489	771.5642	24245	3484	1951	1705	4.0823	128.9692	18.4338	10.3227	9.0211
iperov/DeepFaceLab	1	16 months	26	1534	1104.5313	39566	6932	2956	2445	3.4953	125.2088	21.9367	9.3544	7.7433
apache/airflow	14	1 day	308	5871	76536.4345	355963	31728	15077	13652	34.1375	158.77029	14.1516	6.7247	6.0892
openai/gpt-2	10	22 months	309	5905	76551.2254	356466	31846	15113	13663	34.053	158.5702	14.1663	6.7228	6.0778
s0md3v/XSStrike	10	33 months	301	5842	76451.1221	356182	31816	15049	13561	33.933	158.0923	14.1216	6.6595	6.019
hardikvasa/google-images-download	16	6.5374 Months	302	5870	76593.24515	357222	31894	15077	13612	33.89081644	158.0628319	14.11239	6.671238938	6.02300885
s0md3v/Photon	6	44.4812 Months	302	5899	76672.14514	358307	31975	15107	13612	33.67243968	157.3592446	14.0426	6.63460694	5.978041282
karpathy/neuraltalk	10	10.3482 Months	305	5919	76718.3418	358954	32023	15127	13638	33.63364393	157.3669	14.03902	6.63174	5.978957
xonsh/xonsh	12	4.2707 Months	383	10375	78041.43	420317	37203	19677	15327	31.22906	168.1941	14.88715	7.87395	6.133253
shobrook/rebound	9	45.0396 Months	384	10425	78059.73476	421077	37312	19728	15361	31.18647014	168.2289253	14.90691	7.8817419	6.137035557
atomicals/atomicals-electrumx	4	0.56 Month	62	1253	551.57	22635	1513	962	937	10.407	427.075	28.54	18.15	17.67
pi-hole/pi-hole	5	3.02 months	14	82	276.66	4892	446	192	99	3.07	54.36	4.95	2.13	1.1
giuspen/cherrytree	3	0.82 month	4	177	280.004	4528	646	178	70	10	161.8	23.07	6.35	2.5
WikidPad/WikidPad	4	52.6284 Months	180	10650	3921.54	150851	15990	10590	6145	14.68	564.98	59.88	39.66	23.01
zim-desktop-wiki/zim-desktop-wiki	12	1.6425 Months	13	5371	2055.08	74471	9788	5375	4837	9.383943446	340.0502283	44.69406393	24.543379	22.08675799
ASKBOT/askbot-devel	1	6.2746 Months	76	3073	1770.981	59904	6560	3161	1796	4.878734	165.0248	18.07163	8.707989	4.947658
ankitects/anki	9	0.4599 Month	103	285	2044.554	40545	7014	3334	2257	9.046698	179.4027	31.0354	14.75221	9.986726
enthought/mayavi	3	7.0631 Months	35	3979	13942.41	75244	8845	3936	2978	24.24767	130.8591	15.38261	6.845217	5.17913
reviewboard/reviewboard	2	1 month	149	7489	4050.121	243178	15266	9444	9882	4.166791	250.1831	15.70576	9.716049	10.16667
kassoulet/soundconverter	8	19.5796 Months	5	78	495.1899	1197	216	79	22	35.3707	85.5	15.42857	5.642857	1.571429

Figure 6.1: Summary of the repositories



```
df.head()
```

	repo_name	File_Name	Comments	Total_Comments	Class_Names	Method_Names	Comment_Percentage	Total_Lines_of_Code	atfd	total_wmc	rfc
0	3b1b/manim	copy_of_project_template.py	# -*- coding: utf-8 -*-\n# Importing Matplotli...	28.0	NaN	predict_diabetes	17.177914	163.0	2.0	1.0	0.0
1	3b1b/manim	extract_two_excel.py	NaN	0.0	NaN	download_and_unzip_github_repository, analyze_...	0.000000	136.0	29.0	6.0	0.0
2	3b1b/manim	client.py	# 100 for 5 as leng is 5 for HELLO	1.0	NaN	send_message	5.000000	20.0	3.0	1.0	0.0
3	3b1b/manim	server.py	NaN	0.0	NaN	NaN	0.000000	24.0	0.0	0.0	0.0
4	3b1b/manim	client2.py	# 100 for 5 as leng is 5 for HELLO	1.0	NaN	send_message	4.000000	25.0	3.0	1.0	0.0

Figure 6.2: First few lines of the dataset

Chapter 7

Preliminary Analysis

7.1 Initial Data Analysis

7.1.1 Relationships Between Number of Developers, Comment Percentage, and Project Duration

In this section, we aim at establishing correlation between Number of Developers, Comment Percentage and Project Duration. These metrics helps us understand relative size of teams and efforts towards documentation and the period that projects were completed.

