Enhancing Readability of Automatically Generated Unit Tests: Leveraging LLMs for Meaningful Identifier Naming

Jason liu
Master Student Software Engineering
University of Antwerp
Antwerp, Belgium
jason.liu@student.uantwerpen.be

***Abstract*—Automatically generated unit tests often use generic or meaningless identifiers, which makes test code hard to read and understand. When test identifiers lack semantic clarity, maintenance suffers and developers spend extra effort interpreting intent rather than focusing on behavior. We show that a Large Language Model, when properly instructed, can generate meaningful and consistent test method names and local variable names and is consistent across tests—and outperform existing ML-based naming heuristics. Others have tried to do this with machine learning, rule based algorithms and others, meaning that it is indeed a relevant problem that others want to tackle.**

***Index Terms*—LLM, Testing, Readability, Identifiers, Code refactor, Unit test, Automated test generation**

I. Introduction

Naming identifiers is a crucial aspect of the readability and maintainability of codebases. Additionally, automated testing has become increasingly popular due to its high return on investment (ROI) compared to the effort required by a tester to implement a test [1].

More specifically what we mean by identifiers, are test method names and local variables used for that specific test method. Both categories are essential for readability: method names provide an overview of intent, while variable names support clarity within the test logic.

In the domain of automated testing, there is a subcategory known as automated test generation, which aims to reduce the manual effort of writing tests by automatically generating them for a given application. However, this approach presents a significant challenge: it often conflicts with code readability and maintainability. Automatically generated tests typically use simplistic or non-meaningful identifiers, which can hinder understanding and long-term maintenance.

```
import org.junit.FixMethodOrder;
import org.junit.Test;
import org.junit.runners.MethodSorters;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class RegressionTest0 {

    public static boolean debug = false;

    @Test
    public void test001() throws Throwable {
        if (debug)
            System.out.format("%n%s%n", "RegressionTest0.test001");
        java.lang.Object obj0 = new java.lang.Object();
        java.lang.Class<?> wildcardClass1 = obj0.getClass();
        org.junit.Assert.assertNotNull(wildcardClass1);
    }

    @Test
    public void test002() throws Throwable {
        if (debug)
            System.out.format("%n%s%n", "RegressionTest0.test002");
        Message message1 = new Message("");
        java.lang.Class<?> wildcardClass2 = message1.getClass();
        org.junit.Assert.assertNotNull(wildcardClass2);
    }

    @Test
    public void test003() throws Throwable {
        if (debug)
            System.out.format("%n%s%n", "RegressionTest0.test003");
        Message message1 = new Message("");
        java.lang.String str2 = message1.printMessage();
        java.lang.String str3 = message1.printMessage();
        java.lang.Class<?> wildcardClass4 = message1.getClass();
        org.junit.Assert.assertEquals("'" + str2 + "' != '" + "" + "'
        org.junit.Assert.assertEquals("'" + str3 + "' != '" + "" + "'
```

Fig. 1: Randoop example [2]

This leads to additional problems. For new software developers, tests are often an easy way to understand the codebase by simply reading them. For long-time team members, the meaning of tests may gradually fade over time, making the tests less useful when errors occur—because we no longer clearly understand what the tests are meant to verify [3].

Well-defined identifiers have a significant impact on code quality and future development. However, determining what constitutes a well-defined identifier is not a trivial task. Even for humans, it can be difficult to decide whether an identifier is appropriately named. One developer might assign a name with a specific intent, while another might interpret it differently. This is why consistent coding conventions are crucial for maintainability and readability [4]. This is also where large language models (LLMs) have a major advantage: being trained on vast datasets allows them to adopt and follow a consistent naming convention, leading to more uniform and predictable interpretations [5].

This work focuses on improving the naming of identifiers in unit tests, generated through automated test generation, by leveraging large language models.

II. Working title

"Enhancing Readability of Automatically Generated Unit Tests: Leveraging LLMs for Meaningful Identifier Naming"

III. Research project 1 / Research project 2 / Thesis

This research will be conducted as a **Master Thesis**.

IV. The problem

In this project, we will explore how to leverage the power of Large Language Models (LLMs) to assign meaningful names to local variables and test method names generated by automated test code. Automated test generation typically focuses on producing unit tests based on predefined requirements, often without adhering to proper code conventions. As a result, the generated tests may lack readability and be difficult for developers to understand. To address this issue, we will utilize LLMs trained to follow specific code conventions that we define, enabling them to generate well-structured and semantically meaningful identifiers.

