Teste de Software

Instrutor: Tarik Ponciano

Senac Fecomércio Sesc

Links da Disciplina

1. Discord: https://discord.gg/wt5CVZZWJs

2. Drive: https://drive.google.com/drive/folders/1hOl0DaPeAor7gnhKBUIlZ5n8lLRDNvUY?usp=sharing

3. Github: https://github.com/TarikPonciano/Programador-de-Sistema-SENAC

Teste de Software– O que é?

Os testes representam uma etapa de extrema importância no processo de desenvolvimento de software, pois visam validar se a aplicação está funcionando corretamente e se atende aos requisitos especificados.

Nesse contexto existem diversas técnicas que podem ser aplicadas em diferentes momentos e de diferentes formas para validar os aspectos principais do software.

O teste de software é uma parte crucial do desenvolvimento de um software. Ele irá, como o nome sugere, testar e verificar se o software consegue entregar corretamente tudo que ele propõe.

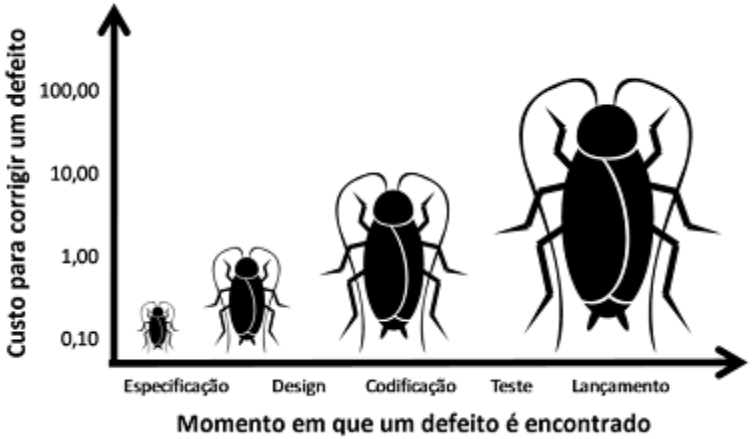
Esses testes podem ser realizados em qualquer uma das partes do software, desde a unidade pequena até seu funcionamento como um todo, analisando também o número de dados e a sua segurança.

Teste de Software– O que é?

•O teste de software é uma maneira de avaliar a qualidade da aplicação e reduzir o risco de falha em operação.
•Testar não consiste apenas em executar testes (executar o software e verificar os resultados). **Executar** testes é apenas umas das atividades.
•Planejamento, análise, modelagem e implementação dos testes, relatórios de progresso, resultado e avaliação da qualidade, também são partes de um **processo de testes**.

Testar software não é somente **verificar** *se os requisitos foram atendidos ou se as estórias de usuário (User Stories), bem como demais especificações, foram contempladas. Atribui-se ao teste de software também a* **validação***, ou seja, verificar se o sistema atenderá às necessidades do usuário e de outras partes interessadas em seu(s) ambiente(s) operacional(is).*

Teste de Software– Tipos



Teste de Software – Etapas

Para que o teste de software funcione, é preciso que os organizadores estabeleçam uma série de passos dentro de sua execução. O primeiro deles é o planejamento, que diz respeito ao momento em que será escolhido qual tipo de teste será realizado e em qual software.

Após isso, é feita a modelagem do teste, que além do planejamento, leva em consideração todos os aspectos da empresa que podem, de alguma forma, interferir no resultado dos testes. Por último, vem a execução.

Teste de Software – Tipos

Existem, de uma forma geral, cerca de 17 tipos diferentes de testes de software e a diferenciação deles serve para que consigam se adequar melhor ao tipo de softwares que eles irão testar.

Os tipos de testes de software são: de unidade, integração, operacional, positivo-negativo, regressão, caixa-preta, caixa-branca, funcional, interface, performance, carga, aceitação do usuário, volume, stress, configuração, instalação e segurança.

