

Design Document for Mutation Testing

Team Info

Name: Sim_Hackers

Members: Ramya Panambayil Rajan

Sarath Sasidharan Nair

Pai Radhesh Udyavara

Vignesh Sekar

Problem Statement

Automation of Mutation Testing in project environments.

Purpose

This document outlines the manual steps for Mutation testing, which is used to evaluate the test quality of unit test by introducing small code changes(mutations) to the original source code

Pre-requisites

- VS code
- LLM Farm
- Cantata

Selection of LLMs and prompting

Creating realistic mutants for mutation testing is essential to simulate actual bugs developers might introduce. Realistic mutants should reflect likely human mistakes, not just any syntactic change

So below points to be considered while deriving a prompt

Mutation Type	Example	Description
Relational Change	$> \rightarrow >=, == \rightarrow !=$	
Logical operator change	$\&\& \rightarrow \parallel$	Logical bugs
Arithmetic change	$+ \rightarrow -, * \rightarrow /$	Numerical errors are very common
Return value Change	<code>return b;</code> \rightarrow <code>return a;</code>	Misplaced return
Assignment error	<code>x = y</code> \rightarrow <code>x = 0</code>	
Off-by-One Error	<code>i < n</code> \rightarrow <code>i <= n</code>	Boundary issue
Break semantics	<code>int x = 1;</code> \rightarrow <code>int x = 0;</code>	
Control Flow Alteration	<code>if (x > 0) \rightarrow if (!(x > 0))</code> remove body of if statement remove a break in loop	
Value change	Literal Change <code>6</code> \rightarrow <code>5</code>	
Wrong Variable Use	<code>return x;</code> instead of <code>return y;</code>	Misused identifiers

Pointer Arithmetic	$ptr \neq NULL \rightarrow ptr == NULL$ $*(ptr) = 10 \rightarrow *(ptr + 1) = 10$	
Null or Zero Bugs	$int *ptr = \&value; \rightarrow int *ptr = NULL$	Skipping null checks Crashes/undefined behavior

In proper mutation testing, each mutant should have only one change. This principle is known as "first-order mutation", and it's the standard because it ensures that:

- precisely track the effect of one mutation.
- Tests can target a single behavioral deviation.
- Mutation score remains meaningful.

Prompts:

We have used the following prompt to prompt different LLMs

Generate specific mutation variants of this C function by changing ONLY ONE operator at a time.

For each mutation:

- 1. Give it a name (Mutant_Arithmetic, Mutant_Relational, etc.)*
- 2. Show the complete function with the mutation highlighted or commented*

Target these operators for mutation:

*- {**Mutant_type}*

*generate all possible combination of mutants by changing only one {**Mutant_type} at a time*

****Add c function here*****

***Mutant_type can be Arithmetic operators (+ - * /) or Relational operators (<, >, ==, !=) or etc..*

LLMs comparison

Following factors to be considered while comparing different LLMs

Dimension	Measures
Correctness	Does it compile/run?
Coverage	Types of bugs mutated
Redundancy	Duplicate detections
Realism	How close is it to human mistakes?
Test Value	Does it help test suites evolve?

The below results are derived by prompting on a sample function

Factors	GPT-4o mini	Gemini 1.5 Flash
Mutants count	More	Less
Correctness	All mutants can be compiled	Few mutants cannot be compiled (Error Mutants)
Realism	Good	Ok
Redundancy	No	No
Coverage	Good	Ok

Based on this table, we decided to use **GPT-4o mini** for Mutation test

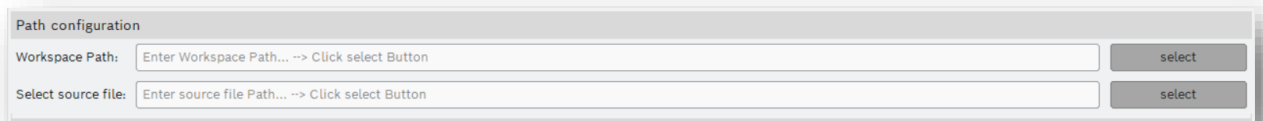
Automation workflow

Prerequisite:

- LLM model is identified
- Prompts are identified
- CLI commands for Cantata tool are known

Workflow

1. Selection of Cantata environment path



User must select a path where the Cantata files and test suites are available

The mutants, log files, test report and generated test cases will be added in this path

