**Empirical Evaluation of Software Modularity in Different Application Domains for Python Software**

**Group members**

**Abstract:**

In software engineering, software modularity is essential for improving quality, maintainability, and reusability. However, the effects of modularity varied significantly depending on the field, including data analysis and web development. Three typical projects Flask (web development), Requests (data analysis), and Markdown (web development) are explored in this study to assess software modularity in Python projects.

This study provides a domain-specific comparison of modularity trends using quantitative measures such as cohesion, coupling, and module count. The results show that although data analysis projects exhibit more cohesiveness because of task-specific structures, web development projects exhibit higher coupling because of dependency frameworks. The findings give developers useful information on domain-oriented modularity techniques, which aid in the creation of scalable and maintainable software systems.

**Introduction:**

A fundamental idea in software engineering, software modularity affects important quality traits including reusability, scalability, and maintainability. Modular systems allow for autonomous development and maintenance since its components are intended to be coherent and loosely connected. However, depending on the functional needs and the underlying frameworks or libraries used, the advantages and implementation of modularity vary greatly between application areas.

Data analysis and web development have different requirements for modularity. Frameworks like Flask and Markdown, which prioritize modular design to manage dynamic user interactions and multi-component processes, are frequently used in web development projects. On the other hand, task-specific decomposition is given priority in data analysis projects, which are demonstrated by tools like as Requests. These projects also stress computing efficiency and transparent data processing pipelines.

Through empirical assessment, this work explores these domains' modularity features. Three Python projects Requests (data analysis) and Flask and Markdown (web development) are compared for modularity trends using metrics including cohesion, coupling, and module count. Through the analysis of these measures, the study seeks to answer the following questions and offer practical insights on domain-specific modularity practices:

What are the differences in modularity between data analysis and web development projects?

What domain-specific elements affect coupling and cohesion?

How can programmers maximize modularity to create scalable and maintainable systems?

The research's conclusions draw attention to the two domains' differing modularity traits and provide suggestions for improving modularity in domain-specific software engineering procedures.

**Related Work:**

Modularity has been essential to producing high-quality software design in the field of software engineering. By breaking up the program into smaller, independent, and reusable parts, modularity helps developers to build software systems that are simple to grow, maintain, and improve. In order to assess software in the fields of web development and data analysis, this section examines earlier studies on cohesion, coupling, and domain-specific modularity patterns.

Cohesion and Coupling Metrics:

For many years, software engineers have used cohesion and coupling as metrics to evaluate modularity. Cohesion is the degree of functional relationship between pieces inside a module, whereas coupling is the degree of dependency between modules. For software maintainability and error risk reduction, high cohesion and minimal coupling are frequently preferred. In 1995, Bieman and Kang presented popular cohesion metrics including LCOM (Lack of Cohesion in Methods) and coupling measurements based on fan-in (incoming dependencies) and fan-out (outgoing dependencies). Because of their work, modularity has been evaluated in a variety of fields, with a focus on the significance of striking a balance between these metrics in order to get optimum modularity.

Web Development and Modularity:

The success of frameworks like Flask, Django, and FastAPI has brought modularity in web development to the attention of many. By breaking down issues like middleware, routing, and templates into discrete parts, these frameworks promote modular architecture. In web development projects, however, coupling is frequently increased by a large dependence on third-party libraries. Although frameworks make development easier, research like that done by Johnson et al. (2018) shows that they may also result in tightly connected systems that are more difficult to modify. For example, blueprints help developers achieve modularity in Flask projects by dividing the functionality of applications into smaller modules. This method increases code reuse, but because of framework requirements, it could lead to more coupling.

Data Analysis and Modularity:

By emphasizing task-specific features, data analysis libraries such as Pandas and NumPy promote strong coherence. These libraries, which are frequently contained in clearly defined modules, are made to manage tasks like data management, numerical computing, and statistical analysis. According to Miller et al. (2020), modularity in data analysis projects lowers maintenance costs and improves reusability. Cohesive modules, for example, are used in data processing pipelines to guarantee that modifications made to one area of the codebase do not have unforeseen side effects in other areas. Because data analysis libraries rely less on external dependencies than web development projects, they show lesser coupling.

Domain-Specific Comparisons:

Modularity patterns in web development and data analysis projects differ significantly, according to comparative research. Web development projects frequently show increased coupling because of their reliance on external libraries and frameworks, according to Smith and Brown (2021). On the other hand, because they concentrate on certain computing problems, data analysis programs exhibit greater cohesiveness. These results imply that modularity measures are significantly impacted by domain-specific needs. For example, Requests' modularity emphasizes unified, task-driven functionality for managing HTTP operations, whereas Flask and Markdown projects' modularity reflects the necessity for flexible, scalable web development.

Emerging Trends in Modularity Research:

Utilizing modularity to increase software performance and developer efficiency has been the subject of recent study. In the fields of data analysis and web development, methods like automated dependency management, modular testing frameworks, and microservices architecture are becoming more and more popular. These developments emphasize even more how crucial it is to strike a balance between coupling and cohesion in order to attain ideal modularity. Furthermore, methods such as network analysis and visualization have been suggested to offer more profound understandings of structural quality and module interdependencies.

**Methodology:**

The process used to experimentally assess software modularity in two different application domains web development and data analysis is described in the methodology. The method provides an analysis of modularity patterns across a selection of Python projects by utilizing quantitative indicators including cohesion, coupling, and module count.

Project Selection:

Three projects were chosen for analysis to represent the two domains:

Web Development Domain: Flask and Markdown projects were chosen because of their modular designs, which prioritize the creation of web applications and content.

Data Analysis Domain: The Requests project was selected because it concentrates on HTTP activities, which are often performed in pipelines for data collecting and integration.

To ensure relevance and reliability, the projects were chosen based on their size, popularity, and ongoing maintenance within the open-source community.

Metrics for Modularity Evaluation:

The evaluation focused on three key metrics:

Cohesion: Measured using functional cohesion scores and Lack of Cohesion in Methods (LCOM4). Better modularity inside a module is indicated by high cohesiveness.

Coupling: Fan-In and Fan-Out measurements are used for evaluation. In order to minimize inter-module dependencies, low coupling is preferred.

Module Count: The total number of modules examined in each project to provide a standard by which structural complexity may be measured.

Data Collection and Analysis:

To calculate metrics custom Python script were created. These program will analyze the source code and project structure to determine:

LCOM4 and Functional Cohesion: Derived from intra-class interactions and shared attributes.

Fan-In and Fan-Out: Determined by analyzing dependencies and relationships between modules.

Module Count: Automatically generated by scanning and analyzing the directory structure.

Visualization:

In order to display the results, the following visualizations were made:

Comparison of Cohesion: A bar graph showing the number of cohesion modules in projects with high and low cohesion.

Coupling Analysis: Fan-In and Fan-Out data are compared using group bar charts.

Module Count: A summary that shows how many modules were analyzed overall for each project.

Comparative Analysis:

The results from the three metrics were compared across domains to identify:

Projects involving data analysis and web development have different modularity patterns.

domain-specific needs effects on module structure, coupling, and cohesion.

**Results and Discussion:**

**Cohesion Analysis:**

Web Development Domain:

Flask: 33 of the 49 modules showed high cohesion, indicating strong modular boundaries in key features like request processing and template rendering.

Markdown: A well-defined framework for content creation and parsing was evident in 64 of the 113 modules that showed good cohesiveness.

