MANIPAL
ACADEMY *of* HIGHER EDUCATION
*(Institution of Eminence Deemed to be University)*

# Static Code Analyser Tool

**Mini-Project Synopsis**

*submitted to*

Manipal School of Information Sciences, MAHE, Manipal

| Reg. Number | Name | Branch |
|---|---|---|
| **241059027** | **Anusha S Patil** | **CYS** |
| **241059032** | **K Anusha Rao** | **CYS** |
| **241059033** | **S Srinivas** | **CYS** |

**26/01/2025**

MANIPAL SCHOOL OF INFORMATION SCIENCES
MANIPAL
*(A constituent unit of MAHE, Manipal)*

# Table of Contents

## Project Description

Good code quality is a fundamental need in today’s modern world. Since source code is used universally to develop every software that runs on an electronic device, it is essential that source code must be well written and maintained in order to handle updates and enhancements.

When a software developer or code security analyst needs to analyse source code to detect security flaws and maintain secure, high-quality code, manual review may not uncover all issues. Even experienced security analysts can overlook vulnerabilities. A source code analysis tool addresses this challenge by automatically and efficiently identifying potential security flaws without executing the code. It serves as a reliable complement to manual inspection, enhancing accuracy and thoroughness. A Static Code Analyser is a software tool that examines source code for potential errors, vulnerabilities, and adherence to coding standards without executing the program. This report provides an in-depth view of a Python-specific static code analyser, detailing its architecture, functionality, and use in maintaining high-quality Python projects.

## Objectives

- Identify Common Code Defects and Vulnerabilities.
- Develop a Basic Static Code Analysis Tool.
- Design an interface for users to upload, scan, and review code analysis reports.

## Literature Survey

The literature survey gives a brief overview about the various security vulnerabilities which can be detected by a Static code Analyser and also the existing tools available.

### Evaluating Python Static Code Analysis Tools Using FAIR Principles

**Authors:** H. B. Hassan, Q. I. Sarhan, and Á. Beszédes
**Tools:** Prospector, Pylint, SonarQube, Pynocle
**Methodology:** Evaluates Python static code analysis tools using FAIR principles to enhance usability.
The study assesses various static analysis tools based on FAIR principles to improve their usability and accessibility. The research highlights the strengths and limitations of each tool in terms of findability, accessibility, interoperability, and reusability. As a future development, a web application is proposed to help developers choose and compare tools based on their programming language, analysis needs, and ease of integration.

### Static Code Analyser to Enhance Developer Productivity

**Authors:** D. R. I. Peiris and N. Kodagoda
**Tools:** None specified
**Methodology:** Develops a static code analysis tool using Abstract Syntax Trees (AST) to improve coding practices.
The research introduces a tool designed to enhance developer productivity by detecting structural code issues through static analysis. It aims to improve coding practices by identifying potential problems early in the development process. The study suggests future improvements, including refining the algorithm to detect complex issues such as tightly coupled code and collusion. Additionally,

integrating AI and ML techniques could enhance issue detection and predictive capabilities, making the tool more efficient and valuable for developers.

**Towards More Sophisticated Static Analysis Methods of Python Programs**

**Authors:** H. Gulabovska and Z. Porkoláb
**Tools:** Code Sonar, Klocwork, Coverity, PyLint, Pyflakes
**Methodology:** Investigates symbolic execution as an advanced static analysis method compared to traditional approaches.
The study explores symbolic execution as an alternative to traditional static analysis methods, demonstrating its ability to improve the accuracy of error detection. By leveraging symbolic execution, the research highlights how this technique can analyze code paths more effectively than conventional static analysis methods. The study suggests that future tools could incorporate powerful SAT solvers, such as Z3, to further improve precision, automate complex code evaluations, and provide better vulnerability detection for Python programs.

