**MuTest – Requirements**

Team Members: Vinnicius Castro

Description of project:

Mutest is a software tool that applies mutation testing techniques to evaluate the effectiveness of existing automated test suites. The system works by introducing small modifications, called mutations, into the source code. These mutations can include changes such as flipping Boolean values, altering arithmetic operators, or modifying conditional statements. Once mutations are applied, the tool executes the project’s test suite against the modified code.

**Section 1:**

I’m a software engineering major with a certificate in quality assurance, and this project is relevant to QA roles. Using an AI chatbot (Open AI, Chat GPT) I research ideas and how those ideas could help me get a better chance to not only develop QA knowledge but also enhance chances to find a job. Researching more about mutation tools and their capabilities I realized how mutation tools can take unit testing to the next level by showing if the test created are effective or not since traditional test coverage only shows what code is executed.

The project purpose is to design and implement a tool that introduces controlled mutations into Python source code and reports a mutation to evaluate the effectiveness of the test suite.

Mutest scope includes researching mutation testing and AST concepts, creating a core mutation engine with basic operator types, and implementing test execution with mutation score calculation. The project will also provide a command-line interface with simple text or HTML reporting and limited CI/CD integration using GitHub Actions. Advanced features like graphical dashboards, multi-language support, or extensive analytics are not included. The goal is to deliver a functional and well-documented prototype ready for demonstration and future improvements.

Technologies Used:

Languages: Python.   
Tools: Git, GitHub, GitHub Actions, Code IDE, Command Line Interface (CLI)

**Section 2a:**

Must Have Requirements:

* The system shall parse source code files into an Abstract Syntax Tree (AST).
* The system shall generate code mutants by applying mutation operators to the AST.
* The system shall execute the existing test suite against each generated mutant.
* The system shall track and record whether each mutant was killed or survived based on test results.
* The system shall calculate the mutation score by dividing the number of killed mutants by the total number of mutants.
* The system shall display mutation testing results, including the mutation score and surviving mutants.
* The system shall provide a command-line interface (CLI) that allows users to run mutation tests and specify test functions.
* The system shall generate a report summarizing mutation testing results in text, HTML or Json formats.
* The system shall integrate with a CI/CD pipeline (such as GitHub Actions) to allow automated mutation testing during continuous integration.
* The system shall provide error handling and user-friendly messages for invalid inputs or failed test executions.

**Section 2b:**

Stretch Requirements:

* The system may provide a simple log of all mutation testing activities, including timestamps, file names, and operator types used.
* The system may allow users to choose which mutation operators to use or turn off certain types.
* The system may save previous test results so users can compare them later.
* The system may color-code the CLI output (e.g., green for killed mutants, red for surviving mutants) for easier reading.
* The system may include a small help menu in the CLI that explains available commands and options.
* The system may automatically create a backup of original source files before mutation.
* The system may allow users to limit the number of mutants generated per file for faster testing.

**Section 2c:**   
Weekly schedule:

|  |  |  |  |
| --- | --- | --- | --- |
| **Week(s)** | **Goal** | **Must-have Requirements** | **Stretch Requirements** |
| 1-2 | |  | | --- | |  |  |  | | --- | | Research mutation testing and AST basics | | |  | | --- | | Parse source code files into an Abstract Syntax Tree (AST) |  |  | | --- | |  | | - |
| 3-4 | Build basic mutant generator | Generate code mutants by applying mutation operators to the AST | - |
| 5-6 | Implement test execution and kill/survive tracking | Execute existing test suite against each mutant; Track and record killed/survived mutants | Color-code CLI output; Include a simple help menu in the CLI |
| 7 | |  | | --- | |  |  |  | | --- | | Add mutation score calculation | | Calculate mutation score by dividing killed mutants by total mutants | Save previous test results for comparison |
| 8-9 | |  | | --- | |  |  |  | | --- | | Develop CLI and reports | | |  | | --- | |  |   Provide CLI to run mutation tests and specify test functions; Display mutation results including score and surviving mutants; Generate reports in text, HTML, or JSON | Allow users to choose which mutation operators to use or disable certain types |
| 10-11 | Add minimal CI/CD integration | Integrate with CI/CD pipeline (GitHub Actions) for automated testing | Allow users to choose which mutation operators to use or disable certain types |
| 12 | Write user and developer documentation | Provide error handling and user-friendly messages | |  | | --- | |  |  |  | | --- | | Automatically create backups of original source files; Allow limiting the number of mutants per file | |
| 13 | |  | | --- | |  |  |  | | --- | | Perform tool testing, bug fixing, and stretch feature implementation | | Ensure all core features are stable | Any remaining minor enhancements |
| 14 | Final polish and presentation preparation | Demo-ready, all must-have features functional | Any remaining minor enhancements |

**Section 3:** Design Overview of the Product.

Workflow:

**User Prepares the Project**

* User selects the Python project or source files to test.
* User ensures that unit tests are present and runnable (e.g., via pytest).

