

UNIVERSITY OF MACEDONIA
SCHOOL OF INFORMATION SCIENCES
DEPARTMENT OF APPLIED INFORMATICS

EVALUATION OF PYTHON CODE QUALITY USING MULTIPLE SOURCE CODE ANALYZERS

Bachelor's thesis

of

George David Apostolidis

Thessaloniki, June 2023

EVALUATION OF PYTHON CODE QUALITY USING MULTI-METRIC ANALYSIS

George David Apostolidis

Undergraduate Student of Applied Informatics at University of Macedonia

Bachelor's Thesis

submitted for the partial fulfilment of its requirements

BACHELOR'S DEGREE IN APPLIED INFORMATICS

Supervisor

Alexander Chatzigeorgiou

Approved by the three-member examination committee on 13/06/2023

Alexander
Chatzigeorgiou

Apostolos
Ampatzoglou

Stylios
Xinogalos

.....

George David Apostolidis

.....

Abstract

Python has become one of the most popular programming languages in recent years, particularly in the fields of data analysis and scientific computing. To further improve code quality, this thesis proposes the development of a service tool that records the progress of GitHub projects and conducts a comprehensive multi-metric analysis of Python code. The static code analysis is performed using a variety of Python tools, and the thesis begins with a literature review that introduces the fundamental components of good code in Python. The tool's features are then presented, followed by a description of the analysis procedure and code examples. Two real-world GitHub projects from the domains of data analysis and image processing are examined, and the findings of the analyses are reported. This thesis emphasises the importance of code quality in software development and provides developers with a valuable tool for enhancing their code.

Keywords: Python, code quality, static code analysis, multi-metric analysis, Pytest, Pipreqs, Pylint, Duplication Code Detection Tool, GitHub, service tool, software development, software quality assurance

Preface

It is with great pleasure and gratitude that I present this thesis. This work is the culmination of a lengthy journey that would not have been achieved without the help, advice, and encouragement of various people who were crucial to its completion.

First and foremost, I want to thank my thesis advisor, Professor Alexander Chatzigeorgiou, for his great assistance, patience, and constant support during this project. His knowledge, devotion, and excitement for scientific inquiry have been a continual source of inspiration and motivation for me, and I am thankful for the chance to have worked under his supervision.

Also, I want to express my thanks to Nikolaos Nicolaides for his insightful comments, technical support, and assistance. I am sincerely appreciative of his assistance in guiding me through the challenges of this study because of his knowledge and encouragement.

Last but not least, I would like to extend my sincere gratitude to my family, whose everlasting love, support, and encouragement have served as the cornerstone of my academic years. The mental and monetary support they provided me allowed me to get past the numerous obstacles I had to confront while working on this project, and without it, I would not have been able to finish it.

Table of contents

1 Introduction	1
1.1 Purpose – Objectives	1
1.2 Contribution	2
1.3 Structure of the study	3
2 Bibliographic Review – Theoretical Background	4
2.1 Python (programming language)	4
2.2 Code quality	5
2.3 Code Quality Applications	9
2.4 Code quality in Python	12
2.4.1 Linters	13
2.4.2 Code Coverage	15
2.4.3 Test Coverage	16
2.4.4 Dependencies	17
2.5 Related work	18
3 Technology Stack	22
3.1 Java	22
3.2 Spring / Spring-Boot Framework	23
3.3 PostgreSQL	26
3.4 TypeScript	27
3.5 React Framework	27
3.6 Gitlab CI	28
3.7 Python Tools	29
3.8 Pylint	30
3.9 Pytest (--cov)	31
3.10 Pipreqs	32
3.11 Duplicate code detection tool	33
4 App functionality	34
5 Presentation of the Application	41
6 Results	55
6.1 TorchIO	55
6.1.1 Results	56

<u>6.2 DPPy</u>	60
<u>6.2.1 Results</u>	61
<u>6.3 Results of random files</u>	65
<u>7 Conclusion</u>	69
<u>7.1 Summary and conclusions</u>	69
<u>7.2 Suggestions for future research</u>	70

List of images

<u>Figure 1 - Product Quality Model (ISO/IEC 25010)</u>	8
<u>Figure 2 - Code analysis automation</u>	9
<u>Figure 3 - SonarQube User Interface</u>	10
<u>Figure 4 - Crucible Example</u>	11
<u>Figure 5 - Flake8 Example</u>	14
<u>Figure 6 - Unittest Example</u>	16
<u>Figure 7 - Pytest Example</u>	17
<u>Figure 8 - Spring Framework Architecture</u>	25
<u>Figure 9 - Gitlab CI/CD</u>	28
<u>Figure 10 - Pylint Example</u>	30
<u>Figure 11 - Pytest --cov Example</u>	31
<u>Figure 12 - Pipreqs Example</u>	32
<u>Figure 13 - Duplicate Code Detection Tool Example</u>	33
<u>Figure 14 - Application user interface</u>	34
<u>Figure 15 - Analysis results of a project</u>	35
<u>Figure 16 - Gitlab CI</u>	37
<u>Figure 17 - GET API JSON response</u>	37
<u>Figure 18 - Example of Project Analysis GET Response</u>	38
<u>Figure 19 - Example of file response from GET API</u>	39
<u>Figure 20 - Clone repository Method</u>	41
<u>Figure 21 - Capture SHAs code</u>	42
<u>Figure 22 - Determination of analysis</u>	43
<u>Figure 23 - Search for unique SHA</u>	44
<u>Figure 24 - Selecting files with the .py file extension</u>	45
<u>Figure 25 – Threads</u>	46
<u>Figure 26 – First Runnable</u>	47
<u>Figure 27 - Pylint Runnable</u>	47
<u>Figure 28 - Execute Command Method</u>	48
<u>Figure 29 - Counting lines of requirements.txt</u>	49
<u>Figure 30 - Pytest Response Method</u>	49
<u>Figure 31 - Store Similarity Method</u>	50

<u>Figure 32 - Store Comments Method</u>	52
<u>Figure 33 - Regex Patterns</u>	53
<u>Figure 34 - Delete Directory Method</u>	54
<u>Figure 35 - TorchIO Dependencies</u>	56
<u>Figure 36 - TorchIO Total Statements</u>	57
<u>Figure 37 - TorchIO Total Miss</u>	59
<u>Figure 38 - TorchIO Total Coverage</u>	60
<u>Figure 39 - DPPy Total Statements</u>	62
<u>Figure 40 - DPPy Dependencies</u>	63
<u>Figure 41 - DPPy Total Miss</u>	64
<u>Figure 42 - DPPy Total Coverage</u>	65

List of tables

<u>Table 1 - Standalone Python Linters</u>	15
<u>Table 2 - data_parser.py Results</u>	66
<u>Table 3 - intensity_transform.py Results</u>	67

1 Introduction

Software development is a complex and dynamic field that demands high-quality code to ensure efficient and reliable software systems. Code quality, which encompasses various aspects such as readability, maintainability, and efficiency, directly impacts the performance, stability, and overall success of software applications. Therefore, understanding and managing code quality is crucial for software development teams.

This thesis focuses on the development of a tool for automating the evaluation of Python code in GitHub projects across various research fields, particularly those where professionals and researchers utilise code to automate specific functions. To provide a comprehensive understanding of software quality, the theoretical background will be explored. This paper will primarily concentrate on the tool's features, its presentation, and its functionality, as well as the results obtained from analysing the aforementioned projects. The quality of software is a crucial aspect of software development, as it determines how well the software meets its intended purpose and how easy it is to maintain and modify. Thus, it is essential to examine the quality of software projects and identify areas for improvement. This thesis proposes a tool that automates the evaluation of Python code in GitHub projects, which can aid in identifying areas for improvement in software quality.

1.1 Purpose – Objectives

The study's objectives are to discover and examine various metrics for evaluating code quality, create an application to combine these metrics into a comprehensive quality score, and include the application as a tool that developers can use to enhance their code.

According to the research's background, maintaining high-quality code is crucial for the development of software, but doing it manually may be time-consuming and challenging. Automatic code analysis tools can aid programmers in finding possible problems and fixing their code, although Python-specific tools are frequently restricted in their capabilities and reach.

The Python programming language, which is extensively used in scientific, educational, and industrial applications, serves as the research's setting. Both individual developers and businesses that use Python for their software development can benefit from the tool created in this study.

1.2 Contribution

The research conducted in this study encompasses several key areas that contribute to the understanding of code quality in the chosen technology stack. These areas include a comprehensive literature review, technology stack analysis, application development, results analysis, and conclusion and recommendations.

The literature review is an in-depth exploration of existing literature on code quality, including theoretical background, best practises, and tools. It provides readers with a thorough understanding of the current state of the field and lays the foundation for the research conducted in this study. The technology stack analysis focuses on the specific technology stack used in this study, including programming languages, frameworks, and development tools. Through careful investigation, this analysis provides insights into the practical aspects of managing code quality in the context of the chosen technology stack, highlighting the unique challenges and opportunities associated with code quality in this environment.

The application development component of this study involves the development of a practical application that demonstrates effective code quality management practises in the chosen technology stack. This application serves as a concrete example for readers, showcasing the application of code quality concepts in a real-world context and illustrating the practical implementation of code quality practises. The results analysis entails the evaluation of the performance and effectiveness of the developed application in managing code quality. Through empirical evidence, this analysis provides insights into the practical implications and impact of code quality practises in the chosen technology stack, offering valuable insights for researchers, practitioners, and

stakeholders interested in improving software development processes and ensuring high-quality code.

Finally, the conclusion and recommendations section summarises the findings and conclusions drawn from the study and provides suggestions for future research in the field of code quality in a specific technology stack. These insights offer valuable directions for further exploration and contribute to the body of knowledge on code quality in the chosen technology stack, providing valuable insights for researchers, practitioners, and stakeholders interested in enhancing software development processes and ensuring high-quality code.

1.3 Structure of the study

The thesis consists of several chapters, including a bibliographic review and theoretical background chapter, a technology stack analysis chapter, an application development chapter, a presentation of the application chapter, and a results analysis chapter. The bibliographic review chapter discusses relevant literature on Python, code quality, code quality applications, and code quality in Python. The technology stack analysis chapter explores the specific technologies used in the study, including Java, Spring/Spring-Boot framework, PostgreSQL, TypeScript, React framework, Gitlab CI, and Python tools. The application development chapter details the functionality of the developed application, while the presentation of the application chapter showcases its design and interface. The results analysis chapter presents the findings of the study. The thesis concludes with an epilogue summarising the conclusions, limitations, and future extensions for further research.

2 Bibliographic Review: Theoretical Background

2.1 Python (programming language)

Python is a computer language that is ideal for real-world programming and is also simple to learn. Guido van Rossum created it in 1991, and it is known for its powerful high-end features. Python is a widely used dynamically typed language used in a wide range of fields, including automation, data analysis, web development, and scientific computing [1]. The extensive collection of libraries and frameworks that make it simple to handle difficult programming tasks is another factor contributing to its popularity. It's a fantastic language for beginners because of its simple syntax and ease of understanding [2]. Building large-scale applications, developing machine learning models, and manipulating massive data are just some of the sophisticated programming tasks that can be accomplished using the powerful Python programming language. The rich library ecosystem of this language is one of its main advantages. About 400,000 packages are available in the Python Package Index (PyPI) [3], which can be used to increase Python's functionality. These programmes cover a wide range of topics, including web development, scientific computing, and data analysis.

Python offers language constructs that enable comprehensible programming at all scales, no matter how large or tiny the program is. It is important to note that it supports a variety of programming paradigms, including procedural, functional, and object-oriented programming [4].

A powerful language with simple and straightforward syntax, Python includes high-level data structures such as lists, tuples, sets, and dictionaries, as well as dynamic type and binding, modules, classes, and exceptions. It is a popular choice in the machine learning industry due to its extensive collection of libraries, which reduces the amount of code developers need to write. In addition, it can be used on parallel computing systems thanks to the JPython interpreter. Although some people may have trouble getting to grips with it, research on the range of libraries that Python offers [5] can encourage users to use the language and advance their coding abilities. Python is a popular alternative for both novice and experienced programmers due to a number of features. Lutz [6] argues

that Python's emphasis on software quality is one of its main benefits. The language incorporates features such as object-oriented programming to promote code reuse and is intended to be simple to read and understand even without significant annotations.

Another benefit of Python is its development productivity. The language enables programmers to write code quickly, effectively, and concisely, which is sometimes impossible in other object-oriented languages like Java and C++. Both initial code writing and subsequent debugging can be sped up this way. With programs that can run on many operating systems and user interfaces, Python is also highly customizable. It is therefore a versatile choice for a variety of applications. A fairly large number of support libraries are also included in Python, which can help developers add functionality to their projects quickly and easily. Additionally, these libraries are highly extensible, allowing developers to create their own custom versions. In addition, Python has component integration, which allows developers to use Python applications as extensions to other programs and to integrate third-party libraries written in languages such as Java and C [2].

Overall, a developer's productivity and work happiness can benefit from the simplicity and fun of Python programming. In general, Python is quite attractive for a variety of programming tasks because of these features.

2.2 Code quality

According to Kitchenham and Pfleeger, in a report made in 1996 in the IEEE journal [7], it is pointed out the importance of programmers' employment with software quality. A good definition, according to them, is important so that developers and customers are now sure of what they mean when they refer to it and enable us to effectively measure quality.

The effectiveness of software quality metrics and the relationship between process and product quality are both determined through metrics. It must also be clarified whether the investment of time and resources to ensure good quality will result in higher profits or greater market share. This is because the quality of what we build can have an

impact on how the product is used after delivery. In other words, it is important to know whether quality software development is profitable. The majority of people believe that quality matters and can be improved.

Software quality therefore refers to a set of requirements that must be identical in a project [8]. Properties such as readability, maintainability, modularity, performance, and security are all important aspects of software, whether in a social media application or financial clearinghouse software. Code quality is affected by several factors, including architecture, API design, coding style, library choice, and adherence to best practices. Some of these, such as architecture and design, require human insight, but others, such as code analysis, can be automated. Tooling is a simple first step towards prioritising quality and can be a great way to ensure uniform standards and integrate analytics tools into the development process. It is important to consider some software quality factors that can be controlled automatically:

- **Readability**

According to Robert C. Martin in Clean Code: A Handbook of Agile Software Craftsmanship [9], "the ratio of time spent reading to writing is well over 10 to 1" in the development process. Because of this, code readability is a critical component of development productivity. The variable name is a component of readability that is highly circumstantial and even subjective. Examples of simple automated checks that improve readability include using a standardised coding style and avoiding the use of ambiguous terminology.

- **Performance**

Although system profiling and stress testing are necessary to fully understand application performance, static code analysis can identify some performance anti-patterns. Examples here could be looking at code that can be moved outside of loops, looking for concurrency issues that could cause trouble for threads, as well as pointless map searches.

- **Reliability**

Site reliability engineering has become a practical technique for system-scale reliability tests. However, using static analysis approaches, a variety of component reliability problems can be identified. In some cases, in multi-threaded programs, data race conditions can lead to a variety of unexpected behaviors. Crashing from null pointer exceptions can sometimes lead to denial-of-service flaws, and in some circumstances, bad exception handling can lead to unexpected behavior.

- **Security**

Red teaming and bug bounty programs have become popular approaches to finding vulnerabilities in deployed systems. However, such approaches are reactive and do not address the root cause of the vulnerabilities, which lies in the code itself. Automated static code analysis is a proactive approach to security that can find a wide range of security flaws early in the software development lifecycle. These flaws include, but are not limited to, weak encryption, insecure configurations of the framework, hard-coded secrets, and other vulnerabilities that can be exploited by attackers. By integrating automated static code analysis into the development process, organisations can identify security flaws before they are deployed and significantly reduce the risk of successful attacks.

- **Dependency Management**

On average, 90% of software applications created by third parties as part of the software supply chain depend on open source. Because of these third-party dependencies, the performance, reliability, security, and other quality characteristics of your application are mostly inherited. As a result, it is important to carefully evaluate the libraries used and monitor security vulnerabilities in the application's dependencies. The general term for the technology that can extract a list of dependencies is Software Composition Analysis (SCA) and determine the problems in any of them.

A method for evaluating the quality of a product is based on the quality model. The quality characteristics that will be taken into account when evaluating the properties of a software product are determined by the quality model.

The degree to which a system satisfies the explicit and implicit demands of its many stakeholders and thus adds value is referred to as its quality. The quality model, which divides product quality into attributes and sub-attributes, accurately represents the requirements of these stakeholders (functionality, performance, security, maintainability, etc.).

The eight quality factors shown in the accompanying illustration make up the product quality model described in ISO/IEC 25010 [10]:



Figure 1 - Product Quality Model (ISO/IEC 25010)

Ensuring code quality is an ongoing process. To ensure that the code base is maintainable while it changes, regular evaluation of the code quality must be done. It is important that developers follow the best practises outlined for the project by establishing code style guidelines and using automatic support mechanisms such as Linters. The first stage of automation is, as mentioned, static code analysis.

Static code analysis, as the name suggests, is just a simple evaluation of code (without its context) to find parts of code that don't follow a certain set of rules and principles. It is a technique of code review and detection of software defects and code conformance before execution [11].

We usually automate the analysis of static code in multiple stages, such as in developers' IDEs, where Linters are used to automate code quality control on its development, and in the quality gates of CIs for pull requests (or merge requests) in git, where rules also apply in a CI environment.

