

---

---

# Testes unitários como ferramentas de design de código

FISL 17 - 15 de Julho de 2016

---

---

# Paula Grangeiro

28 anos

Bacharel em Sistemas de Informação

Duque de Caxias

Gateira

Python & Arquitetura de Software





elogroup▶

Onde me encontrar...

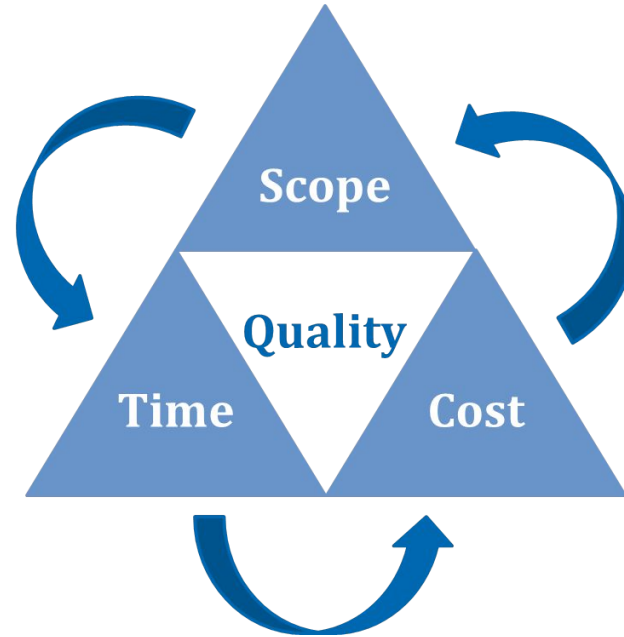
---

---

**Por que pensar em  
design de código?**

---

# Problemas do mundo real



---

# Problemas do mundo real



---

**+ Coesão**  
**- Acoplamento**

---


## + Coesão

Desenvolver estruturas de código especialistas ao máximo naquilo que fazem e que colaboram entre si para o funcionamento do sistema.



## - **Acoplamento**

Desenvolver estruturas de código isoláveis de maneira que a alteração ou remoção de um componente impacte o mínimo possível no sistema.

```
1004 
S>1005     temp_contato = {
1006 +---- 6 linhas: 0: objEleitor.values_list('numero_1', flat=True).count(),-----
1012     }
1013
1014     count_contato = (
1015 +---- 7 linhas: float(objEleitor.values_list('numero_1', flat=True).exclude(numero_1_isnull=True).count() +-----
1022     )
1023
1024     graph_data['telefone'] = {
1025 +---- 7 linhas: 0: float(float(temp_contato[0]) / count_contato * 100),-----
1032     }
1033
1034     email_informed = len(objEleitor.values_list('email', flat=True).exclude(email_isnull=False))
1035     email_uninformed = len(objEleitor.values_list('email', flat=True).exclude(email_isnull=True))
1036
1037     graph_data['email'] = {
1038 +---- 2 linhas: 0: float((float(email_uninformed) / graph_data['temp_count']) * 100),-----
1040     }
1041
1042     objRenda = objEleitor.values_list('renda_mensal_presumida', flat=True)
1043
1044     renda = dict(dict.fromkeys(['renda_mais_1356', 'renda_mais_2034', 'renda_mais_3390', 'renda_nao_informada'], 0))
1045
1046     for item in range(len(objRenda)):
1047 +---- 8 linhas: if objRenda[item] > 1356.00 and objRenda[item] < 2033.99:-----
1055
1056     count_renda = sum(renda.values())
1057
1058     graph_data['renda'] = {
1059 +---- 5 linhas: 0: (float(renda['renda_nao_informada']) / count_renda * 100),-----
1064     }
1065
1066     return graph_data
```

