LAMBDA3

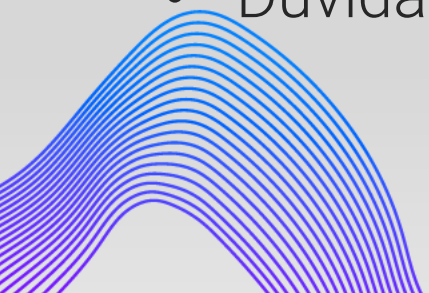
Mentalidade de testes para iniciantes

Sobre o palestrante

- **Nome:** William Porto
- **Desenvolvedor:** Full-Stack
- **Experiência:** 7 anos na Lambda3;

O que veremos hoje?

- O que é um teste?
- Testes automáticos? Por quê?
- Tipos de teste;
- Aprofundar nos tipos de testes;
- Estudos de caso;
- Exemplos em código;
- Dúvidas;

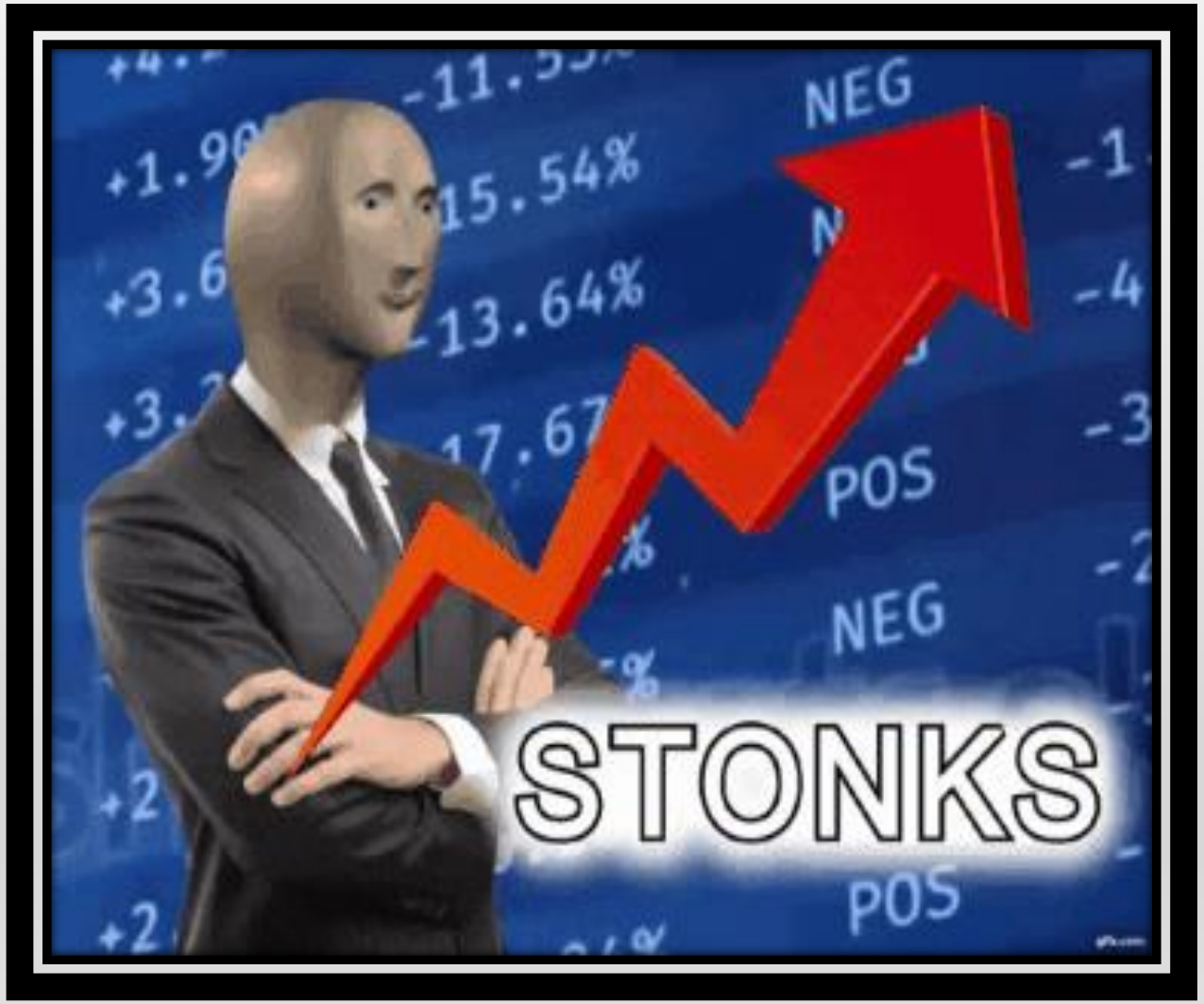


O que define um teste?

- Testes manuais;
- Testes automáticos;

Testes automáticos? Por quê?

- Aumenta qualidade de código;
- Aumenta chance de novas *features* não afetarem as mais antigas;
- Boas práticas;
- Diminui o tempo que leva para acharmos bugs no código;



Benefícios sutis:

- Documentar o código;
- Facilitar mapear Bugs;
- Mapear regras de negócio;
- Melhorar o design do código;

Mas... não é o dobro do trabalho?

