Universidade do Minho
Escola de Engenharia

Análise e Teste de Software 2024/25

TESTE DE PROJETOS JAVA

Junho 2025

Gustavo Barros
A100656

Enzo Vieira
A98352

Pedro Ferreira
A97646

Introdução

Neste trabalho, foi nos pedido para realizar a testagem de multiplos projetos recorrendo a diferentes técnicas de teste e diferentes ferramentes, todos os projetos foram realizados em Java.

1. Projetos

Os projetos "Solucao1 e Solucao2" que foram utilizados foram fornecidos pela equipa docente e foram desenvolvidos por diferentes pessoas, apenas os estámos a usar para teste. Ambos os projetos tem como tema a realização de uma aplicação de exercício fisíco.
Decidímos ainda usar o projeto da UC de Programação Orientada a Objeto de um dos membros do grupo, que foi desenvolvido no ano letivo de 2022/2023, logo este incede sobre um diferente tema que era a realização de uma loja online de venda e revenda de vestuário.

2. Maven

Primeiramente decidimos que nos iria simplificar o trabalho se estes projetos fossem transformados em projetos Java Maven, pois era mais fácil para gerir as dependências e os plugins utilizados, bem como controlar os seus comportamentos.
O uso do Maven facilita ainda a execução dos testes, pois com "mvn clean test" é possível executar todos os testes do projeto, "mvn clean verify" para obter as estatísticas geradas pelo JaCoCo e "mvn clean org.pitest:pitest-maven:mutationCoverage" para executar os testes para "Mutant Testing" com o PIT.

3. Escrita de Testes Unitários

A escrita de testes unitários foi apenas feita para as classes dos modelos de cada projeto.

3.1 Testes unitários escritos

Começou-se por escrever testes JUNIT para diferentes classes do projeto, mas sendo este um processo bastante trabalhoso e demorado, cada vez que a complexidade do projeto aumentava, a complexidade de teste do mesmo também aumentava e ainda iria demorar bastante tempo até conseguir cobrir uma grande parte do projeto, deste modo decidimos recorrer ao uso de LLMs para geração de testes e complementação dos escritos anteriormente.

```java
@Test
public void testCalcularPrecoFinal_LisoOuNovo() {
    TShirt tshirt = new TShirt("Liso Novo", "Branca", "Uniqlo", 304, 25.0f, 1.0f, true, 0, "Novo", "FedEx", 20, "ana", "S", "LISO", 0);
    float result = tshirt.calcularPrecoFinal();
    assertEquals(25.0f, result, 0.01f);
}

@Test
public void testCalcularPrecoFinal_NaoLisoENaoNovo() {
    tshirt1.setPadrao("RISCAS");
    tshirt1.setPrecoBase(40.0f);
    tshirt1.setNovo(false);
    float result = tshirt1.calcularPrecoFinal();
    assertEquals(20.0f, result, 0.01f);
}
```

Testes escritos

3.2 Testes unitários gerados com recurso a LLMs

- **Gemini 2.5 Pro**: Os testes gerados pela LLM não vieram em condições de ser usados, para além de bastante complicados de serem lidos, pois o código não é limpo e contém demasiados comentários e documentação(apesar de ser algo bom, eram em tão grande quantidade que não se conseguir perceber o código a ser escrito pois ficava uma total confusão.)
- **ChatGPT 4.1**: Os testes gerados continham bastantes erros, de simples correção, todos os testes exigiram correções, apesar disso o código fornecido era de simples percepção o que facilitava a correção do teste, um dos problemas dos testes gerados era também que optava por criar classes "Mock" no ficheiro dos testes, devido a existirem classes abstratas, em vez de usar as classes que extendiam essa classe abstrata.
- **Claude Sonnet 4**: Definitivamente a LLM que facilitou mais o trabalho, para além de raramente gerar testes com erros, completou os ficheiros com os test cases que faltava, após revisão ainda foi possível detetar testes que não faziam sentido ou não entravam no contexto do modelo, sendo estes apagados.

```java
@Test
public void testConsumoCalorias() {
    BenchPress bp = new BenchPress(
            LocalDateTime.now(),
            LocalTime.of(0, 20, 0),
            freqCardiaca:120,
            repeticoes:100,
            peso:80.0
    );
    double calorias = bp.consumoCalorias(utilizadorMock);
    assertTrue(calorias > 0);
}

@Test
public void testGeraAtividade() {
    double calorias = 200.0;
    Atividade atividade = benchPressDefault.geraAtividade(utilizadorMock, calorias);
    assertTrue(atividade instanceof BenchPress);
    assertEquals(0, atividade.getFreqCardiaca());
    assertTrue(((BenchPress)atividade).getPeso() > 0);
}
```

