

TRABALHO PRÁTICO 2 - ESII - GRUPO1213

WHITE BOX TESTS

Version 3.0
21/01/2022

Histórico de Versões

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	Bruno Ferreira Gonçalo Oliveira Jorge Correia Nuno Castro	14-01-2022	Bruno Ferreira Gonçalo Oliveira Jorge Correia Nuno Castro	14-01-2022	Análise testes caixa branca para o módulo de transações
2.0	Bruno Ferreira Gonçalo Oliveira Jorge Correia Nuno Castro	18-01-2022	Bruno Ferreira Gonçalo Oliveira Jorge Correia Nuno Castro	18-01-2022	Análise testes caixa branca para o módulo de custos de envio e comparação das taxas de cobertura em relação à versão anterior.
3.0	Bruno Ferreira Gonçalo Oliveira Jorge Correia Nuno Castro	21-01-2022	Bruno Ferreira Gonçalo Oliveira Jorge Correia Nuno Castro	21-01-2022	Análise testes caixa branca para o módulo expedição e comparação das taxas de cobertura em relação à versão anterior.

Tabela de Conteúdos

1. Introdução	4
1.1. Identificador do documento	4
1.2. Âmbito	4
1.3. Repositório	4
1.4. Glossário	5
1.5. Dados disponibilizados pelo Jacoco	6
2. Análise de cobertura para cada Classe	7
2.1. Classe BasicLinhaTransacao	7
2.2. Classe BasicGestaoEncomendas	8
2.3. Classe Estatísticas	9
2.4. Classe Custos	10
2.5. Classe Expedições	11
2.6. Classe BasicContentor	12
2.7. Observações	13

1. Introdução

1.1. Identificador do documento

WhiteBoxTests_Grupo1213

1.2. Âmbito

Este documento foi criado no âmbito de apresentar os resultados dos testes de caixa branca, ou seja, apresentar os respetivos resultados de cobertura.

Após a codificação e execução dos casos de teste, identificados nos BlackBoxTests, é necessário realizar os WhiteBoxTests que consiste basicamente em testar a cobertura de ramos/instruções que os testes realizados alcançaram. Uma alta taxa de cobertura significa que uma grande parte de todo o código foi testado e, conseqüentemente, será menos propício a erros.

Uma vez que temos o plugin Jacoco, podemos utilizar o mesmo para analisar as informações descritas anteriormente. Além de identificar as taxas de cobertura, o Jacoco ainda indica quais as instruções que não foram cobertas.

Após o registo das instruções não cobertas, algumas delas serão eventualmente testadas para ser atingida uma maior taxa.

Numa fase inicial, foi definido pelo grupo que a taxa média de cobertura mínima seria de 65%, uma vez que existem vários métodos que ainda não podem ser testados e, conseqüentemente, possuem 0% de taxa de cobertura. No entanto, a taxa mínima para métodos que foram testados é de 80%, sendo que os testes devem garantir que cobrem a maior parte das instruções.

Com o desenvolver do projeto esta taxa de aceitação vai aumentando, exigindo uma maior cobertura a cada módulo.

1.3. Repositório

A hiperligação para o repositório onde foi realizado o trabalho é a seguinte:

<https://gitlab.estg.ipp.pt/esii.grupo1213/esii-grupo1213-tp2>

1.4. Glossário

Testes de software - Um teste de software é um software que executa outro software, validando se os resultados são os esperados (teste de estado) ou se executa a sequência de eventos esperado (teste de comportamento).

Resumidamente, os testes de software permitem aos programadores verificar se a lógica do programa desenvolvido está de acordo com os requisitos.

A execução automática de testes permite identificar “bugs” resultantes de mudanças no código fonte.

ECP - Técnica destinada a reduzir o número de testes necessários dividindo o domínio de entrada (ou saída) em classes de dados em que os casos de teste podem ser derivados para cada operação, o “tester” deve identificar as classes de equivalência dos argumentos e os estados dos objetos

BVA - Técnica baseada na observação de bugs que ocorrem frequentemente em valores fronteira. É focada em testar valores especiais (null, 0, etc) e limites do domínio de entrada (ou saída) imediatamente acima e abaixo (além de ou em vez de valores intermédios)

JUnit - Framework open-source para realizar testes unitários para código em Java.