Tipos de testes

Testes de propriedade

Testes de caixa branca

Testes de sistema

Testes de fumaça

Testes de unidade

Testes de performance

Testes de aceitação

Testes de integração

Testes exploratórios

Testes de caixa preta

Testes de QA

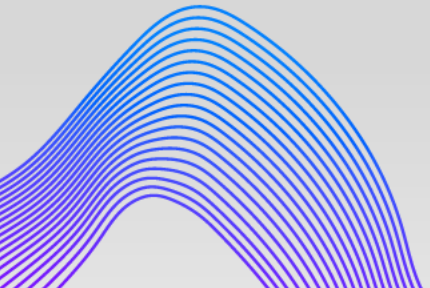
Testes de caixa cinza

... E, do que falaremos? ☺

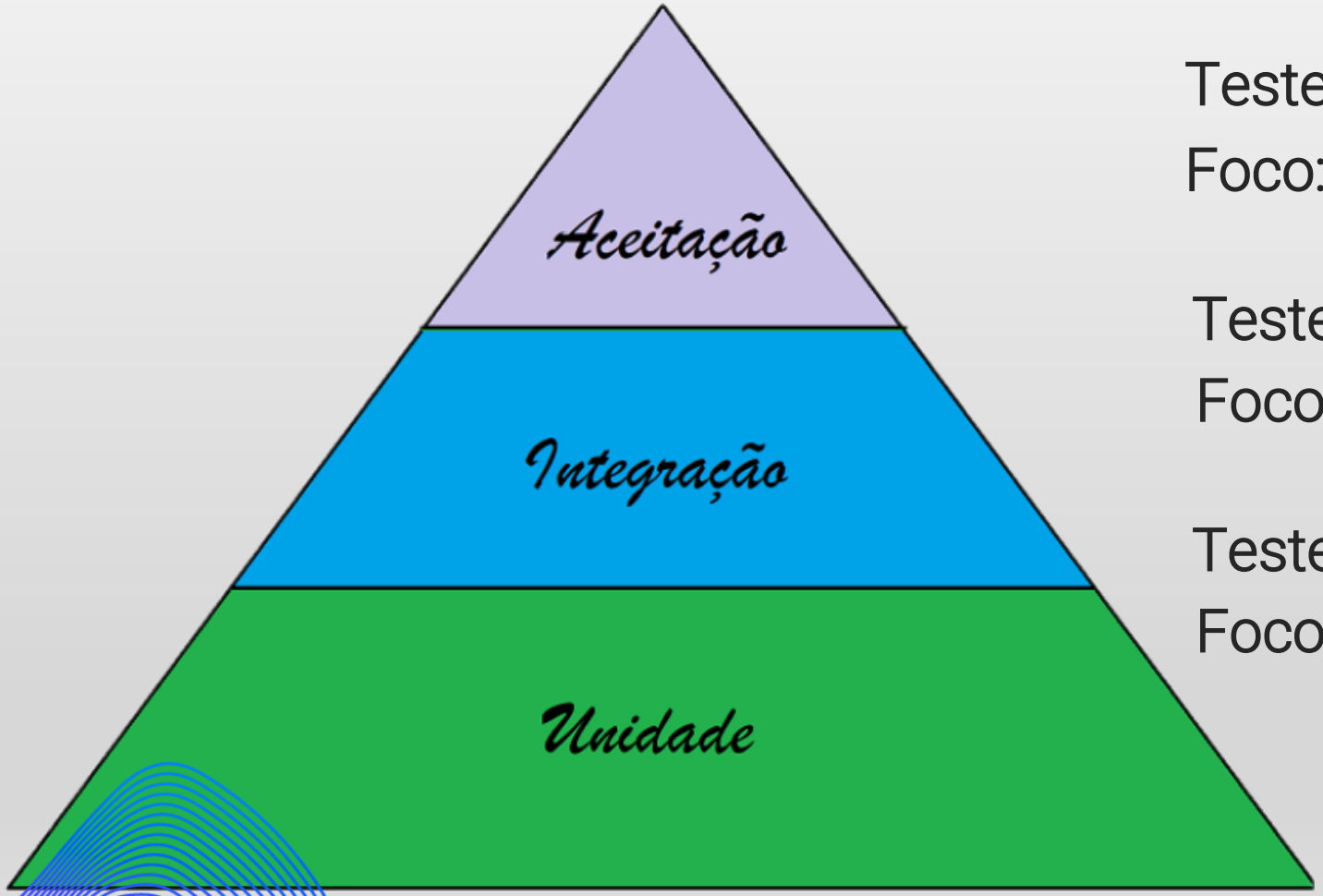
Testes de unidade

Testes de integração

Testes de aceitação



... E, do que falaremos? ☺



Testes de unidade

Foco: isolamento e regras de negócio;

