

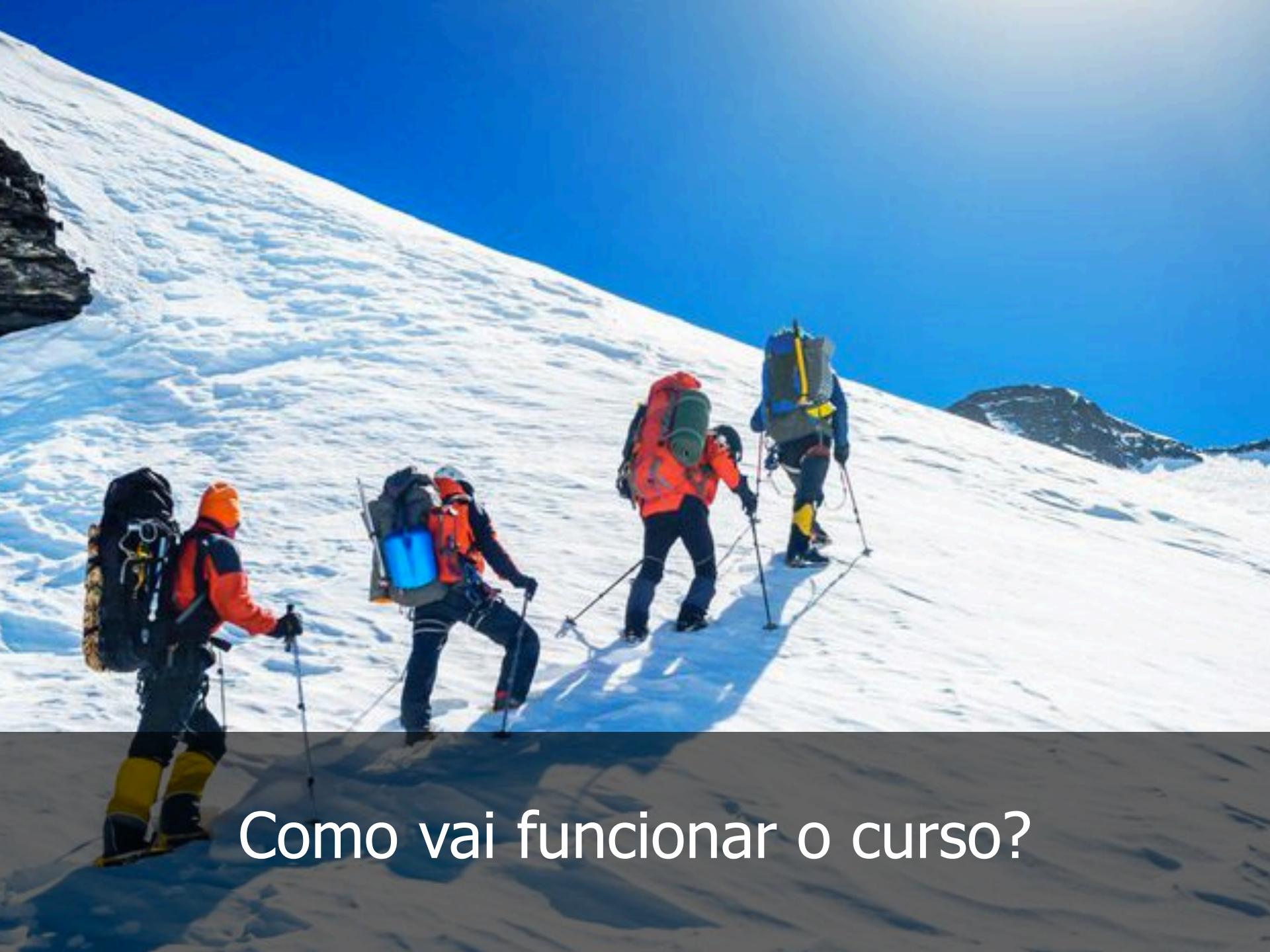


# Clean Code e Clean Architecture

## Rodrigo Branas



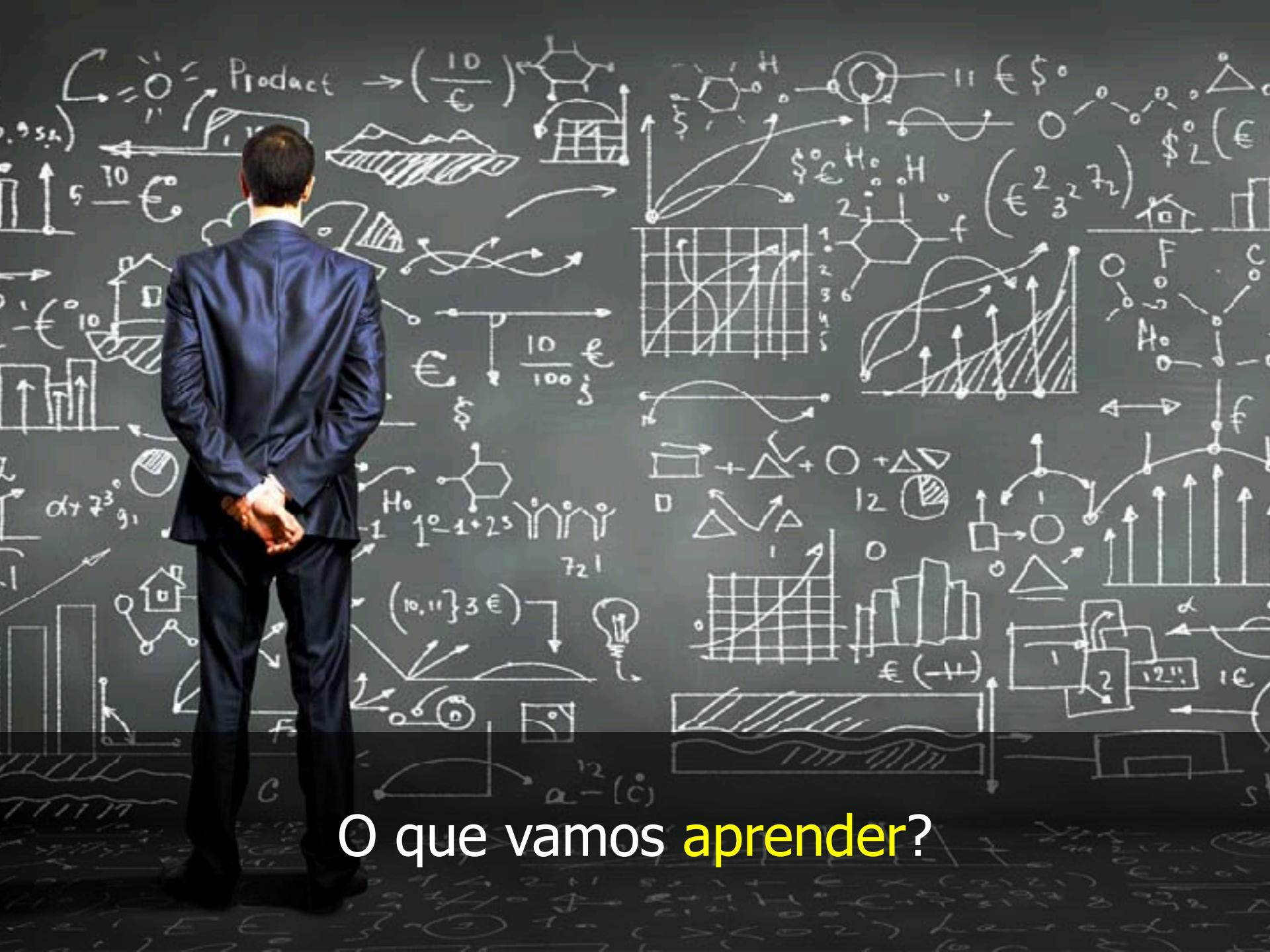
Quem sou eu e como comecei nesse **assunto?**



Como vai funcionar o curso?



As aulas vão ficar gravadas?



O que vamos aprender?

```
check None":  
    ection() {  
        d.getElementsByTagName("input")  
            .filter(e => e.type === "checkbox" &  
                e.checked & !e.hasAttribute("checked"))  
                .forEach(e => e.checked = false);  
        }  
    }  
    checkNone();  
    checkNone();  
    checkNone();
```

Qual **tecnologia** será abordada?



Como será o projeto que vamos desenvolver?



Mostrar modelagem do projeto

# Qual é o resultado esperado?

- Tenha desenvolvido um olhar mais crítico e profissional em relação ao desenvolvimento de software, elevando seu nível de maturidade
- Aprenda a tomar decisões de design e arquitetura de forma consciente, colocando na balança as vantagens e desvantagens de cada uma delas
- Saiba aplicar diversas técnicas de Clean Code e Refactoring com o objetivo de tornar o código mais limpo e comprehensível

# Qual é o resultado esperado?

- Domine diversos Design Patterns tanto do GoF (Gang of Four) quanto do PoEAA (Patterns of Enterprise Application Architecture) entendendo como eles podem ajudar a criar código mais desacoplado e manutenível
- Entenda como desenvolver utilizando Test-Driven Development com Test Patterns como Stub, Spy, Mock e Fake
- Domine os princípios da Programação Orientada a Objetos

# Qual é o resultado esperado?

- Saiba como aplicar Ports and Adapters, Clean Architecture e Domain-Driven Design
- Entenda os princípios de Event-Driven Architecture e a importância do CQRS
- Saiba aplicar os princípios do SOLID de uma vez por todas
- Aprenda como construir uma arquitetura de microservices de uma forma resiliente e escalável

# Qual é o resultado esperado?

- Saiba construir um frontend com o mesmo nível de qualidade do backend, utilizando TDD, Ports and Adapters, Clean Architecture e Design Patterns
- Se torne uma pessoa muito mais confiante e segura no dia a dia de trabalho, em entrevistas de emprego, dando mais um passo na evolução da sua carreira
- Busque fomentar um ambiente de trabalho com foco na qualidade, fazendo a diferença dentro da sua equipe

Porque o projeto que você trabalha tem o  
design e a arquitetura que tem?

1. Não sei, já estava assim quando eu cheguei
2. Copiamos de outro projeto que já existia pra seguir o padrão
3. Mandaram fazer dessa forma
4. Seguimos um tutorial que alguém escreveu
5. A documentação do framework disse que era pra fazer assim
6. Não sei exatamente, foi aleatório

# Clean Code

```
    "}; a = replaceAll(",", "", "", a); a = a.replace(" ", "");
    return a.split(" "); } $($("#unique").val());
  } else a = array_from_string($("#begin").val(), c = use_unique(array_from_string($("#end").val())));
  if (c < 2 * b - 1) { if (c == 0) {
    this.trigger("click"); } else if (c == 1) {
    $("#water_logged").val(); c = array_from_string($("#end").val(), c.length); b++;
    for (b = 0; b < c.length; b++) { if (-1 != a.indexOf(c[b])) {
      this.click(function() {
        $(this).text("Clicked");
        $(this).click(function() {
          $(this).text("Clicked");
        });
      });
    }
  }
}
});
```

Você já trabalhou em um projeto onde...

- O código-fonte é tão bagunçado que **você não sabe nem por onde começar...**

## Você já trabalhou em um projeto onde...

- O código-fonte é tão bagunçado que **você não sabe nem por onde começar...**
- Existem partes do projeto que só uma pessoa **sabe mexer** (talvez essa pessoa seja você)

## Você já trabalhou em um projeto onde...

- O código-fonte é tão bagunçado que você não sabe nem por onde começar...
- Existem partes do projeto que só uma pessoa sabe mexer (talvez essa pessoa seja você)
- Você já mexeu em uma coisa e estragou outra?

## Você já trabalhou em um projeto onde...

- O código-fonte é tão bagunçado que você não sabe nem por onde começar...
- Existem partes do projeto que só uma pessoa sabe mexer (talvez essa pessoa seja você)
- Você já mexeu em uma coisa e estragou outra?
- Tem mais defeito pra corrigir do que funcionalidade nova pra implementar

## Você já trabalhou em um projeto onde...

- O código-fonte é tão bagunçado que você não sabe nem por onde começar...
- Existem partes do projeto que só uma pessoa sabe mexer (talvez essa pessoa seja você)
- Você já mexeu em uma coisa e estragou outra?
- Tem mais defeito pra corrigir do que funcionalidade nova pra implementar
- Tudo é urgente e é necessário parar uma coisa pra resolver outra o tempo todo

## Você já trabalhou em um projeto onde...

- O código-fonte é tão bagunçado que você não sabe nem por onde começar...
- Existem partes do projeto que só uma pessoa sabe mexer (talvez essa pessoa seja você)
- Você já mexeu em uma coisa e estragou outra?
- Tem mais defeito pra corrigir do que funcionalidade nova pra implementar
- Tudo é urgente e é necessário parar uma coisa pra resolver outra o tempo todo
- Você tem medo de fazer deploy?



Porque isso acontece?

```
2445     if ((lowerChar == Character.ERROR) ||
2446         (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
2447         if (lowerChar == Character.ERROR) {
2448             lowerCharArray =
2449                 ConditionalSpecialCasing.ToLowerCaseCharArray(this, i, locale);
2450         } else if (srcCount == 2) {
2451             resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - srcCount;
2452             continue;
2453         } else {
2454             lowerCharArray = Character.toOthers(lowerChar);
2455         }
2456
2457         /* Grow result if needed */
2458         int mapLen = lowerCharArray.length;
2459         if (mapLen > srcCount) {
2460             char[] result2 = new char[result.length + mapLen - srcCount];
2461             System.arraycopy(result, 0, result2, 0,
2462                             i + resultOffset);
2463             result = result2;
2464         }
2465         for (int x=0; x<mapLen; ++x) {
2466             result[i+resultOffset+x] = lowerCharArray[x];
2467         }
2468         resultOffset += (mapLen - srcCount);
2469     } else {
2470         result[i+resultOffset] = (char)lowerChar;
2471     }
2472 }
2473 return new String(0, count+resultOffset, result);
2474 }
```



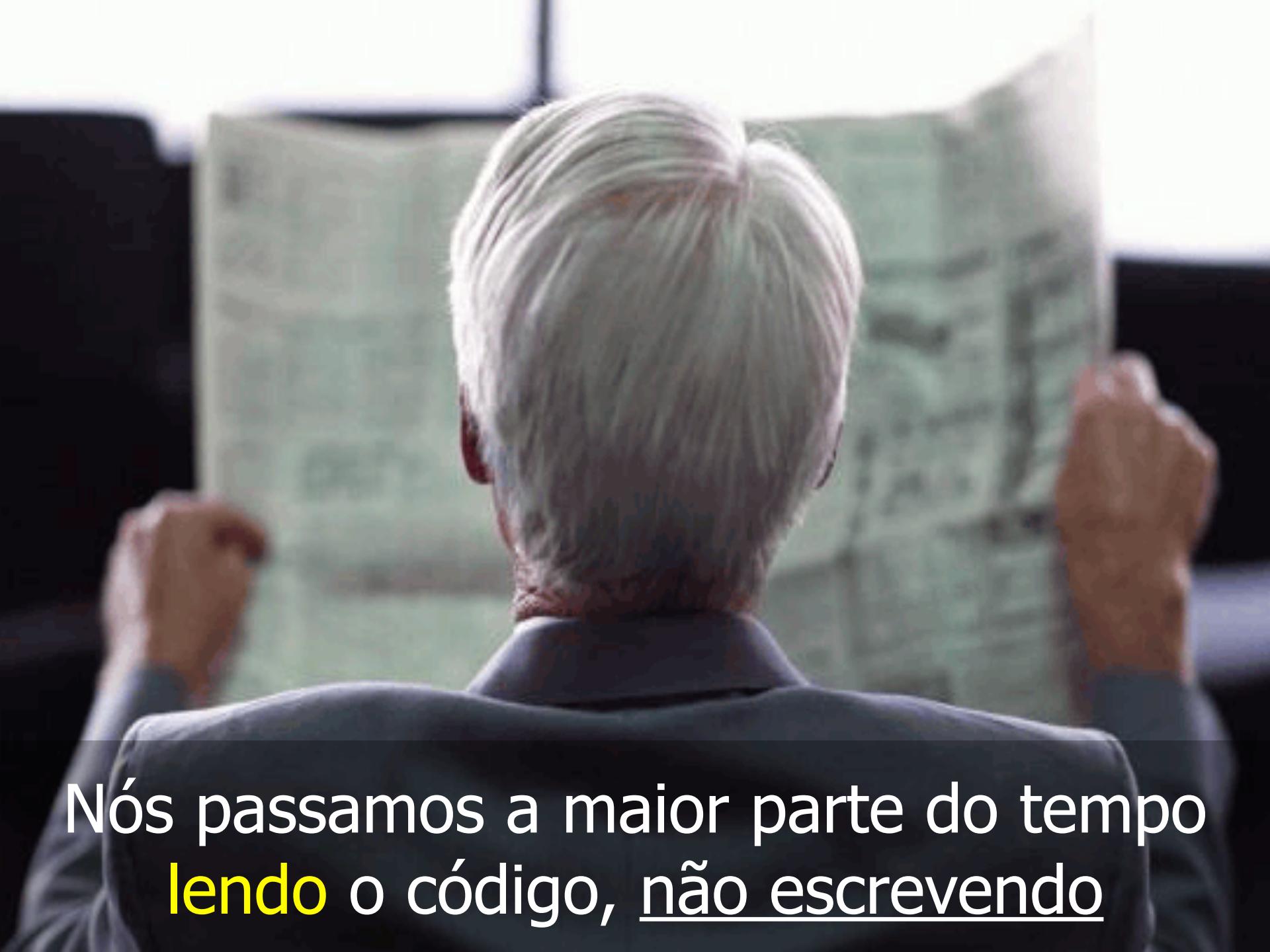
O que tem de **estranho** nesse código?

```
2445     if ((lowerChar == Character.ERROR) ||
2446         (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
2447         if (lowerChar == Character.ERROR) {
2448             lowerCharArray =
2449                 ConditionalSpecialCasing.ToLowerCaseCharArray(this, i, locale);
2450         } else if (srcCount == 2) {
2451             resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - srcCount;
2452             continue;
2453         } else {
2454             lowerCharArray = Character.toOthers(lowerChar);
2455         }
2456
2457         /* Grow result if needed */
2458         int mapLen = lowerCharArray.length;
2459         if (mapLen > srcCount) {
2460             char[] result2 = new char[result.length + mapLen - srcCount];
2461             System.arraycopy(result, 0, result2, 0,
2462                             i + resultOffset);
2463             result = result2;
2464         }
2465         for (int x=0; x<mapLen; ++x) {
2466             result[i+resultOffset+x] = lowerCharArray[x];
2467         }
2468         resultOffset += (mapLen - srcCount);
2469     } else {
2470         result[i+resultOffset] = (char)lowerChar;
2471     }
2472 }
2473 return new String(0, count+resultOffset, result);
2474 }
```

```
2445     if ((lowerChar == Character.ERROR) ||
2446         (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
2447         if (lowerChar == Character.ERROR) {
2448             lowerCharArray =
2449                 ConditionalSpecialCasing.ToLowerCaseCharArray(this, i, locale);
2450         } else if (srcCount == 2) {
2451             resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - srcCount;
2452             continue;
2453         } else {
2454             lowerCharArray = Character.toOthers(lowerChar);
2455         }
2456
2457         /* Grow result if needed */
2458         int mapLen = lowerCharArray.length;
2459         if (mapLen > srcCount) {
2460             char[] result2 = new char[result.length + mapLen - srcCount];
2461             System.arraycopy(result, 0, result2, 0,
2462                             i + resultOffset);
2463             result = result2;
2464         }
2465         for (int x=0; x<mapLen; ++x) {
2466             result[i+resultOffset+x] = lowerCharArray[x];
2467         }
2468         resultOffset += (mapLen - srcCount);
2469     } else {
2470         result[i+resultOffset] = (char)lowerChar;
2471     }
2472 }
2473 return new String(0, count+resultOffset, result);
2474 }
2475 }
```

```
2445         if ((lowerChar == Character.ERROR) ||
2446             (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
2447             if (lowerChar == Character.ERROR) {
2448                 lowerCharArray =
2449                     ConditionalSpecialCasing.ToLowerCaseCharArray(this, i, locale);
2450             } else if (srcCount == 2) {
2451                 resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - srcCount;
2452             continue;
2453         } else {
2454             lowerCharArray = Character.toOthers(lowerChar);
2455         }
2456     }
2457
2458     /* Grow result if needed */
2459     int mapLen = lowerCharArray.length;
2460     if (mapLen > srcCount) {
2461         char[] result2 = new char[result.length + mapLen - srcCount];
2462         System.arraycopy(result, 0, result2, 0,
2463                         i + resultOffset);
2464         result = result2;
2465     }
2466     for (int x=0; x<mapLen; ++x) {
2467         result[i+resultOffset+x] = lowerCharArray[x];
2468     }
2469     resultOffset += (mapLen - srcCount);
2470 } else {
2471     result[i+resultOffset] = (char)lowerChar;
2472 }
2473 return new String(0, count+resultOffset, result);
2474 }
2475 }
```

```
2445     if ((lowerChar == Character.ERROR) ||
2446         (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
2447         if (lowerChar == Character.ERROR) {
2448             lowerCharArray =
2449                 ConditionalSpecialCasing.ToLowerCaseCharArray(this, i, locale);
2450         } else if (srcCount == 2) {
2451             resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - srcCount;
2452             continue;
2453         } else {
2454             lowerCharArray = Character.toOthers(lowerChar);
2455         }
2456
2457         /* Grow result if needed */
2458         int mapLen = lowerCharArray.length;
2459         if (mapLen > srcCount) {
2460             char[] result2 = new char[result.length + mapLen - srcCount];
2461             System.arraycopy(result, 0, result2, 0,
2462                             i + resultOffset);
2463             result = result2;
2464         }
2465         for (int x=0; x<mapLen; ++x) {
2466             result[i+resultOffset+x] = lowerCharArray[x];
2467         }
2468         resultOffset += (mapLen - srcCount);
2469     } else {
2470         result[i+resultOffset] = (char)lowerChar;
2471     }
2472 }
2473 return new String(0, count+resultOffset, result);
2474 }
```

A photograph showing the back of a person's head and shoulders. The person is looking down at a large, open document or blueprint spread out on a table in front of them. The document appears to be a technical drawing or map, with various lines, shapes, and text visible. The person is wearing a dark-colored shirt.

Nós passamos a maior parte do tempo  
**lendo** o código, não escrevendo



Quem já precisou virar a noite por causa de um  
**defeito** que precisava ser corrigido com urgência?

0 contributions in the last year

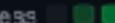
Jul Aug Sep Oct Nov Dec Jan Feb Mar Apr May Jun

Mon

Wed

Fri

Learn how we count contributions

Less  More

## Contribution activity

2023

June 2023

2022

schlemperkaroline has no activity yet for this period.

2021

Show more activity

Seeing something unexpected? Take a look at the [GitHub profile guide](#).

Security

Status

Docs

Contact GitHub

Pricing

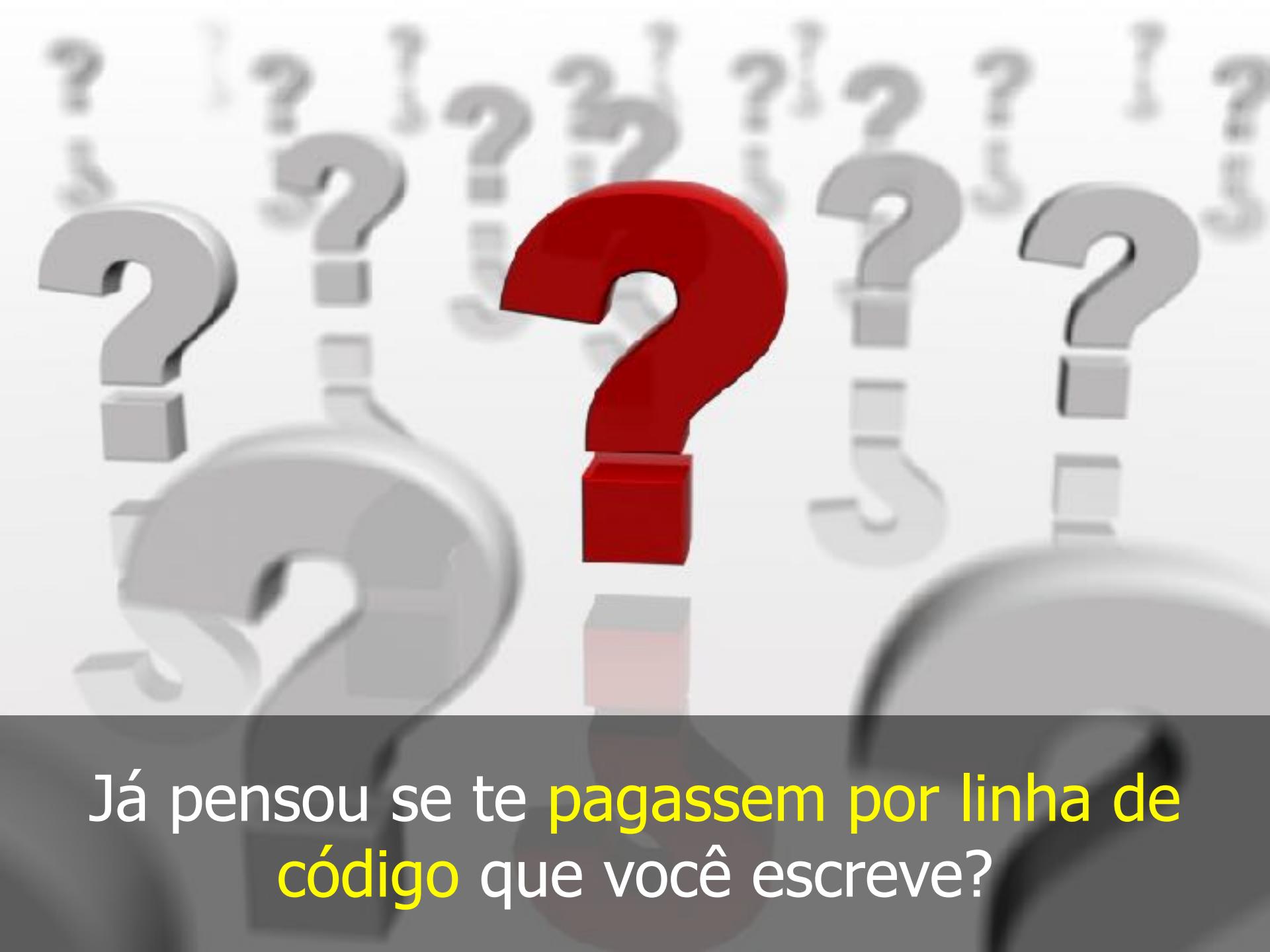
API

Training

Blog

About

Quem já ficou uma semana trabalhando e no final comitou 10 linhas de código?



Já pensou se te pagassem por linha de  
código que você escreve?

## Developers code less than one hour per day

Based on data from 250K+ developers in our global community, **developers code 52 minutes per day** — about 4 hours and 21 minutes during a normal workweek from Monday to Friday.<sup>1</sup>

Code time is defined as time spent actively writing or editing code in an editor or IDE, which we use as an indicator of the amount of focused, uninterrupted time that developers have available to code during the workday.

Based on our estimates, developers spend an additional 41 minutes per day on other types of work in their editors, such as reading code, reviewing pull requests, and browsing documentation.

**Our takeaway:** Our findings suggest that developers frequently face constraints at work that prevent them from finding uninterrupted time to code.





Como você avalia a **produtividade** na sua  
equipe?

Nosso maior desafio é reduzir ou pelo menos manter o esforço de desenvolvimento constante com o passar do tempo



Existe um paradoxo entre dois valores:  
**comportamento e estrutura**

A close-up photograph showing two pairs of hands holding smartphones. The hands are positioned as if the users are interacting with their devices. The background is blurred, focusing on the hands and phones.

O comportamento é o que faz os stakeholders  
ganharem ou economizarem dinheiro



A **estrutura** é o que mantém o comportamento de pé, sem colapsar

Pense na estrutura como: métodos, classes, módulos, serviços e principalmente a relação entre cada uma delas

Quanto mais **comportamento** for adicionado ao software, mais estrutura será necessária para suportá-lo de forma eficaz



Existe uma **disputa** por comportamento de um lado e estrutura por outro



Entre o **comportamento** e **estrutura**, qual  
você prioriza no seu dia a dia?

Se apenas o **comportamento** for priorizado,  
em pouco tempo temos um impacto negativo  
tanto no time quanto na empresa

A close-up photograph showing a person's hands and legs as they work on a car tire. The person is wearing dark pants and a dark jacket. They are using a tool to remove a tire from a wheel. The tire has several holes in its sidewall and the word "TEMPORARY" printed on it. The background is blurred, suggesting an outdoor setting like a driveway or garage.

