Software Requirements Specification for Gen AI Test Bot

Version 2

Group 2

Neha Asthana - Project Manager
Lawrence Gu - Software Developer
Nathan Peereboom - Software Developer
Bahar Abbasi Delvand - UI/UX designer
Krishika Mirpuri - Software Architect

April 1, 2024

Table of contents and Contributions

Name	Contribution
Neha Asthana	Formatting, List of abbreviations and notations, naming conventions and definitions,Table of contents and contributions, The Purpose of the Project
Krishika Mirpuri	Functional Requirements, Risks and Issues, formatting
Bahar Abbasi Delvand	Identifying the clients and stakeholders, determining the project constraints, formatting
Lawrence Gu	Data and Metrics, non-functional requirements, formatting
Nathan Peereboom	Non-functional requirements, Client/Stakeholder information, document formatting

CONTENTS

Revision History

Version #	Description	Date
0	Original SRS document	2023/10/12
1	Modified requirements/components to fit altered plan	2024/01/26
2	Final version	2024/04/01

1 The Purpose of the Project

1.1 The User Background/ Background of project effort

The genAI test BOT aims to address challenges in software development which are encountered while creating test suites and code documentation.

The project's background is rooted in the desire to provide the developers and other stakeholders with an automated solution. As a result, project managers will enjoy improved code quality, faster development cycles, and streamlined project management. Furthermore, quality assurance teams will have access to comprehensive, automatically generated test suites, simplifying testing efforts. Lastly, end users will benefit from higher-quality software with fewer defects, ensuring a more reliable user experience.

1.2 Goals of the project

The primary goals of the Gen AI Test Bot are:

Automation: The aim is to create a fully integrated bot which will automate the process of generating accurate test suites and code documentation for code which is submitted through built in GitHub APIs in the Visual Studio Code environment. This will reduce the need for manual work which will make it more efficient.

Accuracy: Ensure that the test suites and the documentation created have an accuracy rating and coverage data. This will help in maintaining the quality of the code and provide reliability.

Integration: The bot will be integrated into the CI/CD (Continuous Integration/Continuous Flow) Pipeline and Git workflows. This will increase efficiency and will facilitate a smooth development process.

Feedback Mechanism: Implementation of a feedback mechanism that will allow for the evaluation and improvement of the generated test suites.

2 Naming Conventions and Terminology

2.1 Abbreviations

- **AI** - Artificial Intelligence
- **API -** Application Programming Interface
- **LLM** - Large Language Model
- **AST** - Abstract Syntax Tree
- **DOM** - Document Object Model
- **ASCII** - American Standard Code for Information Interchange
- **UI/UX** - User Interface / User Experience
- **NL** - Natural Language

2.2 Terminology

- **Differential Parsing** - a technique used to efficiently modify or update a parsed data structure when only a small portion of data has changed. The approach works by initially parsing the entire input and representing it in an AST, a DOM, or another data structure. From that point onwards, any changes to the data will be identified by the parts of the data structure that have been modified.

- **File Parsing** - the process of analyzing and interpreting the contents of a file to extract meaningful information. The process works by taking in a file, reading it using a software tool, breaking down the file into smaller units called tokens and lexically analyzing the tokens to understand their type and structure. The information is then parsed and extracted for use in the system.

- **Compile Rate** - The use of the phrase compile rate in our document refers to the number of test cases generated by our bot that successfully compile (without human intervention), out of the total number of generated test cases.

- **Natural Language Descriptions** - These are textual explanations or interpretations of concepts, data or information written in a human-readable language.

