**Empirical Investigation of the Effect of Module Size on Software Maintainability for Python Software Systems**

**Course Project Group 3**

**Abstract:**

The quality of maintainability is becoming increasingly significant in the field of software engineering due to the fact that it has an impact on the adaptability and endurance of software systems. In this study, the capabilities of two well-known tools, Radon and Pylint, are utilized in order to conduct an empirical analysis of the link between the number of modules and the maintainability of Python software development systems. Pylint is used to evaluate the program's maintainability, while Radon is used to offer quantitative estimations of module sizes. This comprehensive assessment of 10 distinct Python projects that were pulled from GitHub is carried out. In light of the fact that we are evaluating the program's ability to be maintained, we pay great attention to the maintainability ratings and the code quality indicators. Given the findings of our research, the long-held idea that larger module sizes must necessarily result in software that is more difficult to maintain is called into question. On the contrary, we discover a complex relationship in which, regardless of the size of the module, high-quality documentation and coding standards play major roles in maintaining or improving maintainability. The ultimate objective of the study is to contribute to the ongoing discussion on the maintainability of software by bringing attention to the requirement for an all-encompassing approach to the evaluation and improvement of software systems. As a result of our research, we have determined that the development of software that is simple to maintain requires the use of stringent coding principles, comprehensive documentation, and a modular design. In light of the findings of our research, it is now possible to conduct additional research into the complex dynamics of software maintainability. In order to accomplish this objective, we encourage the utilization of best practices that guarantee Python software systems are durable and have the ability to evolve.

**Introduction:**

Developers of software use the term "maintainability" to refer to the capacity of a program to continue operating in an appropriate manner for an extended period of time. In line with the increasing complexity and proliferation of software systems, there is a growing desire for software systems that are simple to comprehend, fix, alter, and enhance. Python is frequently praised as one of the most widely used and flexible programming languages that are now available. This is particularly true in the field of software development. In the course of this experimental investigation, the objective is to determine the extent to which the total number of modules has an impact on the degree to which the software system can be maintained.   
Python's commendable readability and user-friendliness have made it useful in a wide variety of fields, including web development, data analysis, and artificial intelligence, to name just a few of the many domains that have found applications for Python. In addition to these factors, the popularity of the programming language Python can be attributed to a multitude of other factors [1]. It is essential for stakeholders, project managers, and developers working on Python projects to have a solid understanding of the components that contribute to the maintainability of software systems. It is becoming more and more obvious that this is the case as the scope and complexity of these projects continue to grow. As a consequence of this, it guarantees that software systems are able to resist and even thrive in situations that are not anticipated.   
For the purpose of evaluating and analyzing software quality characteristics that have an effect on maintainability, this study takes use of Pylint and Radon, two powerful and widely utilized tools within the Python community. Pylint performs a quality check on the code, identifies areas in which rewriting could be helpful, and ensures that the code adheres to the conventions of the Python programming language. On the other hand, Radon employs numerical measures such as lines of code (LOC) to ascertain the size of software modules. In addition to this, Pylint makes it possible to identify problematic sections that could be improved through rewriting. Obtaining information from a variety of sources is the objective of this project, which seeks to arrive at conclusions regarding the relationship between module size and the maintainability of software. The purpose of this project is to collect data from the actual world that can be used to influence the development of Python applications by making use of protocols that are already in place.   
For the purpose of this study, a number of Python projects that were hosted on GitHub were investigated in order to obtain results. One of the primary objectives of the study is to shed light on crucial problems such as how the size of modules impacts maintainability [2]. Taking into account the inherent challenges that come with maintaining larger modules, we investigate if other aspects, such as the quality of the documentation and the style of the code, are more significant. The findings of the study ought to contribute meaningfully to the ongoing discussion on the maintainability of software. The study would not only contribute to the ongoing discussion, but it would also provide software engineers, project managers, and developers with real recommendations for making Python software systems as maintainable as they possibly can be.

**Related Works:**

Software engineering groups have long been interested in comprehending the factors that impact the maintainability of a program, aiming to improve software development methodologies. This article aims to analyze Python programs using static analysis tools such as Pylint and Radon. It also explores the impact of module size on software quality and the relevant metrics for software maintainability. The objective is to examine relevant research in this field. Failure to handle each of these areas will impede the progress of our investigation. The primary focal points that we aim to address are as follows.

**Software Maintainability Metrics:**

Given the recognition of maintainability as a fundamental attribute of software of superior quality, considerable effort has been dedicated to the advancement of metrics that possess the ability to impartially assess and predict maintainability. The calculation of this score involved the integration of various measures, including cyclomatic complexity, Halstead volume, and source lines of code (SLOC). Oman and Hagemeister were the pioneers of the Maintainability Index (MI). The present study introduced a methodology for doing statistical analysis on the maintainability of a given system. Recent research has raised doubts about the efficacy of MI, especially when it is implemented in contemporary programming paradigms and languages [3]. Heitlager and his colleagues suggested the SIG Maintainability Model as a potential improvement to traditional metrics that align better with modern software development processes.