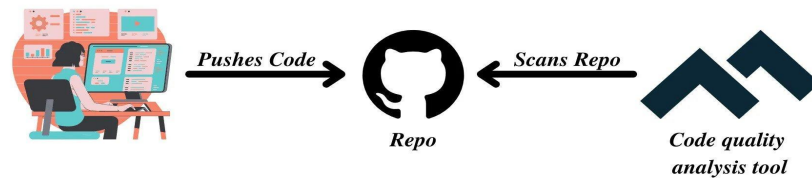


Figure 2 - Code analysis automation

2.3 Code Quality Applications

Maintaining code quality has become more important than ever in the current fast-paced world of software development, where agile techniques and continuous delivery practises are becoming more prevalent. Software development teams now have access to effective tools for ensuring that the code they write adheres to the highest standards of quality, called "code quality apps." These programs employ a variety of ways to examine and analyse code, spot possible problems, and provide developers with useful feedback. Developers may avoid common problems, improve code performance, and produce more dependable and maintainable software with the aid of code quality solutions. In this part, the most well-known code-quality apps will be examined, along with their capabilities and advantages for developers [12].

- **SonarQube**

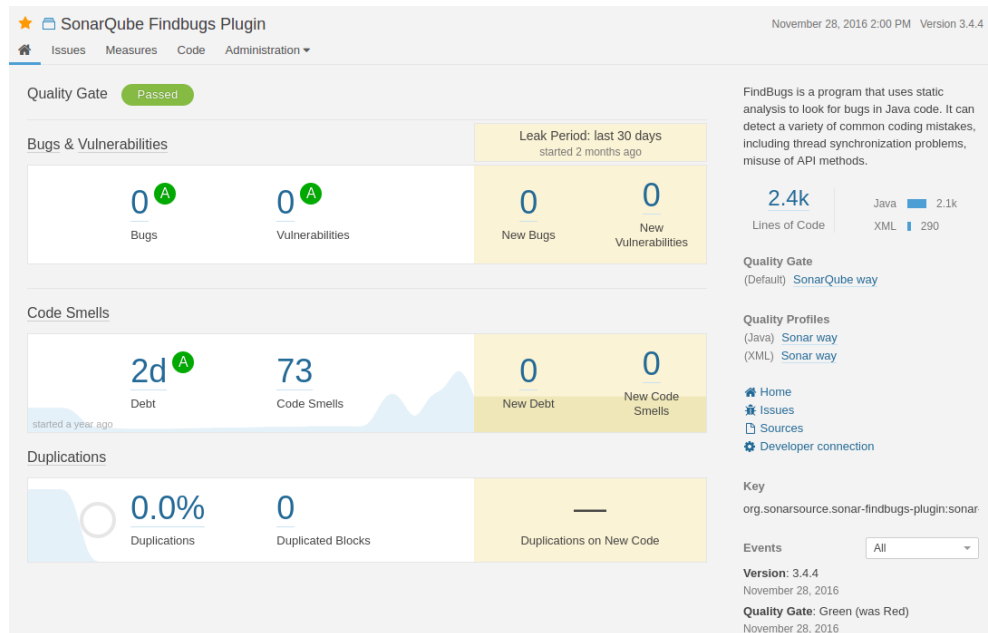


Figure 3 - SonarQube User Interface

A popular open-source static code analysis tool called SonarQube [13] can help developers find and flag security flaws in their code. To scan code and find potential security flaws, it uses a number of approaches, including pattern matching, data flow analysis, and control flow analysis. The fact that SonarQube supports both on-premise and cloud-based installations is one of its key advantages. The on-premise version can be installed on a local server or used locally because it is open source and free to use. Because of this, it's ideal for small teams or companies that want to verify that their code is secure but have a limited budget. The cloud-based version of SonarQube, on the other hand, costs money and offers additional features, including compatibility with platforms and development environments such as GitHub, Bitbucket, and Jenkins. It also offers more thorough reporting and code analysis, which can be useful for larger teams or businesses that need more sophisticated security features. Support for more than 27 programming languages, including well-known ones like Java, C#Go, and Python, is also one of the main advantages of SonarQube. The ability to scan and analyse code written in multiple programming languages using a single tool makes this feature important for developers working on multiple projects.

- **Crucible**



Figure 4 - Crucible Example

Crucible [14] is a well-known application that enables rapid workflow-based code reviews. It allows developers to evaluate code in a fast and simplified way, ensuring compliance with protocols and code quality standards. Crucible's real-time alerts make it easy to organise and track code reviews while staying on top of deadlines. It supports all major programming languages, including Java, C++, Python, and many more.

- **ESLint & JSHint**

JavaScript static code analysis tools such as ESLint [15] and JSHint [16] are widely used. Each of these technologies is accessible as npm packages, allowing developers to easily include them in their projects. They are free and open source, which means that anyone can use them without paying any licence fees. The ability to configure rules and checkers used for code analysis is a key feature of both ESLint and JSHint. Developers can use a variety of configuration settings to tailor the tools' behavior to their particular requirements. This makes it easy to enforce code quality standards and ensure that code conforms to best practices. Another advantage of these tools is that they support

JavaScript. Because JavaScript is a widely used programming language, these tools can be used by developers working on projects in various fields. ESLint and JSHint improve code quality and reduce the risk of vulnerabilities and defects by evaluating code for common errors and mistakes.

2.4 Code quality in Python

There are a few things to keep in mind while attempting to write Python code of high caliber. Strong viewpoints exist over what should constitute high-quality code, particularly when it comes to readability, maintainability, and extensibility. It will open with a mention of the coding style used by Python.

It is legitimate to want code consistency [17] regardless of how white space is represented to each programmer's taste. A consistent method for creating code is defined by a style guide. Typically, this is all just aesthetic and has no bearing whatsoever on the code's logical result. Nonetheless, certain aesthetic choices guard against formal and logical mistakes. The objective of making code simple to comprehend, maintain, and expand is helped by style guides. For Python, there is a generally accepted standard. It was partly developed by the Python programming language's developer.

PEP 8 and PEP 257 are two important documents that provide guidelines for Python programming, and they are widely adopted by developers. PEP 8 specifies coding rules for Python programming, which include guidelines for code layout, syntax, naming conventions, and comments. Adhering to these guidelines can improve the readability and maintainability of the code, making it easier for other developers to understand and work with it. On the other hand, PEP 257 focuses on the documentation of the code through the use of docstrings, which are strings that describe what a module, class, function, or method does. By following these guidelines, developers can ensure that their code is well documented and that others can understand its functionality without having to read the code. Additionally, consistent docstrings can be used to generate documentation directly from the code, which can save time and effort in the long run. Following these guidelines can improve the quality of the code and make it easier to maintain and understand.

2.4.1 Linters

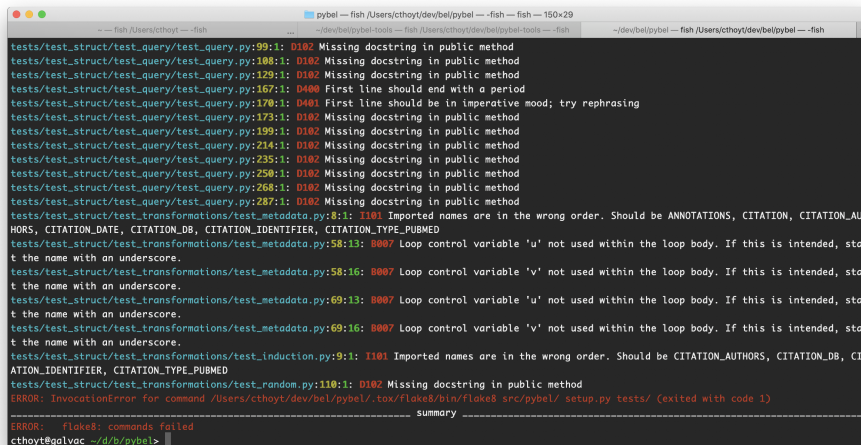
In order to correctly find errors in the code, linters are used. Lint refers to the process of analysing source code to identify and report potential issues such as syntax errors, coding style inconsistencies, and other code quality issues.

Linters play an important role in ensuring that code is readable, maintainable, and free of bugs and vulnerabilities. They are automated tools that scan code and provide feedback on potential issues, allowing developers to identify and fix problems early in the development process. They can be configured to enforce specific coding standards, such as variable naming conventions, indentation styles, and other best practises. By addressing these issues early, linters help reduce the time and effort required for debugging and maintenance. They can also help improve code quality and prevent potential security vulnerabilities by highlighting potentially dangerous code patterns.

Most modern code editors and integrated development environments (IDEs) provide a linter feature, which is built to inspect code as it is written in real time. Potential problems, including syntax errors, inconsistent coding styles, and other quality issues, may be flagged by this feature. Linters can flag problematic areas in code using visual cues such as underlining or highlighting, much like a word processor's spell check does. They sort the possible errors into logical or stylistic groups after studying the code. Although stylistic lints refer to code that deviates from accepted rules, logical lints are problems that can have incorrect or unwanted results. Code defects, potentially dangerous code patterns, and unwanted code results are some examples of logical lint. In contrast, stylistic lints include problems such as inconsistent code practices, improper variable naming practices, and other violations of coding standards.

When considering options for liners, it is important to note that some tools available on the market may actually be a combination of multiple frames packaged together. These complex patterns can be useful as they provide a comprehensive set of features and functions, but it is important to have an understanding of what each linter offers and how they work together. Some popular examples include the following:

- **Flake8**



```
tests/test_struct/test_query/test_query.py:99:1: D102 Missing docstring in public method
tests/test_struct/test_query/test_query.py:108:1: D102 Missing docstring in public method
tests/test_struct/test_query/test_query.py:129:1: D102 Missing docstring in public method
tests/test_struct/test_query/test_query.py:167:1: D400 First line should end with a period
tests/test_struct/test_query/test_query.py:170:1: D401 First line should be in imperative mood; try rephrasing
tests/test_struct/test_query/test_query.py:173:1: D102 Missing docstring in public method
tests/test_struct/test_query/test_query.py:199:1: D102 Missing docstring in public method
tests/test_struct/test_query/test_query.py:214:1: D102 Missing docstring in public method
tests/test_struct/test_query/test_query.py:235:1: D102 Missing docstring in public method
tests/test_struct/test_query/test_query.py:258:1: D102 Missing docstring in public method
tests/test_struct/test_query/test_query.py:268:1: D102 Missing docstring in public method
tests/test_struct/test_query/test_query.py:287:1: D102 Missing docstring in public method
tests/test_struct/test_transformations/test_metadata.py:11:1: I001 Imported names are in the wrong order. Should be ANNOTATIONS, CITATION, CITATION_AUT
HORS, CITATION_DATE, CITATION_DB, CITATION_IDENTIFIER, CITATION_TYPE_PUBMED
tests/test_struct/test_transformations/test_metadata.py:58:13: R007 Loop control variable 'u' not used within the loop body. If this is intended, star
t the name with an underscore.
tests/test_struct/test_transformations/test_metadata.py:58:16: R007 Loop control variable 'v' not used within the loop body. If this is intended, star
t the name with an underscore.
tests/test_struct/test_transformations/test_metadata.py:69:13: R007 Loop control variable 'u' not used within the loop body. If this is intended, star
t the name with an underscore.
tests/test_struct/test_transformations/test_metadata.py:69:16: R007 Loop control variable 'v' not used within the loop body. If this is intended, star
t the name with an underscore.
tests/test_struct/test_transformations/test_induction.py:9:1: I101 Imported names are in the wrong order. Should be CITATION_AUTHORS, CITATION_DB, CIT
ATION_IDENTIFIER, CITATION_TYPE_PUBMED
tests/test_struct/test_transformations/test_random.py:110:1: D102 Missing docstring in public method
ERROR: InvocationError for command /Users/cthoht/dev/bel/pybel/.tox/flake8/bin/flake8 src/pybel/ setup.py tests/ (exited with code 1)
-----
ERROR: Flake8: commands failed
cthoht@galvac ~/d/b/pybel$
```

Figure 5 - Flake8 Example

Flake8 [18] is a robust and popular linter that provides developers with a comprehensive set of tools to improve code quality. With the combination of PyFlakes, pycodestyle, and McCabe Complexity Checker, Flake8 can find logical and stylistic errors in code as well as evaluate the complexity of code. PyFlakes detects common issues such as undefined variables, while pycodestyle focuses on formatting and naming conventions. Additionally, the McCabe Complexity Checker evaluates the complexity of code and can help developers identify areas of code that may be difficult to understand or maintain. By merging these discrete linters, Flake8 provides a powerful solution for developers to find and fix a wide range of bugs in their code. Its flexibility and configurability make it a popular choice among developers, and it can be easily integrated into development workflows to help ensure consistent code quality.

- **Pylama**

Pylama [19] is a popular combination linter that integrates numerous code analysis tools and linters. The linter provides PyFlakes, pycodestyle, and McCabe, which are used in Flake8, as well as Pylint. Pylint is an advanced linter that offers the ability to detect coding errors and poor coding practices in Python code. Pylama also offers Radon, a tool that provides developers with a way to measure the complexity of code and identify areas that may require attention. Radon uses algorithms that measure the code's maintainability, understandability, and complexity. Moreover, Pylama provides gjslint,

which is designed to evaluate the quality of JavaScript code. The combination of these linters and analysis tools in Pylama offers developers a comprehensive approach to improving the quality of their code.

The following standalone Linters come with a brief explanation:

Table 1 - Standalone Python Linters

Linters	Category	Description
PyLint	Logical & Stylistic	Searches for code smells, attempts to impose a coding standard, and checks for flaws.
PyFlakes	Logical	Program analysis and error detection
Pycodestyle	Stylistic	Analysis of programs and discovery of errors
Pydocstyle	Stylistic	Checks for conformance to Python docstring conventions.
Bandit	Logical	Analyzes code to identify common security flaws.
MyPy	Logical	Checks for static types that can be imposed selectively.

2.4.2 Code Coverage

Code coverage is a software test measure that determines the number of successfully verified lines of code in a test method. Code coverage testing can help determine how thoroughly a piece of software has been tested. For test cases involving a large number of lines of code, both human and automated testing approaches are evaluated. The goal is to measure the number of lines of code validated by the test technique as well as the total number of lines of code in a software component. The

primary goal of any development team is to deliver enterprise-grade software with the fewest possible bugs. To do this, the program must be revised to include evaluation, monitoring, and measurement of test methods. This is where code coverage tests come into play as a useful tool for evaluating the overall effectiveness and completeness of tests.

2.4.3 Test Coverage

The first key difference between code coverage and test coverage is that this is a black-box testing technique. It essentially counts the number of tests that have been run and whether the current test cases cover the majority of the various documents included. When all the functions specified in the documents have been performed, test code must be written to validate the product features that have been implemented. The goal is to provide insights into tests performed on a software solution. In software testing, test coverage includes a variety of testing methodologies, such as unit testing, responsive testing, cross-browser testing, integration testing, and acceptance testing. Test coverage is then evaluated and measured against a number of features covered by the test code. Some of the tools for test coverage are the following:

- **Unittest**

```
import unittest

class Test_Foo(unittest.TestCase):
    def test_set_bar(self):
        self.bar = 1

    def test_get_bar(self):
        print(self.bar)

if __name__ == '__main__':
    unittest.main()
```

Figure 6 - Unittest Example

Unittest [20] is a robust and widely used testing framework that is based on the principles of JUnit and other unit testing frameworks in major programming languages.

One of the primary advantages of using Unittest is the ability to perform automated testing and generate shared setup and shutdown code. This setup code is essential for running one or more tests and ensures that any required cleanup actions are performed once testing is complete. Furthermore, Unittest provides a comprehensive solution for handling activities such as creating temporary or proxy databases, directories, or starting a server process. By centralising these setup activities, Unittest minimizes the risk of errors and simplifies the testing process. Its text component is also highly advantageous, allowing for easy documentation of test cases, results, and other important information. These features make Unittest a highly effective tool for ensuring the quality and reliability of code in a scientific and systematic manner.

- **Pytest**

```
(venv_py) λ ~/Documents/python_py_test/ pytest -v test_mathlib.py
===== test session starts =====
platform linux -- Python 3.9.0, pytest-6.2.1, py-1.10.0, pluggy-0.13.1 -- /usr/bin/python3.9
cachedir: .pytest_cache
rootdir: /home/preetpal/Documents/python_py_test
collected 5 items

test_mathlib.py::test_calculation[1-11] PASSED [ 20%]
test_mathlib.py::test_calculation[2-22] PASSED [ 40%]
test_mathlib.py::test_calculation[3-33] PASSED [ 60%]
test_mathlib.py::test_calculation[4-44] PASSED [ 80%]
test_mathlib.py::test_calculation[5-55] PASSED [100%]

===== 5 passed in 0.01s =====
(venv_py) λ ~/Documents/python_py_test/ █
```

Figure 7 - Pytest Example

Pytest [21] is a popular and widely supported third-party Python testing framework that is considered a better choice compared to Unittest. It is known for its sophisticated features and extensive community support. Pytest aims to reduce the overheads associated with test production by providing developers with a framework for writing tests quickly and efficiently. Pytest is compatible with various Python versions, including 2.7, 3.4, 3.5, 3.6, Jython, and PyPy-2.3, and it can be used on Unix/Posix and Windows operating systems, making it a versatile choice for testing Python applications.

