VVS Report Assignment 2

Verificação e Validação de Software

2019/20

André Nicolau

47880

Contents

1 Introdução

Neste projeto vamos por em prática alguma ferramentas que são utilizadas para facilitar e tornar mais independentes os testes. A ferramenta HtmlUnit é utilizada para poupar tempo em introduzir os dados nas páginas web. O seu objetivo é automatizar esse mesmo processo. O DBSetup e Mockito têm como objetivo tornar possível o teste de determinadas funcionalidades sem ainda ter a implementação completa das mesmas.
Assim, o objetivo deste projeto é testar uma web app com as ferramentas já referidas e refletir neste relatório o processo usado e notas, se necessário. Para além da utilização das ferramentas já referidas, neste projeto também é necessário reportar falhas e sugerir resoluções com estimativas a *bugs* reportados pela a equipa usando o site `www.backlog.com`.

2 BugTracking

Como já foi referido, neste projeto é necessário recorrer a uma aplicação online para reportar e sugerir resoluções de *bugs*. Da minha parte foram reportados cerca de 3 bugs com os id's: VVS_ASSIGN2-1, VVS_ASSIGN2-14 e VVS_ASSIGN2-18. Nos quais foram sugeridas resoluções com as respetivas estimativas de tempo.

3 HtmlUnit

3.1 Organização

Os testes usando esta ferramenta estão organizados por classes, ou seja cada class contém um teste. Tomei esta decisão pois considero os testes algo extensos por isso para não haver uma class muito longa, fica dividida em classes. Foi criada também uma classe auxiliar aos testes (TestUtils.java) que tem o objetivo de adicionar e remover Customers e adicionar sales. Isto deve-se a não repetição de código entre classes. Se todas as classes têm de adicionar e remover Customers, é benefico ter essas funções numa só classe para assim haver uma melhor gestão caso seja necessário alterar código futuramente. Foram criadas vari áveis globais para que certas informações(por exemplo o *vat* de um customer) podessem ser acedidas nas funções de *setup*, *test* e *teardown*. De notar que no *teardown* apenas removi o customer e não as *addresses* ou *sales* pois supostamente a aplicação teria de os remover se um customer for removido.

3.2 Resultados

test	Result
test1	no Faults found
test2	no Faults found
test3	no Faults found
test4	no Faults found
test5	no Faults found

4 DBSetup

Em primeiro lugar foi necessário criar uma operação na class DBSetupUtils de forma a que seja possivel apenas adicionar *addresses* para assim termos o leque todo de informações indispensáveis para a realização dos testes. Não foi necessário usar a função *dbSetupTracker.skipNextLaunch()* pois não existe nenhum teste em que não haja operações de escrita. Tentei sempre usar os valores já inseridos na base de dados mas isso trás a desvantagem de cada vez que precisar de um *customer* ter de ir buscar todos à base de dados e escolher

um (neste caso o primeiro). Isto faz com que os testes sejam mais demorados. Assim pensei em implmentar uma função de *@BeforeAll* para ir á base de dados apenas uma vez para todos os testes. Como existiram erros que não consegui resolver a tempo, decidi retomar a abordagem mais demorada.
Neste conjunto de testes foi encontrado um bug de uma sale não ser eliminada quando um customer o é (alínea e)).

5 Mockito

A minha abordagem passa pelos serviços serem uma dependência para com a camada de apresentação. Assim irei tentar fazer o *mock* das classes *CustomerServices* e *SaleServices* para a utilização da camada de aplicação.

5.1 Validação

A camada de negócio no projeto em teste encontra-se no package *services* sendo as classes *CustomerServices* e *SaleServices*. A partir de qualquer destas classes podemos observar que são enumerados. Usando a ferramenta Mockito, sabemos que não é possível fazer um *mock* de enumerados. Por isso, não é possivel fazê-lo sem haver alterações na camada de negócio.

5.2 Mudanças

A mudança seria mudar os enumerados para classes(que não *final* ou anónimas). Assim o Mockito já seria utilizável.
Um exemplo seria mudar o enumerado *CustomerServices* para uma class. O resultado da mudança contas no ficheiro que tem o nome de *CustomerServiceClass*.
No ficheiro TestMock.java está um exemplo de um mock válido usando a class modificada. Este mock faz uso das funções *addCustomer()* e *getCustomerByVat()*.

6 Modificações no SUT

As modificações feitas no SUT foram feitas a partir das sugestões de resolução dos bugs feitas na aplicação web. Muitas das modificações passaram por adicionar condicionais para verificar campos, alterar os ficheiros de criação da base de dados(usando sql), a adição de funções como a verificação se um determinado *vat* existe, modificação do construtor da class SaleRowDataGateway ou criação de condicionais por haver dados duplicados.

7 Conclusão

Em suma, podemos verificar que os testes foram bem sucedidos e que os bugs encontrados(quer seja no *backlog* ou nos testes requiridos) foram resolvidos(mas não na totalidade tendo em consideração todos os bugs descritos no *backlog*).