SOLID - Single Responsibility Principle (SRP)

Foi definido por Tom DeMarco e Meilir Page-Jones, eles o chamavam de coesão, ou seja, o nível de relacionamento entre elementos de um módulo, entendendo como módulo uma classe, um componente, um arquivo.

Na visão de Robert Martin, devemos ir além do relacionamento, separando as coisas que mudam por motivos diferentes e nesse contexto a palavra responsabilidade significa motivo para mudar.

If a class has more then one responsibility, then the responsibilities become coupled and changes to one responsibility may affect others.

A class should have only one reason to change.

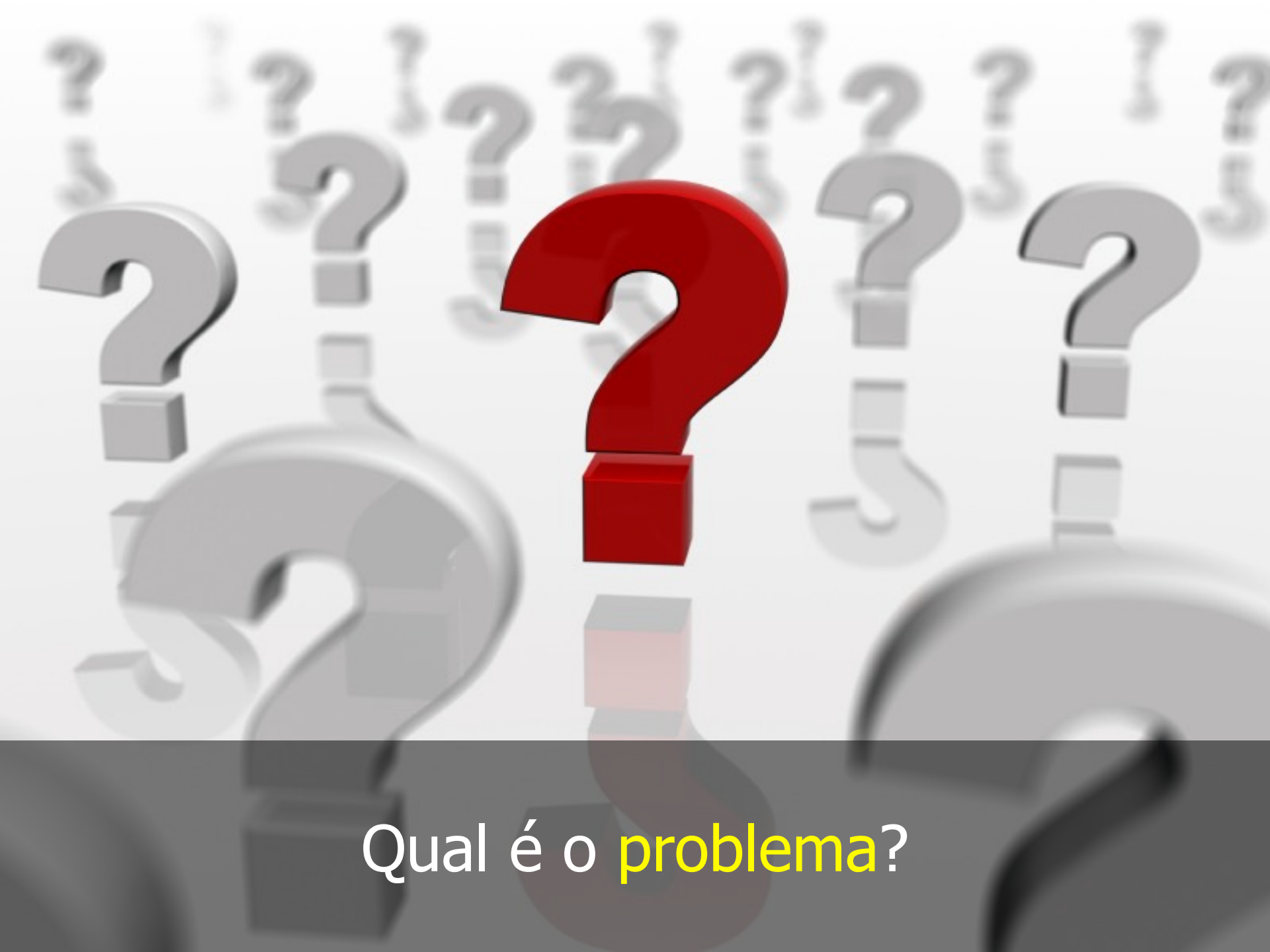
Robert C. Martin


Arquitetura Hexagonal
(Ports and Adapters)

user.php — cccat13_2

user.php U

```php
<?php
    import_request_variables("pg", "form_");
    $db = mysql_connect("localhost:/export/mysql/mysql.sock");
    mysql_select_db("app", $db);
    $rows = mysql_query("select * from transaction where hash = '$form_hash'", $db);
    if(mysql_num_rows($rows) == 0) {
        $error = "Transaction not found";
    } else {
        $transaction_hash = mysql_result($rows, 0, 0);
        $transaction_amount = mysql_result($rows, 0, 1);
        $transaction_due_date = mysql_result($rows, 0, 2);
        $today = time();
        $diff_in_days = ($today - $transaction_due_date) / (60 * 60 * 24);
        $transaction_penalty = ($transaction_amount * 2) / 100;
        $transaction_interest = (($transaction * 0.033) / 100) * $diff_in_days;
        $transaction_total_amount = $transaction_amount + $transaction_penalty + $transaction_interest;
    }
?>
<html>
    <head>
        <title>Transaction</title>
    </head>
    <body>
        <?php
            if($error) {
                echo "<h1>Error accessing transaction</h1>\n";
                echo "<p>$error</p>\n";
            } else {
                echo "<h1>Information about $form_hash</h1>\n";
                echo "<p>amount: $transaction_amount</p>\n";
                echo "<p>due date: $transaction_due_date</p>\n";
                echo "<p>penalty: $transaction_penalty</p>\n";
                echo "<p>interest: $transaction_interest</p>\n";
                echo "<p>total: $transaction_total</p>\n";
```


Qual é o problema?

user.php — cccat13_2

user.php U

```php
<?php
    import_request_variables("pg", "form_");
    $db = mysql_connect("localhost:/export/mysql/mysql.sock");
    mysql_select_db("app", $db);
    $rows = mysql_query("select * from transaction where hash = '$form_hash'", $db);
    if(mysql_num_rows($rows) == 0) {
        $error = "Transaction not found";
    } else {
        $transaction_hash = mysql_result($rows, 0, 0);
        $transaction_amount = mysql_result($rows, 0, 1);
        $transaction_due_date = mysql_result($rows, 0, 2);
        $today = time();
        $diff_in_days = ($today - $transaction_due_date) / (60 * 60 * 24);
        $transaction_penalty = ($transaction_amount * 2) / 100;
        $transaction_interest = (($transaction * 0.033) / 100) * $diff_in_days;
        $transaction_total_amount = $transaction_amount + $transaction_penalty + $transaction_interest;
    }
?>
<html>
    <head>
        <title>Transaction</title>
    </head>
    <body>
        <?php
            if($error) {
                echo "<h1>Error accessing transaction</h1>\n";
                echo "<p>$error</p>\n";
            } else {
                echo "<h1>Information about $form_hash</h1>\n";
                echo "<p>amount: $transaction_amount</p>\n";
                echo "<p>due date: $transaction_due_date</p>\n";
                echo "<p>penalty: $transaction_penalty</p>\n";
                echo "<p>interest: $transaction_interest</p>\n";
                echo "<p>total: $transaction_total</p>\n";
```

