MuMuTestUp: Mutation-based Multi-Agent Test Case Update

ANONYMOUS AUTHOR(S)

ACM Reference Format:
Anonymous Author(s). 2018. *MuMuTestUp*: Mutation-based Multi-Agent Test Case Update. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

A Tools

In this section, we describe the functionality and design rationale of each tool in detail.

rank_hunks: Inspired by the study of Yaraghi et al. [**?**], this tool ranks all diff hunks using a combination of Repetition and Test TF-IDF Similarity, and returns an ordered list of diff hunks. The ranking prioritizes code change diff hunks that are more likely to be relevant to test case updates.

extract_test_code: This tool extracts the updated test code and the required import statements from the output of the Test Update agent, and writes them back to the corresponding locations in the repository, ensuring that the test execution environment is correctly configured.

run_test_coverage: This tool executes the updated tests and generates coverage reports using JaCoCo; the reports are later consumed by the Coverage Analysis agent.

run_test_mutation: This tool performs mutation testing and generates mutation reports using PITest, the reports are later consumed by the Mutation Analysis agent.

record_best: This tool records the best test case produced during the iterative update process. Each updated test is scored using the predefined scoring function, and the test case with the highest score is retained as the existing best result.

$$score = \begin{cases} -1000 & for \quad compile\ error \\ -100 & for \quad assert\ fail \\ 1000 + (line\ cov + branch\ cov + mutation\ score) \times 1000 & for \quad other \end{cases} \tag{1}$$

has_done: This tool determines whether the update process should terminate based on the test execution results and the iteration count. It returns True if the test case compiles and executes successfully and both coverage and mutation scores reach the predefined thresholds, or if the maximum number of iterations has been reached.

choose_agent: Based on the test execution results, this tool selects the appropriate analysis agent. If compilation fails or assertion errors occur, it returns Error Analysis; if line or branch coverage does not meet the threshold, it returns Coverage Analysis; if the mutation score is below the threshold, it returns Mutation Analysis.

merge_instructions: This tool extracts update instructions from the outputs of different analysis agents and merges them into a key-value representation.

- If the instructions originate from the Error Analysis agent, they are stored under the key "error analysis result".
- If the instructions originate from the Coverage Analysis agent, they are stored under the key "coverage analyzsis result". When easier-to-cover lines or branches exist, only the corresponding instructions are retained; otherwise, instructions for all uncovered lines or branches are included.
- If the instructions originate from the Mutation Analysis agent, they are stored under the key "mutation analysis result". When surviving mutants exist, instructions targeting surviving mutants are returned; otherwise, instructions for uncovered mutants are included.

extract_error_info: This tool analyzes test execution failure logs and extracts failure messages using regular expressions. It returns both the error information and the set of symbols involved in the failure.

distinguish_unknown_symbols: Based on a predefined set of standard library and testing framework symbols, this tool classifies symbols extracted from error logs into known symbols and unknown symbols. Symbols not appearing in the predefined set are marked as unknown.

locate_error: This tool locates the exact line of code that caused the error using the file path and line number reported in the error log, and returns the corresponding line number and code snippet.

gen_method_error_annotations: This tool annotates the focal method with comments indicating compilation or assertion errors. Given the focal method and a set of ⟨location, annotation⟩ pairs, it returns the annotated method code.

extract_uncover_info: This tool parses coverage reports to extract coverage information for the focal method, and returns lists of uncovered lines and uncovered branches.

gen_method_coverage_annotations: This tool annotates the test case with comments indicating covered and uncovered lines or branches. Given the test case and a set of ⟨location, annotation⟩ pairs, it returns the annotated method code.

extract_mutation_info: This tool parses mutation testing reports to identify mutants in the focal method that are not killed, and returns the corresponding list of surviving mutants.

gen_method_mutation_annotations: This tool annotates the test case with with comments indicating killed and survived mutants. Given the test case and a set of ⟨location, annotation⟩ pairs, it returns the annotated method code.

create_chromaDB:. This tool builds a ChromaDB vector database for a selected module of the project, enabling semantic similarity–based retrieval.

generate_embedding: This tool generates vector representations for query texts using text-embedding-ada-002 and returns the corresponding embeddings.

query_info: This tool provides a unified retrieval interface. It computes cosine similarity between embeddings and returns the query results.