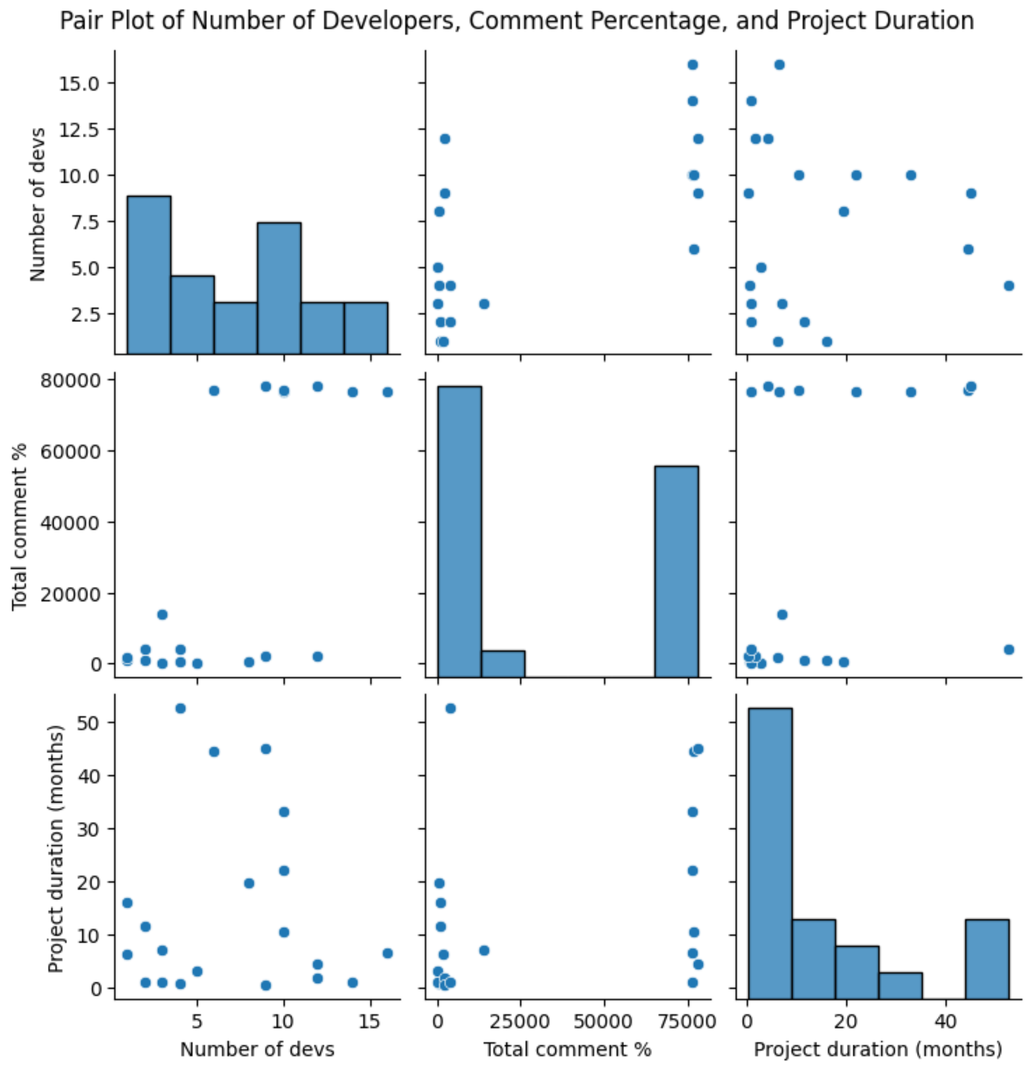


Figure 7.1: Pair Plot of Number of Developers, Comment Percentage, and Project Duration

The pair plot presents the relationships among Number of Developers, Comment Percentage, and the Project Duration. The correlation between Number of Developers and Comment Percentage is visible which suggests that larger teams tend to produce more detailed documentation. However, there is weak relationship between Project Duration and Number of Developers which indicates that larger teams do not necessarily lead to shorter or longer timelines of project.