user.php — cccat13_2

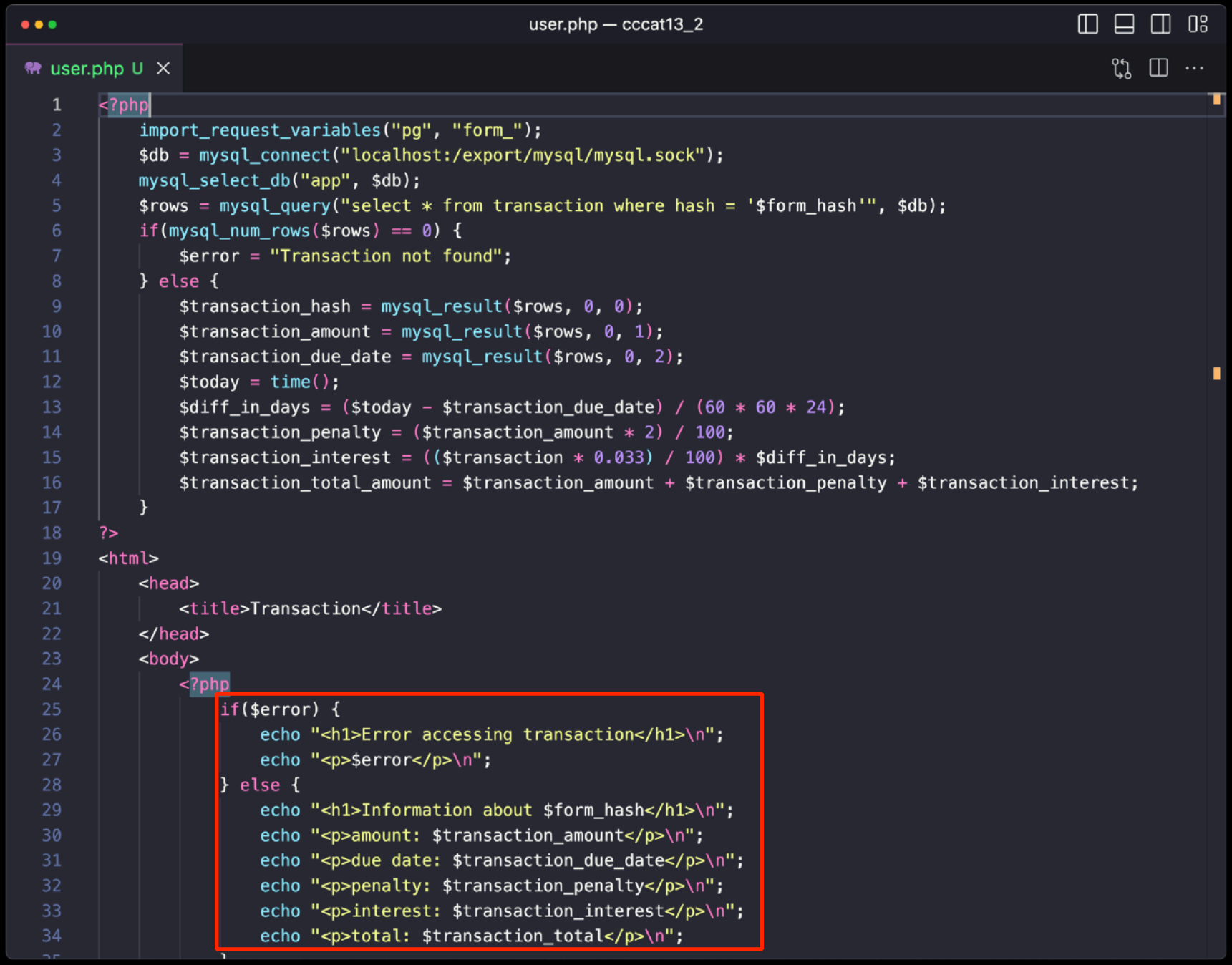
user.php U

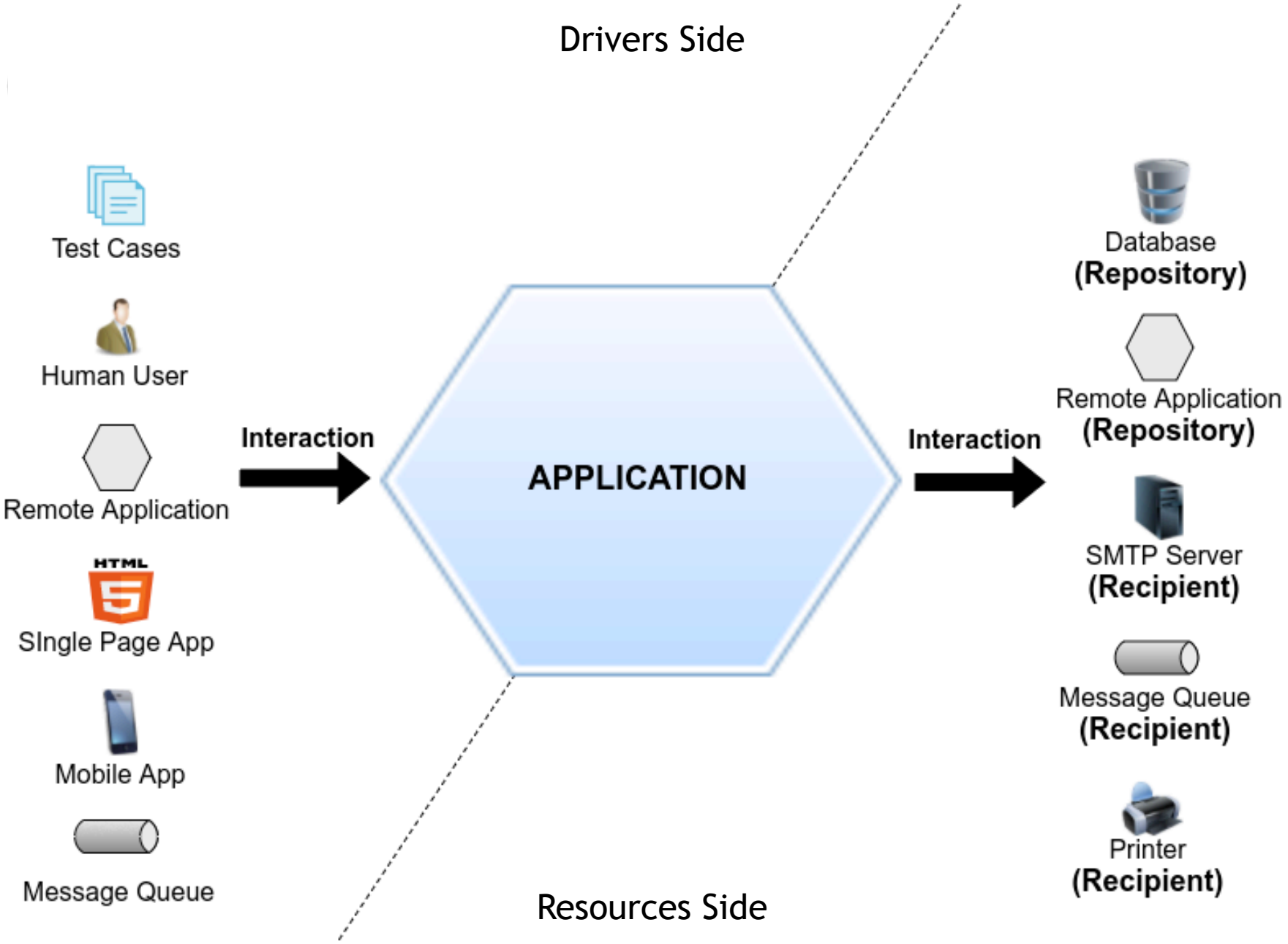
```php
<?php
    import_request_variables("pg", "form_");
    $db = mysql_connect("localhost:/export/mysql/mysql.sock");
    mysql_select_db("app", $db);
    $rows = mysql_query("select * from transaction where hash = '$form_hash'", $db);
    if(mysql_num_rows($rows) == 0) {
        $error = "Transaction not found";
    } else {
        $transaction_hash = mysql_result($rows, 0, 0);
        $transaction_amount = mysql_result($rows, 0, 1);
        $transaction_due_date = mysql_result($rows, 0, 2);
        $today = time();
        $diff_in_days = ($today - $transaction_due_date) / (60 * 60 * 24);
        $transaction_penalty = ($transaction_amount * 2) / 100;
        $transaction_interest = (($transaction * 0.033) / 100) * $diff_in_days;
        $transaction_total_amount = $transaction_amount + $transaction_penalty + $transaction_interest;
    }
?>
<html>
    <head>
        <title>Transaction</title>
    </head>
    <body>
        <?php
            if($error) {
                echo "<h1>Error accessing transaction</h1>\n";
                echo "<p>$error</p>\n";
            } else {
                echo "<h1>Information about $form_hash</h1>\n";
                echo "<p>amount: $transaction_amount</p>\n";
                echo "<p>due date: $transaction_due_date</p>\n";
                echo "<p>penalty: $transaction_penalty</p>\n";
                echo "<p>interest: $transaction_interest</p>\n";
                echo "<p>total: $transaction_total</p>\n";
```

user.php — cccat13_2

user.php U

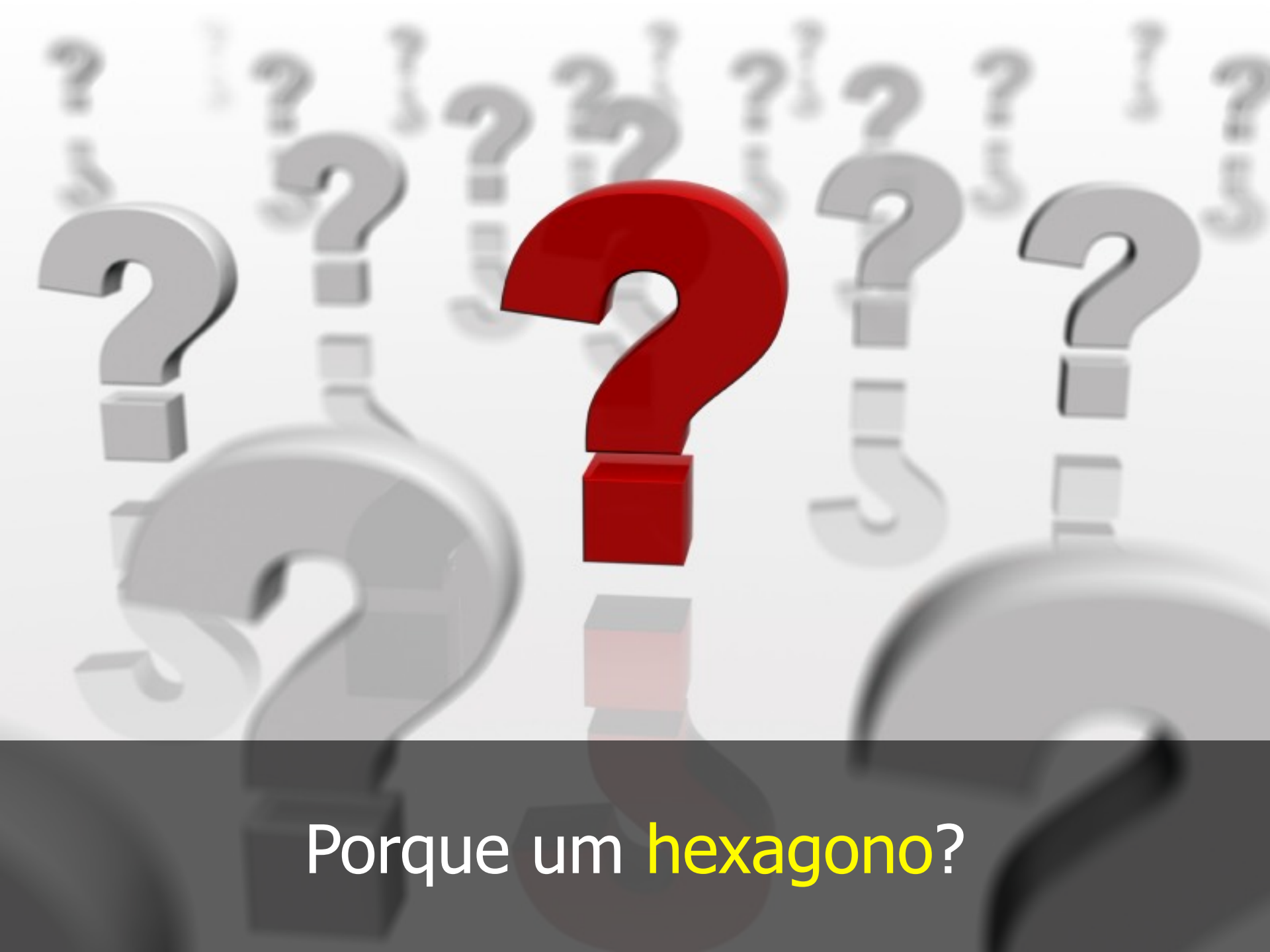
```php
<?php
    import_request_variables("pg", "form_");
    $db = mysql_connect("localhost:/export/mysql/mysql.sock");
    mysql_select_db("app", $db);
    $rows = mysql_query("select * from transaction where hash = '$form_hash'", $db);
    if(mysql_num_rows($rows) == 0) {
        $error = "Transaction not found";
    } else {
        $transaction_hash = mysql_result($rows, 0, 0);
        $transaction_amount = mysql_result($rows, 0, 1);
        $transaction_due_date = mysql_result($rows, 0, 2);
        $today = time();
        $diff_in_days = ($today - $transaction_due_date) / (60 * 60 * 24);
        $transaction_penalty = ($transaction_amount * 2) / 100;
        $transaction_interest = (($transaction * 0.033) / 100) * $diff_in_days;
        $transaction_total_amount = $transaction_amount + $transaction_penalty + $transaction_interest;
    }
?>
<html>
    <head>
        <title>Transaction</title>
    </head>
    <body>
        <?php
            if($error) {
                echo "<h1>Error accessing transaction</h1>\n";
                echo "<p>$error</p>\n";
            } else {
                echo "<h1>Information about $form_hash</h1>\n";
                echo "<p>amount: $transaction_amount</p>\n";
                echo "<p>due date: $transaction_due_date</p>\n";
                echo "<p>penalty: $transaction_penalty</p>\n";
                echo "<p>interest: $transaction_interest</p>\n";
                echo "<p>total: $transaction_total</p>\n";
```


user.php — cccat13_2
user.php U
<?php
import_request_variables("pg", "form_");
$db = mysql_connect("localhost:/export/mysql/mysql.sock");
mysql_select_db("app", $db);
$rows = mysql_query("select * from transaction where hash = '$form_hash'", $db);
if(mysql_num_rows($rows) == 0) {
$error = "Transaction not found";
} else {
$transaction_hash = mysql_result($rows, 0, 0);
$transaction_amount = mysql_result($rows, 0, 1);
$transaction_due_date = mysql_result($rows, 0, 2);
$today = time();
$diff_in_days = ($today - $transaction_due_date) / (60 * 60 * 24);
$transaction_penalty = ($transaction_amount * 2) / 100;
$transaction_interest = (($transaction * 0.033) / 100) * $diff_in_days;
$transaction_total_amount = $transaction_amount + $transaction_penalty + $transaction_interest;
}
?>
<html>
<head>
<title>Transaction</title>
</head>
<body>
<?php
if($error) {
echo "<h1>Error accessing transaction</h1>\n";
echo " $error \n";
} else {
echo "<h1>Information about $form_hash</h1>\n";
echo " amount: $transaction_amount \n";
echo " due date: $transaction_due_date \n";
echo " penalty: $transaction_penalty \n";
echo " interest: $transaction_interest \n";
echo " total: $transaction_total \n";


