Abstract

It's becoming significantly easier to develop, write and maintain code with the addition of natural language learning models (such as Chat-GPT). This combined with the ability to obfuscate your codebase makes it more difficult to detect similarity between programs. The research problem was to discover whether through regression testing via fuzz testing and profiling classes that based on the data that is extracted a similarity score between programs could be determined.

The quest to effectively detect code similarities has prompted innovative approaches beyond traditional static analysis methods. This research investigates the viability of utilizing runtime information as a metric for code similarity detection, with the aim of reshaping current paradigms in this domain. Various methodologies were explored, including leveraging Java Quick Fuzzing (JQF) in conjunction with tools like VisualVM and Java Command-Line Management (JCMD), to capture runtime data such as CPU usage and memory behavior. Despite encountering challenges in accurately profiling target applications and extracting desired data formats, these initial attempts provided valuable insights into potential metrics for consideration.

The evaluation phase involved analyzing runtime data extracted from a diverse dataset of user submissions. The dataset was organized and formatted into a spreadsheet for comprehensive analysis. Through meticulous examination, both similarities and differences between user submissions were uncovered. Notable findings include instances of code segments exhibiting similar structures and variable naming conventions, indicative of potential code reuse or plagiarism. Moreover, comparisons were made between the runtime scores obtained from the dataset and similarity scores generated by the compare50 tool. Results revealed discrepancies in the closest pairs identified by runtime scores and those identified by compare50, underscoring the need for further investigation and refinement of code similarity detection methodologies.

In conclusion, this research contributes to the ongoing discourse on code similarity detection by proposing runtime information as a promising avenue for exploration. By shedding light on the challenges and opportunities associated with leveraging runtime data, this study lays the groundwork for future advancements in this field. Ultimately, the findings advocate for a holistic approach to code similarity detection, one that incorporates runtime metrics alongside existing static analysis techniques to enhance the accuracy and effectiveness of plagiarism detection tools.