Also, user must select the source (*.c) file to be tested

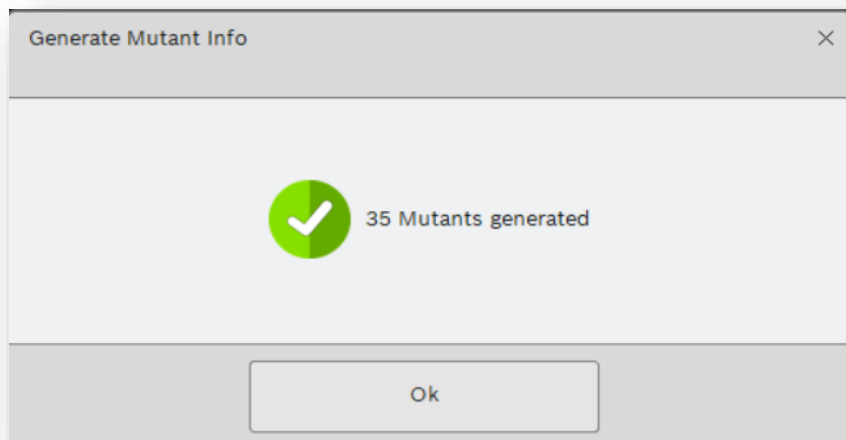
2. Generate and Execute mutants for different mutant type

Here is a drop down to select the function name and mutant type. User can select a particular function and mutant type and generate mutants for that type or there is a provision to select All and generate.

On clicking on Generate Mutants, respective prompting is given to LLM and generated mutants are added in the workspace path, by adding a folder named Mutants

Copy the contents of original file to this mutant file, and add the function generated from LLM. Also, information bar showing how many mutants are generated

Create Cantata workspace by adding mutant source file



By clicking on Execute test suite, tests run for the generated mutants using Cantata CLI commands. Here also user can choose the function and mutant type

3. Show the report- how many live and killed mutant

It shows the count of how many Live Mutants and killed mutants after test execution

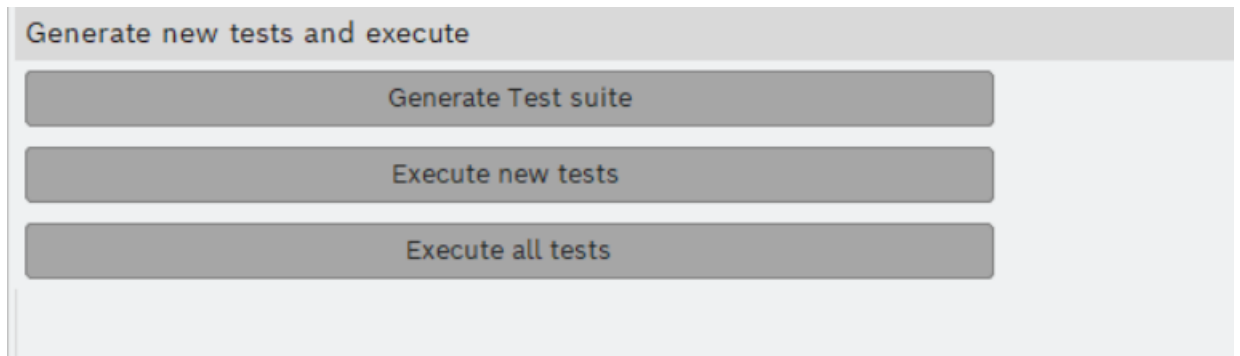
Live mutants are the one in which test case is passed and Killed mutants are in which test cases are failed

Mutation score is calculated and displayed in the box

Mutation Score= (Number of Killed Mutants/ Total Number of Mutants)×100

Also, a log is generated and saved in the workspace path, which includes detailed information of each mutant generated and its type and the test case status

4. Create test cases for live mutant



Identify which line is changed in the mutant code. Identify the test case for respective function, create new test case by adding the check for line of change.

For all the live mutants, test cases to be generated.

5. Execute the tests for newly added test suite and ensure all mutants are killed
6. Execute all tests – original and new tests for all mutants
7. Ensure a good score is achieved

Entire GUI is designed as below

Mutation Testing Tool

File

Option

Information

Mutation Testing Tool

General

Path configuration

Workspace Path:

select

Select source file:

select

Generate and Execute Mutants

Function name: All

Mutant Type: All

Generate Mutant

Function name: All

Mutant Type: All

Execute Test Suite

Test results info

Live Mutants count:

Killed Mutants count:

Mutation score:

Open log file

Generate new tests and execute

Generate Test suite

Execute new tests

Execute all tests