

## Template para Estudo de Caso com Testes Estruturais

### Conteúdo:

- Processo de Engenharia de Software;
- Etapas da atividade de teste no processo de Engenharia de Software: planejamento/plano de teste, projeto de casos de teste, execução ,análise e aperfeiçoamento;
- Técnica: teste estrutural ou teste de caixa branca;
- Ferramentas de escrita de casos de teste e cobertura de código;

**Facilitador:** Docente nas disciplinas de Engenharia de Software ou áreas afins.

**Objetivos de Aprendizagem:** Deseja-se que ao final desse estudo o aluno seja capaz de:

- Compreender o processo de engenharia de software e a sua importância na construção de um produto de software sustentável;
- Compreender os princípios fundamentais dos testes estruturais e seu papel no ciclo de desenvolvimento de software;
- Ser capaz de criar casos de **teste estruturais detalhados com base em critérios de fluxo de controle e/ou fluxo de dados**;
- Conhecer ferramentas de desenvolvimento de testes estruturais e cobertura de código;
- Conhecer ferramentas de geração de dados de teste;

- Realizar testes estruturais de acordo com planos de teste e documentar os resultados;

### **Estratégias de Uso:**

- É desejável que o facilitador faça uma breve apresentação sobre o processo de engenharia de software (Figuras 1 e 2);
- É desejável que o facilitador faça uma breve apresentação sobre as etapas da atividade de teste (Figuras 3 e 4);
- Apresentar o modelo de estudo de caso Estória Estrutural - Teste de Unidade Método de Desconto, disponível em <https://docs.google.com/document/d/1DM-ARr2XJHJJxnhRJLf6E-IQNsj6cULhWZ16RvcJyuQ/edit?usp=sharing>;
- Apresentar definições de teste estrutural ou caixa branca (Tópico 3. Teste Estrutural ou Caixa Branca: Definição deste Template);
- Reunir os alunos em grupos (por projeto), e sugerir que eles selecionem, em seus projetos ou em outros projetos, um ou mais **trechos de código**, para o planejamento e criação de testes estruturais;
- Sugerir aos alunos, o uso de ferramentas para a abstração de código fonte em grafo ou fluxos (<https://codetoflow.com/>);
- Apresentar aos alunos a estruturas genéricas de Plano de Teste e Projeto de Caso de Teste (Tópicos 4);
- Apresentar aos alunos as ferramentas de escrita de casos de teste e análise de cobertura de código;

**Requisitos mínimos:** Conhecimento em lógica de programação, algoritmos, estruturas de dados e alguma linguagem de programação.

## GUIA

1. Processo de Engenharia de Software	4
2. Processo e Etapas da Atividade de Teste	6
3. Teste Estrutural ou Caixa Branca: Definição	7
4. Planejamento/Plano de Teste e Projeto de Casos de Teste	10
5. Cobertura de Código / Cobertura de Teste	13
6. Sugestões	20
7. Referências	21

# 1. Processo de Engenharia de Software

## 2.1 Um modelo de processo genérico

No Capítulo 1, *processo* foi definido como um conjunto de atividades de trabalho, ações e tarefas realizadas quando algum artefato de *software* deve ser criado. Cada uma dessas atividades, ações e tarefas se aloca dentro de uma metodologia ou modelo que determina sua relação com o processo e uma com a outra.

O processo de *software* está representado esquematicamente na Figura 2.1. De acordo com a figura, cada atividade metodológica é composta por um conjunto de ações de engenharia de *software*. Cada ação é definida por um *conjunto de tarefas*, o qual identifica as tarefas de trabalho que devem ser completadas, os artefatos de *software* que serão produzidos, os fatores de garantia da qualidade que serão exigidos e os marcos utilizados para indicar progresso.

