
Testes unitários como ferramentas de design de código

Caipyra - 26 de Junho de 2016

Paula Grangeiro

28 anos

Bacharel em Sistemas de Informação

Duque de Caxias

Gateira

Python & Arquitetura de Software



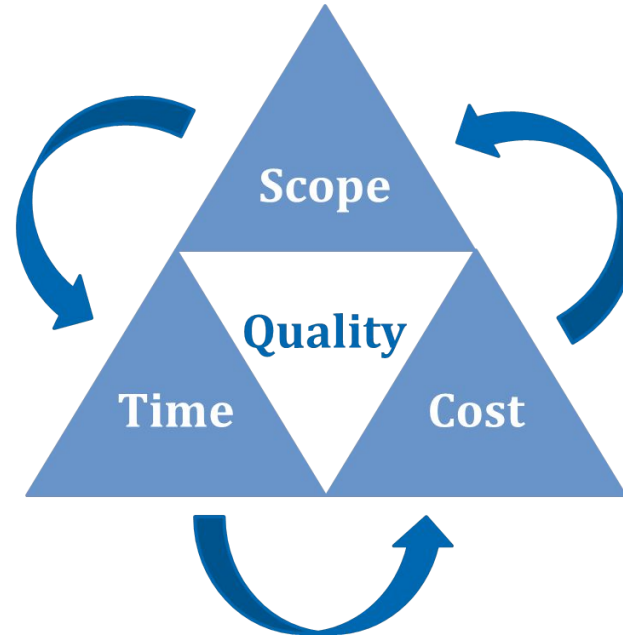


elogroup▶

Onde me encontrar...

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

Problemas do mundo real



Problemas do mundo real



```
832
833 [] def get_report_overall(self, filter_args):
834     from collections import Counter
835
836     objEleitor = Eleitor.objects.filter(**filter_args)
837
838     graph_data = {}
839     graph_data['temp_count'] = objEleitor.count()
840
841     graph_data['sexo'] = Counter(objEleitor.values_list('sexo', flat=True))
842     graph_data['sexo'] = {
843         'null': (
844             float(graph_data['sexo'][None]) / graph_data['temp_count']
845         ) * 100,
846         'F': (
847             float(graph_data['sexo']['F']) / graph_data['temp_count']
848         ) * 100,
849         'M': (
850             float(graph_data['sexo']['M']) / graph_data['temp_count']
851         ) * 100,
852     }
853
854     temp_escolaridade = Counter(
855         objEleitor.exclude(
856             escolaridade__isnull=True
857         ).values_list('escolaridade', flat=True)
858     )
859     count_escolaridade = len(
860         Eleitor.objects.filter(
861             **filter_args
862         ).values('escolaridade').exclude(escolaridade__isnull=True)
863     )
864
865     if count_escolaridade != 0:
866         graph_data['escolaridade'] = {
867 +----- 11 linhas: 1: (float(temp_escolaridade[1]) / count_escolaridade) * 100,-----
878         }
879     else:
880 +---- 14 linhas: graph_data['escolaridade'] = {-----
894
895     graph_data['aprova_o_governo'] = Counter(
896         objEleitor.values_list('aprova_o_governo', flat=True)
```

custom_modelmanager.py

833:3[Syntax: line:1 (26)]

4 custom_modelmanager.py|145 col 13 error| continuation line missing indentation or outdented [E122]

[Lista de locais] :SyntasticCheck flake8 (python)

4,1

12%

```
894  
895     graph_data['aprova_o_governo'] = Counter(
896         objEleitor.values_list('aprova_o_governo', flat=True)
897     )
898
899     graph_data['aprova_o_governo'] = {
900 +---- 35 linhas: 2: (-----
901         }
902
903     temp_graph_data = Counter(objEleitor.values_list('idade', flat=True))
904
905     idades = dict(
906 +---- 9 linhas: dict.fromkeys(-----
907         )
908
909     for item in range(15, 25):
910         idades['idade16_24'] += temp_graph_data[item]
911
912     for item in range(25, 35):
913         idades['idade25_34'] += temp_graph_data[item]
914
915     for item in range(35, 45):
916         idades['idade35_44'] += temp_graph_data[item]
917
918     for item in range(45, 60):
919         idades['idade45_59'] += temp_graph_data[item]
920
921     for item in range(60, 100):
922         idades['idade60'] += temp_graph_data[item]
923
924     count_idade = sum(idades.values())
925
926     graph_data['idade'] = {
927 +---- 6 linhas: 0: (float(idades['idade16_24']) / count_idade) * 100,-----
928         }
929
930     temp_prioridade = Counter(objEleitor.values_list(
931         'qual_a_sua_prioridade', flat=True)
932     )
933
934     prioridade = dict(dict.fromkeys(
935         ['agua', 'asfalto', 'educacao', 'emprego', 'saude', 'seguranca'], 0
```

custom_modelmanager.py

4 custom_modelmanager.py|145 col 13 error| continuation line missing indentation or outdented [E122]

