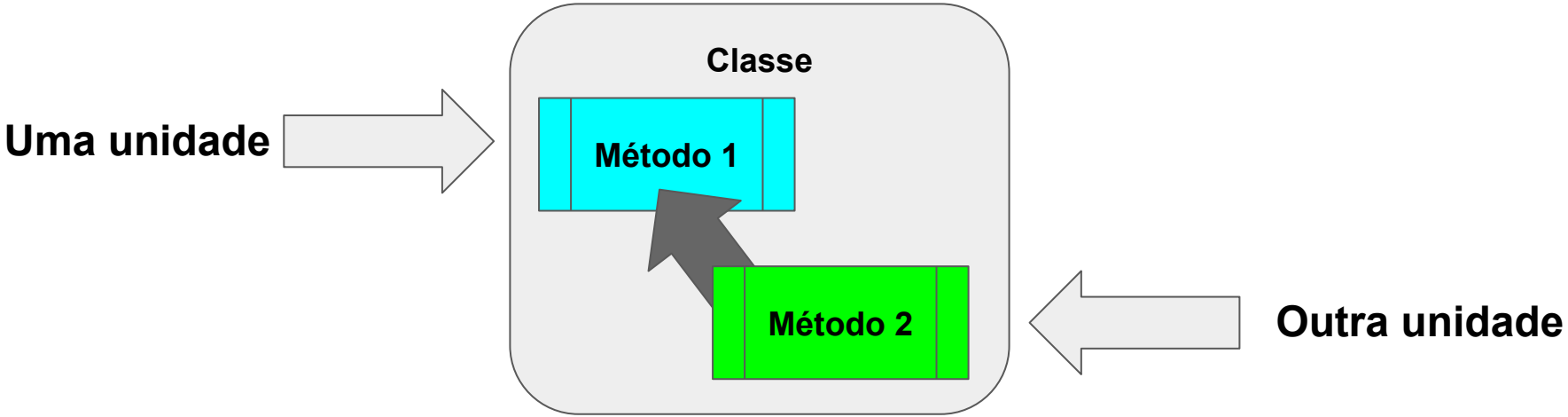
Criando testes unitários

Cleuton Sampaio, M.Sc.

O que é uma unidade?

Uma unidade é um procedimento, função ou método. Uma classe ou módulo não é uma unidade.



Todas as dependências da unidade devem ser mimetizadas (mock). Na figura, o “Método 2” deve invocar um “mock” do método 1.

Tio, devo usar um database in memory?

Muita gente usa databases em memória para criarem testes “unitários”. Não faça isso!

Primeiro, Você está testando mais do que deveria. Segundo, há diferenças entre databases in memory e as versões reais. Terceiro, você precisa ter controle sobre a massa de dados de teste.

Use Mock Objects!

Analise a interface

Busque:

1) Valores aceitos e rejeitados
2) Valores limítrofes
3) Retornos declarados
4) Exceções lançadas

Null
NaN
Infinito
Negativo
zero
Vazio
Acima do limite
Abaixo do limite
No limite

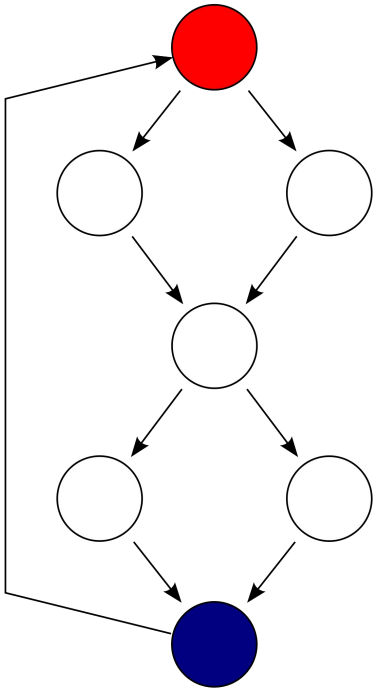
```
func xpto(z: Boolean?, x: Cliente) List<Analise>?
{...}
```

Crie os testes de interface primeiro!

Teste os argumentos, faça combinações de valores válidos e inválidos, verifique o resultado (Exceção ou erro), teste valores limítrofes, faça vários testes de pré condições.

Você já descobrirá vários problemas!

Analise cada caminho possível



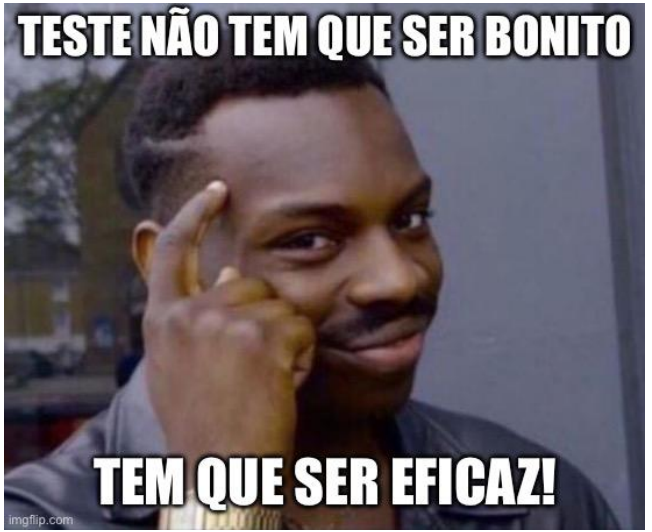
A complexidade ciclomática do código funcional pode complicar a criação de testes!

Cada caminho possível deve ser testado e suas dependências, “mockadas”. A massa de dados e os parâmetros determinam os caminhos por onde seu teste passará.

Antes de assumir que determinado caminho é impossível de acontecer, verifique!

Conselhos finais

- Teste tem que ser eficaz
- Deixe cleancode e design patters para outro momento
- Teste eficaz é aquele que pega erros
- Teste é difícil de fazer e de manter
- Teste com “mocks” é mais eficiente, porém mais difícil e caro de fazer
- Vai mexer no código? Mexa nos testes



Me siga aqui!