**Start Mutation Testing via CLI**

* User opens the terminal and runs Mutest CLI.
* Optional parameters can be provided:
  + Target files or folders
  + Test functions to run
  + Specific mutation operators to enable/disable
  + Maximum number of mutants per file

**Test Execution**

* Each mutant is executed against the existing unit tests.
* Mutest tracks whether each mutant is killed (tests fail) or survives (tests pass).
* Logs are optionally created with timestamps, file names, and operator types (stretch feature).

**Mutation Score Calculation**

* Mutest calculates the mutation score: killed mutants ÷ total mutants.
* Surviving mutants are recorded for reporting.

**Results Reporting**

* Results are displayed in the CLI:
  + Mutation score
  + Surviving mutants
  + Optional color-coded output for easy reading (stretch feature)
* A report is generated in the selected format: text, HTML, or JSON.

**CI/CD Integration**

* If configured, the mutation testing can be run automatically via a CI/CD pipeline (e.g., GitHub Actions) on code commits.

**Post-Processing (Optional)**

* Historical results may be saved for later comparison (stretch feature).
* Users may review logs, reports, and surviving mutants to improve tests.

**End of Workflow**

* User closes the CLI or proceeds to fix surviving mutants and rerun tests.

Resources:

Mutest uses Python 3.8+, libraries including astor (AST manipulation), pytest (testing framework), click (CLI framework), black (code formatting), pytest-cov (coverage analysis), and colorama (colored output). Development is done with Visual Studio Code, Git for version control, and GitHub for hosting and CI/CD via GitHub Actions. All resources are free and work on any operating system (Linux, macOS, Windows).

Data:

Data will be stored in files and data needed to run the program (the unit test) need to be created by the user.

Pictures/ Diagrams:  
A screenshot of a video game

AI-generated content may be incorrect.

**Section 4:** Verification:

**Demo:**

To verify the mutation testing tool works as intended, I will create three unit test functions: two robust tests that should kill all generated mutants, and one intentionally weak test that will allow mutants to survive. This test suite will validate that Mutest correctly distinguishes between killed and surviving mutants and successfully identifies weak or insufficient test coverage.

**Testing:**

|  |  |
| --- | --- |
| **All Requirements** | **Acceptance testing criteria** |
| The system shall parse source code files into an Abstract Syntax Tree (AST). | Verify that a Python file is successfully converted into an AST using test files and inspect the AST structure programmatically. |
| The system shall generate code mutants by applying mutation operators to the AST. | |  | | --- | |  | |  |   Check that mutants are created for sample files; confirm mutated code differs from the original AST and follows operator rules. |
| The system shall execute the existing test suite against each generated mutant. | Run the system on a project with unit tests and ensure each mutant triggers the test suite. Confirm test results are produced for each mutant. |
| The system shall track and record whether each mutant was killed or survived based on test results. | Inspect reports to verify that each mutant has a status of killed or survived matching actual test outcomes. |
| The system shall calculate the mutation score by dividing the number of killed mutants by the total number of mutants. | Compare the reported mutation score with manually calculated score using killed and total mutants; they should match. |
| The system shall display mutation testing results, including the mutation score and surviving mutants. | Run the CLI and check that the mutation score and list of surviving mutants are correctly displayed in the output. |
| The system shall provide a command-line interface (CLI) that allows users to run mutation tests and specify test functions. | |  | | --- | |  |  |  | | --- | | Test the CLI commands for running tests, specifying files/functions, and validate that the system executes as expected. | |
| The system shall generate a report summarizing mutation testing results in text, HTML or JSON formats. | |  | | --- | |  |  |  | | --- | | Generate reports in all three formats and verify that they contain mutation score, killed/survived mutants, and other key data. | |
| The system shall integrate with a CI/CD pipeline (such as GitHub Actions) to allow automated mutation testing during continuous integration. | |  | | --- | |  |  |  | | --- | | Commit changes to a test repository with GitHub Actions configured and verify that mutation tests run automatically and produce a report. | |
| The system shall provide error handling and user-friendly messages for invalid inputs or failed test executions. | |  | | --- | |  |  |  | | --- | | Intentionally provide invalid inputs and check that meaningful error messages are displayed without crashing the program. | |
| The system may provide a simple log of all mutation testing activities, including timestamps, file names, and operator types used. | Enable logging, run mutation tests, and verify that logs record timestamps, filenames, operator types, and test outcomes. |
| The system may allow users to choose which mutation operators to use or turn off certain types. | Run tests with different operator configurations and confirm only selected operators generate mutants. |
| The system may save previous test results so users can compare them later. | Run tests multiple times and verify that previous results are stored and accessible for comparison. |
| The system may include a small help menu in the CLI that explains available commands and options. | Run the help command (--help or -h) and check that it displays all available commands and usage instructions. |
| The system may color-code the CLI output (e.g., green for killed mutants, red for surviving mutants) for easier reading. | Execute tests in CLI and visually confirm color-coded output matches killed/survived status. |
| The system may automatically create a backup of original source files before mutation. | |  | | --- | |  |  |  | | --- | | Enable backup feature, run mutations, and confirm that original source files are preserved in a backup folder. | |
| The system may allow users to limit the number of mutants generated per file for faster testing. | Run tests with a mutant limit and verify that no more than the specified number of mutants is generated per file. |

**Sources/Citation/Resources** Links: