

Executando Testes de Mutação com o Mutmut em Python

Mutação de teste é uma técnica avançada de teste de software usada para garantir a robustez dos casos de teste. O **Mutmut** é uma ferramenta de teste de mutação para Python que ajuda a identificar pontos fracos nos testes, introduzindo alterações sutis no código-fonte (chamados de mutantes) e verificando se os testes existentes conseguem detectar essas mudanças.

1. Instalação do Mutmut

Para começar, instale o Mutmut junto com as outras dependências que estão definidas no arquivo `requirements.txt` usando o `pip`, que é o gerenciador de pacotes para Python, execute os comandos abaixo:

```
sudo apt install python3-venv
cd cal_python
python3 -m venv /cal_python/venv
source venv/bin/activate
pip install -r requirements.txt
```

2. Preparando o Projeto para Testes de Mutação

Antes de executar os testes de mutação, você precisa garantir que o seu projeto Python está pronto para testes unitários. Certifique-se de que você tem um diretório de testes estruturado adequadamente e que seus testes estão passando. Aqui está um exemplo de estrutura de projeto:

```
[SEU PROJETO]/
├── todo/
│   └── todo.py
└── tests/
    └── test_todo.py
```

No exemplo acima, `todo.py` é o módulo que queremos testar e `test_todo.py` contém os testes unitários para o módulo.

3. Executando o Mutmut

Para executar testes de mutação usando o Mutmut, siga os passos abaixo:

1. **Navegue até o diretório do seu projeto:** Certifique-se de estar no diretório raiz do seu projeto.

```
cd T0-D0-LIST
```

2. **Configure o Mutmut:** Execute o comando de configuração do Mutmut para iniciar o processo de configuração dos testes de mutação.

```
## Tests

## Execute testes normalmente com pytest
pytest -vv tests/todo.py

# Gere o relatório de testes em linha
pytest -vv tests/test_todo.py --cov=todo

# Gere o relatório de cobertura de testes
pytest -vv test_todo.py --cov=todo --cov-branch --cov-report html

# Execute testes de mutação com mutmut
mutmut run --paths-to-mutate=[SEU DIRETÓRIO]/To-Do-List/tests
```

Esse comando vai executar os testes unitários do projeto e criar mutantes, que são pequenas mudanças no código para ver se os testes detectam a mudança.

3. **Verifique os Resultados:** Após a execução do comando, verifique os resultados para ver quais mutantes sobreviveram e quais foram mortos.

```
mutmut results
```

O comando **results** fornece um resumo de quais mutantes foram detectados pelos testes e quais não foram. O objetivo é ter o maior número possível de mutantes "mortos" (detectados pelos testes).

4. Analisando e Melhorando os Casos de Teste

Se alguns mutantes sobreviverem (não forem detectados pelos testes), isso significa que os testes não estão cobrindo completamente o código modificado. Nesse caso, você precisa revisar e melhorar seus casos de teste.

Para isso é necessário identificar o trecho de código que faz referência ao mutante vivo e assim alterar esse teste ou inserir um novo caso de teste para matar o mutante (testar o caso de uso detectado que não está coberto pelos testes).

5. Reexecutando o Mutmut

Após adicionar o novo caso de teste, reexecute o Mutmut para verificar se o mutante agora é detectado pelos testes:

```
mutmut run --paths-to-mutate=[SEU DIRETÓRIO]/To-Do-List/tests
```

Depois de executar o Mutmut novamente, verifique os resultados para garantir que todos os mutantes foram mortos:

```
mutmut results
```

Se todos os mutantes forem mortos, isso significa que os testes agora são robustos o suficiente para detectar a maioria das alterações no código, garantindo uma maior cobertura de teste.