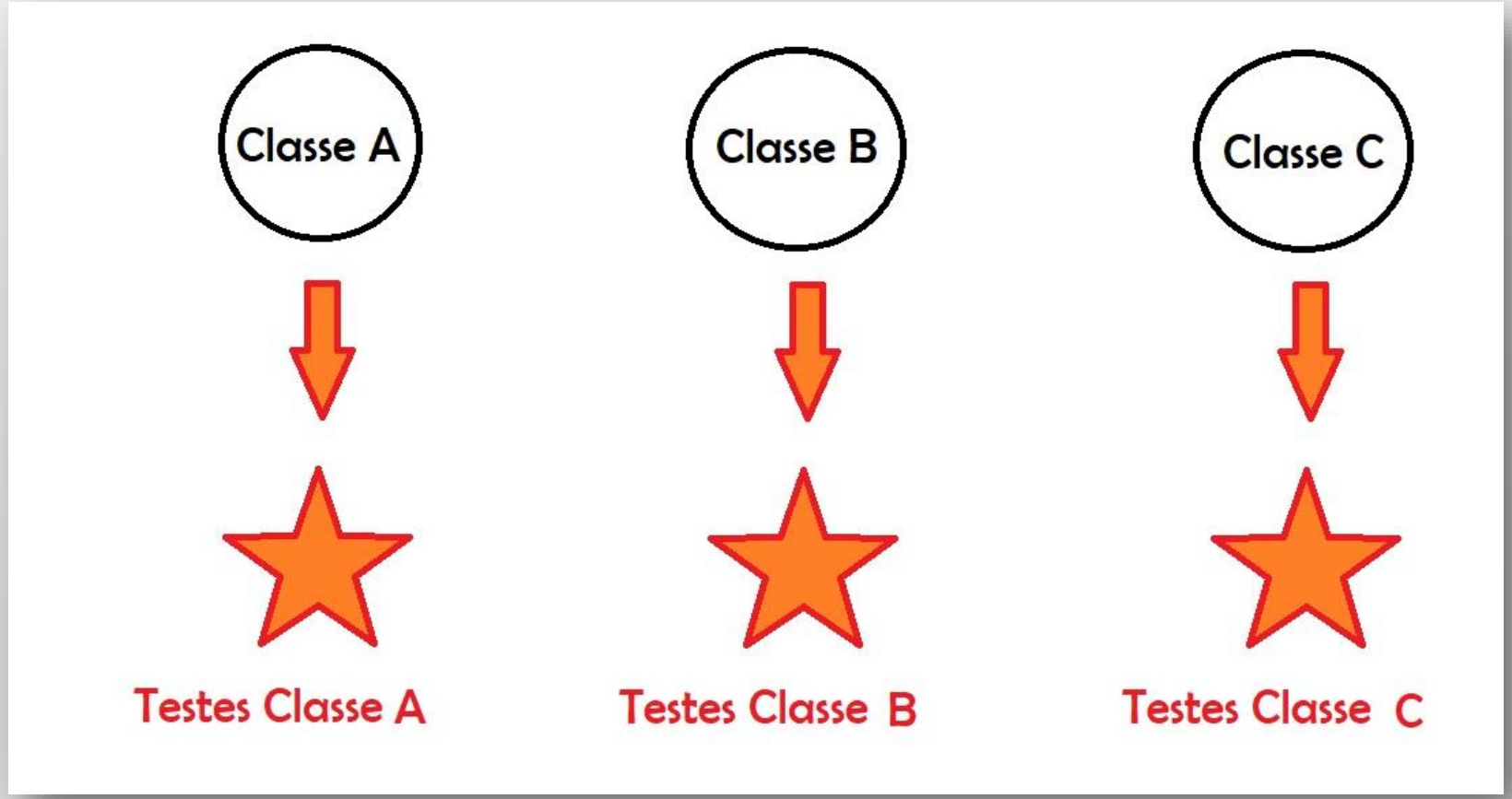
Testes de integração

Foco: trabalho em equipe, sempre!

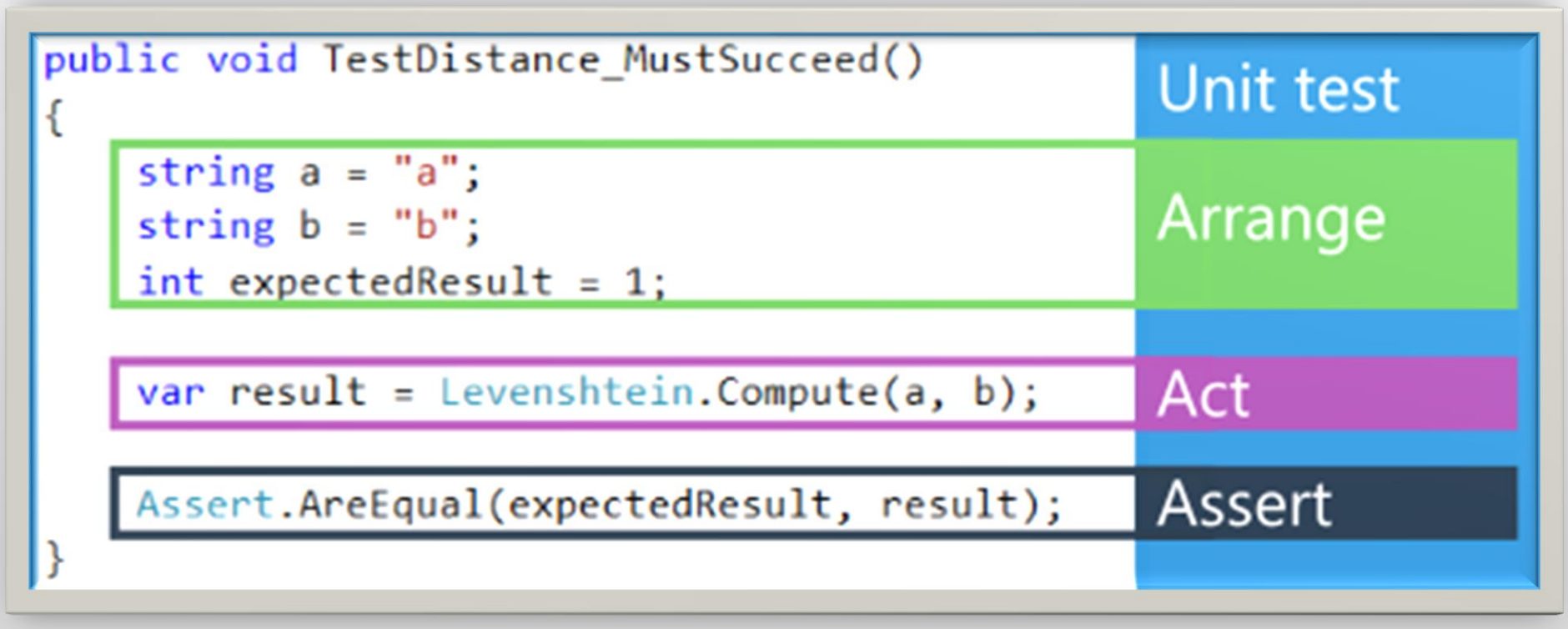
Testes de aceitação

Foco: como o usuário interage com isso?