A. Research questions?

Some research questions that might be interesting to keep in mind are as follows:

1) Does LLM perform better than existing ML renaming algorithms?
   - To assess the effectiveness of LLMs, a comparative analysis with existing machine learning techniques is necessary. Evaluating multiple dimensions of performance will allow us to determine whether the use of LLMs represents a meaningful improvement or an unjustified effort.
   - How can this be achieved: We will be benchmarking both my new developed method against existing methods like the one mentioned in Arno thesis [6]. Based on different metrics we will see if one outperforms the other. Metrics can be overall readability, complexity, exact match (F-match), and others. Additionally this benchmark is improved with custom dataset, that is made by assumptions of anecdotal evidence.
2) How do different prompt engineering strategies affect the quality and consistency of identifier name suggestions generated by LLMs?
   - Working with LLMs allows us to experiment with different prompts to achieve varying results. To ensure we obtain the most optimal outcomes, it is essential to evaluate multiple prompt strategies and validate which one performs best. This process helps confirm whether our chosen approach is indeed the most effective.
   - How can this be achieved: Here we will mainly look at the outputs of different prompts and see what differs. We will look if the code is semantically still the same, error less, and which provides more meaning. Combining this with existing prompt engineering, we can easily find what parts of the prompt affect what.
3) How does fine-tuning an LLM on test-specific code (e.g., unit tests from GitHub) affect naming quality compared to using a general-purpose code LLM?
   - Determining whether fine-tuning is required is critical, as it affects both performance and project cost. This evaluation helps managers and industry leaders make informed decisions when considering the adoption of LLM-based solutions.
   - How can this be achieved: In this section, we examine two different models and, as in Question 1, we will benchmark them against each other. Additionally, we will document the extra time and resources required to develop each model, in order to account for the associated cost and effort.

V. Why is this a relevant / challenging problem?

As previously discussed, code quality is a fundamental aspect of reliable and maintainable software systems [7]. This issue becomes even more significant with the rise of automated test generation, which, despite its benefits, often produces test cases that are difficult to interpret due to the absence of meaningful and consistent identifiers. As a result, the potential value of these tests can be diminished, particularly in collaborative and long-term development environments.

A key difficulty in this context stems from the large number of test cases typically generated by automated testing tools, making it a non-trivial task for developers to interpret and maintain them. To mitigate this, various automated naming algorithms have been proposed to improve readability. Yet, these methods produce suboptimal results [6], [8], [9].

As a potential solution, we propose the use of Large Language Models (LLMs) to generate more meaningful and human-readable identifiers. However, this introduces additional complexity, as LLMs are not inherently optimized for this task. Therefore, careful model selection and configuration tuning are essential to realize their full potential in this domain.

VI. Why are you the one to solve it?

I am personally a strong advocate for code quality. This is one of the reasons why I enjoy working with software testing, as it naturally encourages developers to write cleaner, more maintainable code. However, when test code itself fails to meet standards of code quality—particularly in terms of readability and maintainability—it undermines the very goal it aims to support.

Another key motivation for this project is my passion for automation. I believe that automating improvements in code quality is not only efficient but also essential in modern software development practices.

Finally, I have a deep interest in Large Language Models (LLMs). I actively follow the latest developments in this field and possess a solid understanding of the capabilities and limitations of current-generation LLMs. This makes the intersection of code quality, automation, and LLMs an ideal and exciting area for me to explore.

VII. How will I solve it? In what direction?

To tackle this challenge, we propose the use of Large Language Models (LLMs) to generate more descriptive and human-readable identifier names in test code. Our methodology builds on existing literature in the field of identifier renaming within general codebases, which we will tailor and apply specifically to the domain of automated test generation.

This thesis will have quantitative and qualitative comparisons. Where we compare with anecdotal evidence and previous solutions.

VIII. Rough plan

This is a preliminary outline of the steps I intend to follow to complete my thesis in a structured and effective manner. These steps may be adjusted or expanded as the work progresses.

