Graduate School of Science and Technology Master's Thesis Abstract

Laboratory name (Supervisor)	Software Engineering (Ken-ichi Matsumoto (Professor))		
Student ID	2311335	Submission date	2025 / 1 / 21
Name	INDIRA FEBRIYANTI		
Thesis title	Comparing the Maintainability Between Different Code Proficiencies in PyPI Projects		

Abstract

Python's popularity stems from its ease of learning and a supportive community, fostering diverse contributions to reusable codebases that require active maintenance. As a vital resource for the Python community, PyPI offers an extensive repository of Python packages supporting diverse applications. However, the growing demand for rapid software development has led developers to increasingly modify existing frameworks rather than build new code. Despite the community's emphasis on simplicity in code structure, ensuring proficiency in contributed code remains challenging, particularly as structural complexities emerge during maintenance. This study explores the correlation between proficiency levels and maintainability in Python projects. It analyzes 312 Python projects, comprising 371,369 Python files and 940,205 code snippets, classified by proficiency and maintenance levels (high and low), using metrics such as cyclomatic complexity, Halstead, lines of code, and readability. The research addresses two questions: (1) What is the proficiency of safe and risky code? (2) What proficient code elements are more likely to be risky? The study finds that while proficient-rated Python code is generally low risk, high-risk cases persist across all proficiency levels due to structural complexities. Commonly observed code elements, such as Simple List Comprehensions, Generator Expressions, "Enumerate" Call Functions, "Zip" Call Functions, and List Comprehensions with 1 If Statement, frequently appear in both safe and risky code, with their maintainability significantly influenced by their context within classes, functions, or methods. The results provide insights into when using different code proficiencies can lead to different maintenance efforts, and that using proficient code does not always result in low maintenance.