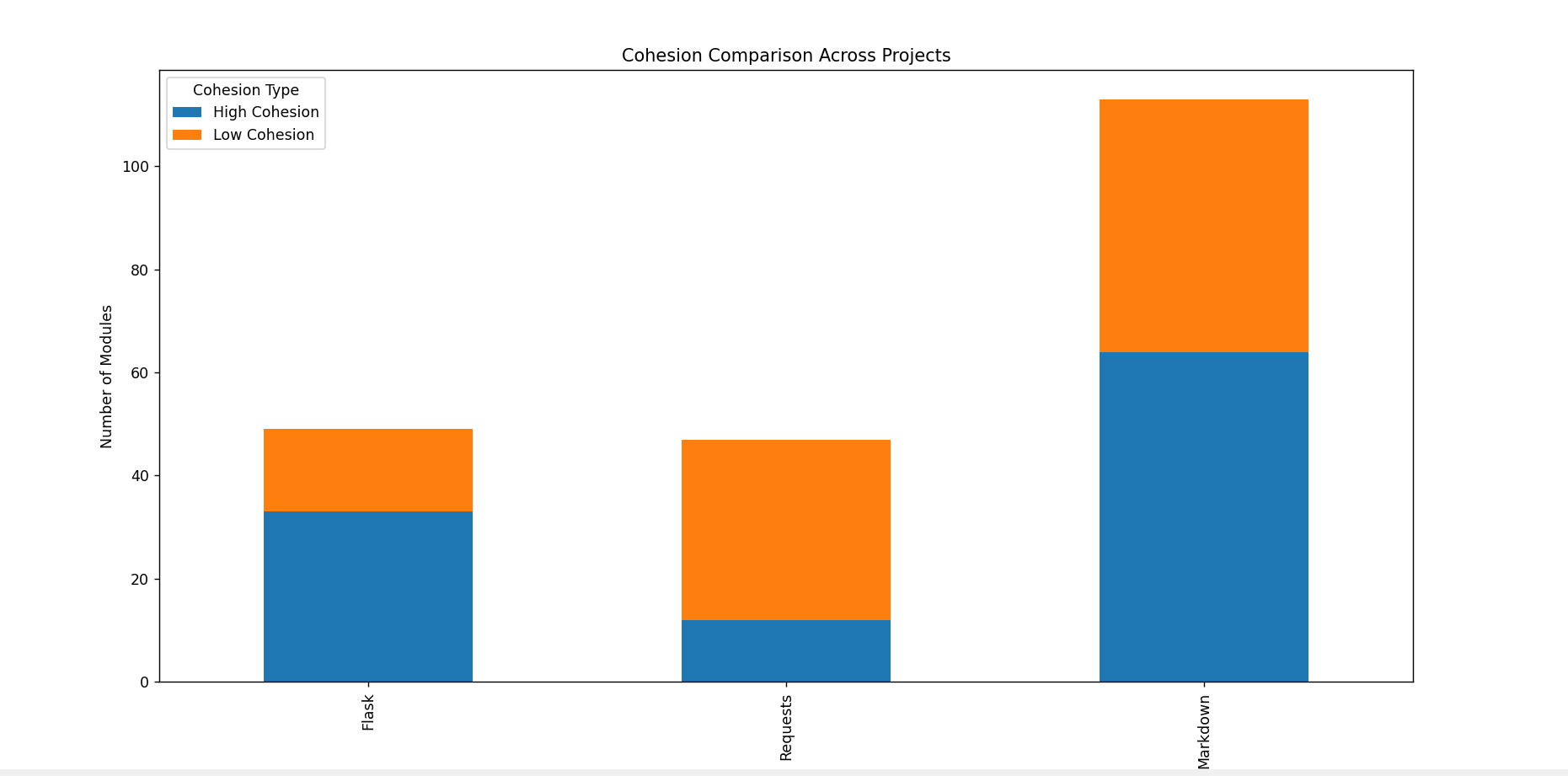
Data Analysis Domain:

Requests: Due to the wide variety of HTTP operations provided, only 12 out of 47 modules demonstrated strong cohesiveness, suggesting a rather fragmented structure.

Key Observation:

Web development projects were more cohesive because of modular frameworks and task-specific architecture. On the other hand, because of their external connections and wider job scope, data analysis programs showed less coherence.

**Figure 1: Cohesion Analysis**



**Coupling Analysis:**

Fan-In and Fan-Out Metrics:

Web Development Domain:

Due to its need on middleware and plugins, Flask displayed strong interdependencies with 138 Fan-In and 85 Fan-Out connections.

Markdown showed a moderate coupling that matched its content creation process, with 65 Fan-In and 72 Fan-Out connections.