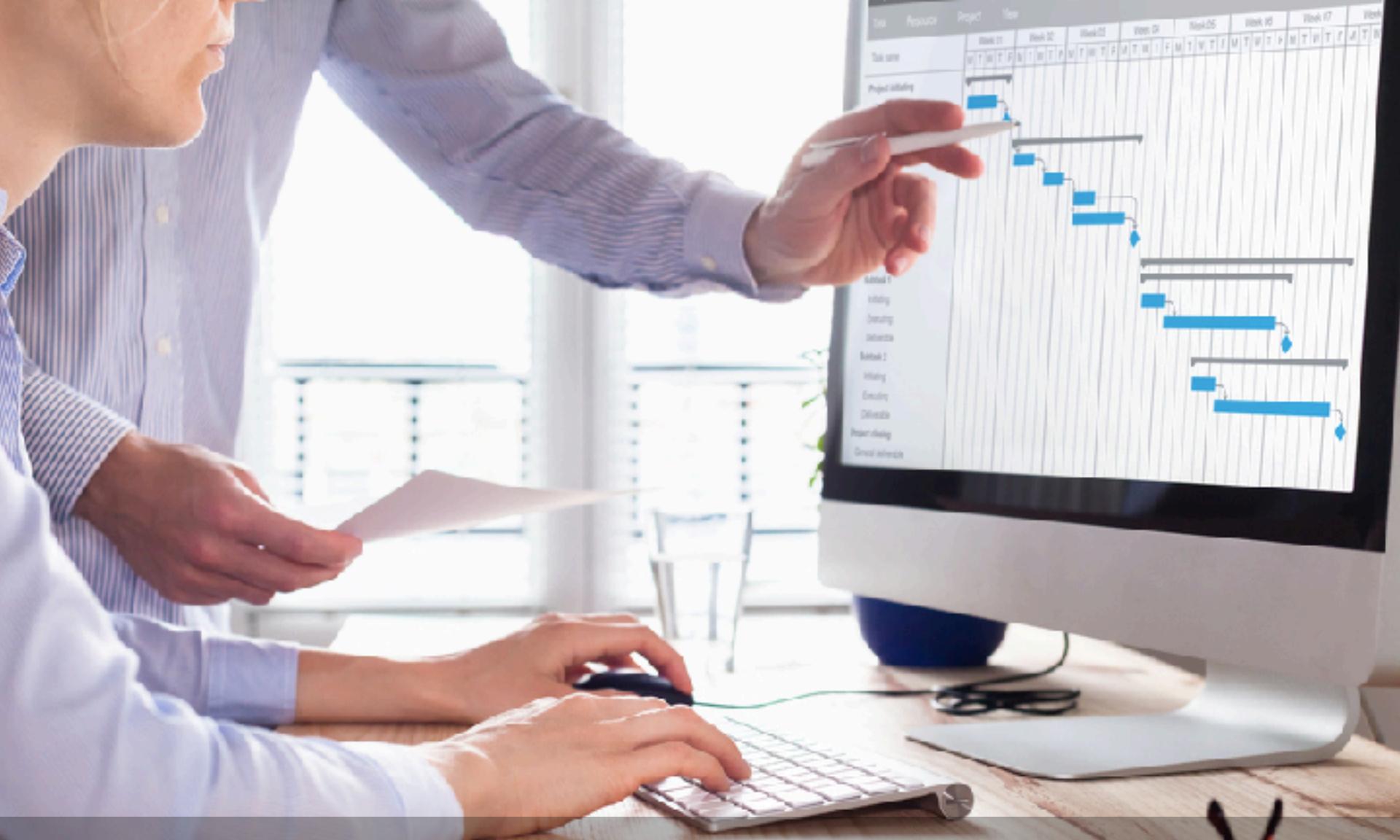
Trabalho chato, braçal e desgastante



Sensação de estar atrasado e improdutivo



Muitos defeitos para corrigir



# Imprevisibilidade nas entregas



**Insatisfação dos clientes**



Evolução lenta do produto

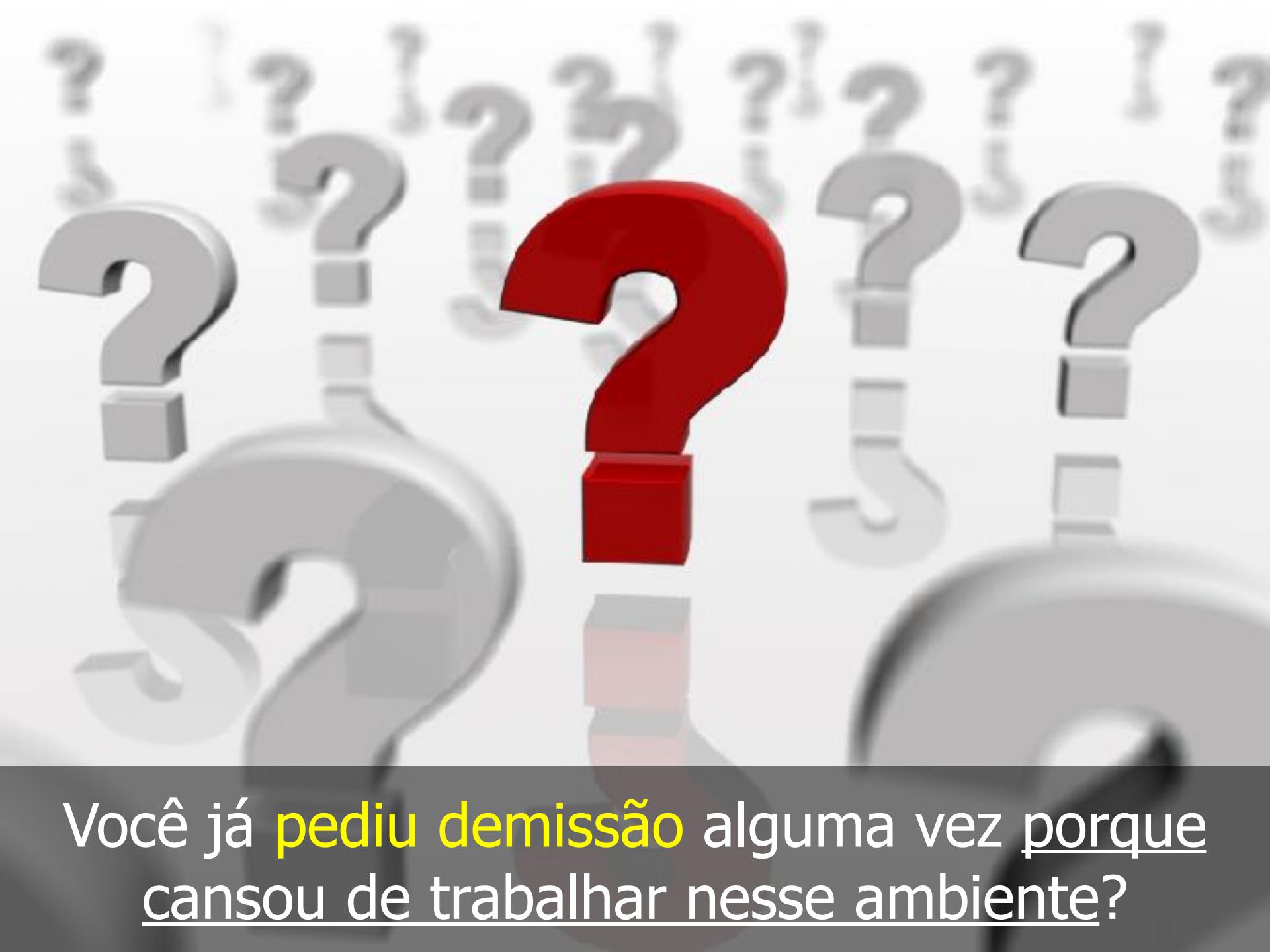


Com a concorrência da vez maior a empresa  
fica **menos competitiva**

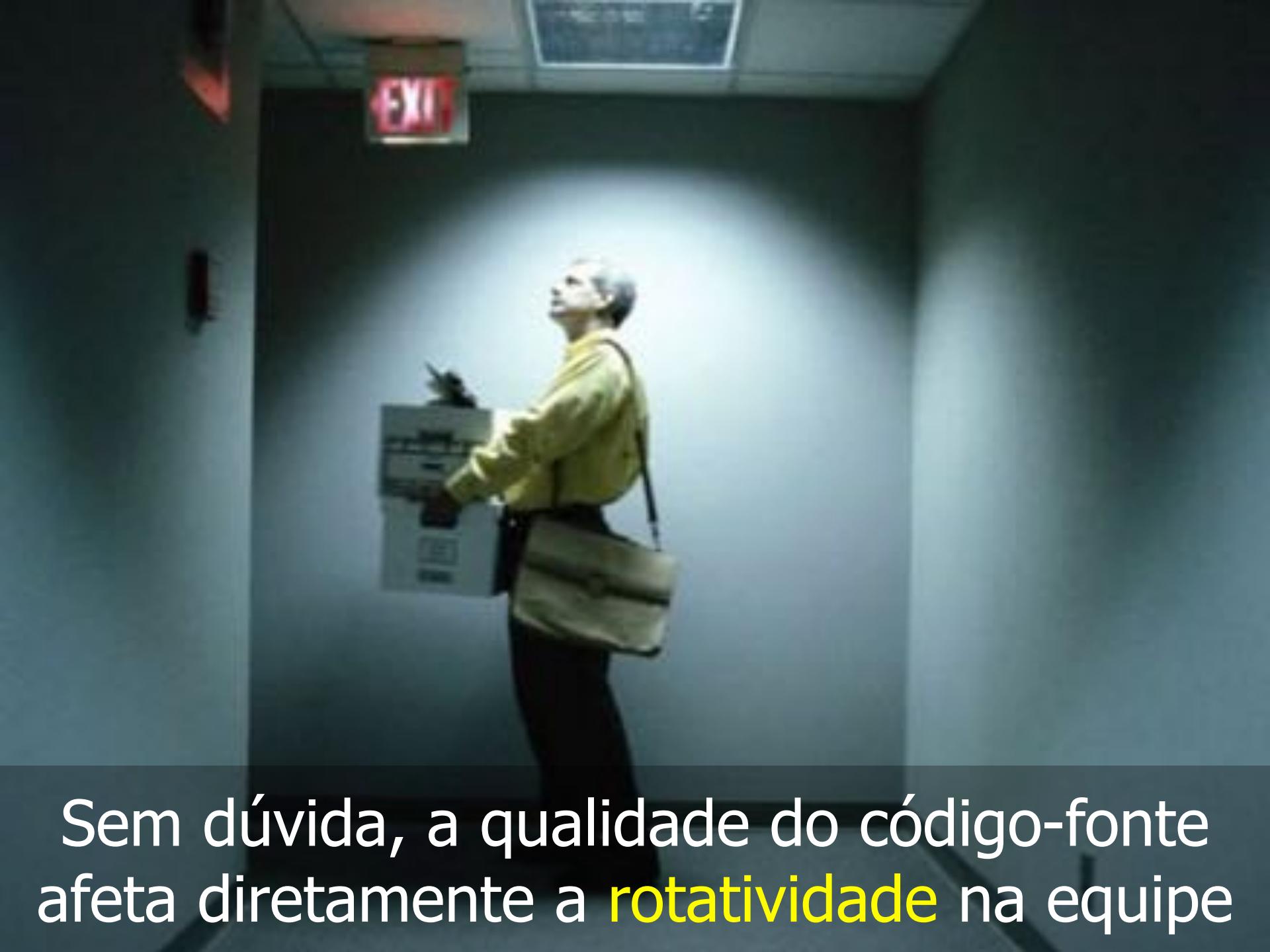


Impacto financeiro na empresa

Existem várias empresas que vão à falência com um software bem feito, mas poucas dão certo e tem sucesso ao longo do tempo com um software mal feito



Você já **pediu demissão** alguma vez **porque**  
cansou de trabalhar nesse ambiente?



Sem dúvida, a qualidade do código-fonte afeta diretamente a **rotatividade** na equipe



O que te **motiva**?

# O que motiva uma pessoa?

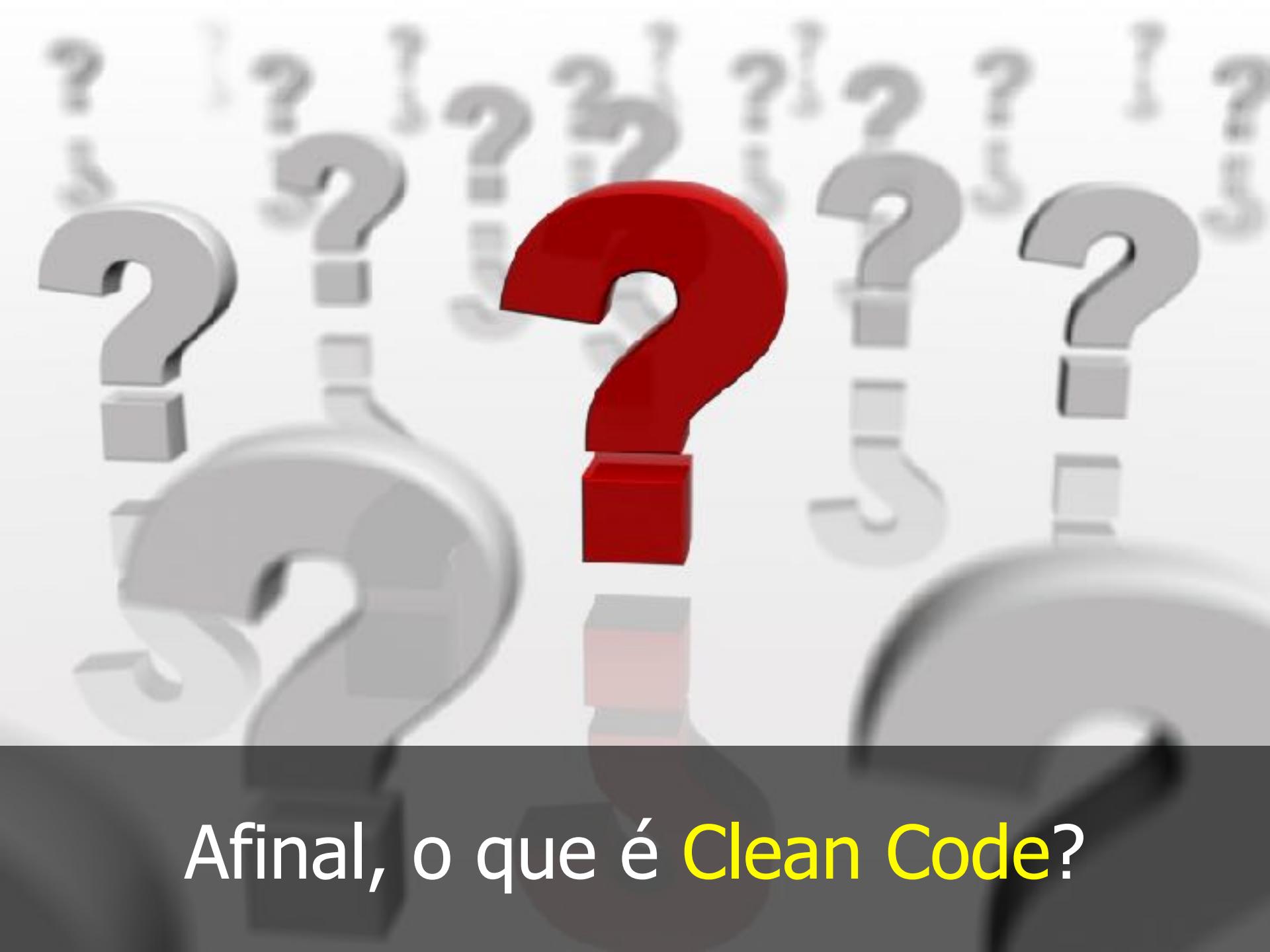
- Dinheiro

# O que motiva uma pessoa?

- Dinheiro
- Ambiente de trabalho

# O que motiva uma pessoa?

- Dinheiro
- Ambiente de trabalho
- Crescimento profissional



Afinal, o que é Clean Code?