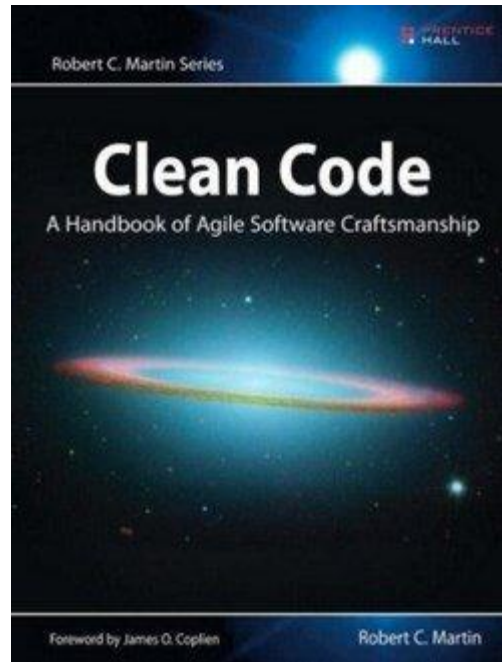


```
1004 
S>1005     temp_contato = {
1006 +---- 6 linhas: 0: objEleitor.values_list('numero_1', flat=True).count()
1012     }
1013
1014     count_contato = (
1015 +---- 7 linhas: float(objEleitor.values_list('numero_1', flat=True).count()) +
1022     )
1023
1024     graph_data['telefone'] = {
1025 +---- 7 linhas: 0: float(float(temp_contato[0])
1032     }
1033
1034     email_informed = len(objEleitor.values_list('email', flat=True).filter(email_isnull=False))
1035     email_uninformed = len(objEleitor.values_list('email', flat=True).filter(email_isnull=True))
1036
1037     graph_data['email'] = {
1038 +---- 2 linhas: 0: float((float(email_uninformed) / (email_informed + email_uninformed)) * 100),
1040     }
1041
1042     objRenda = objEleitor.values_list('renda', flat=True)
1043
1044     renda = dict(dict.fromkeys(['renda_mais_2034', 'renda_mais_2034', 'renda_nao_mais_2034'], 0))
1045
1046     for item in range(len(objRenda)):
1047 +---- 8 linhas: if objRenda[item] > 1356.00:
1055         if objRenda[item] < 2033.99:
1056             count_renda = sum(renda.values())
1057
1058     graph_data['renda'] = {
1059 +---- 5 linhas: 0: (float(renda['renda_nao_informada'] / count_renda) * 100,
1064     }
1065
1066     return graph_data
```

---

# Bad smell do dia-a-dia

- Código duplicado
  - Complexidade desnecessária
  - Linhas de código muito extensas
  - Feature envy
  - Intimidade inapropriada
  - Classes preguiçosas
  - Conascência
  - Dowcasting
  - Muitos parâmetros
  - Nomes de variáveis muito longos
  - Nomes de variáveis muito curtos
  - Comentários muito longos
  - Comentários desatualizados
  - Variáveis órfãs
  - etc.
-



Lidando com bad smells

---

---

**“Escrever código limpo é o que você deve fazer para que possa se intitular como profissional. Não existem desculpas plausíveis para fazer menos do que o seu melhor.” - Uncle Bob em Código Limpo**

---

---

# Design Patterns

## Padrões de Projeto

# GoF

Abstract Factory

Decorator

Mediator

Builder

Facade

Memento

Factory Method

Flyweight

Observer

Prototype

Proxy

State

Singleton

Chain of Responsibility

Strategy

Adapter

Command

Template Method

Bridge

Interpreter

Visitor

Composite

Iterator



# Design Patterns

Elements of Reusable  
Object-Oriented Software

Erich Gamma  
Richard Helm  
Ralph Johnson  
John Vlissides



Foreword by Grady Booch

ADDITION-WESLEY PROFESSIONAL COMPUTING SERIES

## APPLYING UML AND PATTERNS

An Introduction to Object-Oriented Analysis  
and Design and the Unified Process

SECOND EDITION



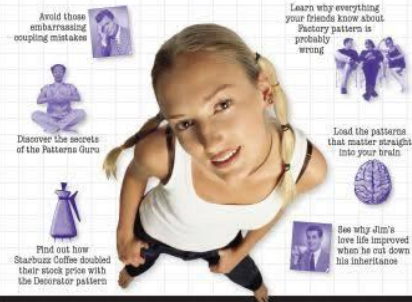
"People often ask me which is the best book to introduce them to the world of OO design. Ever since I came across it, Applying UML and Patterns has been my unreserved choice."