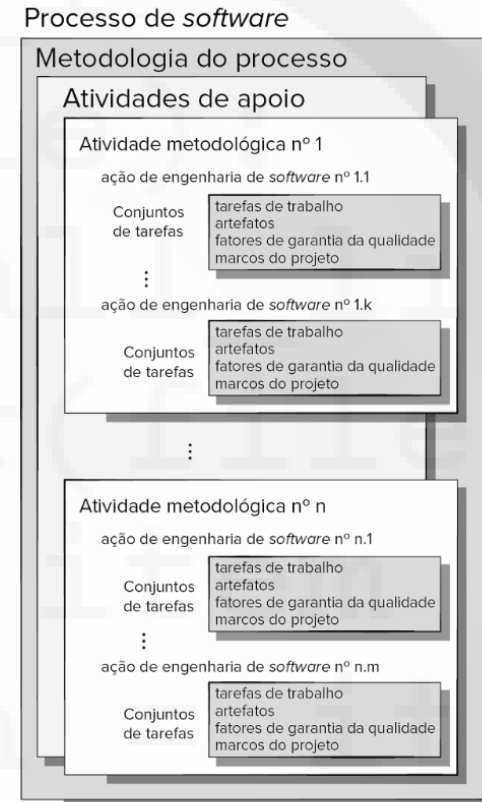


Figura 1: Figura 2.1 Engenharia de Software - 9ª Edição (PRESSMAN et al., 2021)

**Figura 2.1**

Uma metodologia do processo de *software*.

Como discutido no Capítulo 1, uma metodologia de processo genérica para engenharia de *software* estabelece cinco atividades metodológicas: **comunicação, planejamento, modelagem, construção e entrega**. Além disso, um conjunto de atividades de apoio é aplicado ao longo do processo, como o **acompanhamento e o controle do projeto, a administração de riscos, a garantia da qualidade, o gerenciamento das configurações, as revisões técnicas**, entre outras.

Um aspecto importante do processo de *software* ainda não foi discutido. Este aspecto – chamado de *fluxo de processo* – descreve como são organizadas as atividades metodológicas, bem como as ações e tarefas que ocorrem dentro de cada atividade em relação à sequência e ao tempo, como ilustrado na Figura 2.2.

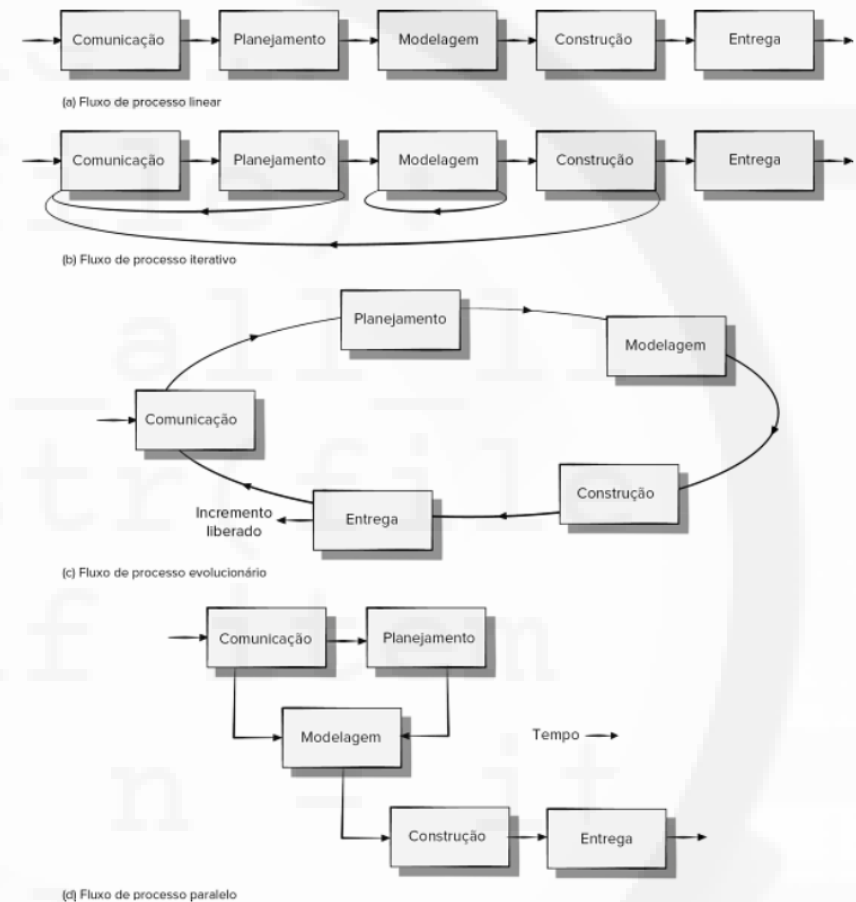


Figura 2: Figura 2.2 Engenharia de Software - 9ª Edição (PRESSMAN et al., 2021)