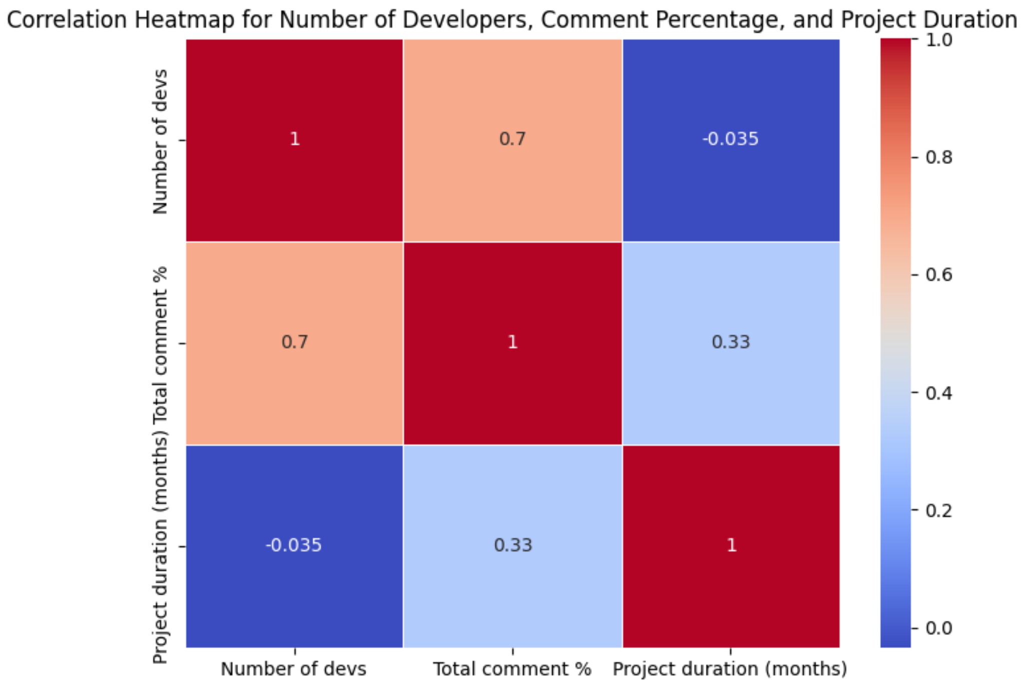


Figure 7.2: Correlation Heatmap of Number of Developers,Project Duration, and Comment Percentage

The correlation heatmap also shows the relationships between the Number of Developers, Comment Percentage, and the Project Duration. A strong positive correlation between Number of Developers and Comment Percentage highlights the impact of team size on documentation quality. However, the relationship between Project Duration and Number of Developers is weak, reinforcing that project length is not significantly influenced by the number of developers involved.

Discussion: Both visualizations show that team size has a notable effect on documentation quality but it does not significantly affect the length of the project.

7.1.2 Relationships Between Complexity Metrics and LOC

This section explores the relationships between the key complexity metrics: WMC (Weighted Methods per Class), RFC (Response for a Class), and ATFD (Access to Foreign Data) and LOC (Lines of Code). It focusing on their interactions and how they affect code size and complexity.

