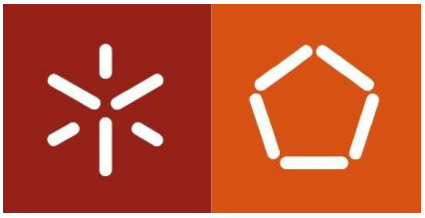

Universidade do Minho

Departamento de Informática

Análise e Teste de Software
Trabalho Prático

16 de maio de 2023

Afonso Magalhães (A95250)

Rui Armada (A90468)

Diana Teixeira (A97516)

Conteúdo

Capítulo 1

Introdução

Para a realização deste trabalho prático, e após a realização de uma breve análise dos projetos realizados na disciplina de Programação Orientada a Objetos, optou-se por se tomar partido de duas soluções distintas, o **SmartHouse_1** e o **SmartHouse_2**. Ambos relativos ao ano letivo 2021/2022, cujo o objetivo era para desenvolver uma aplicação capaz de gerir e monitorizar *SmartDevices* de uma dada habitação.

A inclusão de múltiplos elementos de monitorização e de execução neste trabalho levou à existência de muitas classes e funções que proporcionam a necessidade de diversos testes para assegurar o bom funcionamento do programa.

A maior razão pela decisão de utilizar duas soluções diferentes da mesma aplicação foi para comparar os resultados obtidos entre um trabalho que, incialmente, permanece inalterado e outra solução na qual foram aplicados os conhecimentos de *Unit Testing* e outras ferramentas que foram lecionadas na presente cadeira. Outra razão foi pelo facto de que algumas ferramentas utilizam versões de java que não são compativeis com uma dada solução daí a utilização de duas soluções distintas ajudou na realização de todos os requisitos mínimos, e alguns extras, propostos pelo enunciado.

Capítulo 2

Testes Unitários

2.1 SmartHouse_1

Nesta primeira fase do projeto, decidiu-se efetuar alguns testes unitários mais generalistas de forma a conseguir-se verificar o bom funcionamento de algumas funções vitais para o modelo lógio da aplicação que foi resolvida. Com isto foram determinados os pontos vitais da aplicação onde os primeiros testes seríam desenvolvidos de forma a garantir que todos os requisitos de uma aplicação deste género devería cumprir. Assim sendo, as classes que foram indentificadas como vitais são as seguintes:

- Casa
- Fornecedor
- SmartDevice
- EstadoPrograma

2.1.1 Casa

A classe `Casa` possui métodos que se responsabilizam pela alteração dos estados dos *SmartDevices*, quer seja apenas um de cada vez, todos os aparelhos num determinado quarto ou até mesmo todos de uma vez. Para tal, foram implementaos métodos de teste para as funções `setAllDevicesStateRoom`, `setAllDevicesState` e `setDeviceState`, de forma a garantir que os estados dos *SmartDevices* são alterados corretamente. Adicionalmente, foi testado o método `consumoDispositivos`, que se responsabiliza pelo cálculo do consumo total de todos os dispositivos presentes numa habitação.

Para além disso, cada Casa incluí uma lista de faturas emitidas que detalham o período de consumo referente á mesma, o consumo energético total nesse período e o custo monetário resultante. Com o intuito de garantir que os cálculos de consumo em função do tempo estão a ser efetuados de forma eficiente, foi construido um teste unitário de forma a verificar o bom funcionamento do método `consumoPeriodo` de duas formas diferentes:

1. É especificado o período específico que se pretende obter o consumo
2. É considerado o instante de emissão da fatura até ao tempo percorrido `LocalDate`

Finalmente, optou-se ainda por testar dois métodos presentes na classe `Casa`. Ambos estes métodos são utilizados no projeto original para facilitar a manipulação dos diferentes quartos em cada casa, algo que não é necessariamente realista, mas útil para testar *setups* distintos. Estes métodos são `mudaDeviceRoom` e `juntaRooms`, que alteram um dispositivo de uma divisão para outra e agregam divisões distintas para que sejam identificadas pelo mesmo nome, respetivamente.

2.1.2 Fornecedor

A principal função dos fornecedores envolve a criação e emissão de faturas. Tal já foi mencionado, estas faturas contêm um conjunto de informações que são essenciais no funcionamento do programa. Dito isto, foi testado o método `criarFatura` presente na classe fornecedor, com o intuito de garantir que os parâmetros definidos nas faturas são registados corretamente. Estes parâmetros incluem:

- Data de Início
- Data de Fim
- Fornecedor
- NIF do cliente
- Consumo

Adicionalmente, verificou-se o bom funcionamento do método `faturacao` que, em função do conjunto de casas para as quais um fornecedor fornece energia, calcula o valor monetário faturado por um dado fornecedor num dado período.

2.1.3 SmartDevice

A principal particularidade da classe *SmartDevice* baseia-se na necessidade de calcular o consumo dos três possíveis aparelhos de forma distinta. Para se testar estes consumos, foi implementado um método que testa a função `Consumption` para as *SmartBulbs*, as *SmartCameras* e os *SmartSpeakers* de forma a verificar se estes *Devices* são adicionados ao programa correta para garantir que cada habitação possui a informação necessária para idenficar cada dispositivo constituinte de uma determinada divisão.