Testes de Unidade – Isolamento é obrigatório



Interlúdio – Triple A



Testes de Unidade – Exemplo I

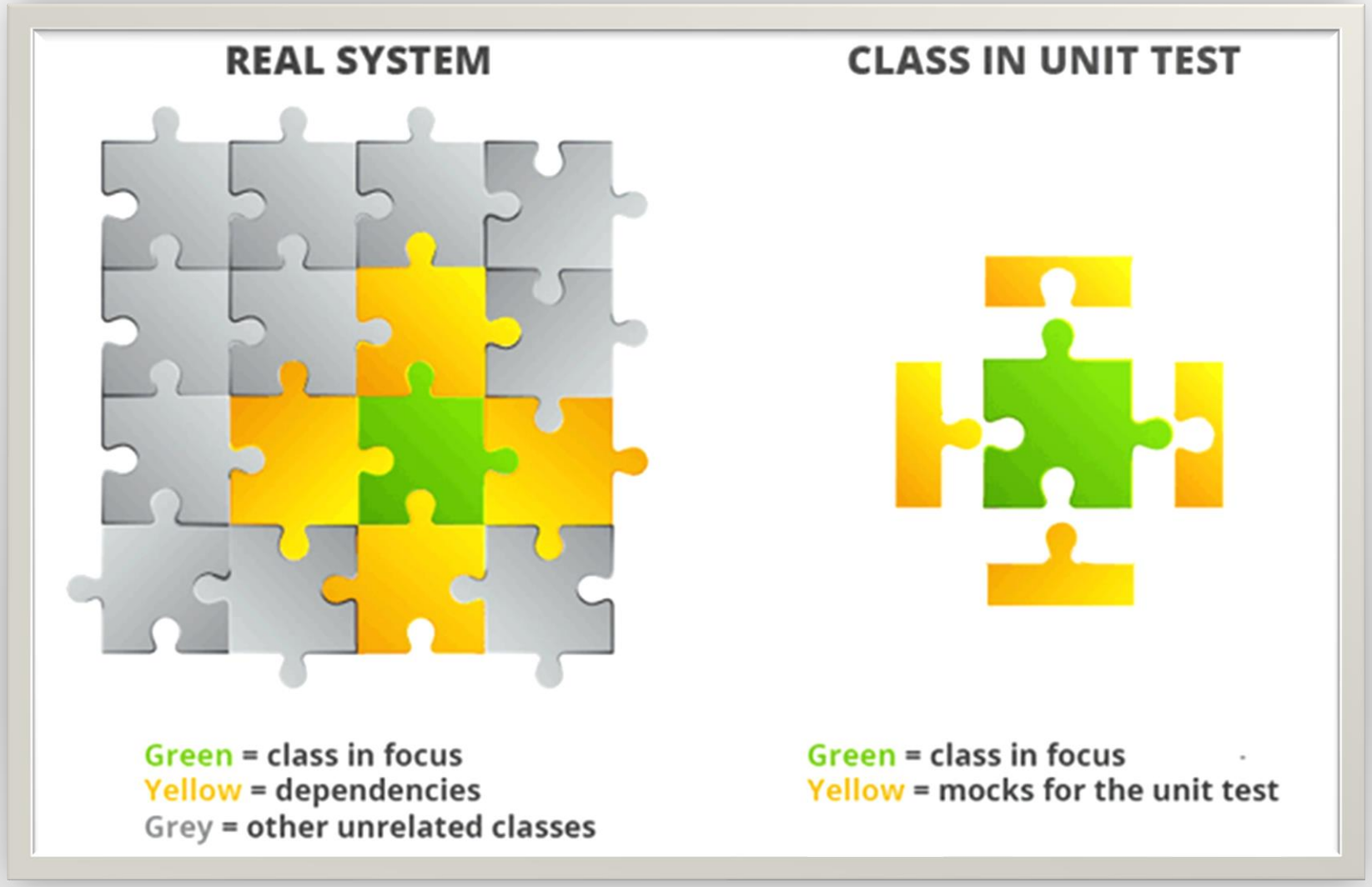
```
1 reference | 0 changes | 0 authors, 0 changes
public class ServicoCalculadora
{
    2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public long SomarNumeros(List<int> numeros)
    {
        if (numeros.Count == 0)
            return -1;

        return numeros.Sum();
    }
}
```

```
[Test]
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public void DeveRetornarMenosUmCasoListaEstejaVazia()
{
    //arrange
    var resultadoEsperado = -1;
    var listaVazia = new List<int>();

    //action
    var resultado = _servicoCalculadora.SomarNumeros(listaVazia);

    //assert
    resultado.Should().Be(resultadoEsperado);
}

[Test]
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public void DeveRetornarSomaDosNumerosNaLista()
{
    //arrange
    var lista = new List<int>() { 3, 4, 5, 10 };
    var resultadoEsperado = 22;

    //action
    var resultado = _servicoCalculadora.SomarNumeros(lista);

    //assert
    resultado.Should().Be(resultadoEsperado);
}
```

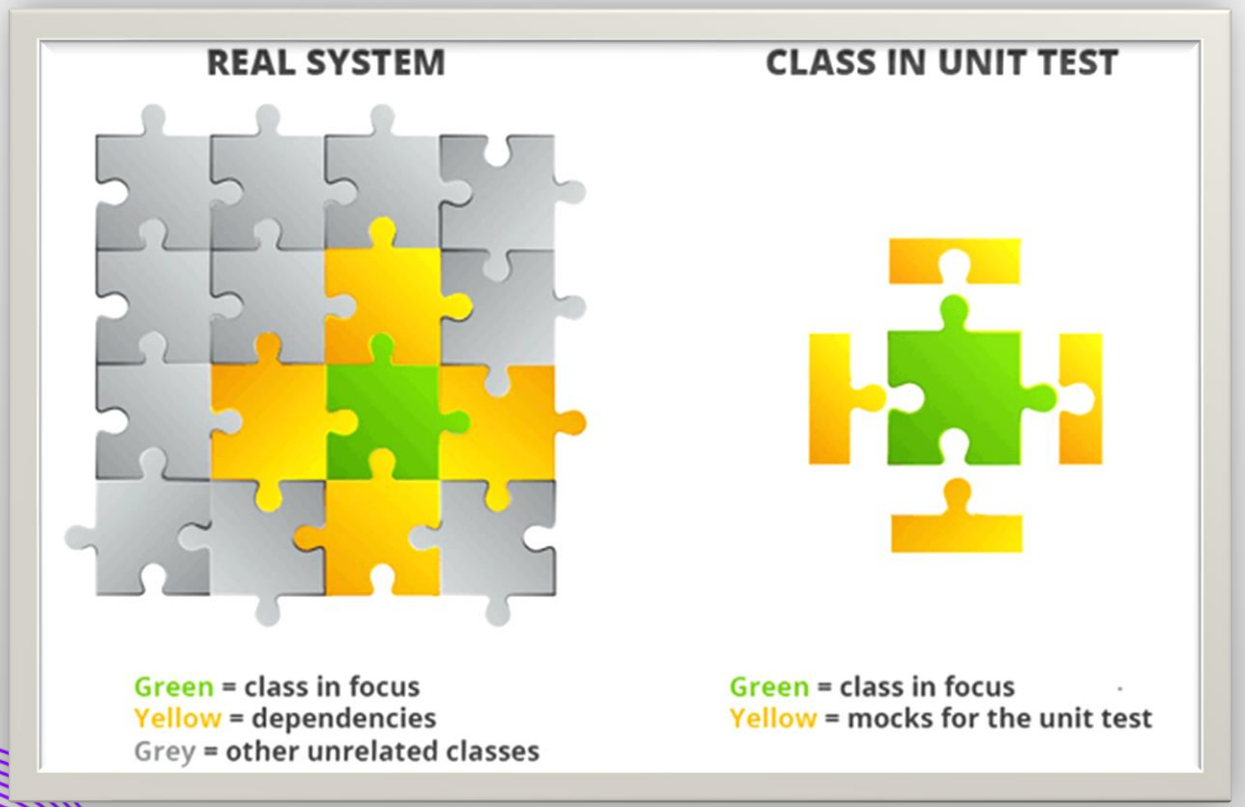
Testes de Unidade – Exemplo II

```
public class ServicoCatalogo
{
    private readonly IRepositorio<Livro> _repositorioLivros;
    private readonly ILivroValidador _validador;

    1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
    public ServicoCatalogo(IRepositorio<Livro> repositorioLivros, ILivroValidador validador)
    {
        _validador = validador;
        _repositorioLivros = repositorioLivros;
    }

    5 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public List<string> CadastrarNoCatalogo(Livro livro)
    {
        var result = _validador.Validate(livro);

        if (!result.IsValid)
            return result.Errors.Select(x => x.ErrorMessage).ToList();

        _repositorioLivros.Salvar(livro);
        return new List<string>();
    }
}
```

Testes de Unidade – Como isolar? Mocks!