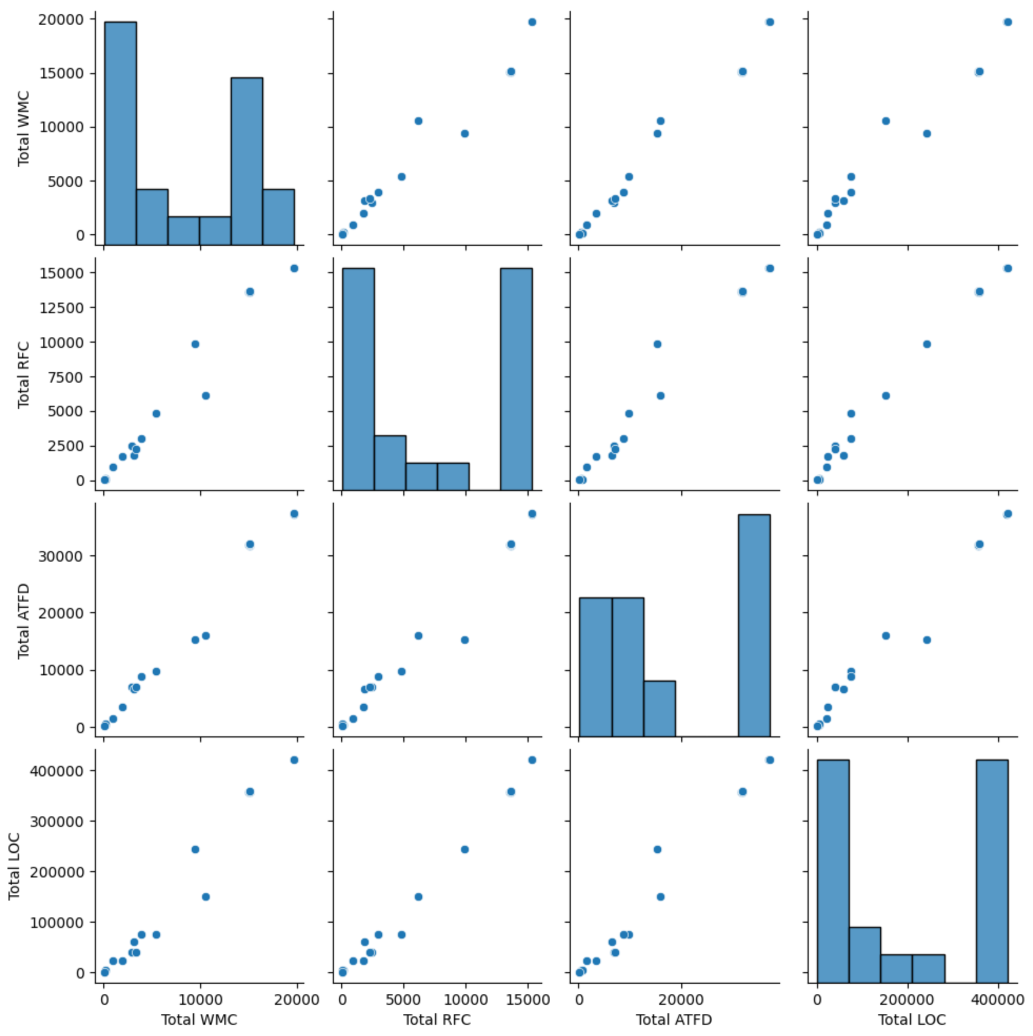


Figure 7.3: Pair Plot of WMC, RFC, ATFD, and LOC

The pair plot provides a detailed view of the pairwise relationships between WMC, RFC, ATFD, and LOC. Positive correlations are observed between WMC and RFC which indicates that if class complexity increases, method coupling also tends to increase. Also, the relationships between LOC and the complexity metrics (WMC and RFC) show that larger codebases tend to indicate higher complexity, which can make the system more challenging to maintain.