2.1.4 EstadoPrograma

Para concluir este primeiro ponto, foram implementados métodos de teste para a classe *EstadoPrograma*. Esta classe responsabiliza-se pela realização das estatísticas sobre o estado do programa, estas sendo:

- Qual é a casa que mais gastou naquele período?
- Qual o comercializador com maior volume de faturação?
- Listar as faturas emitidas por um comercializador.
- Ordenar os maiores consumidores de energia durante um período a determinar.

Portanto, os métodos que foram testados são:

- `getCasaMaisGastadora`
- `maiorConsumidorPeriodo`
- `getFornecedorMaiorFaturacao`
- `podiumDeviceMaisUsado`

O processo utilizado para testar estes métodos é semelhante em maior parte dos casos. Numa fase inicial, começou-se por inicializar alguns elementos que vão influenciar o resultado da estatística pretendida. Estes elementos podem variar entre *Faturas*, *Fornecedores*, *SmartDevices* e *Casas*. Em seguida, colocaram-se estes elementos nos seus locais respetivos, adicionando *SmartDevices* em casas, designando fornecedores a casas, aplicando o custo às faturas, etc. Assim que a preparação está concluída, foi iniciada uma nova instância da classe *EstadoPrograma*, garantindo que o resultado da estatística, aplicada aos elementos que foram adicionados, é igual ao esperado.

2.2 SmartHouse_2

No que toca à segunda solução que foi utilizada, é importante referir que já se encontravam alguns testes unitários desenvolvidos do ano letivo anterior, alterando-se apenas alguns pedaços de código de forma a que a versão de java utilizada pela solução corresponde-se com o `SDK java 1.8`.

Assim, e tal como referido anteriormente, nota-se que este projeto foi apenas utilizado para utilizar o *plug-in* do IntelIJ, **Evosuite**, o qual não foi possivel utilizar na solução no *SmartHouse_1*, mesmo após vários esforços e tentativas com outras versões de evosuite e de java pelo que a unica forma para se utilizar o *evosuite* seria dar um *refactor* relativamente extensivo a um código mais complexo do que comparado com a segunda solução utilizada. Nota-se que também foi utilizado o **PITest**, servindo este projeto de forma de comparação e análise dos resultados das ferramentas de análise e teste de software, no que toca à cobertura e a robustez dos testes, utilizadas durante este projeto.

Capítulo 3

Geração Automática de Testes com EvoSuite

Com o intuito de garantir o bom funcionamento do `EvoSuite`, tal como foi mencionado anteriormente, optou-se por utilizar a solução *SmartHouse_2*, devido a problemas levantados pela elavada comprexidade da primeira solução que geravam conflitos de versões do *Java* que não possibilitaram a utilização da ferramenta desejada.

Assim sendo, de forma a testar a eficiência desta ferramenta, foram analisados os valores de *Coverage* dos testes que estavam presentes originalmente, em comparação com os valores obtidos após a geração automática de testes efetuada pelo `EvoSuite`. Como foi referido anteriormente, os testes, presentes neta solução da aplicação, foram realizados aquando o desenvolvimento da mesma. Assim sendo, e de forma a possibilitar uma melhor compreenção do funcionamento e resultados produzidos pelo `EvoSuite`, estes testes permaneceram inalterados durante toda a duração deste projeto prático.

Com isto, foram construidos alguns gráficos de forma a comparar os resultados obtidos pela utilização do `EvoSuite` e pela análize dos testes originais com o recurso à ferramenta `PITest`. Estes gráficos podem ser encontrados na secção de Anexos, Figura 6.1.

Portanto, ao comparar os testes já presentes na solução, com aqueles gerados pelo `EvoSuite`, foi observado que os testes automatizados apresentam uma cobertura mais abrangente do código e possuem uma capacidade de deteção de falhas muito mais eficiente e precisa. Tendo isso em conta, pode-se afirmar que o `EvoSuite` é capaz de identificar casos de teste que podem não ter sido considerados inicialmente ou que poderíam ter sido negligenciados durante o desenvolvimento da aplicação. A quantidade e qualidade considerável dos testes gerados é razão suficiente para justificar a utilização da ferramento pelo que foram obtidas percentagens elevadas de *Line Coverage* e *Mutation Coverage* que comprovam a existencia de testes capazes de abrangir o código de tal forma a evitar alterações maliciosas de código e assegurar o bom funcionamento do serviço que foi desenvolvido.

Em suma, com base nos resultados obtidos, pode-se concluir que o `EvoSuite` é uma ferramenta altamente eficiente, tendo em conta que a sua abordagem automatizada não só é capaz de aprimorar a qualidade dos testes, como também pode poupar tempo e esforço aos desenvolvedores de *Software*.

Capítulo 4

Mutação de Código

4.1 Resultados iniciais

Com o intuito de avaliar a cobertura dos testes unitários mencionados anteriormente, optou-se por utilizar o plugin de **PITest** do InteliJ nas quatro classes vitais identificadas na secção 2.1.

