Universidade Federal de Sergipe
Centro de Ciências Exatas e Tecnologia
Departamento de Computação

Atividade 2 – Testes de Mutação

Teste de Software - COMP0444 - T01
Professor: Glauco de Figueiredo Carneiro

João Pablo da Paz de Jesus

SÃO CRISTÓVÃO, SERGIPE.
2024

Tutorial: Testes de Mutação em Projetos Python

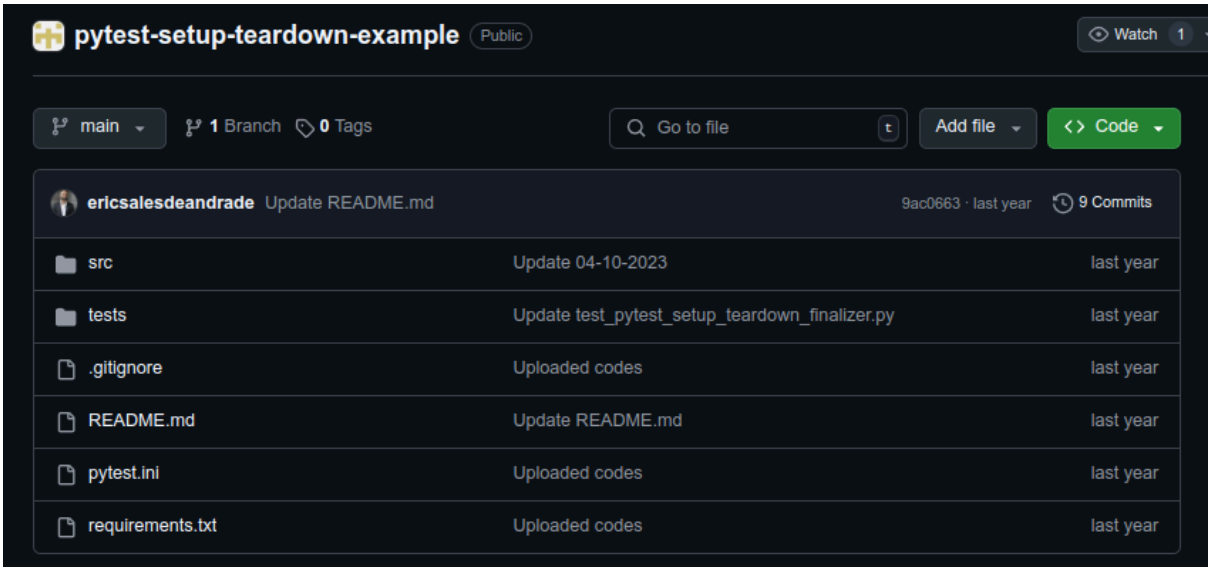
Link do Github:
https://github.com/Pablo-oficial/Teste_Software_Mutantes_2024_Jesus_Joao
Link do Vídeo: Joao_Pablo_atividade_2.mkv

Este tutorial explora o uso de testes de mutação como uma ferramenta para avaliar a qualidade dos casos de teste em um projeto de software. Testes de mutação inserem propositalmente defeitos no código, permitindo que se avalie a eficácia dos casos de teste ao detectar esses defeitos.

No contexto deste tutorial, será configurado e utilizado um projeto de software em Python, aplicando um framework ou toolkit para testes de unidade, um toolkit para avaliação de cobertura de testes, e um terceiro para realizar testes de mutação.

Escolha do Repositório GitHub

Para cumprir a etapa sobre a escolha do código disponível no GitHub do Pytest, escolhi um exemplo que realiza a validação e classificação de endereços IP. Este código, escrito em Python, é destinado à validação e classificação de endereços IP, tanto no formato IPv4 quanto IPv6.



Verifica a validade do endereço IP. Retorna "Valid IPv4" se for um endereço IPv4 válido, "Valid IPv6" se for um endereço IPv6 válido e "Invalid IP" para endereços IP inválidos.

```python
def validate_it(self):
        """
        Check the validity of an IP address.
        :return: The type of IP address. Returns 'Invalid IP' for invalid IP address.
        """
        # Regex expression for validating IPv4
        regex = "^((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\\.){3}(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])$"

        # Regex expression for validating IPv6
        regex1 = "((([0-9a-fA-F]){1,4})\\:){7}([0-9a-fA-F]){1,4}"

        # Compiling the regex expressions
        p = re.compile(regex)
        p1 = re.compile(regex1)

        # Checking if it is a valid IPv4 address
        if (re.search(p, self.IP)):
            return "Valid IPv4"

        # Checking if it is a valid IPv6 address
        elif (re.search(p1, self.IP)):
            return "Valid IPv6"

        return "Invalid IP"
```