Testes gerados pelo ChatGPT 4.1

```java
@Test
@DisplayName("Test constructTrainingPlanByObjectives with invalid maxActivitiesPerDay")
void testConstructTrainingPlanByObjectivesInvalidMaxActivitiesPerDay() {
    TrainingPlan trainingPlan = trainingPlanManager.createTrainingPlan(userCode, startDate);

    IllegalArgumentException exception = assertThrows(IllegalArgumentException.class,
        () -> trainingPlanManager.constructTrainingPlanByObjectives(
            trainingPlan, index:1.5f, hardActivities:false, -1, maxDifferentActivities…3, 5, 1000));
    assertEquals("Invalid input.", exception.getMessage());

    exception = assertThrows(IllegalArgumentException.class,
        () -> trainingPlanManager.constructTrainingPlanByObjectives(
            trainingPlan, index:1.5f, hardActivities:false, maxActivitiesPerDay:4, ma…3, 5, 1000));
    assertEquals("Invalid input.", exception.getMessage());
}

@Test
@DisplayName("Test constructTrainingPlanByObjectives with invalid maxDifferentActivities")
void testConstructTrainingPlanByObjectivesInvalidMaxDifferentActivities() {
    TrainingPlan trainingPlan = trainingPlanManager.createTrainingPlan(userCode, startDate);

    IllegalArgumentException exception = assertThrows(IllegalArgumentException.class,
        () -> trainingPlanManager.constructTrainingPlanByObjectives(
            trainingPlan, index:1.5f, hardActivities:false, maxActivitiesPerDay:2, -1, we…5, 1000));
    assertEquals("Invalid input.", exception.getMessage());
}
```

Testes gerados pelo Claude Sonnet 4

4. Geração de Testes com EvoSuite

O EvoSuite apesar de ser uma ferramenta que foi concebida para facilitar o trabalho de um tester, está com baixo suporte para as outras ferramentas e versões atuais do java, não foi possível gerar testes para a "Solução 2" pois necessitaria de um refactor devido ao código ter sido escrito após a versão Java11 que mudou alguma sintaxe, sendo assim apenas foram usados testes escritos para este projeto.

Para os restantos projetos foi possível gerar bastantes testes, garantindo assim uma grande variedade principalmente para as classes que não eram modelos, pois essas apenas são testadas com o Evosuite devido a uma grande complexidade de escrever testes para as mesmas.

Apesar da ferramente estar muito desatualizada, é uma grande ajuda para iniciar a testagem de um programa ou complementar trabalho já feito. Contudo apenas poderá ser usada em projetos com sintaxe compatível com o Java8 (versão em que é funcional).

```java
@Test(timeout = 4000)
public void test01()  throws Throwable  {
    GestorDesportivo gestorDesportivo0 = new GestorDesportivo();
    Clock clock0 = MockClock.systemUTC();
    LocalDate localDate0 = MockLocalDate.now(clock0);
    Month month0 = Month.OCTOBER;
    LocalDate localDate1 = MockLocalDate.of(1245, month0, 6);
    // Undeclared exception!
    try {
      gestorDesportivo0.kmsPercorridos(codUtilizador:1245, localDate0, localDate1);
      fail("Expecting exception: NullPointerException");

    } catch(NullPointerException e) {
       //
       // no message in exception (getMessage() returned null)
       //
       verifyException("org.example.GestorDesportivo", e);
    }
}
```

Teste Evosuite

5. Análise de Cobertura

5.1 JaCoCo

A análise de cobertura foi feito com recurso ao JaCoCo que foi utilizada para medir a cobertura dos projetos "Solucao1" e "Solucao2" mostrou que apesar de o EvoSuite gerar uma grande gama de testes e juntamente com os testes escritos e anteriormente existentes, continuou a existir uma grande variedade de linhas e caminhos no projeto que não estavam cobertos, desta forma teriam que ser escritos mais testes para cobrir esses caminhos, sendo que o Evosuite não foi capaz de os cobrir.

