



**ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT**

<http://www.eternalis-software.com/images/tdd.jpg>

TESTES E DEPURAÇÃO

Testes: Para quê?



O código funciona?

- ☐ “Funciona porque sim!”
- ☐ “Não funciona.”
- ☐ “Funciona às vezes.”
- ☐ “Deve funcionar quase tudo.”
- ☐ Estas respostas são inaceitáveis!

Testes



- Não se pode gerir o que não se conhece.
- Responder à questão implica conhecer o comportamento do programa.
 - O que funciona?
 - Como funciona?
 - Que problemas apresenta?
 - ...

Testes: Como?



- Comparar resultados obtidos com resultados esperados.
- Testar correção: casos dentro do domínio, tanto normais, como extremos!
- Testar robustez: casos fora do domínio.

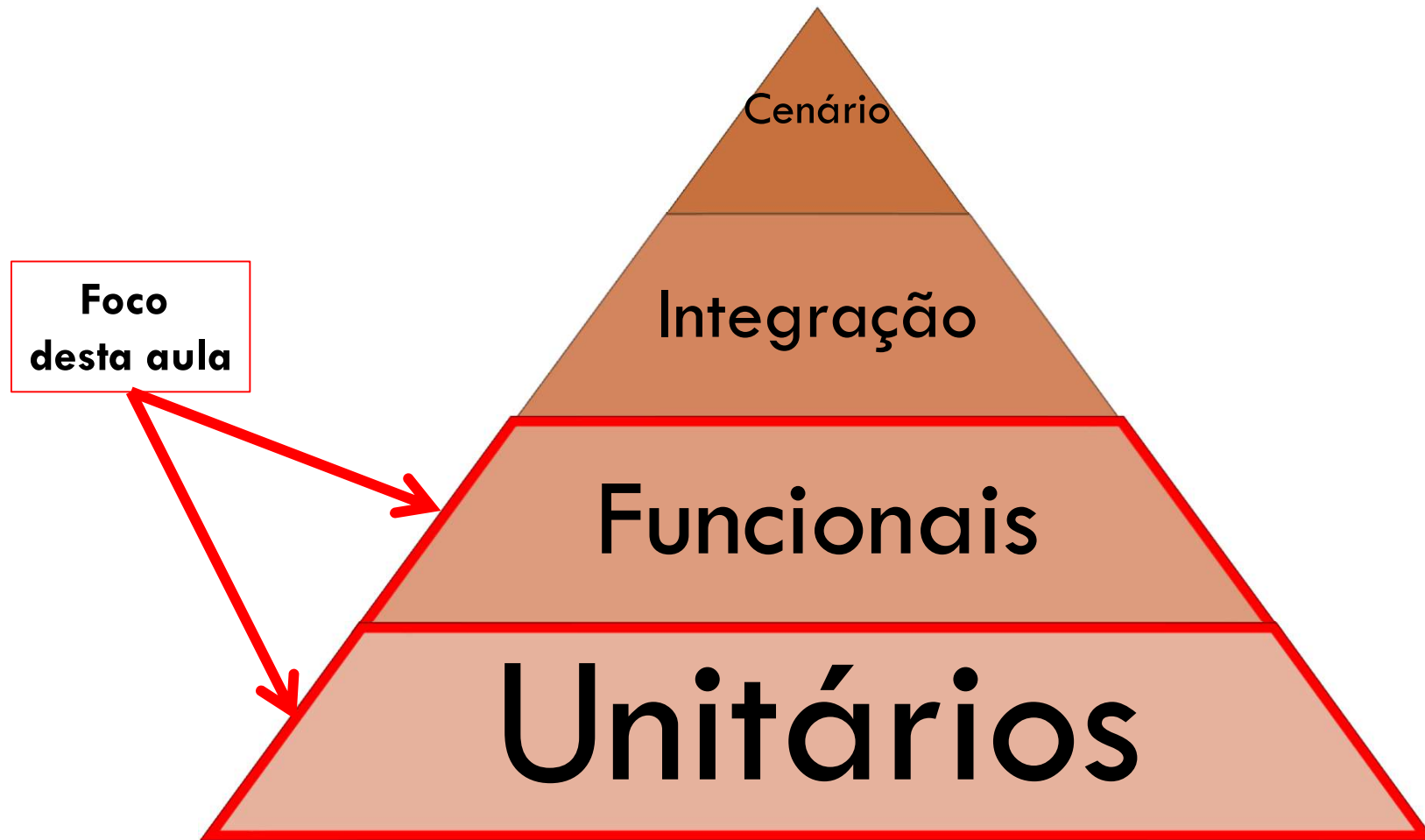
Testes: Interpretação

- Programa correto \Rightarrow todos os testes satisfeitos!
- Isso significa:
 - ▣ Um teste falha \Rightarrow programa incorreto!
 - ▣ Todos os testes passam **não implica** um programa correto! (Só se fossem exaustivos.)

