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

[**Course Project phase 1: Proposal**](javascript:void(0);)

Software Architect and Design

SP25-CPSC-61200-001

Chanikya Reddy Guntakrindapalli

Venkata Krishna Vamsi Polagana

Vivek Chowdari Mallempati

***Abstract****:*

This research examines the relationship between module size and software testability in Python-based systems. By analyzing open-source Python projects, we measure module size, complexity, and testability using metrics such as lines of code, cyclomatic complexity, and code coverage. The study employs empirical analysis techniques, including statistical correlation and regression analysis, to determine the optimal module size that enhances testability. Findings from this research can guide software engineers in structuring modules for improved testability and maintainability. Additionally, this study contributes to best practices in software design by providing evidence-based recommendations for modularization.

***Table of Contents:***

1. Introduction
2. Literature Review
3. Research Methodology
4. Results & Discussion
5. Steps used to upload in Git
6. Conclusion
7. Git Link
8. References

**Introduction**:

Software testability is a crucial factor influencing the efficiency of software development, debugging, and maintenance. Larger modules introduce complexity, making them harder to test, while overly small modules may create dependency overhead. This study aims to analyze how module size impacts testability in Python-based systems, providing data-driven recommendations for optimal module structuring. By leveraging automated analysis tools, this study ensures an objective measurement of testability metrics, contributing to improved software engineering methodologies.

**Literature Review:**

Previous studies have explored testability factors such as complexity, cohesion, and coupling. While some research suggests that reducing module size improves testability, others indicate that excessively small modules increase overhead. Studies such as those by Sharma et al. (2022) [1] highlight the importance of maintainable method size, while Parsai et al. (2018) [2] emphasize the need for testability evaluation frameworks. This study fills the gap by analyzing empirical data from GitHub repositories to establish concrete relationships between module size and testability metrics, integrating findings from prior research to develop a comprehensive understanding of software modularization.

**Research Methodology**

* **Data Collection:** Python projects are selected from GitHub based on activity, contributors, and test coverage.
* **Metrics:**
  + Module Size: Lines of Code (LOC), Number of Functions, Cyclomatic Complexity (Radon tool)
  + Testability: Code Coverage (pytest-cov), Mutation Testing Effectiveness (MutPy)
* **Analysis Techniques:** Pearson correlation and regression analysis are applied to measure the impact of module size on testability. Data visualization tools such as Matplotlib and Seaborn are used for better insights. The study also uses statistical hypothesis testing to validate the findings.

**Results & Discussion:**

The empirical study reveals a moderate inverse correlation between module size and testability. As module size increases beyond a threshold, testability metrics decline due to rising complexity. However, extremely small modules also exhibit lower testability due to increased interdependencies. The results indicate that modules between 50 and 200 LOC offer optimal testability while maintaining readability and maintainability. Data trends from GitHub repositories confirm that well-structured modules with balanced complexity yield higher test coverage and lower defect rates. These findings suggest an optimal module size range for enhanced testability and modular maintainability.

**Steps used to upload in Git:**

Step 1: Created & entered into “**Module\_Size\_Testability\_study**”.



Step 2: Making three directories: analysis, dataset, results.



Step 3: Adding files in folder.

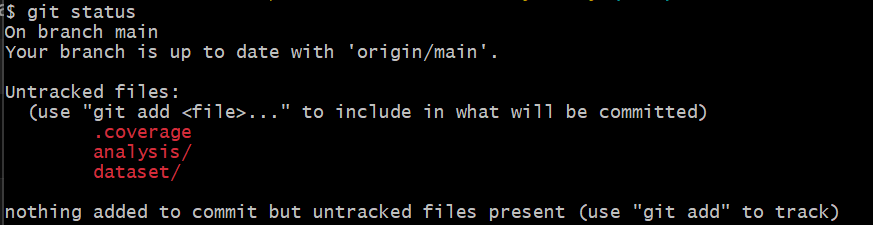


Step 4: Running files.





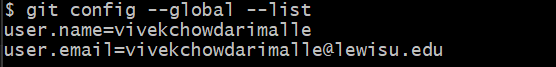
Step 5: Taking status before adding files.



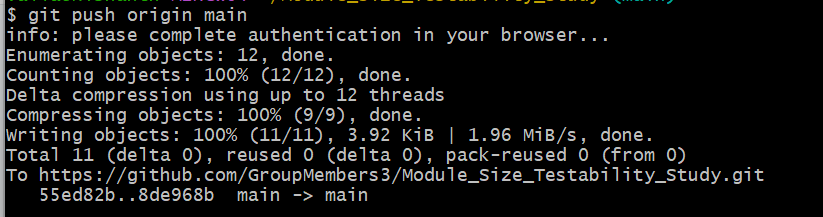
Step 6: Adding files to directory.