Após a análise dos resultados, é possível aferir que, em maior parte das classes, o *Test Strength* obtido é bastante positivo. Isto, por si só, induz a conclusão de que os testes unitários inicialmente adicionados são eficazes em validar o comportamento expectável do programa. Porém, e tal como se pode observar na Tabela 4.1, a *Line Coverage* e a *Mutation Coverage* possuem percentagens demasiado baixas, o que, por sua vez, indica que o programa não se encontra capaz de detetar pequenos *bugs* e alterações ao código original.

	Line Coverage	Mutation Coverage	Test Strength
Casa	50%	45%	79%
Estado programa	27%	17%	93%
SmartDevice	24%	8%	50%
Fornecedor	33%	32%	92%

Figura 4.1: Tabela com os resultados inciais do PITest.

4.2 Alterações efetuadas

Face a isto, e com o objetivo de se melhorar esta vertente, chegou-se à conclusão de que sería necessário implementar uma nova vaga de testes, mais focados em deteção de mutantes, visto que, dessa forma, podería-se melhorar a eficiência geral dos testes presentes na solução.

Optando então por realizar uma nova vaga de testes unitários mais concisos e minuciosos, a fim de garantir que todos os métodos das quatro classes mencionadas na secção 2.1 possuem alguma cobertura, foram criados mais testes direcionados para os, por exemplo, os getters, setters, equals, toStrings, construtores, etc.

Eis algumas estratégias que foram adotadas para tornar os nossos testes mais abrangentes:

- Múltiplos testes para um só método, onde cada um tem em consideração possíveis comportamentos diferentes do método em questão. Por exemplo, considerar o comportamento de uma das estatísticas da classe EstadoPrograma onde múltiplos elementos possuem o mesmo "Maior valor"
- Testar métodos abstratos considerando cada classe que o implementa
- Testar todos os tipos de construtores, sejam parametrizados e consequentemente clonados
- Garantir que todos os métodos possuam pelo menos um teste que avalie o seu comportamento, certificando que mutantes inseridos em métodos simples são facilmente detetáveis

