EXPLORING TEST SUITE COVERAGE OF LARGE LANGUAGE MODEL-ENHANCED UNIT TEST GENERATION

A Study on the Ability of Large Language Models to Improve the Understandability of Generated Unit Tests Without Compromising Coverage

TU Delft Delft University of Technology

Author: Andrei Drăgoi (A.Dragoi@student.tudelft.nl)

Supervisor: Amirhossein Deljouyi
Responsible Professor: Andy Zaidman

1. INTRODUCTION

- Search-based software testing (SBST) tools, such as EvoSuite [1], use genetic algorithms to generate test suites that can achieve adequate coverage [2].
- The growing popularity of Large Language Models (LLMs) is becoming increasingly evident, and researchers are actively pursuing techniques to automate test generation with the help of LLMs [3; 4].
- UTGen [5], a tool integrating LLMs with EvoSuite, produces more understandable tests than EvoSuite; however, the generated tests suffer a coverage drop.
- This research explores the ability of Large Language Models to improve the understandability of generated unit tests without compromising coverage.

2. RESEARCH QUESTIONS

The key research goal is to identify the causes behind coverage shortages in LLM-guided SBST compared to conventional SBST and, subsequently, address these shortfalls. This overarching objective can be further subdivided into the following sub-questions:

RQ1 Which of UTGen's phases impact the test suite coverage?

RQ2 How can the factors contributing to inferior coverage in UTGen be mitigated?

3. UTGEN ANALYSIS SETUP

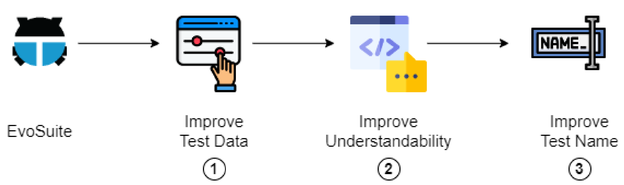

Figure 1: Overview of UTGen's phases.

Figure 1 illustrates the phases that UTGen undergoes:

- EvoSuite uses a Generic Algorithm to produce a test suite for a given project.
- UTGen's Test Data improvement phase (**1** in Fig. 1) enhances test data, such as the parameters used in method calls.
- UTGen's Understandability improvement phase (**2** in Fig. 1) focuseson adding descriptive comments and improving variable names.
- UTGen's Test Name improvement phase (**3** in Fig. 1) focuses on giving a descriptive name to each test.

We investigated the factors that negatively affect the coverage of the generated test suite in UTGen via two studies: **Phase isolation** and **Manual inspection**.

A) Phase isolation

For each class under test, we ran EvoSuite once. We used the generated tests as a basis in **four independent runs** of UTGen: only Test Data, only Understandability, only Test Name, and full UTGen.

B) Manual inspection

We manually compared the test suites generated in the initial UTGen experiment and classified them into several categories based on the reasons believed to cause the coverage drop.

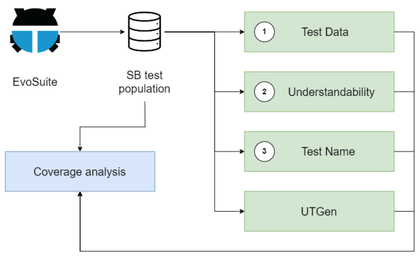

Figure 2: Study setup for running each of UTGen's phases in isolation, using the same test population as the foundation.

The dataset used as a basis for our analysis is a subset of EvoSuite's SF110 [6]. The subset was chosen based on the results of the UTGen experiments. In particular, classes that manifested a decrease in coverage in UTGen compared to EvoSuite were included in our investigations.