## 2. Processo e Etapas da Atividade de Teste

De modo geral, as etapas da atividade de teste são: planejamento/plano de teste, projeto de caso de teste, execução e análise.



Figura 3: Figura 8.3 Modelo do processo de teste de software 10ª Edição (SOMMERVILLE, 2019)

Uma boa prática ao desenvolver um bom conjunto de testes, ou seja, o que irá revelar o maior número de defeitos, é construí-los paralelamente com as etapas do desenvolvimento como sugere a figura abaixo que traz o Modelo V, que faz parte dos modelos de Ciclo de Desenvolvimento de Software (SDLC, do inglês *software development life cycle*), que traz um exemplo prático coerente ao processo processo de construção do software.

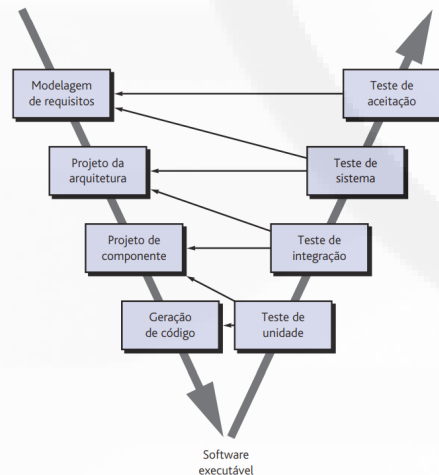


Figura 4: Figura 4.2. Modelo V Engenharia de Software - 8ª Edição (PRESSMAN; MAXIM, 2016)

### 3. Teste Estrutural ou Caixa Branca: Definição

O teste estrutural ou caixa branca é uma técnica de teste que visa exercitar toda estrutura do código fonte implementado para determinada funcionalidade.

Dessa forma, é fornecido ao responsável pela realização do teste, um trecho de código ou módulo a ser testado e também é esperado que o mesmo tenha o conhecimento mínimo necessário da estrutura a ser testada.

A ideia principal desse tipo de teste é fazer com que o caso de teste implementado, exercite pelo menos uma vez a estrutura implementada (possuindo ela  $n$  combinações de execução), garantindo o bom funcionamento da implementação esperada e/ou revelando comportamentos inesperados que devem ser corrigidos.

Para facilitar a análise da estrutura interna do código são utilizados os critérios de teste:

- Grafo de Fluxo de Controle (GFC): abstrair a complexidade do código e gerar um grafo genérico que o representa;
- Fluxo de dados: atribuição de variáveis que podem estar realizando algum desvio condicional ou sendo utilizada em cálculos;