Teste de Software – Tipos

– Teste da caixa branca – utiliza o aspecto interno do programa/sistema, o código fonte, para avaliar seus componentes. Ele também é conhecido como teste orientado à lógica ou estrutural. Podem ser analisados itens como: fluxo dos dados, condição, ciclos etc. Na hora de implementá-lo é preciso verificar a criticidade, a complexidade, a estrutura e o nível de qualidade que se pretende obter do programa, envolvendo confiança e segurança;

– **Teste da caixa preta** – diferente do teste anterior, que prioriza os aspectos internos, o teste da caixa preta verifica aspectos externos. Os requisitos funcionais do sistema são avaliados. Não se observa o modo de funcionamento, sua operação, tendo como foco as funções que deverão ser desempenhadas pelo programa. Desse modo, avalia-se se um grupo de entrada de dados resultou nas saídas pretendidas, levando-se em consideração a especificação do programa. Ou seja, o que se esperava que o software deveria fazer. É conhecido também como técnica funcional;

Teste de Software – Tipos

– **Teste da caixa cinza** – esse tipo de teste une os dois anteriores, por isso o termo “cinza”. Avalia tanto os aspectos internos quanto os externos, de entrada e saída. Pode utilizar-se de engenharia reversa;

– **Teste de regressão** – esse consiste em realizar testes a cada versão de um software, onde se modificam-se funcionalidades. Desse modo, evita-se que erros que foram corrigidos antes no software antes voltem a aparecer na hora de se incrementar algo novo a ele.

– **Teste de unidade** – testa-se unidades menores de um software, de modo isolado, para ver se todas funcionam adequadamente;

– **Teste de integração** – depois das unidades testadas, realiza-se uma verificação se elas funcionam juntas, integradas. Pode ocorrer delas apresentarem incompatibilidades ao funcionarem em conjunto, mesmo após terem sido aprovadas no teste de unidade;

Teste de Software – Tipos

– **Teste de carga** – esse teste é feito para avaliar os limites de uso do software, o quanto ele suporta em volume de informações, tráfego etc. sem que apresente erros;

– **Teste de usabilidade** – esse teste é feito por um pequeno grupo de usuários para ver se o software satisfaz as suas necessidades. Nesse teste analisa-se como o usuário usa o sistema, verificando onde ele tem mais dificuldade. Ouve-se também suas impressões, porém é preciso confrontá-las com as observações do avaliador;

– **Teste de stress** – aqui leva-se o software ao seu limite de potência e funcionamento, para mais ou para menos, de modo a avaliar em qual ponto ele deixa de funcionar adequadamente. Isso é feito para verificar se suas especificações máximas ou mínimas de uso estão corretas.

Plano de Testes – O que é?

O plano de teste é um dos documentos produzidos na condução de um projeto. Ele funciona como:

- Um ‘integrador’ entre diversas atividades de testes no projeto;
- Mecanismo de comunicação para os stakeholders (i.e. a equipe de testes e outros interessados);
- Guia para execução e controle das atividades de testes.

Plano de Testes – O que é?

O plano de teste, que pode ser elaborado pelo gerente de projeto ou gerente de testes, visa planejar as atividades a serem realizadas, definir os métodos a serem empregados, planejar a capacidade necessária, estabelecer métricas e formas de acompanhamento do processo. Nesse sentido, deve conter:

- Introdução com identificação do projeto (definições, abreviações, referências), definição de escopo e objetivos;
- Conjunto de requisitos a serem testados;
- Tipos de testes a serem realizados e ferramentas utilizadas;
- Recursos utilizados nos testes;
- Cronograma de atividades (e definição de marcos de projeto).

Plano de Testes – O que é?

Em outras palavras, um plano de teste deve definir:

1. Os itens a serem testados: o escopo e objetivos do plano devem ser estabelecidos no início do projeto.

2. Atividades e recursos a serem empregados: as estratégias de testes e recursos utilizados devem ser definidos, bem como toda e qualquer restrição imposta sobre as atividades e/ou recursos.

3. Os tipos de testes a serem realizados e ferramentas empregadas: os tipos de testes e a ordem cronológica de sua ocorrência são estabelecidos no plano.

4. Critérios para avaliar os resultados obtidos: métricas devem ser definidas para acompanhar os resultados alcançados.

Perceba que o planejamento é necessário a fim de antecipar o que pode ocorrer e, portanto, provisionar os recursos necessários nos momentos adequados. Isto significa coordenar o processo de teste de modo a perseguir a meta de qualidade do produto (sistema de software).