4.3 Resultados finais

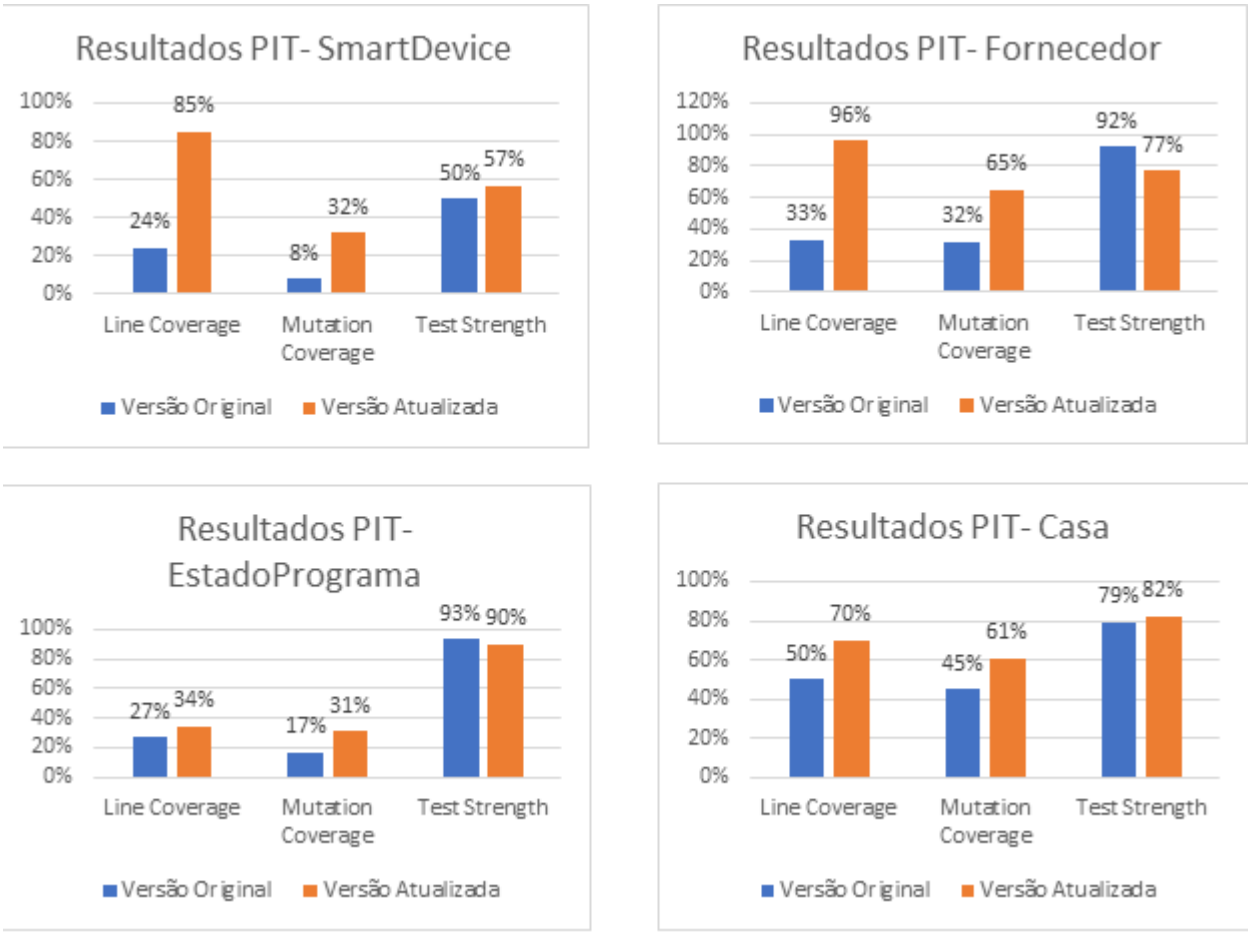

Figura 4.2: Comparação dos resultados relativos ao PIT.

Analisando os resultados obtidos pela execução do *plugin* do `PITest`, é possível ver que, após a introdução de uma nova vaga de testes unitários, algo que melhorou significativamente em todos os casos foi a *Line Coverage* e a *Mutation Coverage*. Este resultado é a consequência direta do aumento do número de testes em cada classe, visto que a maior abrangência da nova cobertura leva a que as diferenças subtis introduzidas pelos mutantes gerados no *PIT* sejam detetadas com mais frequência. Apesar disso, é possível detetar que no parâmetro *Test Strength*, os valores tendem a diminuir, estagnar ou subir de forma insignificante. Acreditamos que esta flutuação dos valores se deve à valorização da quantidade de testes ao invés da qualidade dos mesmos pela nossa parte. De notar que na classe `EstadoPrograma` não se desenvolveram tantos testes como o pretendido, pois, como se trata de uma classe que junta diversas outras classes para testar os seus métodos, garantir uma abrangência elevada sería um processo demorado.

Capítulo 5

Geradores de Logs

De acordo com o que é pedido no enunciado, e com base no conhecimento obtido das aulas, foi utilizado, quer o *quickcheck*, quer o sistema *hypotesis*, para gerar ficheiros de *log* que sejam, não só semelhantes aos fornecidos para POO, como também possíveis de utilizar agora nas soluções utilizadas. Ou seja, foi necessário criar um ficheiro em *haskell* e outro em *python* capazes de gerar um ficheiro com a seguinte sintaxe:

```
Fornecedor:EDP Comercial
Fornecedor:Galp Energia
Fornecedor:Iberdrola
Fornecedor:Endesa
Fornecedor:Gold Energy
Fornecedor:Coopernico
Fornecedor:Enat
Fornecedor:YIce
Fornecedor:MEO Energia
Fornecedor:Muon
Fornecedor:Luzboa
Fornecedor:Energia Simples
Fornecedor:SU Electricidade
Fornecedor:EDA
Casa:Vicente de Carvalho Castro,365597405,Iberdrola
Divisao:Sala de Jantar
SmartBulb:Warm,11,4.57
SmartBulb:Neutral,12,4.73
Divisao:Sala de Jantar 1
SmartBulb:Neutral,7,9.35
SmartCamera:(1280x720),65,3.84
SmartSpeaker:2,Radio Renascenca,LG,5.54
SmartBulb:Neutral,5,6.36
```

5.1 Quickcheck

Assim, e para tornar a sua realização mais fácil, optou-se por implementar as seguintes características:

- Uma função *rounTo* que arredonda os *doubles* para um certo número de casas pedidas, neste caso, tal como se pode ver em cima, duas;
- Uma função para gerar os nifs, que gera um número entre 100000000 e 999999999, não assumindo a possibilidade de que este pode ser duplicado, visto a probabilidade de isto acontecer ser muito reduzida e não se pretender gerar muitas linhas;
- Para além da função anterior, foram também implementadas duas outras funções, uma que gera um inteiro após receber o intervalo a qual este tem que pertencer, e outra que gera um *double*;
- Com isto, e após serem ponderados diferentes cursos de ação, decidiu-se implementar o código dos Fornecedores, Nomes, Cores, Divisões, e outras Strings, com base em listas, das quais são retirados *randomly* os valores nelas contidos
- Assim, e com base em tudo o que foi dito, foi também possível a implementação das funções `formatRegisto`, `generateRegisto` e `generateFile` que visão, tal como o nome indica, gerar um ficheiro com o número de linhas indicadas, gerando assim um registo de forma arbitrária que é formato e passado para uma String pronta a imprimir.

Pode-se encontrar na secção 6.1 a solução desenvolvida para gerar automáticamente o ficheiro de *Logs* com recurso ao *QuickCheck*

5.2 Hypothesis

Por sua vez, no que toca ao *Hypotesis*, após aplicar-se o mesmo raciocínio, obtive-se um ficheiro *python* com as seguintes características:

- Várias listas declaradas das quais se poderão gerar, por exemplo, os Fornecedores, a Resolução das Camaras, etc.
- A construção de métodos *strategy* que tratam de funções individuais necessárias para gerar cada secção dos *logs*
- De resto, utiliza-se métodos como *round*, para limitar numero de casas décimais, e outras estratégias de formatação do ficheiro.

