

Testes Automatizados na plataforma .net

Primeira parte: Testes unitários e TDD

Professor: Alexandre Rech

- **Graduado em Sistemas de Informação**
- **Pós-graduado em Engenharia de Software**
- **9 anos de experiência com desenvolvimento de software**
 - **5 anos de experiência ensinando programação**
- **E testes automatizados?**

Experiência do Professor

➤ Testes unitários

➤ TDD

➤ Mock Objects

➤ Testes de Integração

➤ Refatoração de Código

➤ Testes Funcionais

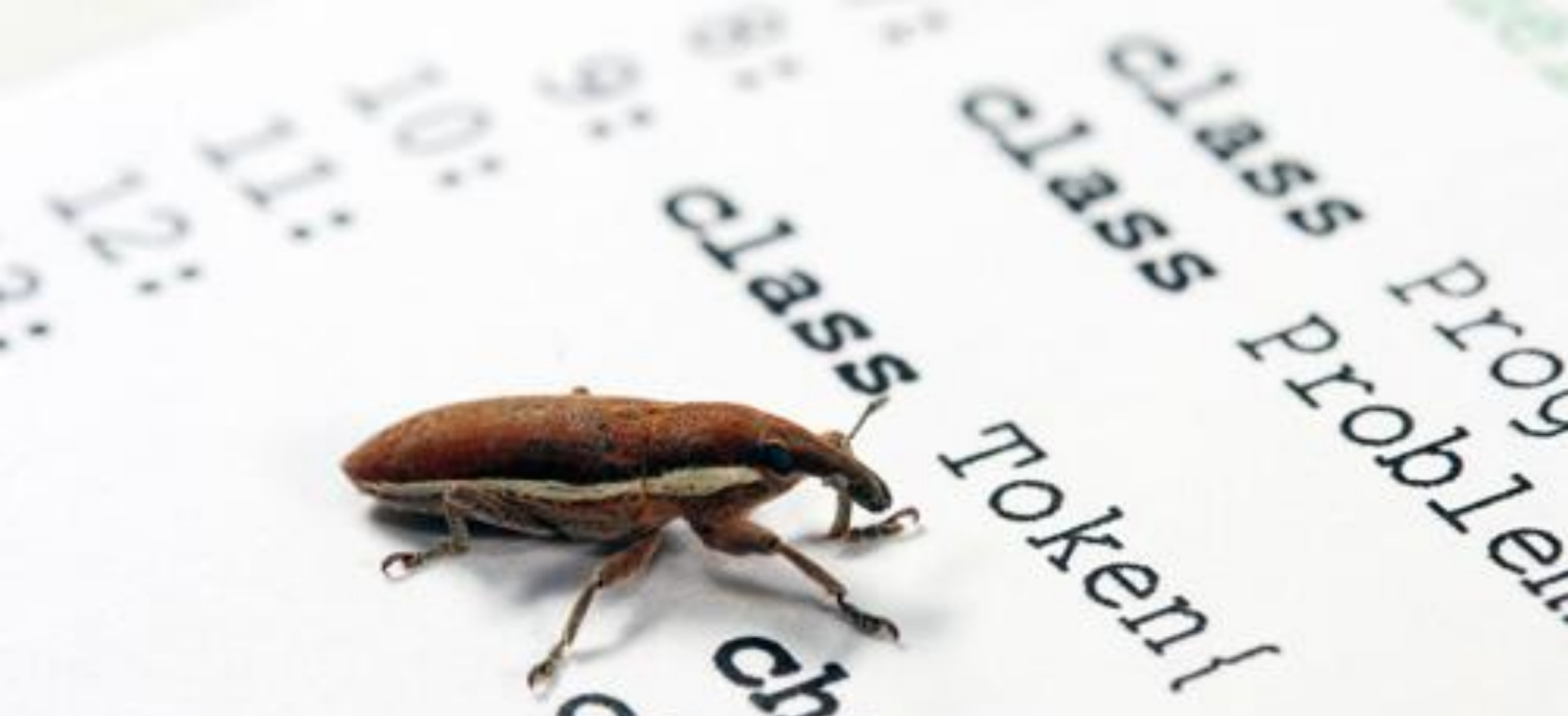
➤ BDD

➤ Integração Contínua

O que vamos ver neste curso?



Começando com Testes de Unidade



Com certeza, **todo desenvolvedor** de software já **escreveu** um trecho de **código** que **não funcionava**.



Para cada **bug** resolvido usando **eXtreme Go Horse (XGH)**, mais uns 7 são criados. Mas todos eles serão resolvidos da forma **XGH**. **XGH** tende ao infinito.



O pior, é que muitas vezes só descobrimos que o código não funciona quando **nosso cliente** nos **reporta** o **bug**.



O Cliente **perde a confiança** na equipe de desenvolvimento porque ela **não entrega código de qualidade**

**Desenvolvedores perdem a
confiança no seu código porque o
número de bugs é alto**

**Acabam ficando com medo de
alterar os códigos do sistema.**





Defeito de
requisito



Defeito de
Análise



Defeito de
Desenho



Defeito de
Configuração



Defeito de Teste



Defeito de
Implementação

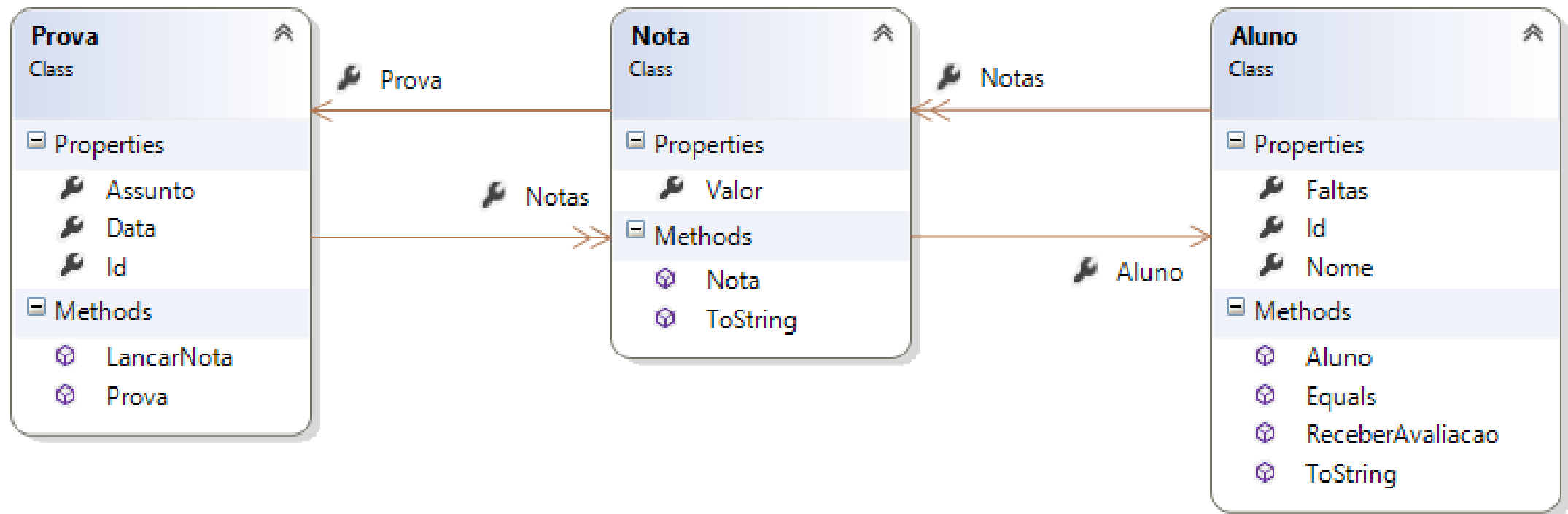


Defeito de
Arquitetura

■ ■ ■



Diário da Academia do Programador NDD



Módulo de Provas no Diário da Academia



Vamos implementar a funcionalidade de **encontrar a maior** nota de uma prova



**CHANGE
AHEAD**



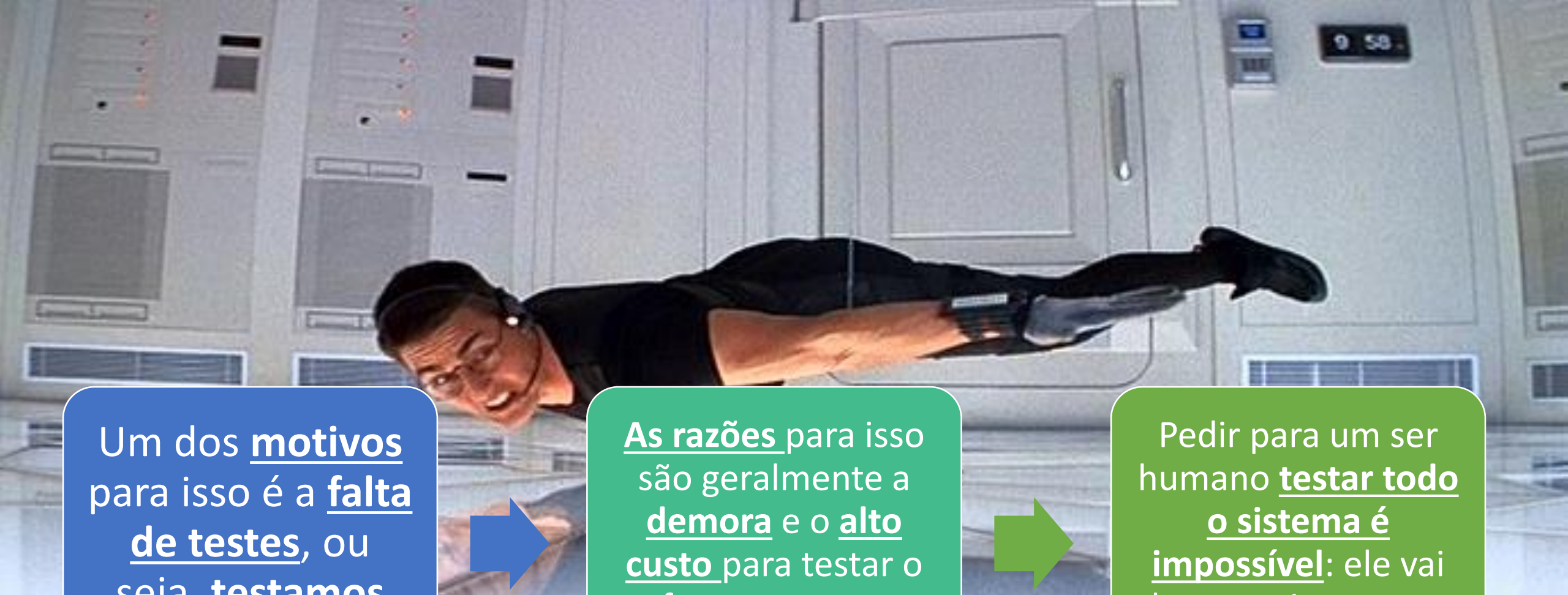
Vamos implementar a funcionalidade de **encontrar a menor nota** de uma prova



10

1,79769313486232E+208





Um dos motivos para isso é a falta de testes, ou seja, testamos muito pouco!