Drivers Side
Test Cases
Human User
Remote Application
SIngle Page App
Mobile App
Message Queue
Interaction
APPLICATION
Interaction
Database
(Repository)
Remote Application
(Repository)
SMTP Server
(Recipient)
Message Queue
(Recipient)
Printer
(Recipient)
Resources Side

"Allow an application to equally be driven by users, programs, automated test or batch scripts, and to be developed and tested in isolation from its eventual run-time devices and databases"

Alistair Cockburn

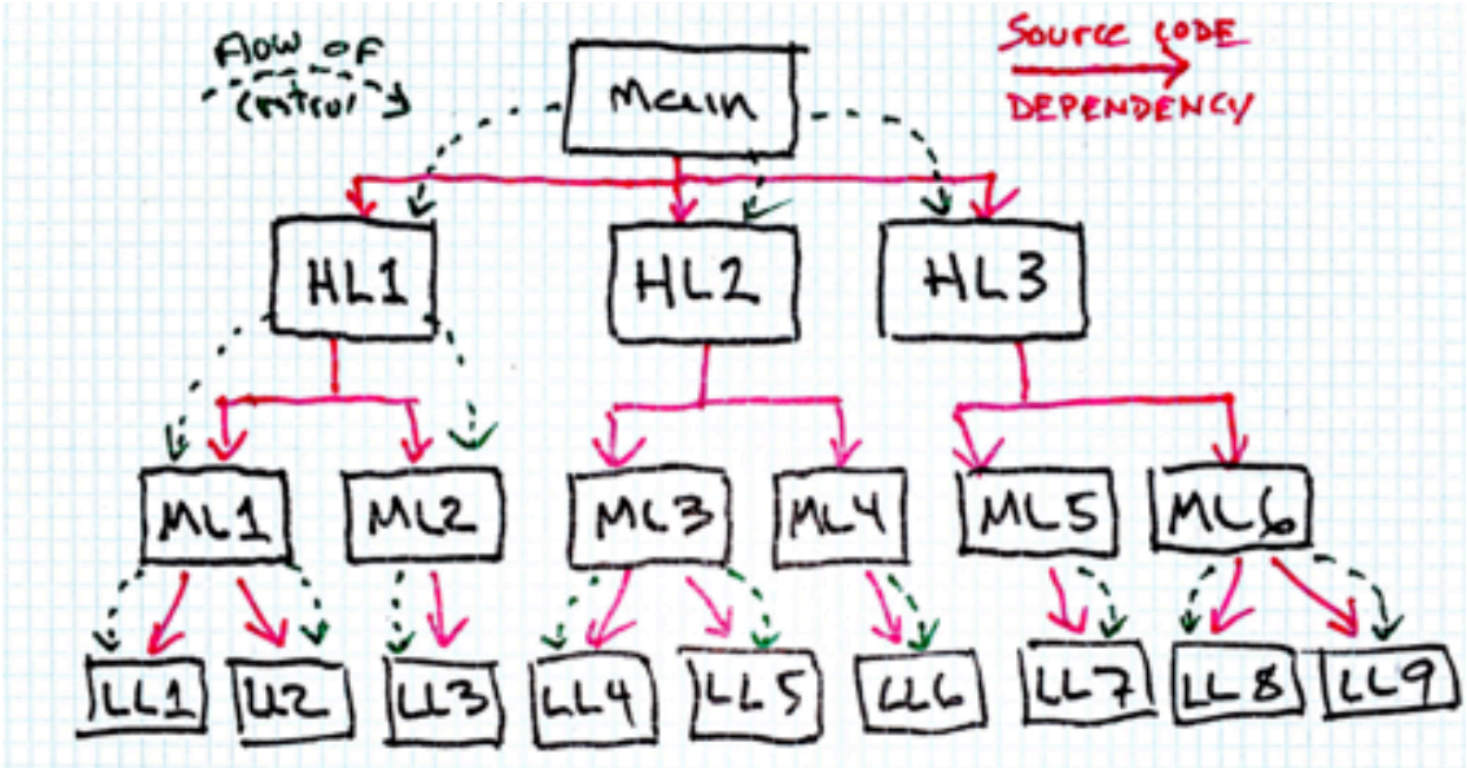

Porque um hexagono?

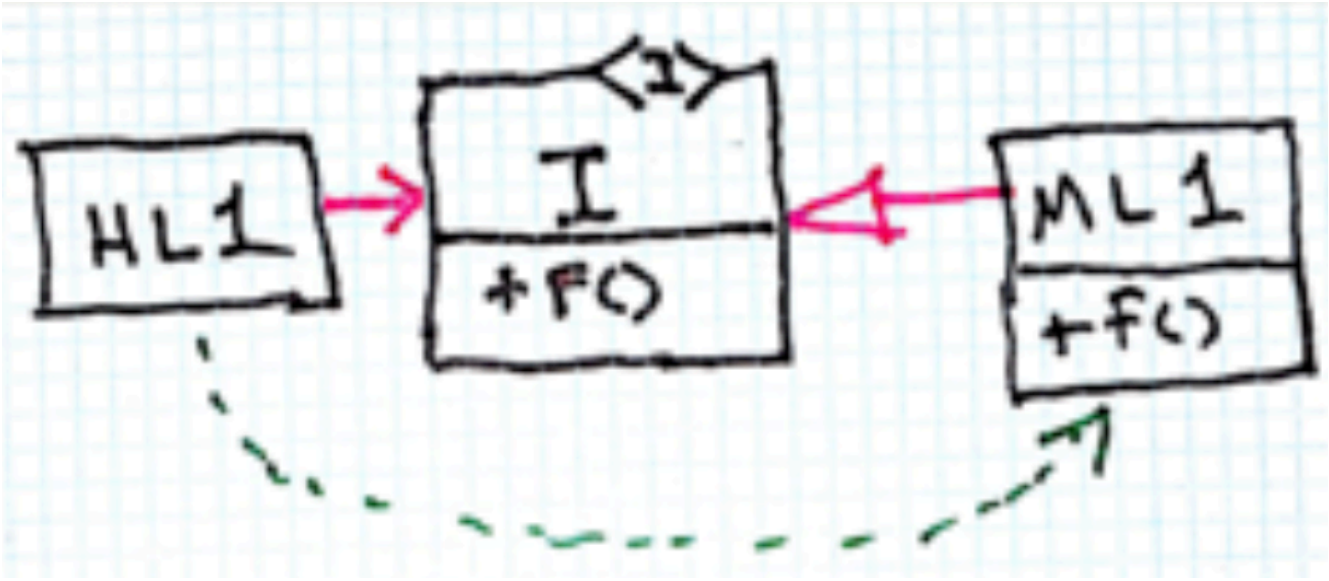
"The hexagon is not a hexagon because the number six is important, but rather to allow the people doing the drawing to have room to insert ports and adapters as they need, not being constrained by a one-dimensional layered drawing. The term hexagonal architecture comes from this visual effect"

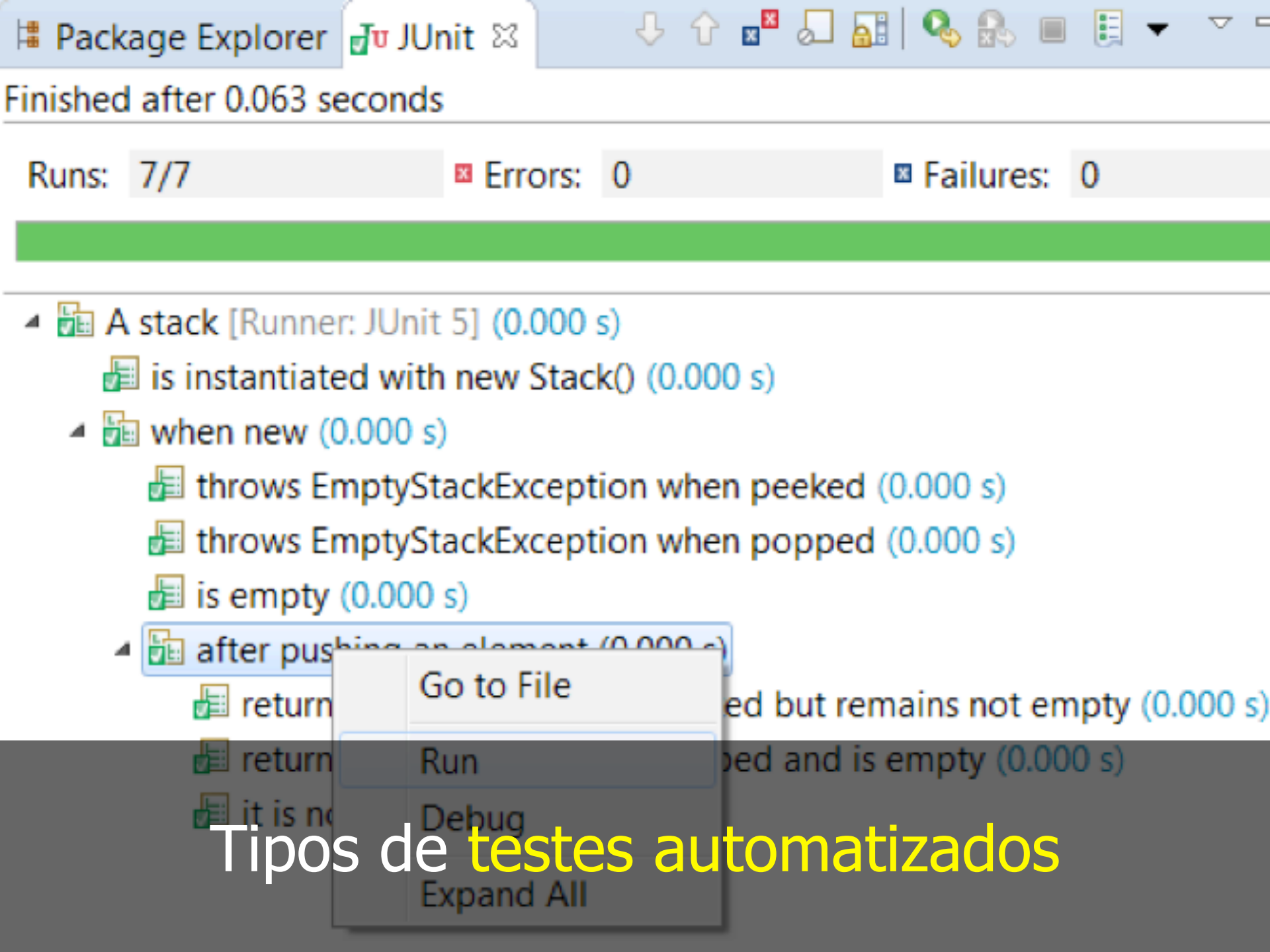
Alistair Cockburn

SOLID - Dependency Inversion Principle (DIP)

