Cassiopeia: A Retrieval-Augmented Generation Tool for Automated Test-Case Creation

Flávia Camila Oliveira
Sidia Institute of Science and Technology
Manaus, Brazil
flavia.oliveira@sidia.com

Leonardo Tiago
Sidia Institute of Science and Technology
Manaus, Brazil
leonardo.albuquerque@sidia.com

Lennon Correa Chaves
Sidia Institute of Science and Technology
Manaus, Brazil
lennon.chaves@sidia.com

Abstract

This study, conducted within a software testing team that performs exploratory and functional testing, evaluates the feasibility of using artificial intelligence to automatically update and create test cases, thereby reducing the workload of the test case management. We built **Cassiopeia**, a tool that parses testers' task submissions, formulates prompts for the large language model `gpt-oss-120b`, and supplies the test-case dataset context via Retrieval Augmented Generation (RAG) using the FAISS library for similarity search. In an experiment, 45 real change requests were processed; 16 generated test cases were independently rated by three testers on a 1–5 quality scale. Cassiopeia achieved 95.35% recall for test-case IDs, 53.32% BLEU-4, and 56.89% ROUGE-1. Overall, 58% of ratings indicated partial or full quality, 31% expressed disagreement, and 11% were uncertain. Results suggest the tool is promising but still requires complete task descriptions and human review before deployment.

Keywords

Test Case Generator, RAG, LLM, Software Test, Requirements

1 Introduction

A test team from a software institute which is responsible for executing functional tests and exploratory tests possesses a set of test cases that needs to be modified constantly due to frequent changes in the associated requirements [1, 4]. Some of the members of the test team are responsible for managing the test cases, creating and updating new test cases based on test case change tasks, which are created via the Jira [1] system by other members of the team when a requirement is updated or a new requirement applicable to the team's test scopes is received [3]. This way, it is guaranteed that the test case will cover the requirement and the validation will be performed adequately. After the test case is created or updated, another member of the team is responsible for reviewing it, and only after this step the test case is made available for the team.

However, the whole process of creating and updating a test case is performed manually, and it becomes costly to do because it is competing with the high demand of test requests that the team handles as well. It is also an activity that needs to be executed quickly due to it being critical, as if the requirements are not adequately covered by the test cases, this may lead to issues and costs that could have been prematurely avoided [2].

Our approach leverages Large Language Models (LLMs) together with Retrieval-Augmented Generation (RAG) to enhance answer accuracy by grounding generation in reliable indexed sources and reducing hallucinations. FAISS[2] is employed to efficiently retrieve the most relevant contextual vectors, allowing quick identification of the test case linked to a change task when no explicit ID is provided. Built on this foundation, the Cassiopeia tool supports test-case management by analyzing testers' change requests and automatically updating the associated test case or creating a new one. It uses the open-source gpt-oss-120b model (Apache 2.0) and incorporates the team's full test-case repository as RAG context, with FAISS handling the similarity search.

[1] https://www.atlassian.com/software/jira
[2] https://ai.meta.com/tools/faiss/

2 Test Case Management

In this test team, when a tester discovers that the steps for a feature have changed or a new requirement arises, they create a Jira task to update an existing test case or to add a new one. Five team members handle test case management, alternating between analyst and reviewer roles. The analyst first validates the task, confirming the necessity and supporting evidence for updates or creations, then performs the required change. Afterward, a second team member acts as reviewer, checking that the implementation fully satisfies the task's specifications. If the reviewer approves, the revised or new test case is deployed to production, becoming available for execution in subsequent testing cycles.

There are two types of test cases that are maintained in the management activity: exploratory test cases and functional test cases. Both types share a similar structure with two sections: "Details" and "Test Script". The section that is the same for both types of test cases is Details, which is described below:

- **Test ID:** Contains the test case's unique ID code, composed by letters and numbers. For example: "T-123" refers to test case number 123.
- **Title:** Contains a phrase that summarizes the main goal of the test case. For example: "Verify the Bluetooth feature".
- **Objective:** Contains a detailed description of what must be validated in the test case and what is expected when it is executed. For example: "Perform this test to verify if the Bluetooth feature can be used to pair the device with other compatible devices".
- **Preconditions:** Contains any conditions that need to be met before the test case execution begins. For example: "Have a compatible support device available to test the Bluetooth connection with the device under test".

The main difference between them is in the Test Script section, where the actions that the tester must take during the test execution are described. For functional tests, the structure of the Test Script follows a step-by-step format, with each step containing the following fields:

- **Step:** Contains the next action to be taken by the tester. For example: "In the Connections screen, tap the Bluetooth button".
- **Test Data:** Contains additional information to aid the tester. For example, it might contain a screenshot of the screen in which the tester is expected to take the action described in "Step".
- **Expected Result:** Contains the expected result for the action indicated in "Step". For example: "When the Bluetooth button is tapped, the Bluetooth Connections screen must be displayed".