**Static Type Analysis for Python**

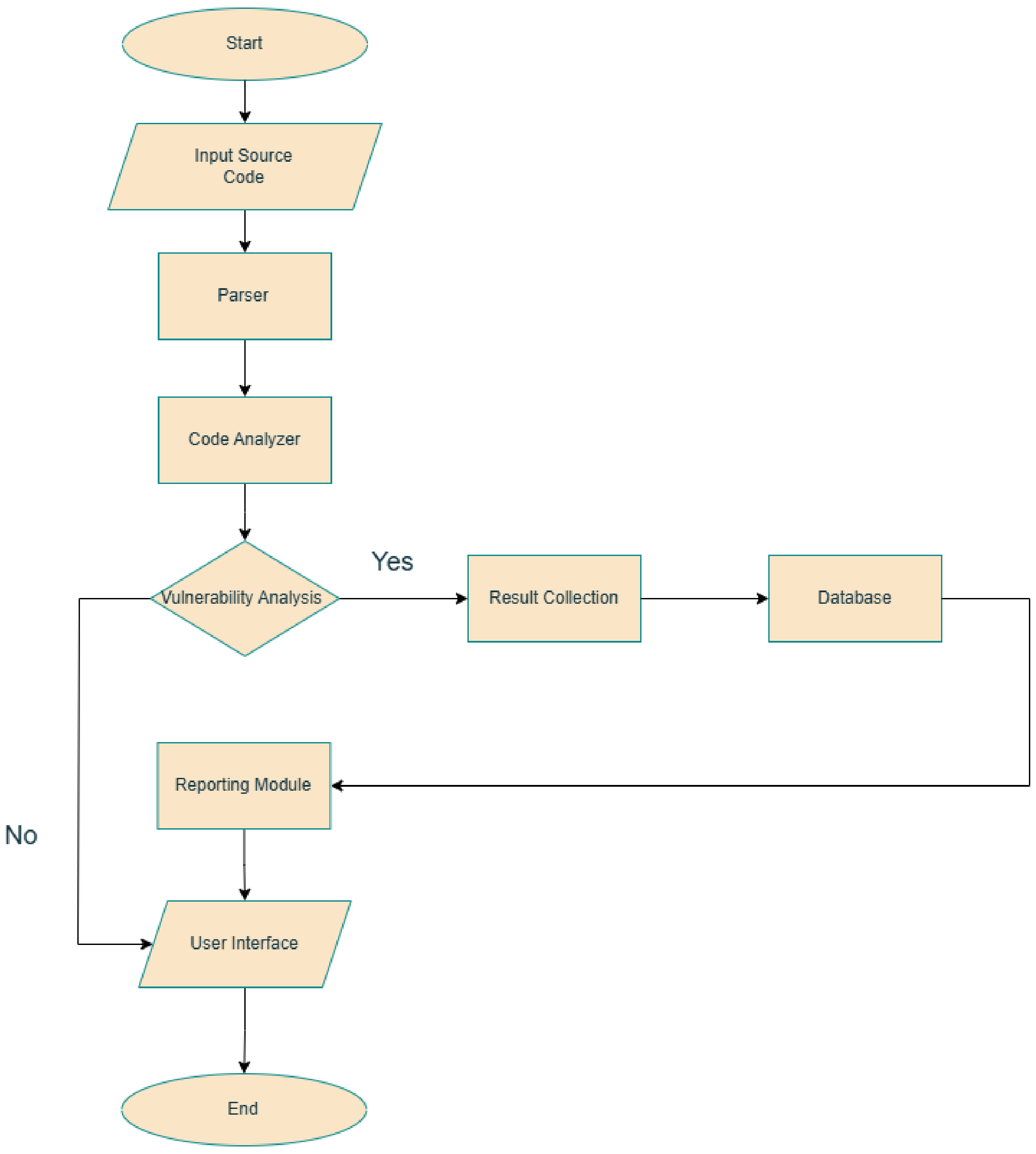
**Authors:** T. Dong, L. Chen, Z. Xu, and B. Yu
**Tools:** PType
**Methodology:** Develops PType, a tool for static type analysis in Python using type inference and constraint graphs.
This research presents PType, a static type analysis tool designed to improve Python's type checking by automatically annotating classes, functions, and built-in modules. The tool constructs a constraint graph to infer types and detect potential inconsistencies in codebases. The study emphasizes the importance of type inference in large-scale Python projects where dynamic typing can lead to unexpected runtime errors. Future enhancements include expanding PType's type-checking capabilities and integrating it with other static analysis tools to

provide a more comprehensive solution for developers working on complex Python applications.