Hierarquia de Testes



Hierarquia de Testes



Testes Unitários



- Focam-se no teste de uma unidade
- Uma unidade realiza 1 função
 - ▣ Peça de código
 - ▣ Função ou método
 - ▣ Classe pequena
- Testes feitos pelo próprio programador

Testes Funcionais



- Focam-se no teste de uma funcionalidade
- Uma funcionalidade envolve vários algoritmos
 - ▣ Pode envolver várias classes ou funções
 - ▣ Resultado externo de um módulo ou programa
- Testes feitos pelo programador ou equipa

Testes



□ Integração

- ▣ Funcionamento coerente entre módulos diferentes
- ▣ Funcionalidades compostas
- ▣ P. ex: Testar integração mapa em página

□ Cenário

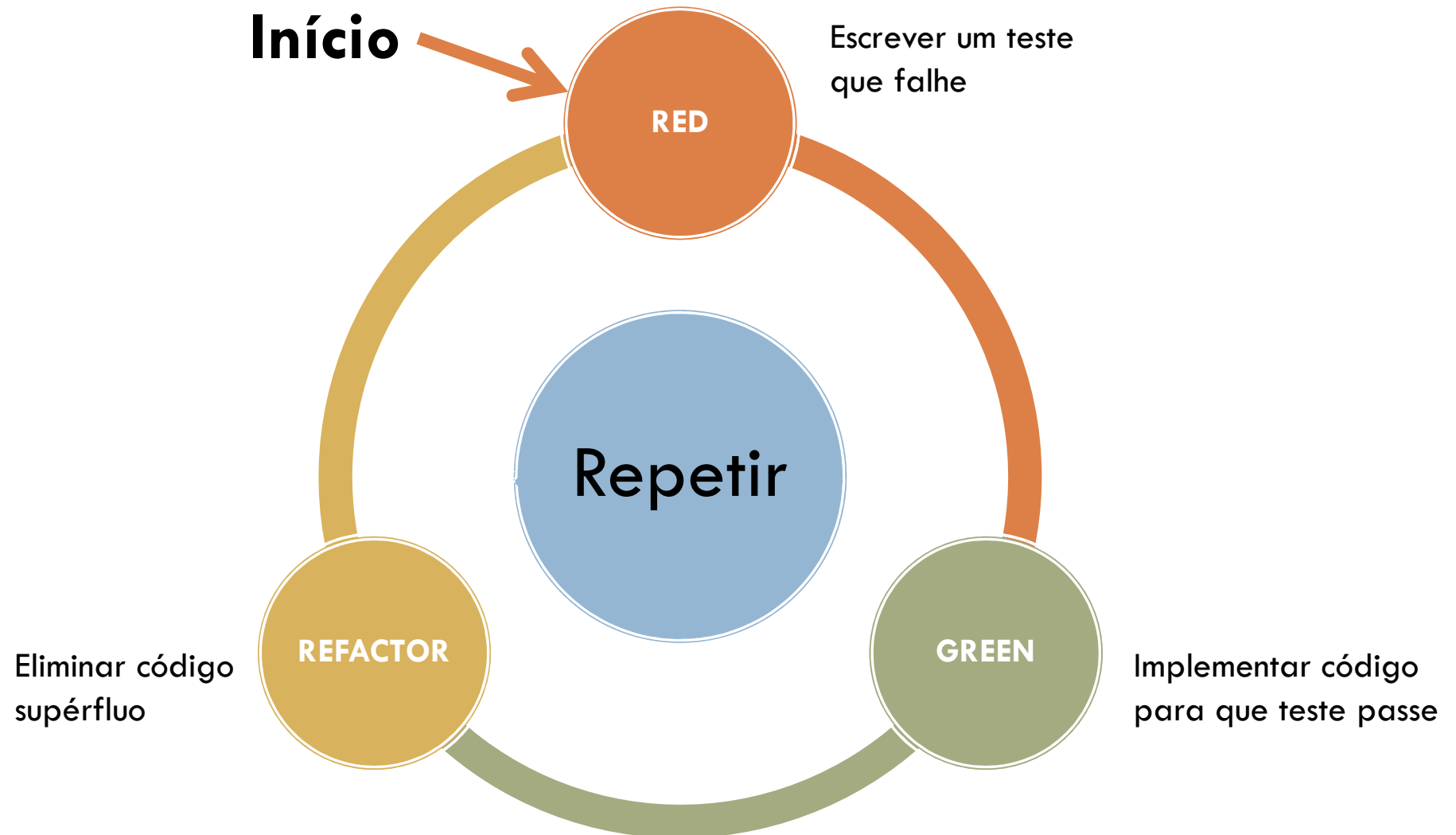
- ▣ Concretizar corretamente um cenário inteiro
- ▣ P. Ex: Página permite localizar rede de lojas