[Lista de locais] :SyntasticCheck flake8 (python)

894:0[Syntax: line:1 (26)]

4,1

12%


```
983     ))
984
985     prioridade['agua'] = (temp_prioridade[u'ÁGUA'] + temp_prioridade[u'AGUA'] + temp_prioridade[u'Água'] + temp_prioridade[u'água'] + temp_prioridade[u'agua'])
986     prioridade['asfalto'] = (temp_prioridade[u'ASFALTO'] + temp_prioridade[u'Asfalto'] + temp_prioridade[u'asfalto'])
987     prioridade['educacao'] = (temp_prioridade[u'EDUCAÇÃO'] + temp_prioridade[u'EDUCACAO'] + temp_prioridade[u'Educação'] + temp_prioridade[u'Educacao'] + temp
prioridade[u'educação'] + temp_prioridade[u'educacao'])
988     prioridade['emprego'] = (temp_prioridade[u'EMPREGO'] + temp_prioridade[u'Emprego'] + temp_prioridade[u'emprego'])
989     prioridade['saude'] = (temp_prioridade[u'SAUDE'] + temp_prioridade[u'SAUDE'] + temp_prioridade[u'Saúde'] + temp_prioridade[u'Saude'] + temp_prioridade[u'sa
úde'] + temp_prioridade[u'saude'])
990     prioridade['seguranca'] = (temp_prioridade[u'SEGURANÇA'] + temp_prioridade[u'SEGURANCA'] + temp_prioridade[u'Segurança'] + temp_prioridade[u'Seguranca'] +
temp_prioridade[u'segurança'] + temp_prioridade[u'seguranca'])
991
992     count_prioridade = sum(prioridade.values())
993
994     graph_data['prioridade'] = {
995 +---- 7 linhas: 0: (float(prioridade['agua']) / count_prioridade) * 100,-----
1002     }
1003
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
```

custom_modelmanager.py

983:3[Syntax: line:1 (26)]

4 custom_modelmanager.py[145 col 13 error] continuation line missing indentation or outdented [E122]

[Lista de locais] :SyntasticCheck flake8 (python)

4,1

12%

```
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
```

custom_modelmanager.py

1004:0[Syntax: line:1 (26)]

4 custom_modelmanager.py[145 col 13 error] continuation line missing indentation or outdented [E122]

[Lista de locais] :SyntasticCheck flake8 (python)

4,1

12%


+ 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
```

custom_modelmanager.py

1004:0[Syntax: line:1 (26)]

4 custom_modelmanager.py[145 col 13 error] continuation line missing indentation or outdented [E122]

[Lista de locais] :SyntasticCheck flake8 (python)

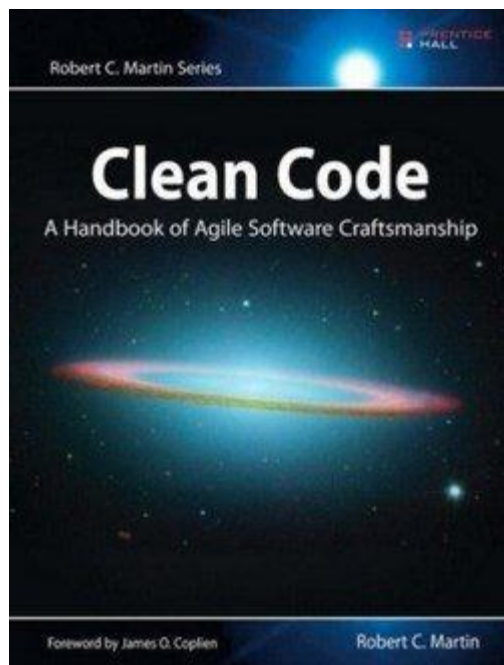
4,1

12%