As razões para isso são geralmente a demora e o alto custo para testar o software como um todo.



Pedir para um ser humano testar todo o sistema é impossível: ele vai levar muito tempo para isso!

Por que nossos sistemas apresentam tantos bugs?

- “Ninguém gosta de testar”
- “Testar é chato”
- “Ah isso é fácil que nem precisa testar”
- “Deixa pro estagiário fazer...”

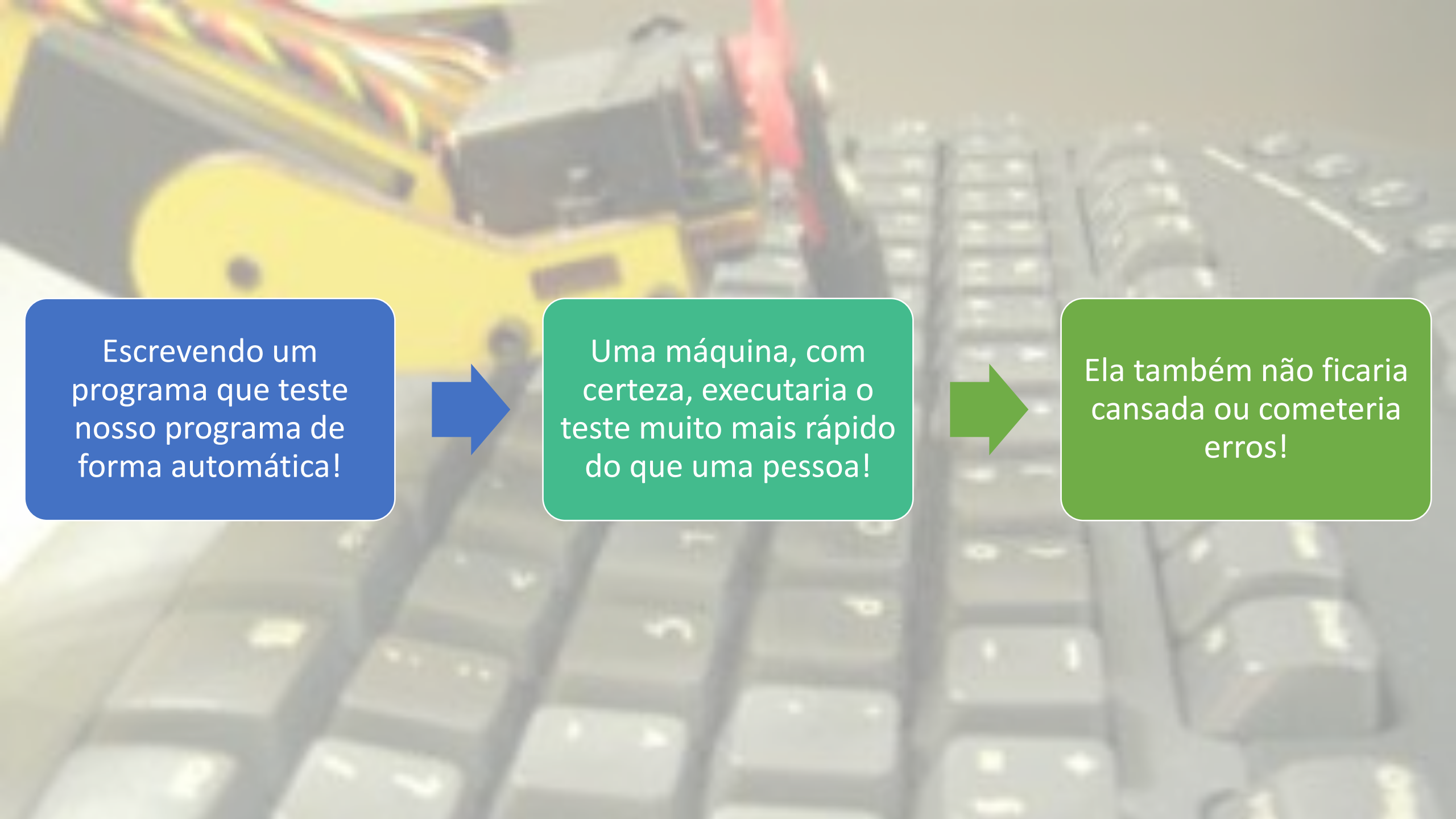




Como resolver este problema?



Fazendo a máquina testar!



Escrevendo um
programa que teste
nosso programa de
forma automática!



Uma máquina, com
certeza, executaria o
teste muito mais rápido
do que uma pessoa!



Ela também não ficaria
cansada ou cometeria
erros!

Cadastre-se

Nome de Usuário

✓ Nome de usuário disponível

E-mail

✳️ Confirmando...

Crie sua senha

✓ A senha está boa

Reescreva sua senha

✗ A senha está errada

Cadastrar

Pensa num
cenário



Executa uma
ação



Valida a
Saída

Um teste automatizado é muito parecido com um teste manual.

```
static void Main(string[] args)
{
    Prova prova = new Prova(DateTime.Now);

    prova.LancarNota(8, new Aluno("João"));
    prova.LancarNota(10, new Aluno("José"));
    prova.LancarNota(5, new Aluno("Maria"));

    AvaliadorProva avaliador = new AvaliadorProva();

    avaliador.Avaliar(prova);

    // imprimi 10
    Console.WriteLine(avaliador.ObtemMaiorNota());

    // imprimi 5
    Console.WriteLine(avaliador.ObtemMenorNota());

    Console.ReadKey();
}
```

Cenário

Ação

Valida a saída

A classe Program é um teste automatizado?



Test Explorer

Search

Streaming Video: Improving quality with un

Run All | Run... | Playlist: All Tests

AvaliadorProvaTest (1)

✖ TestMethod1 39 ms

TestMethod1

Source: AvaliadorProvaTest.cs line 12

✖ Test Failed - TestMethod1

Message: Assert.AreEqual failed.
Expected:<5>.
Actual:<1,79769313486232E+308>.

Elapsed time: 39 ms

Aluno.cs AvaliadorProva.cs **AvaliadorProvaTest.cs***

DiarioAcademiaTests DiarioAcademiaTests.AvaliadorProvaTest

```
1 using System;
2 using Microsoft.VisualStudio.TestTools.UnitTesting;
3 using DiarioAcademia.Dominio;
4
5 namespace DiarioAcademiaTests
6 {
7     [TestClass]
8     // 0 references | 0 authors | 0 changes
9     public class AvaliadorProvaTest
10    {
11        [TestMethod]
12        // 0 references | 0 authors | 0 changes
13        public void TestMethod1()
14        {
15            Prova prova = new Prova(DateTime.Now);
16
17            prova.LancarNota(5, new Aluno("Maria"));
18            prova.LancarNota(8, new Aluno("João"));
19            prova.LancarNota(10, new Aluno("José"));
20
21            AvaliadorProva avaliador = new AvaliadorProva();
22
23            avaliador.Avaliar(prova);
24
25            // imprimi 10
26            Assert.AreEqual(10, avaliador.ObtemMaiorNota());
```

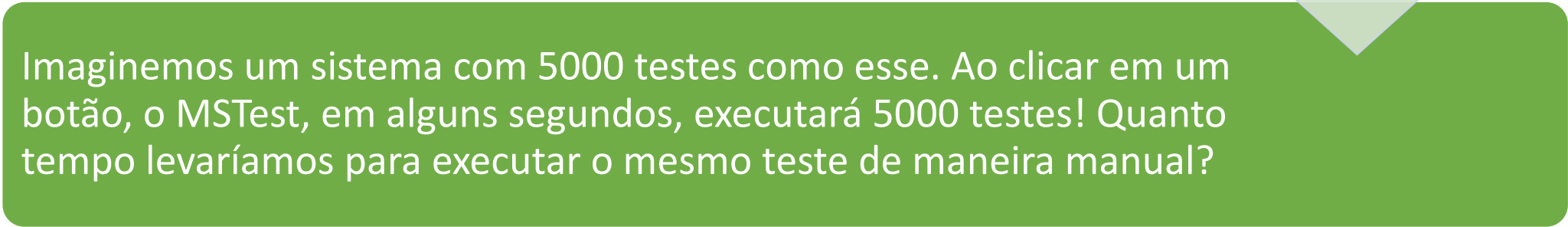

O tempo que o teste leva para ser executado são alguns milissegundos.



Podemos rodar esse teste O TEMPO TODO, que é rápido e não custa nada. É a máquina que roda!



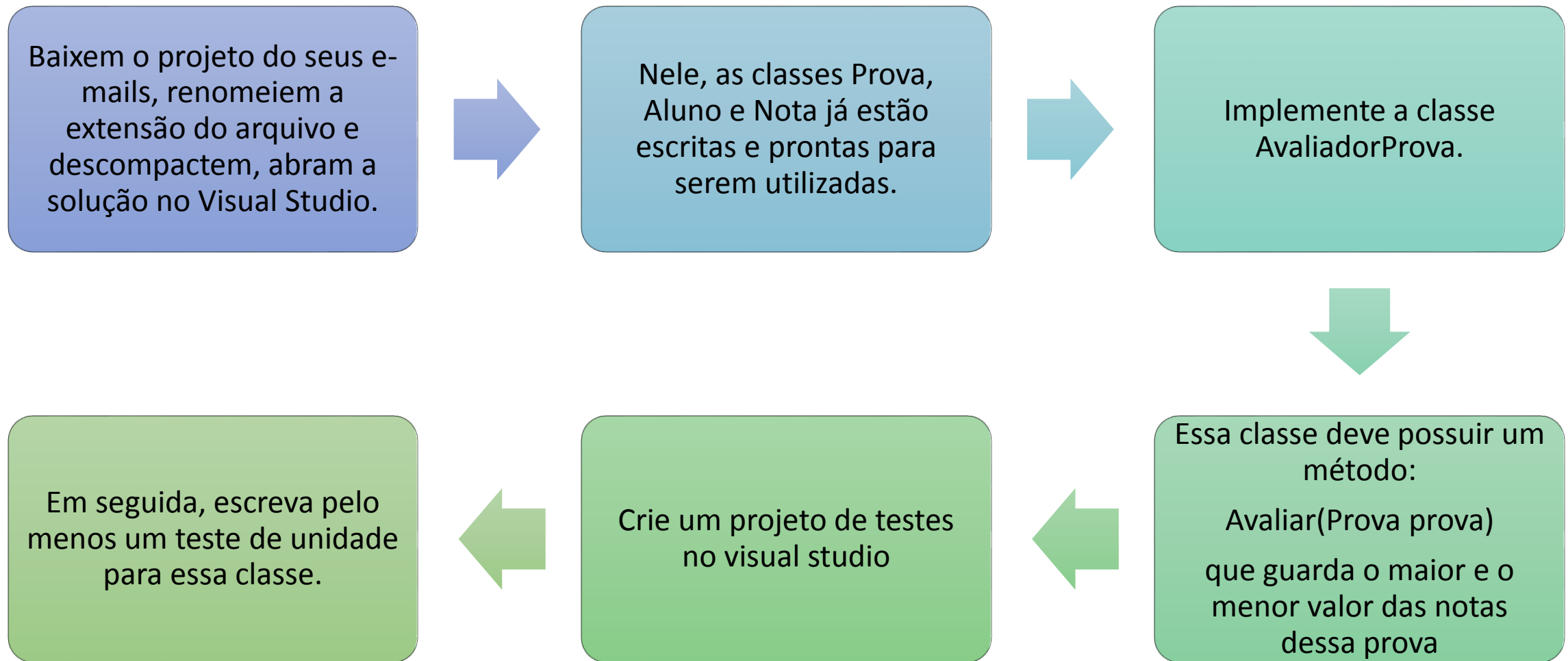
Imaginemos um sistema com 5000 testes como esse. Ao clicar em um botão, o MSTest, em alguns segundos, executará 5000 testes! Quanto tempo levaríamos para executar o mesmo teste de maneira manual?