Plano de Testes – Exemplos

https://www.cin.ufpe.br/~gta/rup-vc/extend.formal_resources/guidances/examples/resources/test_plan_v1.htm

https://pt.slideshare.net/LeandroRodrigues221/exemplo-de-plano-de-testes

https://github.com/danielle8farias-zz/plano-de-testes

Testes Unitários – O que é

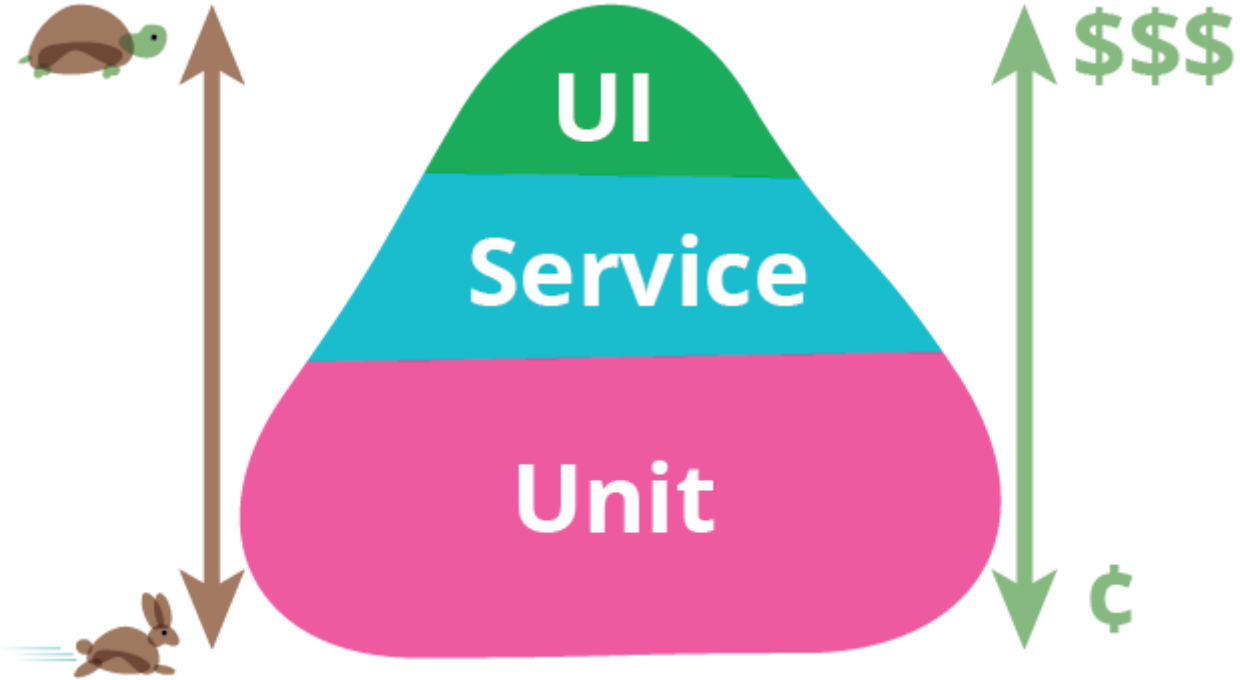
Basicamente, **testes unitários ou testes de unidades aferem a qualidade do código em sua menor fração.**

Eles também podem ser identificados como testes de algoritmos e, por isso, não dependem do uso de recursos externos.

Sendo assim, são os testes mais baratos da pirâmide de testes, como você verá a seguir. No entanto, geram menos valor para a empresa, visto que sozinhos eles não garantem a corretude do código.

Mas, em síntese, os testes unitários são os mais usados em qualquer projeto e, por isso, quem está começando a dar os primeiros passos na programação deve ficar atualizado sobre este tema, aprofundando os seus estudos.

Testes Unitários – O que é



Senac
Fecomércio
Sesc

Testes Unitários – O que é

Perguntar para que servem testes unitários ou qualquer outro teste automatizado é uma ótima pergunta, afinal existem diversas formas que aparentemente são mais rápidas de testar se minha função está fazendo o que deveria. Eu poderia simplesmente executar o código para verificar se está funcionando. Então, por que motivo eu vou escrever outro código para testar o meu código? O Que garante que o segundo código funciona? Quem testa o teste?

Testes unitários, assim como qualquer teste automatizados não servem principalmente para verificar se uma função específica está funcionando, mas sim para garantir que sua aplicação continue funcionando após alguma alteração em sua base de código.

Testes Unitários – O que é

Pode parecer tentador em um primeiro momento não escrever testes para uma função que você acaba de desenvolver, afinal, costuma-se escrever mais código para testar uma função, que o próprio código da função. Mas você deve lembrar que irá passar a maior parte do tempo de desenvolvimento de um sistema trabalhando em sua manutenção.