```
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).count())
1035     email_uninformed = len(objEleitor.values_list('email', flat=True).filter(email_isnull=True).count())
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).filter(resumida=False)
1043
1044     renda = dict(dict.fromkeys(['renda_mais_2034', 'renda_mais_2035', 'renda_mais_2036', 'renda_mais_2037', 'renda_mais_2038', 'renda_mais_2039', 'renda_mais_2040', 'renda_mais_2041', 'renda_mais_2042', 'renda_mais_2043', 'renda_mais_2044', 'renda_mais_2045', 'renda_mais_2046', 'renda_mais_2047', 'renda_mais_2048', 'renda_mais_2049', 'renda_mais_2050', 'renda_mais_2051', 'renda_mais_2052', 'renda_mais_2053', 'renda_mais_2054', 'renda_mais_2055', 'renda_mais_2056', 'renda_mais_2057', 'renda_mais_2058', 'renda_mais_2059', 'renda_mais_2060', 'renda_mais_2061', 'renda_mais_2062', 'renda_mais_2063', 'renda_mais_2064', 'renda_mais_2065', 'renda_mais_2066', 'renda_mais_2067', 'renda_mais_2068', 'renda_mais_2069', 'renda_mais_2070', 'renda_mais_2071', 'renda_mais_2072', 'renda_mais_2073', 'renda_mais_2074', 'renda_mais_2075', 'renda_mais_2076', 'renda_mais_2077', 'renda_mais_2078', 'renda_mais_2079', 'renda_mais_2080', 'renda_mais_2081', 'renda_mais_2082', 'renda_mais_2083', 'renda_mais_2084', 'renda_mais_2085', 'renda_mais_2086', 'renda_mais_2087', 'renda_mais_2088', 'renda_mais_2089', 'renda_mais_2090', 'renda_mais_2091', 'renda_mais_2092', 'renda_mais_2093', 'renda_mais_2094', 'renda_mais_2095', 'renda_mais_2096', 'renda_mais_2097', 'renda_mais_2098', 'renda_mais_2099', 'renda_mais_2100'], 0))
1045
1046     for item in range(len(objRenda)):
1047 +---- 8 linhas: if objRenda[item] > 1356.00:
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
```

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

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

APPLYING UML AND PATTERNS

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

SECOND EDITION



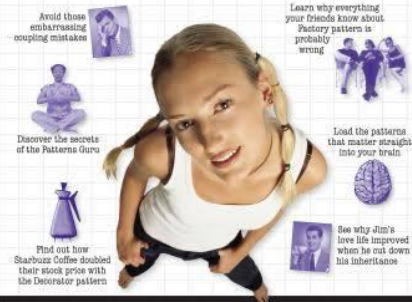
"People often ask me which is the best book to introduce them to the world of OO design. Ever since I came across it, Applying UML and Patterns has been my unreserved choice."

—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

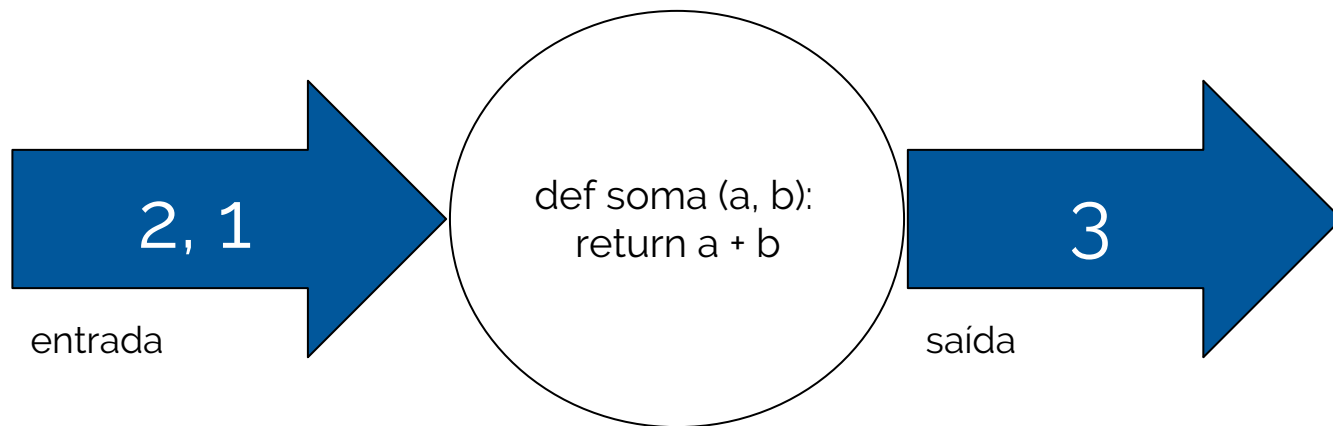
**Design Patterns não são
uma bala de prata**

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

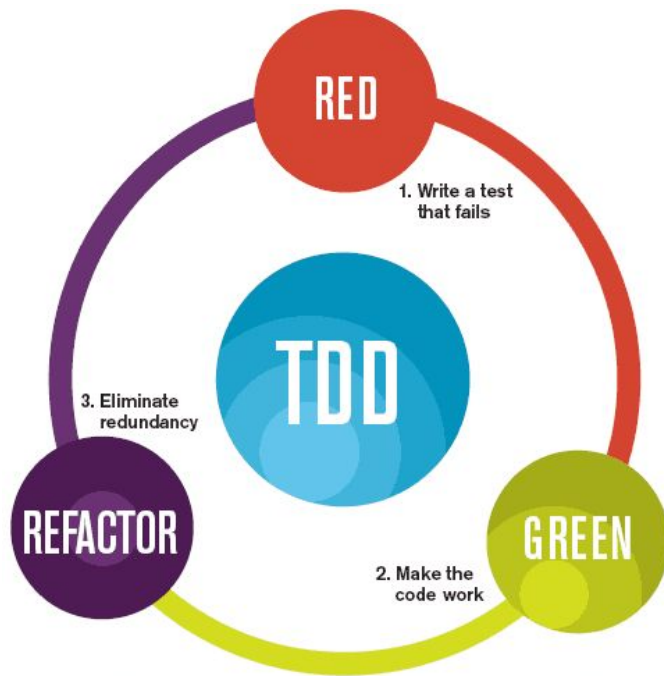
- unittest
 - Py.test
 - Doctest
-

—

Testes unitários como ferramentas de design?

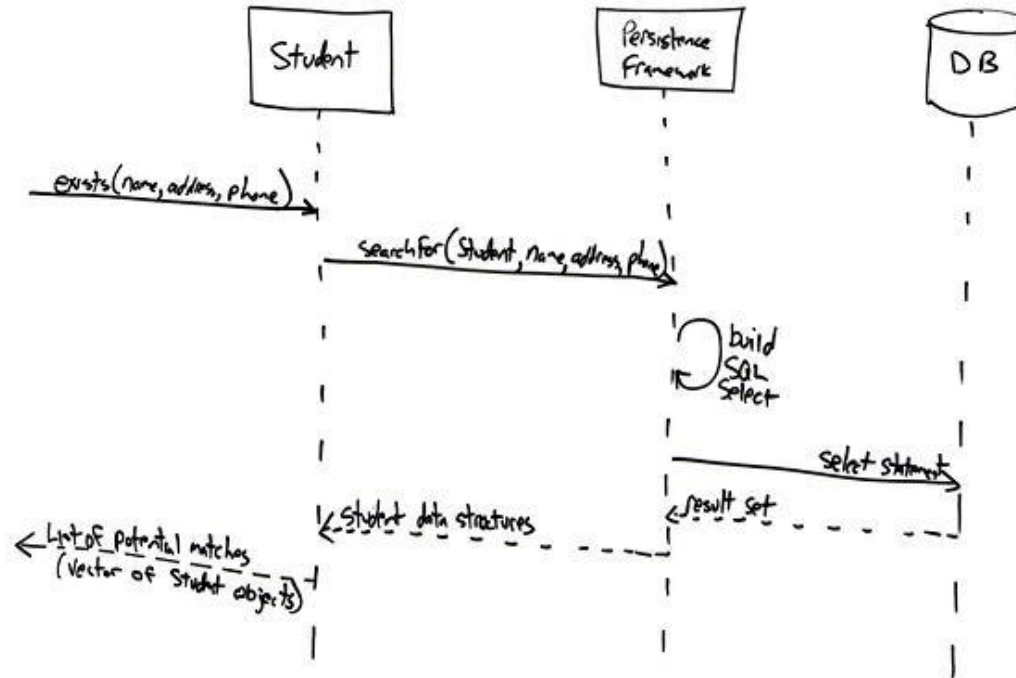
1. Pratique TDD

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