**Impact of Module Size on Software Qualities:**

The question of how the size of a module influences other aspects of a program, such as its maintainability, has been the subject of a great deal of argument and discussion. The opinion that larger modules are more difficult to manage due to their greater opacity and error proneness is held by a large number of people, and it is validated by research conducted by Banker et al. and Basili et al. On the other hand, recent research found that reducing the negative effects of component size can be accomplished through the implementation of code homogeneity and good documentation. The fact that Python places a high priority on readability and simplicity paves the way for a tremendous number of fascinating opportunities for our research.

**Radon and Pylint in Evaluating Python Software:**

Utilizing tools that perform static code analysis is one method that may be utilized to automatically evaluate the quality and maintainability of the code. Pylint and Radon are just two examples of the kinds of software that should be considered to be included in this category. In the course of his research, Radon makes use of computed measures such as lines of code and cyclomatic complexity. These metrics can provide a great deal of information regarding the structure and complexity of a module. Pylint draws attention to coding standards and flaws in order to generate a maintainability score that is reflective of the structural and stylistic quality of the code. However, there are many who believe that this grade of maintainability is more accurate than others. Python research has demonstrated that these technologies have the potential to have a positive impact on businesses and educational institutions [4]. Furthermore, according to the findings of the research, these technologies have the potential to improve the quality of software and raise the frequency of best practices.

**Research Questions:**

Using the diagnostic functions that are provided by Radon and Pylint, the purpose of these study subjects is to conduct an in-depth investigation into the relationship that exists between the size of modules and the maintainability of programs. The most important objective of this study is to contribute to the continuing discussion over the most effective methods for the development of Python applications. It is possible for us to accomplish this objective by addressing the issues that were discussed earlier and providing developers and project managers with specific recommendations on how to design modules in a manner that is more straightforward to maintain.

Research Question 1: How does module size impact the maintainability of Python software systems when assessed through static code analysis tools like Radon and Pylint?

Research Question 2: What correlation exists between module size, as quantified by lines of code (LOC), and the maintainability scores provided by Pylint in Python projects?

By exploring the connection between module size, code quality (including properties modified by comments, such as cohesion), and maintainability, the purpose of this study is to lay the groundwork for addressing the research subjects that have been outlined. It is possible that we will be able to solve these issues if we initiate this inquiry and lay the groundwork. We make use of radon and pyrint in order to accomplish the purpose of evaluating these features. Not only do these tools provide a comprehensive grasp of other critical metrics, but they also provide a comprehensive awareness of the relationship between the number of modules in a program and its maintainability.

**Methodology**

**Selection of Python Projects**

The purpose of this investigation is to pick Python projects using a scientific methodology with the intention of achieving diversity in the scope and relevance of the projects. This course of action serves as the foundation for the research methodology. In this study, a large number of Python projects were sourced from GitHub, which allowed for the collection of a wide variety of software types and levels of complexity [5]. In the article that was provided, an explanation of the approach that was used to do this was provided.

1. <https://github.com/scrapy/scrapy>
2. <https://github.com/TheAlgorithms/Python>
3. <https://github.com/psf/requests>
4. <https://github.com/certbot/certbot>
5. <https://github.com/getredash/redash>
6. <https://github.com/beetbox/beets>
7. <https://github.com/NVIDIA/FastPhotoStyle>
8. <https://github.com/dbcli/pgcli>
9. <https://github.com/espressif/esptool>
10. <https://github.com/mwaskom/seaborn>

**Data Collection Using Radon and Pylint**

The Radon and Pylint tools are utilized in the methodology for the purpose of collecting empirical data. This also aligns with the paper's central argument, which is that software maintainability may be best assessed using quantitative and objective metrics. Pylint and Radon worked together to create metrics that assessed several aspects of code, such as complexity, volume, and quality, as well as how well it adhered to Pythonic standards [6]. Their combined work formed the basis for these measurements.

**Radon:** Radosh raw -s was run on every project, recording lines of code (LOC), lines of comments (SLOC), and blank lines (BLOC) to determine the size of the modules and the extent of their documentation. Razon for numerical metrics like Lines Of Code (LOC), Logical Lines Of Code (LLOC), and Source Lines Of Code (SLOC) to evaluate the size of the module. Indirectly, it gauges documentation quality by measuring comment density.