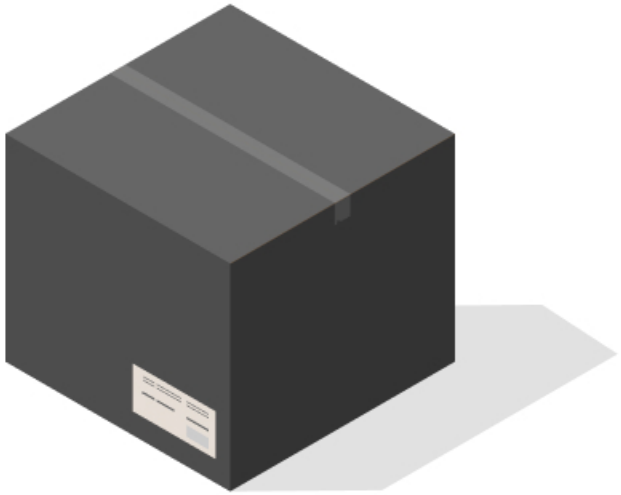
High-level modules should not depend on low-level modules, both should depend on abstractions


Flow of control
Main
Source code dependency
HL1
HL2
HL3
ML1
ML2
ML3
ML4
ML5
ML6
LL1
LL2
LL3
LL4
LL5
LL6
LL7
LL8
LL9


HL1
<I>
I
+f()
ML1
+f()


Package Explorer
JUnit
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)
Go to File
Run
Debug
Expand All
Tipos de testes automatizados

