Project Development Plan

Version 3

Team #2:
Bahar Abbasi Delvand
Krishika Mirpuri
Neha Asthana
Nathan Peereboom
Lawrence Gu

Table 1: Revision History

#	Author(s)	Description	Date
Rev 0	All	Creation of the first draft	22nd October, 2023
Rev 1	All	Update to Plan	7th February, 2024
Rev 3	All	Final Version	1st April, 2024

The genAI test BOT is an ambitious initiative centered on addressing the challenges faced in software development, particularly in the creation of test suites and code documentation.

The project's origins are rooted in our aspiration to provide developers and various stakeholders with an automated solution. Consequently, project managers will experience enhancements in code quality, faster development cycles, and more efficient project management. In addition, quality assurance teams will gain access to comprehensive test suites that are automatically generated, streamlining their testing efforts. Ultimately, end users will reap the benefits of superior software quality, resulting in fewer defects and a more dependable user experience.

Team Meeting Plan

At least two weekly meetings, once on discord on Tuesday afternoons, once in person, on Friday morning. The schedule may be adjusted in case a team member is unable to make it, and there may be more meeting times set to accommodate and guarantee that the milestones would be met.

Team Communication Plan

- Weekly Discord calls for progress check-ins
- Weekly in-person meetings
- Sharing google docs so we can work simultaneously on the documents
- Email to share project milestones and important files and documents
- Github for version control and to share and work on the code together
- Gitlab

Team Member Roles & Responsibilities

For now, every person on the project will be responsible for creating issues, coding, testing, reviewing, and commenting on issues & bugs. In the future, more specific roles will be created and assigned as the system and its design is better understood.

Person	Role	Responsibility
Neha Asthana	Project Manager	Creating issues, coding, testing, reviewing.
Nathan Peereboom		

Krishika Mirpuri	Developer	Commenting on issues & bugs
Bahar Abbasi Delvand		
Lawrence Gu		

Workflow Plan:

The version control system that will be used is git and GitHub.

- **Version Control**: The project will be managed through a git repository which will be used to manage the source code, documentation and any related files. The repository will also have several branches to support the different aspects/stages of the project like fixing bugs, optimizing etc.
- **Pull requests**: If any member of the group wants to propose any changes, they will use pull requests to facilitate code review before merging it to the main branch. This will help to maintain the code quality, identify any bugs or fixes and adhere to the project coding standards.

Proof of Concept Demonstration Plan:

For our proof of concept demonstration, we will build the baseline test generation loop that the rest of the system will be built off of. This will include the following modules:

- **File Parser:** The file parser will take as input a single Python source code file (the completed system will be able to take an entire repository as input). The input will be split into multiple chunks, each of a size less than the input size of the LLM we will use.

- **Diff parser:** The diff parser will take as input a single source file outputted from git diff (same note about the completed system for the file parser applies here). For our proof of concept, the diff parser will only consider code that was added in the commit; deleted code will be handled in the completed system. Similar to the file parser, the input will be split into chunks of a lesser size than the input size of the LLM used to generate test cases.

- **Test Case Generator:**

    - The chunks from a single source file created by one of the parsers will be run through the LLM with this prompt:
      “Given the following source code, generate a Pytest test suite for it.”
    - If any errors are found with the test suite, the following prompt will then be given to the LLM:
      “Here is the result from the python interpreter after executing your generate test suite:
      {test_suite}
      ---
      {result}
      Generate a new test suite that fixes these errors.”
    - If it is found that the test suite does not cover at least 90% of the code (in terms of branch coverage), the following prompt will be given to the LLM:
      "Here is the coverage metrics for your previous test suite:
      {test_suite}
      --
      {coverage}
      Generate a new test case to achieve 100% branch coverage."