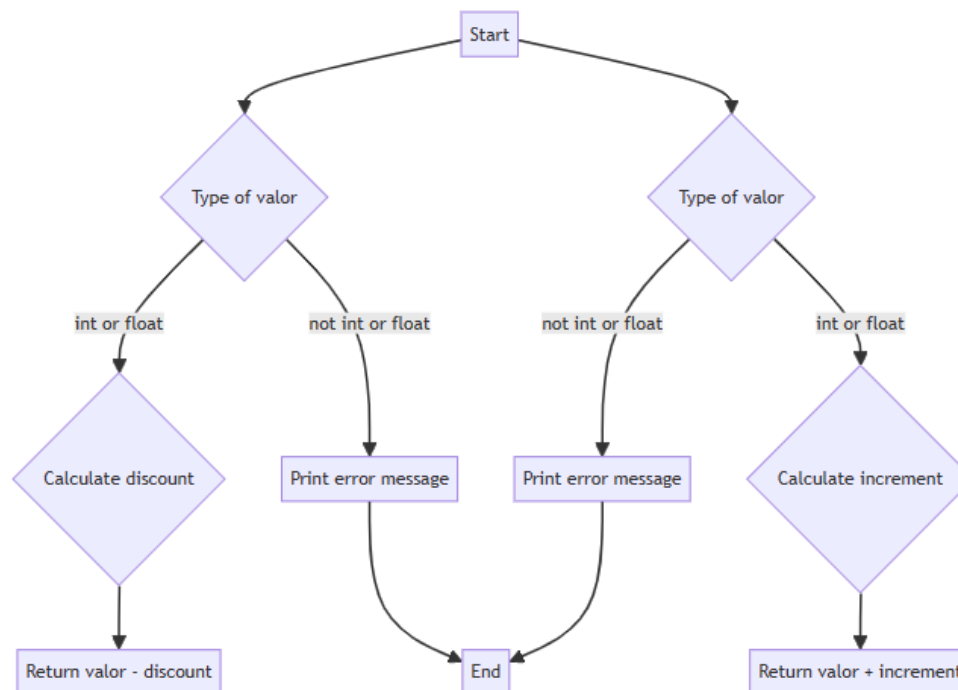
Tomemos como exemplo, a seguinte estrutura em python gerado pela ferramenta (<https://codetoflow.com/>) que abstrai o código fonte em estruturas mais simples (diagrama de fluxos, diagrama sequencial entre outros).

Como o teste é estrutural, o responsável pela realização deste terá em mãos, além da especificação, o código fonte da funcionalidade.

No exemplo abaixo, dada a especificação da [Estória Estrutural - Teste de Unidade Método de Desconto - Documentos Google](#) é fornecido também um código fonte como demonstrado nas figuras abaixo.

### Estrutura da Classe Operações

```
class Operacoes:  
  
    def calcula_preco_com_desconto(self, valor, porcentagem):  
        desconto = (valor * porcentagem) / 100  
        return valor + desconto  
  
    def calcula_preco_com_acrescimo(self, valor, porcentagem):  
        acrescimo = (valor * porcentagem) / 100  
        return valor + acrescimo
```