"Clean code is simple and direct"

Grady Booch (Criador da UML)

"The logic should be straightforward and make it hard for bugs to hide"

Bjarne Stroustrup (Criador do C++)

"Clean code always looks like it was written  
by someone who cares"

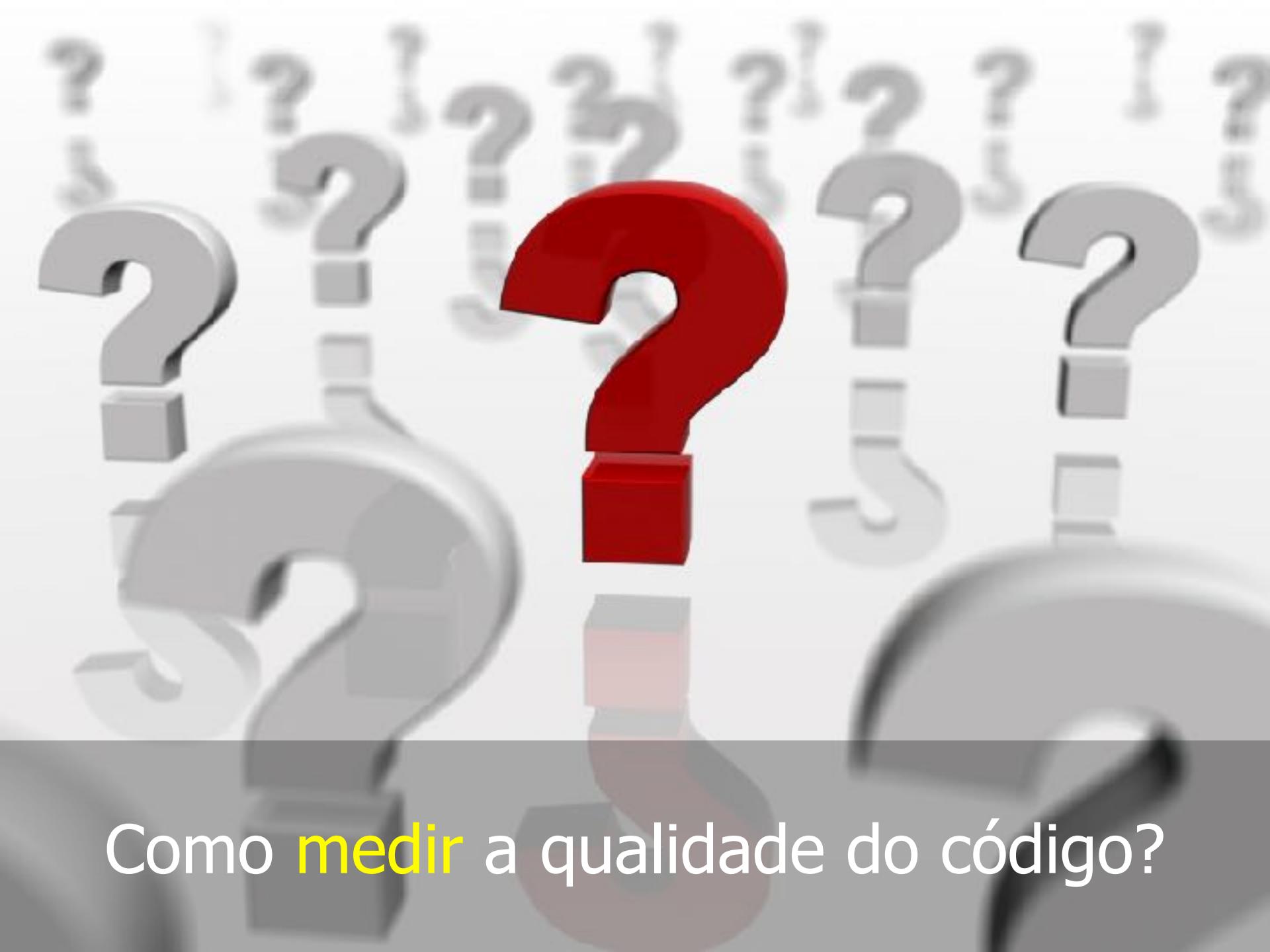
Michael Feathers (Autor do livro Working  
Effectively with Legacy Code)

"You know you are working on clean code  
when each routine you read turns out to be  
pretty much what you expected"

Ward Cunningham (Criador da Wiki)

"Any fool can write code that a computer  
can understand. Good programmers write  
code that humans can understand"

Martin Fowler (Um dos maiores autores sobre  
desenvolvimento de software)



Como **medir** a qualidade do código?

Linhas de código?

Número de métodos?

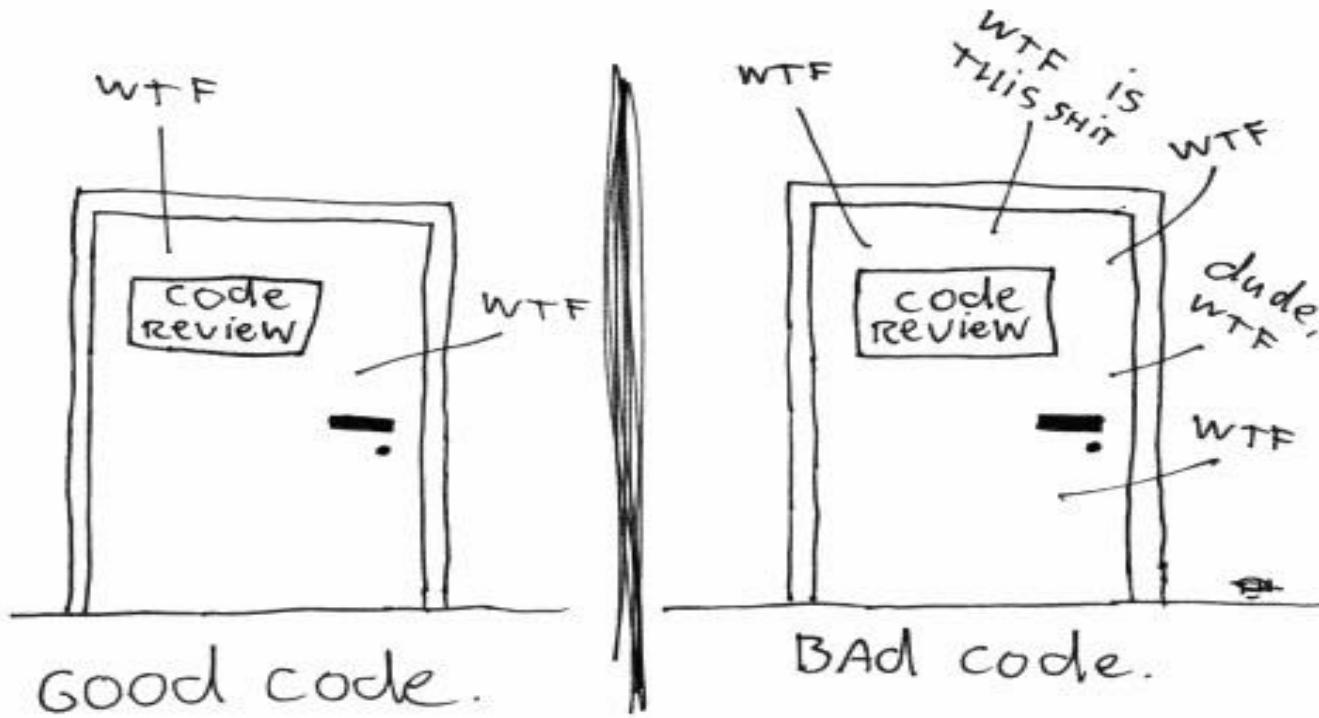
Número de classes?

Tamanho dos método?

Complexidade?

???  
- - -

# The ONLY VALID MEASUREMENT OF Code QUALITY: WTFs/MINUTE



```
2445     if ((lowerChar == Character.ERROR) ||
2446         (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
2447         if (lowerChar == Character.ERROR) {
2448             lowerCharArray =
2449                 ConditionalSpecialCasing.ToLowerCaseCharArray(this, i, locale);
2450         } else if (srcCount == 2) {
2451             resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - srcCount;
2452             continue;
2453         } else {
2454             lowerCharArray = Character.toOthers(lowerChar);
2455         }
2456
2457         /* Grow result if needed */
2458         int mapLen = lowerCharArray.length;
2459         if (mapLen > srcCount) {
2460             char[] result2 = new char[result.length + mapLen - srcCount];
2461             System.arraycopy(result, 0, result2, 0,
2462                             i + resultOffset);
2463             result = result2;
2464         }
2465         for (int x=0; x<mapLen; ++x) {
2466             result[i+resultOffset+x] = lowerCharArray[x];
2467         }
2468         resultOffset += (mapLen - srcCount);
2469     } else {
2470         result[i+resultOffset] = (char)lowerChar;
2471     }
2472 }
2473 return new String(0, count+resultOffset, result);
2474 }
```

WTF?

```
2445 if ((lowerChar == Character.ERROR) ||
2446     (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
2447     if (lowerChar == Character.ERROR) {
2448         lowerCharArray =
2449             ConditionalSpecialCasing.ToLowerCaseCharArray(this, i, locale);
2450     } else if (srcCount == 2) {
2451         resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - srcCount;
2452         continue;
2453     } else {
2454         lowerCharArray = Character.toOthers(lowerChar);
2455     }
2456
2457     /* Grow result if needed */
2458     int mapLen = lowerCharArray.length;
2459     if (mapLen > srcCount) {
2460         char[] result2 = new char[result.length + mapLen - srcCount];
2461         System.arraycopy(result, 0, result2, 0,
2462                         i + resultOffset);
2463         result = result2;
2464     }
2465     for (int x=0; x<mapLen; ++x) {
2466         result[i+resultOffset+x] = lowerCharArray[x];
2467     }
2468     resultOffset += (mapLen - srcCount);
2469 } else {
2470     result[i+resultOffset] = (char)lowerChar;
2471 }
2472 }
2473 return new String(0, count+resultOffset, result);
2474 }
```

```
2445 if ((lowerChar == Character.ERROR) ||
2446     (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
2447     if (lowerChar == Character.ERROR) {
2448         lowerCharArray =
2449             ConditionalSpecialCasing.ToLowerCaseCharArray(this, i, locale);
2450     } else if (srcCount == 2) {
2451         resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - srcCount;
2452         continue;
2453     } else {
2454         lowerCharArray = Character.toOthers(lowerChar);
2455     }
2456
2457     /* Grow result if needed */
2458     int mapLen = lowerCharArray.length;
2459     if (mapLen > srcCount) {
2460         char[] result2 = new char[result.length + mapLen - srcCount];
2461         System.arraycopy(result, 0, result2, 0,
2462                         i + resultOffset);
2463         result = result2;
2464     }
2465     for (int x=0; x<mapLen; ++x) {
2466         result[i+resultOffset+x] = lowerCharArray[x];
2467     }
2468     resultOffset += (mapLen - srcCount);
2469 } else {
2470     result[i+resultOffset] = (char)lowerChar;
2471 }
2472 }
2473 return new String(0, count+resultOffset, result);
2474 }
```

WTF?

WTF?

```
2445 if ((lowerChar == Character.ERROR) ||
2446     (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
2447     if (lowerChar == Character.ERROR) {
2448         lowerCharArray =
2449             ConditionalSpecialCasing.ToLowerCaseCharArray(this, i, locale);
2450     } else if (srcCount == 2) {
2451         resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - srcCount;
2452         continue;
2453     } else {
2454         lowerCharArray = Character.toOthers(lowerChar);
2455     }
2456
2457     /* Grow result if needed */
2458     int mapLen = lowerCharArray.length;
2459     if (mapLen > srcCount) {
2460         char[] result2 = new char[result.length + mapLen - srcCount];
2461         System.arraycopy(result, 0, result2, 0,
2462                         i + resultOffset);
2463         result = result2;
2464     }
2465     for (int x=0; x<mapLen; ++x) {
2466         result[i+resultOffset+x] = lowerCharArray[x];
2467     }
2468     resultOffset += (mapLen - srcCount);
2469 } else {
2470     result[i+resultOffset] = (char)lowerChar;
2471 }
2472 }
2473 return new String(0, count+resultOffset, result);
2474 }
2475 }
```

WTF?

WTF?

WTF?

WTF?







Muitas vezes, existe um ponto de não retorno,  
evite chegar lá, poderá ser muito caro e muito  
arriscado fazer qualquer mudança



Refatore antes que seja tarde demais



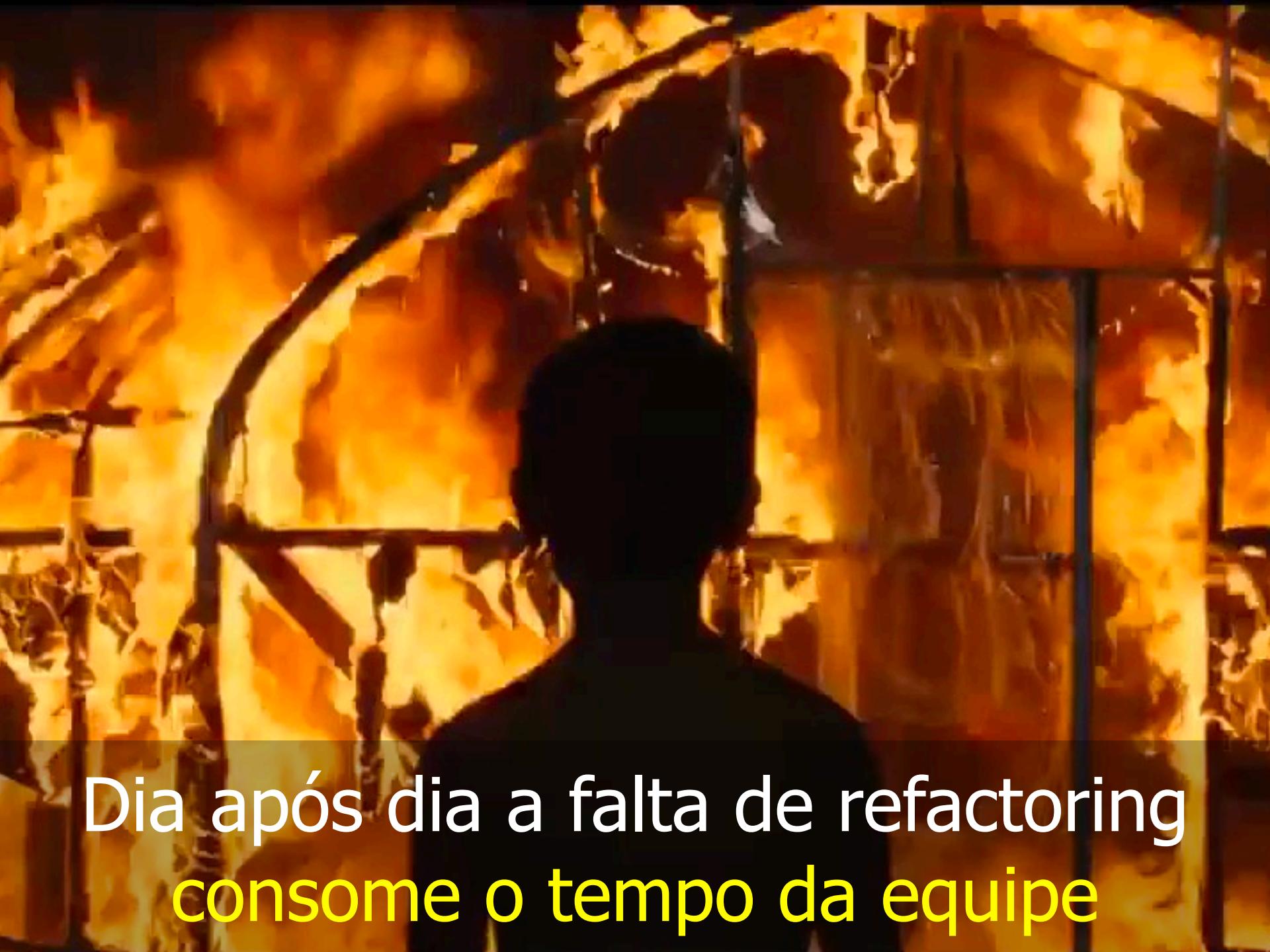
# Refactoring

"Alteração feita na estrutura interna do software para torná-lo mais fácil de ser entendido e menos custoso de ser modificado, sem alterar o seu comportamento observável"

Martin Fowler



Refactoring é um **investimento**, torna o  
software sustentável e competitivo

A photograph of a person from behind, sitting at a desk and working on a computer keyboard. The scene is dimly lit with warm, golden light, creating a dramatic and somewhat somber atmosphere. The person's hands are visible on the keyboard.