—Martin Fowler, author, UML: Unified Modeling Language

**CRAIG LARMAN**

Foreword by Philippe Kruchten

## A Brain-Friendly Guide Head First Design Patterns



O'REILLY\*

Eric Freeman & Elisabeth Freeman  
with Kathy Sierra & Bert Bates



## Aprendendo Design Patterns

---

**Design Patterns não são  
uma bala de prata**

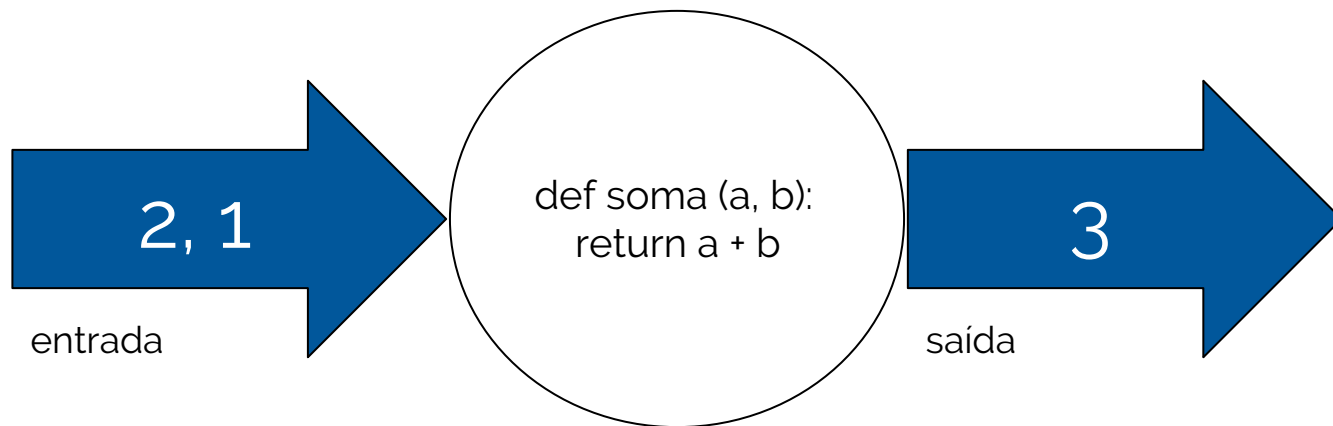
---

# Testes Unitários

# Testes unitários

Verificar o funcionamento de um fluxo garantindo que, a partir de uma entrada, uma saída esperada seja reproduzida.

# Testes Unitários



# Testes Unitários

```
from unittest import TestCase  
import soma
```

```
class SomaTestCase(TestCase):
```

```
    def test_sums_params_correctly(self):  
        expected = 3  
        value = soma(2, 1)  
        self.assertEqual(expected, value)
```