Sua aplicação em pouco tempo terá algumas centenas de funções sendo executadas, e em muitas vezes executando umas às outras, sua base de código fica enorme e logo torna-se humanamente impossível de ser testada de forma manual após qualquer alteração. Testes unitários na maioria das vezes levam apenas alguns segundos para testar toda sua aplicação.

Testes Unitários - Vantagens

•Eles não servem apenas para verificar se uma função está funcionando, mas sim para garantir que a função se mantenha em funcionamento após alguma alteração na base do código;

•Prevenir problemas futuros;

•Garantir um software de qualidade:

•Aumentar a produtividade da pessoa desenvolvedora;

•Aumentar a velocidade do código;

•Garantir a eficiência do escopo escolhido.

Testes Unitários - Vantagens

•Você economiza muito tempo nos testes de regressão.

•Você recebe um alerta sempre que, inadvertidamente, quebra um comportamento existente. Permitindo que você o enfrente imediatamente enquanto ainda está totalmente imerso no que você está trabalhando. Significa menos bugs escapando.

•Menos bugs escapando significa que você tem mais tempo para desenvolver valor.

•Trabalhando sem medo de quebrar o código existente sem saber, você possui mais recursos cognitivos preciosos. Significa que você será mais criativo e inovador, e capaz de criar soluções melhores.

•Você recebe documentação viva, respirando, nunca desatualizada – pelo menos quando você dá a cada teste unitário um nome significativo (mais sobre isso depois).

•Uma documentação sempre atualizada permite que você acelere os novos contratados.

•Trabalhando sem medo de quebrar o código existente sem saber, novos membros da equipe se tornam produtivos mais rapidamente.

•Menos bugs também significa que você reduzirá a carga da equipe de suporte e os liberará para se concentrar no sucesso do cliente ao invés de controlar os danos.

Testes Unitários - PyUnit

O PyUnit é uma biblioteca para realização de testes unitários em Python. É a forma mais difundida para realizar a prática de testes unitários pela comunidade Python.

O PyUnit foi criado com o intuito de trazer ao **Python** todos os recursos que já haviam no framework de teste unitário JUnit da linguagem **Java**, a ferramenta evoluiu de forma a explorar plenamente os recursos da linguagem e se adequar ao jeito pythonico de escrever código.

Por exemplo, quando precisamos testar se um método de consulta está buscando o registro certo, o mais comum é chamar esse método dentro da cláusula if __name__ == "__main__":, imprimir o resultado na tela e executar o módulo onde o método é declarado.

```
1  if __name__ == "__main__":
2  cliente_dao = ClienteDAO()
3  cliente = cliente_dao.find_by_id(10)
4  print(cliente.name())
5  # o resultado esperado é "Edson Arantes do Nascimento"
```

Testes Unitários - PyUnit

Em um projeto maior esse tipo de teste tende a se tornar inviável e inseguro a medida que a complexidade do módulo e da unidade aumentam, é natural sentir falta de uma metodologia de testes, que permita medir a cobertura de teste do código, tornando fácil criar os cenários nos quais as falhas serão percebidas. Os testes unitários suprem essas e outras carências do processo de teste de código.

```
1  def test_find_by_id(self):
2          cliente_dao = ClienteDAO()
3          cliente = cliente_dao.find_by_id(10)
4          self.assertEqual(cliente_dao, 'Edson Arantes do Nascimento')
```

Testes Unitários - PyUnit

Em um projeto maior esse tipo de teste tende a se tornar inviável e inseguro a medida que a complexidade do módulo e da unidade aumentam, é natural sentir falta de uma metodologia de testes, que permita medir a cobertura de teste do código, tornando fácil criar os cenários nos quais as falhas serão percebidas. Os testes unitários suprem essas e outras carências do processo de teste de código.

```
1  def test_find_by_id(self):
2          cliente_dao = ClienteDAO()
3          cliente = cliente_dao.find_by_id(10)
4          self.assertEqual(cliente_dao, 'Edson Arantes do Nascimento')
```

Testes Unitários - PyUnit

Para criar testes com PyUnit é necessário criar uma Classe de Testes que deve extender a classe TestCase presente no módulo unittest.