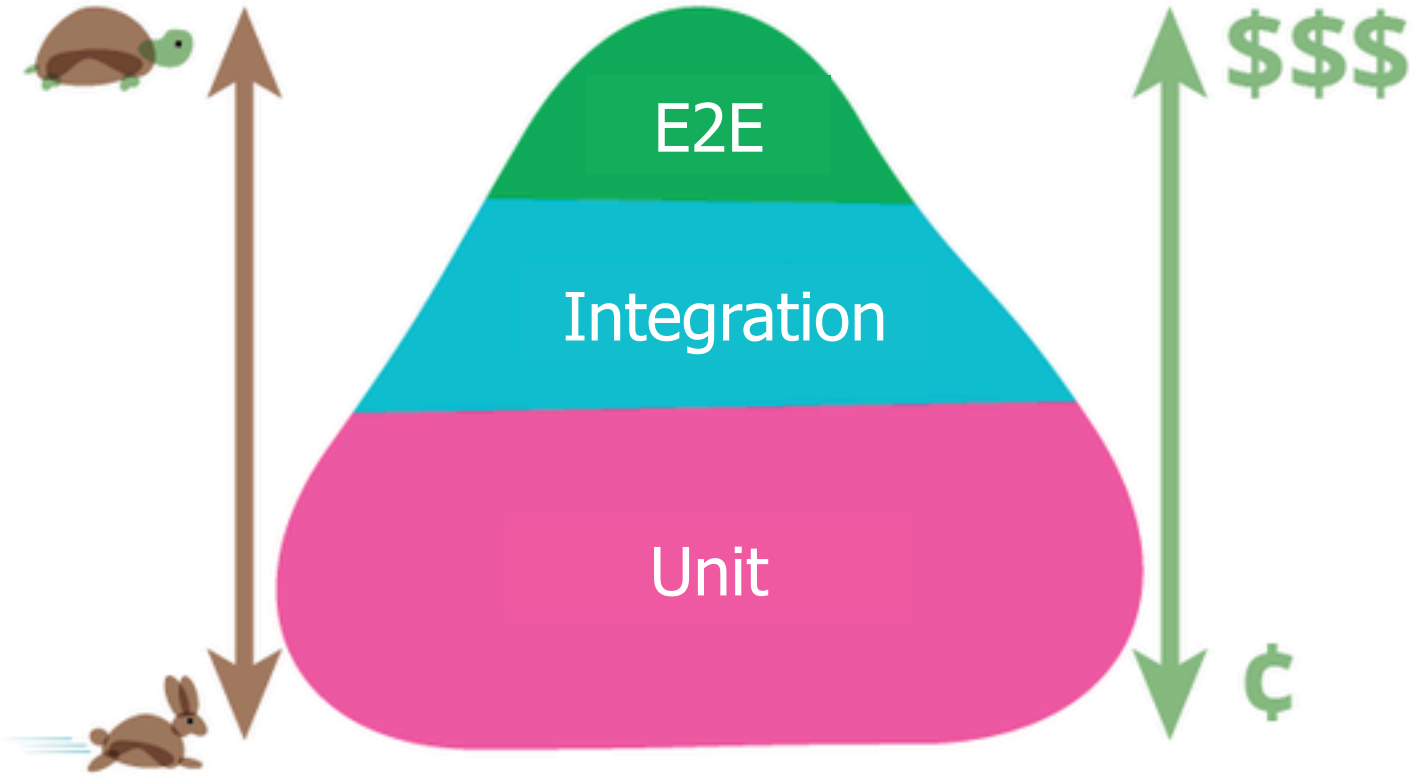
QA testers

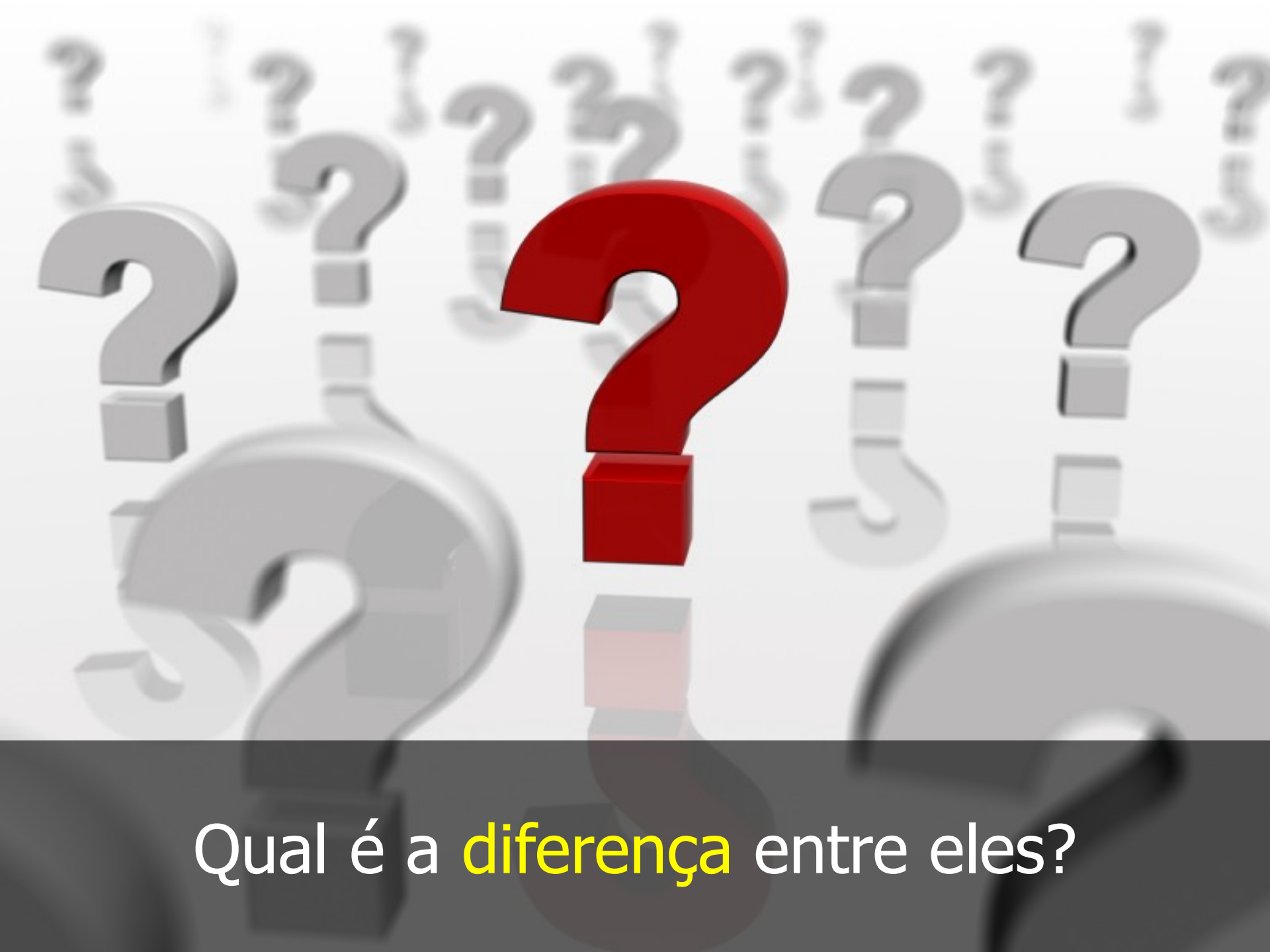


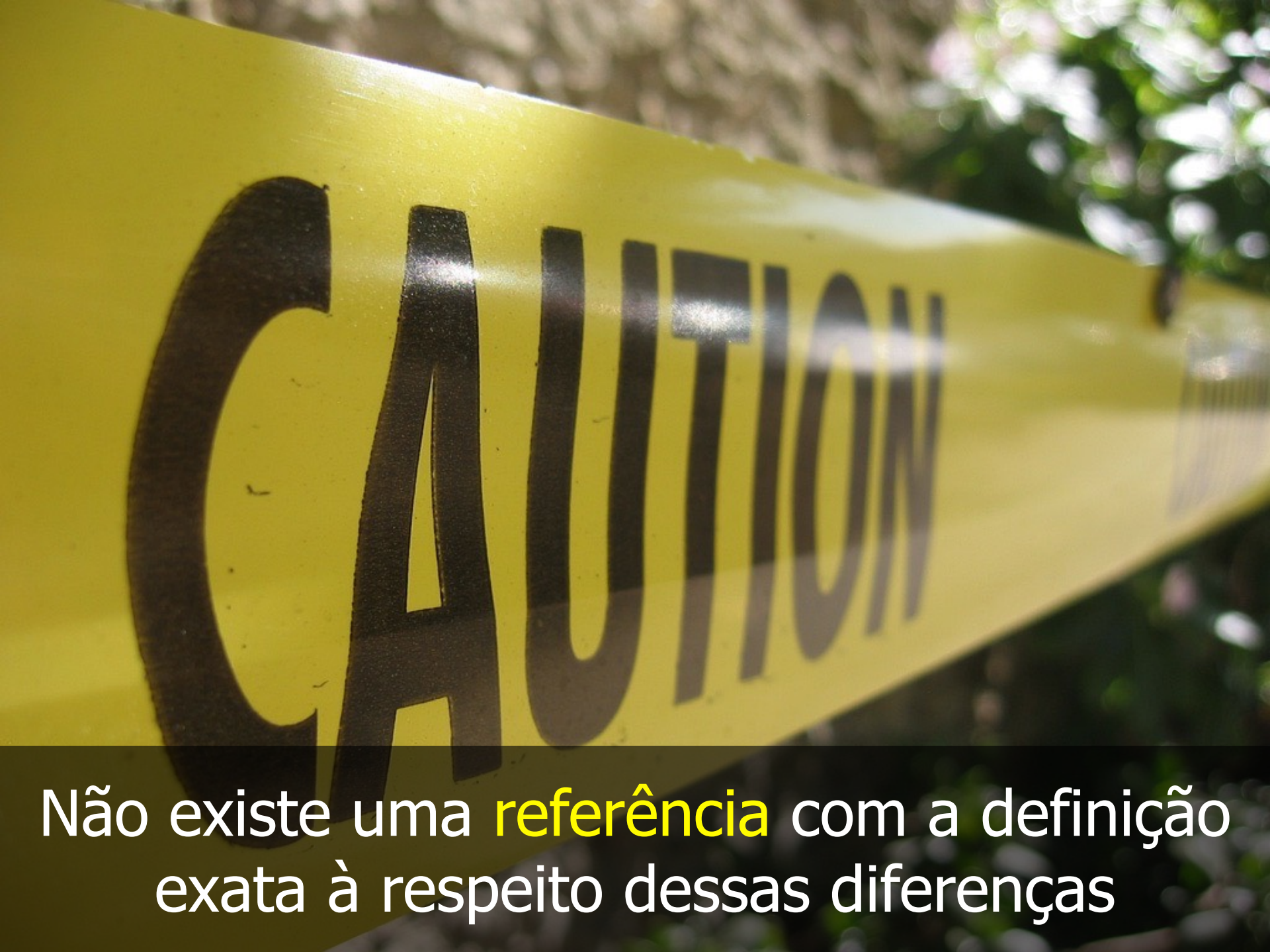
Black box - we do not know anything

Developers

White box - we know everything


E2E
Integration
Unit
$$$
¢


Qual é a diferença entre eles?

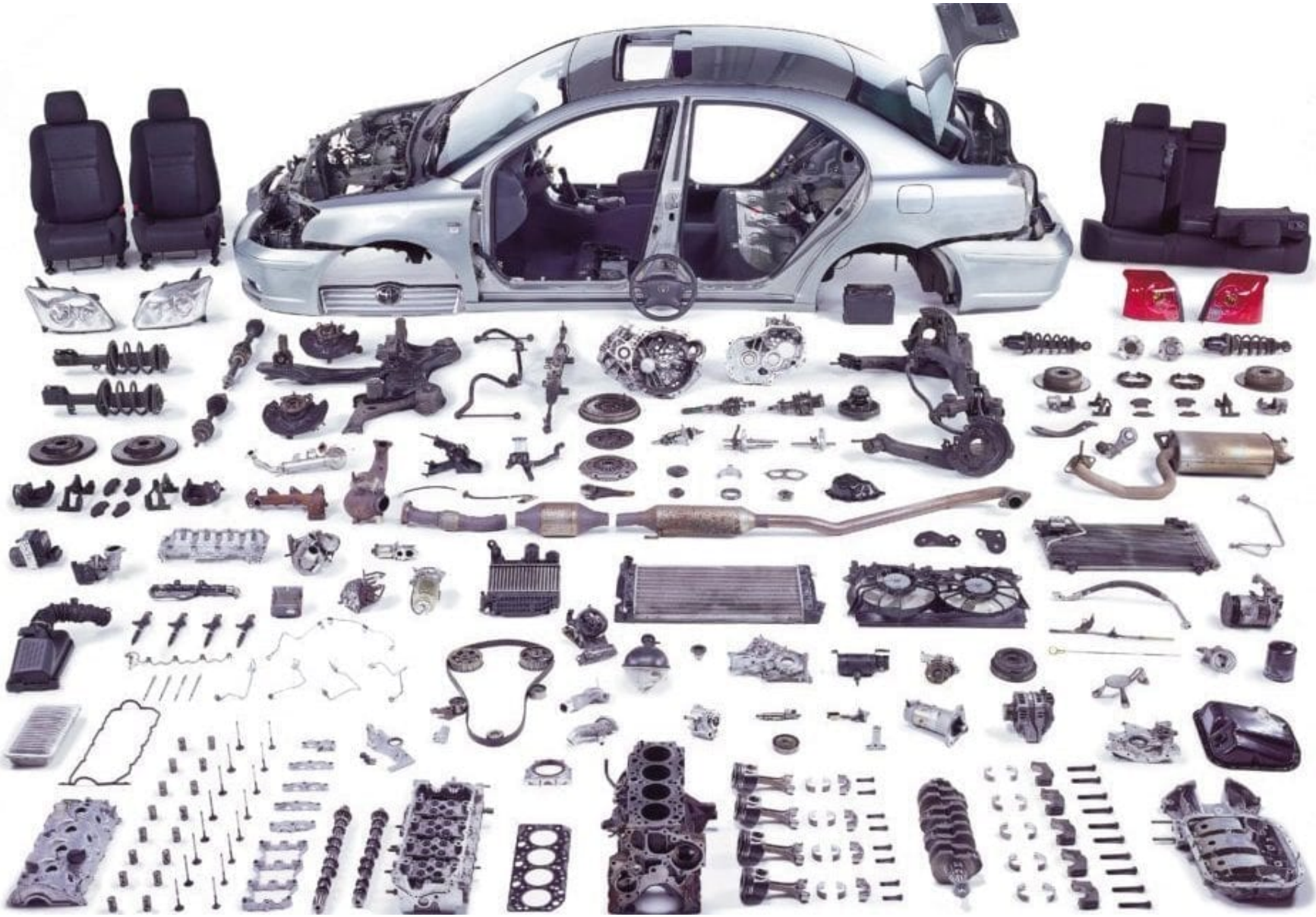


Não existe uma referência com a definição exata à respeito dessas diferenças

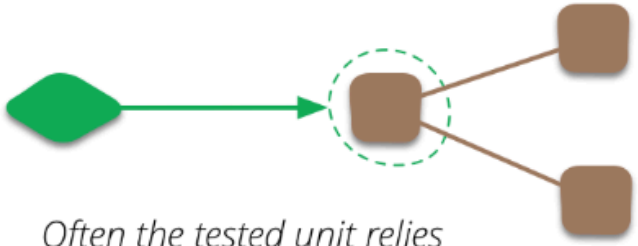
Unit Tests

São testes de unidade, não necessariamente unitários, que podem ou não envolver mais de uma classe, que não representam um requisito funcional e sem qualquer interação com recursos externos como um banco de dados, uma API ou o sistema de arquivos

São muito rápidos, estáveis e resistentes

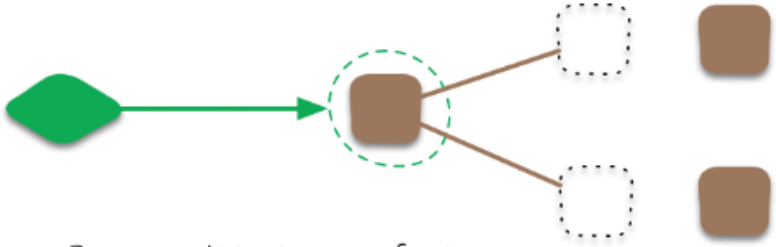


Sociable Tests



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.

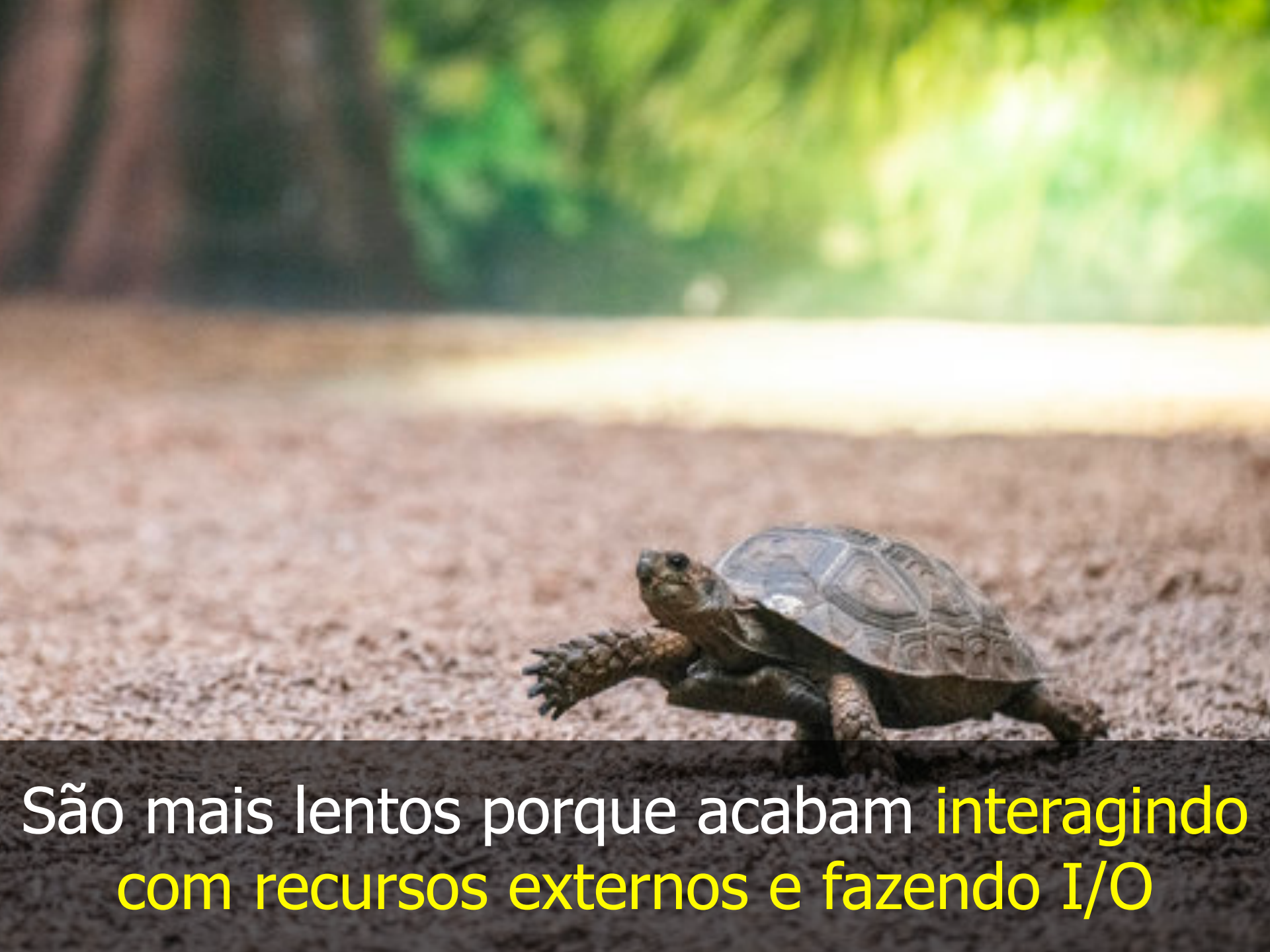
Indeed using sociable unit tests was one of the reasons we were criticized for our use of the term "unit testing". I think that the term "unit testing" is appropriate because these tests are tests of the behavior of a single unit. We write the tests assuming everything other than that unit is working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