---

---

# Libs para Testes Unitários

- unittest
- Py.test
- Doctest

—

# Testes unitários como ferramentas de design?

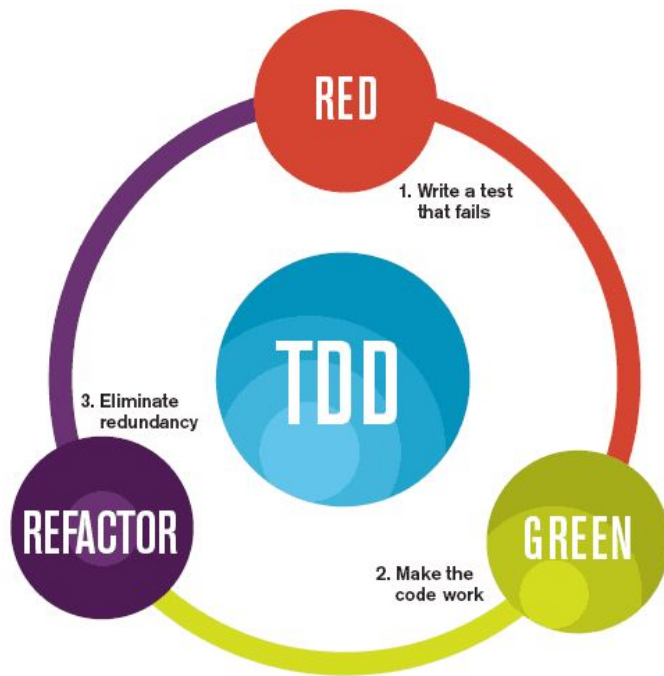


---

# 1. Pratique TDD

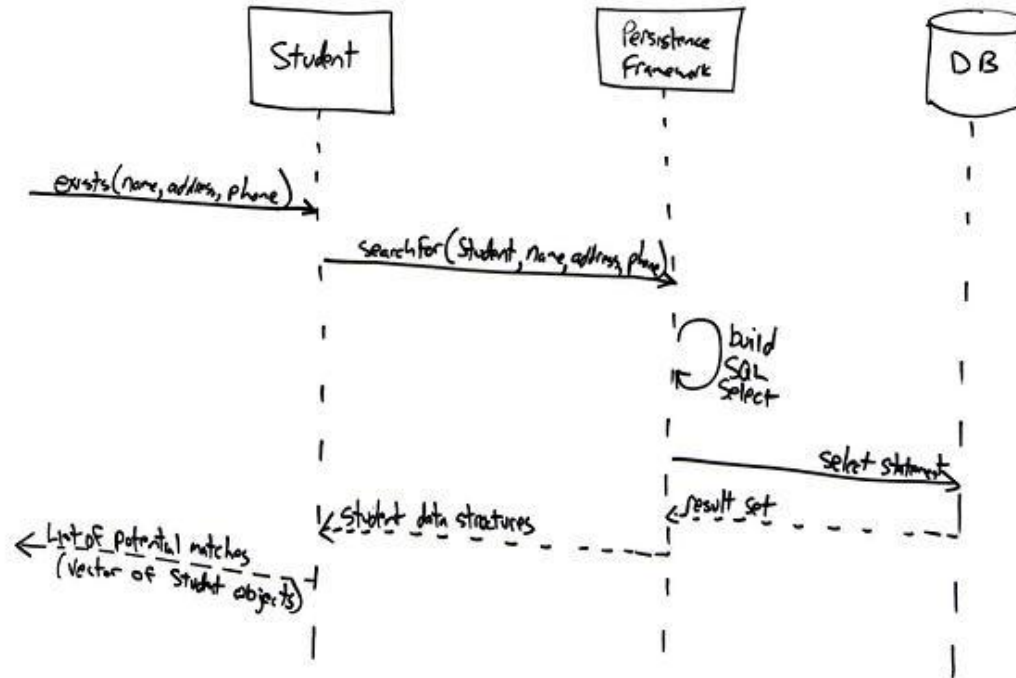
---

# Ciclo do TDD



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

# Como pensar em TDD?



# Praticar TDD

Auxilia no processo de definição do fluxo do código a partir do momento que você define as APIs de chamadas antes de implementá-las