Pode-se encontrar na secção 6.1 a solução desenvolvida para gerar automáticamente o ficheiro de *Logs* com recurso ao *Hypothesis*

Capítulo 6

Conclusão

Após terem sido utilizadas grande parte das diferentes ferramentas que foram lecionadas ao longo deste semestre, pode-se concluir que as funcionalidades oferecidas por estas foram utilizadas de forma eficiente e concisa, o que levou a uma análise detalhada dos projetos mencionados para se poderem atingir os melhores resultados.

Através do uso do *PIT*, foi-nos proporcionada uma nova perspetiva dos testes que se desenvolveu, permitindo a melhoria da sua cobertura. Por sua vez, o uso do `EvoSuite` tornou evidente a importância fa geração automática de testes unitários, proporcionando uma alternativa à criação de testes de forma manual. Já o uso de *QuickCheck* e de *Hypotheis* levou â descoberta de uma forma de facilitar a criação de ficheiros de logs que são carregados no inicio da aplicação abordada.

Apesar disso, não foi possível a utilizaçãp da ferramenta `SonarQube` de forma correta, tendo em conta que não se conseguiu completar a sua definição inicial. Ainda assim, foi fornecida a pasta gerada pela ferramenta nos ficheiro entregues.

Em suma, estamos bastante satisfeitos com a forma que utilizamos as ferramentas lecionadas nas aulas e acreditamos que fomos capazes de atingir os objetivos traçados no enunciado de forma positiva. Quanto a dificuldades encontradas, a necessidade do EvoSuite ter de trabalhar com JDK8 causou imensos problemas, visto que esse nível de linguagem Java não suportava muitas expressões usadas nos trabalhos, especialmente no *SmartHouse_1*, daí termos utilizado o *SmartHouse_2* em alternativa.