Solucao1

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.example		45%		34%	394	663	914	1,728	162	362	4	24
Total	4,203 of 7,719	45%	374 of 569	34%	394	663	914	1,728	162	362	4	24

Solucao1 JaCoCo

Solucao2

Solucao2

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
MakeItFit.views		0%		0%	138	138	693	693	95	95	3	3
MakeItFit		0%		0%	118	118	263	263	103	103	4	4
MakeItFit.users		60%		43%	48	90	72	180	21	52	0	3
MakeItFit.queries		71%		61%	41	80	42	157	11	28	1	7
MakeItFit.trainingPlan		77%		65%	27	77	38	174	0	29	0	2
MakeItFit.menu		0%		0%	21	21	46	46	11	11	2	2
MakeItFit.utils		75%		55%	12	38	12	52	6	28	0	3
MakeItFit.activities.types		84%		55%	23	50	15	80	7	32	0	4
MakeItFit.activities		88%		40%	9	28	10	71	4	23	0	2
MakeItFit.exceptions		38%		n/a	4	6	4	6	4	6	1	3
MakeItFit.activities.implementation		97%		96%	2	50	3	92	1	36	0	4
MakeItFit.users.types		100%		100%	0	22	0	44	0	19	0	3
MakeItFit.time		100%		100%	0	5	0	10	0	4	0	1
Total	5.204 of 8.092	35%	275 of 501	45%	443	723	1.198	1.868	263	466	11	41

Solucao2 JaCoCo

No projeto "ProjetoPOO" não foi possível usar o JaCoCo, apesar de este ter suporte para testes gerados em EvoSuite, testes que usem JUNIT4 ou JUNIT5, neste projeto, os mesmos não eram detetados, eram apresentados e detetados pelo Maven ao usar o terminal, mas no relatório do JaCoCo não eram contabilizados. Após consulta da documentação do Evosuite foram tentadas as soluções apresentadas na mesma, que não tiveram efeito apenas neste projeto.

ProjetoPOO

ProjetoPOO

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Vintage.mvc		0%		0%	334	334	1,175	1,175	128	128	4	4
Vintage.qql		0%		0%	272	272	610	610	175	175	13	13
Total	7,447 of 7,447	0%	566 of 566	0%	606	606	1,785	1,785	303	303	17	17

ProjetoPOO JaCoCo

5.2 PIT

Também é possível observar a cobertura do projeto com a ferramenta de "Mutant Testing" PIT, esta apresenta a Line Coverage de linhas em que foram aplicados mutantes num projeto, que no geral, foi sempre superior ao que o JaCoCo apresentou sendo que este apresenta uma cobertura geral. Desta ferramenta também existe informação de cobertura de mutantes para o projeto "ProjetoPOO" sendo assim possível obter algum conhecimento do estado da cobertura do mesmo. Esta cobertura é só relacionada com a cobertura aos mutantes, contudo é sempre útil também para ser comparada com a cobertura geral, ajudando um bocado em situações em que o JaCoCo não funcione, como foi o caso do "ProjetoPOO"

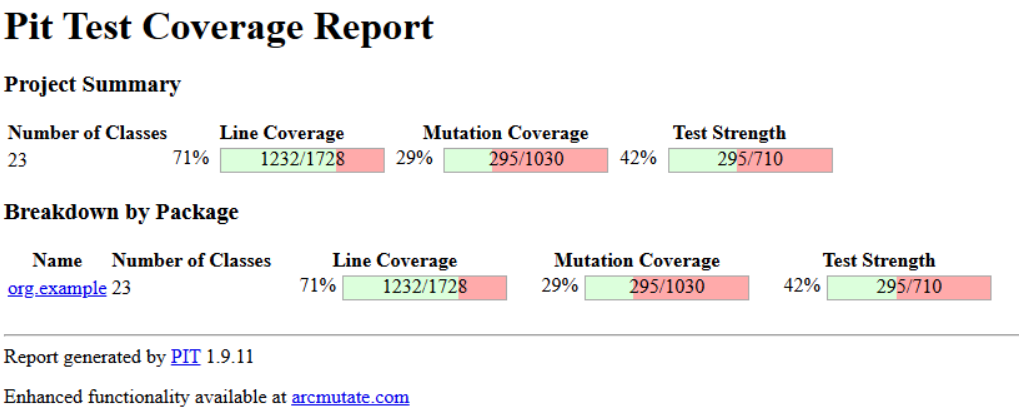
6. Testes por Mutação com PIT

Para testes com mutantes utilizámos a ferramenta do java PIT, como o Evosuite, esta também necessita de uma versão própria para funcionar, que é o caso do Java17 ou menor, foi possível utilizá-la em todos os projetos, pois já suporta a sintaxe pós Java11. Um dos grandes problemas da mesma é não suportar os testes gerados pelo Evosuite, após bastante tempo perdido a perceber a causa, que é o facto do Evosuite usar uma flag "separateClassLoader = true" no "@RunWith e @EvoRunnerParameters" que teria de ser desativada para o PIT conseguir usar os testes gerados.

```
@RunWith(EvoRunner.class) @EvoRunnerParameters(mockJVMNonDeterminism = true, useVFS = true, useVNET = true, resetStaticState = true, separateClassLoader = true, useJEE = true)
public class Controller_ESTest extends Controller_ESTest_scaffolding {
```