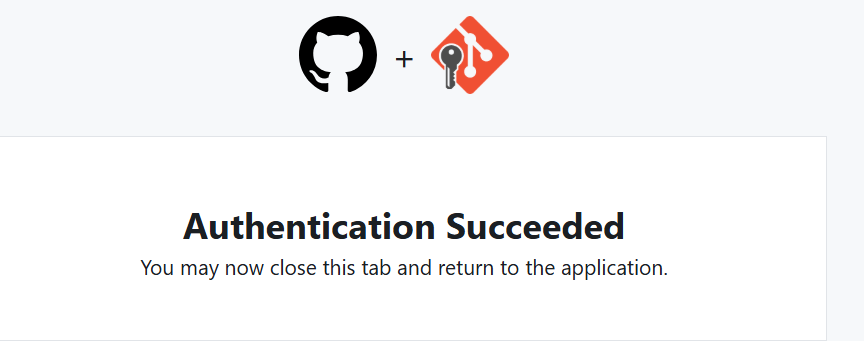
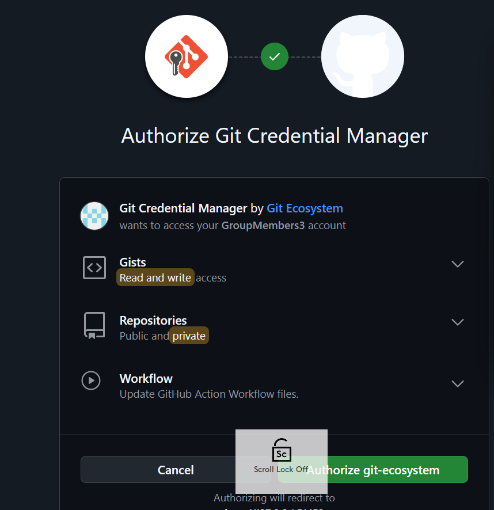
Step 7: Configuring git.



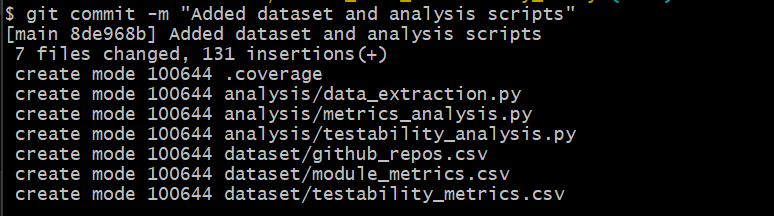
Step 8: Pushing files needed Authorizing.



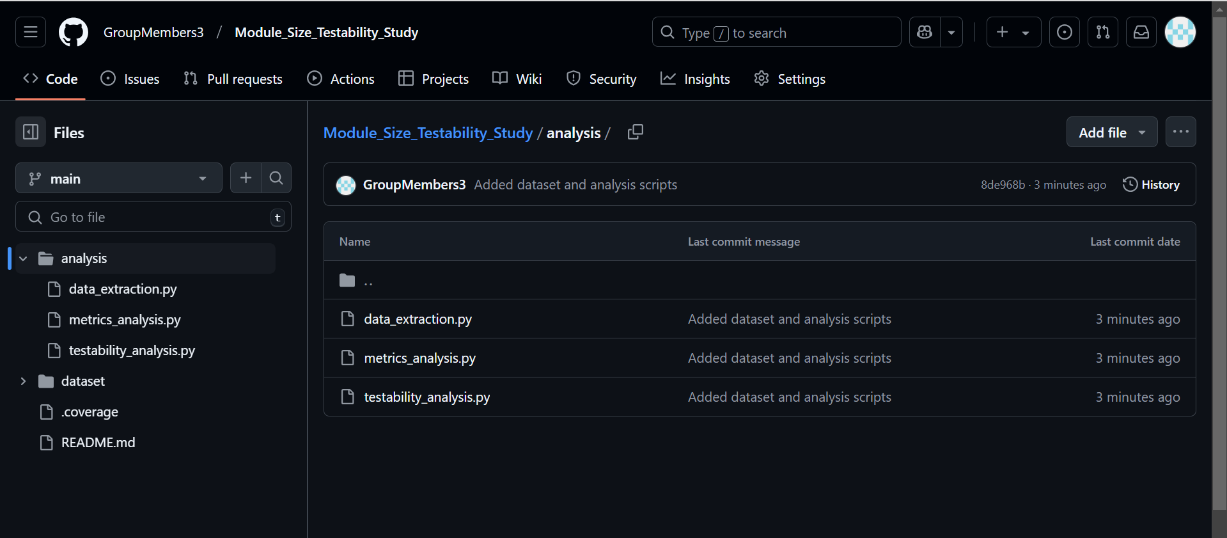
Step 9: Authorizing Git to local and successes.



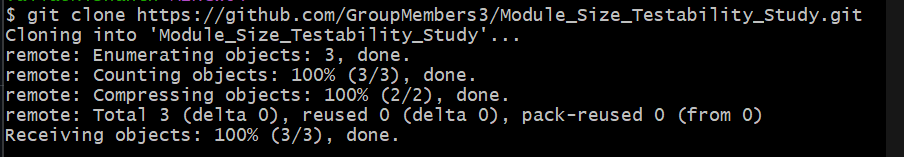
Step 10: Coping files to Git.



Step 11: Verifying files in Git.



Step 12: Cloning files successfully.



**Conclusion**

This study provides valuable insights into software modularization practices, emphasizing the balance between module size and testability. The findings support the hypothesis that software modularization should aim for medium-sized modules with moderate complexity. Future work can explore different programming languages, additional testability metrics, and machine learning techniques to predict optimal modular structures dynamically. A deeper investigation into the impact of software frameworks and architectural patterns on testability could further enhance software engineering best practices.

**Git Link.**

[**https://github.com/GroupMembers3/Module\_Size\_Testability\_Study/tree/main**](https://github.com/GroupMembers3/Module_Size_Testability_Study/tree/main)

**References:**

Sharma, A., Nagappan, M., & Shihab, E. (2022). "An Empirical Study on Maintainable Method Size in Java."

https://arxiv.org/pdf/2205.01842

[2] Parsai, A., Briand, L. C., & Bianculli, D. (2018). "A Survey on Software Testability."

https://arxiv.org/pdf/1801.02201

[3] Terragni, V., Salza, P., & Pezzè, M. (2020). "Measuring Software Testability Modulo Test Quality."

https://valerio-terragni.github.io/assets/pdf/terragni-icpc-2020.pdf