O PyUnit localiza os métodos de teste pelo prefixo test em seu nome. Por fim é necessário chamar a função main do módulo unittest dentro da expressão if __name__ == "__main__": do módulo para executar os testes criados, como ilustrado abaixo:

```
import unittest

class TestClass(unittest.TestCase):

    def test_meu_metodo(self):
        self.assertEqual(valor_esperado , valor_real, "mensagem caso o teste falhe")

if __name__ == "__main__":
    unittest.main()
```

Os métodos assert (do inglês afirmar) são responsáveis por verificar se o resultado gerado durante o teste é igual ao esperado, caso contrário o teste resultará em uma falha e exibirá a mensagem, caso definida.

Testes Unitários - PyUnit

Por exemplo, considere que queremos testar isoladamente o seguinte método frete_gratis da classe Compra:

```
class Compra:
    def frete_gratis(self, valor):
        return valor >= 150
```

Criamos então a classe CompraTest dentro da pasta tests, como o método test_frete_gratis, de acordo com o código abaixo.

```
import unittest
from compra import Compra

class CompraTeste(unittest.TestCase):
    def test_frete_gratis(self):
        nova_compra = Compra()
        self.assertTrue(nova_compra.frete_gratis(200))

if __name__ == "__main__":
    unittest.main()
```

Senac Fecomércio Sesc

Testes Unitários - PyUnit

assertEqual(expected_value,actual_value)	Asserts that expected_value == actual_value
assertTrue(result)	Asserts that bool(result) is True
assertFalse(result)	Asserts that bool(result) is False
assertRaises(exception, function, *args, **kwargs)	Asserts that function(*args, **kwargs) raises the exception

Testes Unitários - PyUnit

```python
import math

def is_prime(num):
    '''Check if num is prime or not.'''
    for i in range(2,int(math.sqrt(num))+1):
        if num%i==0:
            return False
    return True
```

```
>>> from prime_number import is_prime
>>> is_prime(3)
True
>>> is_prime(5)
True
>>> is_prime(12)
False
>>> is_prime(8)
False
>>> assert is_prime(7) == True
```

Senac
Fecomércio
Sesc

Testes Unitários - PyUnit

```python
import unittest
# import the is_prime function
from prime_number import is_prime
class TestPrime(unittest.TestCase):
    def test_two(self):
        self.assertTrue(is_prime(2))
    def test_five(self):
        self.assertTrue(is_prime(5))
    def test_nine(self):
        self.assertFalse(is_prime(9))
    def test_eleven(self):
        self.assertTrue(is_prime(11))
if __name__=='__main__':
    unittest.main()
```

```
$ python test_prime.py
```

```
Output
....
----------------------------------
Ran 4 tests in 0.001s
OK
```

Senac Fecomércio Sesc

Testes Unitários - PyUnit

```
import unittest
from prime_number import is_prime
class TestPrime(unittest.TestCase):
    def test_prime_not_prime(self):
        self.assertTrue(is_prime(2))
        self.assertTrue(is_prime(5))
        self.assertFalse(is_prime(9))
        self.assertTrue(is_prime(11))
```

Senac
Fecomércio
Sesc

Testes Unitários - Exercícios

1. Crie uma função que checa se um número é par. Se for par ela retorna True, se não for ela retorna False. Crie 5 testes unitários para checar essa função. Após testar a primeira vez, quebre a função(mude o valor que ela retorna) e observe o resultado.

2. Crie uma classe Funcionário que deve conter, pelo menos, os atributos nome e idade e um método apresentar() em que é retornada uma mensagem com o nome e idade do funcionário.
   Ex: “Olá meu nome é Fulano, tenho X anos”

3. Realize um teste unitário em alguma operação de banco de dados.

Obrigado!!

Senac Fecomércio Sesc

https://www.objective.com.br/insights/testes-de-software/

https://www.devmedia.com.br/guia/tecnicas-e-fundamentos-de-testes-de-software/34403

https://cwi.com.br/blog/o-que-e-teste-de-software-por-que-e-necessario/

https://www.devmedia.com.br/plano-de-teste-um-mapa-essencial-para-teste-de-software/13824

https://www.devmedia.com.br/e-ai-como-voce-testa-seus-codigos/39478

Senac Fecomércio Sesc

https://coodesh.com/blog/dicionario/o-que-sao-testes-unitarios/

https://dayvsonlima.medium.com/entenda-de-uma-vez-por-todas-o-que-s%C3%A3o-testes-unit%C3%A1rios-para-que-servem-e-como-faz%C3%AA-los-2a6f645bab3

https://www.devmedia.com.br/teste-unitario-com-pyunit/41233

Senac Fecomércio Sesc