Permite a execução de testes de forma:

- Fácil
- Regular
- Fiável

Contém várias funcionalidades para testing entre elas a capacidade de testar cada componente de um programa de forma independente do resto do programa

Gradle - É uma ferramenta que permite integrar e automatizar várias tarefas relacionadas com o processo de desenvolvimento de software em várias linguagens de programação.

O Gradle determina quais os componentes do projeto que estão atualizados, evitando a recompilação de todo o projeto.

Jacoco - É uma biblioteca que disponibiliza taxas de cobertura e que identifica que parte do código não foi coberta.










1.5. Dados disponibilizados pelo Jacoco

Podemos concluir que pela análise disponibilizada pelo Jacoco, temos algumas classes que têm uma taxa de cobertura desejada pelo grupo é atingida (83%).

No entanto, nem sempre significa que a cobertura tenha sido mal feita uma vez que, a metodologia seguida pelo nosso grupo implica o desenvolvimento, análise e testagem para cada módulo, pelo que alguns métodos devem apenas ser testados em módulos posteriores.

Alguns dos métodos não são testados, também pela sua baixa complexidade, tais como gets e sets, sendo que são métodos em que a taxa de erro é muito pequena pelo que, apenas foram testados alguns que implicam uma complexidade superior ao habitual.

ESII-Grupo1213-TP2

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
estg.ipp.pt.Utils		33%		34%	26 36	109 158	14 20	0 3
estg.ipp.pt.Modulo_Transacoes		86%		79%	21 110	33 276	5 58	0 7
estg.ipp.pt.Modulo_Expedicoes		89%		79%	8 33	10 73	1 16	0 2
estg.ipp.pt.Modulo_Calculo_Custos		100%		100%	0 5	0 18	0 4	0 1
estg.ipp.pt.Enums		100%	n/a	n/a	0 9	0 22	0 9	0 3
Total	610 of 3 648	83%	49 of 172	71%	55 193	152 547	20 107	0 16




Neste relatório, para cada método, será realizada uma análise de cobertura e uma identificação de testes adicionais, caso seja necessário.

2. Análise de cobertura para cada Classe

2.1. Classe BasicLinhaTransacao

Como podemos observar na imagem seguinte, na classe BasicLinhaTransacao foram cobertos 100% das instruções, o que significa que os testes feitos são suficientes para testar todas as instruções, o que por sua vez, ajuda a evitar a bugs.

BasicLinhaTransacao

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● toString()		100%		n/a	0	1	0	4	0	1
● BasicLinhaTransacao(Produto_int)		100%		n/a	0	1	0	3	0	1
● getProduto()		100%		n/a	0	1	0	1	0	1
Total	0 of 35	100%	0 of 0	n/a	0	3	0	8	0	3













2.2. Classe BasicGestaoEncomendas

Em relação à versão anterior podemos observar que a taxa de cobertura de instruções desceu 17%. Esta descida deve-se ao facto de terem sido adicionadas novas instruções que apenas devem ser testadas posteriormente, no próximo módulo.













No entanto, a taxa revela-se suficiente uma vez que é superior à taxa de cobertura mínima definida pelo grupo para cada método (80%).

Versões anteriores:

BasicGestaoEncomendas

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● countEncomendas()		0%		n/a	1	1	1	1	1	1
● addEncomenda(Encomenda)		100%		83%	1	4	0	8	0	1
● findEncomenda(String)		100%		100%	0	4	0	7	0	1
● cancelarEncomenda(Encomenda)		100%		100%	0	3	0	7	0	1
● BasicGestaoEncomendas()		100%		n/a	0	1	0	3	0	1
● getEncomendas()		100%		n/a	0	1	0	1	0	1
Total	4 of 98	95%	1 of 16	93%	2	14	1	27	1	6















BasicGestaoEncomendas

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● addEncomenda(Encomenda)		83%		83%	1	4	1	8	0	1
● countEncomendas()		0%		n/a	1	1	1	1	1	1
● findEncomenda(String)		100%		100%	0	4	0	7	0	1
● cancelarEncomenda(Encomenda)		100%		100%	0	3	0	7	0	1
● BasicGestaoEncomendas()		100%		n/a	0	1	0	3	0	1
● getEncomendas()		100%		n/a	0	1	0	1	0	1
Total	9 of 98	90%	1 of 16	93%	2	14	2	27	1	6

Quanto à cobertura de ramos, podemos observar que se manteve nos 83%, pelo mesmo motivo apresentado na versão anterior, o facto de que será apenas testado posteriormente, no próximo módulo.