1) Research Fundamentals: We will begin by conducting foundational research on the various aspects of the problem. This includes reviewing related work, particularly studies focused on identifier renaming in general codebases. By understanding existing approaches and methodologies, we can identify strategies that may be applicable or adaptable to test code.
2) Benchmark Setup: Before any implementation begins, we will establish a benchmark framework. This allows us to have a point of comparison, so that when we generate results later on, we can directly assess whether our approach performs better, worse, or comparably to existing methods.
3) Solution Exploration: In this phase, we will brainstorm and evaluate potential solutions. This involves analyzing which LLM to use as a baseline, identifying any existing tools or libraries that can assist us, selecting appropriate datasets, and determining how all these components can be effectively integrated. Planning and prioritization will also be key—deciding what is feasible within the scope and timeframe of the project.
4) Implementation: We will implement the proposed solution(s) based on our earlier planning. This includes integrating the selected model, applying it to test code, and developing any additional tooling required to support the pipeline. This matches the phase of LLM base fine-tuning where we start fine tuning our LLM based on the solutions we found. During these iterations we also try to improve our dataset, this is done by finding anecdotal evidence. Based on these evidences we found, we can categories our dataset in easy, medium and hard cases.
5) Evaluation and Conclusion: In the final phase, we will apply the predefined metrics and use the established benchmark to evaluate the effectiveness of our approach. This comparison will help determine whether the LLM-based identifier naming provides a measurable improvement and offer insights into its practical value. While this work includes a comparative evaluation using quantitative metrics (e.g., readability, complexity, F-match), it also goes beyond numbers.

A. GANTT CHART

A preliminary Gantt chart is proposed to visualize the distribution of time throughout the year and to highlight the phases where the most effort will be concentrated. The Gantt chart will be based on an incremental development approach, similar to sprints. Where each week counts as one iteration, where we have a meeting with the supervisor each week.

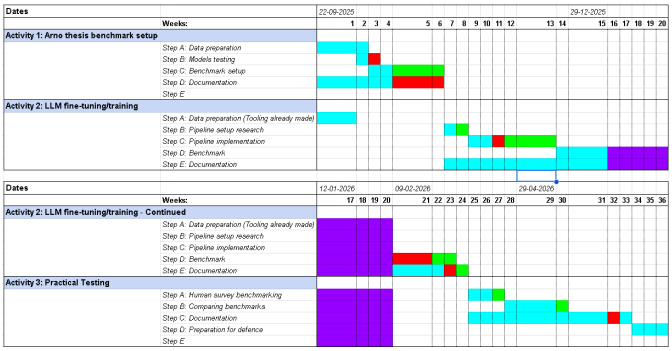

Fig. 2: Gantt chart

Here a small legend, decoding what each color means:

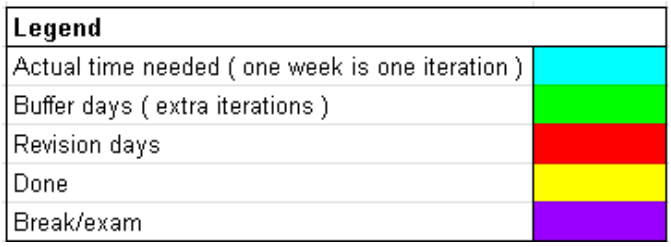

Fig. 3: Gantt chart legend

While the rough plan is a bit different compared to the Gantt chart, the Gantt chart will mainly show the main activities. The rough plan is a more general description of how everything will follow.

References

[1] S. Palamarchuk, "The True ROI of Test Automation." [Online]. Available: https://abstracta.us/blog/test-automation/true-roi-test-automation/

[2] GeeksforGeeks, "Generate JUnit Test Cases Using Randoop API in Java." [Online]. Available: https://www.geeksforgeeks.org/java/generate-junit-test-cases-using-randoop-api-in-java/

[3] Test Smell Catalog, "Bad Naming — Violating Coding Best Practices." [Online]. Available: https://test-smell-catalog.readthedocs.io/en/latest/Code%20related/Violating%20coding%20best%20practices/Bad%20Naming.html

[4] M. Cechetto, "Why Naming is Hard in Programming and Why LLMs Struggle Write Real Code." [Online]. Available: https://dev.to/mateuscechetto/why-naming-is-hard-in-programming-and-why-llms-struggle-write-real-code-3bee

[5] A. S. Molison, M. Moraes, G. Melo, F. Santos, and W. K. G. Assunção, “Is LLM-Generated Code More Maintainable & Reliable than Human-Written Code?.” Aug. 2025.
[6] A. D. Keersmaeker, “Enhancing Test Code Understandability with Machine Learning-Based Identifier Naming.” Antwerp, Belgium, Jun. 2024.
[7] SonarSource, “What is Code Quality? Definition Guide.” [Online]. Available: https://www.sonarsource.com/learn/code-quality/
[8] A. Mastropaolo, E. Aghajani, L. Pascarella, and G. Bavota, “Automated Variable Renaming: Are We There Yet?,” *Empirical Software Engineering*, vol. 28, no. 2, pp. 385–400, Feb. 2023, doi: 10.1007/s10664-022-10274-8.
[9] B. Lin, C. Nagy, G. Bavota, and M. Lanza, “On the Impact of Refactoring Operations on Code Naturalness,” in *Proceedings of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE Computer Society, 2019, pp. 594–598.