https://martinfowler.com/bliki/UnitTest.html

Integration Tests

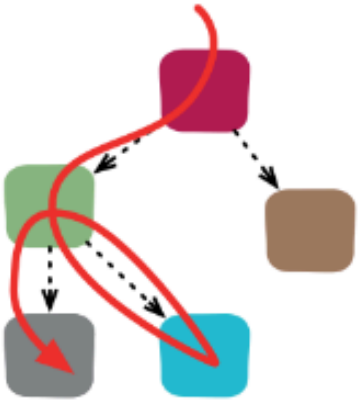
Testam componentes diferente, pertencentes à múltiplas camadas, normalmente envolvendo recursos externos, sejam eles reais ou não, ou seja, o fato de utilizar um Test Pattern como um stub ou mock não torna o teste de unidade.

Em geral são mais lentos por fazerem I/O

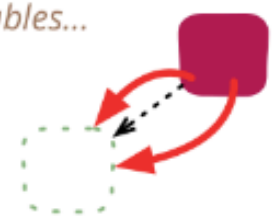


São mais lentos porque acabam interagindo com recursos externos e fazendo I/O

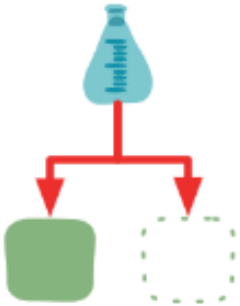
Integration Testing commonly refers to broad tests done with many modules active...



...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



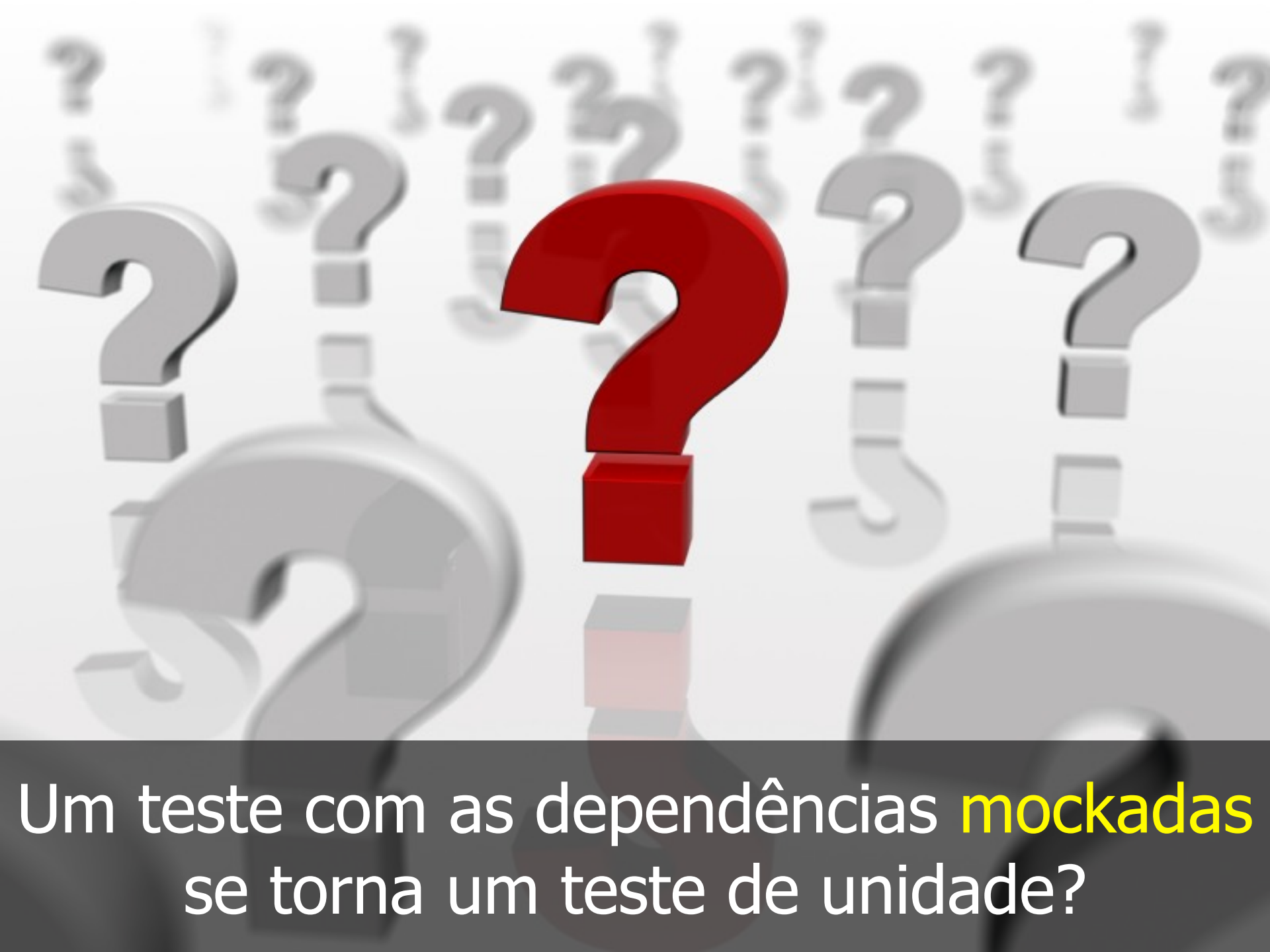
narrow integration tests

- 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

https://martinfowler.com/bliki/IntegrationTest.html


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

E2E Tests

Replicam o ambiente do usuário final, ou seja, são testes executados de ponta a ponta envolvendo o frontend, backend e banco de dados

Ferramentas

Esse tipo de teste tem uma dependência muito grande de ferramentas como o Selenium, Cypress, Playwright ou variações envolvendo WebDriver ou Puppeteer

Além de lentos, são frágeis e sensíveis a modificações, principalmente no frontend