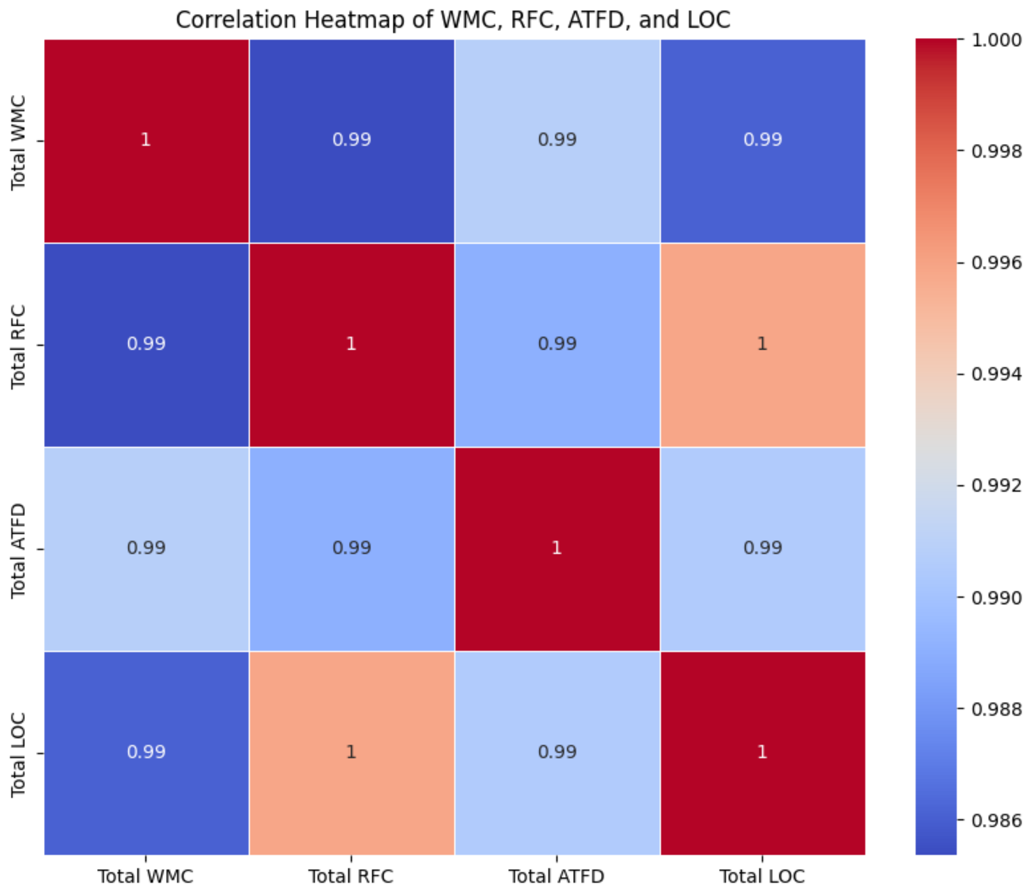


Figure 7.4: Correlation Heatmap of WMC, RFC, ATFD, and LOC

The correlation heatmap numerically presents the relationships between the four metrics using correlation coefficients. Strong correlations (near 0.99) are observed between WMC, RFC, ATFD, and LOC, confirming that these metrics are tightly coupled. As complexity increases, the codebase size (LOC) grows proportionally which highlights the importance of managing complexity early in the development process to maintain project scalability and prevent risks.

Discussion: The pair plot and correlation heatmap both show that the increasing class complexity and method coupling has a strong correlation with the size of codebase. It indicates that as projects grow, the internal complexity also grows, which could complicate future maintenance. The findings from these visualizations will be further explored in the next section through regression analysis to determine how well these complexity metrics predict project growth and scalability.

7.2 Applications of Regression Models

The second analysis is to find relations between metrics. We tried to find some correlations between the software metrics. Such as LOC vs WMC, ATFD and RFC. But in this case, we used the big dataset and found a positive result here.

We applied regression models such as RandomForestRegressor, DecisionTreeRegressor, KNeighborsRegressor, and GradientBoostingRegressorto our dataset in order to predict "total lines of code" based on ATFD, WMC and RFC.

RandomForestRegressor

RandomForestRegressor is one of the complex machine learning algorithm that performs decision trees and at the time of training, it produces several trees and finally outputs the average prediction of all the trees. It is one of the most commonly used methods that are applied for regression by using an ensemble approach [22]. This approach is useful for making the model more accurate and avoiding over-fitting to maximize accuracy and robustness for complex datasets where the relationship between features is not linear. The strength of the model is that it is able to handle data with a large number of features. It can explain non-linear patterns that are not obvious by a simple linear relationship. RandomForestRegressor is also one of the best methods for determining the contribution of each feature to the prediction and hence allows one to understand which metrics have the most influence on the total LOC. The model has an R-squared of 0. 92 indicates that it can explain about 92 percent of the total LOC in the projects. The high level of performance shows that it is efficient in detecting the patterns and outliers in the data.

GradientBoostingRegressor

Gradient Boosting constructs an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage, n classes regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. This model is particularly suited for complex datasets where both bias and variance are concerns. It improves predictions iteratively by focusing on mistakes of previous iterations, which enhances its ability to adapt to diverse data patterns effectively. It is especially powerful in scenarios where data might exhibit varying scales of importance across features, allowing it to dynamically adjust to the most predictive features. With an R-squared of 0.83, this model demonstrates robust performance, effectively adapting to the dataset's complexity and providing strong predictive power despite the potential for data irregularities.

DecisionTreeRegressor

Decision Tree Regressor constructs a model that uses learned decision rules based on data features to predict the value of the target variable. It is easy to interpret and visually display, and therefore, it is a good first choice to visualize the relationship between features and the response variable. They are especially useful for decision support due to their simplicity and interpretability. Each node in the tree corresponds to a feature in the data whose value has the greatest effect on the response and thus how the decisions are arrived at. But they are prone to overfitting if not properly tuned or if the tree is being allowed to grow too deep. It gives an R-squared of 0. 86 which is a nice level of simplicity and predictive accuracy. This makes it a good candidate for beginner analysts who need to understand the workings of a model as much as the accuracy of the predictions.

KNeighborsRegressor

The KNeighborsRegressor algorithm is based on feature similarity. This non-parametric method is particularly applicable in the scenario where there is a need to capture local variations in the pattern that might not be captured by the global smoothness assumed by the model. It forecasts the value of a new data instance by using the 'k' closest points in the feature space which means it is relying significantly on the local structure of the data. This approach is especially useful when data is not spread equally and relations between data points are irregular. Another critical hyperparameter of the model is 'k' – the number of considered neighbors; it is necessary to find the optimal 'k' to strike a balance between bias and variance. : How to get R-squared of 0. 89, the KNeighborsRegressor successfully presents its ability to replicate the relationship in data. It performs where dynamic and rapid changes in the dataset require prediction to follow.