Dia após dia a falta de refactoring  
consome o tempo da equipe

A close-up photograph of a yellow caution tape. The word "CAUTION" is printed in large, bold, black capital letters. The tape is slightly curved, and the background shows some green foliage and a bright sky.

CAUTION

Refatore com um propósito, evite  
refatorar apenas por refatorar



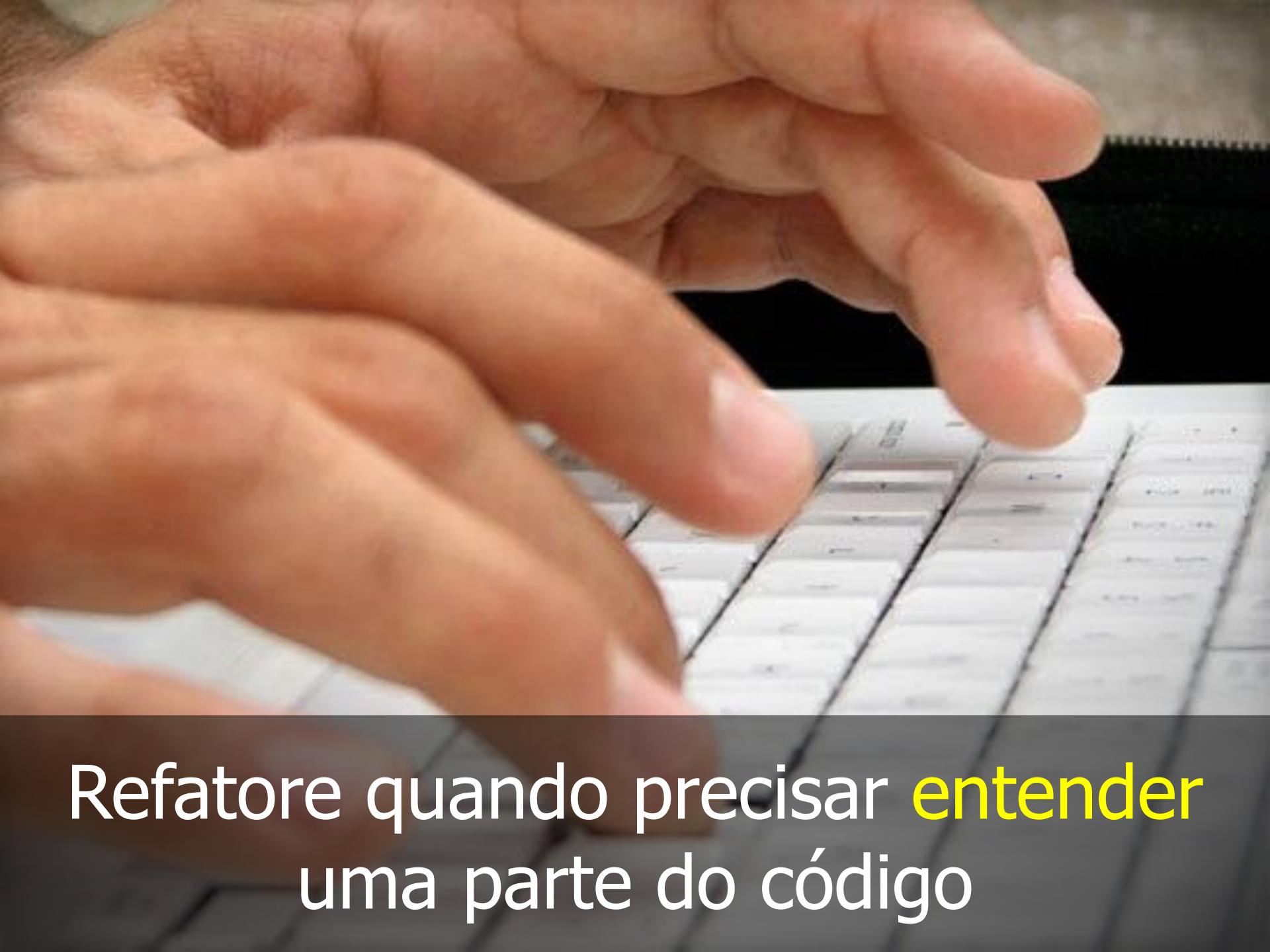
Fique atento as oportunidades



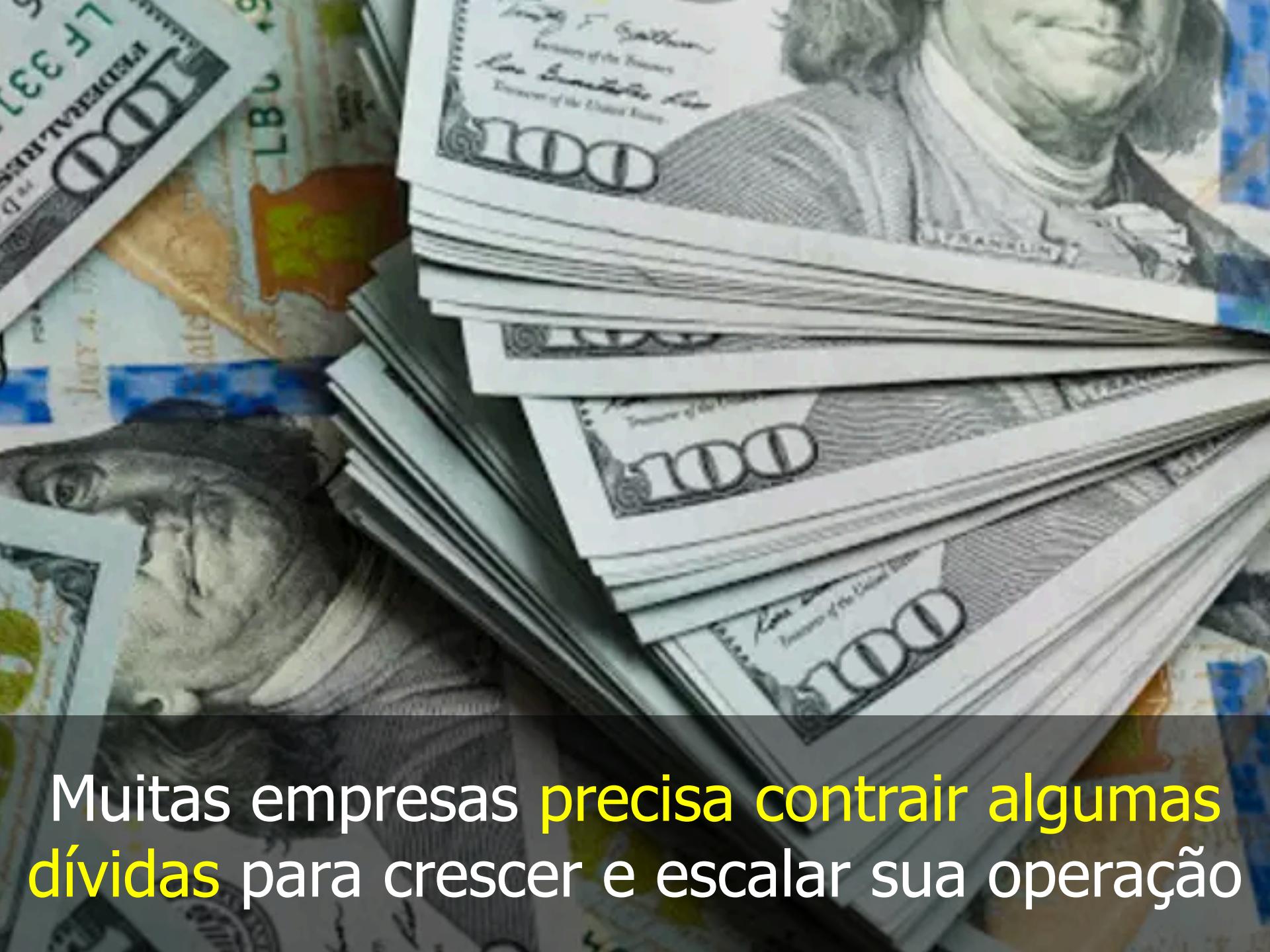
Refatore na hora de **adicionar** novas  
funcionalidades



Refatore quando for **corrigir** um  
defeito



Refatore quando precisar **entender**  
uma parte do código



Muitas empresas precisa contrair algumas dívidas para crescer e escalar sua operação



Mas cuidado com o aumento do **dívida**  
**técnico**, os juros são bem altos



Como **evitar** que o problema aconteça?



Reconhecer e lidar no dia a dia com os  
code smells, tomando ações necessárias

Um smell é um **sintoma** que ocorre dentro do código fonte e que pode ser um indicador de problemas



Mostrar código-fonte do projeto e identificar  
e discutir os code smells

A Rubik's cube is positioned in the center-right of the frame, set against a background of blurred, glowing colors in shades of blue, red, yellow, and purple. The cube itself has standard white, red, blue, orange, green, and yellow faces.

# Test-Driven Development e automação de testes em geral

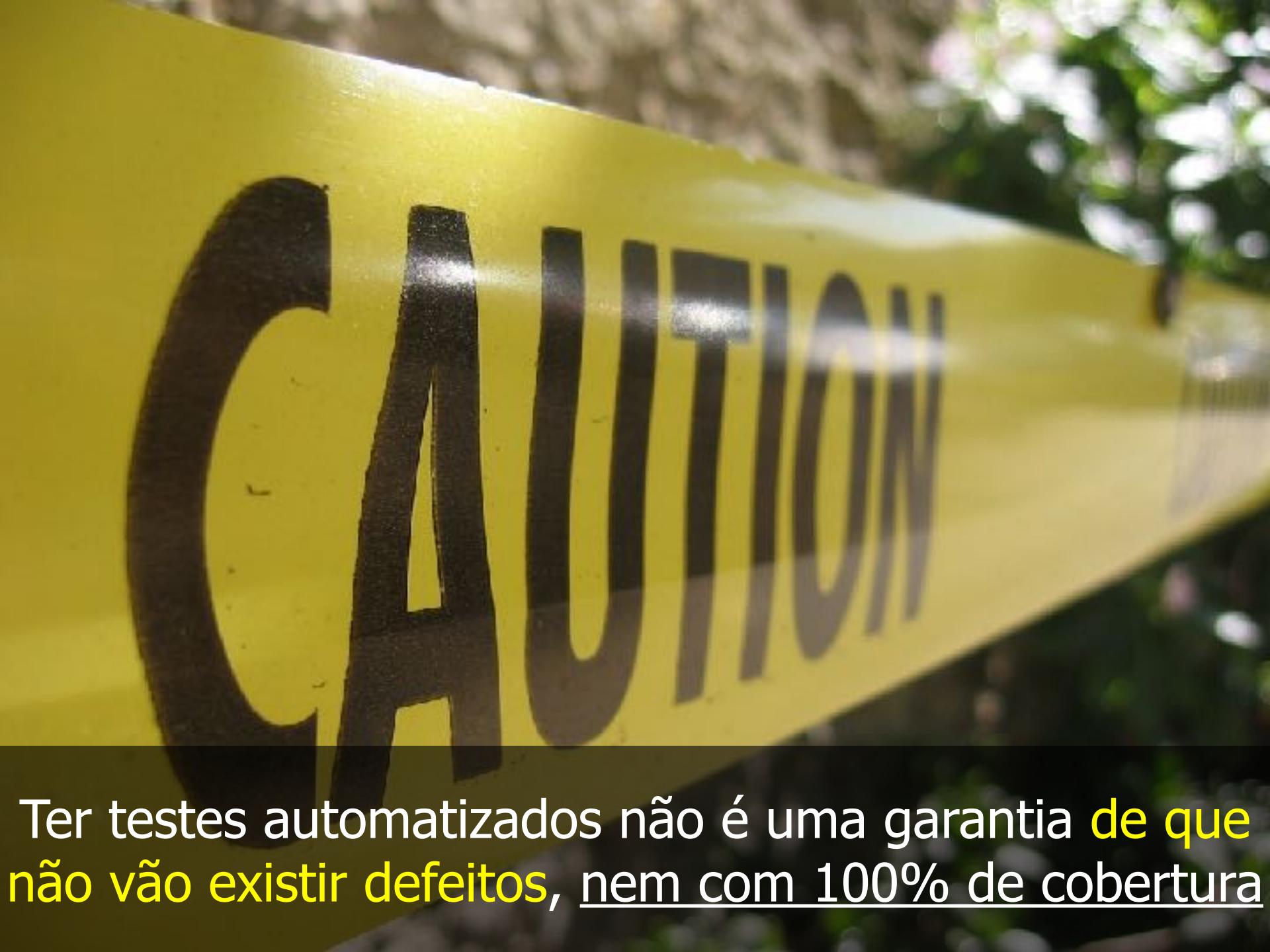


Os testes automatizados são a única forma que temos para garantir que o código funciona e continuará funcionando



Isso quer dizer que não faz sentido ter  
**testes manuais?**

Os testes manuais são importantes, principalmente para a aceitação por parte de usuários-chave, mas devem ser complementares e não garantem que não existirá a regressão ao longo do tempo

A close-up photograph of a yellow caution tape. The word "CAUTION" is printed in large, bold, black capital letters. The tape is positioned diagonally across the frame, with a blurred background of green foliage and trees.