2.4.4 Dependencies

Dependencies are additional code that a developer wants to invoke. The process of building, testing, debugging, and maintaining a particular code unit is avoided by

adding a dependency. According to Cox [22], software dependencies pose significant risks that are too often neglected. The move to fine-grained software reuse has happened so quickly that we still don't understand the best methods for efficiently selecting and using dependencies, or even determining when they are appropriate and when they aren't. There are case studies like that of Ivan Pashchenko, Henrik Plate, Serena Elisa Ponta, Antonino Sabetta, and Fabio Massacci [23], whose results show us the importance of having knowledge of the amount of these dependencies in the code. More specifically, their study states that this practice allows software development companies to gain important information about their library dependencies and, as a result, appropriately spend expensive development and testing resources that are otherwise wasted when metrics are misinterpreted.

Python provides various tools that are essential for managing dependencies and libraries in projects. One such tool is PipReqs [24], which is particularly important for Python projects as it maintains comprehensive information about the libraries required for project execution. Typically, this information is obtained using the `pip freeze` command, which generates a virtualized list of all installed libraries. PipReqs allows developers to easily manage and track project dependencies, ensuring that the required libraries are installed and properly maintained throughout the development process. This helps in maintaining a clean and organised environment for Python projects, ensuring smooth execution, and minimising potential issues related to dependencies.

2.5 Related work

Static code analysis is a critical process in software development that helps identify defects, vulnerabilities, and other issues in code before it is executed. While static code analysis has been widely used in Java-based systems, its application in other programming languages such as Python is growing rapidly. Several researchers have explored the use of static code analysis in Python-based systems, and their findings have made significant contributions to the field of software engineering. In this section, we review some of the related work on static code analysis for Python code and its contributions to the field.

The construction of static call graphs from Python source code is a critical task in many procedural analysis and software understanding tools. However, automating this process remains challenging due to the dynamic nature of Python. To address this issue, the research team led by Gharib Gharibi, Rashmi Tripathi, and Yugyung Lee [25] has developed a prototype Python tool called code2graph. This tool automates the extraction of the structure of Python source code, the construction of static call graphs, and the creation of a similarity matrix for all possible execution paths in the system.

The goal of the code graph is two-fold. First, it aims to help developers understand the overall structure of a system by providing visual representations of the static call graph, which can highlight the relationships between different components and modules in the codebase. This can facilitate the identification of key software elements and their interactions, aiding developers in gaining insights into the system's architecture and design.

Second, the code2 graph provides a foundation for future research in software search and similarity detection applications. By grouping execution paths into a logical system workflow, the code2 graph enables the automation of specific software tasks. For example, identifying similar patterns of code execution across different projects or detecting similarities in the behavior of different functions or methods. This can be valuable in tasks such as code reuse, code refactoring, and identifying potential code clones.

The effectiveness of code2graph has been demonstrated by the research team through its application to three popular open-source deep learning projects, namely TensorFlow, Keras, and PyTorch. The tool has been used to generate static call graphs and path similarity tables, providing valuable insights into the structure and behavior of these projects.

The development of code graphs represents a significant contribution to the field of software analysis and understanding, as it addresses the challenge of automating the construction of static call graphs from Python source code. This tool has the potential to enhance software development practices by providing developers with a deeper

understanding of the structure and behavior of Python projects and by enabling the automation of specific software tasks through the identification of similarities in code execution patterns.

PyTA [26], a wrapper for a popular Python static analysis tool, has been specifically designed to help novice developers find and fix common errors in their code. PyTA provides custom checks for common beginner errors and improved prompts to assist students in understanding and resolving problems. The effectiveness of PyTA was evaluated by integrating it into an existing online system used to deliver programming exercises to CS1 students and comparing the results with previously collected data.

The analysis showed that PyTA can help students identify and resolve errors faster, resulting in a reduction in repeated errors and the time taken to complete programming problems. These findings suggest that static analysis tools like PyTA can be effective in supplementing traditional feedback methods to assist students in better understanding and debugging their code.

The Scalpel framework [27] is a significant advancement in the field of Python static analysis, providing developers with a comprehensive set of pre-built tools for performing fundamental static analysis tasks such as call graph construction, flow control graph construction, and alias analysis. These tools enable developers to more effectively identify and fix bugs, vulnerabilities, and other issues in their Python code, leading to improved software quality and reliability. The incorporation of these fundamental static analysis functions in the Scalpel framework makes it easier for developers to implement dedicated Python static parsers, streamlining the process of analysing Python code for potential issues. This can greatly benefit software development teams by helping them identify and address issues early in the development lifecycle, reducing the likelihood of introducing bugs or vulnerabilities into the final codebase. One notable advantage of the Scalpel framework is its open-source nature, which makes it accessible to a wide range of developers and promotes continuous development and improvement over time. The availability of the framework to the broader community fosters collaboration, feedback, and contributions from the community, leading to ongoing enhancements and refinements to the framework.

With the Scalpel framework, developers can leverage advanced static analysis techniques to gain insights into their Python code, identify potential issues, and make informed decisions about code changes. This can result in more robust and secure Python applications with improved code quality, reliability, and maintainability. The availability of the Scalpel framework promises to enable new and innovative approaches to tackling the challenges associated with writing high-quality Python code, benefiting the Python development community as a whole.

In conclusion, while there are several existing tools and frameworks available for static code analysis of Python code, there is still a need for a service-based tool that integrates these tools and provides fast and flexible static code analysis. The proposed thesis aims to address this gap by designing and implementing a service-based tool that can efficiently perform static code analysis on Python code. By leveraging the strengths of existing tools and frameworks, this service-based tool can provide a more comprehensive and efficient solution for finding bugs, vulnerabilities, and other issues in Python code. Overall, this tool has the potential to significantly improve the quality of Python code and improve developer productivity.

3 Technology Stack

When building a service-based application, such as the tool presented in this study, choosing the right technology stack is critical to providing a reliable and scalable solution. A service-based architecture involves dividing the application into smaller, independent services that communicate with each other to provide overall functionality. The technology stack for a service-based application must therefore be chosen with careful consideration of the unique requirements of the architecture. Factors such as scalability, reliability, and flexibility are important when choosing it. In addition, the chosen technologies must allow easy integration with other systems and services while providing excellent performance and security. In this chapter, we'll explore the various options available for building a service-based application and the key considerations for choosing the right tools and frameworks.

3.1 Java

Java [28], a widely used and popular programming language, was created by James Gosling at Sun Microsystems in 1995. Initially, Java's core components, including compilers, virtual machines, and class libraries, were released under proprietary licences. However, in May 2007, Sun Microsystems relicensed most of its Java technologies under the GNU General Public Licence (GPL) in compliance with the Java Community Process (JCP) specification. This move aimed to promote open-source development and community involvement in the evolution of Java.

One significant outcome of this relicensing was the emergence of the OpenJDK project as the official reference implementation of the Java platform. The OpenJDK JVM (Java Virtual Machine) is now widely recognized as a free and open source software (FOSS) implementation of the Java platform, and it has become the default JVM for almost all major Linux distributions. The OpenJDK project is hosted and maintained by the open source community, which fosters collaboration, innovation, and transparency in the development of Java technologies.

Oracle, a key player in the Java ecosystem, continues to provide the HotSpot JVM, which is a commercially supported and optimized implementation of the Java Virtual Machine. However, the OpenJDK JVM has gained widespread adoption due to its open-source nature, community-driven development model, and compatibility with the Java Community Standard (JCS). The availability of the OpenJDK JVM as free and open source software has democratized access to Java technology, enabling developers around the world to use, modify, and distribute Java software without proprietary restrictions. The shift towards open-source licensing and the establishment of the OpenJDK project have had significant implications for the Java community. It has promoted greater transparency, collaboration, and innovation in the development of Java technologies and has encouraged widespread adoption of the OpenJDK JVM as the de facto standard implementation of the Java platform. This has resulted in a vibrant ecosystem of open-source libraries, tools, and frameworks built on top of the OpenJDK JVM and has facilitated the growth and evolution of the Java programming language as a widely-used and versatile technology for a wide range of applications.

The language allows developers to write code once and run it on any platform that supports Java without the need for recompilation. Java applications are typically compiled to bytecode, which can be executed on any Java Virtual Machine (JVM) regardless of the underlying computer architecture. Although the syntax of Java is similar to that of C and C++, it has low-level facilities. The Java runtime offers dynamic capabilities not available in traditional compiled languages. As of 2019, Java was among the most popular programming languages in use, particularly for client-server web applications, with approximately 9 million developers.

3.2 Spring / Spring-Boot Framework

The Spring Framework is a widely used open-source application development framework for Java and J2EE environments that aims to enhance developer productivity. With a reported 30% adoption rate among Java developers [29], it is considered one of the most popular frameworks in the Java ecosystem. The Spring Framework provides a range of features that facilitate the efficient development of diverse applications, from simple web applications to complex enterprise-level systems.

The Spring Framework is founded on several fundamental concepts that shape its design and functionality. These include inversion of control (IoC), dependency injection (DI), aspect-oriented programming (AOP), and the Java Persistence API (JPA). These concepts form the cornerstone of the framework's architecture, providing developers with powerful tools for building flexible, modular, and scalable Java applications.

IoC is a general concept where flow control is reversed. Instead of developers controlling program flow, external resources (frameworks, services, and other components) control program flow. AOP is a programming paradigm that enables the separation of cross-cutting concerns from the core logic of the application, improving both modularity and code structure, while JPA is responsible for how to query between objects and how object state is mapped to database fields.

Spring Boot consists of about 20 modules. Core Container, Data Access/Integration, WEB, AOP, Instrumentation, Messaging, and Test are some of the most critical modules. For developing Web applications, the Web module is very important. Web, WebSockets, Portlets, and Servlets are included. The Servlet module contains the definitions of the two most commonly used concepts nowadays. The first is Spring Model-ViewController (MVC). The second is RestFull Web Service (REST WS) integrations.

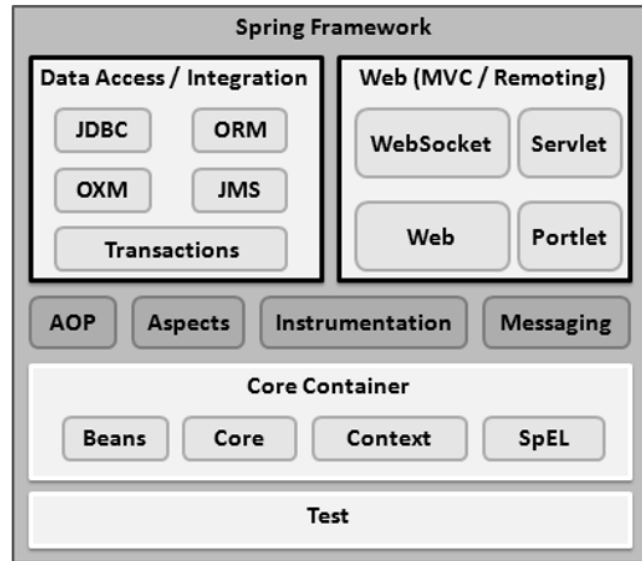


Figure 8 - Spring Framework Architecture

One of the key advantages of Spring Boot is its automatic configuration capability, which allows applications to be configured as standard Spring applications with sensible defaults. This eliminates the need for extensive manual configuration, reducing complexity and allowing developers to focus on writing business logic instead of spending time on tedious configuration tasks.

Another notable feature of Spring Boot is its seamless integration of startup dependencies. Based on the application's configuration and project dependencies declared in the build file, Spring Boot automatically resolves and integrates all required dependencies. This ensures smooth application startup and minimizes configuration errors, leading to more reliable and stable applications.

The command-line interface (CLI) provided by Spring Boot is also a significant productivity booster for developers. It allows developers to effectively manage and configure applications through the console, streamlining development and deployment workflows. This enables faster and more efficient development iterations, making it easier to build, test, and deploy applications.

Additionally, the Spring Boot Actuator is a powerful toolset that provides real-time insights into the runtime behavior of the application. It offers various endpoints

for monitoring and managing application health, metrics, logging, and more. This enables effective troubleshooting, performance optimization, and monitoring of the application, ensuring its smooth operation in production.

Spring Boot offers a faster and more accessible approach to Spring development by providing a variety of non-functional features common to many projects. These features include embedded servers, security, metrics, health checks, and external configuration and can be easily integrated without requiring XML configurations or code generation. Additionally, Spring Boot simplifies application testing, making it easier. Users can create a starter project with custom dependencies using the <https://start.spring.io/> website. This site allows users to choose between Maven or Gradle projects and select the necessary dependencies. To develop a REST web service application, the WEB module dependency is required.

3.3 PostgreSQL

PostgreSQL [30] is a client-server relational database that is open source. PostgreSQL provides a distinct feature set that compares well with major commercial databases such as Sybase, Oracle, and DB2. One of the most important advantages of PostgreSQL is that it is open source. There is no company that owns PostgreSQL exclusively. It is developed, maintained, and repaired by a global community of volunteer developers.

PostgreSQL provides all the standard features of a relational database as well as some unique features, provides inheritance, and allows the creation of a data type based on the needs of the database systems that allow you to rename an existing type. Some systems even allow the creation of complex formulas. New basic data types are available in PostgreSQL. Geometric data types such as point, line segment, box, polygon, and circle are supported by PostgreSQL. PostgreSQL uses indexing techniques that speed up geometric data types and is extensible because new functions, operators, and data types can be written in any language the programmer wishes.

3.4 TypeScript

TypeScript [31] is a superset of JavaScript that adds a module system, classes, interfaces, and a static type system. It supports standard JavaScript development standards and offers tools and IDE experiences previously associated with other programming languages, such as Java. It helps developers by catching errors automatically and providing appropriate methods to invoke on an object. Its class support is compliant with the requirements of EcmaScript 6. Additionally, it adds syntax for defining and expressing types, annotating properties, variables, arguments, and return values with types, and confirming the type of an expression in JavaScript. It introduces new language features such as classes, modules, and lambda expressions. These constructs are backports of future JavaScript capabilities, but they interact with the type system in a meaningful way without changing its core properties. TypeScript's design intent is to support existing JavaScript styles and idioms while still being relevant to popular JavaScript libraries, rather than creating a new programming language.

The TypeScript compiler validates TypeScript programs and generates JavaScript, allowing them to run in a variety of runtime environments. Since its debut in late 2012, the compiler has been widely used within Microsoft to build important JavaScript applications, and it has also been used outside of Microsoft. The TypeScript type system offers sophisticated features and ideas such as structural type equivalence, object-based programming, progressive typing, recursive type subtyping, and type operators. These features greatly add to a pleasant programming experience but can be difficult to incorporate into standard JavaScript idioms and codebases. The designers of TypeScript made a strategic decision not to insist on static precision because it is critical to the usability of the language to allow common patterns in popular APIs, even if it means not ruling out correctness in some cases [32].

3.5 React Framework

React is a powerful and widely adopted JavaScript library that is specifically designed for building user interfaces. It was created by Jordan Walke, a skilled software engineer at Facebook, in 2011. After being initially used internally at Facebook, React

was open-sourced in 2013, allowing developers worldwide to access and contribute to its development.

Since its open-source release, React has continued to evolve and receive regular updates, driven by an active community of developers and contributors. In 2015, React Native, a mobile framework based on React, was also open sourced, further expanding the capabilities of React for mobile app development. In 2017, React360, a virtual reality development toolkit, was published, extending the reach of React into the realm of virtual reality applications.

Currently, React is actively developed and maintained by Meta and a vibrant community of individual users who leverage React's capabilities in various projects, including popular applications such as Facebook and Instagram. The widespread adoption of React by major companies and individual developers alike is a testament to its robustness, versatility, and effectiveness in building modern user interfaces for web, mobile, and virtual reality platforms.

3.6 Gitlab CI

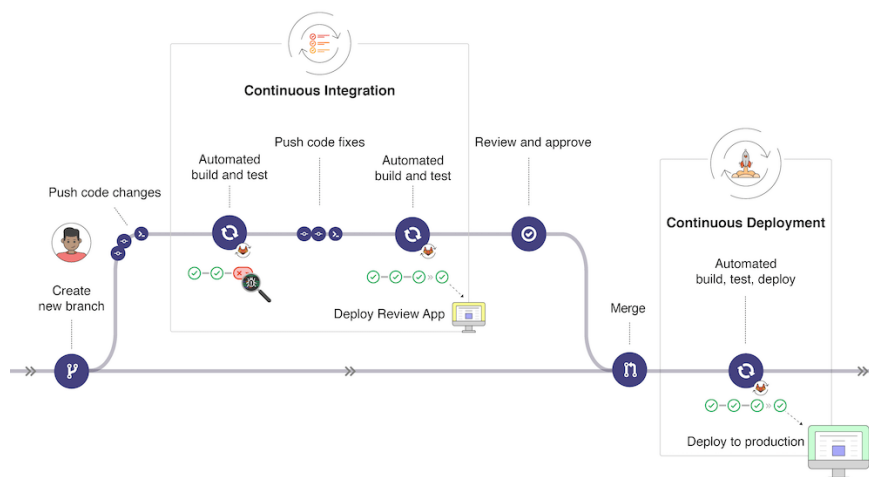


Figure 9 - Gitlab CI/CD

GitLab Runner [34] is a versatile and powerful tool that seamlessly integrates with GitLab CI/CD, enabling users to automate tasks in their pipelines. With GitLab Runner, users can leverage hosted runners provided by GitLab, which are fully integrated with GitLab.com, making it convenient and easy to get started. By default, all projects on

GitLab.com have access to these runners, although project owners have the flexibility to disable them if needed.