Não existe um teste melhor que outro, o ideal é ter uma combinação deles

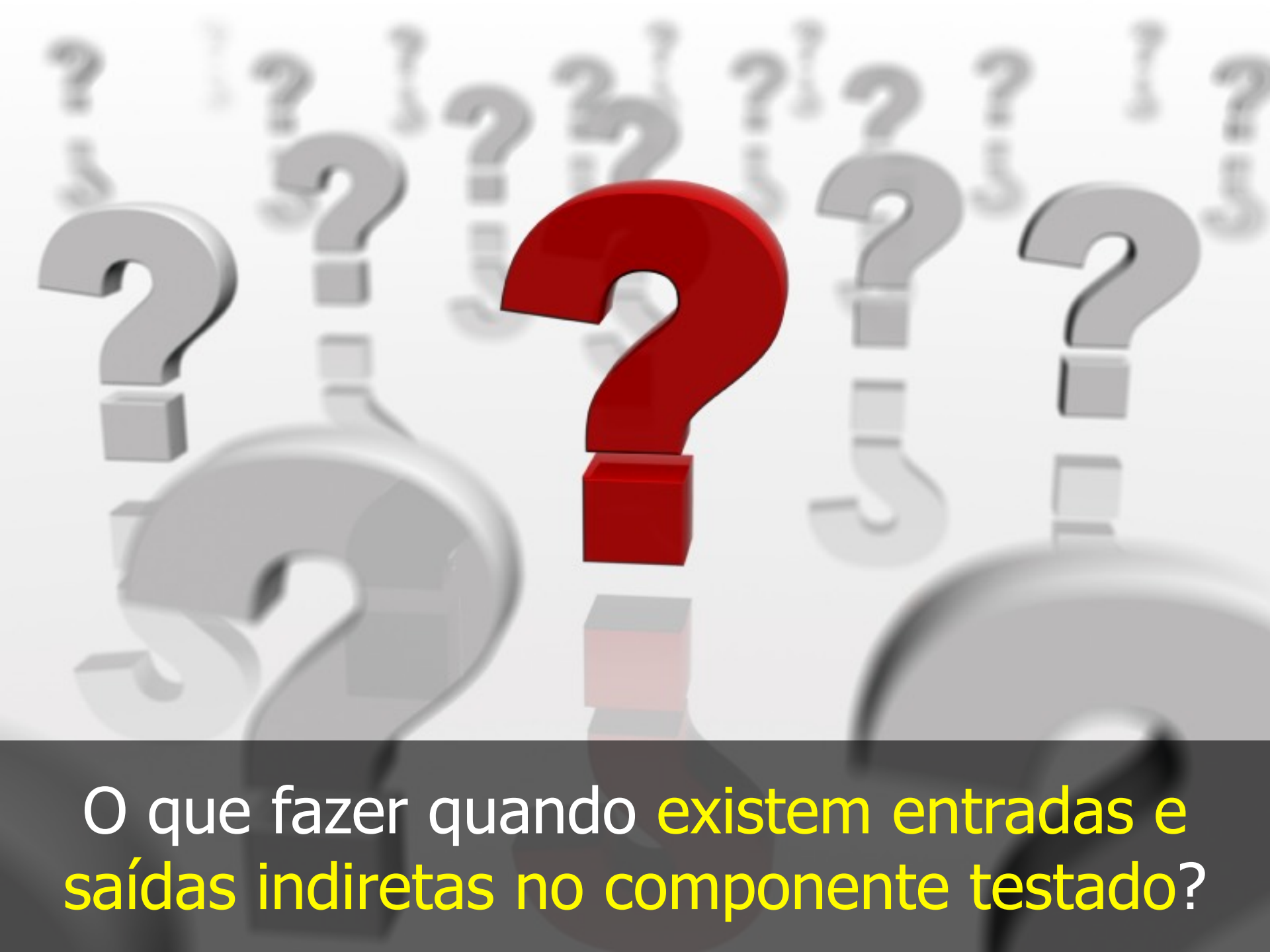

Test Patterns

Testar nem sempre é simples

Exemplo

Deve fazer um pedido com 3 itens em dólares

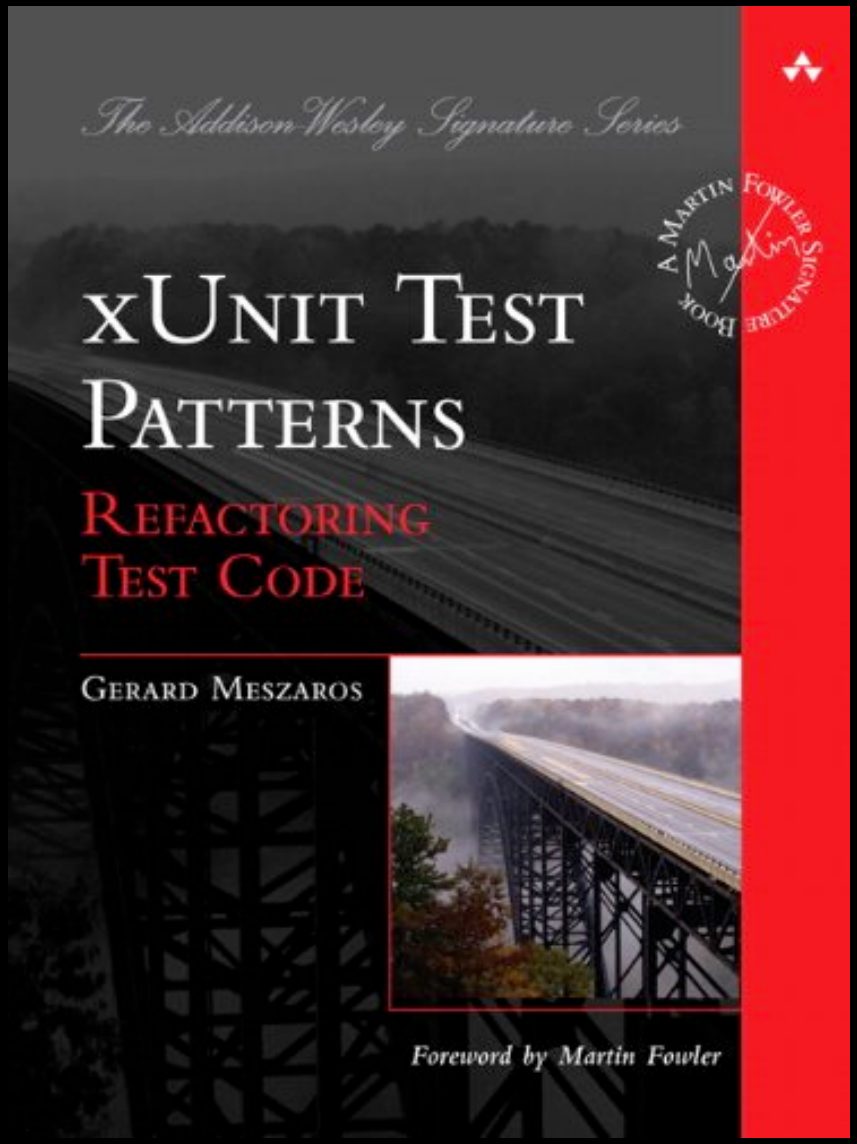
Dado um novo pedido com 3 itens associados, um Livro de $50,00, um CD de $20,00 e um DVD de $30,00
Quando o pedido for realizado
Então deve ser retornado uma confirmação do pedido contendo o código, juntamente com o total do pedido de R$550,00, se a cotação do dólar for R$5,50 e o status aguardando pagamento

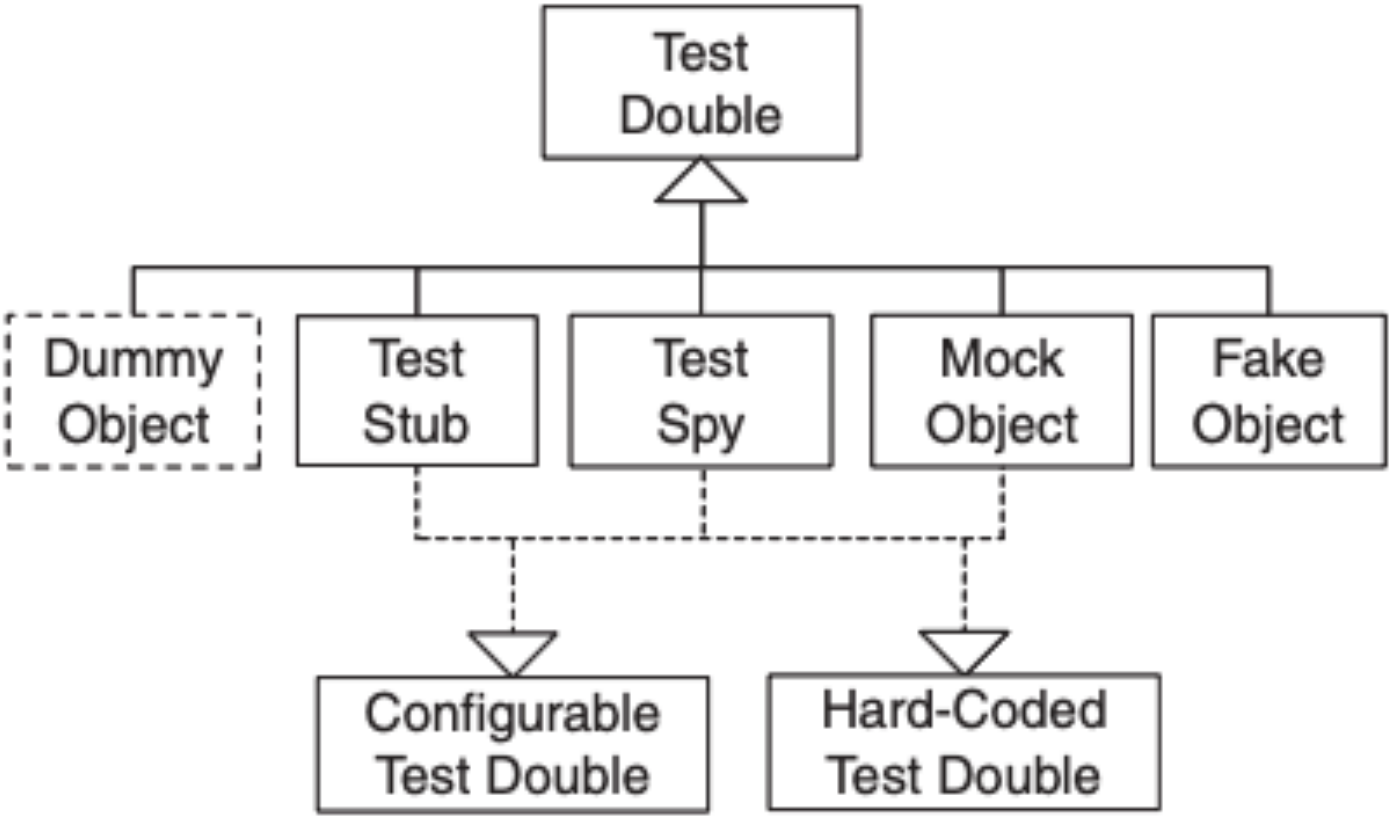


O que fazer quando existem entradas e saídas indiretas no componente testado?

Um test double é um padrão que tem o objetivo de substituir um DOC (depended-on component) em um determinado tipo de teste por motivos de performance ou segurança

Nem tudo é mock


The Addison Wesley Signature Series
A Martin Fowler Signature Book
xUnit Test Patterns
Refactoring Test Code
Gerard Meszaros
Foreword by Martin Fowler


Test
Double
Dummy
Object
Test
Stub
Test
Spy
Mock
Object
Fake
Object
Configurable
Test Double
Hard-Coded
Test Double

Test Patterns

Dummy: Objetos que criamos apenas para completar a lista de parâmetros que precisamos passar para invocar um determinado método

Test Patterns

Dummy: Objetos que criamos apenas para completar a lista de parâmetros que precisamos passar para invocar um determinado método

Stubs: Objetos que retornam respostas prontas, definidas para um determinado teste, por questão de performance ou segurança (exemplo: quando eu executar o método fazer pedido preciso que o método pegar cotação do dólar retorne R$3,00)

Test Patterns

Dummy: Objetos que criamos apenas para completar a lista de parâmetros que precisamos passar para invocar um determinado método

Stubs: Objetos que retornam respostas prontas, definidas para um determinado teste, por questão de performance ou segurança (exemplo: quando eu executar o método fazer pedido preciso que o método pegar cotação do dólar retorne R$3,00)

Spies: Objetos que "espionam" a execução do método e armazenam os resultados para verificação posterior (exemplo: quando eu executar o método fazer pedido preciso saber se o método enviar email foi invocado internamente e com quais parâmetros)

Test Patterns

Dummy: Objetos que criamos apenas para completar a lista de parâmetros que precisamos passar para invocar um determinado método

Stubs: Objetos que retornam respostas prontas, definidas para um determinado teste, por questão de performance ou segurança (exemplo: quando eu executar o método fazer pedido preciso que o método pegar cotação do dólar retorne R$3,00)

Spies: Objetos que "espionam" a execução do método e armazenam os resultados para verificação posterior (exemplo: quando eu executar o método fazer pedido preciso saber se o método enviar email foi invocado internamente e com quais parâmetros)

Mocks: Objetos similares a stubs e spies, permitem que você diga exatamente o que quer que ele faça e o teste vai quebrar se isso não acontecer