Ter testes automatizados não é uma garantia de que  
não vão existir defeitos, nem com 100% de cobertura



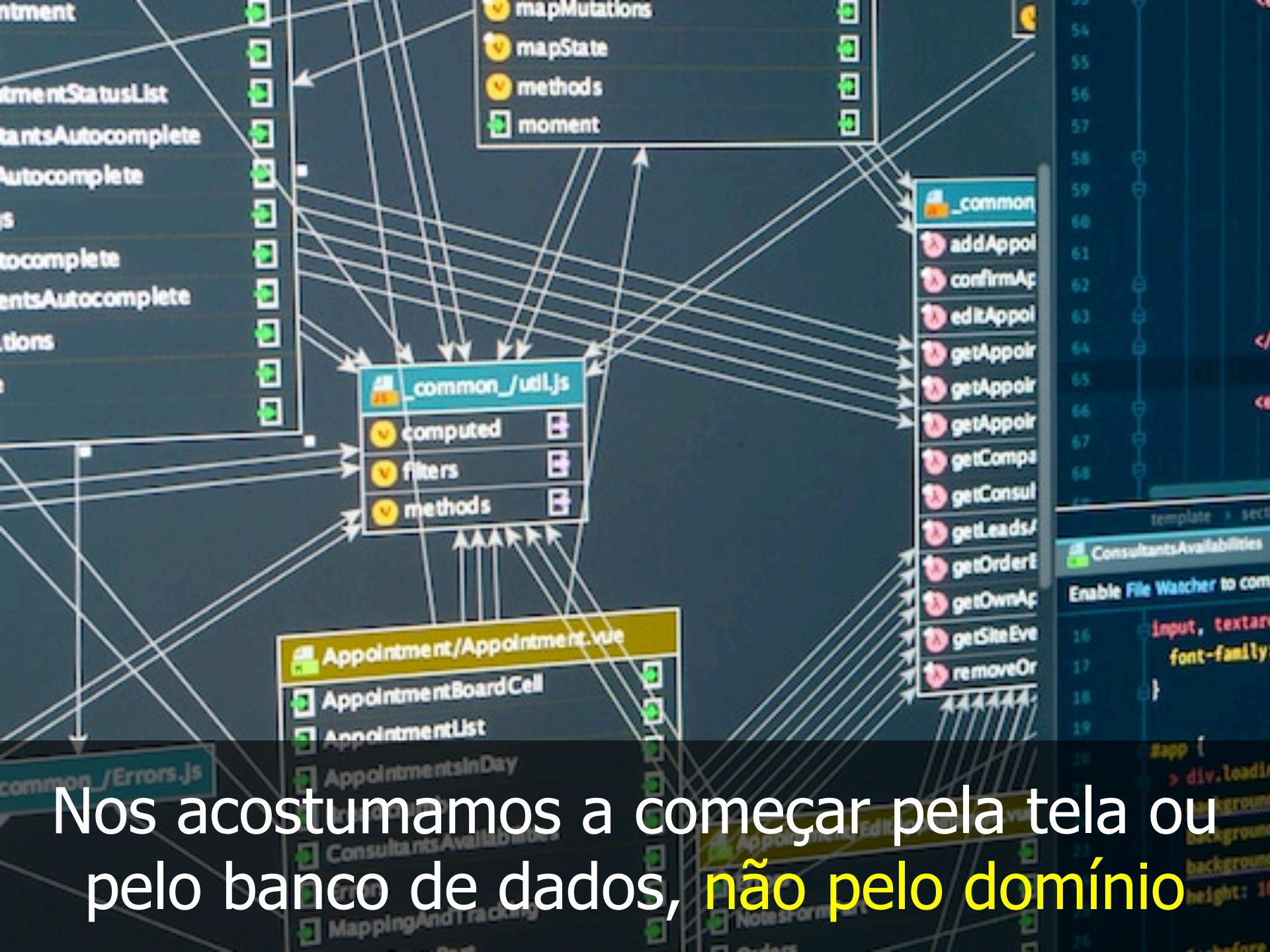
Porque muita gente tenta escrever testes e  
se frustra, **pelo menos no início?**



Escrever os testes requer muita disciplina

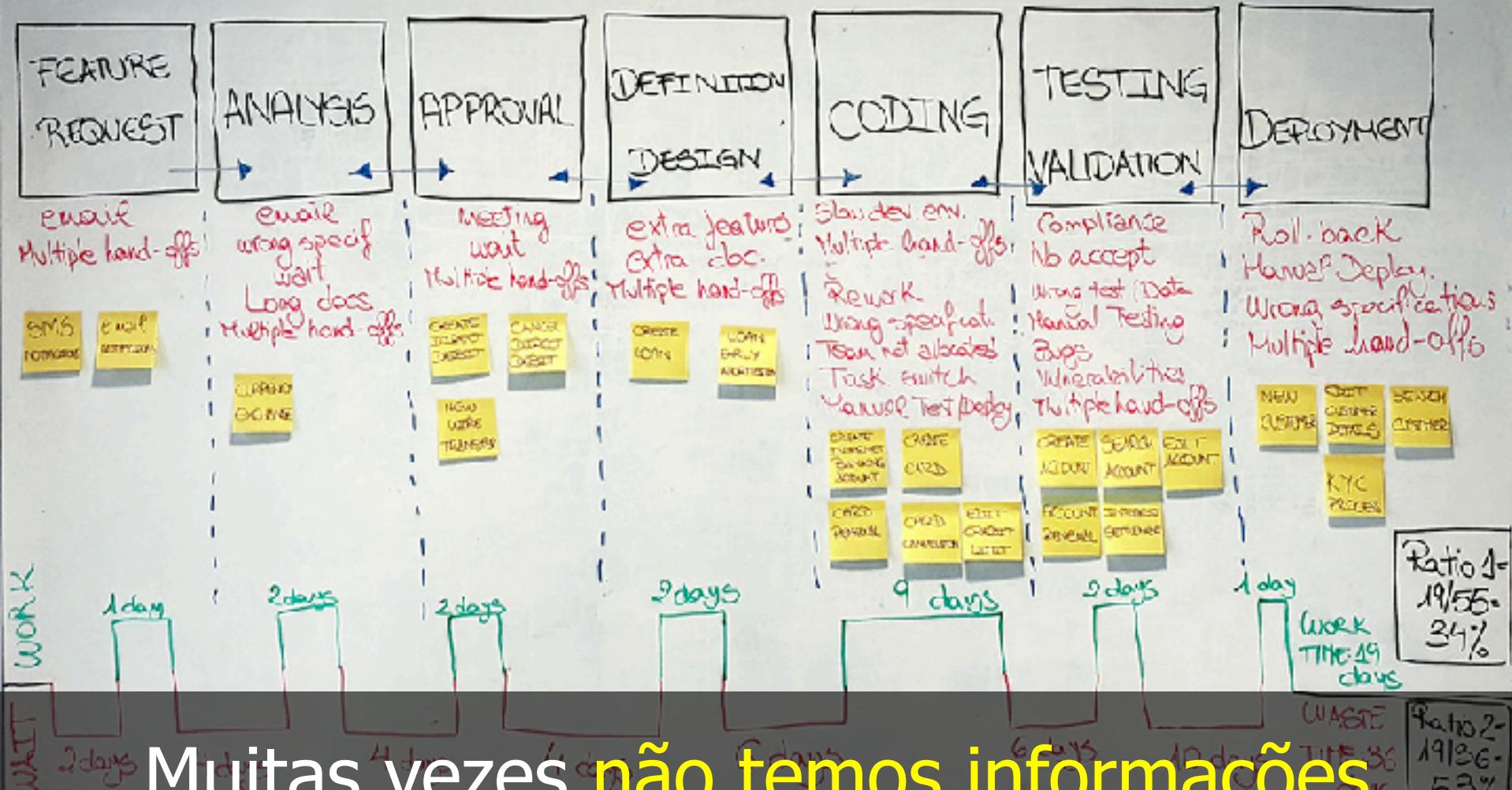


Existe muita **ansiedade**, queremos sempre  
ver tudo "funcionando"



Nos acostumamos a começar pela tela ou pelo banco de dados, **não pelo domínio**

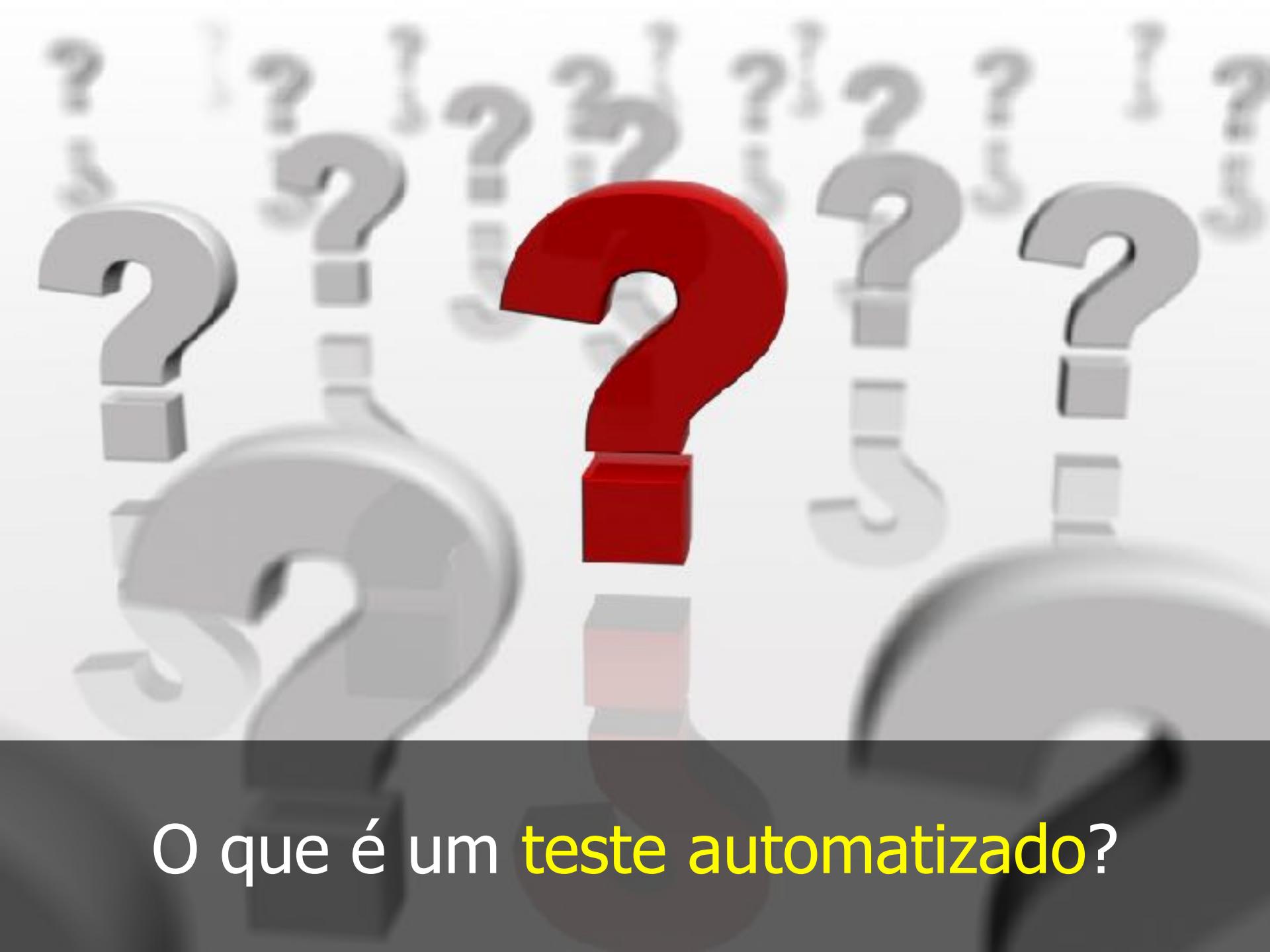
# VSM



Muitas vezes não temos informações suficientes para começar a desenvolver



Muitas vezes o design e a arquitetura não favorecem a **automação dos testes**



O que é um teste automatizado?

Dado um conjunto de entradas, quando algo acontecer a saída deve suprir as expectativas

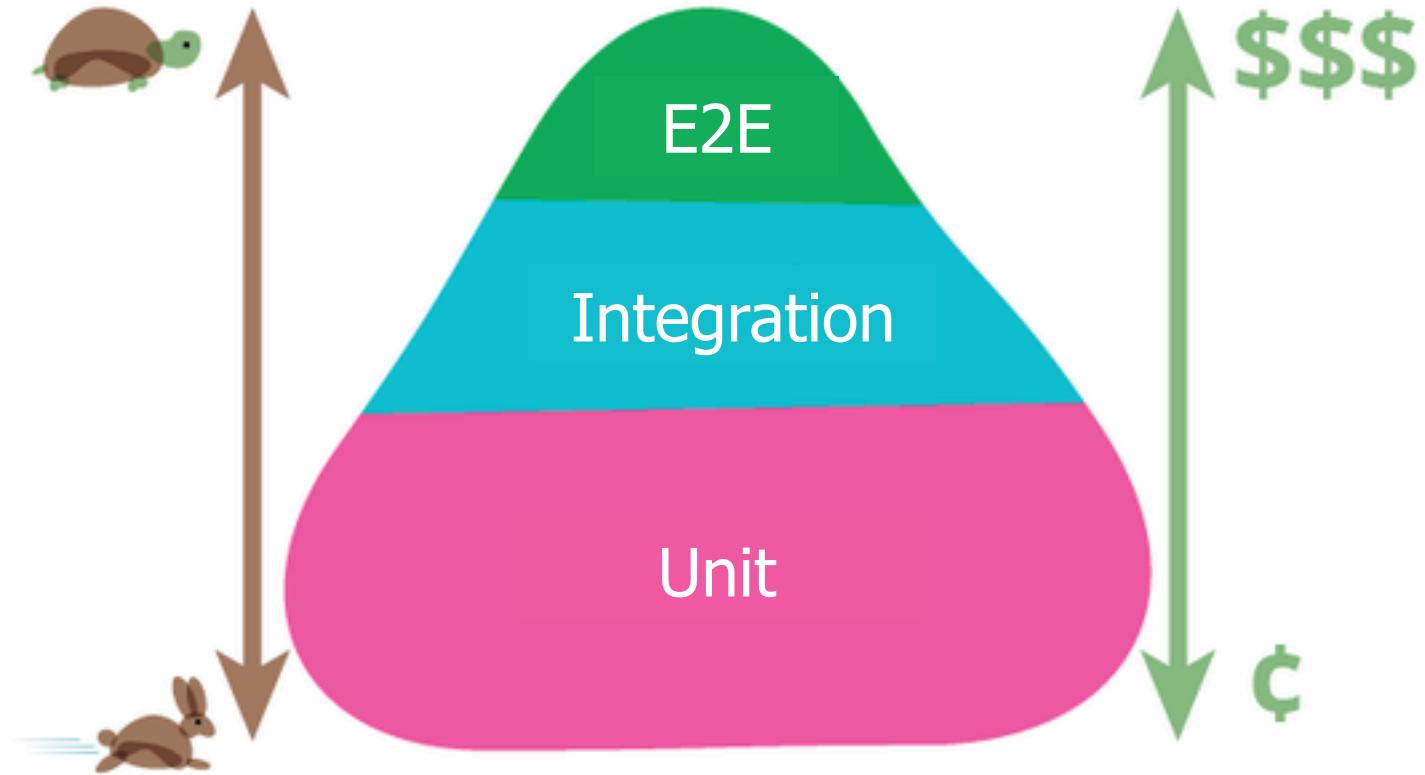
**Given/Arrange:** Definição de todas as informações necessárias para executar o comportamento que será testado