```
>> 1 class CustomTestCase(TestCase):
2     def setUp(self):
3         print('Initializing Graph test case')
>> 4     usuarioTeste = coreModels.Usuario.objects.create(
5         username='teste',
6         nome='nome',
7     )
8
>> 9     campanhaTeste = coreModels.Campaign.objects.create(
10     name='CampanhaTeste',
11     description='Description',
>> 12     start_date=datetime.datetime(2014, 01, 01),
>> 13     end_date=datetime.datetime(2014, 01, 01),
14     coordinator=usuarioTeste,
15     manager=usuarioTeste,
16 )
17
>> 18     perfilTeste = coreModels.Profile.objects.create(
19     name='perfilTeste',
20     description='Description',
21     contact_type=1,
22     goal=100,
23     profile_type=1,
24     target_filter={}
25 )
26
>> 27     taskTeste = coreModels.Task.objects.create(
28     name='Task Teste',
29     goal=100,
30     description='',
31     contact_type=1,
32     task_type=1,
>> 33     start_date=datetime.datetime(2014, 01, 01),
>> 34     end_date=datetime.datetime(2014, 01, 01),
35     operator=usuarioTeste,
36     coordinator=usuarioTeste,
37     campaign=campanhaTeste,
38     profile=perfilTeste,
39     status=1,
40     visible_fields={},
41     fields={},
```

custom_modelmanager_test.py

41:6[Syntax: line:1 (112)]