Identifica a classe do endereço IP, se for um IPv4 válido. Retorna a classe como "A", "B", "C", "D" ou "E". Retorna "N/A" para endereços não IPv4.

```python
    def find_class(self):
        """
        Find out the class of an IP address.
        :return: The class of the IP address.
        """
        if self.validate_it() != "Valid IPv4":
            return "N/A"
# Not applicable for non-IPv4 addresses

        ip = self.IP.split(".")
        ip = [int(i) for i in ip]


# If ip[0] >= 0 and ip[0] <= 127 then the IP address is in c
lass A
        if ip[0] >= 0 and ip[0] <= 127:
            return "A"


# If ip[0] >= 128 and ip[0] <= 191 then the IP address is in
class B
        elif ip[0] >= 128 and ip[0] <= 191:
            return "B"


# If ip[0] >= 192 and ip[0] <= 223 then the IP address is in
class C
        elif ip[0] >= 192 and ip[0] <= 223:
            return "C"


# If ip[0] >= 224 and ip[0] <= 239 then the IP address is in
class D
        elif ip[0] >= 224 and ip[0] <= 239:
            return "D"

        # Otherwise the IP address is in Class E
        else:
            return "E"
```

Formata o endereço IP para a notação padrão. Retorna o endereço IP formatado ou "Invalid IP" se o endereço não for válido.

```
    def format_ip(self):
        """
        Format the IP address to standard notation.
        :return: The formatted IP address.
        """
        try:
            parts = [str(int(part)) for part in self.IP.split('.')]
            return '.'.join(parts)
        except ValueError:
            return "Invalid IP"
```

Converte o endereço IP para sua representação inteira. Retorna o valor inteiro correspondente ou None se o endereço não for válido.

```
    def ip_to_int(self):
        """
        Convert the IP address to an integer.
        :return: The integer representation of the IP address.
        """
        try:
            parts = [int(part) for part in self.IP.split('.')]
            return (parts[0] << 24) + (parts[1] << 16) + (parts[2] << 8) + parts[3]
        except ValueError:
            return None
```

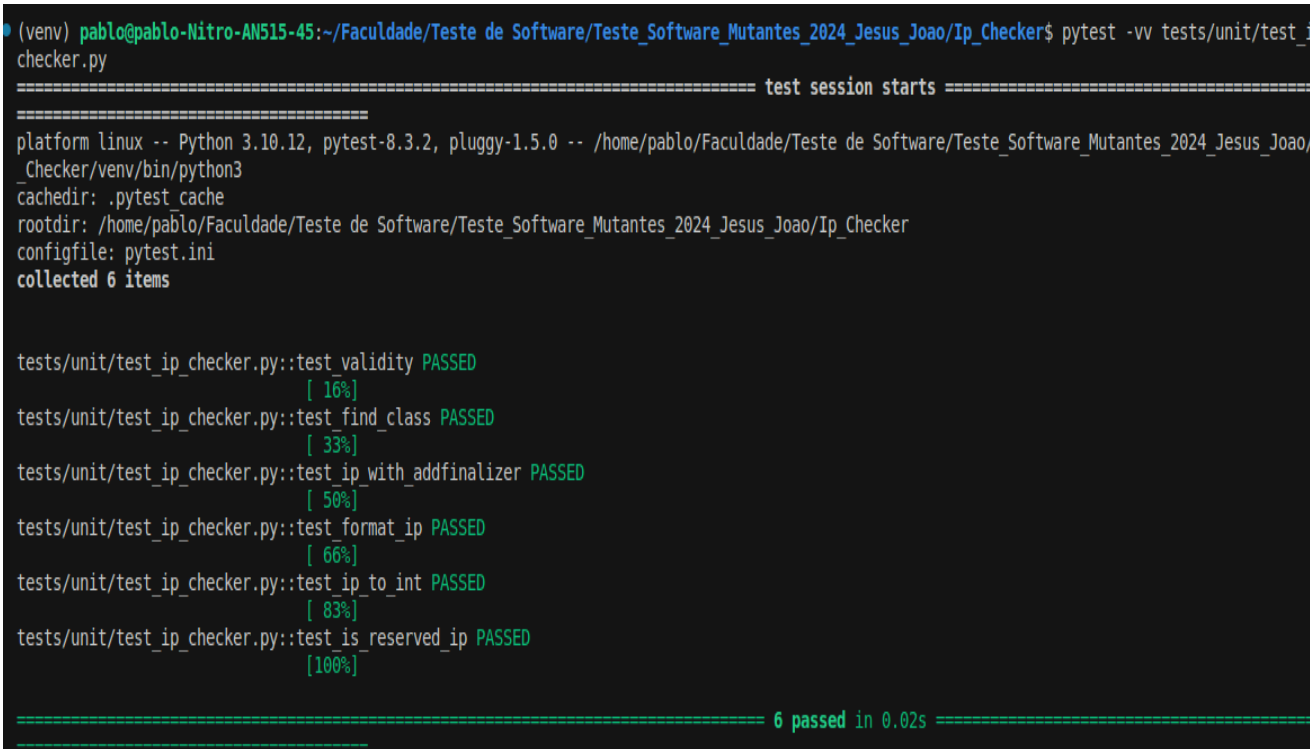
Verifica se o endereço IP é reservado (rede privada). Retorna `True` se o IP estiver dentro dos intervalos reservados para redes privadas, caso contrário, retorna `False`.

```
    def is_reserved_ip(self):
        """
        Check if the IP address is a reserved IP address (private network).
        :return: True if the IP address is reserved, otherwise False.
        """
        if self.validate_it() != "Valid IPv4":
            return False

        ip_parts = [int(part) for part in self.IP.split('.')]
        # Check if the IP falls within the reserved ranges
        if (ip_parts[0] == 10) or \
           (ip_parts[0] == 172 and 16 <= ip_parts[1] <= 31) or \
           (ip_parts[0] == 192 and ip_parts[1] == 168):
            return True

        return False
```

