



Indo Mais Fundo em Teste

Testes Unitários:

Uma das verdades sobre testes unitários é que mesmo eles não são bons como indicadores de que o software tem qualidade, não se trata de serem ou terem o objetivo de serem um indicador de positivo, mas sim seu oposto. Ele serve muito melhor como indicador negativo que aponta os pontos de melhoria e fragilidade dentro de um código, e seu objetivo, auxiliar num processo de desenvolvimento saudável.

Em geral, o desenvolvimento sem teste inicialmente é mais rápido, mas conforme vai evoluindo o desenvolvimento do software, manutenção, refatoração, novas versões, o tempo de desenvolvimento e as horas gastas vão crescendo exponencialmente, enquanto um projeto onde os testes são modelados desde o início tem seu início muito mais lento, porém com o tempo ele vai se estabilizando.

Um copo de água como software:

Dentro da Física temos um conceito básico da termodinâmica, a entropia, que diz a respeito da quantidade de desordem em um sistema, se fosse lembrar um pouco poderíamos lembrar do exemplo que eu acredito: primeiro temos um copo de água em um estado original, em um segundo momento esquentamos essa água, então as moléculas de água começam a se esquentando, vão se agitando e causando desordem e temos nosso segundo estado, então para finalizar, depois tiramos a água e esperamos ela esfriar até a temperatura original, seria certo afirmar que temos de volta o primeiro estado, não, uma das características da **Entropia é que quando gerada desordem dentro de um sistema, ele não consegue retornar ao estado original**, na verdade é gerado um novo estado, com uma nova organização ou melhor, mais desordenado, no caso do copo de água agora temos um terceiro estado que se parece com o primeiro, mas quando olhado a fundo as moléculas se organizam de forma diferente.

Agora vamos trazer esse contexto para dentro de desenvolvimento de software:

- 1º Estado: O software funciona normalmente como esperado, considerado um estado estável.
- 2º Estado: Um engenheiro realiza uma implementação de uma nova feature, no software gerando mais desordem dentro do código.
- 3º Estado: É aqui que os testes brilham aos olhos, para facilitar o processo de um estado estável, que ainda não é como o primeiro, ele tem mais desordem, ele é um código, é um código que tem mais código e mais regras e por isso se torna mais desorganizado. Todavia, os testes ajudam a chegar nesse terceiro estado mais estável em menor tempo, do que teríamos sem eles.

Eai Sabem o que é regressão?

Provavelmente, vão seja deve ter ouvido falar dos teste de regressão, se não tudo bem. também. Se fossemos explicar o que e testes de regressão seria: “peguei todos os testes que criamos antes dessa modificação e rode tudo de novo para garantir que tudo esta funcionando normalmente e não a bug novos no software’.

Porem gostaria de chamar a atenção para a palavra regressão, muitas pessoas conhecem os testes de regressão, mas não tem ideia do que significa a palavra regressão e isso e muito comum saibam vocês. Regressão é quando feature, para de funcionar após um determinado evento, muitas vezes poderemos ver referencias a regressões e bugs sendo usados, por que até então muitas vezes acabam sendo sinônimos.

Então quanto estamos falando de testes de regressão, estamos falando de testes contra regressões, e isso e uma característica muito importante para um desenvolvimento saudável.

A Armadilha nos Testes de Cobertura:

Primeira o que são os testes de cobertura, é um teste que avalia e retorna um porcentual de quanto do seu código e testado utilizando os testes unitários ou testes de integração. ou seja quanto do seu código realmente esta sendo testado

Mas os testes de cobertura são uma das grandes armadilhas, que muitas pessoas decidem acreditar, se eu perguntasse: você acredita que ter 100% dos de testes de cobertura para o seu código, é um indicativo que o seu código tem qualidade, você pode responder como a maioria que, sim, alias são 100%, isso tem que indicar algo positivo. E no caso chegamos a armadilha ter 100% de cobertura de teste, não diz muita coisa, no sentido positivo da coisa, na maioria dos casos pode indicar coisas ruins, Okay, Okay, então o que você imagina que poderiam ser essas coisa ruins? pense um pouco...E então já descobriu o que é?.

Mas antes de entregarmos a resposta, vamos falar um pouco sobre os testes de cobertura, para que você conheça um pouco mais os tipos. Ou seja vamos deixar você sofrendo um pouco, com essa crise existencial, voltadas a testes.("hihihi!" - nota do autor).