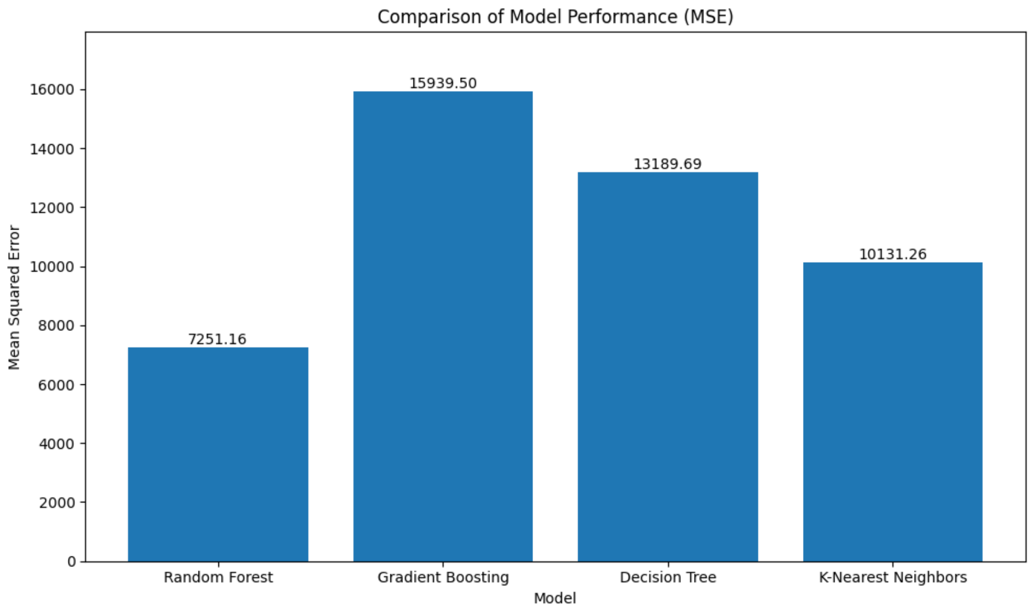


Figure 7.5: Comparison of Model Performance (MSE)