---

## 2. TestCases devem ser pequenas

---

# Analisar TestCases

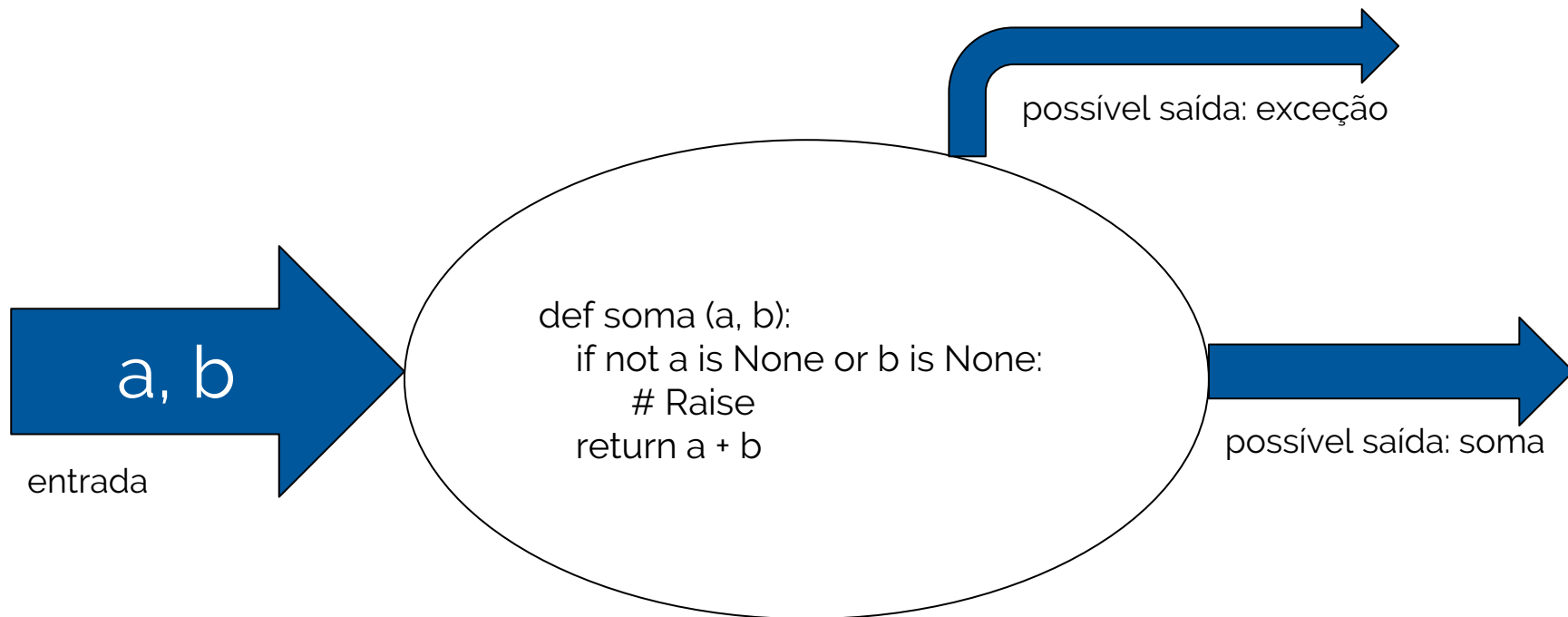
Auxilia na identificação de responsabilidades da classe testada, e possíveis bad smells, contribuindo com a coesão e diminuição do acoplamento do código que será desenvolvido.