6. Demonstração de eliminação de mutantes no projeto

Ao executar o mutmut no projeto atual com o comando `mutmut run --paths-to-mutate=[SEU DIRETÓRIO]/To-Do-List/tests` a ferramenta nos retorna 23 mutantes vivos que precisam ser eliminados.

```
~/workspace/ufs/To-Do-List master !9 78 mutmut run --paths-to-mutate=/Users/ed1/workspace/ufs/To-Do-List/tests
- Mutation testing starting -
These are the steps:
1. A full test suite run will be made to make sure we
   can run the tests successfully and we know how long
   it takes (to detect infinite loops for example)
2. Mutants will be generated and checked
Results are stored in .mutmut-cache.
Print found mutants with 'mutmut results'.
Legend for output:
🔥 Killed mutants. The goal is for everything to end up in this bucket.
⌚ Timeout. Test suite took 10 times as long as the baseline so were killed.
😟 Suspicious. Tests took a long time, but not long enough to be fatal.
😄 Survived. This means your tests need to be expanded.
🚫 Skipped. Skipped.
mutmut cache is out of date, clearing it...
1. Running tests without mutations
! Running...Done
2. Checking mutants
! 59/59 🚫 36 ⌚ 0 😟 0 😄 23 🚫 0
```

Agora é necessário identificar qual é esse mutante, para isso executamos o comando `mutmut results` para visualizar todos os mutante vivos, após isso executado `mutmut show 10` para ver o trecho do código referente ao mutante, conforme a image abaixo:

```
~/workspace/ufs/To-Do-List master !9 78 mutmut results
To apply a mutant on disk:
  mutmut apply <id>
To show a mutant:
  mutmut show <id>
Survived 😄 (23)
---- /Users/ed1/workspace/ufs/To-Do-List/tests/test_todo.py (23) ----
9-14, 17, 20-21, 25, 27, 34-35, 37, 39, 46-53
~/workspace/ufs/To-Do-List master !9 78 mutmut show 10
---- /Users/ed1/workspace/ufs/To-Do-List/tests/test_todo.py
+++ /Users/ed1/workspace/ufs/To-Do-List/tests/test_todo.py
@@ -29,7 +29,7 @@
 @pytest.fixture
 # this fixture creates and returns temporary JSON File - "db_file" with a single to-do list item. The tmp_path is a pathlib.Path object that pytest uses to provide a temporar
y directory for testing purposes
def mock_json_file(tmp_path):
-     todo = [{"Description": "Get milk", "Priority": 2, "Done": False}]
+     todo = [{"Description": "XXGet milkXX", "Priority": 2, "Done": False}]
     db_file = tmp_path/"todo.json"
     with db_file.open("w") as db:
         json.dump(todo, db, indent=4)
```

Portanto, no arquivo `tests/test_todo.py` inserimos mais item de teste que eliminará esse mutante.

```
def mock_json_file(tmp_path):
    todo = [{"Description": "Get milk", "Priority": 2, "Done": False},
            {"Description": "XXXGet milkXXX", "Priority": 2, "Done":
```

```
False}] ## <- Novo item
  db_file = tmp_path/"todo.json"
  with db_file.open("w") as db:
    json.dump(todo, db, indent=4)
  return db_file
```

Agora só precisamos rodar novamente os testes e ver se o mutante foi realmente eliminado.

```
## Execute os testes de mutação com mutmut novamente.
mutmut run --paths-to-mutate=[SEU DIRETÓRIO]/To-Do-List/tests
```

Observe na imagem abaixo que o mutante 10 não aparece mais na lista, significando que foi eliminado.

A terminal window showing the output of the mutmut command. The window title is '~ /workspace/ufs/To-Do-List' with a status bar showing 'master !9 78' and 'mutmut results'. The output text is: 'To apply a mutant on disk: mutmut apply <id>', 'To show a mutant: mutmut show <id>', 'Survived 🍌 (29)', '---- /Users/ed1/workspace/ufs/To-Do-List/tests/test_todo.py (29) ----', and a list of surviving mutants: '17, 20-21, 25, 27, 34-35, 37, 39, 46-53, 60-71'.

```
~/workspace/ufs/To-Do-List master !9 78 mutmut results
To apply a mutant on disk:
  mutmut apply <id>

To show a mutant:
  mutmut show <id>

Survived 🍌 (29)
---- /Users/ed1/workspace/ufs/To-Do-List/tests/test_todo.py (29) ----
17, 20-21, 25, 27, 34-35, 37, 39, 46-53, 60-71
```