Versão atual:

BasicGestaoEncomendas

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● atribuirEncomendas(String)		0%		0%	4	4	9	9	1	1
● importEncomendas(String)		0%		0%	2	2	8	8	1	1
● findEncomenda(String)		100%		100%	0	4	0	7	0	1
● cancelarEncomenda(Encomenda)		100%		83%	1	4	0	7	0	1
● addEncomenda(Encomenda)		100%		100%	0	3	0	6	0	1
● BasicGestaoEncomendas()		100%		n/a	0	1	0	3	0	1
● getEncomendas()		100%		n/a	0	1	0	1	0	1
Total	88 of 173	49%	9 of 24	62%	7	19	17	41	2	7














Nesta nova versão podemos observar que o método addEncomenda atingiu os 100% de cobertura. Contudo existem 2 métodos que possuem 0% de cobertura, isto acontece devido ao facto de serem métodos que serão utilizados no programa para testar a API e que por isso não foram testados..

2.3. Classe Estatísticas

Para esta classe foi obtida uma boa percentagem geral de cobertura.

No caso do construtor não existe cobertura visto que não deve ser possível a instanciação da classe Estatísticas.

Estatísticas














Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
findOldestDate(Iterator)		53%		50%	2	4	3	8	0	1
Estatísticas()		0%	n/a	n/a	1	1	2	2	1	1
getNumMedioProdutosPorTransacao(Iterator)		100%		100%	0	5	0	13	0	1
getTotalCustosEnvio(Iterator, LocalDate, LocalDate)		100%		92%	1	8	0	11	0	1
getValorMedioTransacoes(Iterator)		100%		100%	0	4	0	12	0	1
getNumMedioEncomendasPorDia(Iterator)		100%		100%	0	3	0	8	0	1
iteratorToArray(Iterator)		100%		75%	1	3	0	7	0	1
Total	19 of 229	91%	5 of 42	88%	5	28	5	61	1	7

Quanto à cobertura de ramos, apenas o método findOldestDataIterator apresenta uma taxa de cobertura inferior a 100%, devido a não ser testado diretamente.














Já o método getTotalCustosEnvio no Missed Branches está a 92% devido a falta de um teste para quando a data de uma das encomendas seja menor do que a dataInicio.

Devido a esse teste em falta, o mesmo foi adicionado e, como se pode observar, os 100% de cobertura foram atingidos.

Estatísticas

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
findOldestDate(Iterator)		53%		50%	2	4	3	8	0	1
Estatísticas()		0%	n/a	n/a	1	1	2	2	1	1
getNumMedioProdutosPorTransacao(Iterator)		100%		100%	0	5	0	13	0	1
getTotalCustosEnvio(Iterator, LocalDate, LocalDate)		100%		100%	0	8	0	11	0	1
getValorMedioTransacoes(Iterator)		100%		100%	0	4	0	12	0	1
getNumMedioEncomendasPorDia(Iterator)		100%		100%	0	3	0	8	0	1
iteratorToArray(Iterator)		100%		75%	1	3	0	7	0	1
Total	19 of 229	91%	4 of 42	90%	4	28	5	61	1	7

Estatísticas




Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
findOldestDate(Iterator)		53%		50%	2	4	3	8	0	1
Estatísticas()		0%	n/a	n/a	1	1	2	2	1	1
getValorMedioVendasComprasPorDistrito(Iterator)		100%		100%	0	8	0	29	0	1
getNumMedioProdutosPorTransacao(Iterator)		100%		100%	0	5	0	13	0	1
getTotalCustosEnvio(Iterator, LocalDate, LocalDate)		100%		100%	0	8	0	11	0	1
getValorMedioTransacoes(Iterator)		100%		100%	0	4	0	12	0	1
getNumMedioEncomendasPorDia(Iterator)		100%		100%	0	3	0	8	0	1
Total	19 of 341	94%	3 of 52	94%	3	33	5	83	1	7

Nesta nova versão foi adicionado o método “getValorMedioVendasComprasPorDistrito” que possui 100% de cobertura.