4. UTGEN ANALYSIS RESULTS

A) Phase isolation

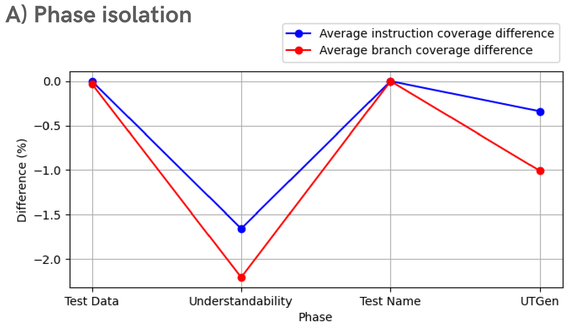

Figure 3: Coverage difference between EvoSuite and each phase of UTGen: Test Data, Understandability, Test Name, and full UTGen.

- The **Understandability** phase of UTGen has the biggest negative impact, with a **2.2%** decrease in branch coverage.
- The Test Data phase has a minor impact on coverage, with a decrease of **0.03%** in branch coverage.
- The **Test Name** phase does **not impact** the test suite coverage.

B) Manual inspection

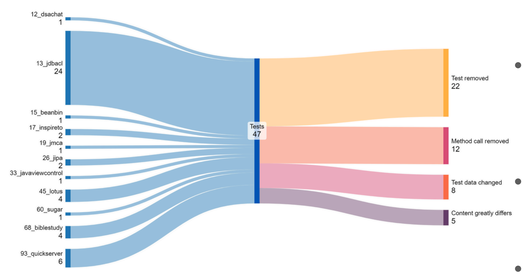

Figure 4: Manual inspection results: The 47 tests, part of 11 projects identified to have a negative impact on coverage were categorized into groups based on the cause of the coverage drop.

- **Test removed**: 47% - EvoSuite removes unstable/non-compiling tests.
- **Method call removed**: 25% - tests missed method calls / constructors after UTGen's improvements.
- **Test data changed**: 17% - parameter changes by UTGen lead to uncovered corner cases.
- **Content greatly differs**: 11% - tests had significant changes in test logic (e.g., different objects were instantiated).

- Coverage impact estimation was based on the number of tests showing the behavior, as a direct correlation with the effect on coverage is impossible.
- The underlying cause remains uncertain. We hypothesize that all issues stem from hallucinations within the LLM, potentially arising from insufficient contextual information or the inherent constraints of the chosen LLM model.
- Five inspected tests exhibited clear hallucination, with method calls not present in the test case or the method under test.

5. UTGENCOV APPROACH

Based on the findings of *RQ1*, we constructed **UTGenCov**, an extension of UTGen:

- It uses a grounding technique [7] to limit LLM hallucinations by providing more context.
- Specifically, in the Understandability phase, we include, in the LLM prompt, the source code of the methods used in the test to be improved.

To assess the efficacy of UTGenCov, we set up an experiment where:

- we compared the coverage of EvoSuite to UTGen's Understandability phase run in isolation and to the new UTGenCov approach.
- we disabled Test Data and Test Name in UTGenCov for this experiment.

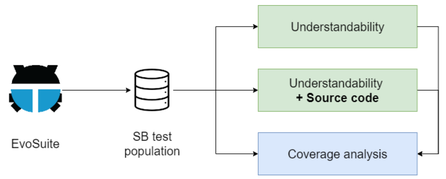

Figure 5: Experiment setup for comparing UTGen's Understandability phase with the improved UTGenCov approach, using the same test population as the foundation.

6. UTGENCOV RESULTS

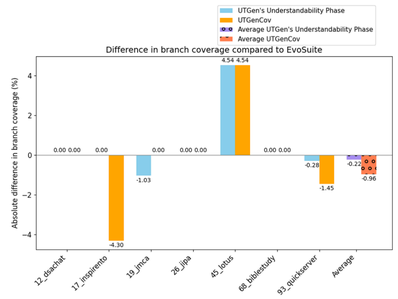

Figure 6: Branch coverage difference between UTGen's Understandability phase and UTGenCov compared to EvoSuite for all projects in the experiment.