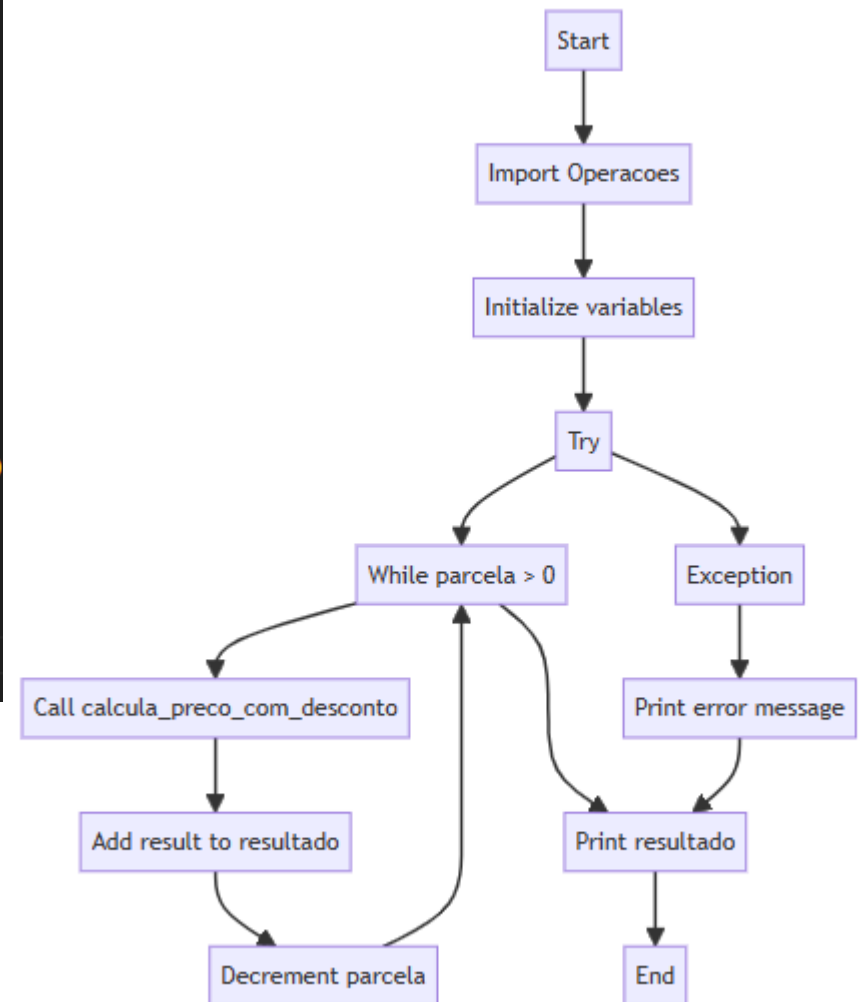


## Estrutura da classe Main

```
from operacoes import Operacoes

valor = 100.00
porcentagem = 20
# Constroi o objeto da classe Operações
operacoes = Operacoes()
resultado = 0

# São 10 parcelas fixas com 20% de desconto fixo e valor variável de 100.00
parcela = 10
try :
    while parcela > 0:
        resultado += operacoes.calcula_preco_com_desconto(valor, porcentagem)
        parcela-=1
        print(resultado)
except Exception as e:
    print(e)
```




## 4. Planejamento/Plano de Teste e Projeto de Casos de Teste

O planejamento ou plano de teste e o projeto de casos de teste são fatores que propõem uma melhor rastreabilidade e segurança de todo o processo de teste.

O IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos), sugere um padrão (“ISO/IEC/IEEE 29119-3:2021(en), Software and systems engineering — Software testing — Part 3: Test documentation”, [s.d.]) de planejamento e especificação dos casos de teste que pode ser adaptado para cada organização ou projeto.

As tabelas abaixo são uma simplificação do modelo sugerido pela IEEE.

Planejamento / Plano de Teste - Método de Desconto					
O que testar?	Como testar?	Qual a base de dados?	Quem irá testar?	Cronograma	Outros(riscos, requisitos do ambiente)
Ex.: Operação de Calcular Preço com Desconto;	Qual critério de teste?  <b>Sugestão:</b> grafo de fluxo de controle	Conjunto de dados que serão utilizados como entrada e formato  Ex.: csv, json, txt, docx, xls, etc.	Nome do Desenvolvedor ou Testador	Dia XX/XX/2023 a Dia XX/XX/2023	Riscos: Não de Aplica (N/A). Requisitos do Ambiente: <ul style="list-style-type: none"><li>- IDE(vscode/pycharm)</li><li>- python versão igual ou superior a 3.9</li><li>- gerenciador pip</li><li>- pytest</li><li>- coverage</li></ul>

## Projeto de Casos de teste - Método de Desconto

ID	Módulo	Descrição	Roteiro	Comportamento esperado	Comportamento real, evidências e incidentes.	Responsável	Status
1	Classe Operações	Método Calcula Preço com Desconto - Entrada positiva	Inserir entrada <b>valor positivo</b>	Retornar cálculo do preço com o desconto.  Ex: valor = 100 porcentagem = 20 %  resultado = 80	O resultado esperado era 80 e retornou 120.	João	Em andamento com erro.
2	Classe Operações	Método Calcula Preço com Desconto - Entrada negativa	Inserir entrada com <b>valor negativo</b>	Informar mensagem 'entrada inválida'.	Para a entrada negativa, houve a mensagem 'entrada inválida', como demonstrado na imagem: <b>teste_entrada_negativa.jpg</b>	Maria	Concluído
3	Classe Operações	Método Calcula Preço com	Inserir entrada com <b>valor nulo</b>	Informar mensagem	Para a entrada negativa, houve a	João	Concluído