---

# 3. Testes unitários devem ser unitários

---

# Testar unitariamente fluxos alternativos





# Testar unitariamente fluxos alternativos

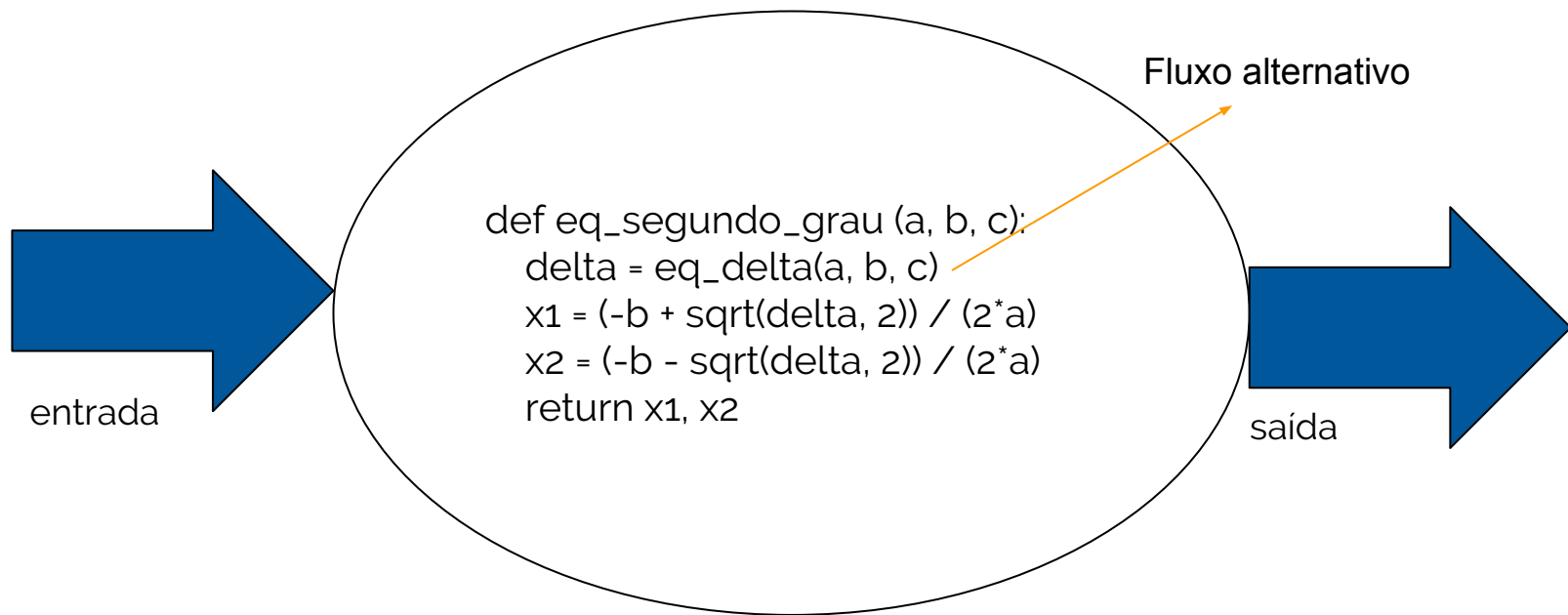
```
from unittest import TestCase
import soma
import SomeCustomException
```

```
class SomaTestCase(TestCase):
```

```
    def test_sums_params_correctly(self):
        expected = 3
        value = soma(2, 1)
        self.assertEqual(expected, value)
```

```
    def test_sum_raises_exception_when_some_param_has_is_none(self):
        self.assertRaises(SomeCustomException, soma, 2, None)
```