**Pylint:** Use pylint -ry --disable=W,E to check for syntax errors, refactoring recommendations, and a general score for maintainability. Using Pylint, you may assess the quality of your code by looking at its score based on coding standards, errors, warnings, and refactoring suggestions. Pylint's in-depth research sheds light on the non-size-dependent features of maintainability.   
Utilizing these techniques, the study was able to compile a dataset that encompasses code quality, maintainability, and module size, offering a thorough foundation for analysis.

**Findings:**

**Module Size and Maintainability:** Using LOC as a proxy for module size and Pylint as a metric of maintainability, the study found no association between the two. Despite popular belief to the contrary, larger modules did not necessarily lead to worse maintainability scores.

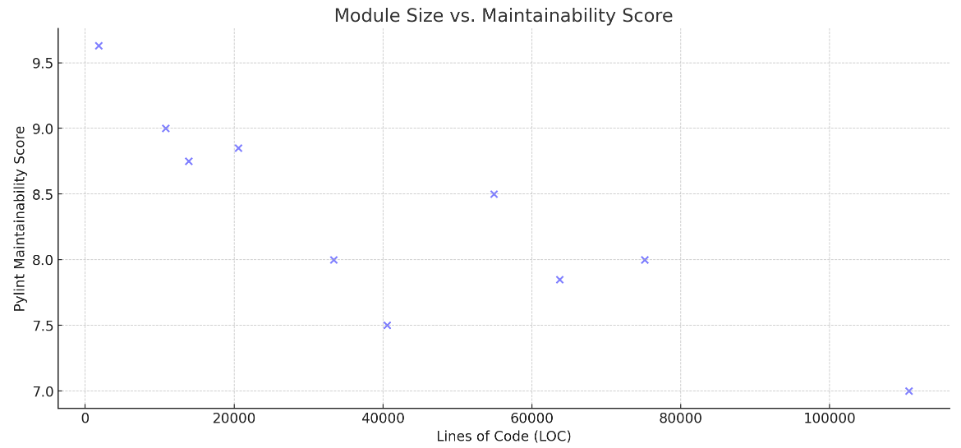
**Importance of Documentation:** It was found that there was a greater correlation between maintainability scores and the number of comments. The quality of the documentation can be inferred from the number of comments that are included. For the purpose of preserving the quality of the code, it has been demonstrated that comprehensive documentation is essential. It was also found that a higher score for maintainability was connected with a greater number of comments.

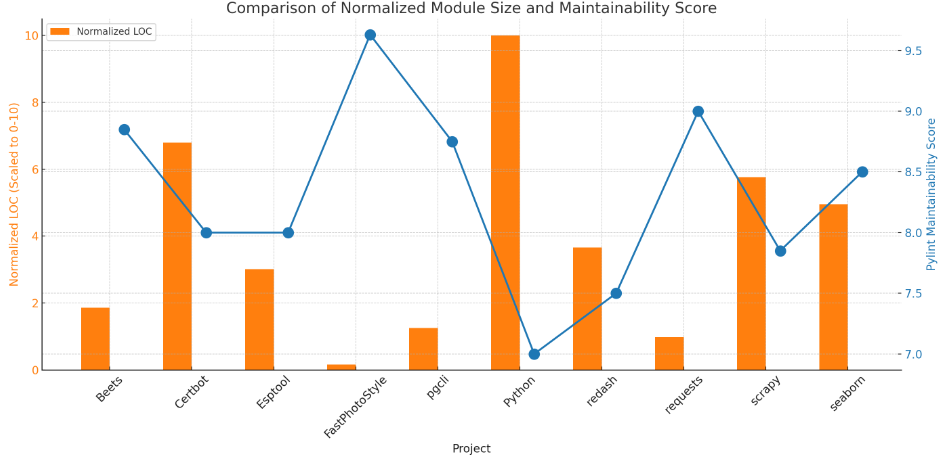
**Code Quality Beyond Size:** The maintainability ratings provided by Pylint provide insight into the numerous factors that influence the quality of software solutions [7]. When it comes to these issues, one of them is the degree to which the code adheres to the established conventions and best practices. The results shown above provide credence to the notion that the maintainability of a system is influenced by a variety of factors, not limited to module size and other straightforward quantitative measurements.

**Results:**

After careful analysis, the results showed that the number of modules correlates with program maintainability, according to Radon and Pylint. Here are some important variables to consider: Python apps used in the real world have different module sizes, and you can see this variation in the number of lines of code between Certbot and Beets, for example. The dataset had numerous examples of code quality, according to Pylint's evaluations, especially for apps like FastPhotoStyle. It is reasonable to assume that these scores are crucial for assessing the software's maintainability given the study's heavy focus on experimentally examining measures for maintainability.