Vantagens do teste de unidade



Exercícios Testes de Unidade



Exercício 1: Meu primeiro teste de unidade

```

[TestClass]
public class AvaliadorProvaTest
{
    [TestMethod]
    public void Deveria_avaliar_notas_ordem_crescente()
    {
        Prova prova = new Prova(DateTime.Now);

        prova.LancarNota(5, new Aluno("Maria"));
        prova.LancarNota(8, new Aluno("João"));
        prova.LancarNota(10, new Aluno("José"));

        AvaliadorProva avaliador = new AvaliadorProva();

        avaliador.Avaliar(prova);

        // imprimi 10
        Assert.AreEqual(10, avaliador.ObtemMaiorNota());

        // imprimi 5
        Assert.AreEqual(5, avaliador.ObtemMenorNota());
    }
}

```

```

public class AvaliadorProva
{
    private double _maiorNota = double.MinValue;
    private double _menorNota = double.MaxValue;

    public double ObtemMaiorNota()
    {
        return _maiorNota;
    }

    public double ObtemMenorNota()
    {
        return _menorNota;
    }

    public void Avaliar(Prova prova)
    {
        foreach (Nota nota in prova.Notas)
        {
            if (nota.Valor > _maiorNota)
            {
                _maiorNota = nota.Valor;
            }
            if (nota.Valor < _menorNota)
            {
                _menorNota = nota.Valor;
            }
        }
    }
}

```

Exercício 1: Meu primeiro teste de unidade

Qual a ordem correta dos parâmetros do `Assert.AreEqual()` (e de todos os outros métodos similares) da classe `Assert`?

- a) ☐ `Assert.AreEqual(calculado, esperado)`
- b) ☐ `Assert.AreEqual(esperado, calculado)`
- c) ☐ `Assert.AreEqual(esperado, esperado)`
- d) ☐ `Assert.AreEqual(calculado, calculado)`

Exercício 2: Ordem dos parâmetros do Assert

Qual a ordem correta dos parâmetros do `Assert.AreEqual()` (e de todos os outros métodos similares) da classe `Assert`?

- a) ☐ `Assert.AreEqual(calculado, esperado)`
- b) ☒ `Assert.AreEqual(esperado, calculado)`
- c) ☐ `Assert.AreEqual(esperado, esperado)`
- d) ☐ `Assert.AreEqual(calculado, calculado)`

Exercício 2: Ordem dos parâmetros do Assert

Qual o padrão para nomenclatura de classes de teste?

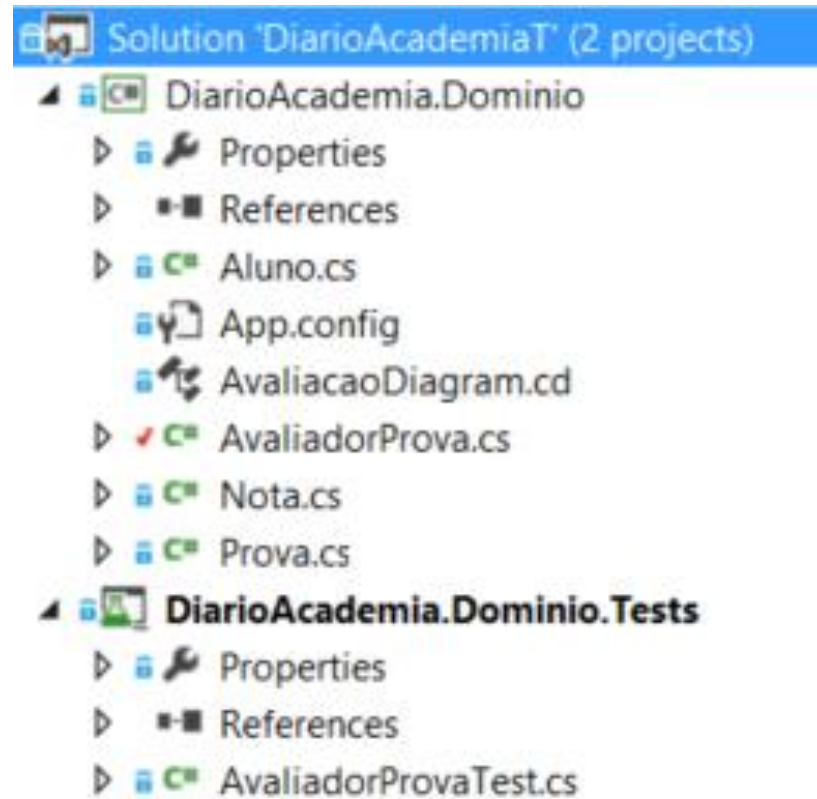
- a) ☐ "TestandoAClasse"NomeDaClasse"
- b) ☐ "TesteDaClasse"NomeDaClasse"
- c) ☐ "NomeDaClasse"Teste
- d) ☐ "NomeDaClasse"Test
- e) ☐ "TodosOsTestesDa"NomeDaClasse"

Exercício 3: Nome da Classe de Teste

Qual o padrão para nomenclatura de classes de teste?

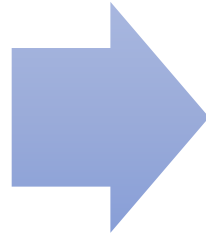
- a) ☐ "TestandoAClasse"NomeDaClasse"
- b) ☐ "TesteDaClasse"NomeDaClasse"
- c) ☐ "NomeDaClasse"Teste
- d) ☒ "NomeDaClasse"Test
- e) ☐ "TodosOsTestesDa"NomeDaClasse"

Exercício 3: Nome da Classe de Teste



Exercício 4: Onde guardamos os testes?

Implemente um método no **AvaliadorProva** que devolve o valor médio das notas.



Escreva também o teste para isso.

Exercício 5: Média das Notas

```

[TestClass]
public class AvaliadorProvaTest
{
    [TestMethod]
    public void Deveria_calcular_media_das_notas()
    {
        Prova prova = new Prova(DateTime.Now);

        prova.LancarNota(5, new Aluno("Maria"));
        prova.LancarNota(10, new Aluno("José"));

        AvaliadorProva avaliador = new AvaliadorProva();

        avaliador.Avaliar(prova);

        Assert.AreEqual(7.50, avaliador.ObtemMedia(
    }
}

```

```

public void Avaliar(Prova prova)
{
    double total = 0;

    foreach (Nota nota in prova.Notas)
    {
        if (nota.Valor > _maiorNota)
        {
            _maiorNota = nota.Valor;
        }
        if (nota.Valor < _menorNota)
        {
            _menorNota = nota.Valor;
        }

        total += nota.Valor;
    }

    CalcularMedia(total, prova.Notas.Count);
}

```

Exercício 5: Média das Notas

Um humano pode eventualmente executar um teste incorreto. A máquina nunca fará isso.

A partir do momento que você escreveu o teste, ela sempre vai executar o mesmo teste.

É muito mais divertido escrever um teste automatizado do que testar manualmente.

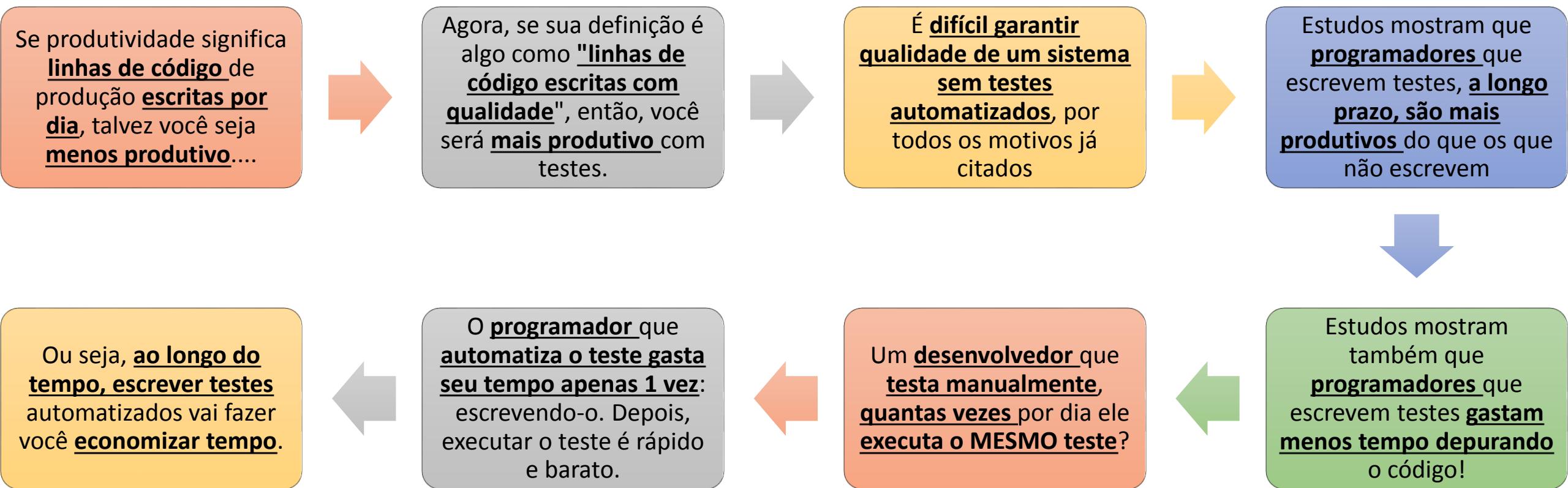
Um teste de unidade executa muito rápido.

Imagine o tempo que um humano levaria.

Testes automatizados de unidade nos trazem diversas vantagens. Nesse momento, as vantagens que são fáceis de ver são:



Exercício 6: Quais as Vantagens do Teste de Unidade?