		Desconto - Entradas vazia		‘entrada inválida’.	mensagem ‘entrada inválida’, como demonstrado na imagem: <b>teste_entrada_vazia.jpg</b>		
4	Classe Operações	Método Calcula Preço com Desconto - Entrada texto	Inserir entrada de <b>tipo string</b>	Informar mensagem ‘entrada inválida’.	Para a entrada de texto, houve a mensagem ‘entrada inválida’, como demonstrado na imagem: <b>teste_entrada_texto.jpg</b>	João	Concluído
5	Classe Operações	Método Calcula Preço com Desconto - Entrada valor 0	Inserir entrada com <b>valor 0</b>	Retornar cálculo do preço com o desconto.  Ex: valor = 0 porcentagem = 20 %  resultado = 0		Maria	Em andamento.
6	...	...					

## 5. Cobertura de Código / Cobertura de Teste

De modo geral, a cobertura de código ou cobertura de teste, é métrica em porcentagem de o quão sua implementação/aplicação está coberta por testes.

Essa métrica permite com que enxerguemos com mais precisão quais módulos/unidades da aplicação devem ser revisadas e consequentemente aprimoradas.

Na linguagem Python as bibliotecas `pytest`, `unittest` e `coverage`, são frameworks que auxiliam a escrita de testes e análise da cobertura de código.

A estrutura de diretórios padrão de testes em python é sugerida da seguinte forma:

```
pyproject.toml
[src/]mypkg/
  __init__.py
  app.py
  view.py
  test/
    __init__.py
    test_app.py
    test_view.py
    ...
```

>> diretório raiz do projeto além de possuir arquivos padrões da aplicação `[src/mypkg]`, terá também agora um ou mais diretórios para os testes `[src/test]`

>> cada diretório de teste deverá ter o arquivo de inicialização `[__init__.py]`

>> cada arquivo de teste deverá ter o prefixo `test_nomedoarquivo.py`

Para instalar as bibliotecas, você deve possuir o python instalado em sua máquina ([fazer download aqui](#)) e a seguir, instalar também o gerenciador de instalação do python pip ([fazer download aqui](#)).

Com isso, execute os seguintes comandos no terminal do projeto:

```
>> pip install -U pytest
>> pip install coverage
```

Para checar as instalações basta executar os seguintes comandos:

```
>> pytest --version
pytest 7.1.3

>> coverage --version
Coverage.py, version 7.3.2 with C extension
Documentation at https://coverage.readthedocs.io/en/7.3.2
```

Escrevendo alguns casos de teste da tabela com `unittest` em python no arquivo `test_operacoes.py` e `pytest`

```
# importação da biblioteca 'unittest' que possibilita o teste de unidade
import unittest

# importação da classe a ser testada
from operacoes import Operacoes

# Classe de teste que herda de unittest.TestCase
class TestOperacoes(unittest.TestCase):

    def test_calcula_preco_com_desconto(self):

        valor = 100

        porcentagem = 20

        operacoes = Operacoes()

        resultado = operacoes.calcula_preco_com_desconto(valor, porcentagem)

        # função que valida se a saída real é igual a saída esperada
        self.assertEqual(resultado, 80, 'FALHA: retorno do método não é compatível com a saída esperada!')

    def test_calcula_preco_com_acrescimo(self):
```

```
valor = 100

porcentagem = 20

operacoes = Operacoes()

resultado = operacoes.calcula_preco_com_acrescimo(valor, porcentagem)

# função que valida se a saída real é igual a saída esperada
self.assertEqual(resultado, 120, 'FALHA: retorno do método não é compatível com a saída esperada!')

#chamada da main(função principal onde é iniciada a execução do teste)
if __name__ == '__main__':
    unittest.main()
```

Para executar este código, basta escrever o seguinte comando em seu terminal:

```
>> python -m unittest
OU
>> pytest
```