Execução básica dos testes: O comando `pytest -vv tests/unit/test_ip_checker.py` é utilizado para rodar os testes unitários com o Pytest, proporcionando uma saída detalhada de cada teste.



Cobertura de linha (node): Para obter um relatório de cobertura de linhas do código, usamos o comando `pytest -vv tests/unit/test_ip_checker.py --cov=src.ip_checker`, que analisa quantas linhas do código foram cobertas pelos testes.

```
collected 6 items


tests/unit/test_ip_checker.py::test_validity PASSED
[ 16%]
tests/unit/test_ip_checker.py::test_find_class PASSED
[ 33%]
tests/unit/test_ip_checker.py::test_ip_with_addfinalizer PASSED
[ 50%]
tests/unit/test_ip_checker.py::test_format_ip PASSED
[ 66%]
tests/unit/test_ip_checker.py::test_ip_to_int PASSED
[ 83%]
tests/unit/test_ip_checker.py::test_is_reserved_ip PASSED
[100%]

---------- coverage: platform linux, python 3.10.12-final-0 --------
Name                Stmts   Miss  Cover
---------------------------------------
src/ip_checker.py      49     13    73%
---------------------------------------
TOTAL                  49     13    73%
```

Cobertura de ramificações (branch): Além da cobertura de linha, também podemos verificar a cobertura de ramificações (branches), garantindo que tanto os fluxos de execução principais quanto alternativos sejam testados. O comando `pytest -vv tests/unit/test_ip_checker.py --cov=src.ip_checker --cov-branch --cov-report html` gera um relatório HTML com cobertura de linha e ramificação.

```
collected 6 items

tests/unit/test_ip_checker.py::test_validity PASSED
tests/unit/test_ip_checker.py::test_find_class PASSED
tests/unit/test_ip_checker.py::test_ip_with_addfinalizer PASSED
tests/unit/test_ip_checker.py::test_format_ip PASSED
tests/unit/test_ip_checker.py::test_ip_to_int PASSED
tests/unit/test_ip_checker.py::test_is_reserved_ip PASSED

---------- coverage: platform linux, python 3.10.12-final-0 ----------
Coverage HTML written to dir htmlcov


============================================================ 6 passed in 0.05s ==
```

Coverage report: 71%

Files | Functions | Classes

coverage.py v7.6.1, created at 2024-09-03 22:36 -0300

File ▲	statements	missing	excluded	branches	partial	coverage
src/ip_checker.py	49	13	0	26	3	71%
Total	**49**	**13**	**0**	**26**	**3**	**71%**