Testes de Unidade – Como isolar? Mocks!



- Cria uma versão falsa do objeto que você passa para ele;
- Pensamento base: “O que preciso garantir que aconteça para que meu cenário de teste possa ocorrer?”;
- O que posso mockar: **classes concretas** e **interfaces**!

Testes de Unidade – Mockando classes concretas

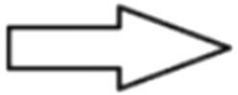
```
var pessoa = new Mock<Pessoa>();
```

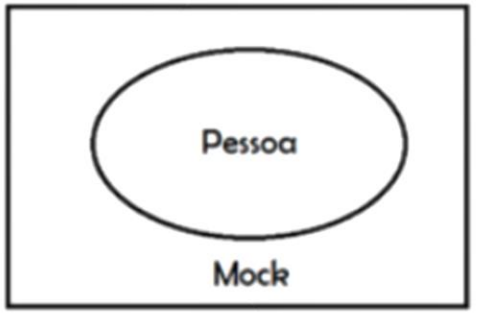
1. Leia as propriedades e os métodos que eu tenho dentro da classe concreta (nesse caso, pessoa);

2. Crie uma instancia daquela pessoa;

3. Dê valores para as propriedades dessa classe;

4. Dê override nos métodos aue forem virtuais:





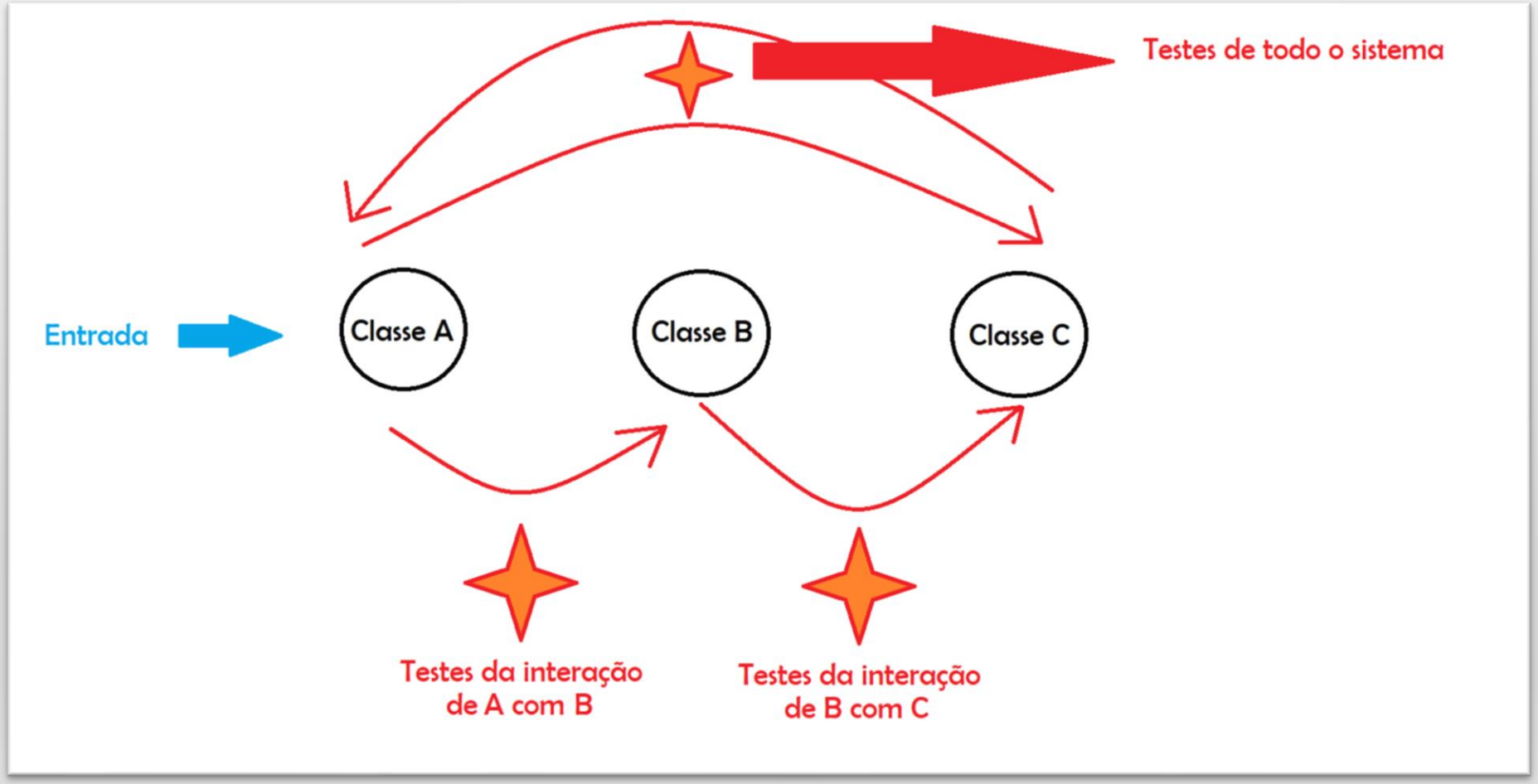
Testes de Unidade – Mockando interfaces

```
var pessoa = Mock<IRepositorioPessoa>();
```