```
4 custom_modelmanager_test.py|12 col 24 error| undefined name 'datetime' [F821]
5 custom_modelmanager_test.py|13 col 22 error| undefined name 'datetime' [F821]
```

[Lista de locais] :SyntasticCheck flake8 (python)

4,1

2%

```
40     visible_fields={},
41     fields={},
42 )
43 eleitorTesteM = coreModels.Eleitor.objects.create(nome_completo='Eleitor Teste', sexo="M")
44 eleitorTesteF = coreModels.Eleitor.objects.create(nome_completo='Eleitor Teste', sexo="F")
45 eleitorTesteNull = coreModels.Eleitor.objects.create(nome_completo='Eleitor Teste')
46
47 def test_gender_graph_data_two_days_same_distribution_m_and_f(self):
48     #Required data
49     eleitorTesteM = coreModels.Eleitor.objects.filter(sexo="M")[0]
50     eleitorTesteF = coreModels.Eleitor.objects.filter(sexo="F")[0]
51     taskTeste = coreModels.Task.objects.all()[0]
52
53     testeM = coreModels.TaskResult.objects.create(
54         elector = eleitorTesteM,
55         task = taskTeste,
56         done_at = datetime.date(2014, 01, 01),
57         status = 1
58     )
59     #unfortunatly done_at is auto_now_add so we need to do this litle hack
60     testeM.done_at = datetime.date(2014, 01, 01)
61     testeM.save()
62
63     testeF = coreModels.TaskResult.objects.create(
64         elector = eleitorTesteF,
65         task = taskTeste,
66         done_at = datetime.date(2014, 01, 01),
67         status = 1
68     )
69     #unfortunatly done_at is auto_now_add so we need to do this litle hack
70     testeF.done_at = datetime.date(2014, 01, 01)
71     testeF.save()
72
73     start_date = datetime.datetime(2014, 01, 01, 0, 0, 0)
74     end_date = datetime.datetime(2014, 01, 02, 23, 59, 59)
75     lookup_field = 'elector__sexo'
76
77     result = coreViews.generateGraphData(start_date, end_date, lookup_field)
78
79     self.assertEqual(result, [[["2014-01-01", 50.0], ["2014-01-02", 50.0]], [{"2014-01-01", 50.0}, {"2014-01-02", 50.0}], [{"2014-01-01", 0.0}, {"2014-01-02", 0.0}]])
```

custom_modelmanager_test.py

41:6[Syntax: line:1 (112)]

4 custom_modelmanager_test.py|12 col 24 error| undefined name 'datetime' [F821]

5 custom_modelmanager_test.py|13 col 22 error| undefined name 'datetime' [F821]

[Lista de locais] :SyntasticCheck flake8 (python)

4,1

2%

```
79 self.assertEqual(result, [[["2014-01-01", 50.0], ["2014-01-02", 50.0]], [{"2014-01-01", 50.0}, {"2014-01-02", 50.0}], [{"2014-01-01", 0.0}, {"2014-01-02", 0.0}]]
80
81 def test_gender_graph_two_days_same_distribution_m_and_f_and_null(self):
82     #Required data
83     eleitorTesteM = coreModels.Eleitor.objects.filter(sexo="M")[0]
84     eleitorTesteF = coreModels.Eleitor.objects.filter(sexo="F")[0]
85     eleitorTesteNull = coreModels.Eleitor.objects.filter(sexo__isnull=True)[0]
86
87     taskTeste = coreModels.Task.objects.all()[0]
88
89     #Elector day 1 Masc
90     test1m = coreModels.TaskResult.objects.create(
91         eleitor = eleitorTesteM,
92         task = taskTeste,
93         done_at = datetime.date(2014, 01, 01),
94         status = 1
95     )
96     #unfortunatly done at is auto_now_add so we need to do this little hack
97     test1m.done_at = datetime.date(2014, 01, 01)
98     test1m.save()
99
100     #Elector day 1 Fem
101     test1f = coreModels.TaskResult.objects.create(
102         eleitor = eleitorTesteF,
103         task = taskTeste,
104         done_at = datetime.date(2014, 01, 01),
105         status = 1
106     )
107     #unfortunatly done at is auto_now_add so we need to do this little hack
108     test1f.done_at = datetime.date(2014, 01, 01)
109     test1f.save()
110
111     #Elector day 1 Null
112     test1Null = coreModels.TaskResult.objects.create(
113         eleitor = eleitorTesteNull,
114         task = taskTeste,
115         done_at = datetime.date(2014, 01, 01),
116         status = 1
117     )
118     #unfortunatly done at is auto_now_add so we need to do this little hack
119     test1Null.done_at = datetime.date(2014, 01, 01)
```

custom_modelmanager_test.py 79:6[Syntax: line:1 (112)]

4 custom_modelmanager_test.py|12 col 24 error| undefined name 'datetime' [F821]

5 custom_modelmanager_test.py|13 col 22 error| undefined name 'datetime' [F821]

[Lista de locais] :SyntasticCheck flake8 (python)

4,1

2%

whitespace after '[' [E201]