2.4. Classe Custos

Como podemos observar na imagem seguinte, na classe BasicCustos foram cobertos 100% das instruções e ramos, o que significa que os testes feitos são suficientes para testar todas as instruções, o que por sua vez, ajuda a evitar a bugs.

BasicCustos

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
BasicCustos()		100%	n/a		0 1	0 3	0 1
calculaCustoEnvio(Encomenda)		100%		100%	0 2	0 12	0 1
getDistance(District, District)		100%	n/a		0 1	0 1	0 1
static {...}		100%	n/a		0 1	0 2	0 1
Total	0 of 1 459	100%	0 of 2	100%	0 5	0 18	0 4













2.5. Classe Expedições

Como podemos observar na imagem seguinte, na classe Expedições foram cobertos 90% das instruções e 88% dos ramos, o que significa que os testes feitos são suficientes para testar uma boa parte do funcionamento desta classe.

O método addContentor possui uma cobertura menor visto que é um método invocado indiretamente pelo que não é testado se o distrito enviado é null visto que este método não pode ser chamado por outras classes.

No caso do construtor não existe cobertura visto que não deve ser possível a instanciação da classe Expedições.

Expedicoes

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
Expedicoes()		0%		n/a	1	1	2	2	1	1
addContentor(District)		66%		50%	1	2	1	4	0	1
assignEncomendas(Iterator, String)		92%		100%	0	6	2	16	0	1
setEncomendaToContentor(Encomenda)		97%		75%	1	3	1	11	0	1
checkContentoresLivres(District)		100%		83%	1	4	0	7	0	1
revertOrderState()		100%		100%	0	3	0	7	0	1
static {...}		100%		n/a	0	1	0	1	0	1
Total	15 of 165	90%	3 of 26	88%	4	20	6	48	1	7

2.6. Classe BasicContentor

Como podemos observar na imagem seguinte, na classe BasicContentor foram cobertos 86% das instruções e 50% dos ramos.

O construtor possui uma cobertura de ramos de 50% dado que não estão a ser testados métodos construtores a não ser que se encontre algum erro.

O método addEncomenda também possui uma cobertura de ramos de 50% porque é um método invocado indiretamente pela classe expedições. Como o construtor desta classe foi instanciado como protected, decidimos não criar testes específicos para este método.

BasicContentor






Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• addEncomenda(Encomenda)		76%		50%	3	4	3	9	0	1
• BasicContentor(District)		88%		50%	1	2	1	8	0	1
• getEncomendas()		100%		n/a	0	1	0	1	0	1
• getSizeEncomendas()		100%		n/a	0	1	0	1	0	1
• getIdContentor()		100%		n/a	0	1	0	1	0	1
• getDistrito()		100%		n/a	0	1	0	1	0	1
• getCapacidade()		100%		n/a	0	1	0	1	0	1
• getEstado()		100%		n/a	0	1	0	1	0	1
• static {...}		100%		n/a	0	1	0	2	0	1
Total	15 of 108	86%	4 of 8	50%	4	13	4	25	0	9

2.7. Observações

Visto todos os dados fornecidos pelo Jacoco podemos analisar alguns métodos em que a taxa de cobertura se revela bastante baixa, no entanto, isso acontece devido ao facto de que alguns métodos devem ser testados apenas posteriormente, uma vez que, não possuem testes a taxa é 0%.

Exemplo:

Files

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● readFile(String)		0%		0%	2	2	8	8	1	1
● Files()		0%		n/a	1	1	2	2	1	1
● isValidPath(String)		72%		n/a	0	1	2	5	0	1
● writeFile(String,String)		100%		n/a	0	1	0	11	0	1
Total	39 of 78	50%	2 of 2	0%	3	5	12	26	2	4