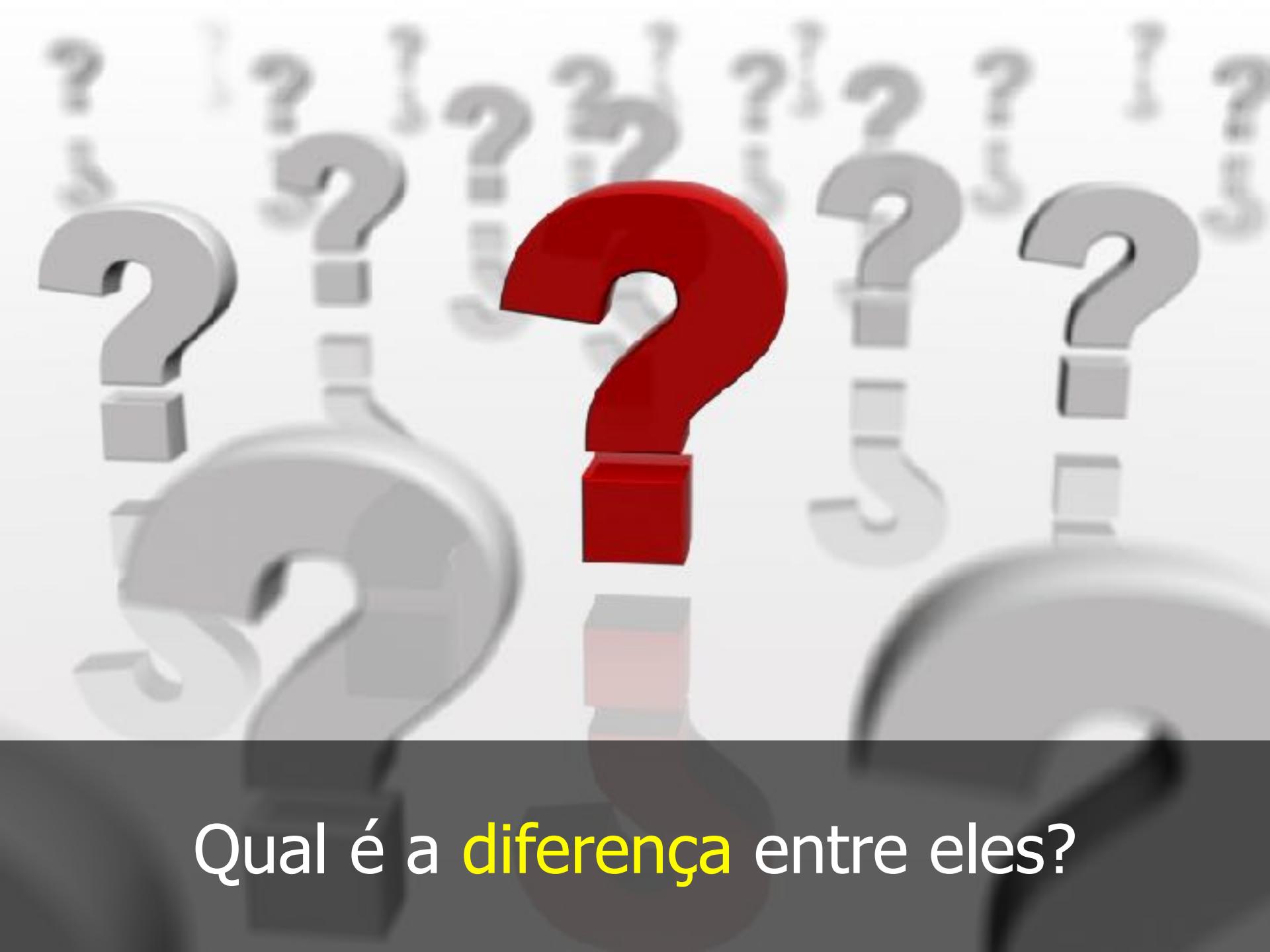
**When/Act:** Executar o comportamento

**Then/Assert:** Verificar o que aconteceu após a execução, comparando as informações retornadas com a expectativa que foi criada



Quais são os tipos de teste automatizado?





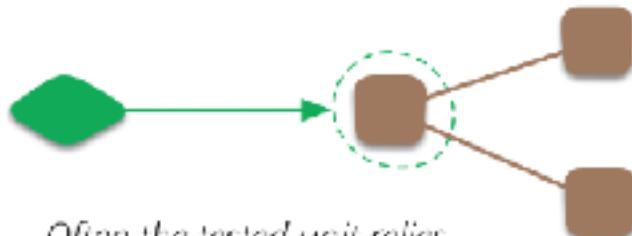
Qual é a diferença entre eles?

## Unit Tests

São testes de unidade, não necessariamente unitários, que podem ou não envolver **vários componentes pertencentes à mesma camada** e sem qualquer interação com recursos externos como um banco de dados, uma API ou o sistema de arquivos

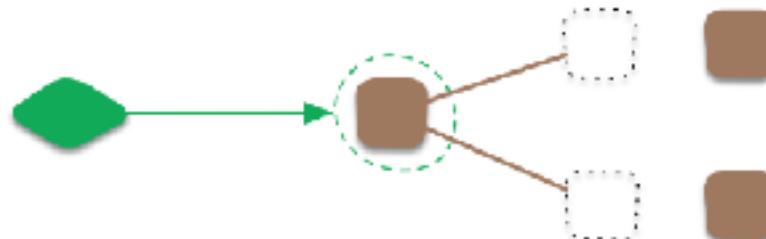
Por não consumirem recursos externos são muito rápidos, sendo executados em poucos milisegundos

### Sociable Tests



*Often the tested unit relies on other units to fulfill its behavior*

### Solitary Tests



*Some unit testers prefer to isolate the tested unit*

But not all unit testers use solitary unit tests. Indeed when xunit testing began in the 90's we made no attempt to go solitary unless communicating with the collaborators was awkward (such as a remote credit card verification system). We didn't find it difficult to track down the actual fault, even if it caused neighboring tests to fail. So we felt allowing our tests to be sociable didn't lead to problems in practice.

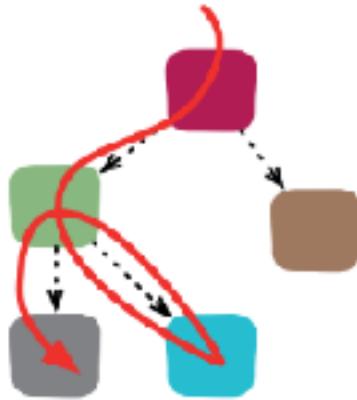
Indeed using sociable unit tests was one of the reasons we were criticized for our use of the term "unit testing". I think that the term "unit testing" is appropriate because these tests are tests of the behavior of a single unit. We write the tests assuming everything other than that unit is working correctly.

## Integration Tests

Testam **componentes pertencentes à múltiplas camadas e normalmente envolvem recursos externos**, sejam eles reais ou não, ou seja, o fato de utilizar um Test Pattern como um stub ou mock não torna o teste de unidade.

Em geral são mais lentos por fazerem I/O

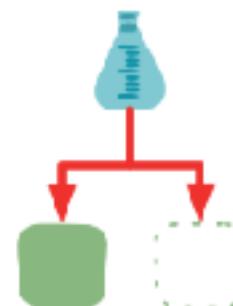
*Integration testing commonly refers to broad tests done with many modules active...*



*...but it can be done with narrow tests of interactions with individual Test Doubles...*



*...supported by Contract Tests to ensure the faithfulness of the double*



#### narrow integration tests

- exercise only that portion of the code in my service that talks to a separate service
- uses test doubles of those services, either in process or remote
- thus consist of many narrowly scoped tests, often no larger in scope than a unit test (and usually run with the same test framework that's used for unit tests)

#### broad integration tests

- require live versions of all services, requiring substantial test environment and network access
- exercise code paths through all services, not just code responsible for interactions



Um teste com as dependências **mockadas**  
se torna um teste de unidade?

## E2E Tests

Replicam o ambiente o usuário final, ou seja, são testes executados de ponta a ponta

Pelas suas características acabam sendo mais lentos e frágeis sendo quebrados facilmente caso a interface com o usuário final seja modificada

A close-up photograph of a yellow caution tape. The word "CAUTION" is printed in large, bold, black capital letters. The tape is positioned diagonally across the frame, with a blurred background of green foliage and trees.

CAUTION

Existem dezenas de tipos de testes, muitos deles são muito similares



Finished after 0.063 seconds

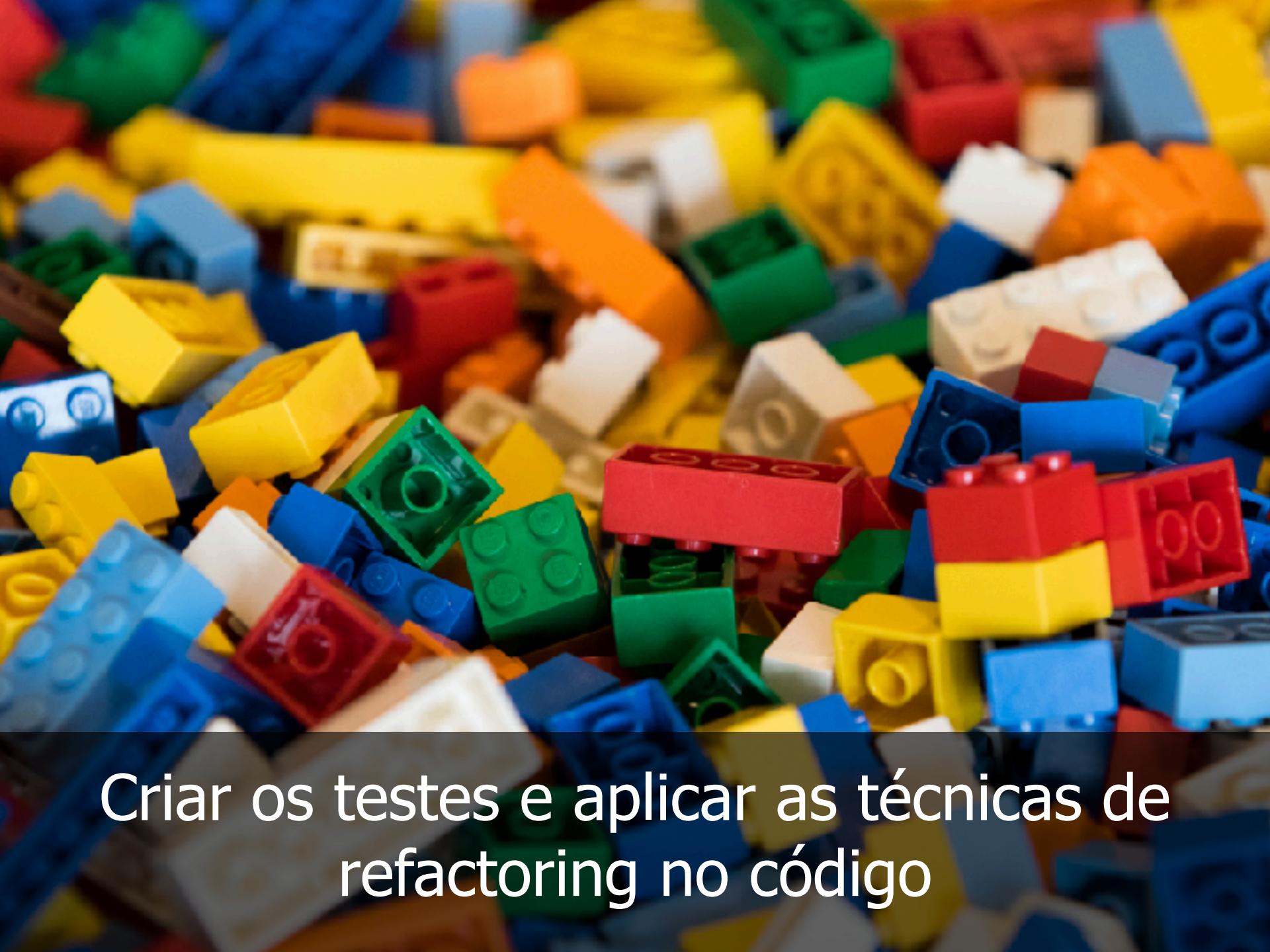
Runs: 7/7

Errors: 0

Failures: 0

- A stack [Runner: JUnit 5] (0.000 s)
  - is instantiated with new Stack() (0.000 s)
  - when new (0.000 s)
    - throws EmptyStackException when peeked (0.000 s)
    - throws EmptyStackException when popped (0.000 s)
    - is empty (0.000 s)
  - after pushing an element (0.000 s)
    - return (0.000 s) Go to File
    - return (0.000 s) ed but remains not empty (0.000 s)
    - return (0.000 s) bed and is empty (0.000 s)

O ideal é ter uma combinação de diferentes tipos de testes, não apenas um



Criar os testes e aplicar as técnicas de refactoring no código

FIRST



# FIRST

- **Fast:** Os testes devem rodar rápido

# FIRST

- **Fast:** Os testes devem rodar rápido
- **Independent:** Não deve existir dependência entre os testes, eles devem poder ser executados de forma isolada

# FIRST

- **Fast:** Os testes devem rodar rápido
- **Independent:** Não deve existir dependência entre os testes, eles devem poder ser executados de forma isolada
- **Repeatable:** O resultado deve ser o mesmo independente da quantidade de vezes que seja executado