```
S> 94         status = 1
95     )
96     #unfortunally done at is auto_now_add so we need to do this litle hack
>> 97     test1m.done_at = datetime.date(2014, 01, 01)
98     test1m.save()
99
100     #Elector day 1 Fem
>> 101     test1f = coreModels.TaskResult.objects.create(
S> 102         elector = eleitorTesteF,
S> 103         task = taskTeste,
S> 104         done_at = datetime.date(2014, 01, 01),
S> 105         status = 1
106     )
107     #unfortunally done at is auto_now_add so we need to do this litle hack
>> 108     test1f.done_at = datetime.date(2014, 01, 01)
109     test1f.save()
110
111     #Elector day 1 Null
>> 112     test1Null = coreModels.TaskResult.objects.create(
S> 113         elector = eleitorTesteNull,
S> 114         task = taskTeste,
S> 115         done_at = datetime.date(2014, 01, 01),
S> 116         status = 1
117     )
118     #unfortunally done at is auto_now_add so we need to do this litle hack
>> 119     test1Null.done_at = datetime.date(2014, 01, 01)
120     test1Null.save()
121
122     start_date = datetime.datetime(2014, 01, 01, 0, 0, 0)
>> 123     end_date = datetime.datetime(2014, 01, 02, 23, 59, 59)
124     lookup_field = 'elector__sexo'
125
>> 126     result = coreViews.generateGraphData(start_date, end_date, lookup_field)
127     print result
S> 128     self.assertEqual(result, [[["2014-01-01", 33.33], ["2014-01-02", 33.33]], [{"2014-01-01", 33.33}, {"2014-01-02", 33.33}]]])
S> 129
```

custom_modelmanager_test.py

109:6[Syntax: line:1 (112)]

4 custom_modelmanager_test.py|12 col 24 error| undefined name 'datetime' [F821]

5 custom_modelmanager_test.py|13 col 22 error| undefined name 'datetime' [F821]

[Lista de locais] :SyntasticCheck flake8 (python)

4,1

2%