6.1 Anexos

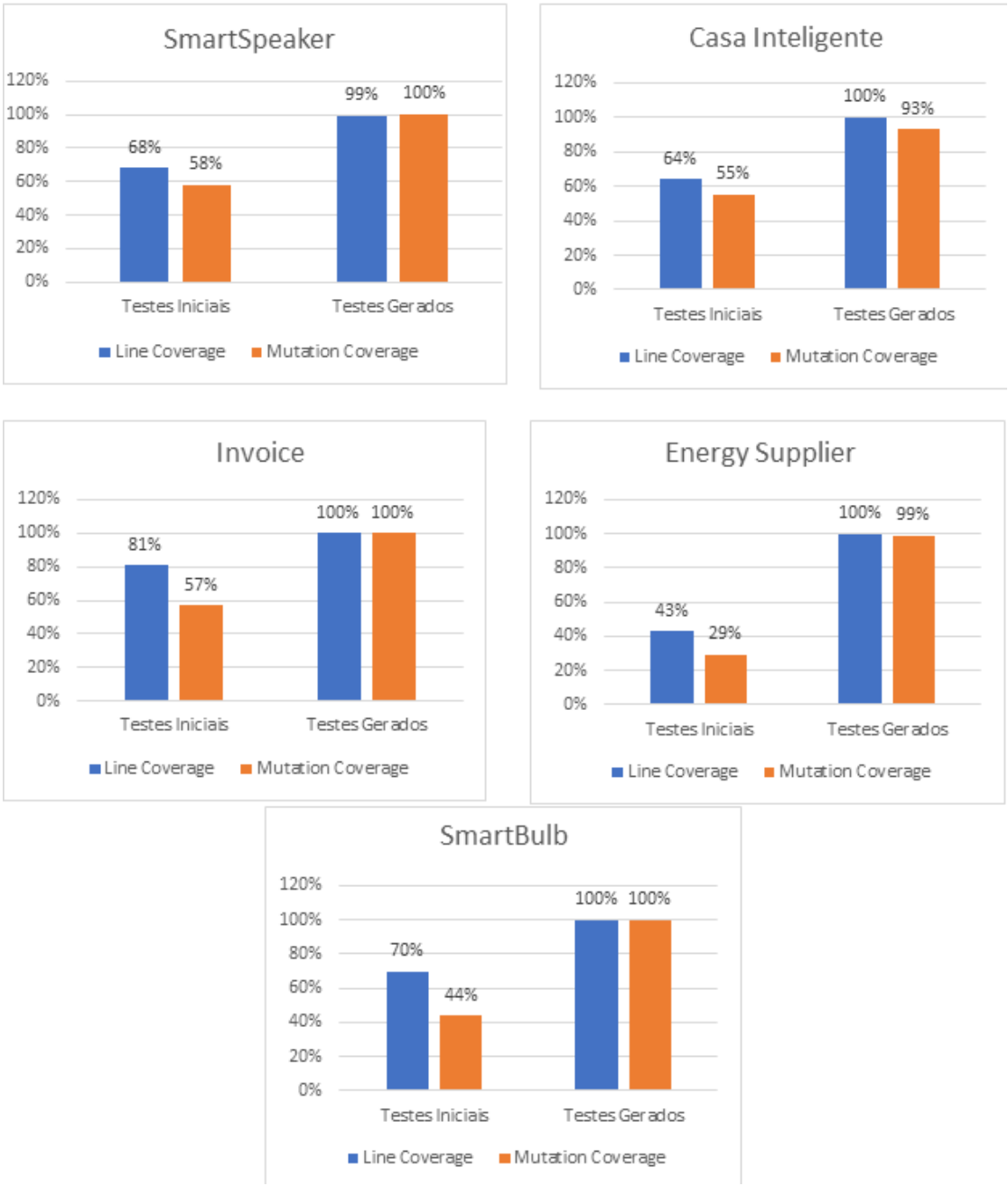

Figura 6.1: Comparação dos resultados obtidos entre os testes originais e os testes do *EvoSuite*.

```haskell
-- QUICKCHECK

import Test.QuickCheck
import System.IO

roundTo :: Int -> Double -> Double
roundTo n x = fromInteger (round $ x * (10^n)) / (10.0^^n)

gerarNumeroUnico :: Gen Int
gerarNumeroUnico = choose (100000000, 999999999)

gerarNumero :: (Int, Int) -> Gen Int
gerarNumero (x,y)= choose (x,y)

gerarDouble :: (Double, Double) -> Gen Double
gerarDouble (x,y)= choose (x,y)

data Fornecedor = EDP | Galp | Iberdrola | Endesa | GoldEnergy | Coopernico | Enat
     | YIce | MEOEnergia | Muon | Luzboa | EnergiaSimples | SUElectricidade | EDA
  deriving (Show, Eq)

data Nome = Diana | Rui | Afonso | Gongas | Artur | Bea
  deriving (Show, Eq)

data Casa = Casa Nome Int Fornecedor
  deriving (Show, Eq)

data Divisao = Quarto | Cozinha | Sotao | Cave | Sala | Escritorio
  deriving (Show, Eq)

data Cores = Vermelho | Amarelo | Azul | Laranja | Verde | Roxo | Branco
  deriving (Show, Eq)

data Estacao = Comercial | RFM | Noticias | AUMINHO | MEGAHITS | M80 | RTP |
    Renascenca
  deriving (Show, Eq)

data Marca =  Sony | Marshall | JBL | Sennheiser | BowersWilkins | Philips
  deriving (Show, Eq)

data SmartDevice = SmartBulb Cores Int Double | SmartCamera Int Int Int Double |
    SmartSpeaker Int Estacao Marca Double
  deriving (Show, Eq)

data Registro = Fornecedor Fornecedor | CasaRegisto Casa | DivisaoRegisto Divisao
    | SmartDevice SmartDevice
  deriving (Show, Eq)

instance Arbitrary Fornecedor where
  arbitrary = elements [EDP, Galp, Iberdrola, Endesa, GoldEnergy, Coopernico, Enat
    , YIce, MEOEnergia, Muon, Luzboa, EnergiaSimples, SUElectricidade, EDA]

instance Arbitrary Nome where
  arbitrary = elements [Diana, Rui, Afonso, Gongas, Artur, Bea]

instance Arbitrary Casa where
  arbitrary = Casa <$> arbitrary <*> gerarNumeroUnico <*> arbitrary

instance Arbitrary Divisao where
  arbitrary = elements [Quarto, Cozinha, Sotao, Cave, Sala, Escritorio]

instance Arbitrary Cores where
```

```
  arbitrary = elements [Vermelho, Amarelo, Azul, Laranja, Verde, Roxo, Branco]

instance Arbitrary Estacao where
  arbitrary = elements [Comercial, RFM, Noticias, AUMINHO, MEGAHITS, M80, RTP,
    Renascenca]

instance Arbitrary Marca where
  arbitrary = elements [Sony, Marshall, JBL, Sennheiser, BowersWilkins, Philips]

instance Arbitrary SmartDevice where
  arbitrary = oneof [SmartBulb <$> arbitrary <*> gerarNumero (0,100) <*>
    gerarDouble (0.0,10.0), SmartCamera <$> gerarNumero (1024,3840) <*>
    gerarNumero (768,2160) <*> gerarNumero (0,100) <*> gerarDouble(0.0,10.0),
    SmartSpeaker <$> gerarNumero (0,100) <*> arbitrary <*> arbitrary <*>
    gerarDouble(0.0,10)]

instance Arbitrary Registro where
  arbitrary = oneof [Fornecedor <$> arbitrary, CasaRegisto <$> arbitrary,
    DivisaoRegisto <$> arbitrary, SmartDevice <$> arbitrary]

formatRegistro :: Registro -> String
formatRegistro (Fornecedor fornecedor) = "Fornecedor:" ++ show fornecedor
formatRegistro (CasaRegisto (Casa nome num fornecedor)) = "Casa:" ++ show nome ++
    "," ++ show num ++ "," ++ show fornecedor
formatRegistro (DivisaoRegisto divisao) = "Divisao:" ++ show divisao
formatRegistro (SmartDevice device) = case device of
  SmartBulb t i d -> "SmartBulb:" ++ show t ++ "," ++ show i ++ "," ++ show (
    roundTo 2 d)
  SmartCamera a b i d -> "SmartCamera:" ++ "(" ++ show a ++ "x" ++ show b ++ ")"
    ++ "," ++ show i ++ "," ++ show (roundTo 2 d)
  SmartSpeaker i s b d -> "SmartSpeaker:" ++ show i ++ "," ++ show s ++ "," ++
    show b ++ "," ++ show (roundTo 2 d)

generateRegistro :: IO Registro
generateRegistro = generate arbitrary

generateFile :: FilePath -> Int -> IO ()