Test Patterns

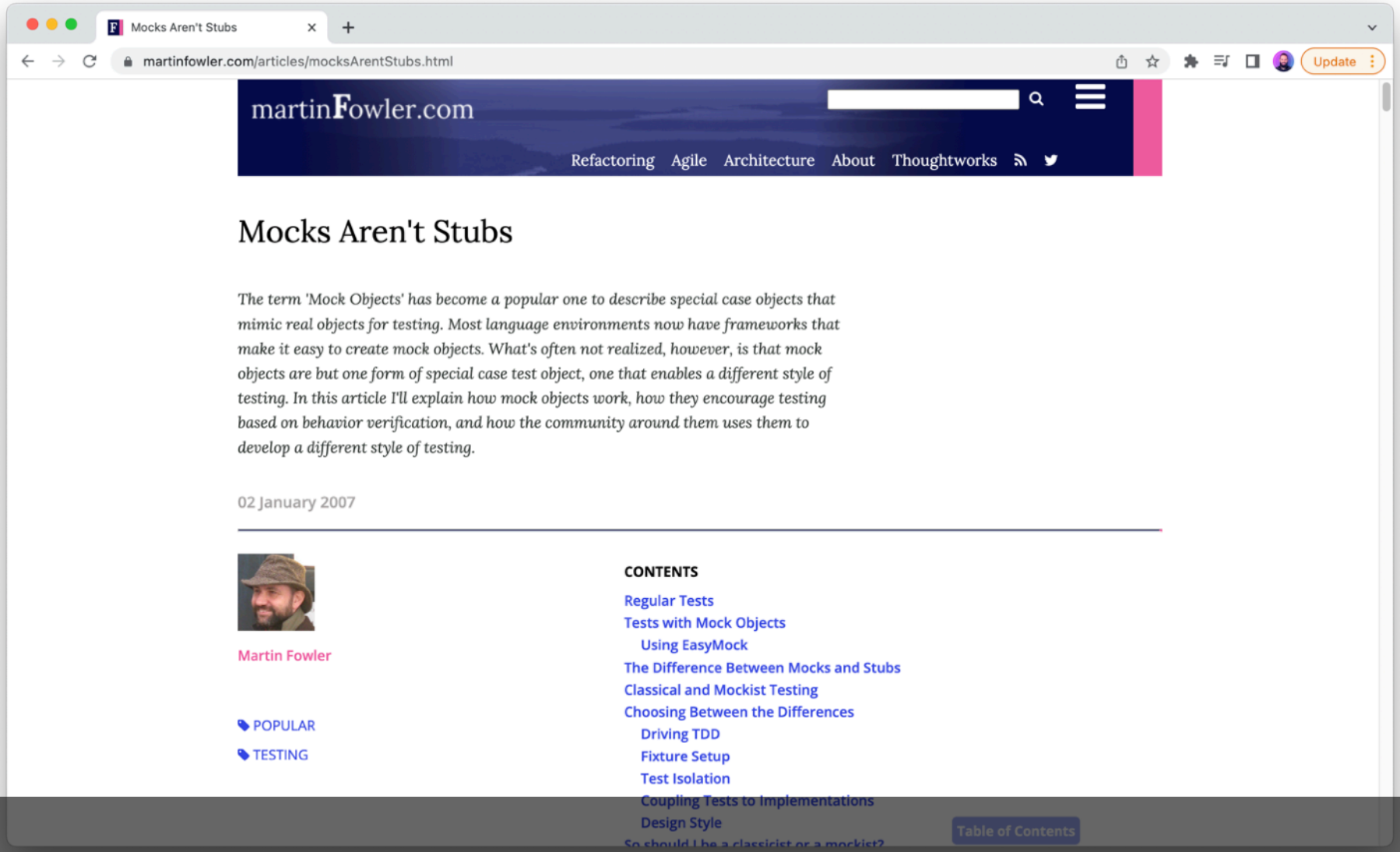
Dummy: Objetos que criamos apenas para completar a lista de parâmetros que precisamos passar para invocar um determinado método

Stubs: Objetos que retornam respostas prontas, definidas para um determinado teste, por questão de performance ou segurança (exemplo: quando eu executar o método fazer pedido preciso que o método pegar cotação do dólar retorne R$3,00)

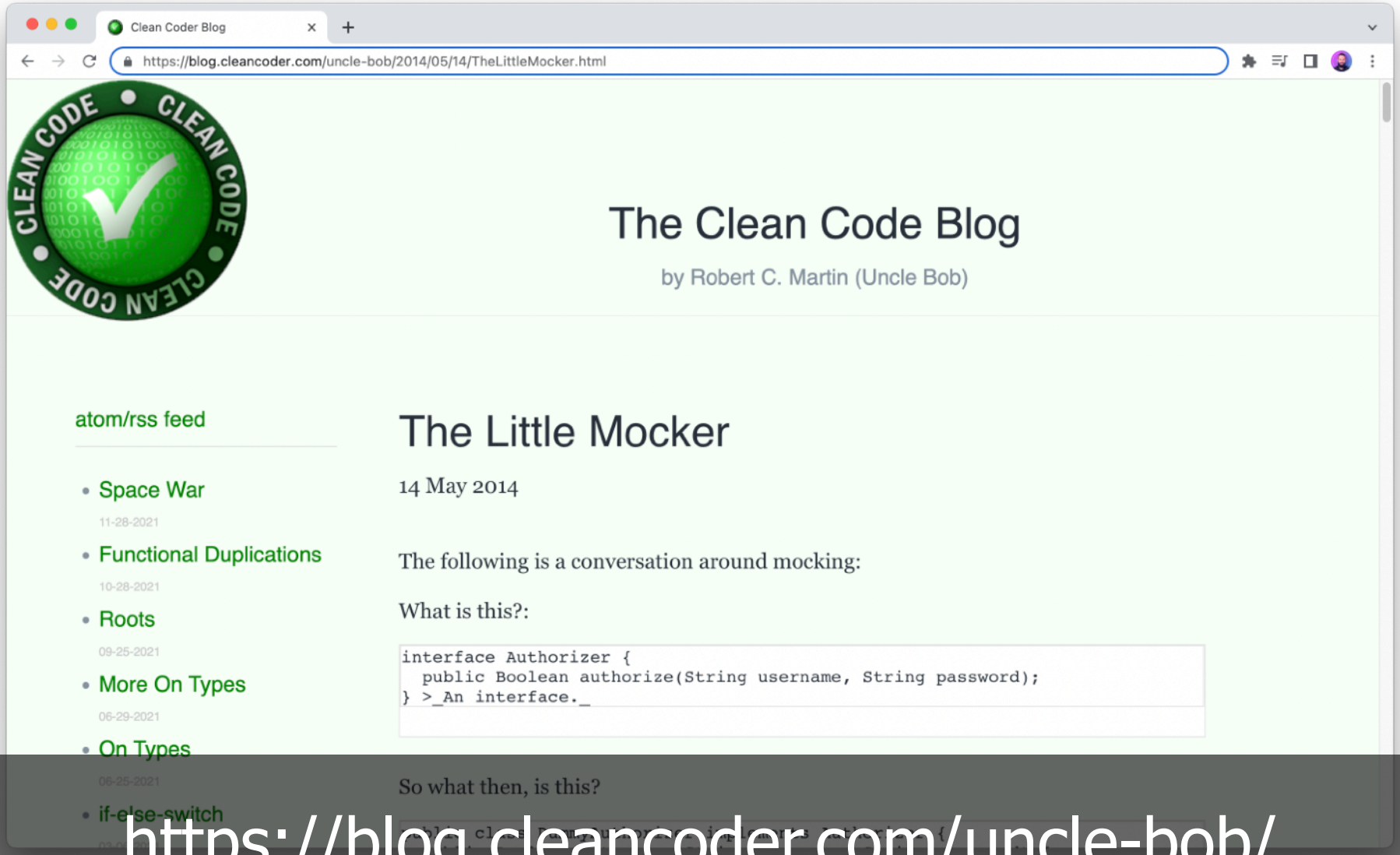
Spies: Objetos que "espionam" a execução do método e armazenam os resultados para verificação posterior (exemplo: quando eu executar o método fazer pedido preciso saber se o método enviar email foi invocado internamente e com quais parâmetros)

Mocks: Objetos similares a stubs e spies, permitem que você diga exatamente o que quer que ele faça e o teste vai quebrar se isso não acontecer

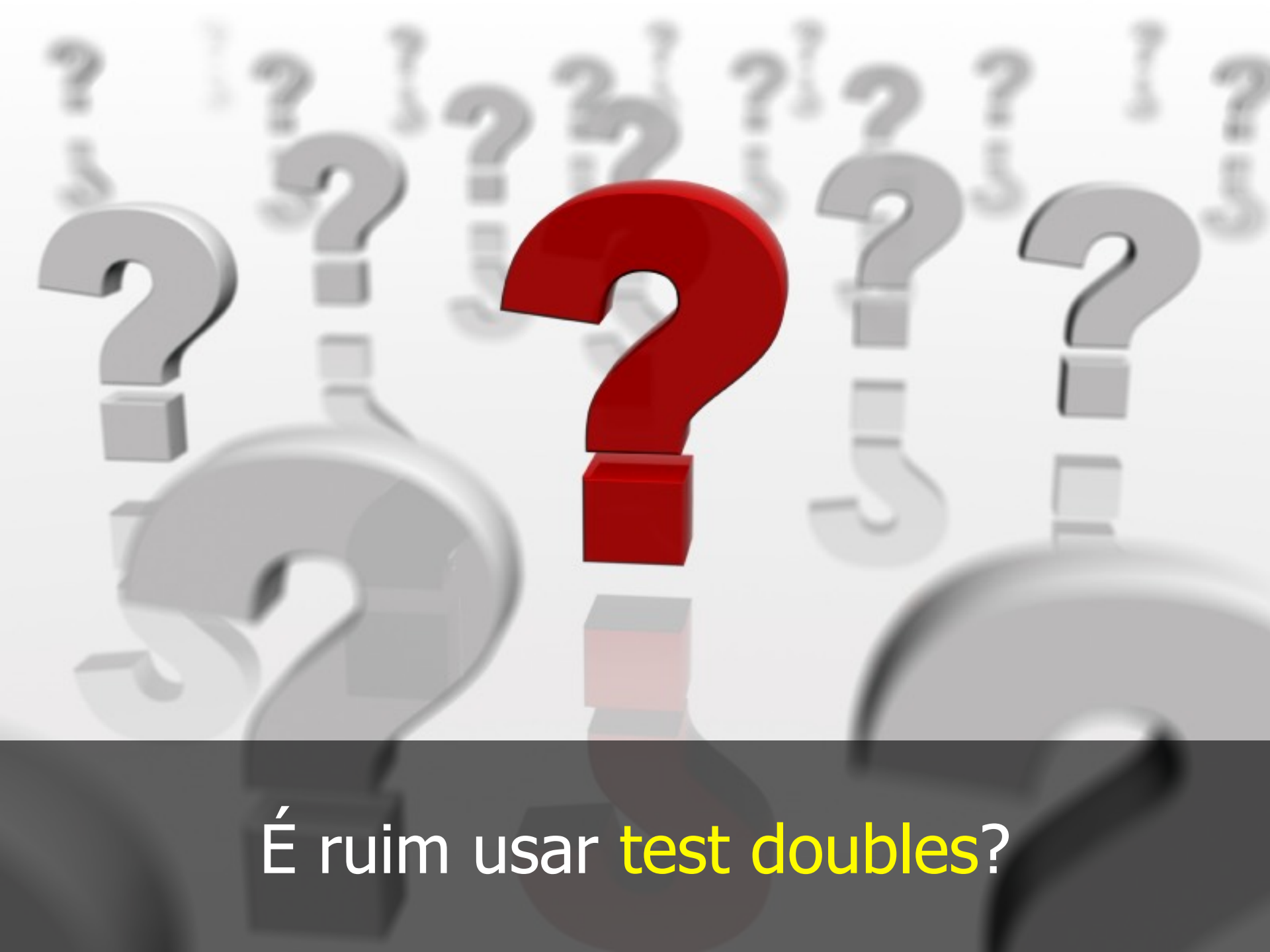
Fake: Objetos que tem implementações que simulam o funcionamento da instância real, que seria utilizada em produção (exemplo: uma base de dados em memória)



https://martinfowler.com/articles/mocksArentStubs.html



https://blog.cleancoder.com/uncle-bob/2014/05/14/TheLittleMocker.html


É ruim usar test doubles?