```
S> 94         status = 1
95     )
96     #unfortunally done at is auto_now_add so we need to do this litle hack
>> 97     test1m.done_at = datetime.date(2014, 01, 01)
98     test1m.save()
99
100     #Elector day 1 Fem
>> 101     test1f = coreModels.TaskResult.objects.create(
S> 102         elector = eleitorTesteF,
S> 103         task = taskTeste,
S> 104         done_at = datetime.date(2014, 01, 01),
S> 105         status = 1
106     )
107     #unfortunally done at is auto_now_add so we need to do this litle hack
>> 108     test1f.done_at = datetime.date(2014, 01, 01)
109     test1f.save()
110
111     #Elector day 1 Null
>> 112     test1Null = coreModels.TaskResult.objects.create(
S> 113         elector = eleitorTesteNull,
S> 114         task = taskTeste,
S> 115         done_at = datetime.date(2014, 01, 01),
S> 116         status = 1
117     )
118     #unfortunally done at is auto_now_add so we need to do this litle hack
>> 119     test1Null.done_at = datetime.date(2014, 01, 01)
120     test1Null.save()
121
122     start_date = datetime.datetime(2014, 01, 01, 0, 0, 0)
>> 123     end_date = datetime.datetime(2014, 01, 02, 23, 59, 59)
124     lookup_field = 'elector__sexo'
125
>> 126     result = coreViews.generateGraphData(start_date, end_date, lookup_field)
127     print result
S> 128     self.assertEqual(result, [[["2014-01-01", 33.33], ["2014-01-02", 33.33]], [{"2014-01-01", 33.33}, {"2014-01-02", 33.33}]]])
S> 129
```

custom_modelmanager_test.py

109:6[Syntax: line:1 (112)]

4 custom_modelmanager_test.py|12 col 24 error| undefined name 'datetime' [F821]

5 custom_modelmanager_test.py|13 col 22 error| undefined name 'datetime' [F821]

[Lista de locais]:SyntasticCheck flake8 (python)

4,1

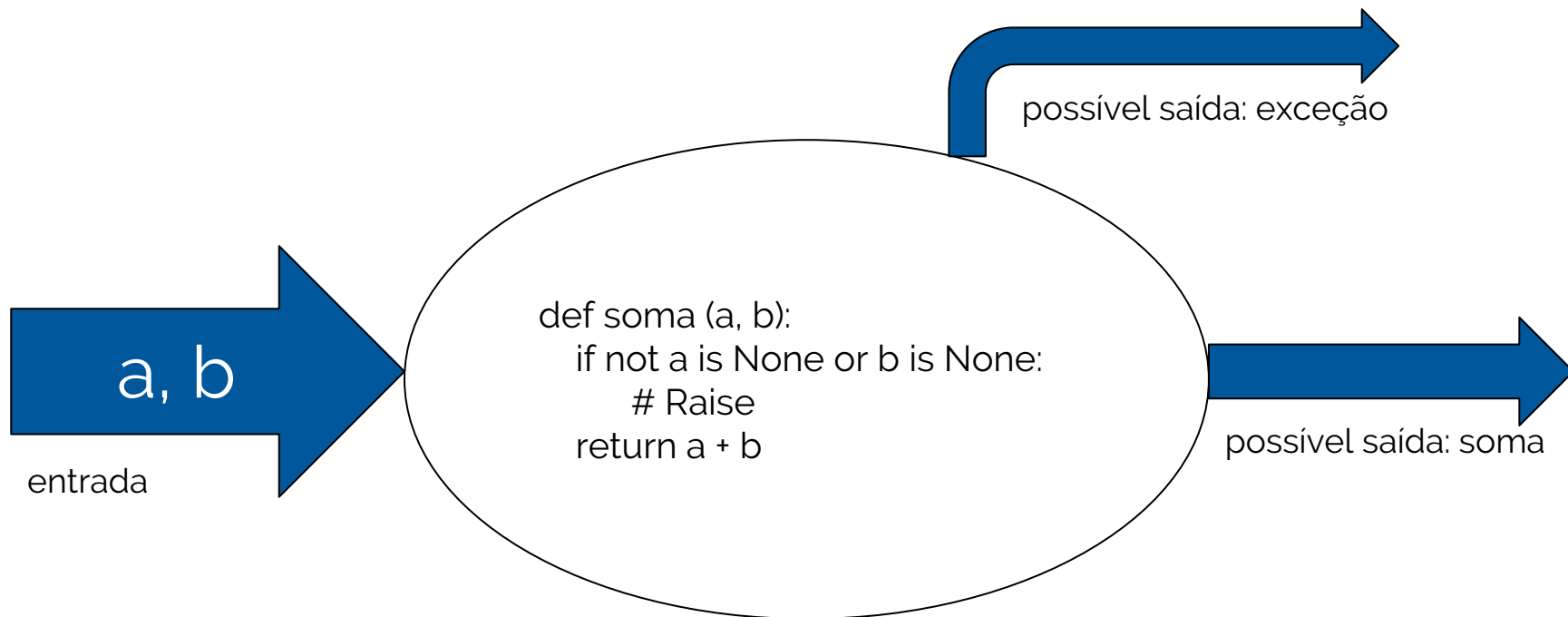
2%

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

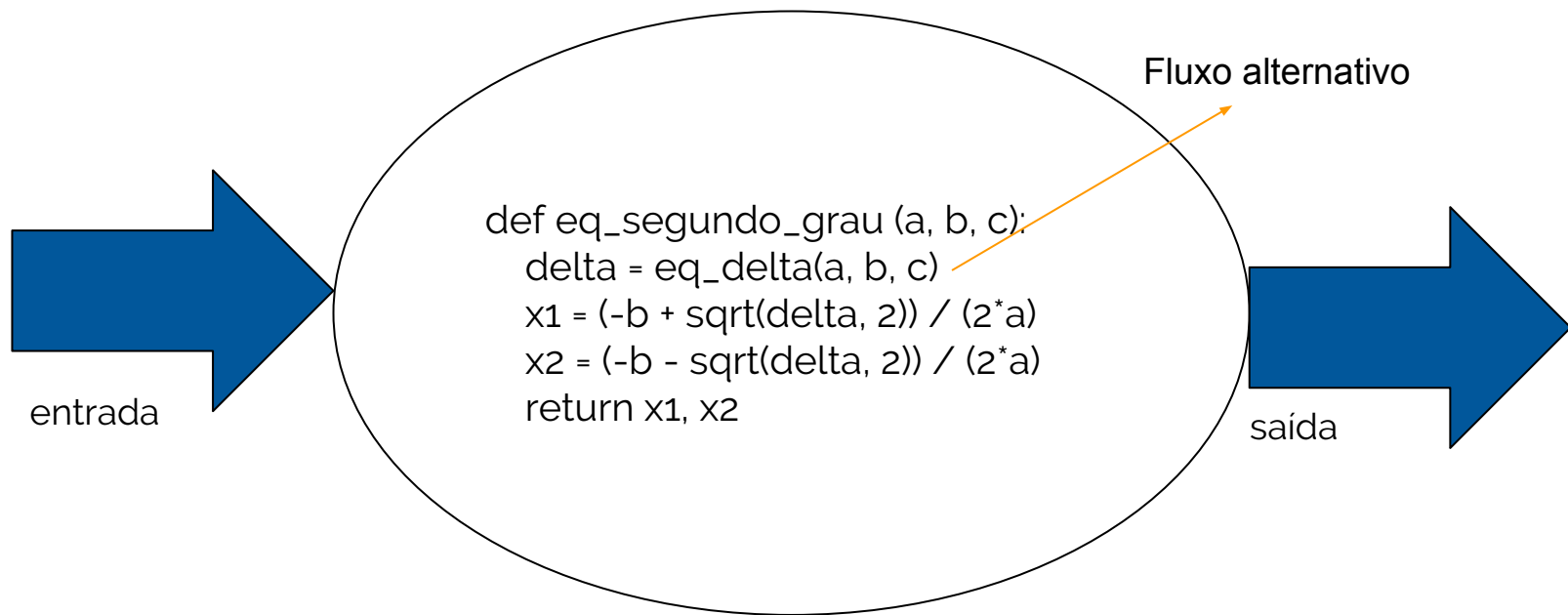
```
from unittest import TestCase
import soma
import SomeCustomException
```