coverage.py v7.6.1, created at 2024-09-03 22:36 -0300

Coverage report: 71%

Files | Functions | Classes

coverage.py v7.6.1, created at 2024-09-03 22:36 -0300

File ▲	function	statements	missing	excluded	branches	partial	coverage
src/ip_checker.py	ip_check.__init__	1	0	0	0	0	100%
src/ip_checker.py	ip_check.validate_it	9	1	0	4	1	85%
src/ip_checker.py	ip_check.find_class	13	7	0	12	1	44%
src/ip_checker.py	ip_check.format_ip	5	2	0	2	0	71%
src/ip_checker.py	ip_check.ip_to_int	5	2	0	2	0	71%
src/ip_checker.py	ip_check.is_reserved_ip	6	1	0	6	1	83%
src/ip_checker.py	ip_check.delete_objects	1	0	0	0	0	100%
src/ip_checker.py	*(no function)*	9	0	0	0	0	100%
Total		**49**	**13**	**0**	**26**	**3**	**71%**

coverage.py v7.6.1, created at 2024-09-03 22:36 -0300

Coverage report: 71%

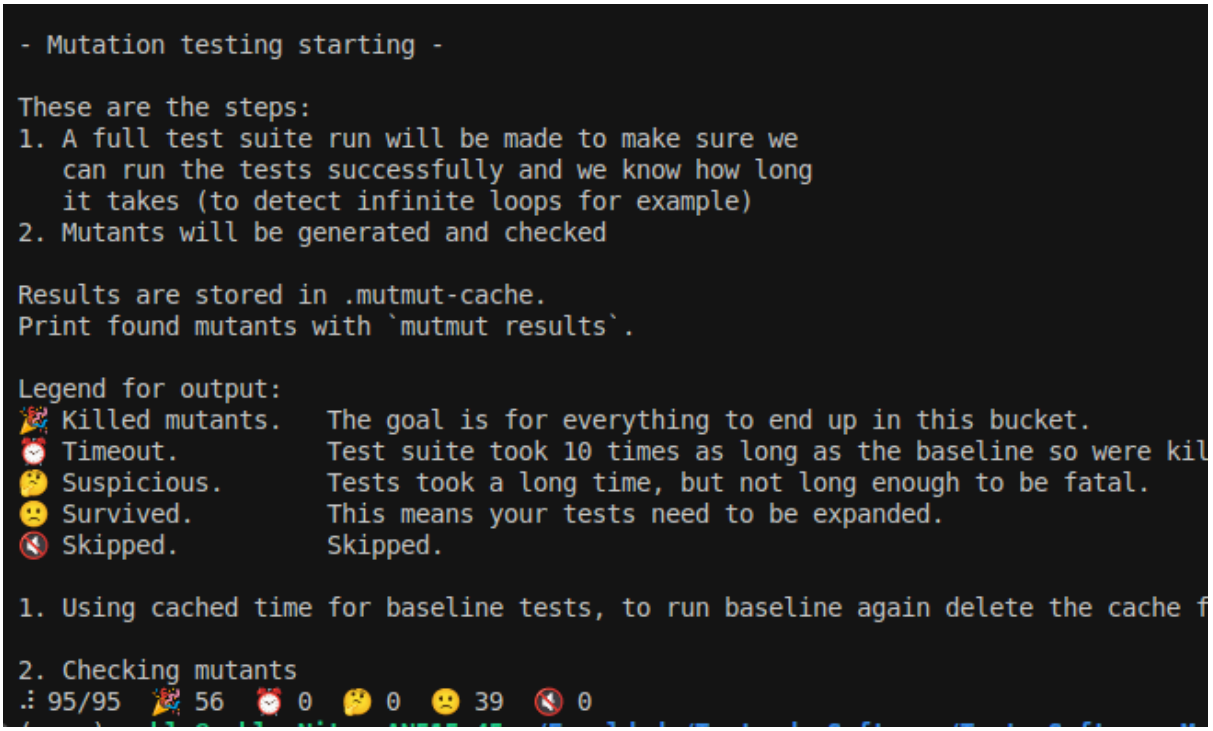
Files | Functions | Classes

coverage.py v7.6.1, created at 2024-09-03 22:36 -0300

File ▲	class	statements	missing	excluded	branches	partial	coverage
src/ip_checker.py	ip_check	40	13	0	26	3	67%
src/ip_checker.py	*(no class)*	9	0	0	0	0	100%
Total		**49**	**13**	**0**	**26**	**3**	**71%**

coverage.py v7.6.1, created at 2024-09-03 22:36 -0300

Testes de mutação com Mutmut: Por fim, para validar a eficácia dos testes, o comando `mutmut run --paths-to-mutate=src/ip_checker.py` executa testes de mutação usando o **Mutmut**, que altera o código de forma intencional e verifica se os testes conseguem capturar essas alterações, avaliando assim a robustez da suite de testes.



Mutation testing report

Killed 56 out of 95 mutants

File	Total	Skipped	Killed	% killed	Survived
src/ip_checker.py	95	0	56	58.95	39

src/ip_checker.py

Killed 56 out of 95 mutants

Survived

Survived mutation testing. These mutants show holes in your test suite.