Test Driven Development



- Metodologia muito comum focada em testes
- Desenvolvimento **inicia-se com especificação dos testes**
 - ▣ Não com desenvolvimento do código!
- Testes podem representar 60-80% do custo de desenvolvimento!
- Sabe-se sempre o que funciona, ou não.

Ciclo TDD



Exemplo: Factorial

- Cliente: Calcular factorial na consola
- Objetivo: Programa que dado um número inteiro positivo, calcule o factorial e o imprima no monitor.

```
$ python3 factorial.py 10  
3628800  
$
```

Metodologia TDD



□ Definir testes unitários que falhem

- ▣ `factorial(-1) == "undefined"`
- ▣ `factorial(0) == 1`
- ▣ `factorial(5) == 120`
- ▣ `factorial(10) == 3628800`

□ Definir testes funcionais

- ▣ `python3 factorial.py` deve mostrar ajuda
- ▣ `python3 factorial.py foobar` deve mostrar ajuda
- ▣ `python3 factorial.py 5` deve imprimir 120

RED: factorial(-1) == undefined

- pytest auxilia realização de testes (Python)
 - ▣ Poderiam ser feitos testes com funções próprias

factorial.py

```
def factorial(x):  
    return x
```

test_unit_factorial.py

```
import pytest  
from factorial import factorial #importar unidade (função)  
  
def test_negativos():  
    assert factorial(-1) == "undefined"
```

RED: factorial(-1) == undefined

```
$ pytest test_unit_factorial.py

===== test session starts =====
platform linux -- Python 3.12.3, pytest-7.4.4, pluggy-1.4.0
collected 1 items

test_unit_factorial.py F

===== FAILURES =====
_____ test_negativos _____

    def test_negativos():
>         assert factorial(-1) == "undefined"
E         assert -1 == 'undefined'
E         + where -1 = factorial(-1)

test_unit_factorial.py:5: AssertionError
===== 1 failed in 0.03 seconds =====
```


Todos os Testes Unitários

```
import pytest
from factorial import factorial

def test_negativos():
    assert factorial(-1) == "undefined"

def test_zero():
    assert factorial(0) == 1

def test_valor_pequeno():
    assert factorial(5) == 120




def test_valor_grande():
    assert factorial(10) == 3628800
```

```
===== test session starts =====
platform linux -- Python 3.12.3, pytest-7.4.4, pluggy-1.4.0
collected 4 items

test_unit_factorial.py ....

===== 4 passed in 0.08 seconds =====
```

Implementação

```
def factorial(x):  
    if x < 0:  Test_negativo  
        return "undefined"  
  
    if x == 0:  Test_zero  
        return 1  
  
    res = 1  
  
    while x > 0:  Test_valor_pequeno  
        Test_valor_grande  
        res = res * x  
        x = x - 1  
  
    return res
```

Testes funcionais

□ Neste caso verifica-se a execução

- ▣ comparação da saída (stdout)
- ▣ comparação do código de execução

test_func_factorial.py

```
import pytest
from subprocess import Popen
from subprocess import PIPE

def test_no_args():
    proc = Popen("python3 factorial.py", stdout=PIPE, shell=True)
    assert proc.wait() == 1 #Check Return Code
    assert proc.stdout.read().decode("utf-8") == "Usage: python3 factorial.py positive
number\n"
```