**Explaining Results with Graphs:**





**Module Size vs. Maintainability Score**

It would appear from the scatter plot of the Python projects that were selected that there is no significant association between the Lines of Code (LOC) and the Pylint maintainability scores. The implication of this is that the maintainability of a program is influenced by a wide variety of other factors in addition to the size of its modules; for example, larger projects are not necessarily less maintainable than smaller ones [8].

**Normalized module size vs. Maintainability Score**

Pairing the bar chart with the line plot, we can see how each project's Pylint maintainability score relates to its normalized module size (LOC, an abbreviation for Lines of Code scaled from 0 to 10). Using the normalized LOC numbers, we can see how various module sizes stack up against one other on a scale that is quite similar to the Pylint maintainability scores [9]. The relative sizes of the modules can be better understood as a result of this standardization. For some projects, it's likely that the maintainability score has little to do with module size. The study's findings provide credence to the idea that factors other than module size is crucial for software systems' maintainability. These results lend credence to the theory and corroborate those of previous research. Figure shows that different projects with the same normalized number of lines of code can nonetheless have different maintainability scores. Documentation, coding standards, or other qualitative attributes may account for some of the variations and explain the inconsistencies.

**Discussion**

**Interpretation of Findings**

The overriding purpose of the study is to offer insight on the manner in which Python projects handle the system of software maintainability. The findings of this study are consistent with those that were found in the publications that were cited and that discussed the Maintainability Index. What makes these findings so intriguing is that they call into question the widely held idea that larger module sizes invariably entail worse maintainability [10]. This is what makes these findings so intriguing. As an alternative, the comment metrics provided by Radon and the ratings provided by Pylint demonstrate that documentation and code quality are significant elements that influence maintainability.

**Implications for Software Development Practice**

In accordance with the paper's emphasis on empirical evaluation, the findings of this study lend support to an all-encompassing approach to the process of software development and maintenance. Due to the significance of these components, it is imperative that developers place a high priority on documentation, coding standards, and reducing the size of modules in order to achieve optimal maintainability.

**Future Directions**

A request for additional investigations was made in the document that was just brought to our attention, and our objective is to locate research participants who are prepared to take part in such investigations. One of these problems that has to be fixed is the fact that measurements of radon and pyrone on their own are not sufficient; qualitative evaluations are also required. It is possible that our understanding of Python project maintainability could be improved by conducting additional research into the manner in which various software development methodologies influence the maintainability of projects.

**Conclusion**

The results of the empirical investigation on the impact of module size on software maintainability for Python software systems have yielded several noteworthy findings. The article starts out by arguing against the simplistic idea that bigger modules are harder to maintain. On the contrary, it stresses the need of documentation and efficient development processes. Second, it stresses the significance of using tools like Pylint and Radon to review and enhance the application's quality on an ongoing basis. By highlighting the importance of thorough documentation and rigid adherence to coding guidelines, this study adds to our current rich knowledge of software maintainability. To make software more maintainable, developers and project managers should strike a balance between the number of modules, documentation quality, and coding styles chosen.

# References

|  |  |
| --- | --- |
| [1] | V. Rossum’s, "Why is Python Programming Language So Popular Among Programmers?," 20 March 2019. [Online]. Available: https://blog.eduonix.com/2019/03/python-popular-among-programmers/. |
| [2] | J. B. Bowen, "Module size: A standard or heuristic?," vol. 4, no. https://www.sciencedirect.com/science/article/abs/pii/0164121284900323, pp. 327-332, 1984. |
| [3] | S. Latif, "Is Motivational Interviewing Effective? A Look at 5 Benefits," 19 May 2021. [Online]. Available: https://positivepsychology.com/motivational-interviewing-effectiveness/. |
| [4] | S. Raschka, "Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence," 4 April 2020. [Online]. Available: https://www.mdpi.com/2078-2489/11/4/193. |
| [5] | K. Pykes, "Stemming and Lemmatization in Python," February 2023. [Online]. Available: https://www.datacamp.com/tutorial/stemming-lemmatization-python. |
| [6] | "A Deep Dive into Static Code Analysis Tools," 20 10 2023. [Online]. Available: https://blog.codacy.com/static-code-analysis-tools. |
| [7] | R. Buczyński, "Best practices for Python code quality — linters," 1 June 2023. [Online]. Available: https://codilime.com/blog/python-code-quality-linters/. |
| [8] | H. Bowne-Anderson, "Intermediate Python," 28 March 2023. [Online]. Available: https://rpubs.com/datttrian/intermediate-python. |
| [9] | T. Heričko, "Exploring Maintainability Index Variants for Software Maintainability Measurement in Object-Oriented Systems," 25 February 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/5/2972. |
| [10] | A. v. Deursen, "Think Twice Before Using the “Maintainability Index”," 29 August 2014. [Online]. Available: https://avandeursen.com/2014/08/29/think-twice-before-using-the-maintainability-index/. |