generateFile filePath numLines = do
  registros <- generate $ vectorOf numLines arbitrary
  withFile filePath WriteMode $ \handle -> do
    hPutStr handle $ unlines $ map formatRegistro registros

main :: IO ()
main = generateFile "Logs/LogsHS.txt" 50
```

```
## HYPOTHESIS

import hypothesis.strategies as st
import hypothesis.extra.numpy as npst
from hypothesis import given, settings
import numpy as np
import random

fornecedores = [
    "EDP Comercial",
    "Galp Energia",
    "Iberdrola",
    "Endesa",
    "Gold Energy",
    "Coopernico",
    "Enat",
    "YIce",
```

```
    "MEO Energia",
    "Muon",
    "Luzboa",
    "Energia Simples",
    "SU Electricidade",
    "EDA"
]

marca = [
    "Sony",
    "Marshall",
    "JBL",
    "Sennheiser",
    "Bowers&Wilkins",
    "Philips"
]

estacao = [
    "Comercial",
    "RFM",
    "Noticias",
    "AUMINHO",
    "MEGAHITS",
    "M80",
    "RTP",
    "Renascenca"
    ]

nomes = [
    "Diana",
    "Rui",
    "Afonso",
    "Bea",
    "Gongas",
    "Artur"
]

rooms = [
    "Quarto",
    "Cozinha",
    "Sala",
    "Sot o",
    "Cave",
    "Sala de Jantar"
]

colors = [
    "Vermelho",
    "Amarelo",
    "Azul",
    "Laranja",
    "Verde",
    "Roxo",
    "Branco"
]

rez = [
    "(1024x768)",
    "(1366x768)",
    "(1920x1080)",
    "(2160x1440)",
    "(3840x2160)"
```

```
]

@st.composite
def fornecedor_strategy(draw):
    return draw(st.sampled_from(fornecedores))

@st.composite
def name_strategy(draw):
    return draw(st.sampled_from(nomes))

@st.composite
def nif_strategy(draw):
    return draw(st.sampled_from(nif))

@st.composite
def room_strategy(draw):
    return draw(st.sampled_from(rooms))

@st.composite
def casa_strategy(draw):
    nome = draw(name_strategy())
    num = draw(st.integers(min_value=100000000, max_value=999999999))
    fornecedor = draw(fornecedor_strategy())
    return f"Casa:{nome},{num},{fornecedor}"

@st.composite
def divisao_strategy(draw):
    nome = draw(room_strategy())
    return f"Divisao:{nome}"

@st.composite
def fn_strategy(draw):
    nome = draw(fornecedor_strategy())
    return f"Fornecedor:{nome}"

@st.composite
def smartbulb_strategy(draw):
    color = draw(st.sampled_from(colors))
    intensity = draw(st.integers(min_value=0, max_value=20))
    power = draw(st.floats(min_value=1.0, max_value=10.0))
    power = round(power, 3)  # Limit to 3 decimal places
    return f"SmartBulb:{color},{intensity},{power}"

@st.composite
def smartcamera_strategy(draw):
    resolution = draw(st.sampled_from(rez))
    fps = draw(st.integers(min_value=20, max_value=160))
    power = draw(st.floats(min_value=0.0, max_value=10.0))
    power = round(power, 3)  # Limit to 3 decimal places
    return f"SmartCamera:{resolution},{fps},{power}"

@st.composite
def smartspeaker_strategy(draw):
    volume = draw(st.integers(min_value=0, max_value=100))
    channel = draw(st.sampled_from(estacao))
    brand = draw(st.sampled_from(["Sony", "Marshall", "JBL", "Sennheiser", "Bowers
    &Wilkins", "Philips"]))
    power = draw(st.floats(min_value=0.0, max_value=10.0))
    power = round(power, 3)  # Limit to 3 decimal places
    return f"SmartSpeaker:{volume},{channel},{brand},{power}"

@st.composite
```

```python
def registro_strategy(draw):
    return draw(st.one_of(
        fn_strategy(),
        casa_strategy(),
        divisao_strategy(),
        smartbulb_strategy(),
        smartcamera_strategy(),
        smartspeaker_strategy(),
    ))

@settings(max_examples=300)
@given(st.lists(registro_strategy(), min_size=100, max_size=150, unique=True))

def generate_file(registros):
    file_content = "\n".join(registros)
    with open("Logs/LogsPY.txt", "w") as f:
        f.write(file_content)

generate_file()
```

```haskell
import Test.QuickCheck
import System.IO