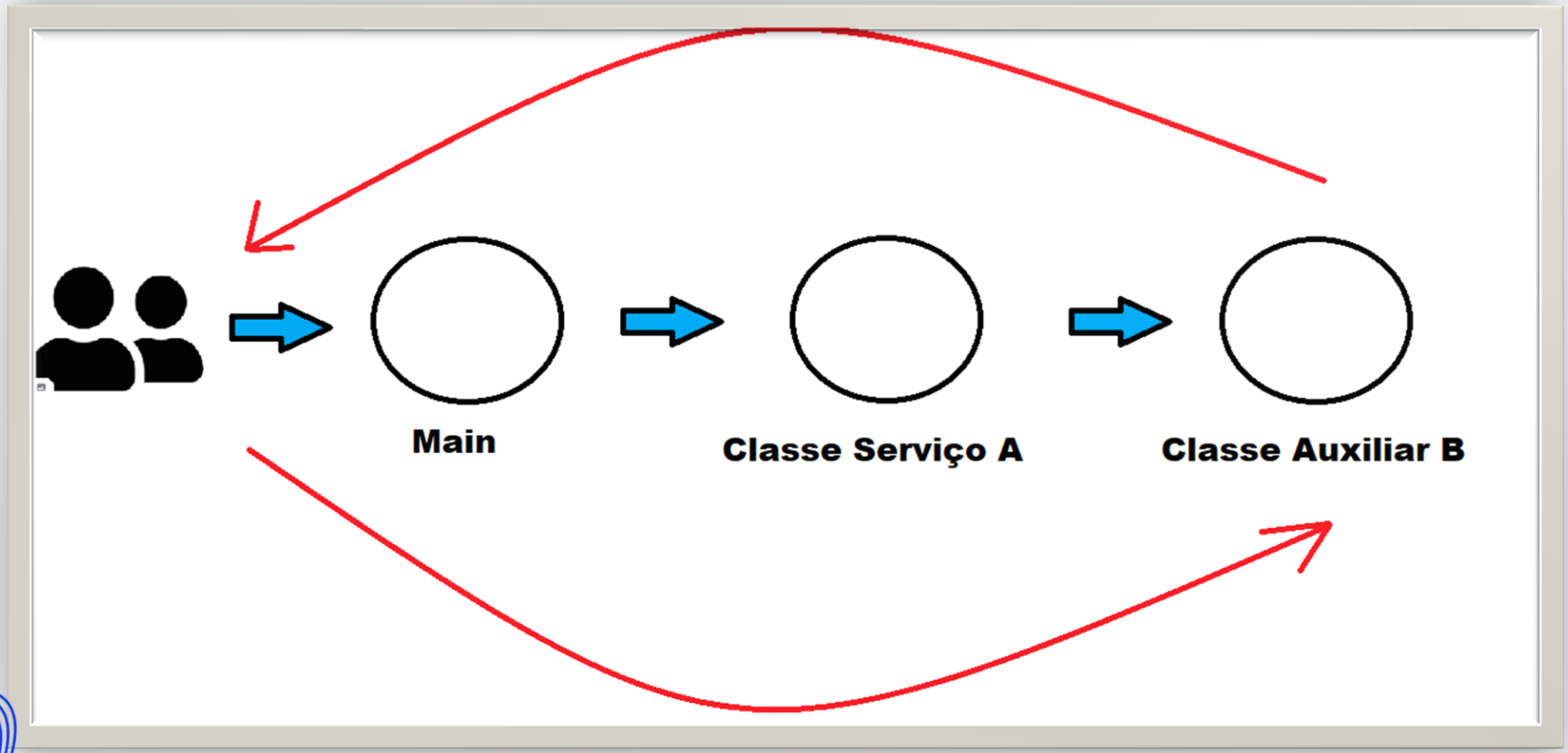
1. Lê os métodos disponíveis na interface
2. Instancia um objeto que implementa a interface
3. Faz com que os métodos se comportem conforme esperamos

Vamos ver um código?

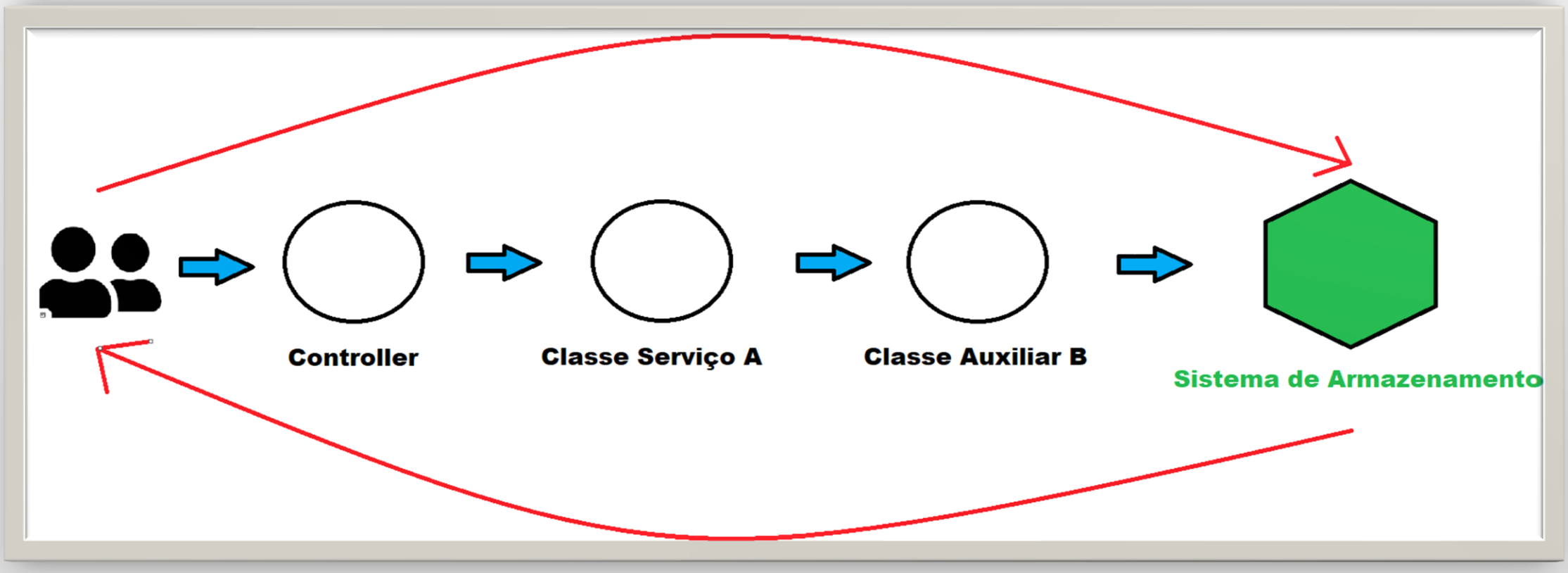
Testes de integração – Unidos jamais serão vencidos!