### A Large-Scale Security-Oriented Static Analysis of Python Packages in PyPI

**Authors:** J. Ruohonen, K. Hjerppe, and K. Rindell
**Tools:** Bandit
**Methodology:** Conducts a large-scale static security analysis of Python packages in PyPI to identify vulnerabilities.
This study focuses on analysing Python packages from the Python Package Index (PyPI) to identify security vulnerabilities using static analysis techniques. By employing Bandit, a security-focused static analysis tool, the research uncovers common security risks in widely used Python packages. The findings suggest that many packages suffer from weak security practices, making them susceptible to attacks. The study recommends collecting software metrics to refine vulnerability prediction models and suggests longitudinal analysis to monitor how security risks evolve over time. These insights can help improve security policies and best practices for open-source Python package development.

## Block Diagram/Flowchart



## Applications

- **Immediate Feedback on Code Issues:** Source code analysis tools, also known as SAST tools, provide instant feedback to developers, helping them address vulnerabilities and errors early in the development process rather than later in the Software Development Life Cycle (SDLC).
- **Improved Code Quality from the Start:** These tools encourage developers to focus on creating secure and high-quality code right from the beginning, reducing the risk of introducing security flaws during development.
- **Cost-Effective Solutions for Security:** While many commercial tools are expensive, there are free and open-source static code analysis tools available, making them ideal for startups and freelancers who need robust security solutions without breaking the budget.
- **Wide Adoption Across Industries:** Due to the increasing demand for secure coding practices, these tools are widely adopted across organizations, ranging from individual developers to large enterprises, to ensure code safety and compliance with standards.

## Software & Hardware Requirements

### Hardware Requirements

| | |
|---|---|
| Processor | Dual-core CPU (e.g., Intel Core i3 or higher) |
| RAM | 4 GB |
| Disk Space | 10 GB |

### Software Requirements

| | |
|---|---|
| OS | Windows, Linux |
| IDE | VS Code |
| Plugins | PyLint |

## References

[1] H. B. Hassan, Q. I. Sarhan and Á. Beszédes, "Evaluating Python Static Code Analysis Tools Using FAIR Principles," in IEEE Access, vol. 12, pp. 173647-173659, 2024, doi: 10.1109/ACCESS.2024.3503493.

[2] D. R. I. Peiris and N. Kodagoda, "Static Code Analyser to Enhance Developer Productivity," 2023 IEEE 8th International Conference for Convergence in Technology (I2CT), Lonavla, India, 2023, pp. 1-6, doi: 10.1109/I2CT57861.2023.10126395.

[3] H. Gulabovska and Z. Porkoláb, "Towards More Sophisticated Static Analysis Methods of Python Programs," *2019 IEEE 15th International Scientific Conference on Informatics*, Poprad, Slovakia, 2019, pp. 000225-000230, doi: 10.1109/Informatics47936.2019.9119307.

[4] T. Dong, L. Chen, Z. Xu and B. Yu, "Static Type Analysis for Python," *2014 11th Web Information System and Application Conference*, Tianjin, China, 2014, pp. 65-68, doi: 10.1109/WISA.2014.20.

[5] J. Ruohonen, K. Hjerppe and K. Rindell, "A Large-Scale Security-Oriented Static Analysis of Python Packages in PyPI," in 2021 18th International Conference on Privacy, Security and Trust (PST), Auckland, New Zealand, 2021, pp. 1-10, doi: 10.1109/PST52912.2021.9647791.