The tester must follow all the steps of the script until they end, and if there are not any unexpected results the test is considered a success. For exploratory tests, the structure of the Test Script section follows a plain text format, and contains the following fields:

- **Heuristics:** Contains a list of heuristics that must be used by the tester as a guide during the exploration of the feature associated with the test case.
- **Known issues:** Contains a table with the identifiers of the bug reports that were registered during prior executions of the test, as well as the title of the report and a brief description of how to reproduce the behavior that caused the issue, so that the tester has an idea of where they could possibly find bugs.

Therefore, for the current version of the Cassiopeia tool, when it receives a task associated to an exploratory test case, the tool searches for the ID of the bug report in the task's body and outputs to the user that that specific report must be added to the test case.

3 Test Case Generation

To develop the Cassiopeia tool we used Python 3.12.4[3] and the model gpt-oss-120b as a basis, and to provide the context of the test case dataset for the model, we employed the RAG technique. We used the FAISS library to perform queries within the context because it was designed for similarity searches and because its code is open source, and in this case the goal was to use it to find an adequate test case to associate to a task when it did not possess a specific associated test case.

The tool's workflow is illustrated in Figure 1 and described below:

- **Creates dataset:** The dataset is built from the test cases and test case change tasks, organized in a spreadsheet.
- **Parsing:** The tool analyses the task to determine if it is associated to a functional or exploratory test case.

The workflow can follow different paths depending on the result of the "Parsing" step. If it is determined that the task is for an exploratory test case, the following steps are performed:

- **Extracts the report ID**: The tool searches for the ID of the bug report that is supposed to be added to the test case in the body of the task.
- **Returns the instruction**: The tool returns an instruction that tells the tester that the identified bug report must be added to the "Known issues" section of the test case.

If it is determined in the "Parsing" step that the task is for a functional test case, the following steps are followed:

- **Chunking:** The tool converts the task case in chunks, dividing them in smaller pieces of text that are easier to manage.
- **Indexing**: Embedding vectors are generated for each chunk.
- **Semantic search:** The FAISS library is used to find the most similar test cases.
- **Prompt**: The tool builds the prompt for the LLM containing general instructions, the text from the test case change task and the test case that will be changed.
- **Request to LLM:** The request to the LLM is made by sending the prompt to the API of the configured LLM.
- **Post-Processing**: The model returns the generated test case in the JSON format, including indications of the changes made to the original test case.

The last step of the tool's workflow is the same regardless of the type of test case associated to the submitted task:

- **Grouping:** Lastly, all test cases generated based on the tasks that were used as input for the tool are saved in a spreadsheet.

The prompt was divided in two parts: the first part contains a definition of the LLM's role so as to direct how the model must act and possibly refine the resulting answer; the second part contains the specific instructions of the activity, and it also includes the text from the task, the test case that will be changed, clear instructions for each expected field that must be part of the generated test case, as well as an example of a valid answer.

4 Methodology

We executed the following steps to evaluate Cassiopeia Tool:

- **Data preparation**
  - Extracted 45 test case change tasks from Jira (XML file) and a dataset of 135 test cases to serve as RAG context.
  - The model generated updated versions of the test cases for each task; an independent analyst manually updated the same 45 cases, creating a gold-standard reference set.
- **Metrics selected**
  - **Recall** – measures the tool's ability to correctly identify test-case IDs.
  - **BLEU-4** – evaluates 4-gram precision (with brevity penalty) by comparing the generated text to the human-written reference.
  - **ROUGE-1** – measures unigram overlap, which is more appropriate for NLP tasks where many wording variants are acceptable.
- **Participants**
  - Conducted an empirical study with 3 testers responsible for test case management (the sub-team has 5 members; the two developers of Cassiopeia were excluded).
- **Instruments**
  - Selected 15 distinct tasks (including one that splits a test case into two).
  - Created an evaluation sheet (5-point Likert scale for the question "Was the test case generated with quality?" plus a free-text justification field) and a text document containing all generated test cases.
  - Designed a post-experiment survey to capture overall impressions, main problems, and suggestions for improvement.
- **Experiment execution**