Conhecendo testes de Cobertura

O primeiro tipo de teste que conversaremos, é o mais comum conhecido como cobertura de código, avaliada a quantidade de código executado para a quantidade pela quantidade de linhas totais do codigo. com podemos ver pela seguinte formula:

$$X = \frac{\Delta l}{l}$$

onde temos:

X - Porcentual de cobertura

Δl - Linhas executadas

l - linhas totais do arquivo

Vamos analisar o seguinte código:

```
def equal_number_four(number:int) -> bool:

    if int(number) == 4:
        return True
    else:
        return False
```

e temos os seguintes testes:

```
def test_equal_number_four_true():

    four= 4

    result = equal_number_four(number = four)
    assert(result, True)

# passa no testes
```

No nosso exemplo, qual seria a cobertura do nosso código:

temos que:

- um total de 5 linhas de códigos lembrando que não devemos considerar as linhas vazias ou linhas de comentário, pois as mesmas não devem ser executadas. (ou pelo menos não deveriam)
- ΔI - E um total de 3 linhas. pois como o testes retornou verdadeiro ele não passou pelo else,

então a nossa cobertura é de:

$$60\% = 0,6 = \frac{3}{5}$$

E claro que poderíamos realizar dois testes um que passar, pelo if quanto pelo else, mas entretanto a uma situação mais interessante que seria comprimir a estrutura e usar um operador ternário:

```
def equal_number_four(number:int) -> bool:

    return True if int(number) == 4 else False
```

▼ entendo operador ternário python

Operador ternário do python segue a seguinte estrutura:

resposta para se **if** condição **else** retorno para o se não

Veja que agora os nossos testes passariam com 100% das linhas de código, muitas vezes, ao invés de realizar vários testes, e necessário avaliar se no código de produção, não existe nenhum ganho de melhoria. e que todas as estruturas presentes, são realmente necessárias.

Cobertura por agencias

Diferente do método de cobertura por código, esse métrica é relativamente melhor, esses testes se foca em estruturas de controle como if e switch, e o seu calculo e similar outro

$$Y = \frac{\Delta f}{f}$$

onde temos:

Y- Porcentual de cobertura

Δf - total de estruturas que o teste passa

I - total de estruturas de controle no código

Vamos pegar o exemplo anterior para entender melhor:

```
def equal_number_four(number:int) -> bool:

    if int(number) == 4:
        return True
    else:
        return False
```

```
def test_equal_number_four_true():

    four= 4

    result = equal_number_four(number = four)
    assert(result, True)

# passa no testes
```

Para esses testes qual seria a cobertura pela métrica de agencia:

Temos:

- 2 estruturas de controle dentro do código ou seja dois caminhos, se for um ele vai para um caminho e se não for ele vai para outro, ou seja temos dois caminhos
- Desses 2 Caminhos o nosso testes só passa por um deles

Então o nossa cobertura fica?

$$50\% = 0,5 = \frac{1}{2}$$

O que não te contaram sobre os testes de cobertura:

Primeiro antes de jogar pedras no pobre testes de cobertura, tenho que aponta que os testes de cobertura com os testes de unidade, não são um bom indicador positivo, mas sim um ótimo indicador negativo, pois quanto temos níveis muito baixo por exemplo 50% ou 60% significa que grande parte ainda no nosso código ainda não foi testada, isso deveria ser alerta, para informar que ainda existam partes importantes no nosso código que ainda não foram devidamente testadas.

Agora se nos tivermos 100% de testes, por muitos motivos podemos dizer que não significa, muito pois primeiramente testes tem limitações, segundo eventualmente nem todos testes tem valor, os testes não nascem iguais, cada testes deve ter um valor específico para o código. muitas vezes o indicativo 100% pode indicar que nos estamos realizando em muitos casos testes triviais e frágeis, que não muito valor, na verdade esses tipos de testes com o tempo podem problemas, acionando em falsos positivos e negativos. e aí está a nossa resposta.

Outras das coisas que limitam os testes de cobertura é essa e uma limitação dos testes unitários e até mesmo de integração, que num podemos testar todas as entradas e saídas, existe uma infinidade de entradas que um usuário pode colocar, e para cada uma dessas entradas uma resposta, também em alguns casos temos que avaliar a entrada como parte de uma de um caminho ou algo que define um caminho.

```
def equal_number_four(number:int) -> bool:

    if int(number) == 4:
        return True
    else:
        return False
```

Neste simples exemplo que vimos anteriormente, vamos explorar se por acaso o usuário colocar "" ou "abc" como entrada para essa função, ele vai gerar um caminho onde retornaria um ValueError e se ele colocasse [] ou None, como entrada ele nos retornaria TypeError, veja que mesmo para um teste simples. nós temos vários caminhos, e ainda outros caminhos que podemos tentar com colocar um valor Float como entrada e etc. Outra das falhas dos testes de cobertura é que ele não considera os possíveis caminhos gerados por bibliotecas externas.

Os testes como um todo devem ser parte do desenvolvimento e devem se dedicar às partes importantes do sistema, com o objetivo de trazer o máximo de valor e custos mínimos de manutenção. ("acho que uma hora você vão se cansar de quantas formas eu consigo dizer isso de várias maneiras, eu já estou, e cansativo, mas não tem o que fazer, né").

Os Testes em geral devem se direcionar a testar as partes mais importantes de código, que geralmente é o modelo de domínio, onde se encontra a parte lógica e onde estão as regras de negócio do sistema. enquanto o resto deve ser testado brevemente ou de forma indireta.