Exercício 7: Produtividade em Testes Automatizados

Testes de aceitação para sistemas Web utilizando BDD

por André Vinícius Buzzo em 01 Ago 2014 | 1 Dê sua opinião

Compartilhar



Favoritos

Marcar como favorito

Gravado em:



Resumo

Após trabalhar por 4 anos em projetos de médio porte, ficou clara a necessidade de organizar e dar manutenção nos testes automatizados. Diversos desafios levaram à busca por um modo maduro de refatorá-los. Nesta palestra são compartilhados 'insights', 'ups' e 'downs' enfrentados nesses projetos baseados em Behavior Driven Development e é apresentado o modelo atual que combina diversas



Exercício 7: Produtividade em Testes Automatizados

Como TDD e Pareamento Aumentam a Produtividade

por [Mike Bria](#), traduzido por [Samuel Carrijo](#) em 04 Jun 2009 | [Dê sua opinião](#)

Compartilhar [+](#) [f](#) [diigo](#) [dz](#) [tw](#) [st](#) [sk](#) [em](#)

[Favoritos](#)

[Marcar como favorito](#)

"Desenvolvimento orientado a testes" (TDD) e "Pareamento" são duas das práticas ágeis mais conhecidas, e mesmo assim não são postas em prática por muitas equipes ágeis. Com frequência, as pessoas afirmam estar "muito ocupadas" para praticarem TDD e pareamento; em essência, deixando a entender que esforçar-se para produzir um código de alta qualidade reduz a produtividade. Mike Hill explica como esta lógica está muito furada.

Mike afirma, essencialmente, que [deve-se "fazer melhor" quem quiser "fazer mais rápido"](#):

Não só não é verdade que pode-se trocar qualidade interna por mais funcionalidades, o que ocorre é o exato oposto: quanto mais produtividade se deseja, mais elevado deve ser o padrão de qualidade interna.

...

Se você quer produzir mais, procure primeiramente aumentar sua qualidade interna.

CONTEÚDO RELACIONADO

[TDD Sem prática para testadores ágeis](#)

Leonardo Cassuriaga - 21 Ago 2015



[Quando TDD não é o suficiente](#)

Camilo Ribeiro - 03 Jun 2015



[Grails - alta produtividade em Java](#)

Henrique Lobo - 20 Mar 2015



[Lean para potencializar a qualidade no software](#)

Dionatan Moura - 14 Ago 2015



Exercício 7: Produtividade em Testes Automatizados



Testando o que é necessário


```
[TestMethod]
public void Deveria_avaliar_notas_ordem_crescente()
{
    Prova prova = new Prova(DateTime.Now);

    prova.LancarNota(5, new Aluno("Maria"));
    prova.LancarNota(8, new Aluno("João"));
    prova.LancarNota(10, new Aluno("José"));

    AvaliadorProva avaliador = new AvaliadorProva();

    avaliador.Avaliar(prova);

    Assert.AreEqual(10, avaliador.ObtemMaiorNota());

    Assert.AreEqual(5, avaliador.ObtemMenorNota());
}
```

Nosso primeiro teste automatizado para o **AvaliadorProva** garantimos que nosso algoritmo funcionará para uma lista de notas em ordem crescente. Mas será que só esse teste é suficiente?

```
[TestMethod]
public void Deveria_avaliar_notas_ordem_crescente_com_outros_valores()
{
    Prova prova = new Prova(DateTime.Now);

    prova.LancarNota(1, new Aluno("Maria"));
    prova.LancarNota(2, new Aluno("João"));
    prova.LancarNota(3, new Aluno("José"));

    AvaliadorProva avaliador = new AvaliadorProva();

    avaliador.Avaliar(prova);

    Assert.AreEqual(3, avaliador.ObtemMaiorNota());

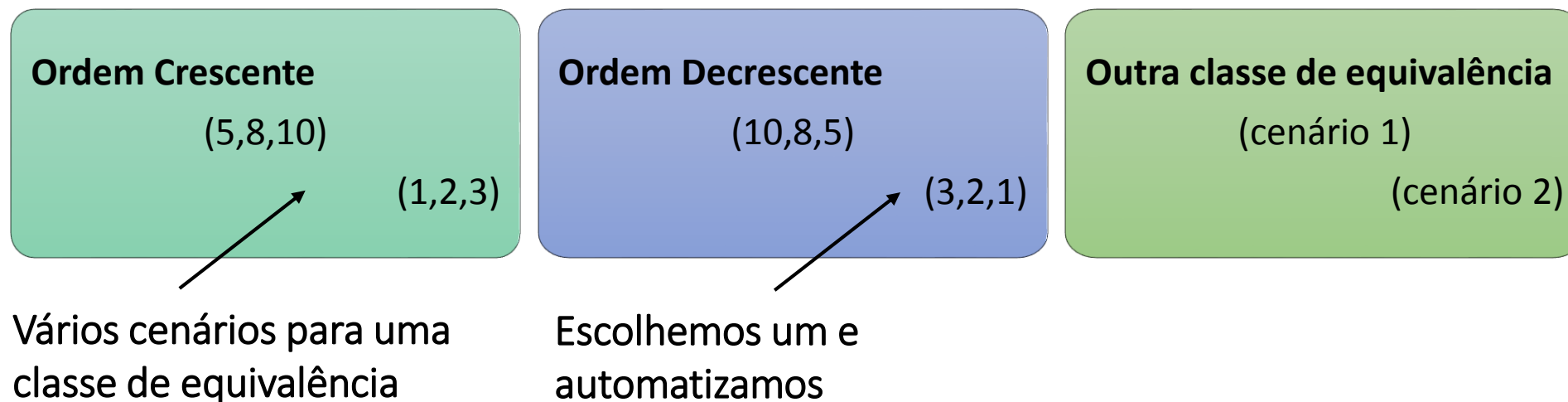
    Assert.AreEqual(1, avaliador.ObtemMenorNota());
}
```

Nosso primeiro teste automatizado para o **AvaliadorProva** garantimos que nosso algoritmo funcionará para uma lista de notas em ordem crescente. Mas será que só esse teste é suficiente?



Só quê, se tentarmos fazer isso, dificultamos a manutenção dos testes!

Classes de equivalência



O ideal é escrevermos apenas um único teste para cada possível cenário diferente!



Por exemplo, um cenário que levantamos é justamente notas em ordem crescente.



Precisamos de um teste por classe de equivalência.



A grande charada então é encontrar essas classes de equivalência.

Para esse nosso problema, por exemplo, é possível enxergar alguns diferentes cenários:

**Notas em ordem
crescente;**

**Notas em ordem
decrecente;**

**Notas sem nenhuma
ordem específica;**

**Apenas uma nota na
lista.**

Notas em ordem crescente;



Apenas uma nota na prova.

Notas em ordem decrescente;

Notas sem nenhuma ordem específica;

Notas em ordem crescente;



Apenas uma nota na prova.



Notas em ordem decrescente;



Notas sem nenhuma ordem específica;





**CHANGE
AHEAD**



Vamos implementar a próxima funcionalidade do AvaliadorProva, ele precisa agora retornar as 3 piores notas

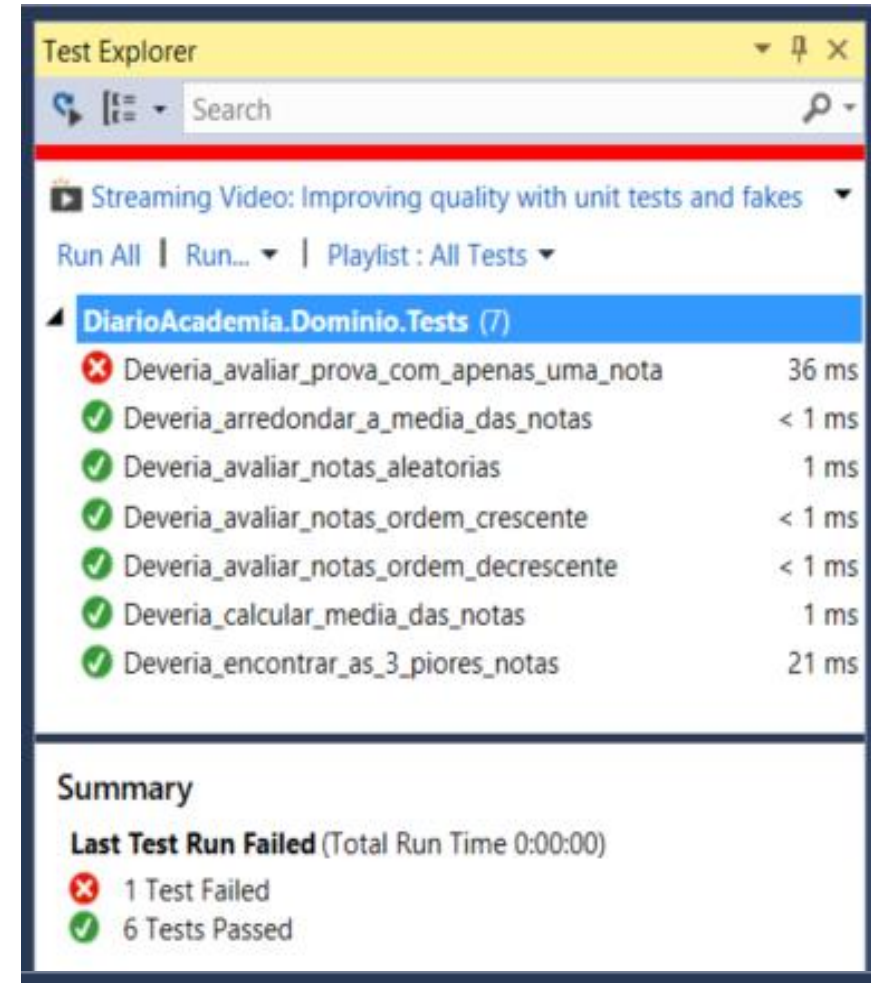
Será que perceberíamos isso se não tivéssemos a bateria de testes de unidade nos ajudando!



Veja a segurança que os testes nos dão.



Implementamos a nova funcionalidade, mas quebramos a anterior, e percebemos na hora!



Vantagens dos testes de regressão



Até agora, a asserção do nosso teste verifica o tamanho da lista. Será que é suficiente? Não!



Esse teste só garante que a lista tem três elementos, mas não garante o conteúdo desses elementos.



Sempre que testamos uma lista, além de verificar o tamanho dela precisamos verificar o conteúdo interno dela.

Verificações em Listas



Exercícios Testando o que é necessário

Escreva o teste para garantir que a classe AvaliadorProva funciona caso seja lançada apenas uma nota.

Para isso, crie um Prova, lance uma nota (com o valor de, por exemplo, 8), e invoque o Avaliar(Prova).

Para validar, verifique que tanto a maior nota quanto a menor sejam iguais a 8.