3 Clients and Stakeholders

3.1 Client

N/A

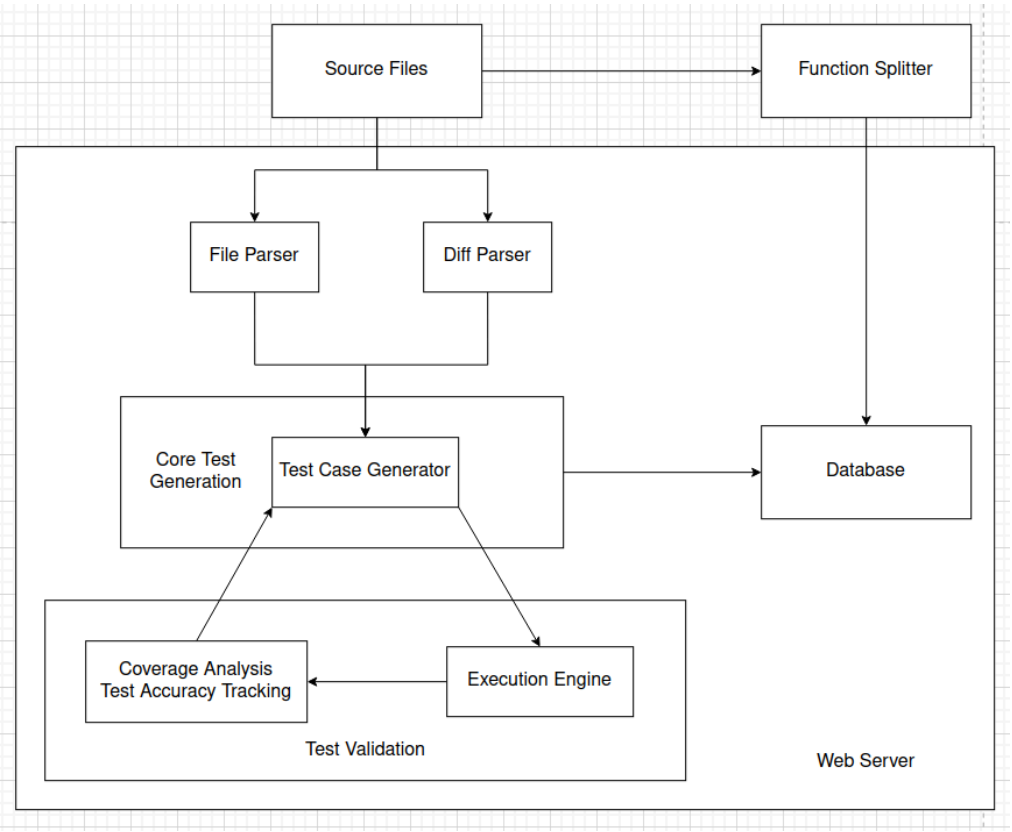
3.2 Stakeholders

Our primary stakeholders are software developers, as this system will be used by developers to help with automated testing and debugging for Python programs. Secondary stakeholders include QA engineers, test engineers & the open source community who will also benefit from this application.

4 Project Constraints

- The project is designed to leverage the capabilities of GitLab and GitHub for version control, collaboration, and code management.
- Both OpenAI's advanced GPT-4 and well-established GPT-3.5 models are an integral part of the project's implementation, showcasing a holistic approach that utilizes the capabilities and features of these state-of-the-art natural language processing technologies.
- The project supports the Python programming language with two well-known tools: Pytest and Pylint.
- The project will use Javascript, React and CSS to create the user interface.
- The product is accessed through a web-based platform and a VSCode extension.
- Considering that LLM imposes a charge for each token usage, we need to optimize our token requests to minimize costs, all the while remaining mindful about the potential cost constraints we may encounter.

5 Functional Diagram



6 Functional Requirements

6.1 User Roles and Permissions

- Due to the uniform approach we are adopting, all users will have the same level of access and permission across all aspects of our codebase.
- Users are not restricted or segmented by sections or components of our code. They have unrestricted access to every part of our codebase.
- Uniform permissions reduce administrative complexity and enable users to work with agility and efficiency.

6.2 System Overview

System Components:

- The system consists of 5 main parts: core test generation module, web server, VSCode extension client, User Interface/Website and test validation module.

Core test generation module

- The core test generation module takes as input a diff of a code commit and generates a unit test case for it using a series of LLM prompts.

Web Server

- The web server interacts with the client to implement the user flow around our test generation module.

- Our current test generation flow requires first explicitly generating a list of assertions/test conditions that the generated test case needs to satisfy.
- The web server is also connected to a database.

VSCode Extension Client

- The VSCode Extension Client acts as a bridge between the user and the server, facilitating the initialization of Websocket connections, detecting Git workspace openings, interacting with the server to mirror repositories, sync commits, generate tests, reporting errors in a user friendly way, and handle the UI aspects of authentication and documentation.

Test Validation Module

- The Test Validation Module executes tests on test cases produced by the core test generator, validating their compliance with coverage, accuracy, coding standards (static analysis). It evaluates whether these methods are covered by the generated test cases through coverage analysis, static code analysis, equivalent partitioning and test accuracy tracking.

Website/User Interface

- The UI is connected to the server and the database, and will receive the statistics of the codes run on the main page of each user.
- The user will also be able to register and login using the UI.

6.3 Accuracy and Reliability

- The user will be provided with all the statistics (pass/fail history for each test case, coverage analysis, and static code analysis) the application obtained by executing the generated test suite. They can choose to generate new tests or edit existing tests to improve accuracy.