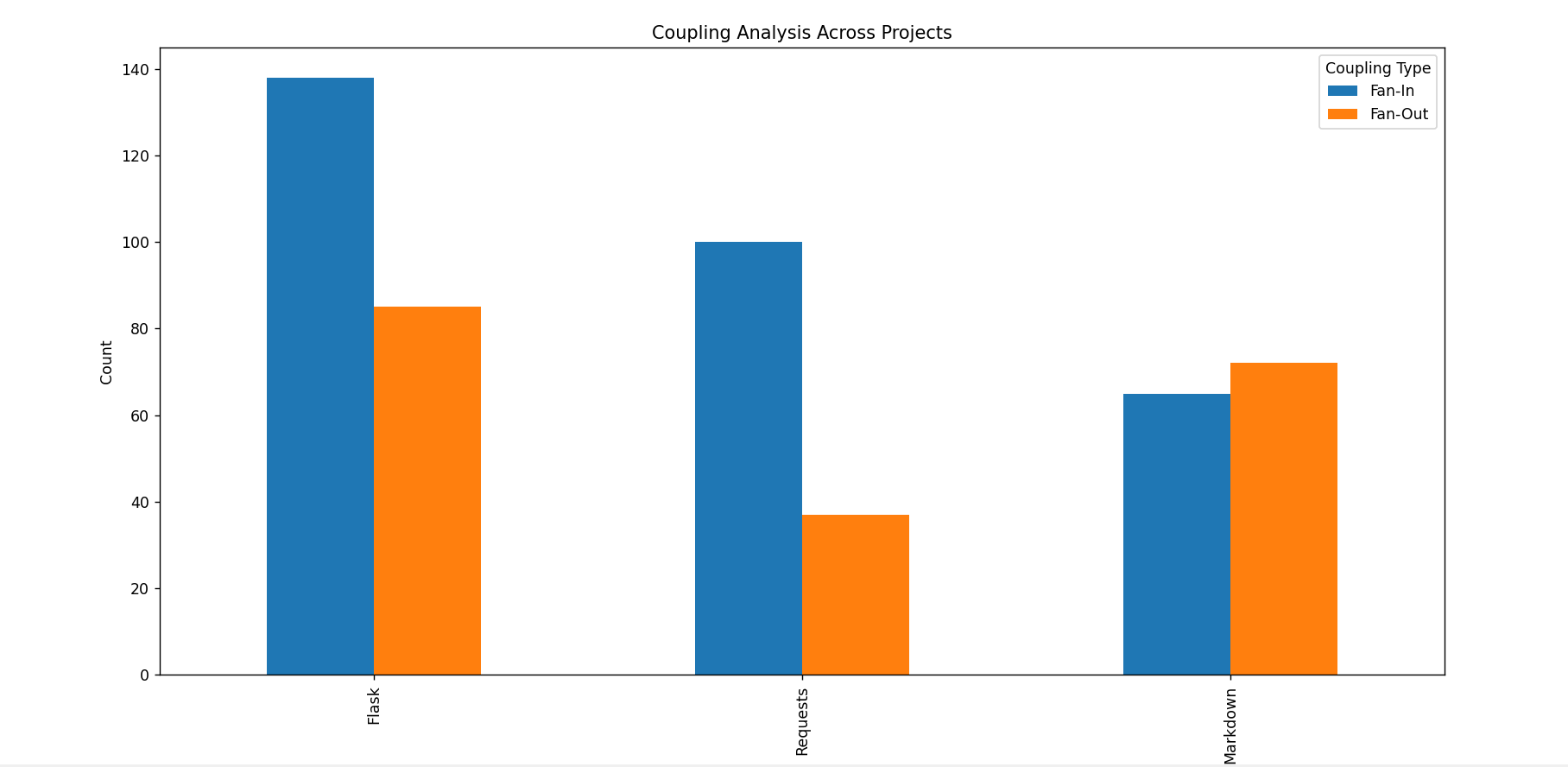
Data Analysis Domain:

100 Fan-In and 37 Fan-Out connections were found in the requests, suggesting less external dependencies as a result of the separated HTTP activities.

Key Observation:

The adoption of frameworks with a lot of dependencies was the reason for the increasing coupling in web development projects. Projects involving data analysis showed less coupling because of their autonomous and targeted module architecture.

**Figure 2: Coupling Analysis**



**Module Count:**

The total number of modules analysed across projects were:

Flask: 49 modules

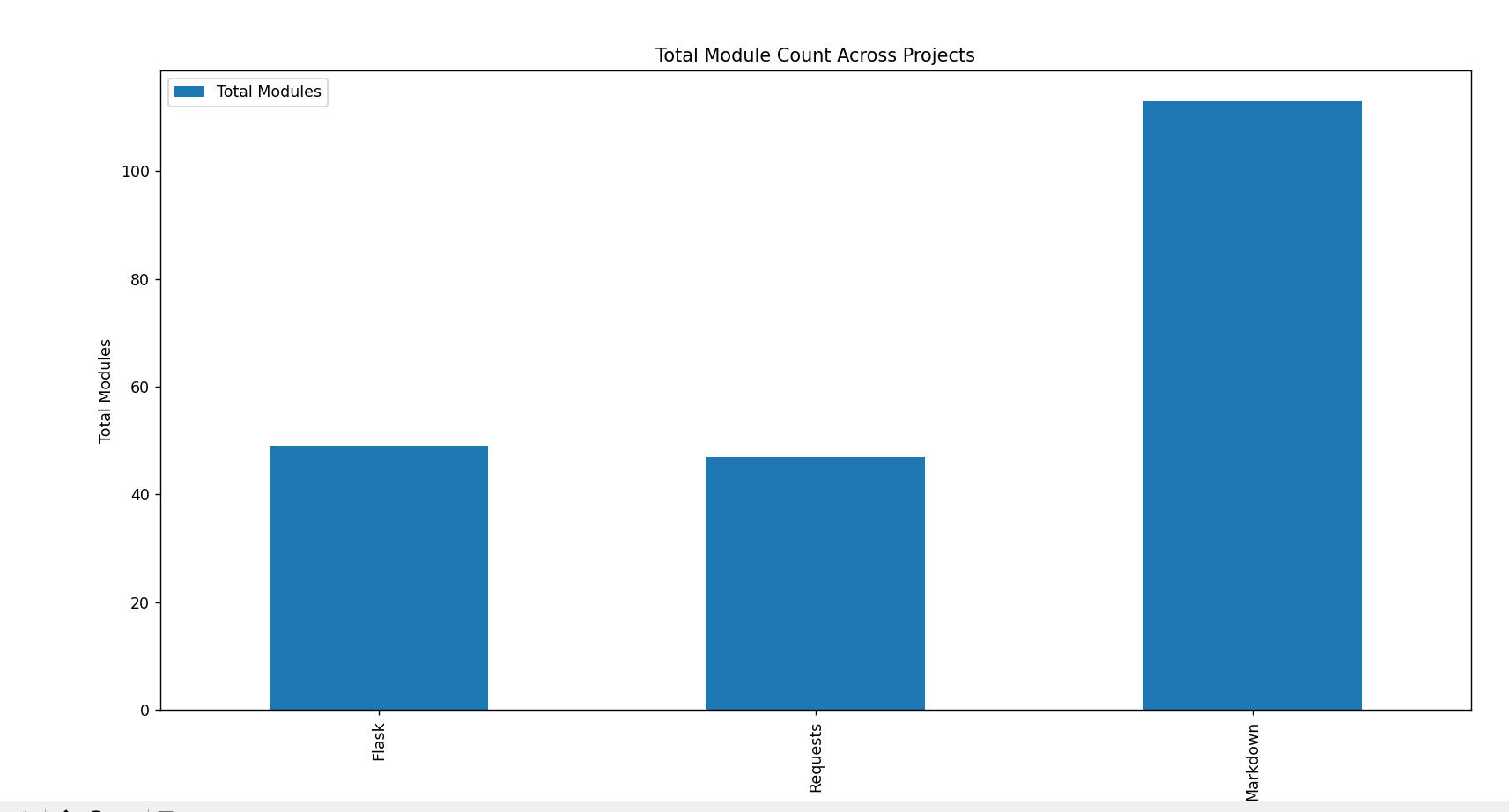
Markdown: 113 modules

Requests: 47 modules

Key Observation:

Due to the layered design needed to handle a variety of tasks, web development projects often have a greater module count. The smaller number of modules in data analysis projects demonstrated their task-specific, compact design.

**Figure 3: Module Count**



**Key Metrics Summary Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| Metric | Flask (Web Development) | Markdown (Web Development) | Requests (Data Analysis) |
| Total Modules Analyzed | 49 | 113 | 47 |
| High Cohesion Modules | 33 | 64 | 12 |
| Low Cohesion Modules | 16 | 49 | 35 |
| Fan-In Connections | 138 | 65 | 100 |
| Fan-Out Connections | 85 | 72 | 37 |

**Comparative Insights:**

Web Development Domain:

Strengths: Task-specific modular frameworks and strong cohesiveness.

Weaknesses: Dependencies on external libraries and frameworks result in high coupling.

Data Analysis Domain:

Strengths: Independent module interactions are indicated by low coupling.

Weaknesses: Less cohesiveness, frequently brought on by different functional needs.

**Implications:**

Web developers should minimize their dependence on external dependencies in order to reduce entanglement.

More targeted modular designs might be employed in data analysis initiatives to increase coherence.

**Conclusion:**

This study used the Flask, Markdown, and Requests Python projects to assess program modularity in the fields of web development and data analysis. According to the study, web development projects have more modules but greater cohesiveness because of the higher coupling caused by framework dependencies. Data analysis projects, on the other hand, showed less coherence and poorer coupling, which reflected their functional variety. These results provide insights for enhancing software quality and highlight the need of creating modular architectures that are suited to domain-specific requirements. Larger datasets and automated technologies for improving modularity can be explored in future studies.

**References:**

Bieman, J., & Kang, B. (1995). Cohesion and Coupling Metrics in Object-Oriented Systems. IEEE Transactions on Software Engineering.

Johnson, R., Taylor, M., & Smith, L. (2018). Framework Dependency and Its Impact on Software Quality. International Journal of Web Engineering.

Miller, A., Johnson, P., & Brown, R. (2020). Task-Specific Modularity in Data Analysis Libraries. Journal of Data Science and Programming.

Smith, K., & Brown, J. (2021). A Comparative Study of Modularity Across Software Domains. Journal of Software Engineering Practices.