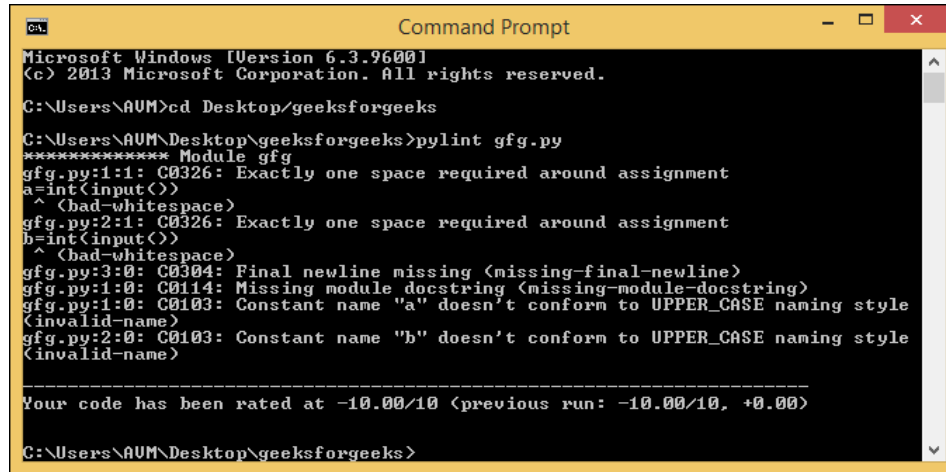
In addition to using GitLab.com runners, users also have the option to install and register their own runners on GitLab.com or on their own infrastructure [34]. This self-managed approach gives users greater control and flexibility, as they can install and maintain GitLab Runner on their own infrastructure, tailoring it to their specific requirements.

Regardless of whether users choose to use GitLab.com runners or self-managed runners, GitLab Runner provides a powerful solution for simplifying pipeline processes. It is accessible at all GitLab levels, ensuring that all GitLab users, regardless of their project size or scope, can benefit from its capabilities. With GitLab Runner, users can automate tasks, streamline their pipelines, and enhance their CI/CD workflows, ultimately improving the efficiency and effectiveness of their software development processes.

3.7 Python Tools

In the previous chapters, the importance of ensuring high-quality code in software development has been highlighted. Python, being a widely used programming language, offers a plethora of tools that can assist developers in achieving this objective. These code quality tools can be employed in a multithreaded service to efficiently and securely analyze various aspects of code quality, such as code duplication and test coverage, among others. Additionally, these tools are extensively used in research to enable developers to analyze and enhance the quality of their code in a fast and effective manner. In this section, we will explore some of the most popular Python code quality tools, such as Pylint, Pytest's plugin on coverage, Pipreqs, and the Duplication Code Detection Tool, and how they can be utilized in the upcoming analysis.

3.8 Pylint



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

C:\Users\AUM>cd Desktop/geeksforgeeks

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

Figure 10 - Pylint Example

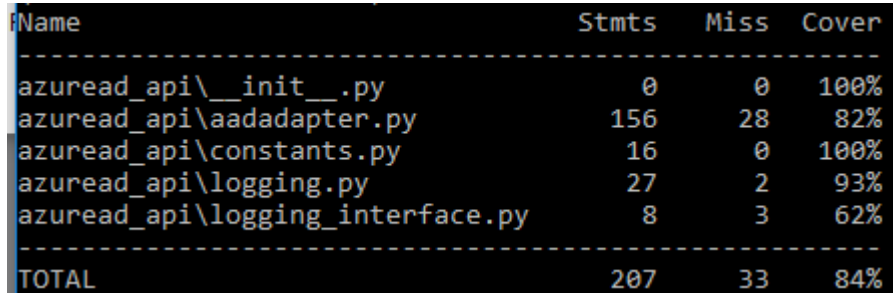
Pylint [35] is a powerful and widely used static code analysis tool for Python that can help developers identify and fix a wide range of problems in their code. With its five types of bug detection, Pylint is able to detect problems related to programming conventions, Python-specific issues, potential bugs, bad code smells, and fatal errors that can prevent code from running. By identifying these issues, developers can improve the quality of their code, increase maintainability, and reduce the risk of security vulnerabilities.

In addition to its powerful debugging capabilities, Pylint is also highly configurable, allowing developers to customize the tool's behavior to meet their specific needs. For example, specific bugs can be disabled if they are not relevant to a particular project or if the developer wishes to focus on other types of issues. This flexibility makes Pylint a versatile tool that can be used in a variety of development environments.

Pylint can also be called programmatically through its library, providing even more flexibility and ease of use. This feature allows developers to integrate it into their development workflows and automate code quality checks, reducing the risk of human error and improving overall performance. As a result, Pylint has become a widely used

tool in both industry and academic research, and its capabilities continue to evolve as the Python language and development practices change.

3.9 Pytest (--cov)



```
-----  
Name                               StmtS  Miss  Cover  
-----  
azuread_api\__init__.py            0      0   100%  
azuread_api\aadadapter.py          156    28    82%  
azuread_api\constants.py           16      0   100%  
azuread_api\logging.py              27      2    93%  
azuread_api\logging_interface.py     8       3    62%  
-----  
TOTAL                               207    33    84%
```

Figure 11 - Pytest --cov Example

As mentioned earlier, Pytest [21] is widely recognized as a popular third-party Python testing framework that aims to simplify the process of writing tests. In this study, Pytest was deliberately chosen as the testing framework due to its user-friendly nature and robust community support. The tooling developed in this project utilized Pytest for its testing requirements, facilitating rapid and efficient test case development. With its compatibility across multiple Python versions and operating systems, Pytest proved to be a flexible and dependable choice for testing the tool.

In addition to Pytest, Pytest-cov [36] was employed as a tool to measure code coverage in Python. Pytest-cov serves as a plugin and command-line utility for Pytest, offering supplementary functionalities to coverage.py. This includes the ability to generate XML or HTML reports as well as providing a user-friendly code coverage analysis through a web browser. However, it should be noted that the utilization of pytest-cov may increase the complexity of terminal commands as more masking options are incorporated.

3.10 Pipreqs

A terminal window with a dark background and light text. The terminal shows the command `$ pipreqs .` and its output: `INFO: Successfully saved requirements file in ./requirements.txt`. Below the output, the contents of the generated `requirements.txt` file are displayed: `numpy==1.21.4`, `pandas==1.3.4`, `pyinstrument==4.0.3`, and `typer==0.4.0`. The terminal window has three colored window control buttons (red, yellow, green) in the top-left corner.

```
# Save requirements in the current directory
$ pipreqs .
INFO: Successfully saved requirements file in
./requirements.txt

"""requirements.txt
numpy==1.21.4
pandas==1.3.4
pyinstrument==4.0.3
typer==0.4.0
"""
```

Figure 12 - Pipreqs Example

Pipreqs [24] is a widely used open-source tool that scans Python source code to identify its dependencies, which are then listed in a requirements file. This file contains all the third-party packages necessary to run the code and is necessary to reproduce the environment in which the code was developed. Pipreqs simplifies the process of creating this requirements file by scanning the source code and automatically identifying the packages that are used without including packages that are not required. This saves time and effort for developers, as they no longer need to manually identify and document each dependency.

A limitation of Pipreqs is that it only focuses on third-party packages and does not pay attention to system libraries and the Python interpreter. This means that the generated requirements file may not include all the necessary dependencies to fully reproduce the development or deployment environment. As a result, developers must be aware of this limitation and manually add any additional dependencies that may be necessary. Despite this limitation, Pipreqs remains a popular choice for creating requirements files due to its simplicity and ease of use, and it will be used in the tool of this study.

3.11 Duplicate code detection tool

```
Code duplication probability for smartcar_shield/src/control/differential/DifferentialControl.cpp
```

File	Similarity (%)
smartcar_shield/src/motor/digital/servo/ServoMotor.cpp	7.42
smartcar_shield/src/motor/analog/pwm/BrushedMotor.cpp	4.81
smartcar_shield/src/sensors/odometer/interrupt/DirectionalOdometer.cpp	2.00
smartcar_shield/src/sensors/odometer/interrupt/DirectionlessOdometer.cpp	4.31
smartcar_shield/src/sensors/heading/gyroscope/GY50.cpp	2.80
smartcar_shield/src/sensors/distance/ultrasound/ping/SR04.cpp	2.50
smartcar_shield/src/sensors/distance/ultrasound/i2c/SRF08.cpp	0.75
smartcar_shield/src/sensors/distance/infrared/analog/InfraredAnalogSensor.cpp	0.97
smartcar_shield/src/sensors/distance/infrared/analog/sharp/GP2Y0A21.cpp	0.73
smartcar_shield/src/sensors/distance/infrared/analog/sharp/GP2D120.cpp	0.59
smartcar_shield/src/sensors/distance/infrared/analog/sharp/GP2Y0A02.cpp	0.59
smartcar_shield/src/runtime/arduino_runtime/ArduinoRuntime.cpp	0.24
smartcar_shield/src/car/simple/SimpleCar.cpp	17.14
smartcar_shield/src/car/heading/HeadingCar.cpp	1.14
smartcar_shield/src/car/distance/DistanceCar.cpp	9.42
smartcar_shield/src/control/ackerman/AckermanControl.cpp	26.91

Figure 13 - Duplicate Code Detection Tool Example

The duplicate code detection tool [37] is a command-line utility developed in Python 3 that leverages the gensim Python library to detect similarities between files within a given code repository. This tool is intended to help developers identify and eliminate duplicate code within a software component, with the ultimate goal of improving the overall software architecture. The development of this tool started in the context of DAT265 (Software Evolution Project), and its creator is Dimitris Platis.

To use this tool, the user provides a directory or list of files to analyze. The gensim library is used to determine similarity between source code files, with default support for C, C++, Java, Python, and C# programming languages.

The duplicate code detection tool has several dependencies, including the nltk, gensim, and astor Python packages. Additionally, the punkt dataset must be downloaded from the nltk library for proper operation. Overall, this tool provides a simple yet effective means of identifying and eliminating code duplication within a software project, thereby promoting improved software design and maintainability.

4 App functionality

With the increasing reliance on software-based tools for scientific research, it is essential to have a clear understanding of the functionality of the tools used to ensure the validity and reliability of research findings. In this context, functionality is a critical aspect of any tool that seeks to support scientific research; it defines the characteristics and capabilities of the tool, and it is necessary to evaluate it to ensure that the tool meets the requirements of the research problem at hand. This chapter will focus on the functionality of implementing a tool designed to assess Python code quality through a multimetric analysis, with the aim of evaluating its usefulness and effectiveness in solving research problems in this area.

The application allows the user to easily log into a dedicated user interface through which they can start analyzing a GitHub repository. This interface is designed with a simple and intuitive layout that includes a text area where the user can enter the repository URL. Additionally, there are three options available to the user: start analysis, Gitlab CI, and show data.

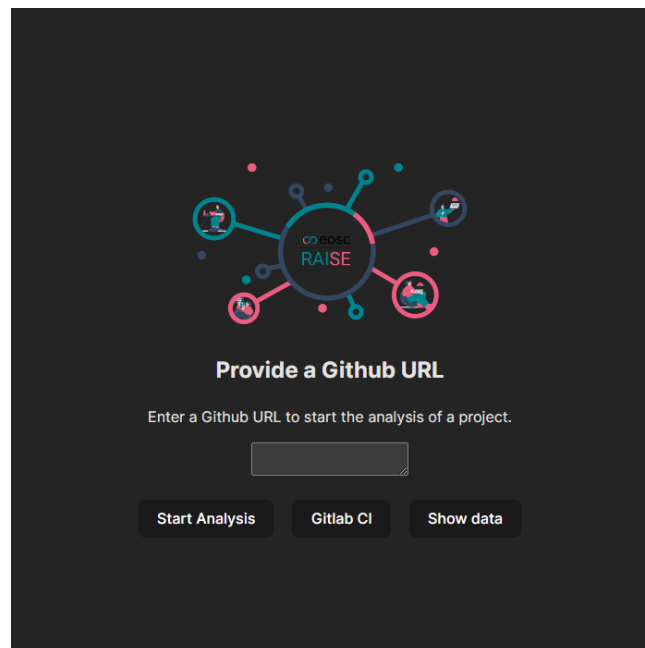


Figure 14 - Application user interface

By clicking on the first option, a “loading” component will appear, indicating that the project analysis is running. Depending on the size of the project and the number of large files included, the analysis may take some time to complete. It is important to note that the analysis is performed on 10 equidistant commits based on the total number of commits in the project, allowing the user to monitor the progress of the project. Once the analysis is complete, the user will be presented with a window that will display the results obtained from the analysis, as shown in the image below.



Figure 15 - Analysis results of a project

The specific example displays the project name, owner, GitHub URL, and the analyzed SHAs. Four charts, each displaying a different measure, are also displayed. First, the total statements are highlighted, i.e., a line of code that executes an action or a group of actions. It can be an assignment, a function call, a loop, a conditional statement, or any other command that the program executes. In the context of this reference, "Stmts" refers to the number of executable statements in each file, that is, the number of lines containing executable code. In addition, misses are also shown, which refer to the number of lines of code that were not executed during the test process. The higher the

number of failures, the lower the code coverage and the higher the chance of undiscovered bugs or errors.

Also, the total coverage that we have covered in previous chapters is shown. As this increases, it means that more and more statements are covered by the tests. This is generally a good sign, as it indicates that the tests are becoming more complete and covering more of the codebase. As coverage approaches 100%, it becomes more difficult to increase further, as there may be some edge cases or extreme scenarios that are difficult to test. However, striving for a high level of coverage can help ensure that the code is thoroughly tested and reduce the risk of bugs and errors in production. It is important to note, however, that achieving high coverage is not a guarantee of bug-free software, and additional testing and validation techniques may be needed to ensure that the software is working properly.

Finally, the dependencies of the code over time are also shown. Having a large number of dependencies in a codebase can have positive and negative effects. On the plus side, it allows for code reuse, making it easier to write and maintain code using pre-existing packages. On the downside, having too many dependencies can slow down code performance and make it harder to manage dependencies and ensure compatibility. So in this case, the developer must be ready to face some error in some dependency.

By selecting the second button, the user is presented with a designated text area where a YAML (.yaml) file can be copied and pasted into their own project. The YAML file, once added to the project, will allow any subsequent commits to be automatically parsed by the Gitlab Continuous Integration (CI) tool, provided the same repository has been previously parsed using the tool's first option. Adding a YAML file allows the analysis tool to be seamlessly integrated with the user's development workflow, promoting early and consistent identification of the metrics that will define code quality. An example of such a file is shown in the image:

```
build-job: # This job runs in the build stage, which runs first.
  stage: build
  script:
    - echo "Build stage..."
    - export GIT_URL=$CI_PROJECT_URL
    - export GIT_BRANCH=$CI_COMMIT_BRANCH
    - 'curl --request POST "http://195.251.210.147:8181/project_analysis/?
gitUrl=$GIT_URL&branch=$GIT_BRANCH
```




Figure 16 - Gitlab CI

The third button will direct the user to a screen displaying the findings of the analysis done on earlier analyses, which are preserved unless changes are made to the code. The data will be consistent and accurately represent the initial analysis, offering a trustworthy standard for comparison and additional research. The tool's functionality is enhanced by the capability to access and contrast previously generated data, which also makes it possible to follow a project's development over time effectively.