Exercício 1: Teste de um único lançamento de nota

```
[TestMethod]
public void Deveria_avaliar_prova_com_apenas_uma_nota()
{
    Prova prova = new Prova(DateTime.Now);

    prova.LancarNota(8, new Aluno("José"));

    AvaliadorProva avaliador = new AvaliadorProva();

    avaliador.Avaliar(prova);

    Assert.AreEqual(8, avaliador.ObtemMaiorNota());

    Assert.AreEqual(8, avaliador.ObtemMenorNota());
}
```

Exercício 1: Teste de um único lançamento de nota

Teste agora que o AvaliadorProva avalie uma prova cujos lançamentos das notas foram dados em ordem aleatória.

Para isso, crie uma Prova e lance notas randômicas, como por exemplo:
2, 9, 4, 3, 10, 5

Ao final, verifique que o menor é 2 e o maior é 10.

Exercício 2: Teste de lançamentos de notas em ordem aleatória


```
[TestMethod]
public void Deveria_avaliar_notas_aleatorias()
{
    Prova prova = new Prova(DateTime.Now);

    prova.LancarNota(8, new Aluno("João"));
    prova.LancarNota(10, new Aluno("José"));
    prova.LancarNota(5, new Aluno("Maria"));

    AvaliadorProva avaliador = new AvaliadorProva();

    avaliador.Avaliar(prova);

    Assert.AreEqual(10, avaliador.ObtemMaiorNota());

    Assert.AreEqual(5, avaliador.ObtemMenorNota());
}
```

Exercício 2: Teste de lançamentos de notas em ordem aleatória

Por fim, teste que o AvaliadorProva funciona para lançamentos de notas em ordem decrescente.

Lance notas com valores de 10, 8, 5 por exemplo, e garanta que a saída bata com os valores da entrada.

Exercício 3: Lançamentos de nota em ordem decrescente

```
[TestMethod]
public void Deveria_avaliar_notas_ordem_decrescente()
{
    Prova prova = new Prova(DateTime.Now);

    prova.LancarNota(10, new Aluno("José"));
    prova.LancarNota(8, new Aluno("João"));
    prova.LancarNota(5, new Aluno("Maria"));

    AvaliadorProva avaliador = new AvaliadorProva();

    avaliador.Avaliar(prova);

    Assert.AreEqual(10, avaliador.ObtemMaiorNota());

    Assert.AreEqual(5, avaliador.ObtemMenorNota());
}
```

Exercício 3: Lançamentos de nota em ordem decrescente

Ao testar uma lista, quantas verificações (quantidade de asserts) geralmente fazemos?

- a) ☐ Apenas 1, para garantir o tamanho da lista
- b) ☐ Apenas 2, uma para o tamanho da lista, e outra para garantir apenas o primeiro elemento dessa lista
- c) ☐ $1 + N$, onde o primeiro é para garantir o tamanho da lista, e depois N Asserts para garantir o conteúdo interno completo dessa lista
- d) ☐ Nenhuma

Exercício 4: Testando listas

Ao testar uma lista, quantas verificações (quantidade de asserts) geralmente fazemos?

- a) ☐ Apenas 1, para garantir o tamanho da lista
- b) ☐ Apenas 2, uma para o tamanho da lista, e outra para garantir apenas o primeiro elemento dessa lista
- c) ☒ $1 + N$, onde o primeiro é para garantir o tamanho da lista, e depois N Asserts para garantir o conteúdo interno completo dessa lista
- d) ☐ Nenhuma

Exercício 4: Testando listas

É sempre interessante tratar de casos especiais no teste. Por exemplo, tratamos o caso da lista com um elemento separado do caso da lista com vários elementos. Isso faz sentido?

Exercício 5: Casos que precisam de atenção no Teste

Tratar o caso da lista com um elemento separado do caso da lista com vários elementos faz todo sentido.



É muito comum pensarmos direto no caso complicado, e esquecermos de casos simples



Quando lidamos com listas é sempre interessante tratarmos o caso da lista cheia, com apenas um elemento e a lista vazia.



O grande desafio da área dos testadores é encontrar todas as classes de equivalência; tarefa essa que não é fácil !



Por exemplo:
`if(salario >= 2000)`
Precisa de três diferentes testes



Se estamos lidando com algoritmos cuja ordem é importante, precisamos testar ordem crescente, decrescente, randômica.

Exercício 5: Casos que precisam de atenção no teste

Escreva os seguintes testes:

- ✓ *Uma prova com 5 lançamentos, deve encontrar as três menores notas*
- ✓ *Uma prova com 2 lançamentos, deve devolver apenas duas notas*
- ✓ *Uma prova sem nenhum lançamento, deve devolver uma lista vazia*

Exercício 6: Três piores notas

```

[TestMethod]
public void Nao_deveria_retornar_notas_caso_nao_haja_lancamentos()
{
    Prova prova = new Prova(DateTime.Now);

    AvaliadorProva avaliador = new AvaliadorProva();

    avaliador.Avaliar(prova);

    Assert.AreEqual(0, avaliador.PioresNotas.Count);
}

[TestMethod]
public void Deveria_retornar_todas_as_notas_caso_nao_haja_no_minimo_3()
{
    Prova prova = new Prova(DateTime.Now);

    prova.LancarNota(1, new Aluno("João"));
    prova.LancarNota(2, new Aluno("José"));

    AvaliadorProva avaliador = new AvaliadorProva();

    avaliador.Avaliar(prova);

    Assert.AreEqual(2, avaliador.PioresNotas.Count);

    Assert.AreEqual(1, avaliador.PioresNotas[0].Valor);
    Assert.AreEqual(2, avaliador.PioresNotas[1].Valor);
}

```

```

[TestMethod]
public void Deveria_retornar_as_3_piores_notas()
{
    Prova prova = new Prova(DateTime.Now);

    prova.LancarNota(1, new Aluno("João"));
    prova.LancarNota(2, new Aluno("José"));
    prova.LancarNota(4, new Aluno("Maria"));
    prova.LancarNota(3, new Aluno("João"));
    prova.LancarNota(2, new Aluno("José"));
    prova.LancarNota(6, new Aluno("Maria"));

    AvaliadorProva avaliador = new AvaliadorProva();

    avaliador.Avaliar(prova);

    Assert.AreEqual(3, avaliador.PioresNotas.Count);

    Assert.AreEqual(1, avaliador.PioresNotas[0].Valor);
    Assert.AreEqual(2, avaliador.PioresNotas[1].Valor);
    Assert.AreEqual(2, avaliador.PioresNotas[2].Valor);
}

```

Exercício 6: Três piores notas

Quando implementamos a regra de negócio para cálculo das três menores notas e quebramos a regra de negócio anterior, será que **sem a bateria de testes** perceberíamos esse erro? Como?

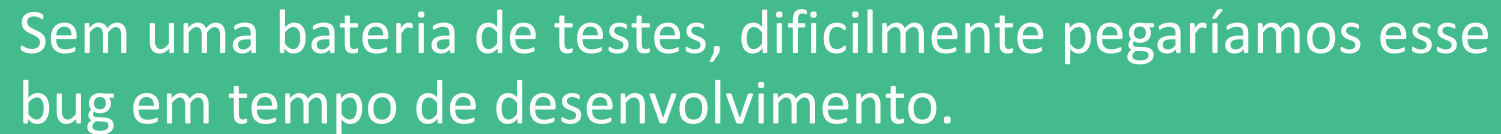
Nesse momento, quais são as vantagens de uma bateria de testes automatizados?

Exercício 7: Perceber erros sem testes de unidade

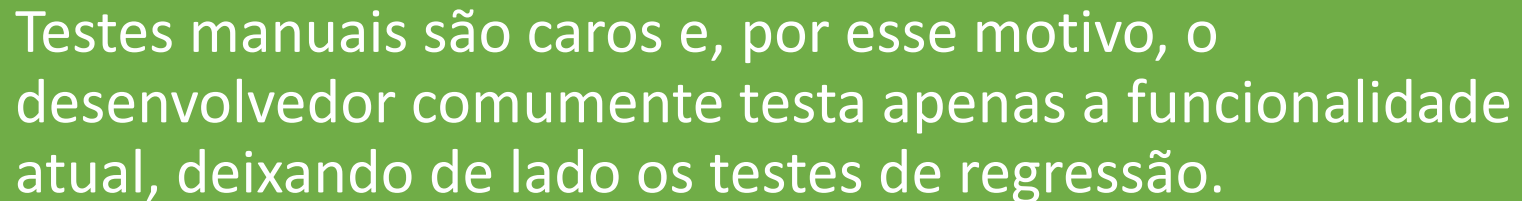
A bateria de testes automatizados nos ajuda a encontrar problemas na nossa implementação de forma muito rápida



Sem uma bateria de testes, dificilmente pegaríamos esse bug em tempo de desenvolvimento.



Testes manuais são caros e, por esse motivo, o desenvolvedor comumente testa apenas a funcionalidade atual, deixando de lado os testes de regressão.



Exercício 7: Perceber erros sem testes de unidade

É importante testarmos todos os caminhos possíveis do nosso código. Abaixo temos o método a ser testado, para garantir que esse método realmente funcione, qual a quantidade mínima de testes que precisamos ter?

- a) ☐ 1
- b) ☐ 20
- c) ☐ 2
- d) ☐ Próximo ao Infinito
- e) ☐ 3
- f) ☐ Nenhum

0 references

```
public int ContaMaluca(int numero)
{
    if (numero > 30) return numero * 4;
    else if (numero > 10) return numero * 3;
    else return numero * 2;
}
```

Exercício 8: Quantidade de Testes

É importante testarmos todos os caminhos possíveis do nosso código. Abaixo temos o método a ser testado, para garantir que esse método realmente funcione, qual a quantidade mínima de testes que precisamos ter?