Saída sem erros:



```
..
-----
Ran 2 tests in 0.000s

OK
```

Quando a saída não está coerente com a saída esperada:

```
Entrada inválida, o valor deve ser maior ou igual a 0 e do tipo inteiro ou decimal!
F.
=====
FAIL: test_calcula_preco_com_acrescimo (test.test_operacoes.TestOperacoes)
-----
Traceback (most recent call last):
  File "C:\Users\... \teste-de-software-um-guia-em-casos\estrutural\unitario\test\test_operacoes.py", line
25, in test_calcula_preco_com_acrescimo
    self.assertEqual(resultado, 120, 'FALHA: retorno do método não é compatível com a saída esperada!')
AssertionError: None != 120 : FALHA: retorno do método não é compatível com a saída esperada!

-----
Ran 2 tests in 0.001s

FAILED (failures=1)
```

Para obter comandos mais detalhados, basta acessar [Unittest — Estrutura de teste de unidade — Documentação do Python 3.12.0](#) ou [Get Started — pytest documentation](#)

Para verificar a cobertura de teste que tem a classe operações, basta escrever o seguinte comando em seu terminal:

```
>> coverage run -m unittest discover
```

OU

```
>> coverage run -m pytest
```

Para gerar o relatório de cobertura, basta executar o comando:

```
>> coverage report -m
```

Name	Stmts	Miss	Cover	Missing
-----	-----	-----	-----	-----
operacoes.py	11	3	73%	9, 14-15
test\__init__.py	0	0	100%	
test\test_operacoes.py	17	1	94%	29
-----	-----	-----	-----	-----
TOTAL	28	4	86%	