Through the application's back end, the tool offers sophisticated capabilities that enable a technically skilled user to carry out extra research. A user can begin a distinct study of the most recent commit on a specific branch in addition to completing historical studies, as was previously indicated, by sending a POST request with parameters that contain a GitHub URL and branch. The application also enables users to perform a GET request to retrieve previously examined projects, which shows them how far along their study is. The following is an illustration of the outcome that the user of such a request will produce:

```

1  {
2  "id": 6,
3  "gitUrl": "https://github.com/jerempa/PDF-OCR-and-data-analysis.git",
4  "owner": "jerempa",
5  "name": "PDF-OCR-and-data-analysis",
6  "directory": "C:\\Files\\Univeristy\\Service-Based-Assessment-of-Python-Code-Quality\\PDF-OCR-and-data-analysis",
7  "projectAnalysis": [ ...
132695 ],
132696 "singleAnalyzedProjectList": [],
132697 "sha": [
132698   "6f3882fb2e44ac93910d63baaeda503d50f0b804",
132699   "326c060dc51db378d6f62b9b9404201331d44d7",
132700   "76ed80b7eb5e0c50ad0ac78223b797c0bd86563e",
132701   "455e2fc44b34066819860d1c0c0ce444d4de0528ed",
132702   "e775c7b1c145ea352c7e172bc99395fbb7c7abfe",
132703   "aeceef942bb60bdcc80b4d920caa4df2a233136",
132704   "7acc10c7ccb75ede252cc47edde11361b548a10e",
132705   "98deceeb0a511000e3e05071f909b9334b6adaa",
132706   "84d0b7a1c07126077b16855b69bfab85647ace31",
132707   "2127623594307c5ae9b42a34eb7eafbd04a22afe"
132708 ]
132709 }

```

Figure 17 - GET API JSON response

In the case shown, the JSON file returned to the user contains additional details such as "projectAnalysis" and "singleAnalyzedProjectList" lists. The former provides a historical analysis, while the latter provides a unique analysis. The answer for such a parsing example is shown in the example below:

```

8  {
9  "id": 27,
10 "gitUrl": "C:\\Files\\Univeristy\\Service-Based-Assessment-of-Python-Code-Quality\\PDF-OCR-and-data-analysis",
11 "owner": "jerempa",
12 "name": "PDF-OCR-and-data-analysis",
13 "directory": "C:\\Files\\Univeristy\\Service-Based-Assessment-of-Python-Code-Quality\\PDF-OCR-and-data-analysis",
14 "dependencies": [ ...
26 ],
27 "dependenciesCounter": 11,
28 "files": [ ...
13274 ],
13275 "totalCoverage": 0,
13276 "totalMiss": 1606,
13277 "totalStmts": 1606,
13278 "sha": "6f3882fb2e44ac93910d63baaeda503d50f0b804"
13279 }

```

Figure 18 - Example of Project Analysis GET Response

This section presents an analysis of the last commit to the project. In the "dependencies" property, the list of dependencies is displayed, giving the developer an insight into the project's dependencies and possible sources of errors. However, it is particularly interesting to explore the contents of the "files" list, which provides a detailed breakdown of the project's files.


```
28     "files": [  
29         {  
30             "id": 282,  
31             "name": "correct_seasons.py",  
32             "stmts": 20,  
33             "miss": 20,  
34             "coverage": 0,  
35             "comments": [...  
72         ],  
73         "rating": 4.0,  
74         "previousRating": null,  
75         "similarity": {  
76             "img_to_string.py": 14.0,  
77             "debt_visualization.py": 7.55,  
78             "extracting_data.py": 8.19,  
79             "values_for_analysis.py": 10.07,  
80             "transfermarkt_data_visualization.py": 6.92,  
81             "fetch_data_from_transfermarkt.py": 9.36,  
82             "download_pdfs.py": 9.12,  
83             "df_operations.py": 3.84,  
84             "data_visualization.py": 9.09,  
85             "img_conversion_and_processing.py": 4.76,  
86             "fetch_data_from_worldfootball.py": 11.8,  
87             "main.py": 9.09,  
88             "fetch_season_data_from_wiki.py": 7.92,  
89             "calculations.py": 6.99,  
90             "errors.py": 5.26,  
91             "__init__.py": 1.1,  
92             "financial_statement_data_visualization.py": 9.38,  
93             "file_handling.py": 10.38  
94         },  
95         "projectName": "PDF-OCR-and-data-analysis",
```

Figure 19 - Example of file response from GET API

This image shows detailed information for each file, including statements, misses, and coverage, sourced from Pytest. In addition, Pylint's comments, ratings, and previous ratings are displayed, as is each file's percentage of similarities to other files, obtained by the Duplicate Code Detection Tool. This analysis is performed for each file and project and can be generated for each of the ten commits to a user-selected project.

Users can update existing projects in the database (PostgreSQL) or delete them. These features enable users to perform targeted analysis and manage their analyzed projects with greater flexibility and control.

The suggested tool's analysis method comprises a number of processes that are carried out in an automated and systematic way. The tool first accepts a GitHub URL as input before utilizing the JGit library to clone the appropriate repository to the local environment. The utility examines the files once the repository has been successfully cloned in order to choose just those with the.py extension. The utility employs five threads to speed up the parsing process, four of which are set aside to execute the Pylint tool on the chosen files consecutively. The fifth is in charge of executing the remaining

tests, if any, that are present in the directory. The information obtained after all analyses is kept in a PostgreSQL database.

This database serves as the main source of information, which can then be presented to the user. The analysis report generated by the tool includes various parameters such as code quality, test coverage, and code duplication, among others. Through the use of such metrics, developers can gain valuable insight into the performance of their code and make informed decisions about refactoring, optimizing, and improving the overall quality of their code base.

In conclusion, the application functionality serves as a powerful tool for identifying code similarities and analysing software evolution through historical analysis. Leveraging advanced techniques such as natural language processing, this tool provides users with comprehensive and accurate insights into coverage, code duplication, and dependencies. Its user-friendly interface and easy-to-use features empower developers to take control of their projects and optimize their workflow.

One of the notable strengths of this tool is its seamless integration with GitLab CI, enabling developers to automate their analysis process and streamline their codebase. The robust back-end of the application ensures efficient and reliable performance, while the integration with GitLab CI adds a layer of automation and optimization to the development process.

Overall, the application functionality discussed in this section underscores the significant impact of modern technologies on software engineering and showcases the potential of data-driven approaches in software development. By leveraging advanced techniques and automation, this tool empowers developers to make informed decisions and optimize their codebase, ultimately leading to higher-quality and more maintainable software applications.

5 Presentation of the Application

The provided application was created using Spring Boot, a well-known Java-based framework that offers a complete infrastructure for creating and delivering web-based applications, as was previously mentioned. By providing pre-built solutions for typical issues that arise while creating web applications, Spring Boot is renowned for its ability to speed up the development process. In this situation, the application makes use of Spring Boot's strengths to provide a reliable and high-quality software solution. Additionally, using Java as the primary programming language assures that the application will run on a variety of platforms and will have the performance and scalability required to satisfy the demands of contemporary online applications.

We'll start by examining the crucial aspects of analysis, from the point at which a GitHub URL enters the code to the point at which the data is chosen and stored.

```
public void cloneRepository(Git git, String owner, String repoName, String cloneDir, String branch) throws Exception {  
  
    RestTemplate restTemplate = new RestTemplate();  
    restTemplate.getInterceptors().add((request, body, execution) -> execution.execute(request, body));  
  
    String apiUrl = "https://api.github.com/repos/" + owner + "/" + repoName;  
    HttpHeaders headers = new HttpHeaders();  
    headers.setAccept(Collections.singletonList(MediaType.APPLICATION_JSON));  
    HttpEntity<String> entity = new HttpEntity<>(headers);  
    ResponseEntity<Map> response = restTemplate.exchange(apiUrl, HttpMethod.GET, entity, Map.class);  
    Map<String, Object> json = response.getBody();  
    String cloneUrl = (String) json.get("clone_url");  
  
    git.cloneRepository().setURI(cloneUrl).setDirectory(new File(cloneDir)).call();  
    if (branch != null) {  
        Ref ref = git.checkout().  
            setCreateBranch(true).  
            setName(branch).  
            setUpstreamMode(CreateBranchCommand.SetupUpstreamMode.TRACK).  
            setStartPoint("origin/" + branch).  
            call();  
    }  
}
```

Figure 20 - Clone repository Method

The method starts by accepting parameters such as the Git object, repository owner, repository name, local directory to clone to, and optional branch information. To interact with GitHub's REST API, the method creates a new RestTemplate object and adds an interceptor to it, setting the accept headers to JSON. It then constructs the API URL using the repository owner and name, forming the query to fetch data from

GitHub's REST API. The method sends a GET request to this API endpoint with the previously specified headers and retrieves the JSON response.

From the JSON response, the method extracts the clone URL, which is the URL required to clone the repository. This clone URL is a critical piece of information needed to clone the repository locally using JGit, a Java library for Git operations. After obtaining the clone URL, the method uses JGit to clone the repository to the specified directory. If a branch is specified, the method creates a new branch using JGit, setting the "SetUpstreamMode" mode to TRACK. This means that the newly created branch will track changes from the remote branch of the same name. Finally, the method sets the starting point of the branch to the remote branch.

In summary, the method uses RestTemplate to interact with GitHub's REST API, retrieves the clone URL from the JSON response, and then uses JGit to clone the repository locally and optionally create and set up a branch for tracking changes from the remote branch. This process allows the method to clone the repository and set up a branch for further processing.

```
106 @ private List<String> captureSHAs(String gitUrl, String owner, String name) throws Exception {
107     List<String> commitSHAs = new ArrayList<>();
108
109     RepositoryService repoService = new RepositoryService();
110     CommitService commitService = new CommitService();
111
112     org.eclipse.egit.github.core.Repository repository = repoService.getRepository(owner, name);
113
114     List<RepositoryCommit> commits = commitService.getCommits(repository);
115
116     for (RepositoryCommit commit : commits) {
117         commitSHAs.add(commit.getSha());
118     }
119
120     return commitSHAs;
121 }
```

Figure 21 - Capture SHAs code

The private method "captureSHAs" is defined with three parameters: gitUrl, owner, and name. It returns a list of SHA commits.

To fetch the list of commits, the method utilizes the `RepositoryService` and `CommitService` classes from the `org.eclipse.egit.github.core` package. These classes provide functionality for interacting with the GitHub REST API to fetch repositories and commit information.

The method first queries the `RepositoryCommit` objects for a specific repository owned by the specified owner and with the specified name using the `RepositoryService` and `CommitService` classes. It then iterates over the list of `RepositoryCommit` objects and extracts the SHA of each commit. The extracted SHA is then added to the "commitSHAs" list. Finally, the method returns the "commitSHAs" list, which contains the SHA values of all the commits retrieved from the repository.

In summary, the "captureSHAs" method uses the `RepositoryService` and `CommitService` classes to fetch the list of commits from a specific repository, extracts the SHA values of each commit, and returns a list of these SHA values.

```
51     if (flag && branch == null) {
52         List<String> SHAs = captureSHAs(gitUrl, project.getOwner(), project.getName());
53         List<String> selectedSHAs = new ArrayList<>();
54
55         if (SHAs.size() > 10) {
56             selectedSHAs.add(SHAs.get(0));
57             int step = SHAs.size() / 9;
58
59             for (int i = 1; i < SHAs.size(); i += step) {
60                 selectedSHAs.add(SHAs.get(i));
61             }
62         } else {
63             selectedSHAs = SHAs;
64         }
65
66         project.setSHA(selectedSHAs);
67
68         for (String sha: selectedSHAs) {
69             ObjectId commitId = repo.resolve(sha);
70             git.checkout().setName(commitId.getName()).call();
71             projectAnalysisList.add(projectAnalysisService.runCommand(project, sha, homeDirectory));
72         }
73     } else {
74         String tempSha = projectAnalysisService.findSHA(project.getOwner(), project.getName());
75         project.setSHA(Collections.singletonList(tempSha));
76         if (branch == null) {
77             projectAnalysisList.add(projectAnalysisService.runCommand(project, tempSha, homeDirectory));
78             project.setProjectAnalysis(projectAnalysisList);
79         } else {
80             singleAnalyzedProjects.add(projectAnalysisService.runCommand(project, tempSha, homeDirectory));
81             project.setSingleAnalyzedProjectList(singleAnalyzedProjects);
82         }
83     }
}
```

Figure 22 - Determination of analysis

If the flag is set to true and the branch is null, the code performs a unique analysis of the project. It first calls the "captureSHAs" method to get a list of SHA commits from the repository. It then selects up to 10 SHA values for analysis, following a specific logic. If there are more than 10 SHA values, the code picks the first one and evenly selects 9 more SHA values from the remaining list. These selected SHA values are then saved in the project object for further analysis.

Next, the code loops through each selected SHA value and calls the "runCommand" method from the "projectAnalysisService" with the current project, SHA value, and home directory as parameters. The results of each "runCommand" call are added to the "projectAnalysisList" on the project object, which accumulates the analysis results for each SHA value.

On the other hand, if the flag is set to false or the branch is null, the code performs a historical analysis of the project. It calls the "findSHA" method from the "projectAnalysisService" to get the latest SHA value for the project, which is then stored in the project object. If the branch is null, the "runCommand" method is called with the latest SHA hash, and the results are added to the "projectAnalysisList" on the project object. Otherwise, the "runCommand" method is called, and the results are added to the "singleAnalyzedProjects" list on the project object, which stores the analysis results for a single SHA value (in the case of historical analysis with a specific branch).

```
106     public String findSHA(String owner, String repoName) {
107         RestTemplate restTemplate = new RestTemplate();
108         restTemplate.getInterceptors().add((request, body, execution) -> execution.execute(request, body));
109
110         String apiUrl = "https://api.github.com/repos/" + owner + "/" + repoName + "/commits";
111         HttpHeaders headers = new HttpHeaders();
112         headers.setAccept(Collections.singletonList(MediaType.APPLICATION_JSON));
113         HttpEntity<String> entity = new HttpEntity<>(headers);
114         ResponseEntity<Map[]> response = restTemplate.exchange(apiUrl, HttpMethod.GET, entity, Map[].class);
115         Map<String, Object> json = response.getBody()[0];
116         String sha = (String) json.get("sha");
117
118         return sha;
119     }
```

Figure 23 - Search for unique SHA

Unlike the captureSHAs method, which retrieves all SHAs from the repository, this method only retrieves the SHA of the most recent commit on the default branch. This

code defines a method called findSHA that takes two parameters, owner and repoName, and returns a string representing the SHA of the last commit in the given repository. It uses the RestTemplate class from Spring to send a GET request to the GitHub API and retrieve information about the most recent commits. The API URL is created by concatenating the owner and repoName parameters. The returned JSON data is parsed into a Map object using the ResponseEntity class from Spring. The SHA of the last commit is then extracted from the first element of the commit information table returned by the API. This method is used in the code snippet provided to retrieve the SHA of the last commit on the default branch.

```
52 Path dir = Paths.get(mainProjectAnalysis.getDirectory());
53 File folder = new File(mainProjectAnalysis.getDirectory());
54 File[] listOfFiles = folder.listFiles();
55 ArrayList<ProjectFile> fileList = new ArrayList<>();
56 HashMap<String, Double> fileSimilarityList = new HashMap<>(listOfFiles.length);
57 try (Stream<Path> stream = Files.walk(dir)){
58     stream
59     .filter(Files::isRegularFile)
60     .filter(path -> path.toString().endsWith(".py"))
61     .forEach(file -> {
62         ProjectFile newFile = new ProjectFile(file.toFile(), fileSimilarityList);
63         newFile.setName(file.toFile().getName());
64         fileList.add(newFile);
65         newFile.setProjectName(mainProjectAnalysis.getName());
66         mainProjectAnalysis.getFiles().add(newFile);
67     });
68 }
69
70 mainProjectAnalysis.setFiles(fileList);
```

Figure 24 - Selecting files with the .py file extension

As for the analysis itself, the first operation is to select those files that have a “.py” extension. This code block processes files in a directory and creates a list of ProjectFile objects to represent each file. It first gets the directory path from mainProjectAnalysis, which is an instance of the ProjectAnalysis class. It then creates a File object for the directory and gets a list of all the files in the directory using the listFiles() method.

It then initializes an array list called fileList and a hash map called fileSimilarityList. fileSimilarityList is intended to store the similarity scores between each pair of files in the directory but is currently empty. The code then uses a

try-with-resources statement to create a stream of all the paths in the directory using the Files.walk() method and filters the paths to include only regular files (not directories) ending with the extension ".py" using the filter() method.

For each file path, the code creates a new ProjectFile object using the file and fileSimilarityList variables, sets the name and projectName attributes of the ProjectFile object, adds the ProjectFile object to the fileList list, and finally adds the ProjectFile object to the file list of its main ProjectAnalysis object.

Finally, the fileList is set to the files attribute of the mainProjectAnalysis object using the setFiles() method.

```
73     int fileListSize = fileList.size()/4;
74     List<ArrayList<ProjectFile>> chunkedLists = new ArrayList<>();
75     for (int i = 0; i < 4; i++) {
76         int startIndex = i * fileListSize;
77         int endIndex = startIndex + fileListSize;
78         if (i == 3) {
79             endIndex = fileList.size();
80         }
81         chunkedLists.add(new ArrayList<>(fileList.subList(startIndex, endIndex)));
82     }
83
84     Thread firstThread = new Thread(testsRunnable(mainProjectAnalysis, fileList, mainProjectAnalysis.getDirectory()));
85     Thread secondThread = new Thread(pyLintRunnable(chunkedLists, mainProjectAnalysis, index 0));
86     Thread thirdThread = new Thread(pyLintRunnable(chunkedLists, mainProjectAnalysis, index 1));
87     Thread fourthThread = new Thread(pyLintRunnable(chunkedLists, mainProjectAnalysis, index 2));
88     Thread fifthThread = new Thread(pyLintRunnable(chunkedLists, mainProjectAnalysis, index 3));
89
90     firstThread.start();
91     secondThread.start();
92     thirdThread.start();
93     fourthThread.start();
94     fifthThread.start();
95
96
97     firstThread.join();
98     secondThread.join();
99     thirdThread.join();
100    fourthThread.join();
101    fifthThread.join();
```

Figure 25 – Threads

Here, the code divides the list of files into four approximately equal chunks using integer division and then creates a thread for each chunk using the pyLintRunnable method. The first thread is created separately and uses the testsRunnable method with the entire file list and mainProjectAnalysis as arguments. After all five threads are created, the code starts them all and then waits for each one to finish using the join method.