- a) ☐ 1
- b) ☐ 20
- c) ☐ 2
- d) ☐ Próximo ao Infinito
- e) ☒ 3
- f) ☐ Nenhum

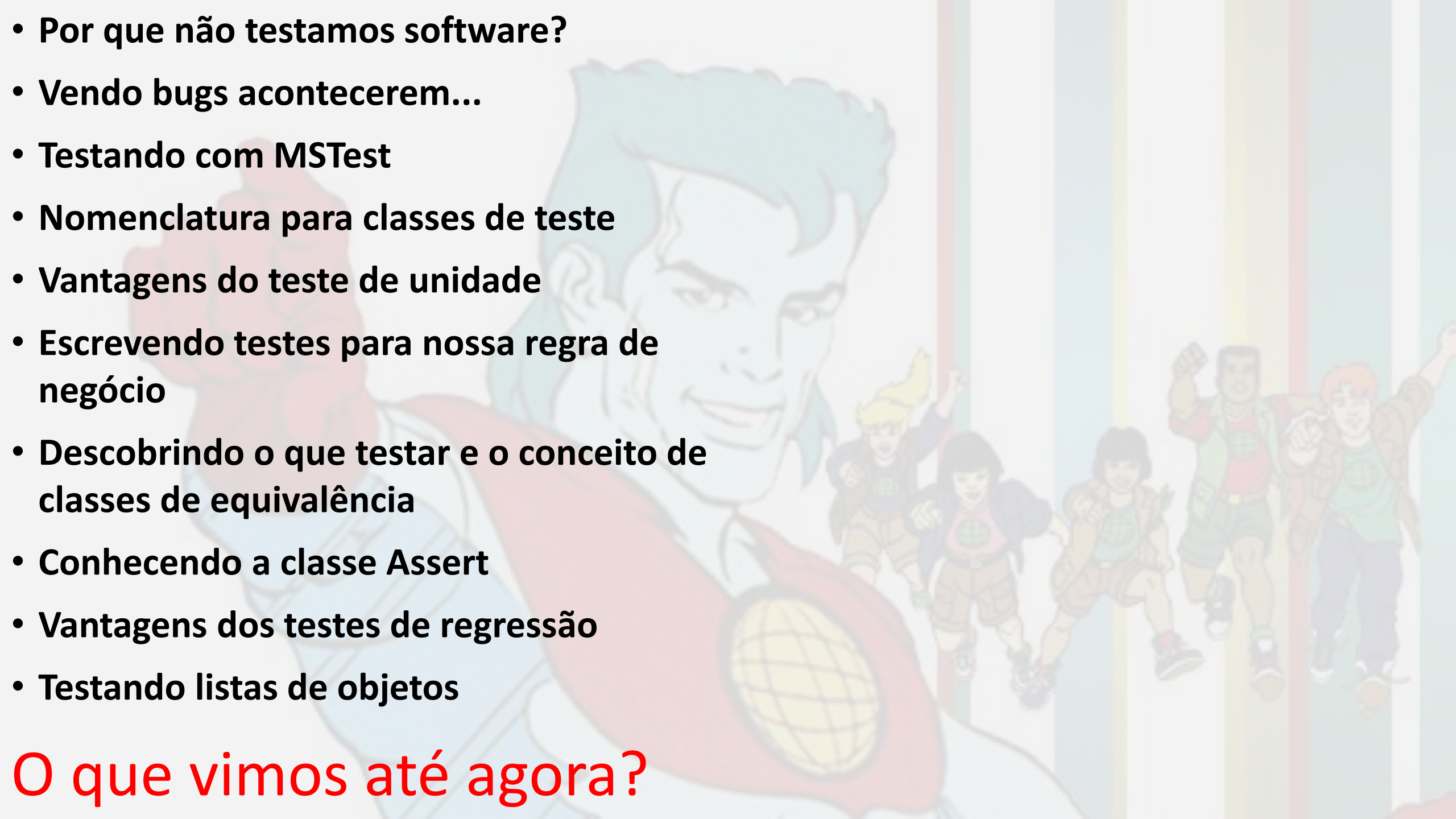
0 references

```
public int ContaMaluca(int numero)
{
    if (numero > 30) return numero * 4;
    else if (numero > 10) return numero * 3;
    else return numero * 2;
}
```

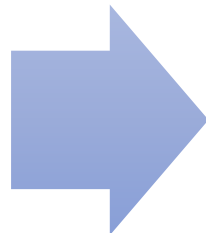
Exercício 8: Quantidade de Testes

- **Por que não testamos software?**
- **Vendo bugs acontecerem...**
- **Testando com MSTest**
- **Nomenclatura para classes de teste**
- **Vantagens do teste de unidade**
- **Escrevendo testes para nossa regra de negócio**
- **Descobrimo o que testar e o conceito de classes de equivalência**
- **Conhecendo a classe Assert**
- **Vantagens dos testes de regressão**
- **Testando listas de objetos**

O que vimos até agora?



Implemente um método no **AvaliadorProva** que **arredonda** o valor médio dos lances.



Escreva também os testes para isso.

Por exemplo:

Condição	Resultado
Se as casas decimais for menor que 0.25	Arredonda para "0"
Se as casas decimais for maior que 0.25 e menor que 0.50	Arredonda para "0.5"
Se as casas decimais for maior que 0.50 e menor que 0.75	Arredonda para "0.5"
Se as casas decimais for maior que 0.75 e menor que 1	Arredonda para "1"

Exercício 9: Arredondando a Média das Notas

1 reference | 1/1 passing

```
public double ObtemMedia()  
{  
    return ArredondarMedia(_mediaNotas);  
}
```

1 reference

```
private double ArredondarMedia(double media)  
{  
    double decimais = media % 1;  
  
    if (decimais < 0.25) decimais = 0;  
  
    else if (decimais < 0.75) decimais = 0.50;  
  
    else decimais = 1;  
  
    return ((int)media) + decimais;  
}
```

Exercício 9: Será que faz sentido termos testes para todas as nossas linhas de código?

TEST ARCHITECTURE ANALYZE WIN...

- Run
- Debug
- Test Settings
- Analyze Code Coverage
- Windows

Test Explorer

Run All | Run...

Passed Tests (3)

- QuickNonZero 15 ms
- RootTestNeg... 13 ms
- SignatureTest 1 ms

Code Coverage Results

ctsoasm_MAIN50531 2012-06-07 02...

Hierarchy	Not Cov...	Not Covered (%...	Cov...
ctsoasm_MAIN50531 201...	44	80.00%	11
fabrikam.math.dll	7	50.00%	7
{ } Fabrikam.Math	7	50.00%	7

```
public double SquareRoot(double x)
{
    if (x < 0.0)
    {
        throw new ArgumentOutOfRangeException();
    }
    double estimate = x;
    double previousEstimate = -x;
    while (System.Math.Abs(estimate - previousEstimate) >...
    {
```

Not covered

Covered

Turn on coloring

Code Coverage

Cobertura de código: É uma métrica que indica a efetividade dos testes feitos em uma aplicação. Expressa em termos de porcentagem do código-fonte da aplicação, mostra exatamente o quanto da aplicação foi testada durante uma dada bateria de testes;

```
public double ObterMedia()  
{  
    return ArredondarMedia(_mediaNotas);  
}  
private double ArredondarMedia(double media)  
{  
    double decimais = media % 1;  
    if (decimais < 0.25) decimais = 0;  
    else if (decimais < 0.75) decimais = 0.50;  
    else decimais = 1;  
    return ((int)media) + decimais;  
}
```

Escreva testes para todos os cenários de arredondamento

Condição	Saída	Notas	Média	Média Arredondada
Se as casas decimais for menor que 0.25	Arredonda para “0”	5 + 0,5 + 10	5,166666667	5
Se as casas decimais for maior que 0.25 e menor que 0.50	Arredonda para “0.5”	5 + 7 + 10	7,333333333	7,5
Se as casas decimais for maior que 0.50 e menor que 0.75	Arredonda para “0.5”	5 + 8 + 10	7,666666667	7,5
Se as casas decimais for maior que 0.75 e menor que 1	Arredonda para “1”	5 + 8 ,5 + 10	7,833333333	8

Exercício 10: Mais testes no arredondando da Média das notas

[TestClass]
0 references | alexandre.rech, 11 hours ago | 7 changes
public class AvaliadorProvaTest
{
 atributos privados

[TestInitialize]
0 references | alexandre.rech, 11 hours ago | 3 changes
public void TestInitialize()...

(Ferramenta de Feedback)

[TestMethod]
✓ | 0 references | 0 authors | 0 changes
public void Deveria_arredondar_a_media_pra_baixo_caso_os_decimais_seja_menor_que_25()...

[TestMethod]
✓ | 0 references | 0 authors | 0 changes
public void Deveria_arredondar_a_media_pra_meio_ponto_caso_os_decimais_seja_menor_que_75()...

[TestMethod]
✓ | 0 references | 0 authors | 0 changes
public void Deveria_arredondar_a_media_pra_cima_caso_os_decimais_seja_maior_que_75()...

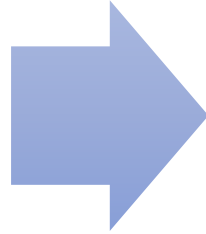
[TestMethod]
✓ | 0 references | alexandre.rech, 11 hours ago | 4 changes
public void Deveria_avaliar_notas_ordem_crescente()...

[TestMethod]
✓ | 0 references | alexandre.rech, 11 hours ago | 3 changes
public void Deveria_avaliar_notas_ordem_decrescente()...

[TestMethod]
✓ | 0 references | alexandre.rech, 11 hours ago | 3 changes
public void Deveria_avaliar_notas_aleatorias()...

Exercício 10: Mais testes no arredondando da Média das notas

Crie uma classe específica para fazer o arredondamento da média das notas da prova.

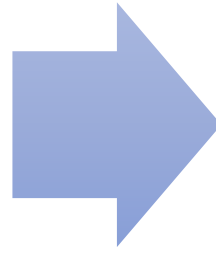


Escreva também os testes para isso.

Exercício 11: Melhorar o Design da funcionalidade de arredondamento (**extrair método para objeto método**)

Na opinião de vocês, qual a parte mais difícil na hora de escrever um teste de unidade?

Desenvolvedores que estão aprendendo a testar geralmente sentem **difficulties** no momento de **levantar** e **escrever cenários** **para o teste.**



Foque-se na classe que você está testando. Pense sobre o que você espera dela. Como ela deve funcionar? Se você passar tais parâmetros para ela, como ela deve reagir?

Exercício 12: Dificuldade nos testes de unidade



Então você é desses? Usa uma muleta pra não fazer testes?



LeanPyramid



Inscrito



130

207 visualizações



Adicionar a



Compartilhar



Mais



16






0

Tarefa: Assistir vídeo do Samuel Crescencio

Core of TDD

③

- **RED**: test fails 
- **GREEN**: test passes 
- **REFACTOR**
↳ CLEAN Code + Tests 

Praticando Test Driven Development

```
public class Prova
{
    public Prova(DateTime data)
    {
        Data = data;
        Notas = new List<Nota>();
    }

    public int Id { get; set; }

    public DateTime Data { get; set; }

    public List<Nota> Notas { get; private set; }

    public string Assunto { get; set; }

    public void LancarNota(Nota nota)
    {
        nota.Prova = this;

        Notas.Add(nota);
    }
}
```

Ainda temos muito código não testado no nosso sistema!

```
[TestClass]
0 references
public class ProvaTest
{
    [TestMethod]
    ✔ | 0 references
    public void Deveria_receber_um_lancamento_de_nota(...)

    [TestMethod]
    ✔ | 0 references
    public void Deveria_receber_varios_lancamentos_de_notas(...)
}
```

Abaixo estão os dois casos a serem averiguados:

- A realização de apenas um lançamento de nota;
- E a realização de mais de um lançamento de notas;