Project	Run type	% tests with no comments added	% tests rolled back to EvoSuite
12_dsachat	Understandability	0%	100%
	UTGenCov	0%	100%
17_inspirento	Understandability	5%	8%
	UTGenCov	11%	10%
19_jmca	Understandability	0%	25%
	UTGenCov	25%	25%
26_jipa	Understandability	46%	2%
	UTGenCov	46%	2%
45_lotus	Understandability	67%	0%
	UTGenCov	67%	0%
68_biblestudy	Understandability	100%	0%
	UTGenCov	100%	0%
93_quickserver	Understandability	61%	0%
	UTGenCov	17%	3%

Table 1: Percentage of tests with no comments added and rolled back to EvoSuite for each approach.

Projects 12, 26, 68: Source code addition had no impact on tests or coverage.

- Project 12: The test got rolled back in both approaches as it failed to compile.
- Project 26: Identical test suites in UTGen and UTGenCov.
- Project 68: All 17 tests concluded without added comments in both approaches.

Project 45:

- Coverage increase in UTGen and UTGenCov tests compared to EvoSuite.
- One line removed led to an extra edge case being covered. For this reason, it was excluded from the average in Figure 6.

Project 19:

- UTGen coverage decreased by 1%, but for UTGenCov, it was the same as EvoSuite's.
- UTGenCov had one extra test without added comments, which might have prevented the coverage decrease seen in UTGen.

Projects 17 and 93: Source code addition resulted in lower coverage.

- Project 93: UTGen had 61% of tests without comments, while UTGenCov had 17%.

7. DISCUSSION

Limitations

- The research was constrained by a limited time frame and resource availability. These constraints could impact the findings, potentially not providing a sufficiently comprehensive overview.
- The LLM is nondeterministic, meaning that experiment outputs may be slightly skewed. However, nondeterminism is necessary because of the improper output format that the LLM sometimes generates.

Future Work

- Future work should include experiments with a broader scope, that involve more projects to strengthen results and uncover new insights.
- An analysis of the impact on coverage using different LLMs, such as Code Llama 70b or Codestral, should be considered. Larger LLMs could reduce hallucinations and produce more understandable tests than Code Llama 7b Instruct.

8. CONCLUSIONS

RQ1:

- The Phase isolation study reveals that the Understandability phase, which replaces the entire test body, causes the most significant difference in coverage.
- From the Manual inspection, the biggest factor negatively impacting coverage is the removal of method calls or tests altogether.

RQ2:

- The experiment showed mixed results, with source code addition having no impact on several projects and reduced coverage in others compared to UTGen.
- Although UTGenCov improved test understandability more frequently, it resulted in an average of 0.74% lower branch coverage than UTGen.

REFERENCES

G. Fraser and A. Arcuri, "Evosuite: automatic test suite generation for object-oriented software," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 416–419. [2] ——, "A large-scale evaluation of automated unit test generation using EvoSuite," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 24, no. 2, pp. 1–42, 2014. [3] N. Alshahwan, J. Chheda, A. Finegenova, B. Gokkaya, M. Harman, I. Harper, A. Marginean, S. Sengupta, and E. Wang, "Automated unit test improvement using large language models at meta," *arXiv preprint arXiv:2402.09171*, 2024. [4] A. M. Dakhel, A. Nikanjam, V. Majdinasab, F. Khomh, and M. C. Desmarais, "Effective test generation using pre-trained large language models and mutation testing," *Information and Software Technology*, vol. 171, p. 107468, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584924000739 [5] A. Deljouyi, "Understandable test generation through capture/replay and LLMs (ICSE 2024 - doctoral symposium) - ICSE 2024." [6] EvoSuite, "SF110 Corpus of Classes | EvoSuite." https://www.evosuite.org/experimental-data/sf110/ [7] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin et al., "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023. [7] A. Addlesee, "Grounding LLMs to in-prompt instructions: Reducing hallucinations caused by static pretraining knowledge," in *Proceedings of Safety4ConvAI: The Third Workshop on Safety for Conversational AI@ LREC-COLING 2024*, pp. 1-7. 2024.