Overall, this code implements a multi-threaded approach to running tests and the Pylint tool in parallel on the project files.

```
168 public Runnable testsRunnable(ProjectAnalysis mainProjectAnalysis, ArrayList<ProjectFile> fileList, String homeDirectory) {
169     return () -> {
170         try {
171             executeCommand(mainProjectAnalysis, fileList, command: "pytest --cov=", homeDirectory);
172             executeCommand(mainProjectAnalysis, fileList, command: "python3 -m ignore " + System.getProperty("user.dir") + File.separator + "duplicate-code-detection-tool/duplicate_code_detection.py -d ", homeDirectory);
173             do {
174                 executeCommand(mainProjectAnalysis, fileList, command: "pipreqs --force ", homeDirectory);
175             } while (!(new File(pathname: homeDirectory + "/requirements.txt")).exists());
176         } catch (IOException e) {
177             throw new RuntimeException(e);
178         } catch (InterruptedException e) {
179             throw new RuntimeException(e);
180         }
181         System.out.println("PipReqs counted " + countLineBufferedReader(mainProjectAnalysis, fileName: homeDirectory + "/requirements.txt") + " dependencies on this project.");
182         new File(pathname: homeDirectory + "/requirements.txt").delete();
183     };
184 }
```

Figure 26 – First Runnable

This is a method that returns an executable that runs multiple tests on a project. The tests include running Pytest to generate code coverage information, running the Duplicate Code Detection Tool, and running pipreqs to generate a list of project dependencies. The results of these tests are recorded on the console. Finally, the temporary requires.txt file generated by pipereqs is deleted.

```
155 public Runnable pylintRunnable(List<ArrayList<ProjectFile>> chunkedLists, ProjectAnalysis mainProjectAnalysis, int index){
156     return () -> {
157         for (ProjectFile file : chunkedLists.get(index))
158             try {
159                 executeCommand(mainProjectAnalysis, chunkedLists.get(index), command: "pylint ", String.valueOf(file.getFirstFile()));
160             } catch (IOException e) {
161                 throw new RuntimeException(e);
162             } catch (InterruptedException e) {
163                 throw new RuntimeException(e);
164             }
165     };
166 }
```

Figure 27 - Pylint Runnable

A pylintRunnable method is defined here that returns a Runnable object that executes a loop that iterates over a chunk of ProjectFile objects stored in the chunkedLists parameter at the given index. For each ProjectFile object, it executes the executeCommand method passed to mainProjectAnalysis, the chunk of ProjectFile objects at the given index, the pylint command, and the file path of the ProjectFile object. Essentially, this method is responsible for running pylint on each file in a chunk of ProjectFile objects. The method takes a list of ProjectFile chunks and a ProjectAnalysis object as inputs and returns a Runnable object that runs the Pylint tool on each file in a

given chunk. This runnable object can then be executed in a separate thread to parallelize the parsing of the files.

```
121 public void executeCommand(ProjectAnalysis projectAnalysis, ArrayList<ProjectFile> fileList, String command, String destination) throws IOException, InterruptedException {
122     ArrayList<String> similarityResponse = new ArrayList<>();
123     ArrayList<String> commentsResponse = new ArrayList<>();
124     Process p = Runtime.getRuntime().exec(command + destination);
125     InputStream is = p.getInputStream();
126     try (BufferedReader reader = new BufferedReader(new InputStreamReader(is))) {
127         List<String> lines = reader.lines().collect(Collectors.toList());
128         for (String line : lines) {
129             if(command.startsWith("python3 -W ignore " + System.getProperty("user.dir")+ File.separator + "duplicate-code-detection-tool/duplicate_code_detection.py -d ")) {
130                 if(line.contains("Code duplication probability for") || line.startsWith(projectAnalysis.getDirectory())) {
131                     similarityResponse.add(line);
132                 }
133             } else if(command.startsWith("pylint")) {
134                 if(line.contains("Your code has been rated at") || line.startsWith("***** Module ") || line.startsWith(projectAnalysis.getName())) {
135                     commentsResponse.add(line);
136                 }
137             } else if(command.startsWith("pytest")) {
138                 if(line.startsWith(projectAnalysis.getName()) || line.startsWith("TOTAL")) {
139                     storeDataInObjects(projectAnalysis, fileList, line, command);
140                 }
141             }
142             System.out.println(line);
143         }
144         if (similarityResponse.size()>0){
145             storeSimilarity(similarityResponse, fileList, projectAnalysis);
146         }
147         if (commentsResponse.size()>0) {
148             storeComments(commentsResponse, fileList, projectAnalysis);
149         }
150     }
151 }
```

Figure 28 - Execute Command Method

At this point, the executeCommand method accepts a ProjectAnalysis object, a list of ProjectFile objects, a command to execute, and a destination. The method executes the specified command with the given destination as an argument and records the standard output of the process. Depending on the command, it processes the output and stores the data in either similarityResponse lists or commentsResponse lists if the output contains relevant data to detect code duplication or generate code analysis comments, respectively. The method also calls the storeDataInObjects method to store the test coverage data if the command is intended to run Pytest. Finally, it prints each line of output to the console

```

25     public static long countLineBufferedReader(ProjectAnalysis projectAnalysis, String fileName) {
26         ArrayList<String> dependencies = new ArrayList<>();
27         String line;
28         try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
29             while ((line = reader.readLine()) != null) {
30                 dependencies.add(line);
31             }
32         } catch (IOException e) {
33             e.printStackTrace();
34         }
35         storeDepsInProject(projectAnalysis, dependencies, dependencies.size());
36         return dependencies.size();
37     }
38 }

```

Figure 29 - Counting lines of requirements.txt

This method counts the number of lines in a text file and also stores the contents of the file, in this case, a list of dependencies, in a ProjectAnalysis object. It first creates an empty array list to store the lines of the file. It then uses a buffered reader object to read the contents of the file line by line and add each line to the dependencies list. Once the entire file has been read, the method stores the dependencies in the ProjectAnalysis object and returns the total number of lines in the file.

Each terminal answer is handled by one of three methods, as can be seen in the executeCommand method.

A ProjectAnalysis object, an ArrayList of ProjectFile objects, a String answer, and a String command are the four inputs for the first method, storeDataInObjects, which accepts four parameters in total.

```

40 @ 1 usage  George David Apostolidis +1 *
41     public static void storeDataInObjects(ProjectAnalysis projectAnalysis, ArrayList<ProjectFile> fileList, String response, String command) {
42         if(command.startsWith("pytest --cov=")) {
43             if(response.startsWith(projectAnalysis.getName())){
44                 Pattern filePattern = Pattern.compile(regexPattern(command, request: "file"));
45                 Matcher fileMatcher = filePattern.matcher(response);
46                 Boolean fileFind = fileMatcher.find();
47
48                 Pattern covPattern = Pattern.compile(regexPattern(command, request: "cov"));
49                 Matcher covMatcher = covPattern.matcher(response);
50                 Boolean covFind = covMatcher.find();
51
52                 Pattern missPattern = Pattern.compile(regexPattern(command, request: "miss"));
53                 Matcher missMatcher = missPattern.matcher(response);
54                 Boolean missFind = missMatcher.find();
55
56                 Pattern stmtsPattern = Pattern.compile(regexPattern(command, request: "stmts"));
57                 Matcher stmtsMatcher = stmtsPattern.matcher(response);
58                 Boolean stmtsFind = stmtsMatcher.find();
59             }

```

Figure 30 - Pytest Response Method

The method first checks if the command starts with the string "pytest --cov=". If it does, then it checks if the response starts with the name of the projectAnalysis object. If it does, it proceeds to parse the response using regular expressions to extract coverage, miss, and statement information for each file in the list of files that matches the filename pattern specified in the command.

If the response starts with "TOTAL", it extracts the total coverage, misses, and statements using regular expressions and sets the corresponding fields in the projectAnalysis object.

The regular expressions used to extract information are created using the regexPattern method, which takes a string and a pattern identifier and returns a string that can be used as a regular expression to match the desired information in the response string.

```
113 @ public static void storeSimilarity(ArrayList<String> similarityResponse, ArrayList<ProjectFile> fileList, ProjectAnalysis projectAnalysis){
114     String mainFile = "";
115     int position = 0;
116     Map<String, HashMap<String, Double>> similarityMap = new HashMap<>();
117
118     for(int i=0; i < similarityResponse.size(); i++) {
119         if (similarityResponse.get(i).contains("Code duplication probability for")) {
120             fileList.get(position).setSimilarity(similarityMap.get(mainFile));
121             mainFile = "";
122             Pattern mainFilePattern = Pattern.compile(regexPattern(command: "duplication", request: "mainFile"));
123             Matcher mainFileMatcher = mainFilePattern.matcher(similarityResponse.get(i));
124             Boolean fileFind = mainFileMatcher.find();
125
126             if (fileFind) {...}
127         } else if (similarityResponse.get(i).startsWith(projectAnalysis.getDirectory())) {
128             Pattern filePattern = Pattern.compile(regexPattern(command: "duplication", request: "file"));
129             Matcher fileMatcher = filePattern.matcher(similarityResponse.get(i));
130             Boolean fileFind = fileMatcher.find();
131
132             Pattern similarityPattern = Pattern.compile(regexPattern(command: "duplication", request: "similarity"));
133             Matcher similarityMatcher = similarityPattern.matcher(similarityResponse.get(i));
134             Boolean similarityFind = similarityMatcher.find();
135             if (fileFind) {...}
136         }
137     }
138     fileList.get(position).setSimilarity(similarityMap.get(mainFile));
139 }
```

Figure 31 - Store Similarity Method

The code above represents the second function, storeSimilarity, which has three inputs: a ProjectAnalysis object, an array list of string-type objects, and an array list of ProjectFile objects. This method's goal is to interpret the similarity data received as input and save it in the relevant ProjectFile objects, which house details on the project files. A

crucial indicator for assessing the quality and maintainability of software is similarity data, which shows the extent of code duplication among project files.

The method starts by initializing some variables, including a hashmap named `similarityMap` that will be used to store the similarity data for each file. The `similarityResponse` parameter is then iterated over, with each element in the list representing a row of the input data.

Inside the loop, the code checks to see if the current line contains the string "Code duplication probability for." If this happens, subsequent lines will contain similarity data for a particular file. The method extracts the name of the file being parsed and creates a new `similarityMap` entry for it. If the current line contains a different file name than the previous line, the method updates the position variable to match the current file being parsed.

If the current line does not contain a file name but starts with the root directory of the project, this means that the line contains similarity data for a specific file. The method extracts the filename and similarity score from the string and stores them in the `similarityMap` under the appropriate filename.

Finally, after all the lines have been processed, the similarity data is stored in the `ProjectFile` objects. This is done by setting the similarity field of each `ProjectFile` object to the corresponding entry in the `similarityMap`. The method uses the position variable to specify which `ProjectFile` object to update and the `mainFile` variable to specify which similarity data to associate with each file.

```

161 @ public static void storeComments(ArrayList<String> commentsResponse, ArrayList<ProjectFile> fileList, ProjectAnalysis projectAnalysis)
162     String mainFile = "";
163     List<Comment> comments = new ArrayList<>();
164     ProjectFile currentProjectFile = null;
165
166     for (int i = 0; i < commentsResponse.size(); i++) {
167         String currentLine = commentsResponse.get(i);
168         if (currentLine.startsWith("***** Module")) {
169
170             Pattern filePattern = Pattern.compile(regexPattern(command: "pylint", request: "file"));
171             Matcher fileMatcher = filePattern.matcher(currentLine);
172             Boolean fileFind = fileMatcher.find();
173             if (fileFind) {...}
174         } else if (currentLine.startsWith(projectAnalysis.getName())) {
175             comments.add(new Comment(currentLine.replace(target: " ", replacement: "\\ ")));
176         } else if (currentLine.contains("Your code has been rated at")) {
177             if (currentProjectFile != null) {...}
178         }
179     }
180 }
181
182 }

```

Figure 32 - Store Comments Method

The storeComments method is represented by the above code. It takes three parameters: a ProjectAnalysis object called projectAnalysis, a ProjectFile object called fileList, and an array list of strings called commentsResponse.

The method iterates through the commentsResponse ArrayList using a for loop, examining each row. If the line begins with "***** Module", a regular expression pattern is used to match the filename of the current project file, and the corresponding ProjectFile object is in the fileList ArrayList. If the current line contains the project name, a new Comment object is created and added to the comments array list, while if it contains the string "Your code has been rated at ", the comments are added to the currentProjectFile object as comments, and a regular expression pattern is used to extract the numeric rating and previous rating values from the line. These values are then set to the currentProjectFile object.

In general, this method processes the output of a static code analysis tool (in this case, Pylint) to extract and store code comments and reviews in the corresponding ProjectFile objects.

```

204 private static String regexPattern(String command, String request){
205     Boolean isWindows = System.getProperty("os.name").toLowerCase().contains("win");
206
207     if(isWindows){...}else {
242         if(command.startsWith("pytest")){...}else if (command.startsWith("duplication")){...}else if(command.startsWith("pylint")){
267         if(request.equals("file")){...}else if(request.equals("rating")) {
270             return "at (\\b[0-9]+\\.?[0-9]+)/10";
271         }else if(request.equals("previousRating")) {
272             return "run: (\\b[0-9]+\\.?[0-9]+)/10";
273         }
274     }
275 }
276
277 return "";
278 }
279

```

Figure 33 - Regex Patterns

In the above methods, there seems to be a specific one named `regexPattern` that takes two string parameters, `command` and `request`, and returns a string regular expression based on the input parameters.

The purpose of this method is to generate regular expressions for parsing the output of various command-line tools used in software testing and code analysis, such as `pytest`, `duplication`, and `pylint`. The regular expressions generated by this method are used to extract specific information from the output of these tools, such as code coverage percentage, number of duplicate lines of code, and code quality score. The method first checks whether the current operating system is Windows or not by checking the value of the `os.name` system property. Depending on the operating system, the method returns different regular expressions for different requests.

For example, if the command is `pytest` and the request is `cov`, the method returns a regular expression that matches a string containing a number representing the percentage of code coverage, such as "95%." Similarly, if the command is a duplicate and the request is a match, the method returns a regular expression that matches a string containing a number representing the percentage of code similarity, such as "89.2%."

The regular expressions returned by this method are used by the calling code to parse the output of the command-line tools and extract the relevant information.

```

92 @      public void deleteDirectory(File directory) throws IOException {
93         File[] files = directory.listFiles();
94         if (files != null) {
95             for (File file : files) {
96                 if (file.isDirectory()) {
97                     deleteDirectory(file);
98                 } else {
99                     file.delete();
100                }
101            }
102        }
103        directory.delete();
104    }

```

Figure 34 - Delete Directory Method

In the last part of the analysis, the cloned repository directory should be deleted. This method does just that. Takes a File object representing a directory and recursively deletes all its contents, including subdirectories and files. The method starts by listing all the files and subdirectories in the specified directory. It then iterates through the list of files and subdirectories, recursively calling the deleteDirectory method if a subdirectory is encountered. For each file encountered, it calls the delete() method to delete the file. After all files and subdirectories are deleted, the method deletes the specified directory itself.

In conclusion, the methods presented in this section demonstrate the effective use of modern programming techniques and tools to create a high-quality, scalable, and reliable web application. Careful analysis and application of these methods are critical to ensuring the success of any software project. Therefore, the insights gained from this analysis will undoubtedly prove valuable to developers looking to build similar applications in the future and serve as a testament to the power of Spring Boot and Java in developing robust and scalable web-based systems.

6 Results

The software created with Spring Boot and Java has been executed on various scientific projects such as TorchIO, DPPy, and others, highlighting its versatility and ability to extract data and results across different fields of study. With its flexibility and adaptability, the tool can be applied in domains ranging from physics and statistics to medicine, providing researchers with a scalable and resilient solution to extract and analyze data with accuracy and efficiency. The application's potential impact on scientific research is substantial, as it can provide valuable assistance to researchers in extracting and analyzing data. Moreover, its simplicity of use and extensive documentation make it accessible to researchers with varying degrees of programming ability.

6.1 TorchIO

TorchIO [38], an open-source Python toolkit, is specifically designed to enhance the loading, preprocessing, augmentation, and patch-based sampling of medical images for deep learning tasks. It is built on the PyTorch coding language and incorporates popular libraries for efficient medical image processing, making it ideal for handling large volumes of data during neural network training. With TorchIO, researchers and practitioners can easily create, duplicate, trace, and extend image transforms, providing flexibility and adaptability to various use cases. Notably, TorchIO is well-suited for test-time augmentation and estimation of aleatoric uncertainty in segmentation tasks, as it offers transforms that can be inverted. In addition to its generic preprocessing and augmentation techniques, TorchIO also provides specialized MRI-specific artifacts, making it a comprehensive tool for medical image analysis in deep learning workflows. Its user-friendly interface and wide range of capabilities make TorchIO a valuable resource for researchers and practitioners working in the field of medical imaging.

Along with a wealth of functionality, TorchIO offers users a wealth of resources and support. TorchIO's official website, <http://torchio.rtfd.io/>, contains the source code, in-depth tutorials, and documentation. By using the command `pip install torchio`, users can quickly install the package from the Python Package Index (PyPI). Additionally, TorchIO has a handy command-line interface that enables users to apply transforms to

image files without having to be proficient in Python programming, making it more widely usable. Moreover, TorchIO provides a graphical user interface (GUI) within a TorchIO extension in 3D Slicer, a well-known medical image processing program, enabling users to easily and intuitively see the consequences of changes. TorchIO is a potent and straightforward framework for medical image analysis in deep learning applications thanks to these resources, which offer thorough support and simplicity for users.