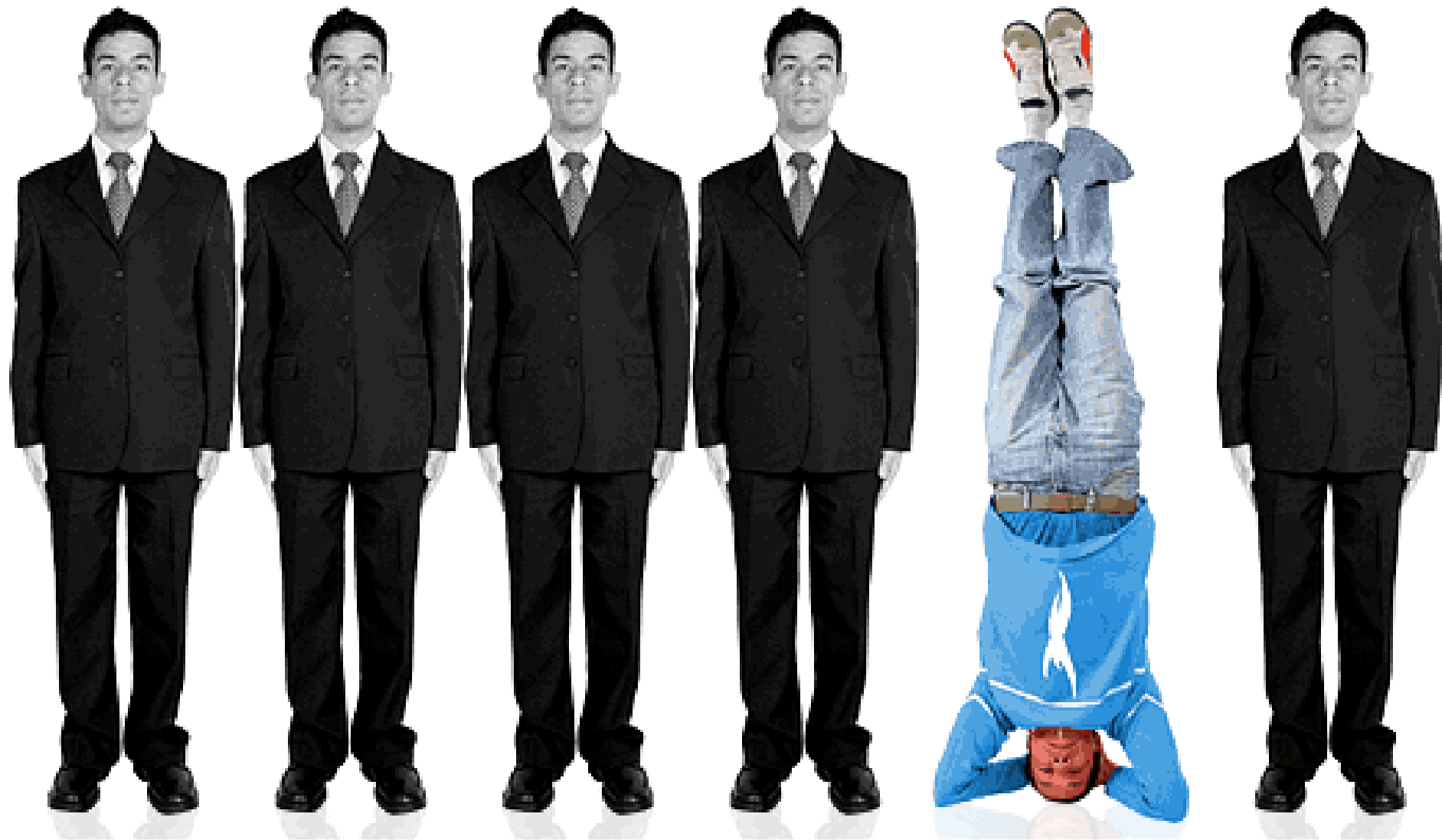


Nova funcionalidade chegando....



Agora implementaremos duas novas regras de negócio no processo de lançamento de notas:

- Um prova não pode receber dois lançamentos de notas para o mesmo aluno
- Um prova não pode receber o lançamento de uma nota caso o aluno esteja reprovado por faltas
 - O aluno está reprovado quando a quantidade de faltas é maior que 5



Estamos acostumados a implementar o código de produção e testá-lo ao final.
Mas será que essa é a única maneira de desenvolver um projeto com testes automatizados?

Primeiro

Implementamos
Funcionalidades

Depois

Implementamos
Testes Automatizados

Test Driven Development ou TDD

The diagram illustrates the TDD cycle with three main steps represented by colored circles and arrows:

- Write tests that fail (Vimos ele falhar):** Represented by an orange circle on the left. An orange arrow points from this circle to the top circle.
- Write code to pass the tests (Escrevemos o código para o teste passar):** Represented by a light green circle at the top. A light green arrow points from this circle to the bottom circle.
- Refactor production code (Refatoramos o código de produção):** Represented by a light green circle on the right. A light green arrow points from this circle back to the top circle, completing the cycle.

Additional text at the bottom left reads: "Escrevemos o código de produção da maneira mais simples" (We write the production code in the simplest way).



Babys Steps

Esrever o código mais simples pro teste passar

- Começar pelo **cenário de teste mais simples** naquele momento
- Começar pelo mais simples **nos possibilita evoluir o código aos poucos**

Códigos simples aumentam a facilidade de manutenção

- Muitas vezes **nós, programadores, escrevemos códigos complicados** desnecessariamente
- TDD nos lembra o tempo todo de ser simples.

Passos de bebê o tempo todo pode ser contraproducente

- Kent Beck diz que você deve **tomar passos pequenos** sempre que sua “confiança sobre o código” **estiver baixa**.
- Se você está trabalhando em um trecho de código e **está bastante confiante** sobre ele, **você pode dar passos um pouco maiores**.

Cuidado

- **Passos grandes** não devem ser a regra, e sim **a exceção**.

Nasce Testado

- Se sempre escrevermos o teste antes, garantimos que todo nosso código já “nasce” testado;

Segurança p/ refatorar

- Temos segurança para refatorar nosso código, afinal sempre refatoraremos com uma bateria de testes que garante que não quebraremos o comportamento já existente;

Melhora a manutenção

- Como o teste é a primeira classe que usa o seu código, você naturalmente tende a escrever código mais fácil de ser usado e, por consequência, mais fácil de ser mantido.

Documentação p/ programadores

- A suíte de testes serve como uma documentação voltada para o programador que tem um entendimento mais rápido e facilitado do que uma parte do código faz através do código que o testa.

Vantagens do TDD

Ele é mais coeso.

- Afinal, um código que faz muita coisa é **mais difícil de ser testado.**

Ele é menos
acoplado.

- Pois testar código acoplado é mais difícil.

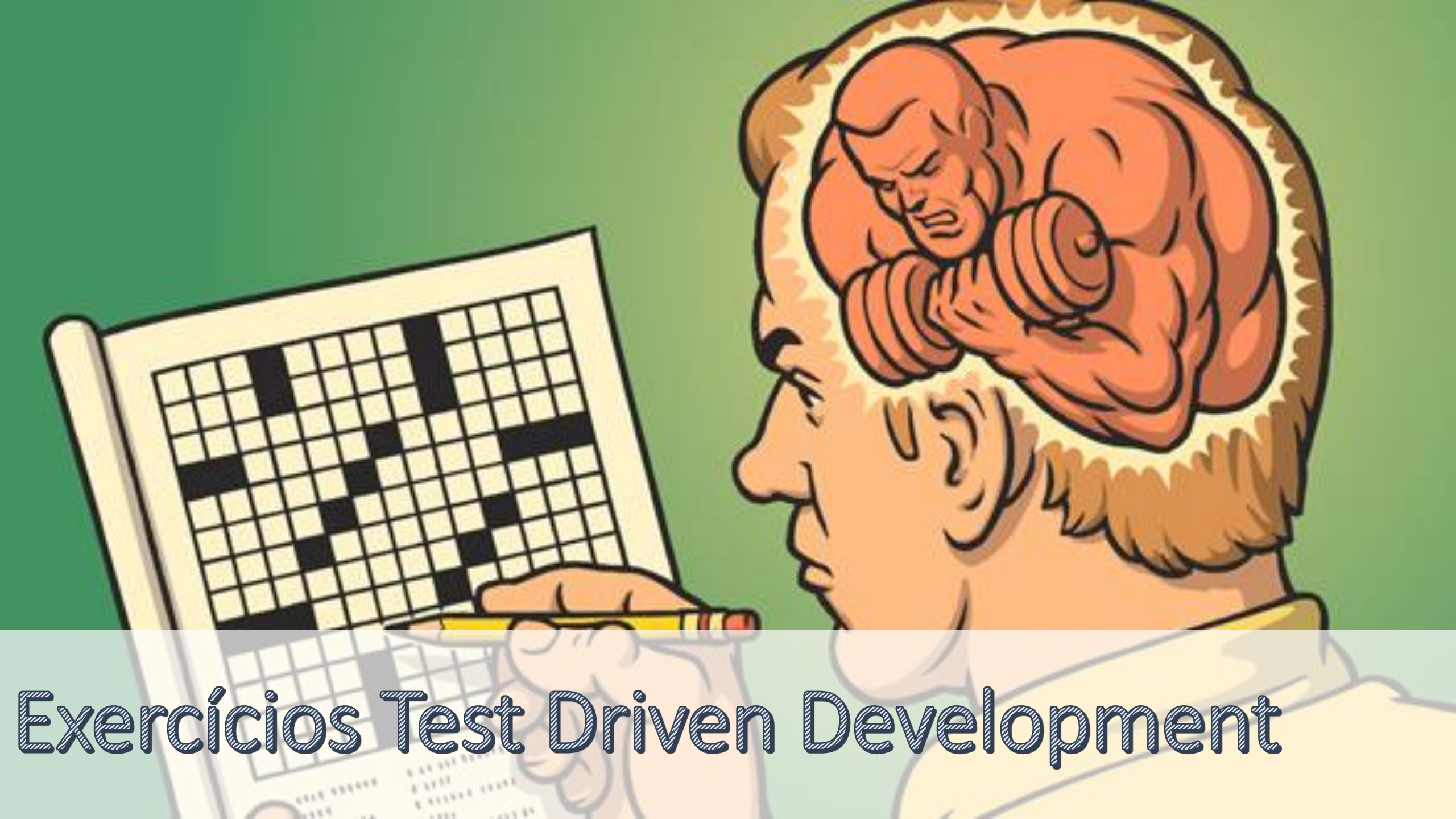
A interface pública
é simples.

- Você não quer invocar 10 métodos para conseguir testar o comportamento.

As pré-condições
são mais simples.

- Você também não quer montar imensos cenários para testar o método.

Vantagens do TDD: Efeito Positivo no Design de Classes



Exercícios Test Driven Development

Implemente as mesmas funcionalidades apresentadas anteriormente.

Uma prova não deve aceitar:

- Não pode receber dois lançamentos de notas para o mesmo aluno
- Não pode receber o lançamento de uma nota caso o aluno esteja reprovado por faltas
 - O aluno está reprovado quando a quantidade de faltas é maior que 5

Em todos os casos, o lançamento da nota deverá ser ignorado

Exercício 1: Praticando TDD

```
public double CalculaImposto(NotaFiscal nf)
{
    if (nf.Valor > 2000) return nf.Valor * 0.03;

    return nf.Valor * 0.02;
}
```

Exercício 2: Testar códigos simples?