```
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

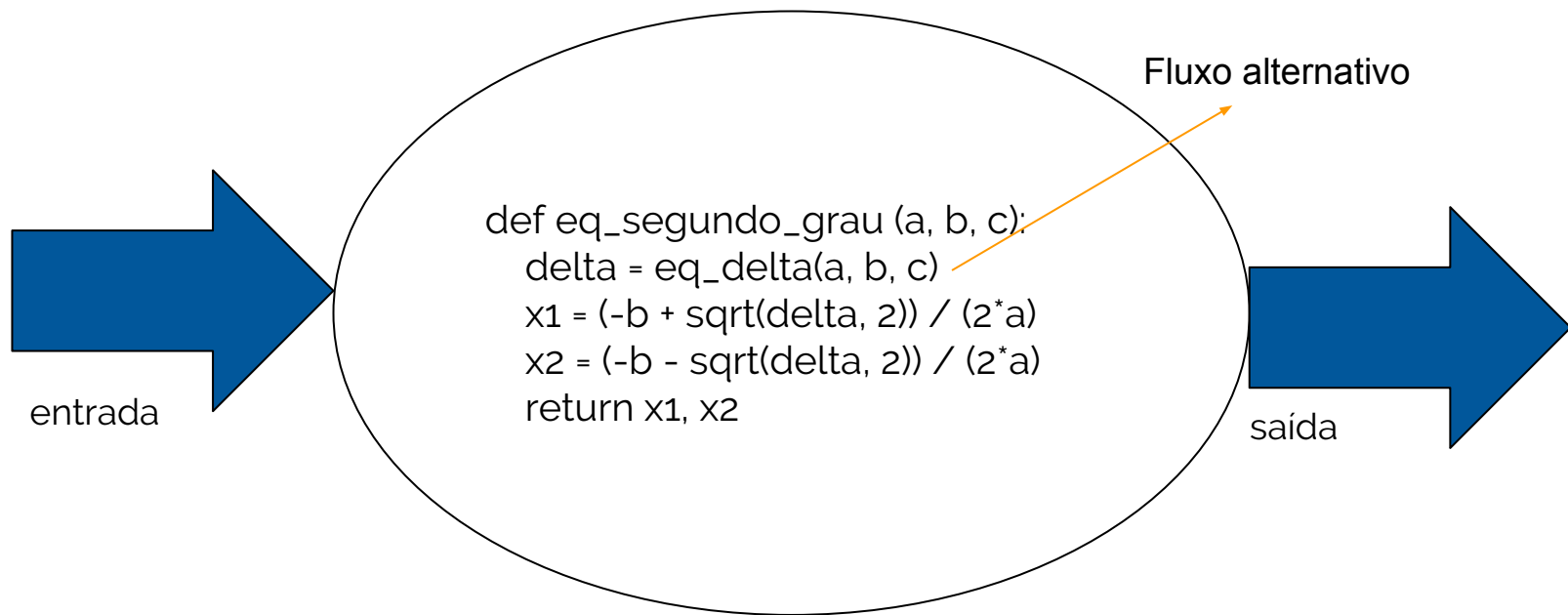


Pensar unitariamente

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

site: www.paulagrangoiro.com.br

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

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

face: fb.me/paula.grangoiro