6.1.1 Results

In the analysis performed for the TorchIO tool, 10 commits that were present in the default branch were analyzed over a period of time. The project was named TorchIO and was owned by Fepegar. The analysis started by recording this information and then delving into the commits. By analyzing them, we are able to gain insights into the development process, the evolution of the codebase, and the different features that were added or improved. This information can prove useful not only for the development team but also for other researchers who are interested in building similar tools for medical image processing. The results of the analysis are depicted in the images below, providing a comprehensive understanding of the changes made to the project over time.

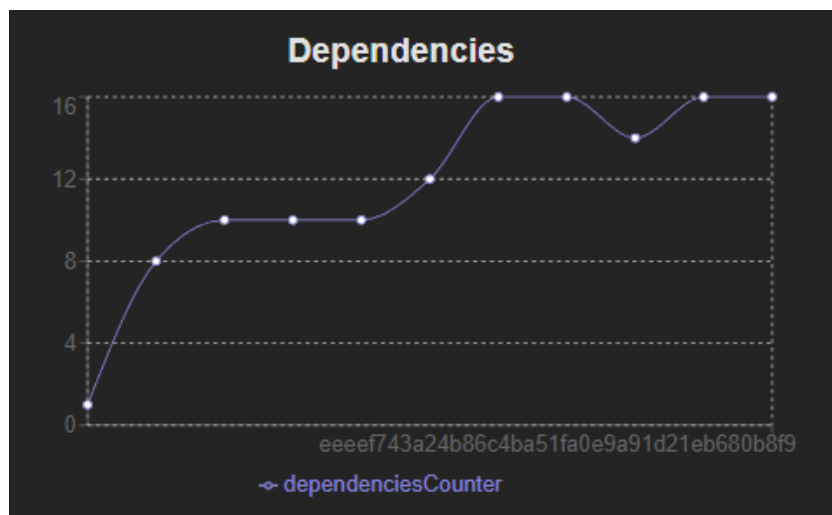


Figure 35 - TorchIO Dependencies

These numbers represent the number of direct dependencies for every particular project and its analysis over time. Considering this, we can make some observations.

The graph shows that the dependencies begin at about 8 and reach a maximum of 16 points. The fact that the number of dependencies fluctuates over time may indicate that the project is undergoing active development and that changes to its dependencies are being made. The highest number of dependencies may represent a period of expansion or the addition of new features to the project, whereas the lower numbers may indicate a period of stabilization or optimization. It's also worth noting that the specific dependencies themselves may be significant. For example, if the project is related to scientific research, which TorchIO does, the dependencies may include libraries or tools specific to those fields. Changes in the dependencies may reflect changes in the underlying research or the tools being used to analyze it.

In this case, developers should pay attention to the dependencies of the project and how they are managed over time. Keeping track of the dependencies and their versions is important to ensure that the project remains stable and secure. They should also monitor the frequency of updates to the dependencies and evaluate the impact of the changes on the project. It is essential to ensure that the updates do not break any functionality or introduce vulnerabilities. Additionally, developers should consider the support and activity level of the dependencies and evaluate the risks of using unmaintained or abandoned libraries.

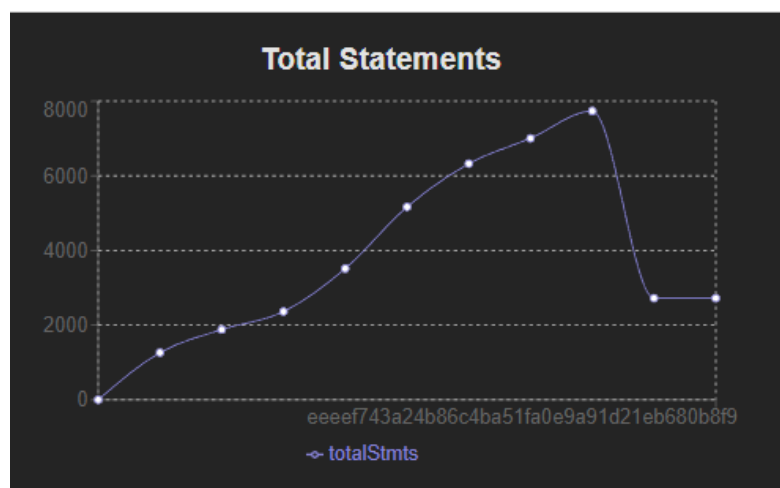


Figure 36 - TorchIO Total Statements

The graph shows that the dependencies begin at 1264, reach a maximum of 7737 points, and then reach 2722.

The increase in the number of lines of code over time can be interpreted as the growth and evolution of the project. As more features are added and the project becomes more complex, the number of lines of code tends to increase. However, an increase in the number of lines of code does not necessarily indicate improved functionality or efficiency. In fact, as the codebase becomes larger, it can become more difficult to maintain and debug, which can lead to increased development time and potential issues with code quality.

Therefore, it is important for developers to be mindful of the codebase size and take steps to manage complexity, such as refactoring and modularization. Regular code reviews and testing can also help ensure that the project is maintainable and performs optimally. In addition to managing code complexity, the increasing number of lines of code over time can also indicate the growth and evolution of the project's scope and functionality. This growth may reflect the development team's response to changing requirements, feature requests, or other external factors that affect the project's goals.

Nevertheless, it's crucial to remember that bigger codebases can also bring additional difficulties, such as possible security flaws or compatibility problems with external dependencies. The project schedule and budget may be affected if these problems require the use of additional resources, such as security audits or compatibility testing.

One could interpret the graph's increase and decrease in the number of statements as the result of an iterative development process. When new features and functionality are added to the project, the amount of code increases. As the development team has a better understanding of the project's needs, they will probably identify possibilities to rework and simplify the coding, which will lead to a decrease in the number of statements.

It's necessary to remember that fewer statements do not always translate into less functionality or efficiency. In fact, it can be a sign that the codebase's quality and maintainability have improved. It's possible that the development team found and removed pointless code, leading to a more streamlined and effective product.

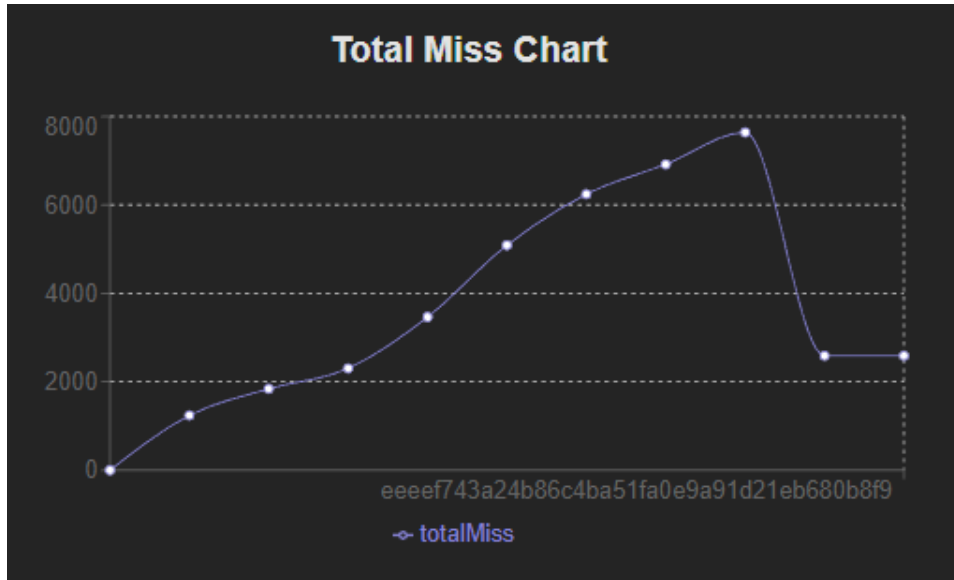


Figure 37 - TorchIO Total Miss

The trend of missed statements in the project shows that initially, the code had a lower number of missed statements, starting at 1239, indicating good code quality and a lower likelihood of bugs. However, as the project evolved and grew in complexity, the number of missed statements increased to 2588. This suggests that the code may have become more difficult to maintain and debug. It is important to note that a high number of missed statements does not necessarily mean that the project is in poor condition, as some missed statements may be harmless. However, a high number of missed statements can indicate a need for further testing and debugging to ensure code quality and maintainability.

The decrease in missed statements towards the end of the project suggests that steps were taken to improve code quality, such as refactoring, testing, or code reviews. By addressing the underlying issues that led to missed statements, the development team may have been able to improve the reliability and maintainability of the codebase.

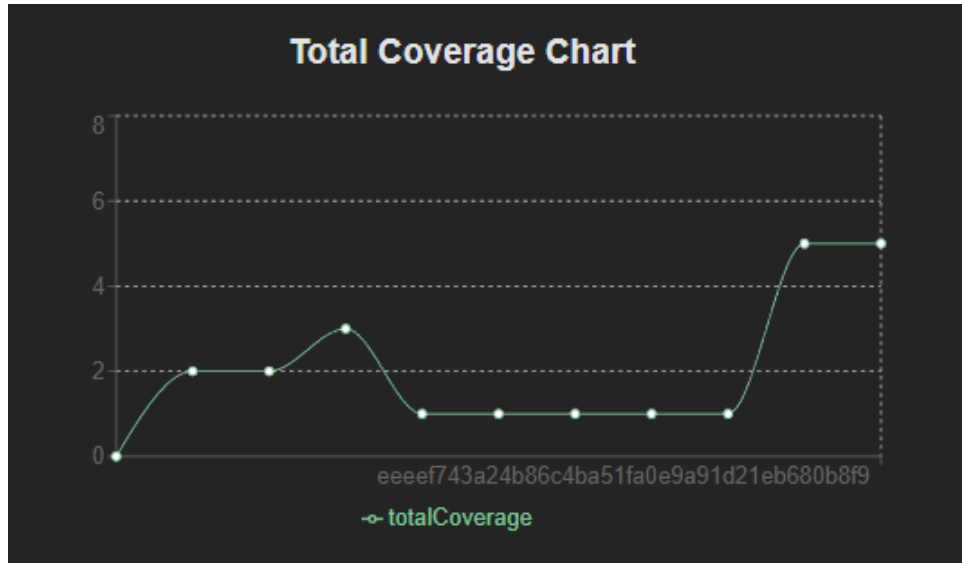


Figure 38 - TorchIO Total Coverage

A higher test coverage generally indicates a more robust and reliable codebase, as it means that more areas of the code have been thoroughly tested and any issues are more likely to be caught before they reach production.

In this case, the numbers indicate that the project has undergone some fluctuations in its test coverage. The initial coverage was relatively low at 2, indicating that only a small portion of the codebase was covered by tests. However, as the project evolved and more features were added, the coverage gradually increased to 5, indicating that more areas of the codebase were being tested.

It's worth noting that higher test coverage doesn't necessarily guarantee that the code is free of bugs or other issues, but it is an important metric for ensuring code quality and minimizing the likelihood of introducing new issues during development. Developers should strive to maintain a high level of test coverage throughout the project's lifecycle and continually update and improve their tests to ensure thorough coverage.

6.2 DPPy

Determinantal point processes (DPPs) are probabilistic models that encode diversity through a kernel function K , and they have a wide range of applications in

various fields such as machine learning, probability, and spatial statistics. In ML, finite DPPs are mostly used as models for diverse sets of items, such as recommendations or text summarization. However, routine inference tasks such as normalization, marginalization, or sampling can become computationally expensive, especially when M is large.

To address this issue, a turnkey Python implementation of known general algorithms to sample finite DPPs, called DPPy [39], has been proposed. In addition to algorithms for non-stationary continuous DPPs, DPPy also includes methods to sample finite DPPs, and it is hosted on GitHub and already being used by the cross-disciplinary DPP community. Through DPPy objects and associated methods, extensive documentation is provided that covers the essential mathematical background and showcases the key properties of DPPs, serving as a tutorial on DPPs. Continuous integration is done through Travis, and test coverage is done through Coveralls. Overall, DPPy aims to provide a more efficient and accessible implementation of DPPs for practitioners in various fields.

6.2.1 Results

In order to gain insights into the development process and evolution of the codebase, an analysis was performed on DPPy, which is, as explained earlier, a turnkey Python implementation of known general algorithms to sample finite DPPs. The project is owned by Guilgautier and was analyzed by recording the information of 10 commits that were present in the default branch over a period of time. The information gained from this analysis can be useful not only for the development team but also for researchers interested in building similar tools for machine learning applications. The results of the analysis are visualized in a comprehensive manner to provide a better understanding of the changes made to the project over time.

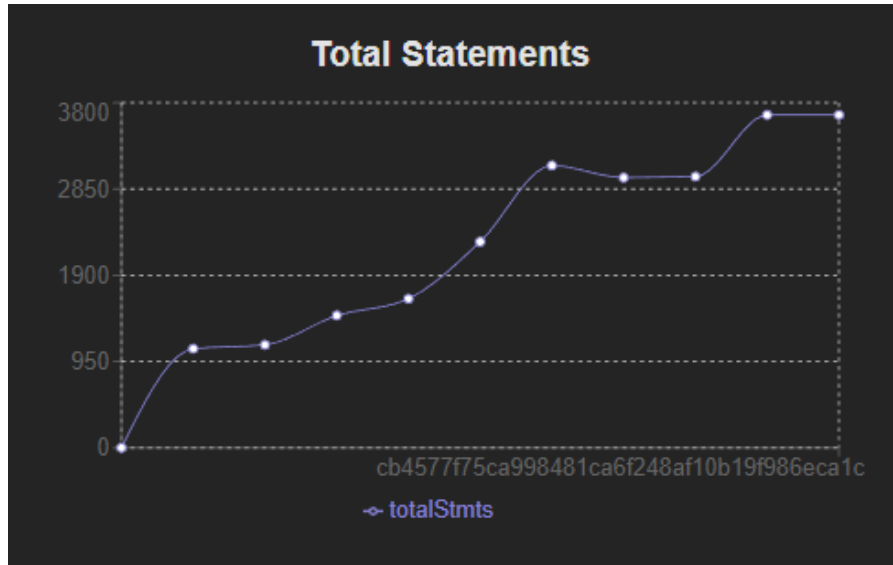


Figure 39 - DPPy Total Statements

This graph of numbers represents the number of code statements at different points in the development of the project. As the numbers increase over time, it suggests that the codebase is expanding and becoming more complex. In fact, as the codebase becomes larger, it can become more difficult to maintain and debug, which can lead to increased development time and potential issues with code quality.

Moreover, the number of code statements is just one aspect of a project's development and should be considered in conjunction with other factors such as functionality, maintainability, and efficiency. Developers should strive to write concise and efficient code while ensuring that it meets the project's requirements and goals. It's crucial to remember that fluctuations in the number of code statements may also be influenced by external factors such as changes in project scope or requirements. Therefore, developers should carefully consider the context of the project and aim for a balance between functionality and maintainability while keeping an eye on the codebase's size.

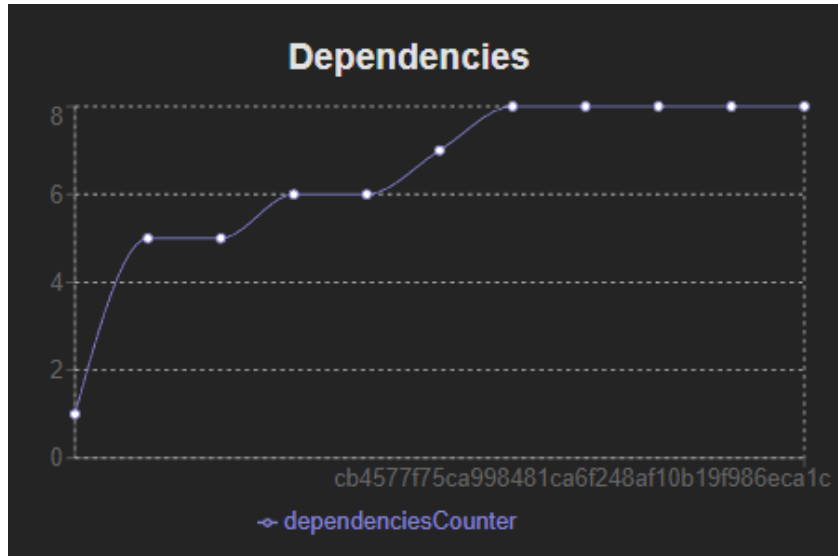


Figure 40 - DPPy Dependencies

This graph represents the number of dependencies in the DPPy project over time. As the numbers increase, it suggests that the project is expanding and relying more heavily on external code. While a high number of dependencies can indicate that the project is taking advantage of existing code and libraries, it may also imply code redundancy or inefficient implementation. Moreover, the number of dependencies can be influenced by external factors such as changes in project scope or requirements. It's common for projects to undergo periods of increasing and decreasing dependencies as developers add and remove features or update their code to utilize newer libraries or more efficient implementation strategies.