# FIRST

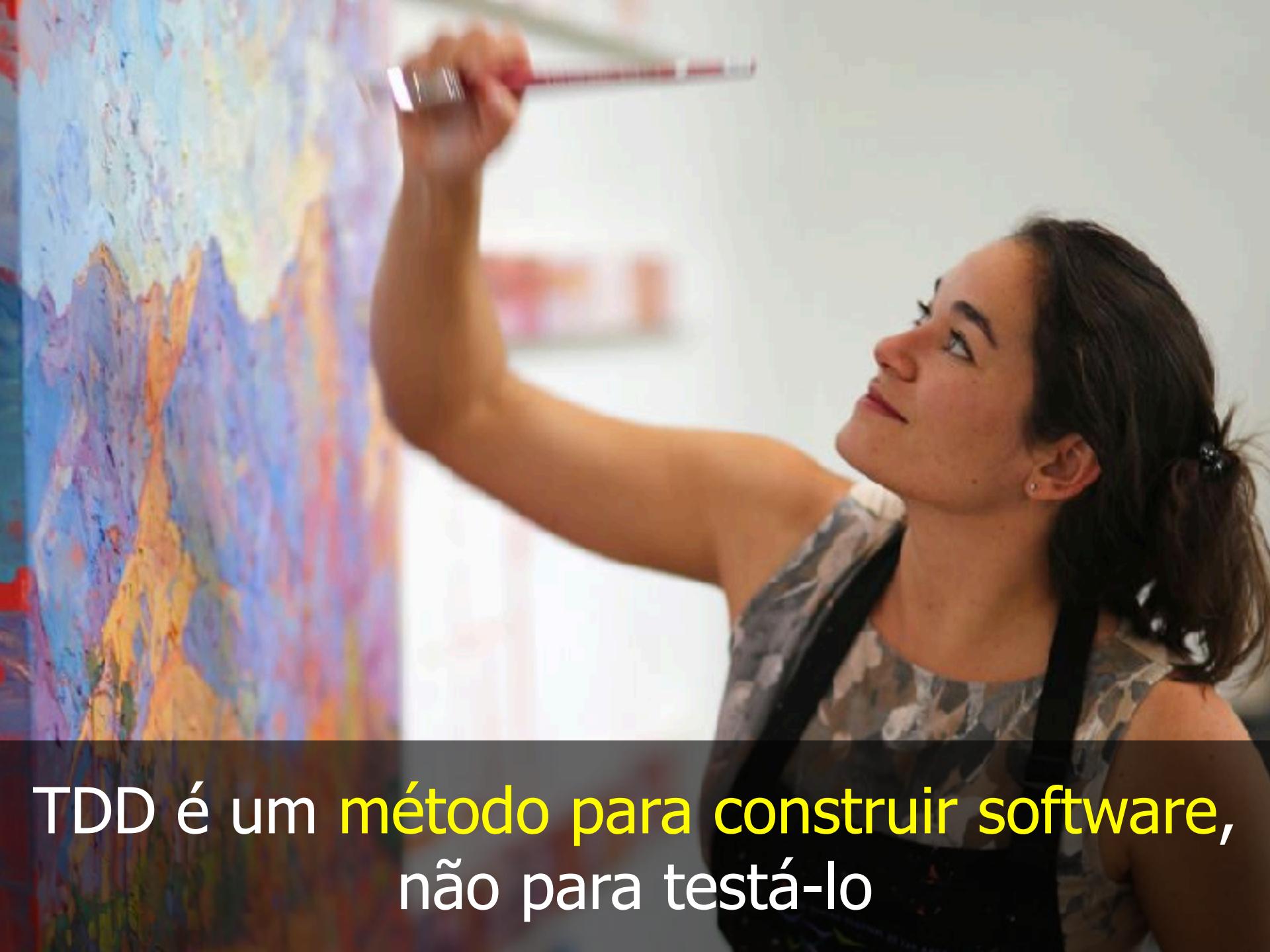
- **Fast:** Os testes devem rodar rápido
- **Independent:** Não deve existir dependência entre os testes, eles devem poder ser executados de forma isolada
- **Repeatable:** O resultado deve ser o mesmo independente da quantidade de vezes que seja executado
- **Self-validating:** O próprio teste deve ter uma saída bem definida que é válida ou não fazendo com que ele passe ou falhe

# FIRST

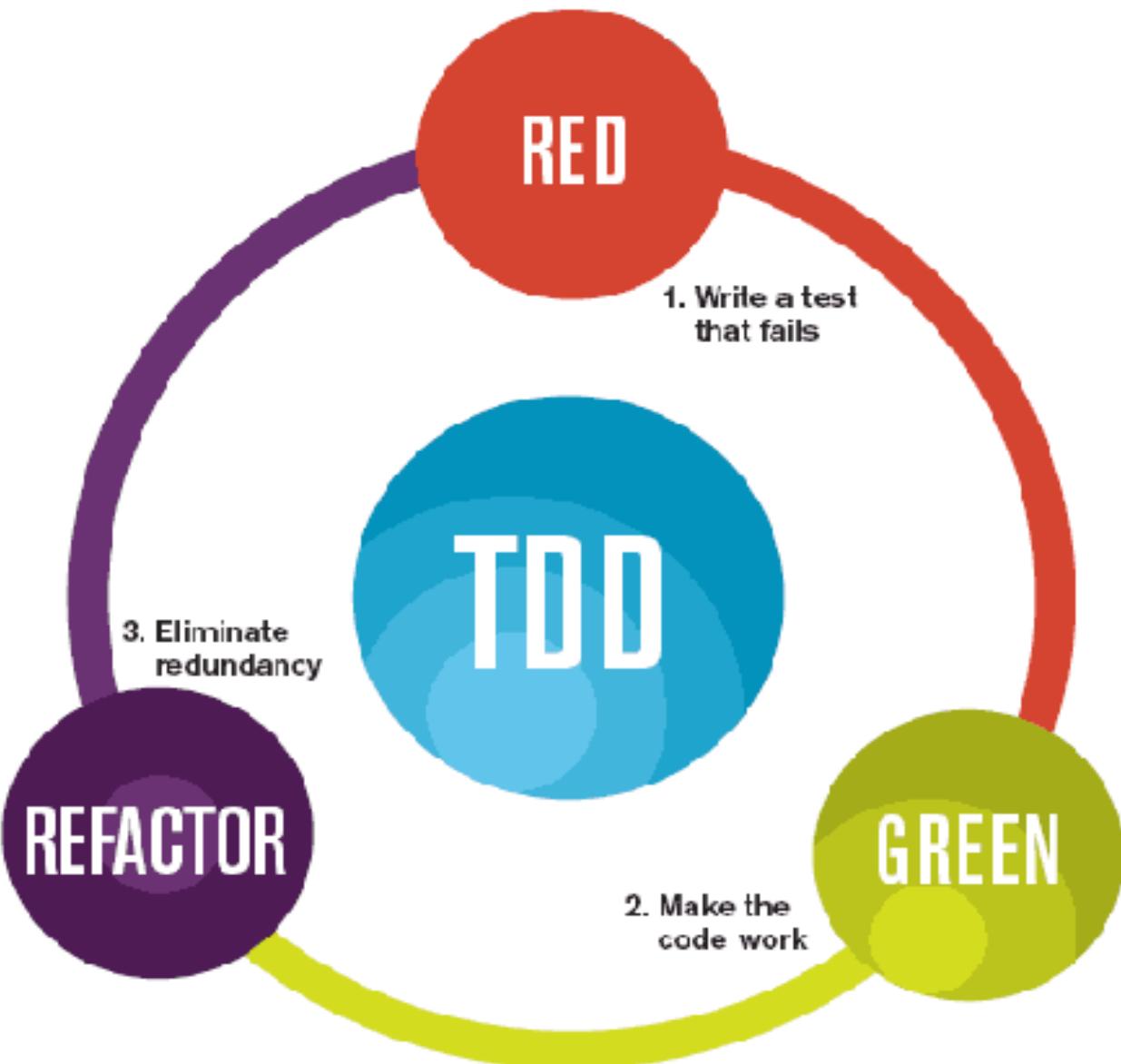
- **Fast:** Os testes devem rodar rápido
- **Independent:** Não deve existir dependência entre os testes, eles devem poder ser executados de forma isolada
- **Repeatable:** O resultado deve ser o mesmo independente da quantidade de vezes que seja executado
- **Self-validating:** O próprio teste deve ter uma saída bem definida que é válida ou não fazendo com que ele passe ou falhe
- **Timely:** Os testes devem ser escritos antes do código-fonte



Como funciona o Test-Driven Development?



TDD é um **método para construir software**,  
não para testá-lo



"TDD is a way of managing fear during programming"

Kent Beck

ArticleS.UncleBob.TheThreeRulesOfTdd X +

Not Secure | butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd

ArticleS.UncleBob

# TheThreeRulesOfTdd [add child]

---

## THE THREE LAWS OF TDD.

Over the years I have come to describe Test Driven Development in terms of three simple rules. They are:

1. You are not allowed to write any production code unless it is to make a failing unit test pass.
2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

You must begin by writing a unit test for the functionality that you intend to write. But by rule 2, you can't write very much of that unit test. As soon as the unit test code fails to compile, or fails an assertion, you must stop and write production code. But by rule 3 you can only write the production code that makes the test compile or pass, and no more.

If you think about this you will realize that you simply cannot write very much code at all without compiling and executing something. Indeed, this is really the point. In everything we do, whether writing tests, writing production code, or refactoring, we keep the system executing at all times. The time between running tests is on the order of seconds, or minutes. Even 10 minutes is too long.

Too see this in operation, take a look at [The Bowling Game Kata](#).

Now most programmers, when they first hear about this technique, think: "This is stupid! It's going to slow me down, it's a waste of time and effort, it will keep me from thinking, it will just break my flow." However, think about what would happen if you walked in a room full of people working this way. Pick any random person at any random time. A minute ago, all their code worked.

Let me repeat that: **A minute ago all their code worked!** And it doesn't matter who you pick, and it doesn't matter when you pick. **A minute ago all their code worked!**

If all your code works every minute, how often will you use a debugger? Answer, not very often. It's easier to simply hit F5 a bunch of times to get the code back to a working state, and then try to write the last minutes worth again. And if you aren't debugging very much, how much time will you be saving? How much time do you spend debugging now? How much time do you spend fixing bugs once you've debugged them? What if you could decrease that time by a significant fraction?

But the benefit goes far beyond that. If you work this way, then every hour you are producing several tests. Every day dozens of tests. Every month hundreds of tests. Over the course of a year you will write thousands of tests. You can keep all these tests and run them any time you like. When would you run them? All the time! Any time you made any kind of change at all.

Why don't we clean up code that we know is messy? We're afraid we'll break it. But if we have the tests, we can be reasonably sure that the code is not broken, or that we'll detect the breakage immediately. If we have the tests we become fearless about making changes. If we see messy code, or an unclean structure, we can clean it without fear. Because of the tests, the code becomes malleable again. Because of the tests, software becomes soft again.

But the benefits go beyond that. If you want to know how to call a certain API, there is a test that does it. If you want to know how to create a certain object, there is a test that does it. Anything you want to know about the existing system, there is a test that demonstrates it. The tests are like little design documents, little coding examples, that describe how the system works and how to use it.

Have you ever integrated a third party library into your project? You got a big manual full of nice documentation. At the end there was a thin appendix of examples. Which of the two did you read? The examples of course! That's what the unit tests are! They are the most useful part of the documentation. They are the living examples of how to use the code. They are design documents that are hideously detailed, utterly unambiguous, so formal that they execute, and they cannot get out of sync with the production code.

But the benefits go beyond that. If you have ever tried to add unit tests to a system that was already working, you probably found that it wasn't much fun. You likely found that you either had to change portions of the design of the system, or cheat on the tests, because the system you were trying to write tests for was not designed to be testable. For example, you'd like to test some function f. However, f calls another function that deletes a record from the database. In your test, you don't want the record deleted, but you don't have any way to stop it. The system wasn't

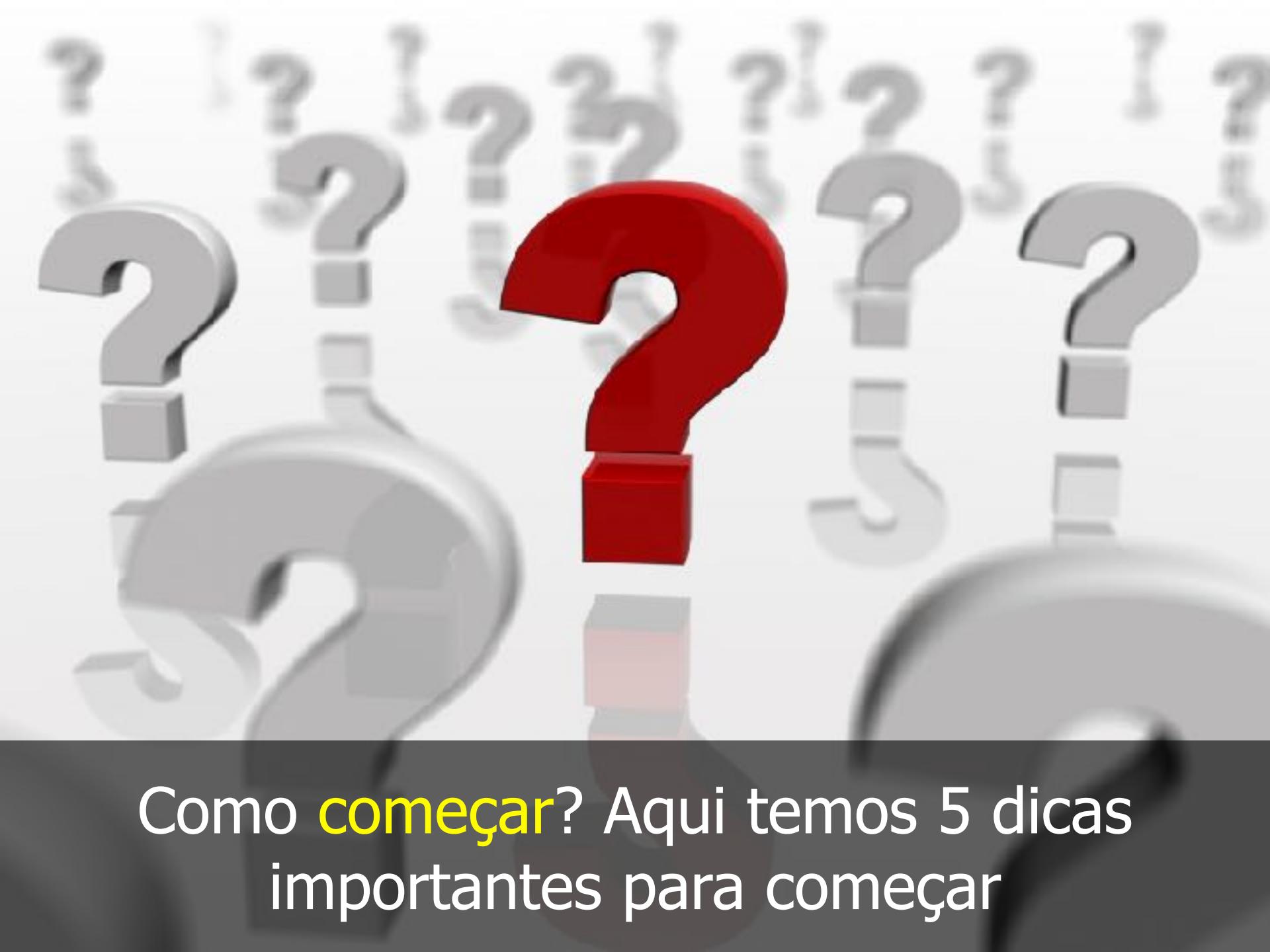
## Three Laws of TDD

Você não pode escrever nenhum código até ter escrito um teste que detecte uma possível falha.

Você não pode escrever mais testes de unidade do que o suficiente para detectar a falha.

Você não pode escrever mais código do que o suficiente para passar nos testes.

(Robert C. Martin)



Como **começar**? Aqui temos 5 dicas  
importantes para começar

Dê o primeiro passo, ou seja, crie um exemplo que as pessoas na equipe possam seguir e utilizar como base, ninguém começa com 100% de cobertura

Foque no que tem mais risco e muda com mais frequência, o sucesso não está em ter 100% de cobertura mas sim em automatizar os testes daquilo que dá mais retorno

Usar test patterns como um stub ou mock  
não é necessariamente ruim, em muitos  
casos é algo necessário, mas só conseguir  
testar se utilizar esses recursos pode indicar  
que o design precisa melhorar

Usar TDD pode parecer **muito complicada no**  
**início, mas certamente faz sentido definir o**  
**cenário desejado antes e escrever o código**

Por fim, use o tipo de teste que for mais viável para a sua realidade, geralmente são os testes de integração ou E2E