Mutant 4

```
--- src/ip_checker.py
+++ src/ip_checker.py
@@ -23,7 +23,7 @@
         regex = "^((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\\.){3}(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])$"

         # Regex expression for validating IPv6
-        regex1 = "((([0-9a-fA-F]){1,4})\\:){7}([0-9a-fA-F]){1,4}"
+        regex1 = "XX((([0-9a-fA-F]){1,4})\\:){7}([0-9a-fA-F]){1,4}XX"

         # Compiling the regex expressions
         p = re.compile(regex)
```

Mutant 9

```
--- src/ip_checker.py
+++ src/ip_checker.py
@@ -35,7 +35,7 @@

         # Checking if it is a valid IPv6 address
         elif (re.search(p1, self.IP)):
-            return "Valid IPv6"
+            return "XXValid IPv6XX"

         return "Invalid IP"
```

Mutant 17

```
--- src/ip_checker.py
+++ src/ip_checker.py
@@ -52,7 +52,7 @@
         ip = [int(i) for i in ip]

         # If ip[0] >= 0 and ip[0] <= 127 then the IP address is in class A
-        if ip[0] >= 0 and ip[0] <= 127:
+        if ip[1] >= 0 and ip[0] <= 127:
             return "A"

         # If ip[0] >= 128 and ip[0] <= 191 then the IP address is in class B
```

Mutant 18

```
--- src/ip_checker.py
+++ src/ip_checker.py
@@ -52,7 +52,7 @@
         ip = [int(i) for i in ip]

         # If ip[0] >= 0 and ip[0] <= 127 then the IP address is in class A
-        if ip[0] >= 0 and ip[0] <= 127:
+        if ip[0] > 0 and ip[0] <= 127:
             return "A"

         # If ip[0] >= 128 and ip[0] <= 191 then the IP address is in class B
```

Mutant 19

```
--- src/ip_checker.py
+++ src/ip_checker.py
@@ -52,7 +52,7 @@
         ip = [int(i) for i in ip]

         # If ip[0] >= 0 and ip[0] <= 127 then the IP address is in class A
-        if ip[0] >= 0 and ip[0] <= 127:
+        if ip[0] >= 1 and ip[0] <= 127:
             return "A"

         # If ip[0] >= 128 and ip[0] <= 191 then the IP address is in class B
```

Mutant 20

```
--- src/ip_checker.py
+++ src/ip_checker.py
@@ -52,7 +52,7 @@
         ip = [int(i) for i in ip]

         # If ip[0] >= 0 and ip[0] <= 127 then the IP address is in class A
-        if ip[0] >= 0 and ip[0] <= 127:
+        if ip[0] >= 0 and ip[1] <= 127:
             return "A"

         # If ip[0] >= 128 and ip[0] <= 191 then the IP address is in class B
```

Mutant 22

```
--- src/ip_checker.py
+++ src/ip_checker.py
@@ -52,7 +52,7 @@
         ip = [int(i) for i in ip]

         # If ip[0] >= 0 and ip[0] <= 127 then the IP address is in class A
-        if ip[0] >= 0 and ip[0] <= 127:
+        if ip[0] >= 0 and ip[0] <= 128:
             return "A"

         # If ip[0] >= 128 and ip[0] <= 191 then the IP address is in class B
```

Referências

Python Software Foundation. Documentação do Python 3.7+. Disponível em: https://www.python.org/. Acesso em: 2 set. 2024.

Python Software Foundation. python3-venv. Disponível em: https://docs.python.org/3/library/venv.html. Acesso em: 2 set. 2024.

pytest. pytest: Documentation. Disponível em: https://docs.pytest.org/en/stable/. Acesso em: 2 set. 2024.

PyPI. pytest-cov. Disponível em: https://pypi.org/project/pytest-cov/. Acesso em: 2 set. 2024.

GitHub. Mutmut - Python Mutation Tester. Disponível em: https://github.com/boxed/mutmut. Acesso em: 2 set. 2024.

YouTube. How to use pytest. Disponível em: https://www.youtube.com/watch?v=FbMpoVOorFI. Acesso em: 5 set. 2024.

GitHub. StevaoAA: Cal Python. Disponível em: https://github.com/stevaoaa/cal_python. Acesso em: 5 set. 2024.

GitHub. Pytest Setup and Teardown Example. Disponível em: https://github.com/Pytest-with-Eric/pytest-setup-teardown-example. Acesso em: 87777 set. 2024.