- **Execution Engine and Coverage Measurement:** The Execution Engine and the coverage measurement module execute test suites generated by the Test Case Generator. If there is an error in the test suite, the Execution Engine parses out the error trace from pytest. The Execution Engine will also parse the coverage metrics from coverage.py (https://coverage.readthedocs.io/en/7.3.2/). The Execution Engine will also track the following statistics:
    - A record of the pass/fail history of each test case. This is important because both the source code could be changed between test generation attempts, and LLMs are notoriously nondeterministic and we need to ensure that test cases are being generated as reliably as possible.

We aim to evaluate everything with both GPT-3.5 and GPT-4 whenever possible and compare their performance. By comparing the performance metrics of these models, we aim to determine scenarios where GPT-3.5's affordability might outweigh the advantages of GPT-4. However, we acknowledge that for particularly large inputs, GPT-4 might be the only option due to its higher token limit.

Technology:

- **Programming Language**: Python
- **Linter Tool**: We will use “pylint” for linting Python code. It is a tool that helps with identifying style inconsistencies and potential bugs.

- **Testing framework**: We will be using PyTest as our unit testing framework.
- **CI/CD**:
    - We will automate the build, test, and deployment process with GitLab CI/CD.
    - We have configured the project to allow CI/CD, and will define the necessary environment variables in our GitLab project settings to securely store API keys.
    - Once our bot is mostly set up, we will configure CI/CD stages and jobs to automate the tasks of running unit tests to validate the bot's generated test cases, scan for code quality and adherence to coding standards using a linter tool and code coverage analysis to measure test coverage with coverage.py.
- **Performance measuring tools**: We may opt to utilize "cProfile," a tool designed specifically for profiling Python applications, if we find it necessary.
- **Libraries**: We are not planning on using any ML libraries as of yet, but we will be using 'coverage.py' as a coding library.
- **Other Tools**: We will use MongoDB for data management.

Coding Standard:

- **Coding Style**: We will adhere to PEP 8, the official Python style guide, for both source code and generated tests.
    - **Formatting tools** - We will use "autopep8" to automatically format the code and maintain consistency.
    - **Docstrings** - All functions will have docstrings explaining their purpose, parameters, and return values.

- **Project Structure and Naming Conventions**:
    - **Directories** - We will organize the project with separate directories for source code, generated test cases, utility scripts, and documentation.
    - **Filenames**: We will use clear, descriptive filenames. For example, a pattern like "gentest_<module>.py" for generated test suites.
    - **Variable and Function Names** - We will use descriptive names that convey their purpose without requiring excessive comments, such as "test_<function_name>_<condition_or_scenario>()".

- **Version Control**
    - **Check-ins** - We will not check in automatically generated tests directly. Instead, we will review them for relevance and correctness, and then commit.
    - **Commit Messages** - These need to be succinct and precise. Notably, we will indicate whether a commit is generated by LLM.

- **Review Process**
    - **Peer Review**: We will have a peer review process in place to ensure that the tests are accurate and meaningful.

Project Scheduling:

The project will be divided into three milestones which we have decided on with the guidance of our supervisor.

- **Milestone 1: Nov 20th, 2023**

  Milestone 1 aims to establish a baseline for test case generation.
    - Establish baseline test case generation loop
    - Milestone deliverables:
        - Module 1: File parsing
        - Module 2: Diff parsing
        - Module 3: Test Case Generator
        - Module 4: Execution Engine