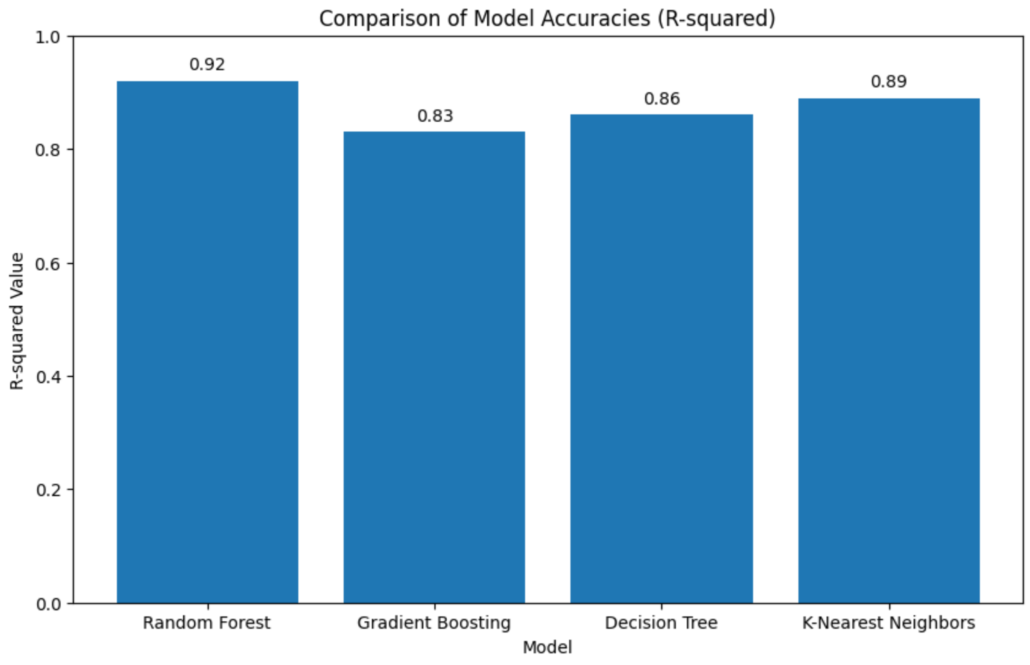


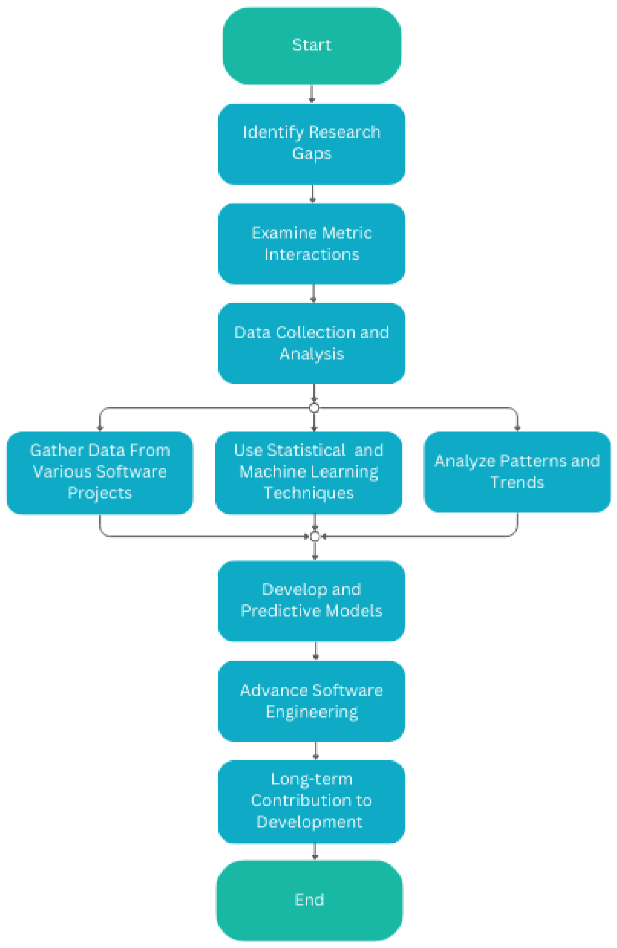
Figure 7.6: Comparison of Model Accuracies (R-squared)

In the analysis of regression models for software quality prediction, the

Random Forest model performed better than the rest of them with a high R-squared of 0. 92 suggesting that 92 percent of the variance has been explained and the lowest Mean Squared Error (MSE) 7251.16. Using the Gradient Boosting methodology we found R-squared of 0. 83 and an MSE of 15939. 50. The Decision Tree model provided a good balance between accuracy (R-squared of 0.86) and understandability of results from the model and MSE of 13189. 69. In K-Nearest Neighbors we found with an R-squared of 0. 89 and an MSE of 10131. 26. In terms of the trade-off between accuracy and model complexity the Random Forest is the best justified as it has the lowest value of MSE and the highest of R-squared value. So, it appears to be the most suitable candidate for software quality prediction. In summary, the choice of the best model for software quality prediction depends on the specific requirements of the problem, with the Random Forest model standing out for its high accuracy and explained variance.

Chapter 8

Work Plan


Start
Identify Research Gaps
Examine Metric Interactions
Data Collection and Analysis
Gather Data From Various Software Projects
Use Statistical and Machine Learning Techniques
Analyze Patterns and Trends
Develop and Predictive Models
Advance Software Engineering
Long-term Contribution to Development
End

The working plan for our thesis, "Analyzing Software Quality and Maintainability in Object-Oriented Systems Using Software Metrics," takes into account both our past successes and our long-term research goals. In our early phases, we have carefully studied key software metrics in existing object-oriented systems, including lines of code (LOC), comment percentage, cyclomatic complexity (CC), weighted methods per class (WMC), access to foreign data (ATFD), and response for a class (RFC). We have carried out a thorough literature analysis, identifying important research gaps and laying the groundwork for our study. In the subsequent stages of our working plan, we examined how the previously described software metrics interact with and affect one another in object-oriented systems. We also investigated the correlation between these metrics and real-world software issues, such as defects, code smells, and maintenance challenges. To ensure the generalizability of our findings, we gathered and examine data from numerous software projects that span a wide range of disciplines and scales. We used rigorous statistical and machine learning techniques to find patterns and trends in the data in order to increase the validity of our findings. In order to improve system quality and maintainability, we also created predictive models that can help software developers and maintainers make wise decisions. By carefully executing this strategy, we want to significantly advance the field of software engineering and, in the long run, aid in the development of higher-quality, more dependable object-oriented systems.

Chapter 9

Conclusion

When it comes to software development software testing is therefore crucial because it acts as a guarantee for the quality of the final product. Before the advent of software metrics, manual testing was the main method used by software developers to determine the measurements associated with their work, although manual testing for most software developers is generally considered costly and time-consuming. This research has examined and attempted to present the evaluation of software metrics used to measure the quality of software at both development stages and end products.

Some of the aspects that have been considered in this assessment include correctness, product quality, potential for extension, performance, and bug tolerance. Organizations are advised to make use of popular software metric tools since the quality standards are not synchronized in various entities. The implementation of these tools allow for a great deal of consistency and quality in the end products; it also increases the chances of re-using certain software and decrease the costs of developing and maintaining the software over time.

