Entrega 5 - Line and Branch Coverage

White-Boxing **1**
- Structural testing 1
  - Line Coverage 1
  - Branch Coverage 1
- JaCoCo 1
  - Instalação da ferramenta 1
  - Execução da ferramenta 2
  - Primeira análise 2

Coverage aos testes realizados nas entregas anteriores **2**
  - Última análise 3

André Cruz nº 51067
Ricardo Madeira nº49434

White-Boxing

Até esta entrega, todos os relatórios eram baseados em técnicas de *black-boxing*, em que os *testers* realizam testes ao software, a partir da documentação do mesmo.
Nesta entrega, somos introduzidos a um novo tipo de testagem chamado *white-boxing*, em que a fonte de informação para os testes é o próprio *software*.

Structural testing

Já se viu anteriormente que é importante realizar análises ao input e, a partir das mesmas, fazer testes condizentes. Contudo, muitas vezes não é possível fazer esta análise a todos os inputs que um projeto de software têm. Assim, o *tester* têm de decidir o que é um conjunto de testes adequado e, para isso, ver o quanto este conjunto de testes cobre o programa. Porque, se um conjunto de código não é executado nos testes, não é possível concluir se esse software não vai resultar em falhas no sistema, daí a importância definir o conjunto de testes a ser feito.

Line Coverage

Um dos critérios para definir esse conjunto de testes apropriado chama-se ***line coverage,*** em que um conjunto de testes é adequado se cobre todas as linhas do código a que se propõe. O *threshold* mínimo para cobrir linhas de código pode depender dos requisitos do sistema, assim como restrições impostas ao *tester*. Este valor pode ser uma percentagem alta, contudo implica uma maior testagem, maiores custos e pode não ser benéfico, enquanto uma percentagem baixa, é sinónimo de um sistema não testado e, por isso, propenso a falhas.

Branch Coverage

Outro critério, mais apropriado a sistemas complexos, é designado por ***branch coverage.*** Esta técnica, similarmente à anterior, cobre linhas de código, contudo, esta, cobre todas as alternativas em condições complexas (Como por exemplo *If/Eles statements)*, que não são cobertas por um só teste.

JaCoCo

Para realizar estas técnicas e para ajudar os *testers* a perceber a cobertura realizada pelas mesmas, é utilizada a ferramenta *JaCoCo*.

Instalação da ferramenta

Para esta entrega, começámos por instalar o jaCoCo, introduzindo-o nas dependências do projeto (através do pom.xml), e reconstruindo o mesmo.

André Cruz nº 51067
Ricardo Madeira nº49434

Execução da ferramenta

Correr o projeto *Maven Build,* atualizar *Goals* com o valor “package” e de seguida clicar “Run/correr”.

Primeira análise

Uma primeira análise demonstrou que a coverage do código é bastante baixa, ficando com um total de 19 838/20 042 instruções não testadas.

jdotxt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.todotxt.todotxttouch		0%		n/a	3	3	5	5	3	3	2	2
com.chschmid.jdotxt.gui.utils		0%		0%	6	6	20	20	3	3	1	1
com.chschmid.jdotxt		0%		0%	25	25	76	76	16	16	2	2
com.chschmid.jdotxt.util		0%		0%	34	34	81	81	19	19	5	5
com.todotxt.todotxttouch.task.sorter		0%		0%	88	88	96	96	54	54	24	24
com.todotxt.todotxttouch.util		4%		6%	121	128	274	285	45	47	5	7
com.todotxt.todotxttouch.task		4%		2%	363	381	815	856	162	176	24	28
com.chschmid.jdotxt.gui		0%		0%	233	233	989	989	163	163	54	54
com.chschmid.jdotxt.gui.controls		0%		0%	722	722	1.705	1.705	369	369	81	81
Total	19.838 of 20.042	1%	1.517 of 1.538	1%	1.595	1.620	4.061	4.113	834	850	198	204

Isto deve-se aos, além dos testes já realizados pelos autores do projeto, poucos testes feitos por nós, em comparação à dimensão do projeto, cujo total são 20 042 instruções. Assim, a coverage total do código aproxima-se do 1%, o que é um valor francamente baixo para poder lançar o programa no mercado, visto que 99% do mesmo pode ter comportamentos inesperados.

Após esta análise, decidimos introduzir os testes provenientes dos *assignments* anteriores para a *coverage* do código ser revista.

Coverage aos testes realizados nas entregas anteriores

Na entrega da Category Partition:

- CursorPositionCalculator
  - Obtivemos coverage de 100 %
- Path
  - Obtivemos coverage de 100 %
- RelativeDate
  - Obtivemos coverage de 100 %
- Strings
  - Obtivemos coverage de 94%
  - Não obtivemos coverage total da classe porque, no método *insertPaddedIfNeeded(String, int, String),* não são verificadas todas as condições das *if statement.*
- Tree
  - Obtivemos coverage de 100 %

André Cruz nº 51067
Ricardo Madeira nº49434

Última análise

jdotxt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.chschmid.jdotxt.gui.controls		45%		28%	497	722	877	1,695	186	369	22	81
com.chschmid.jdotxt.gui		19%		12%	187	233	809	985	118	163	28	54
com.todotxt.todotxttouch.task		44%		20%	271	381	510	856	80	176	8	28
com.todotxt.todotxttouch.task.sorter		35%		0%	67	88	74	94	33	54	9	24
com.todotxt.todotxttouch.util		65%		54%	52	128	87	285	4	47	0	7
com.chschmid.jdotxt		80%		66%	7	25	18	75	1	16	0	2
com.chschmid.jdotxt.util		84%		70%	9	34	9	81	1	19	0	5
com.chschmid.jdotxt.gui.utils		46%		33%	4	6	11	20	2	3	0	1
com.todotxt.todotxttouch		100%		n/a	0	3	0	5	0	3	0	2
Total	11,781 of 20,042	41%	1,113 of 1,538	27%	1,094	1,620	2,395	4,096	425	850	67	204

Após uma melhor cobertura dos testes, podemos ver um incremento na quantidade de instruções verificadas. Inicialmente faltavam testar 19 838 instruções e, após os novos testes feitos, ficaram a sobrar 11 781 instruções, sensivelmente 59% do software desenvolvido. Apesar de ser um número baixo, já é uma boa porção testada, tendo em conta que os packages relacionados com a *Graphical User Interface* não eram para ser abordados.