[3] https://www.python.org/

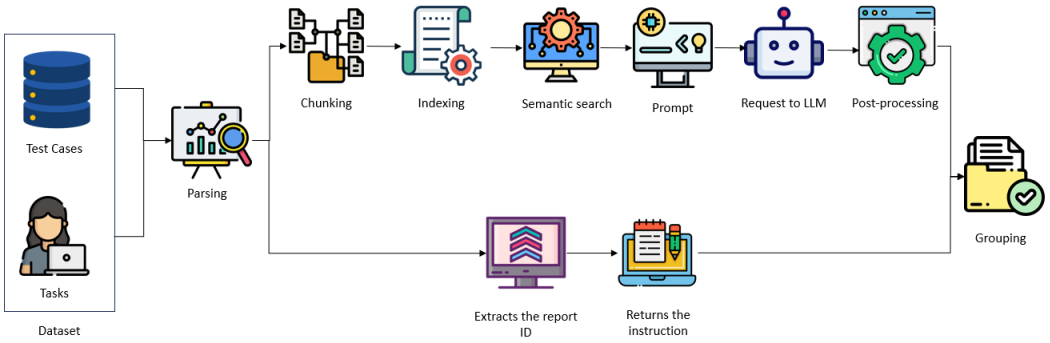

Figure 1: Workflow of the tool to generate test cases

  - Presented the study goals and instructions in a meeting room; participants evaluated the cases using the sheet and the document while the facilitator answered any questions.
  - After completing the evaluation, participants filled out the survey.
- **Result analysis**
  - Computed Recall, BLEU-4, and ROUGE-1 for the 45 tasks, using the analyst-produced gold standard as reference.
  - Summed the Likert scores per participant to gauge agreement on the quality of the generated cases.
  - Performed qualitative analysis of the survey feedback to understand user perceptions and identify improvement opportunities.

5 Results Summary

Quantitative Analysis: Using the manually-curated gold-standard set of 45 updated test cases as reference, the Cassiopeia tool achieved a Recall of **95.35%**, demonstrating that it correctly identified the target test-case IDs in almost all instances. However, the textual similarity metrics were modest: the mean **BLEU-4** score was **53.32%** and the mean **ROUGE-1** score was **56.89%**, indicating that less than two-thirds of the n-gram content matched the human-written versions. While these figures suggest room for improvement in linguistic fidelity, the high Recall confirms that the tool reliably retrieves the correct cases for modification.

Qualitative Analysis: Across 48 Likert evaluations from three participants, **28 scores (58%)** were "Agree" or "Fully agree," whereas **15 scores (31%)** were "Disagree" or "Fully disagree." Comments highlighted three recurring issues: (1) incomplete or shallow task descriptions lead to missing or incorrect requirements; (2) the system sometimes inserts redundant or unnecessary steps, especially when splitting a test case; and (3) despite generally adhering to the team's standard format, the generated cases still require careful human review. All participants agreed that additional contextual information (e.g., clearer task wording or supporting screenshots) is needed, reinforcing the finding that human validation remains essential before the generated test cases can be deployed.

6 Conclusion

In this study, we discussed the importance of the use of test cases in the software development process, approached the difficulty of performing maintenance on test cases manually when there are many test cases and when their associated requirements are constantly changing, and presented our solution to this problem. We can generate updates to test cases or create entirely new test cases via an LLM-based tool that receives test case change tasks through prompts created using RAG. For future works, it is suggested to use other LLMs, add more information from the test team's context to obtain more complete test cases and to integrate the tool with the system that the team uses for maintaining their test cases.

Artifact Availability

Owing to the confidentiality of the project data and the stipulations of the Non-Disclosure Agreement (NDA), the artifacts and data sets used in this study cannot be made publicly available.

Acknowledgements

This paper is a result of the Research, Development & Innovation Project (ASTRO) performed at Sidia Institute of Science and Technology sponsored by Samsung Eletrônica da Amazônia Ltda., using resources under terms of Federal Law No. 8.387/1991, by having its disclosure and publicity in accordance with art. 39 of Decree No. 10.521/2020.

References

[1] Lennon Chaves, Flavia Oliveira, Leonardo Tiago, and Renata Castro. 2024. Benefits and Challenges of a Physical Device Farm for Automated Software Testing: A Case Study of the STELA Tool Implementation. In *Anais do IX Simpósio Brasileiro de Testes de Software Sistemático e Automatizado* (Curitiba/PR). SBC, Porto Alegre, RS, Brasil, 89–91. doi:10.5753/sast.2024.3880

[2] Flávia Oliveira, Alay Nascimento, Ana Silva, Leonardo Tiago, and Lennon Chaves. 2025. Analysis of Contributions at a Software Institute through the Introduction of a Pre-Trained Model for Requirements Classification. In *Anais do XXIV Simpósio Brasileiro de Qualidade de Software* (São José dos Campos/SP). SBC, Porto Alegre, RS, Brasil, 356–364. doi:10.5753/sbqs.2025.13598

[3] Flávia Oliveira, Ana Silva, Leonardo Tiago, and Lennon Chaves. 2025. Ensuring Test Case Coverage through Multilabel Requirements Classification: A Feasibility Study with Pre-Trained Models. In *Anais do X Simpósio Brasileiro de Testes de Software Sistemático e Automatizado* (Recife/PE). SBC, Porto Alegre, RS, Brasil, 1–9. doi:10.5753/sast.2025.13597

[4] Flávia Oliveira, Leonardo Tiago, and Lennon Chaves. 2025. The Influence of Using LLMs on the Activities of a Software Testing Team: An Industry Case. In *Anais do X Simpósio Brasileiro de Testes de Software Sistemático e Automatizado* (Recife/PE). SBC, Porto Alegre, RS, Brasil, 144–146. doi:10.5753/sast.2025.13592