- **Milestone 2 : February 29th, 2024**

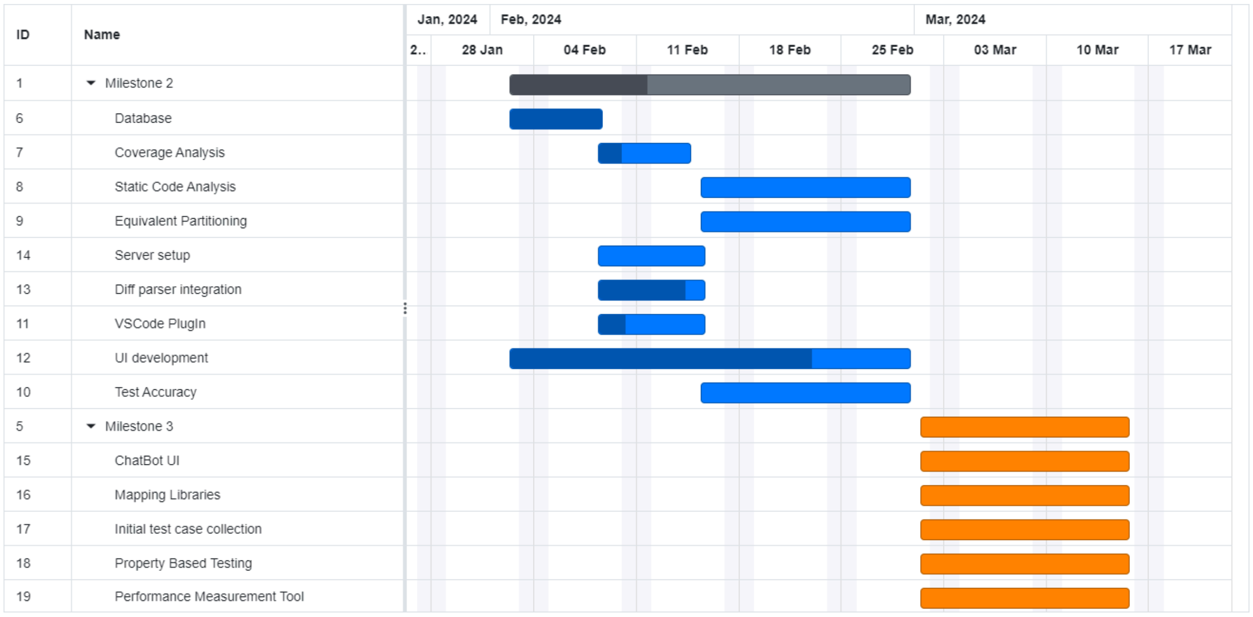
    - Database setup
        - design database schemas to store test cases, source file functions, and dependencies.
    - Coverage Analysis
        - Integrate coverage.py or similar tool for code coverage analysis.
    - Static code Analysis
        - Explore static code analysis tools to enforce coding standards and identify issues.
    - Equivalent Partitioning
        - Implement equivalent partitioning testing to ensure comprehensive coverage.
    - Test Accuracy
        - Develop mechanisms to track and report test case accuracy.
    - VSCode Plugin
        - Develop a VSCode plugin for seamless integration with the coverage reports.
    - UI development
        - Advance UI components
        - Create comprehensive documentation and tutorials within the UI.

- Server Set-up
    - Configure a server to host the GUI components and facilitate communication with the database.
- Diff-Parser Integration
    - Modify the diff parser to run on the server side and communicate with the database.

- **Milestone 3: March 16,2024**

    - Mapping Libraries
        - Map libraries and dependencies to specific functions in the source code.
    - Initial test case collection
        - Prompt users to provide initial test cases for better language model responses.
    - Property Based Testing
        - Explore property-based testing techniques for automatic test case generation.
    - Performance measurement Tool
        - Optionally, develop a performance measurement tool for evaluating application speed and resource usage.
        - **(Apr 1st Update)** Due to time constraints we did not end up being able to incorporate a performance measurement tool into our application.

- **Gantt Chart**
  Milestone 1 has been completed for this version update, so the gantt chart below only displays the updated schedule for milestones 2 and 3.



ID	Name
1	Milestone 2
6	Database
7	Coverage Analysis
8	Static Code Analysis
9	Equivalent Partitioning
14	Server setup
13	Diff parser integration
11	VSCode PlugIn
12	UI development
10	Test Accuracy
5	Milestone 3
15	ChatBot UI
16	Mapping Libraries
17	Initial test case collection
18	Property Based Testing
19	Performance Measurement Tool