Evosuite Flag

Com esta flag ativa, o EvoSuite executa as classes de teste em um carregador de classes separado (separated class loader), e por isso as mutações não têm efeito nos mesmos. A solução é então desativar a flag, mas sendo que estas soluções já são antigas, provavelmente não funcionam atualmente, pois ao desativar esta flag, alguns testes gerados pelo Evosuite falhavam ao serem usados no Maven ou no PIT, uma outra solução seria a utilização de um pluguin partilhado no GitHub , contudo não o chegámos a testar nem investigar o sourcecode, mas é apontado como uma possível solução para o problema.
Desta forma para todos os projetos apenas foram considerados os testes escritos e os que já existiam nos mesmos.



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
23	71%	1232/1728	29%	295/1030	42%	295/710

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
org.example	23	71%	1232/1728	29%	295/1030	42%	295/710

Report generated by PIT 1.9.11

Enhanced functionality available at arcmutate.com

Solucao1 PIT

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
37	27%	502/1858	22%	222/1000	79%	222/282

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
MakeItFit	4	0%	0/263	0%	0/135	100%	0/0
MakeItFit.activities	2	62%	44/71	36%	8/22	57%	8/14
MakeItFit.activities.implementation	4	70%	64/92	55%	41/74	71%	41/58
MakeItFit.activities.types	4	65%	52/80	50%	20/40	67%	20/30
MakeItFit.menu	2	0%	0/46	0%	0/35	100%	0/0
MakeItFit.queries	7	73%	115/157	60%	49/82	88%	49/56
MakeItFit.time	1	70%	7/10	75%	3/4	75%	3/4
MakeItFit.trainingPlan	2	28%	48/174	21%	19/90	90%	19/21
MakeItFit.users	2	59%	104/176	52%	44/85	79%	44/56
MakeItFit.users.types	3	73%	32/44	53%	8/15	80%	8/10
MakeItFit.utils	3	69%	36/52	65%	30/46	91%	30/33
MakeItFit.views	3	0%	0/693	0%	0/372	100%	0/0

Solucao2 PIT

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
12	32%	558/1764	23%	181/789	89%	181/203

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
Vintage.mvc	4	0%	0/1175	0%	0/573	100%	0/0
Vintage.qql	8	95%	558/589	84%	181/216	89%	181/203

Report generated by PIT 1.16.2

Enhanced functionality available at arcmutate.com

ProjetoPOO PIT

7. Geração de Testes com Hypothesis

No projeto 2, experimentou-se o recurso ao módulo Hypothesis, em Python, para gerar automaticamente dados para classes de teste parametrizado, partindo dos testes unitários previamente elaborados. Decidiu-se tomar a classe User e as suas subordinadas Amateur, Occasional e Professional como alvos por assumirem uma complexidade de arquitetura suficientemente interessante.

```python
# user_generator.py
from hypothesis import strategies as st
...
@st.composite
def arbitrary_user(draw):

    return {
        'name'           : draw(name_strategy),
        'address'        : draw(address_strategy),
        'phone'          : draw(phone_strategy),
        'email'          : draw(email_strategy),
        'age'            : draw(age_strategy),
        'gender'         : draw(gender_strategy),
        'weight'         : draw(weight_strategy),
        'height'         : draw(height_strategy),
        'bpm'            : draw(bpm_strategy),
        'level'          : draw(level_strategy),
        'frequency'      : draw(frequency_strategy),
    }

def gen(amount=1):
    return [arbitrary_user().example() for _ in range(amount)]
```

```python
def amateur_test(user_list): return f"""
package TestModels;
import java.io.ByteArrayInputStream;
...
@RunWith(Parameterized.class)
public class AmateurTest {{

    private Amateur amateur;
    private final String name;
    private final int age;
    private final Gender gender;
    private final float weight;
    private final int height;
    private final int goalWeight;
    private final int activityLevel;
    private final String address;
    private final String phone;
    private final String email;

    @Parameters
    public static Collection<Object[]> data() {{
        return Arrays.asList(new Object[][] {{
            {formatted_users(user_list)}
        }});
    }}
...
"""

if __name__ == "__main__":
  print(amateur_test(gen(20)))
```