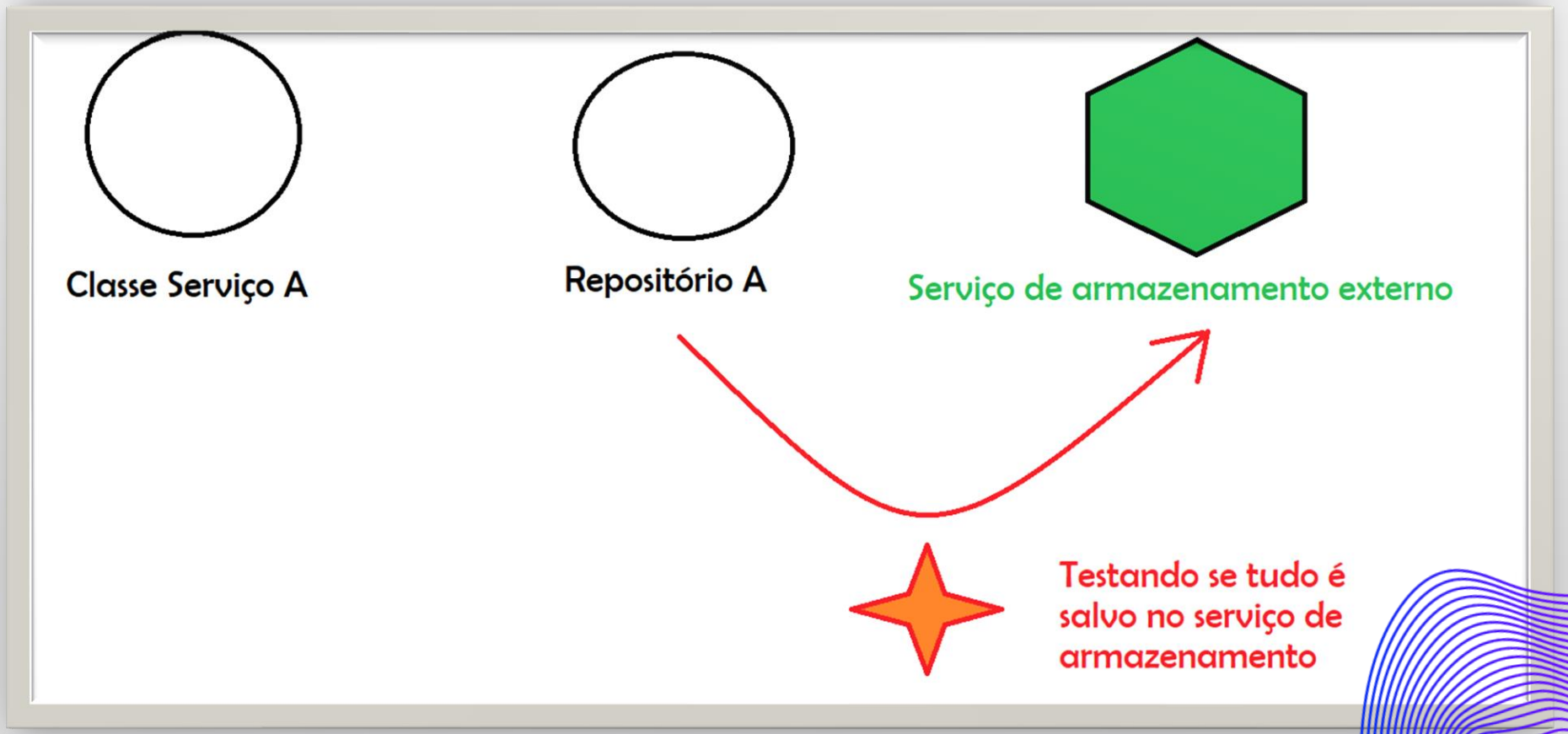
Testes de integração – Testando Consoles APP



Testes de integração – Testando APIs



Testes de integração – Serviço de armazenamento



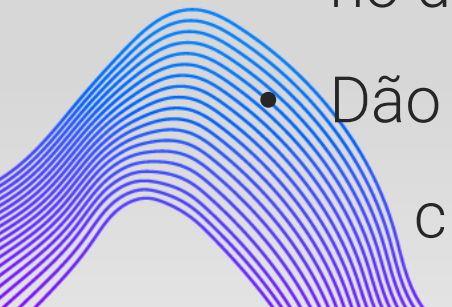
Testes de unidade X integração

Unidade

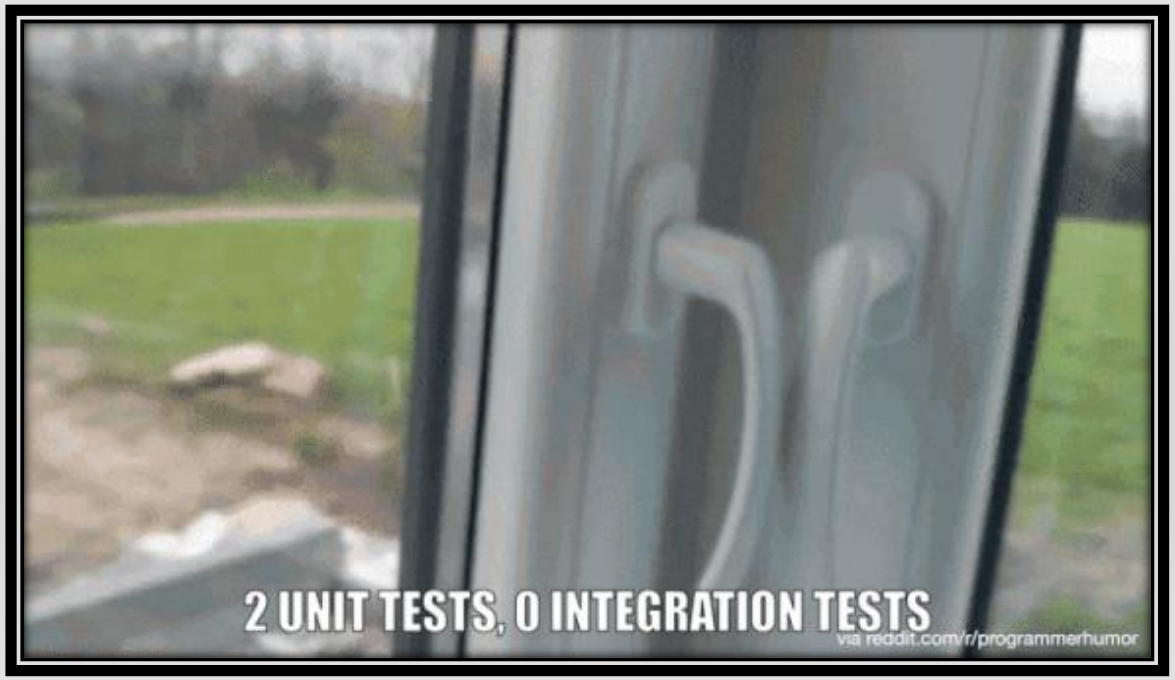
- Setup rápido;
- Trabalha com mocks;
- Roda mais rápido;
- Feedback quase instantâneo;
- Melhor para TDD (auxilia mais no design das classes);
- Dão uma visão apenas da classe;