Name -> Nome do arquivo

Stmts -> Quantidade de Linhas

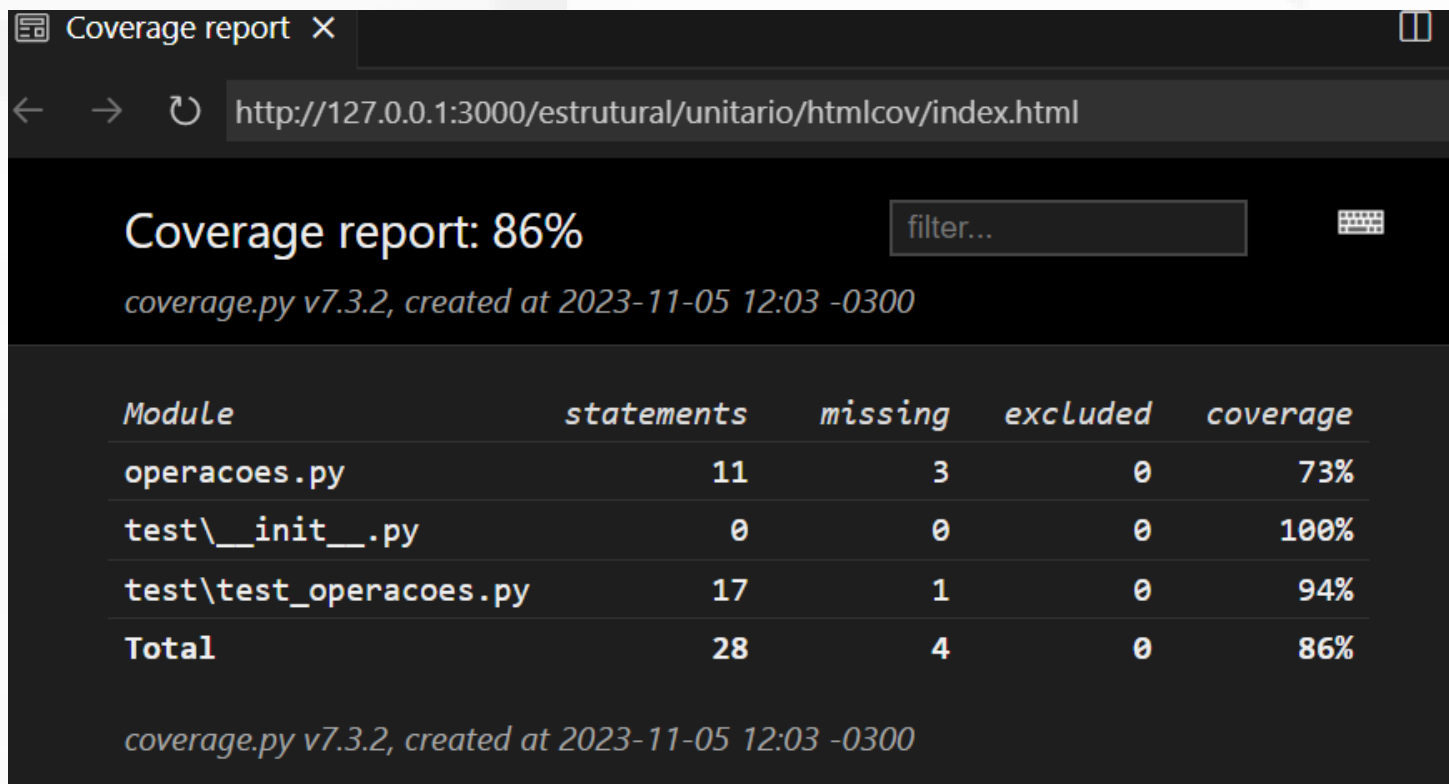
Miss - > Quantidade de Linhas Puladas

Cover -> Cobertura

Missing - > Linhas Puladas

Gerar relatório de cobertura em HTML:

```
>> coverage html
```



Coverage report: 86%

coverage.py v7.3.2, created at 2023-11-05 12:03 -0300

Module	statements	missing	excluded	coverage
operacoes.py	11	3	0	73%
test\__init__.py	0	0	0	100%
test\test_operacoes.py	17	1	0	94%
<b>Total</b>	<b>28</b>	<b>4</b>	<b>0</b>	<b>86%</b>

coverage.py v7.3.2, created at 2023-11-05 12:03 -0300

## 6. Sugestões

💡 Ferramentas para Teste Estruturais.

- Gerador de Grafos:
  - <https://codetoflow.com/>
  - [code2flow - online interactive code to flowchart converter](#)
  - [https://csacademy.com/app/graph\\_editor/](https://csacademy.com/app/graph_editor/)

💡 Sites de Geração de Dados para Teste:

- [generatedata.com](https://generatedata.com)
- [Mockaroo - Random Data Generator and API Mocking Tool | JSON / CSV / SQL / Excel](#)
- [Online test data generator for up to 100.000 Records \(onlinedatagenerator.com\)](#)
- [Cobbl: Mock Data Generator](#)
- [Random Mock Data Generator \(randomtools.io\)](#)

💡 Modelo de Documentação de Teste sugerido pela IEEE:

- <https://www.iso.org/obp/ui/en/#iso:std:iso-iec-ieee:29119:-3:ed-2:v1:en>

## 7. Referências

**Code to Flowchart.** Disponível em: <<https://codetoflow.com/>>. Acesso em: 5 nov. 2023.

**Coverage.py — Coverage.py 7.3.2 documentation.** Disponível em:

<<https://coverage.readthedocs.io/en/7.3.2/index.html>>. Acesso em: 5 nov. 2023.

**ISO/IEC/IEEE 29119-3:2021(en), Software and systems engineering — Software testing — Part 3: Test documentation.** Disponível em: <<https://www.iso.org/obp/ui/en/#iso:std:iso-iec-ieee:29119:-3:ed-2:v1:en>>. Acesso em: 5 nov. 2023.

PRESSMAN, R.; MAXIM, B. **Engenharia de Software - 8ª Edição.** [s.l: s.n.].

PRESSMAN, R. S. et al. **Engenharia de software.** 9ª edição ed. Porto Alegre, RS: AMGH, 2021.

**pytest: helps you write better programs — pytest documentation.** Disponível em:

<<https://docs.pytest.org/en/7.1.x/index.html>>. Acesso em: 5 nov. 2023.

SOMMERVILLE, I. **Engenharia de Software.** 10ª edição ed. [s.l.] Pearson Universidades, 2019.

**unittest — Unit testing framework.** Disponível em: <<https://docs.python.org/3/library/unittest.html>>. Acesso em: 5 nov. 2023.