8. Jqwik

Decidimos explorar a ferramenta Java que tem a mesma função que o Hypothesis (Python) e o QuickCheck (Haskell), no caso, o "Jqwik" que é usado para Property-Based Test. Usámos o mesmo para testar a classe "Artigo" usando uma instância Sapatilha(extende Artigo), para verificar os seus comportamentos por meio de geração aleatória de dados. Foram feitos geradores, no caso utilizamos métodos como randomSapatilha() para gerar instâncias aleatórias e usámos também filtros (usedSapatilha(), premiumSapatilha()) para restringir cenários. Desta forma demonstrando como o Jqwik automatiza a verificação de invariantes e de edge cases, assim não é necessário a construção de exemplos manuais.

```
19
20      @Provide
21      Arbitrary<Sapatilha> randomSapatilha() {
22          return Arbitraries.randomValue(random -> {
23              Sapatilha s = createDefaultSapatilha();
24              s.setNome(randomString(random, 10));
25              s.setDescricao(randomString(random, 20));
26              s.setMarca(randomString(random, 10));
27              s.setCodigo(random.nextInt(1000));
28              s.setPrecoBase(random.nextFloat() * 1000);
29              s.setCorrecaoPreco(random.nextFloat() * 100);
30              s.setNovo(random.nextBoolean());
31              s.setNumDonos(random.nextInt(10));
32              s.setCondicao(random.nextBoolean() ? "New" : "Used");
33              s.setTransportadora(randomString(random, 5));
34              s.setStock(random.nextInt(100));
35              s.setVendedor(randomString(random, 8));
36              s.setTamanho(30 + random.nextInt(21)); // 30-50
37              s.setAtacadores(random.nextBoolean());
38              s.setCor(randomString(random, 5));
39              s.setAnoLancamento(2000 + random.nextInt(24)); // 2000-2023
40              s.setPremium(random.nextBoolean());
41              return s;
42          });
43      }
44
45      @Property
46      void constructorShouldSetAllFields(@ForAll("randomSapatilha") Sapatilha sapatilha) {
47          assertNotNull(sapatilha.getNome());
48          assertNotNull(sapatilha.getDescricao());
49          assertNotNull(sapatilha.getMarca());
50          assertTrue(sapatilha.getCodigo() >= 0);
51          assertTrue(sapatilha.getPrecoBase() >= 0);
52          assertTrue(sapatilha.getNumDonos() >= 0);
53          assertNotNull(sapatilha.getCondicao());
54          assertNotNull(sapatilha.getTransportadora());
55          assertTrue(sapatilha.getStock() >= 0);
56          assertNotNull(sapatilha.getVendedor());
57          assertTrue(sapatilha.getTamanho() >= 30 && sapatilha.getTamanho() <= 50);
58          assertNotNull(sapatilha.getCor());
59          assertTrue(sapatilha.getAnoLancamento() >= 2000 && sapatilha.getAnoLancamento() <= 2023);
60      }
61
62      @Property
63      void copyConstructorShouldCreateEqualObject(@ForAll("randomSapatilha") Sapatilha original) {
64          Sapatilha copy = new Sapatilha(original);
65          assertEquals(original, copy);
66          assertNotSame(original, copy);
67      }
68
69      @Provide
70      Arbitrary<Sapatilha> usedSapatilha() {
71          return randomSapatilha()
72                  .filter(s -> !s.isPremium() && s.getNumDonos() > 0);
73      }
```

Classe Artigojqwik

Conclusões e trabalho futuro

Concluindo, considerámos que o objetivo de implementar os diferentes tipos de situações de teste e utilização das diferentes ferramentas foi concebido, contudo com pouca cobertura, e ainda apesar de as ferramentas não apresentarem suporte para as versões mais recentes do Java e algumas não serem compatíveis entre si.
Ainda considerámos que o uso de LLMs para já não substituirá o uso de ferramentas como o Evosuite ou ferramentas como o Hypothesis para criar geradores, pois estes são mais eficientes que a geração de testes por parte de um LLM, pois estes modelos necessitam de muita maior supervisão do que as ferramentas quando as usámos.
Para trabalho futuro, pretendemos continuar a extender a cobertura dos testes ao projeto, testar o plugin para tentar solucionar a incompatibilidade do Evosuite com o PIT, ou em alternativa tentar arranjar novas ferramentas, expandir a quantidade de testes que usam o Jqwik e criar mais geradores de testes JUNIT com o Hypothesis.