Implementação

```
import sys

def factorial(x):
    ...

def main(argv):
    if(len(argv) != 2):
        print ("Usage: python3 %s positive number" % (argv[0]))
        sys.exit(1)

    if not argv[1].isdigit():
        print ("Usage: python3 %s positive number" % (argv[0]))
        sys.exit(2)

    print(factorial(int(argv[1])))
    sys.exit(0)

if __name__ == "__main__":
    main(sys.argv)
```

Todos os Testes Funcionais - 1

```
import pytest
from subprocess import Popen
from subprocess import PIPE

def test_no_args ():
    proc = Popen ("python3 factorial.py", stdout=PIPE, shell=True)
    assert proc.wait() == 1 #Check Return Code
    assert proc.stdout.read ().decode ('utf-8') == "Usage: python3
factorial.py positive number\n"

def test_invalid_args ():
    proc = Popen ("python3 factorial.py n", stdout=PIPE, shell=True)
    assert proc.wait () == 2 #Check Return Code
    assert proc.stdout.read ().decode ('utf-8') == "Usage: python3
factorial.py positive number\n"

    proc = Popen ("python3 factorial.py -5", stdout=PIPE, shell=True)
    assert proc.wait () == 2 #Check Return Code
    assert proc.stdout.read ().decode ('utf-8') == "Usage: python3
factorial.py positive number\n"
```

Todos os Testes Funcionais -2

```
def test_valid_args ():
    proc = Popen ("python3 factorial.py 0", stdout=PIPE, shell=True)
    assert proc.wait () == 0 #Check Return Code
    assert proc.stdout.read ().decode ('utf-8') == "1\n"

    proc = Popen ("python3 factorial.py 10", stdout=PIPE, shell=True)
    assert proc.wait () == 0 #Check Return Code
    assert proc.stdout.read ().decode ('utf-8') == "3628800\n"
```

```
$ pytest test_func_factorial.py
```

```
===== test session starts =====
platform linux -- Python 3.12.3, pytest-7.4.4, pluggy-1.4.0
collected 3 items

test_func_factorial.py ...

===== 3 passed in 0.21 seconds =====
```

Depuração



- Foi detetado um comportamento incorreto
- Como se deteta o erro específico?
 - ▣ Revisão do código
 - ▣ Execução interativa com depuração
- Depurar: Remover impurezas, sujidade ou imperfeições (em software: bugs)

Depuração



- Integrado num IDE: Eclipse, PyCharm
- Na execução: gdb, ipdb (python)
- Funcionalidades usuais
 - ▣ Interromper programa em qualquer ponto
 - ▣ Inspeccionar memória (variáveis)
 - ▣ Interceptar propagação de erros
 - ▣ Executar passo a passo

Depuração: Conceitos



- Breakpoint: Pontos de pausa do código
 - ▣ Possível verificar variáveis/memória naquele momento
- Step: Executa uma linha
 - ▣ Step-Over: não entra nas funções dessa linha
 - ▣ Step-Into: entra nas funções dessa linha

Python: ipdb


Iniciar Depuração

```
$ python3 -m ipdb factorial.py 10
> /tmp/fact/factorial.py(2)<module>()
----> 2 import sys
ipdb>
```

Criar *Breakpoint* na função factorial(x) e resumir a execução

```
...
ipdb> break factorial
Breakpoint 1 at factorial.py:4
ipdb> continue
> factorial.py(4)factorial()
1      4 def factorial (x)
----> 5      if x < 0: return
```

Execução suspensa
no início da função



Python: ipdb

Ver código atual


Breakpoint 1

Instrução atual

```
ipdb> list
      1  # encoding='utf-8'
      2  import sys
      3
1      4  def factorial(x):
---->     5      if x < 0: return
"undefined"
      6
      7      if x == 0: return 1
      8
      9      res = 1
     10      while x > 0:
     11          res = res * x
```

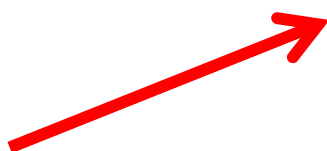
Python: ipdb

Inspeccionar
o valor de x



```
ipdb> p x
10
```

Executar uma linha





```
ipdb> next
> factorial.py(7)factorial()
----> 7      if x == 0: return 1
```

Modificar o valor de x

```
ipdb> next
> factorial.py(9)factorial()
----> 9      res = 1
```

Inspeccionar
o valor de x



```
ipdb> x = 3
ipdb> p x
3
```

Python: PyCharm