Sim. O método `CalculaImposto()` é simples, mas contém uma **regra de negócio importante**. Como **garantir** que **ela funcionará para sempre** e que **mudanças** nesse código **não quebrarão** essa regra?

Sempre que **escrevemos um trecho de código, achamos ele simples**. Afinal, nós o escrevemos, ele está inteiro na nossa cabeça. Não há dúvidas. Mas quantas vezes **voltamos em um código** que escrevemos 1 mês atrás **e não entendemos nada?**

A única maneira de trabalhar com **qualidade e segurança em códigos assim é tendo uma bateria de testes que garanta qualquer mudança feita**. Portanto, não se deixe enganar. **Se o método contém uma regra de negócio, teste-o. Você agradecerá no futuro.**

Exercício 2: Testar códigos simples?

Implementem o método **LancarNota(Gabarito respostas, Aluno aluno)** na classe prova.

Este método deve calcular a nota da prova a partir das respostas do aluno.

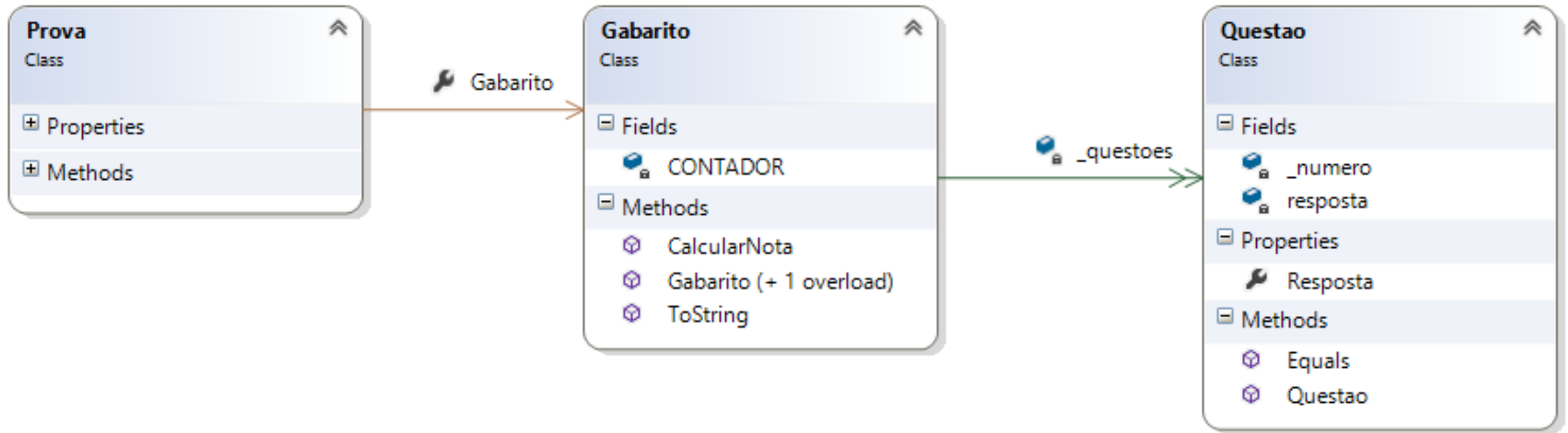
- Reparem que ainda precisamos seguir as regras dos lançamentos de nota:
 - Não pode receber dois lançamentos de notas para o mesmo aluno
 - Não pode receber o lançamento de uma nota caso o aluno esteja reprovado por faltas

Pratiquem TDD 100% do tempo durante a implementação.

Exercício 3: Provas recebendo gabaritos

Implementem o método *LancarNota(Gabarito respostas, Aluno aluno)* na *cli*

Es



Pr

Exercício 3: Provas recebendo gabaritos

```

public class Gabarito
{
    private static int CONTADOR = 0;

    private List<Questao> _questoes = new List<Questao>();

    public Gabarito(string respostas)
        : this(respostas.ToCharArray())
    {
    }

    public Gabarito(params char[] respostas)
    {
        foreach (char resposta in respostas)
        {
            Questao questao = new Questao(++CONTADOR, resposta);

            _questoes.Add(questao);
        }
    }

    public double CalcularNota(Gabarito respostas)
    {
        ///Implementar....
    }

    public override string ToString()
    {
        if(_questoes.Any())
            return string.Join("", _questoes.Select(q => q.Resposta).ToArray());

        return "";
    }
}

```

```

public class Questao
{
    private int _numero;
    private char resposta;

    public Questao(int numero, char resposta)
    {
        this._numero = numero;
        this.resposta = resposta;
    }

    public char Resposta { get { return resposta; } }

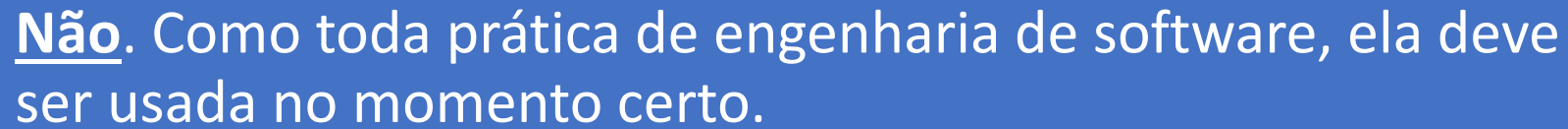
    public override bool Equals(object obj)
    {
        Questao questao = (Questao)obj;

        return questao.resposta == this.resposta;
    }
}

```

Exercício 3: Provas recebendo gabaritos

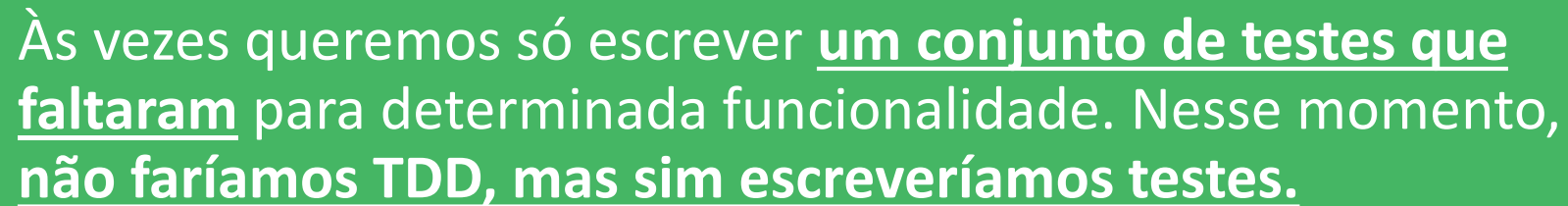
Não. Como toda prática de engenharia de software, ela deve ser usada no momento certo.



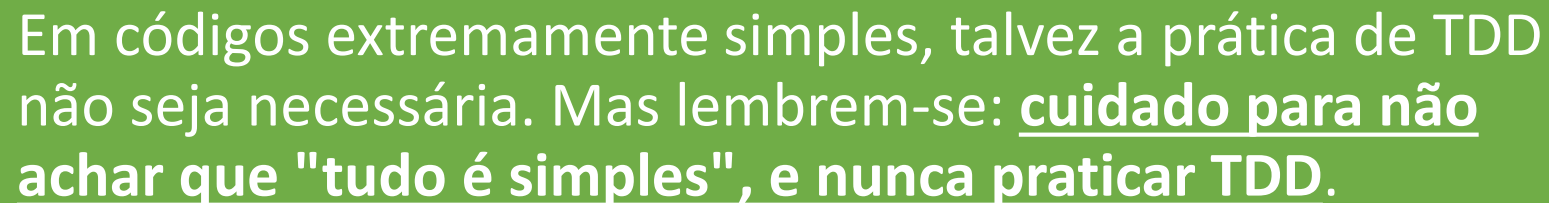
TDD faz muito sentido ao implementar novas funcionalidades, ao corrigir bugs, ao trabalhar em códigos complexos, etc.



Às vezes queremos só escrever um conjunto de testes que faltaram para determinada funcionalidade. Nesse momento, não faríamos TDD, mas sim escreveríamos testes.



Em códigos extremamente simples, talvez a prática de TDD não seja necessária. Mas lembrem-se: cuidado para não achar que "tudo é simples", e nunca praticar TDD.



Exercício 4: TDD 100% do tempo?

Mostrar Provas em meses e anos

Caso a Prova tenha mais de 30 dias uma mensagem deve aparecer
“Prova realizada 1 mês atrás”

Caso a Prova tenha mais de 60 dias a mensagem deve aparecer
“Prova realizada 2 meses atrás”

E assim por diante....

Mostrar Provas em horas, minutos e segundos

Caso a Prova tenha mais de 60 minutos a mensagem deve aparecer
“Prova realizada 1 hora atrás”

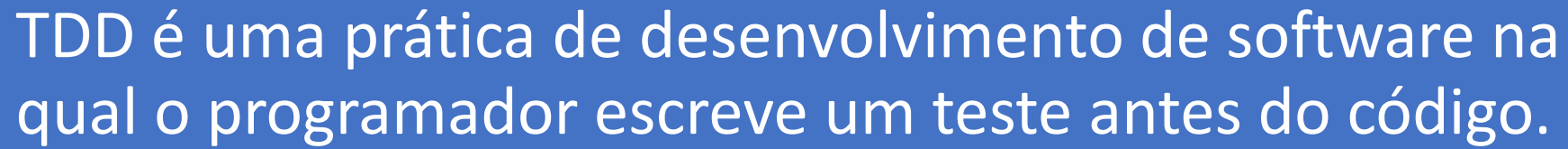
Caso a Prova tenha mais de 120 minutos a mensagem deve aparecer
“Prova realizada 2 horas atrás”

Caso a Prova tenha mais de 60 segundos a mensagem deve aparecer
“Prova realizada 1 minuto atrás”

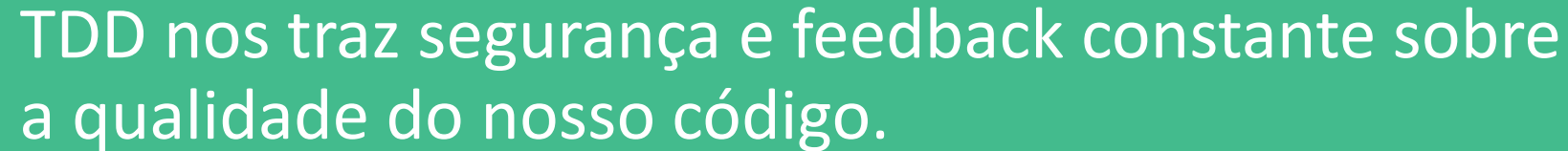
E assim por diante....

Exercício 5: Tempos atrás...

TDD é uma prática de desenvolvimento de software na qual o programador escreve um teste antes do código.



TDD nos traz segurança e feedback constante sobre a qualidade do nosso código.



É uma boa prática para todo desenvolvedor de software!



Exercício 6: O que é TDD?

Testes de Unidade e TDD

Treinamento em Testes automatizados na plataforma .net

Professor: Alexandre Rech