roundTo :: Int -> Double -> Double
roundTo n x = fromInteger (round $ x * (10^n)) / (10.0^^n)

gerarNumeroUnico :: Gen Int
gerarNumeroUnico = choose (100000000, 999999999)

gerarNumero :: (Int, Int) -> Gen Int
gerarNumero (x,y)= choose (x,y)

gerarDouble :: (Double, Double) -> Gen Double
gerarDouble (x,y)= choose (x,y)

data Fornecedor = EDP | Galp | Iberdrola | Endesa | GoldEnergy | Coopernico | Enat
     | YIce | MEOEnergia | Muon | Luzboa | EnergiaSimples | SUElectricidade | EDA
  deriving (Show, Eq)

data Nome = Diana | Rui | Afonso | Gongas | Artur | Bea
  deriving (Show, Eq)

data Casa = Casa Nome Int Fornecedor
  deriving (Show, Eq)

data Divisao = Quarto | Cozinha | Sotao | Cave | Sala | Escritorio
  deriving (Show, Eq)

data Cores = Vermelho | Amarelo | Azul | Laranja | Verde | Roxo | Branco
  deriving (Show, Eq)

data Estacao = Comercial | RFM | Noticias | AUMINHO | MEGAHITS | M80 | RTP |
    Renascenca
  deriving (Show, Eq)

data Marca =  Sony | Marshall | JBL | Sennheiser | BowersWilkins | Philips
  deriving (Show, Eq)

data SmartDevice = SmartBulb Cores Int Double | SmartCamera Int Int Int Double |
    SmartSpeaker Int Estacao Marca Double
  deriving (Show, Eq)

data Registro = Fornecedor Fornecedor | CasaRegisto Casa | DivisaoRegisto Divisao
```

```
    | SmartDevice SmartDevice
  deriving (Show, Eq)

instance Arbitrary Fornecedor where
  arbitrary = elements [EDP, Galp, Iberdrola, Endesa, GoldEnergy, Coopernico, Enat
    , YIce, MEOEnergia, Muon, Luzboa, EnergiaSimples, SUElectricidade, EDA]

instance Arbitrary Nome where
  arbitrary = elements [Diana, Rui, Afonso, Gongas, Artur, Bea]

instance Arbitrary Casa where
  arbitrary = Casa <$> arbitrary <*> gerarNumeroUnico <*> arbitrary

instance Arbitrary Divisao where
  arbitrary = elements [Quarto, Cozinha, Sotao, Cave, Sala, Escritorio]

instance Arbitrary Cores where
  arbitrary = elements [Vermelho, Amarelo, Azul, Laranja, Verde, Roxo, Branco]

instance Arbitrary Estacao where
  arbitrary = elements [Comercial, RFM, Noticias, AUMINHO, MEGAHITS, M80, RTP,
    Renascenca]

instance Arbitrary Marca where
  arbitrary = elements [Sony, Marshall, JBL, Sennheiser, BowersWilkins, Philips]

instance Arbitrary SmartDevice where
  arbitrary = oneof [SmartBulb <$> arbitrary <*> gerarNumero (0,100) <*>
    gerarDouble (0.0,10.0), SmartCamera <$> gerarNumero (1024,3840) <*>
    gerarNumero (768,2160) <*> gerarNumero (0,100) <*> gerarDouble(0.0,10.0),
    SmartSpeaker <$> gerarNumero (0,100) <*> arbitrary <*> arbitrary <*>
    gerarDouble(0.0,10)]

instance Arbitrary Registro where
  arbitrary = oneof [Fornecedor <$> arbitrary, CasaRegisto <$> arbitrary,
    DivisaoRegisto <$> arbitrary, SmartDevice <$> arbitrary]

formatRegistro :: Registro -> String
formatRegistro (Fornecedor fornecedor) = "Fornecedor:" ++ show fornecedor
formatRegistro (CasaRegisto (Casa nome num fornecedor)) = "Casa:" ++ show nome ++
    "," ++ show num ++ "," ++ show fornecedor
formatRegistro (DivisaoRegisto divisao) = "Divisao:" ++ show divisao
formatRegistro (SmartDevice device) = case device of
  SmartBulb t i d -> "SmartBulb:" ++ show t ++ "," ++ show i ++ "," ++ show (
    roundTo 2 d)
  SmartCamera a b i d -> "SmartCamera:" ++ "(" ++ show a ++ "x" ++ show b ++ ")"
    ++ "," ++ show i ++ "," ++ show (roundTo 2 d)
  SmartSpeaker i s b d -> "SmartSpeaker:" ++ show i ++ "," ++ show s ++ "," ++
    show b ++ "," ++ show (roundTo 2 d)

generateRegistro :: IO Registro
generateRegistro = generate arbitrary

generateFile :: FilePath -> Int -> IO ()
generateFile filePath numLines = do
  registros <- generate $ vectorOf numLines arbitrary
  withFile filePath WriteMode $ \handle -> do
    hPutStr handle $ unlines $ map formatRegistro registros

main :: IO ()
main = generateFile "Logs/LogsHS.txt" 50
```