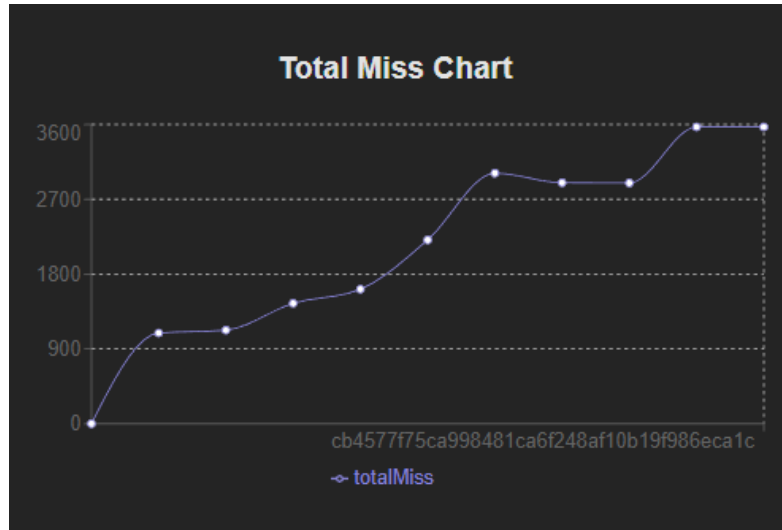


Figure 41 - DPPy Total Miss

The numbers provided represent the total missed statements in the project DPPy over time. The increasing trend of missed statements over time may suggest that the project is becoming more complex and difficult to test thoroughly. As the codebase grows, it can become more challenging to ensure that all code paths are covered by testing. This can lead to an increase in missed statements, which may indicate that bugs or issues could be present in the project. While an increase in missed statements can indicate potential issues, it could also be due to factors such as changes in project requirements or the addition of new features. Additionally, missed statements may not necessarily indicate a critical issue and could be harmless in some cases.

Therefore, developers should aim to minimize the number of missed statements in the project by writing clean and efficient code, implementing effective testing strategies, and regularly reviewing and refactoring the codebase. It's also essential to prioritize and triage the missed statements based on their impact on the project's functionality and stability. By doing so, developers can ensure that the project remains stable and reliable over time.

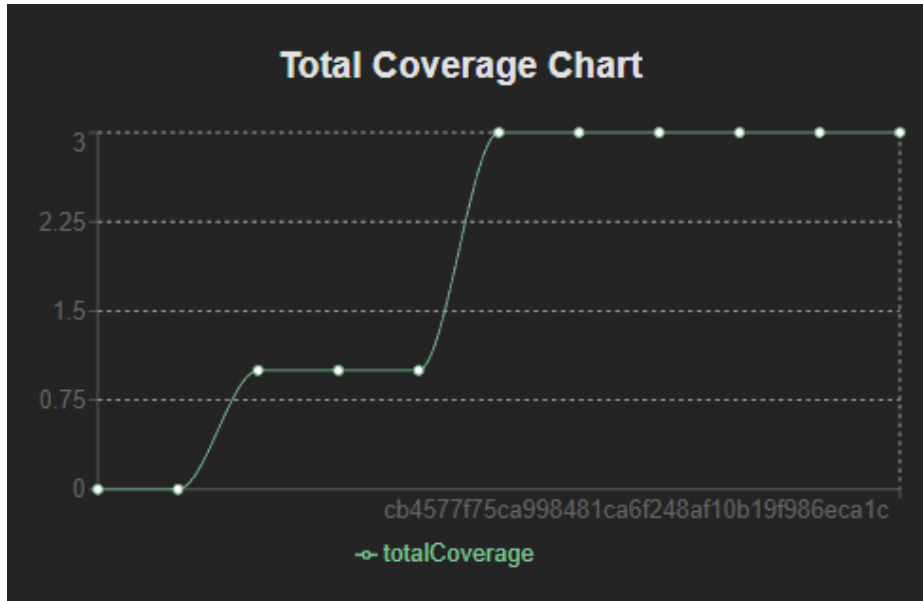


Figure 42 - DPPy Total Coverage

The graph shows that the test coverage starts at 0% and gradually increases to a maximum of 3%, with some fluctuations over time. The low percentage of test coverage in the early stages of the project may suggest that testing was not a priority or that testing frameworks were not yet implemented. The gradual increase in test coverage over time may indicate that the developers recognized the importance of testing and began to implement it more rigorously.

However, it's worth noting that a test coverage of 3% is still relatively low and may indicate that there are still significant areas of the codebase that are not covered by automated tests. This could potentially lead to issues with reliability and maintainability, as bugs or errors may go undetected until they are discovered by users. Therefore, it's important for developers to continue to prioritize testing and strive for higher levels of test coverage to ensure the quality and reliability of the project.

6.3 Results of random files

Two specific files, `data_parser.py` and `intensity_transform.py` from the project TorchIO, were examined in detail. The files were analyzed using various metrics, including statements, misses, coverage, Pylint ratings, comments, and similarity to other

files. The analysis provided insights into how the files changed over time and what factors influenced their evolution.

Table 2 - data_parser.py Results

Statements	99, 99, 99, 0, 0
Missed statements	99, 99, 99, 0, 0
Coverage	0, 0, 0, 0, 0
Ratings (Pylint)	7.84, 7.84, 7.84, 7.96, 7.96
Comments	5, 5, 5, 5, 5
Similarity (to random_motion.py)	11.66, 10.75, 10.32, 11.58

First, let's consider the data_parser.py file. The statements and missed statements metrics both show a consistent value of 99 for the first three measurements, followed by a sudden drop to 0 for the last two measurements. This suggests that the file underwent significant changes during the project, with a large number of statements and misses being removed. It is important to note that these two are the same, which means that no new statements were added to the file. This could indicate that the changes were focused on improving the quality of existing code rather than adding new features or functionality.

The coverage metric is consistently at 0 for the entire project, which means that the file is not being tested. This is a cause for concern, as it suggests that there is no way to know whether the changes made to the file have introduced new bugs or issues.

The ratings from Pylint show a consistent value of 7.84 for the first two measurements, followed by a slight increase to 7.96 for the last two measurements. This suggests that the code quality of the file is relatively high and consistent throughout the project.

The comments are stable, with a value of 5 for all measurements. This indicates that the technical debt associated with the comments in the file did not change significantly over the course of the project. Technical debt refers to the cost of

maintaining code that is not well- documented or is difficult to understand. The fact that the technical debt associated with comments remained stable suggests that the developers maintained consistent levels of documentation throughout the project.

Table 3 - intensity_transform.py Results

Statements	8, 17, 20, 20, 0, 0
Missed statements	8, 17, 20, 20, 0, 0
Coverage	0, 0, 0, 0, 0
Ratings (Pylint)	0, 7.7, 7.92, 7.94, 5.0
Comments	6, 6, 6, 6, 6
Similarity (to random_motion.py)	5.88, 11.65, 10.88, 10.51, 12.44

Now let's consider the intensity_transform.py file. The statements and missed statements both show a gradual increase in value over the first three measurements, followed by a sudden drop to 0 for the last two measurements. This suggests that the file underwent significant changes during the project, with a large number of statements and misses being removed. It is important to note that both of them are also the same, which means that no new statements were added to the file.

The coverage metric is consistently at 0 for the entire project, which means that the file is not being tested.

The ratings from Pylint show a gradual increase in value from 0 to 7.94 over the course of the project, with a slight dip to 5.0 for the last measurement. This suggests that the code quality of the file improved over time, but there may have been some issues introduced towards the end of the project.

The comments are stable, with a value of 6 for all measurements. This indicates that the technical debt associated with the comments in the file also did not change significantly over the course of the project.

Finally, it is worth noting the similarity with another file called `random_morion.py` for both files. This metric shows a gradual increase in value over the first three measurements, followed by a slight decrease for the last measurement. This suggests that the two files became more similar over time, with changes made to one file being reflected in the other file. However, it is important to note that the values for this metric are relatively low, which means that the two files are not very similar to each other.

These values suggest that both files have some level of similarity with each other, although the values are relatively low. The gradual increase in similarity for `data_parser.py` and the fluctuations in similarity for `intensity_transform.py` suggest that changes made to the other file may have influenced these two files to some extent.

7 Conclusion

In summary, this thesis focuses on the development of a tool for evaluating the quality of Python code in GitHub projects across various research fields. By automating the evaluation process, the tool aims to identify areas for improvement in software quality, such as readability, maintainability, and efficiency. The quality of software is critical for meeting its intended purpose and facilitating future modifications, and the proposed tool has the potential to aid in improving software quality in the development process.

7.1 Summary and conclusions

After conducting a comprehensive analysis of Python code in GitHub projects, the proposed tool has proven to be effective in evaluating code quality across various research fields. The tool's features, presentation, and functionality were found to be suitable for automating the evaluation process, which can aid in identifying areas for improvement in software quality. Through the evaluation of the selected projects, it was found that code quality can vary significantly, and some projects may require more attention to improve their overall quality. Additionally, the tool's ability to provide detailed metrics and visualizations can aid in identifying specific areas of improvement and prioritizing efforts in those areas. Overall, the tool's contribution to the field of software development is significant, as it provides a reliable and efficient method for evaluating code quality and improving the overall quality of software systems.

7.2 Suggestions for future research

There are several opportunities for extending the work presented in this thesis. One potential area for further research is the expansion of the tool to support additional programming languages beyond Python. This would enable a broader range of projects to be analyzed and provide a more comprehensive understanding of code quality across multiple languages.

Another potential avenue for future research is the integration of machine learning techniques into the tool. This could include the use of natural language

processing to identify comments and documentation within the code, as well as the use of machine learning algorithms to automatically identify code smells and other quality issues. Furthermore, there may be other areas of code quality that the tool does not currently address, such as security vulnerabilities or performance issues. Future research could explore methods for identifying and addressing these types of quality concerns.

Lastly, exploring ways to automate the identification and resolution of quality issues in real-time would be a valuable contribution to the field. This could involve developing algorithms that automatically suggest fixes for identified quality issues or integrating the tool into a continuous integration and delivery pipeline for automatic code review and improvement.

Bibliography

- [1] A. J. Dhruv, R. Patel, and N. Doshi, “Python: The Most Advanced Programming Language for Computer Science Applications;,” *Proc. Int. Conf. Cult. Herit. Educ. Sustain. Tour. Innov. Technol.*, pp. 292–299, 2020, doi: 10.5220/0010307902920299.
- [2] R. Nitnaware, “Basic Fundamentals of Python Programming Language and The Bright Future,” *Peer-Rev. J. About*, vol. VIII, pp. 71–76, Jun. 2019.
- [3] “PyPI · The Python Package Index,” *PyPI*. <https://pypi.org/> (accessed Apr. 26, 2023).
- [4] A. Sharma, F. Khan, D. Sharma, and S. Gupta, “Python: The Programming Language of Future,” *International Journal of Innovative Research in Technology*, vol. 6, May 2020.
- [5] “What is Difficult in Learning Programming Language Based on Problem-Solving Skills? :: Duhok Polytechnic University.” <https://www.dpu.edu.krd/page/en/2227/> (accessed Apr. 24, 2023).
- [6] J. Darkstar, “Mark Lutz Learning Python 5th Edition”, Accessed: Apr. 24, 2023. [Online]. Available: https://www.academia.edu/34415693/Mark_Lutz_Learning_Python_5th_Edition
- [7] S. L. Pfleeger, R. Jeffery, B. Curtis, and B. Kitchenham, “Status report on software measurement,” *Softw. IEEE*, vol. 14, pp. 33–43, Apr. 1997, doi: 10.1109/52.582973.
- [8] S. Magill, “What is Code Quality? 5 Software Development Checks You Should Be Automating.” <https://blog.sonatype.com/improving-code-quality-with-automation> (accessed Apr. 22, 2023).
- [9] Robert C. Marti, *Clean Code: A Handbook of Agile Software Craftsmanship*. 2008.
- [10] “ISO 25010.” <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010> (accessed Apr. 24, 2023).
- [11] M. Samaranayake, “Code Quality Automation: What’s Around the Corner?,” *Medium*, Mar. 13, 2022. <https://blog.bitsrc.io/code-quality-automation-whats-around-the-corner-16f8eda5a43a> (accessed Apr. 22, 2023).
- [12] “12 BEST Code Quality Tools For Error Free Coding In 2023,” *Software Testing Help*, Mar. 16, 2023. <https://www.softwaretestinghelp.com/code-quality-tools/> (accessed Apr. 22, 2023).
- [13] “SonarQube 10.0.” <https://docs.sonarqube.org/latest/> (accessed May 02, 2023).

- [14] Atlassian, “Crucible Code Review Tool for Git, SVN, Perforce and More,” *Atlassian*. <https://www.atlassian.com/software/crucible> (accessed May 02, 2023).
- [15] “Find and fix problems in your JavaScript code - ESLint - Pluggable JavaScript Linter,” Apr. 21, 2023. <https://eslint.org/> (accessed May 02, 2023).
- [16] “JSHint Documentation.” <https://jshint.com/docs/> (accessed May 02, 2023).
- [17] R. Python, “Python Code Quality: Tools & Best Practices – Real Python.” <https://realpython.com/python-code-quality/> (accessed Apr. 22, 2023).
- [18] “Flake8: Your Tool For Style Guide Enforcement — flake8 6.0.0 documentation.” <https://flake8.pycqa.org/en/latest/> (accessed Apr. 24, 2023).
- [19] K. Klenov, “pylama: pylama -- Code audit tool for python.” Accessed: Apr. 24, 2023. [Online]. Available: <http://github.com/klen/pylama>
- [20] “unittest — Unit testing framework,” *Python documentation*. <https://docs.python.org/3/library/unittest.html> (accessed Apr. 24, 2023).
- [21] “pytest: helps you write better programs — pytest documentation.” <https://docs.pytest.org/en/7.3.x/> (accessed Apr. 24, 2023).
- [22] R. Cox, “Surviving Software Dependencies.” <https://cacm.acm.org/magazines/2019/9/238968-surviving-software-dependencies/fulltext> (accessed Apr. 24, 2023).
- [23] I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci, “Vulnerable open source dependencies: counting those that matter,” in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, in ESEM ’18. New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 1–10. doi: 10.1145/3239235.3268920.
- [24] V. Kravchenko, “pipreqs: Pip requirements.txt generator based on imports in project.” Accessed: Apr. 24, 2023. [Online]. Available: <https://github.com/bndr/pipreqs>
- [25] G. Gharibi, R. Tripathi, and Y. Lee, “Code2graph: automatic generation of static call graphs for Python source code,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, in ASE ’18. New York, NY, USA: Association for Computing Machinery, Sep. 2018, pp. 880–883. doi: 10.1145/3238147.3240484.
- [26] “PyTA.” PyTA, Apr. 03, 2023. Accessed: Apr. 24, 2023. [Online]. Available: <https://github.com/pyta-uoft/pyta>
- [27] L. Li, J. Wang, and H. Quan, “Scalpel: The Python Static Analysis Framework.” arXiv, Feb. 23, 2022. doi: 10.48550/arXiv.2202.11840.
- [28] “Java (programming language),” *Wikipedia*. Apr. 24, 2023. Accessed: Apr. 24, 2023. [Online]. Available:

[https://en.wikipedia.org/w/index.php?title=Java_\(programming_language\)&oldid=1151443812](https://en.wikipedia.org/w/index.php?title=Java_(programming_language)&oldid=1151443812)

- [29] “Professional Java Development with the Spring Framework | Wiley,” *Wiley.com*. <https://www.wiley.com/en-sg/Professional+Java+Development+with+the+Spring+Framework-p-9780764574832> (accessed Apr. 22, 2023).
- [30] P. G. D. Group, “PostgreSQL,” *PostgreSQL*, Apr. 24, 2023. <https://www.postgresql.org/> (accessed Apr. 24, 2023).
- [31] “JavaScript With Syntax For Types.” <https://www.typescriptlang.org/> (accessed Apr. 24, 2023).
- [32] G. Bierman, M. Abadi, and M. Torgersen, “Understanding TypeScript,” in *ECOOP 2014 – Object-Oriented Programming*, R. Jones, Ed., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2014, pp. 257–281. doi: 10.1007/978-3-662-44202-9_11.
- [33] “React.” <https://react.dev/> (accessed Apr. 24, 2023).
- [34] “Unify the DevSecOps lifecycle with GitLab.” <https://about.gitlab.com/stages-devops-lifecycle> (accessed Apr. 24, 2023).
- [35] P. C. Q. Authority, “pylint: python code static checker.”
- [36] M. Schlaich, “pytest-cov: Pytest plugin for measuring coverage.” Accessed: Apr. 24, 2023. [Microsoft :: Windows, POSIX, Unix]. Available: <https://github.com/pytest-dev/pytest-cov>
- [37] D. Platis, “Duplicate Code Detection Tool.” Apr. 19, 2023. Accessed: Apr. 24, 2023. [Online]. Available: <https://github.com/platisd/duplicate-code-detection-tool>
- [38] F. Pérez-García, R. Sparks, and S. Ourselin, “TorchIO: A Python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning,” *Comput. Methods Programs Biomed.*, vol. 208, p. 106236, Sep. 2021, doi: 10.1016/j.cmpb.2021.106236.
- [39] G. Gautier, G. Polito, R. Bardenet, and M. Valko, “DPPy: DPP Sampling with Python,” *J. Mach. Learn. Res.*, vol. 20, no. 180, pp. 1–7, 2019.