6.4 Programming Language Support

- The project will be coded in Python.
- The UI will be coded in React and JavaScript.
- The project will use OpenAI API’s to access pre-trained LLMs.

6.5 Testing Methods

- The bot will generate a detailed accuracy and coverage report after each run.

6.6 Error Handling

- Errors present in the source files cause the generation of test cases to fail upon execution, as expected.
- This will allow the system to inform the user of which test case failed along with the detailed evaluation process, and this will be displayed on the interface.
- Users can click to regenerate new test cases after correcting errors in the codebase, or if they are not satisfied with the result.
- On the UI, users will receive an error message displayed in a snackbox at the bottom of the page if they leave any text boxes unfilled, input an incorrect email

format, attempt to register with an email that's already in use, enter the wrong password for login, or if their passwords don't match during registration.

6.7 Integration

- The bot will use built-in GitHub APIs in Visual Studio Code to pull the user's linked github repositories and listen for commits. The bot will run this function on startup, so the latest code gets pulled from a connected repository to run test cases on. If no GitHub repository is connected to the Visual Studio Code Workspace, this section of the bot will be skipped and other functions will continue to run on files that are present in the workspace.
- The user can only choose to connect the application to public Github repositories at this time, not private repositories.
- Test cases and assertions will only be generated and run on the code file currently being viewed in the Visual Studio Code Workspace.

7 Data and Metrics

7.1 Training Data

- We will not train the LLM from scratch, rather we would utilize pre-trained models from OpenAI, leveraging their extensive training data and sophisticated model architectures.

7.2 Dataset

- For the practical application of our project,we will use one of our pre-established code repositories. This will allow us to evaluate the bot in real-world coding environments, thereby enhancing the reliability of our application.

7.3 Performance metrics

- **Compile rate.** This will be determined by the ratio of test cases that are successfully executed without any discrepancies in a Python interpreter to the total number of test cases administered. This metric provides a clear indication of the reliability of the code generated.
- **Code coverage.** This will be assessed through two distinct methodologies. First, we will employ Pylint to analyze whether the generated test cases are up to established coding standards. And then, Pytest and coverage.py are used to generate coverage metrics. Users can choose to evaluate either branch or line coverage.
- **Financial cost.** This is directly proportional to the number of tokens processed. Monitoring this will ensure that while aiming for optimal performance, we remain within viable economic boundaries.

8 Non-functional requirements

8.1 Look and Feel Requirements

- The system's functionality will be accessed through a VSCode extension, allowing the user to have code automatically tested and download the bot's report as well as the updated code.

- Because there is little the user needs to do while interacting with the bot, the extension will have a minimalist design that is easily comprehensible.
- There will also be a UI that accompanies the extension, which will also have a minimalist design.

8.2 Usability and Humanity Requirements

- A 'README' document will accompany the software package. This document will provide users with an overview of the software's capabilities. Furthermore, it will include step-by-step guidelines, ensuring that even users with less technical expertise can utilize the software's full range of functionalities with ease.
- There will be ‘documentation’ and ‘what do they mean’ buttons within the landing and main pages of the UI, to increase the users’ understanding of their statistics and how our product functions.

9 Risks

9.1 Technical Challenges

- The LLM might grapple with edge cases, boundary scenarios, and a web of interdependencies between various code sections. Such complex nuances make it challenging for larger code commits to consistently achieve a desired compilation and success rate.
- As with all LLMs, OpenAI models have inherent restrictions related to response length and the volume of information it can process. This poses a risk, especially with large commits, where the bot might inadvertently omit or be incapable of managing the requisite data.
- Although we strive for as much coverage as possible for the input code, there remains a risk that pivotal sections of the code might be left out by the bot.

9.2 Performance Issues / Cost bottleneck

- Originally, we planned for an Input Formatter to reduce the number of tokens consumed for certain code lines. Due to time constraints we did not achieve this. With OpenAI's API billing model being on a per-token basis, there is a concern about the long-term economic feasibility of our bot's operations.

9.3 Security Risks

- One of our foundational operational premises involves entrusting OpenAI with both the underlying code of our GenAI Bot and the stakeholder's repository code. We will be leaving all data privacy and security in the hands of OpenAI.
- Sensitive or proprietary code being submitted to the bot may break OpenAI rules, or raise data privacy and security concerns. There have been instances where contents fed to ChatGPT were accidentally leaked due to improper management by the OpenAI team, which are unfortunately out of our control.