# Testar unitariamente fluxos alternativos



# Pensar unitariamente

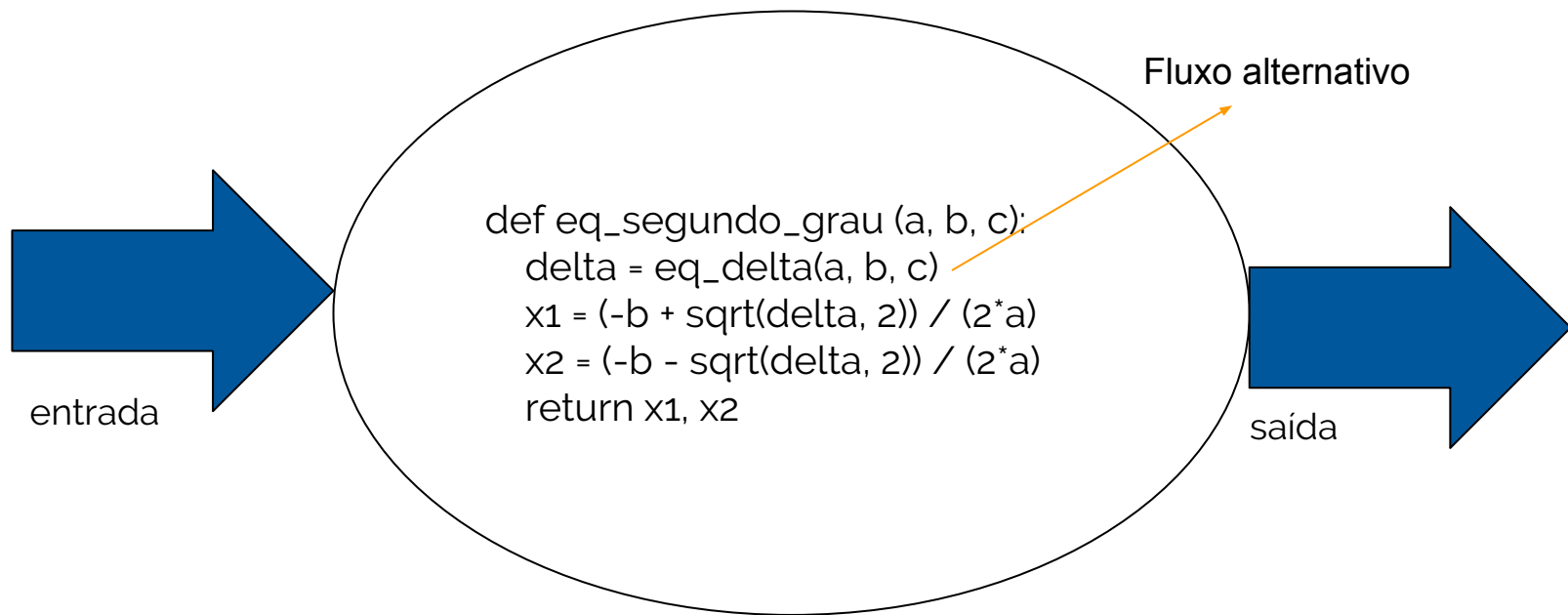
Auxilia no processo descoberta de fluxos de comportamento, também ajudando a manter a coesão e diminuição do acoplamento do código de menores unidades, como métodos de instância.

---

## 4. Testes unitários devem ser isolados

---

# Fluxos externos devem ser isolados



# Fluxos externos devem ser isolados

```
from unittest import TestCase  
import delta
```

```
class DeltaTestCase(TestCase):
```

```
    def test_calcs_delta_correctly(self):  
        expected = 4  
        value = delta(4, 6, -1)
```

```
        self.assertEqual(expected, value)
```

```
from mock import patch, Mock
from unittest import TestCase
import eq_segundo_grau
```

```
class EqTestCase(TestCase):
```

```
    @patch('delta', Mock(return_value=2))
    def test_calcs_x_params_correctly(self, mocked_delta):
        expected = 5, 8
        x1, x2 = eq_segundo_grau(3, -7, 1)

        mocked_delta.assert_called_once_with(3, -7, 1)
        self.assertEqual(expected, (x1, x2))
```

# Isolar fluxos externos

Evita que o comportamento inesperado de um componente externo do fluxo testado influencie no seu resultado



---

**Less is more**

---

---

# Obrigada!

mail: [contato@paulagrangoiro.com](mailto:contato@paulagrangoiro.com)

site: [www.paulagrangoiro.com.br](http://www.paulagrangoiro.com.br)

github: [@pgrangoiro](https://github.com/pgrangoiro)

twitter: [@paulagrangoiro](https://twitter.com/paulagrangoiro)

face: [fb.me/paula.grangoiro](https://fb.me/paula.grangoiro)