In the future, the researcher will assist with the application of each of the software measures and provide more information on how to improve software application quality for future work initiatives. It is possible to achieve such success if an organization applies the recommended changes and aligns the use of software metrics with a valuable impact on the improvement of the software development process itself and consequently on the quality of the product provided for maintenance.

Bibliography

[1] Mansi Agnihotri and Anuradha Chug. Application of machine learning algorithms for code smell prediction using object-oriented software metrics. *Journal of Statistics and Management Systems*, 23(7):1159–1171, 2020.

[2] Alexander Bart, Lisa Francis, Suresh Kumar, and Vandana Debroy. Exploring regular expression usage and context in python. *ResearchGate*, 2016.

[3] Canonical. Ubuntu manpage: Radon - python tool to compute code metrics. Accessed: 2023-10-16.

[4] Sonal Chawla and Gagandeep Kaur. Comparative study of the software metrics for the complexity and maintainability of software development. *International Journal of Advanced Computer Science and Applications*, 4(9), 2013.

[5] Xiaojie Chen. Introduction and analysis of python software. *Frontiers in Computing and Intelligent Systems*, 5(2):41–43, 2023.

[6] Melis Dagpinar and Jens H. Jahnke. Predicting maintainability with object-oriented metrics: An empirical comparison. In *10th Working Conference on Reverse Engineering (WCRE'03)*, pages 155–164. IEEE, 2003.

[7] DataScientest. Pylint: How to boost productivity through code analysis, 2023. Accessed: 2023-06-30.

[8] GeeksforGeeks. Pylint module in python, 2022. Accessed: 2022-04-18.

[9] Samira Gholizadeh. Top popular python libraries in research. *Journal of Robotics and Automation Research*, 3(2):142–145, 2022.

[10] D. M. Rasanjalee Himali and S. R. Kodithuwakku. Object-oriented software quality metrics. *Journal of Software Quality*, 12(1):45–60, 2023.

[11] Zhizhong Jiang, Peter Naudé, and Binghua Jiang. The effects of software size on development effort and software quality. *International Journal of Computer and Information Science and Engineering*, 1(4):230–234, 2007.

[12] Nigussu Bitew Kassie and Jagannath Singh. A study on software quality factors and metrics to enhance software quality assurance. *International Journal of Productivity and Quality Management*, 29(1):24–44, 2020.

[13] Anand Khandare, Nipun Agarwal, Amruta Bodhankar, Ankur Kulkarni, and Ishaan Mane. Analysis of python libraries for artificial intelligence. *Journal Name*, 2023.

[14] Michele Lanza and Radu Marinescu. *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems.* Springer Science & Business Media, illustrated edition, 2007.

[15] Mohammad Y. Mhawish and Manjari Gupta. Predicting code smells and analysis of predictions: Using machine learning techniques and software metrics. *Journal of Computer Science and Technology*, 35:1428–1445, 2020.

[16] Mike. Learning about code metrics in python with radon, 2023. Accessed: 2023-09-18.

[17] Oliyadk. How to get started with pylint, 2023. Accessed: 2023-02-23.

[18] Satya Prasad Raavi and N. V. Syma Kumar Dasari. Maintainability of object-oriented software metrics with analyzability. *International Journal of Computer Science Issues*, 12(3):127, 2015.

[19] Mrinal Singh Rawat, Arpita Mittal, and Sanjay Kumar Dubey. Survey on the impact of software metrics on software quality. *International Journal of Advanced Computer Science and Applications*, 3(1), 2012.

[20] A.L. Sayeth Saabith, T. Vinothraj, and MMM. Fareez. Popular python libraries and their application domains. *International Journal of Advance Engineering and Research Development*, 7(11):18–21, 2020.

[21] Rana Sandouka and Hamoud Aljamaan. Python code smells detection using conventional machine learning models. 9:e1370.

[22] G. Shanmugasundar, M. Vanitha, R. Čep, V. Kumar, K. Kalita, and M. Ramachandran. A comparative study of linear, random forest, and adaboost regressions for modeling non-traditional machining. *Processes*, 9(11):2015, 2021.

[23] Yeresime Suresh, Jayadeep Pati, and Santanu Ku Rath. Effectiveness of software metrics for object-oriented systems. *Procedia Technology*, 6:420–427, 2012.

[24] Unknown. Mastering pylint for python code quality, 2024. Accessed: 2024-10-01.

[25] Moshe Zadka. Pylint: Making your python code consistent, 2023. Accessed: 2023.