ATM: Black-box Test Case Minimization based on Test Code Similarity and Evolutionary Search

1 Idea

Despite the excellent performance of ATM [1] in fault detection rate, its average execution time of 1.1 to 4.3 hours may still be too long for large industrial systems. This may limit its widespread use in real-world development, especially in scenarios that require fast feedback. We can try to speed up the execution time by parallelizing the computation.

In the ATM method, the redundancy removal process relies on similarity measures to identify redundant test cases for elimination. However, this approach may be inefficient, waste plenty of time or fail to effectively identify redundant test cases, especially in complex test sets, or in the context of security testing. Based on the ideas in the AIM (Automated Input Minimization) [2] method, input selection and redundancy removal techniques can be enhanced to further optimize ATM's performance. AIM introduces a black-box clustering-based input selection strategy, which clusters test cases based on their security features, and retains representative test cases from each cluster. This not only reduces redundancy but also ensures the coverage of different vulnerabilities.

2 Describe the process in the form of requirement engineering

Elicitation:

(1) Interview test engineers and developers about their execution time requirements for test case minimization tools in real-world development, especially in scenarios that require rapid feedback. Collect the needs and expectations of the development team for parallelized computing, focusing on the applicability of the tool in multicore processors or distributed computing environments.

(2) Analyze the execution time bottleneck of existing ATM tools and identify computational tasks that can be parallelized, such as similarity calculation and search algorithm execution. Evaluate the potential performance benefits of parallel computing.

(3) Identifying optimization opportunities by analyzing redundancy removal processes in existing tools such as ATM. Evaluating current similarity metrics and redundancy removal strategies, especially their performance on large test sets.

Specification:

(1) Functional: Provides parallel computing capabilities, supports multi-core processors and distributed computing resources. Implement task allocation and result merging mechanism to ensure the correctness and efficiency of parallel computing. Security-Feature-Based Selection: Prioritize test cases that represent diverse vulnerabilities to maximize coverage while minimizing redundancy.

(2) Nonfunctional: The execution time is reduced to less than 1 hour on multicore

processors. Tools should support multiple computing resources, and have ability to handle large test suites.

Validation:

Develop a prototype tool to support parallel computing, redundancy removal and verify its performance improvement in multi-core processors and distributed computing environments. Besides, we also need to verify coverage equivalence between minimized and original test sets. Collect feedback from test engineers and developers on the prototype tool to ensure that the performance improvement of the tool meets the actual needs.

Negotiation:

Negotiate with the development team about the priorities of the parallelized computing functions and the scope of the parallelized computing functions. Determine the human, time, and technical resources required by the development team during the development of parallelized computing functions. Aligning with developers on priorities, scope, and resource allocation. Reaching consensus on priorities, technical requirements, and resource allocation for successful implementation.

3 Reference

[1] Pan R, Ghaleb T A, Briand L. Atm: Black-box test case minimization based on test code similarity and evolutionary search[C]//2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, 2023: 1700-1711.

[2] Chaleshtari N B, Marquer Y, Pastore F, et al. AIM: Automated Input Set Minimization for Metamorphic Security Testing[J]. IEEE Transactions on Software Engineering, 2024.