Integração

- Setup mais demorado;
- Trabalha com ferramentas reais (pensar em limpar banco, em iniciar APIs, etc.);
- Feedbacks costumam demorar mais;
- Não tão legal para TDD;



Mas... Por que os dois?



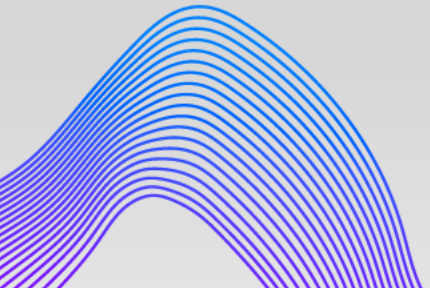
Vamos ver um código?

Unidade e Integração no mundo real – estudo de caso

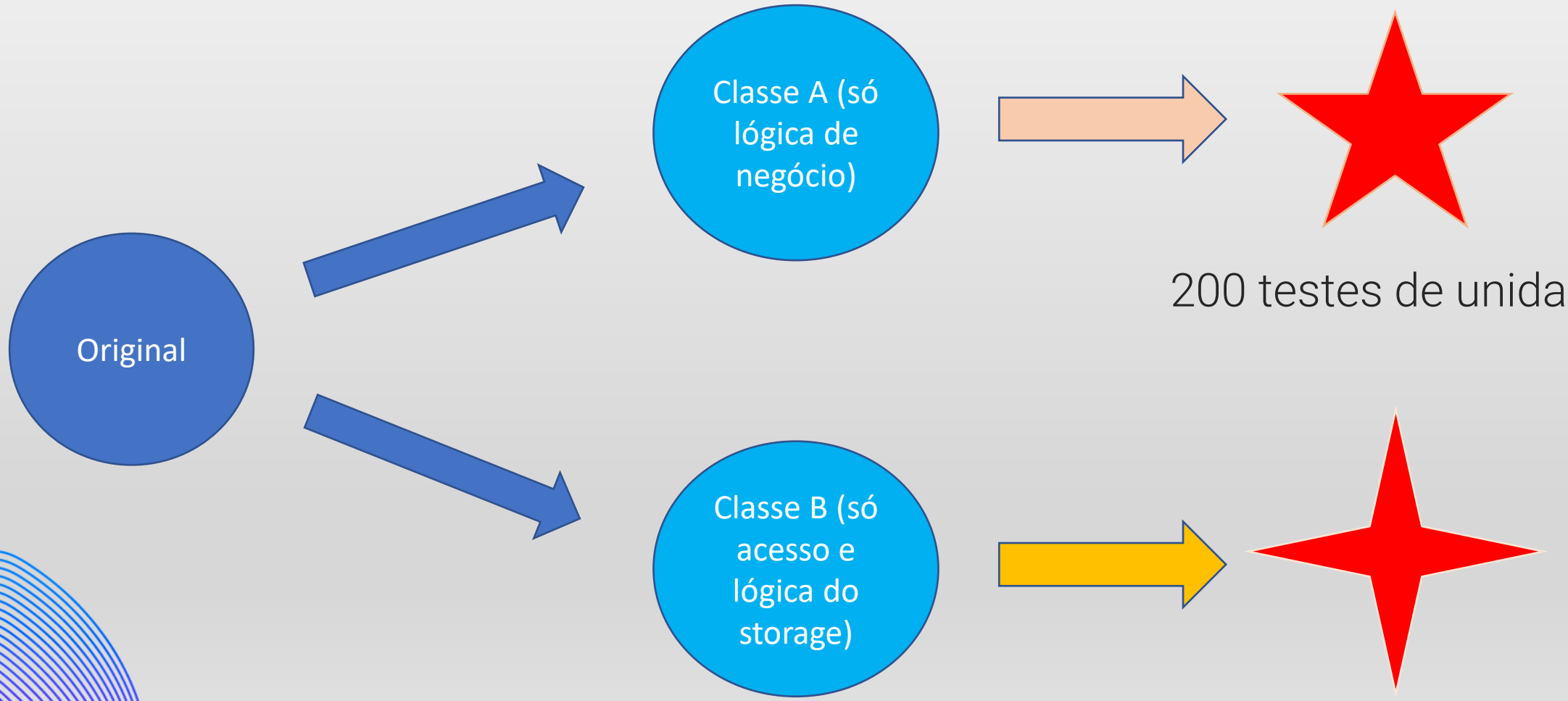
Objetivo: Migração de bibliotecas do Azure Storage

Cenário:

1. Regras de negócio junto do acesso ao sistema de armazenamento;
2. Storage gerava URLs para acesso de sistemas externos;
3. Operações de Upload, de obter dados do storage, de gerar URL, todas eram efetuadas pela biblioteca



Unidade e Integração no mundo real – estudo de caso



Testes de aceitação – Tá, mas e o meu cliente?

Testes de aceitação – Simular comportamentos ~~humanos~~

Vamos ver um código?

Dúvidas?

Obrigado ^^

Twitter: @WillRock19

Github: https://github.com/WillRock19/LivrariaComTestes

Importante: código desatualizado há dois anos! Atualizações em breve ☺