Análise de Teste e Software

Fase 2 - Projeto:UMer

José Dias	Pedro Silva	Tiago Fraga
(A78494)	(PG38935)	(A74092)

November 19, 2018

Contents

1 Introdução

Nesta 2ª Fase iremos avaliar a funcionalidade do código assim como as funcionalidades disponíveis para os utilizadores. Para tal, é necessário incidir sob dois aspetos: **Testes Unitários** e **Testes de Sistema**.

Inicialmente, foram realizados os **Testes Unitários** e, para isso, implementamos testes que permitem encontrar possíveis erros presentes no código. Estes testes foram criados tendo em conta os critérios de qualidade de testes, tais como critérios cobertura e o uso de mutação de software. Também utilizamos o sistema *Evosuite* para geração automática de testes unitários.

De seguida, foram realizados os **Testes de Sistema** e como tal, foi necessário gerar inputs aleatórios, semelhantes aos ficheiros de *log* disponibilizados na 1ª Fase deste Projeto. Para a geração destes novos inputs foi utilizado o sistema *QuickCheck*.

2 Testes Unitários

2.1 Geração de Testes Automáticos

2.1.1 Automáticos

Neste primeiro capítulo, de forma a fazer a geração de testes unitários para o projeto, utilizamos a plataforma **EvoSuite**.

Após gerar o algoritmo de execução da plataforma, conseguimos gerar os testes unitários para todas as classes do projeto. No entanto, por não fazer sentido, não foi feita a geração de testes para as classes *ATS* e *GUI*, visto serem classes de interfaces.

Em suma, as classes que obtiveram testes gerados foram:

- *Bike*
- *Car*
- *Client*
- *Company*
- *CustomProbabilisticDistribuction*
- *DeviantComparator*
- *Driver*
- *Helicopter*
- *MoneyComparatorC*
- *MoneyComparatorD*
- *RatingComparator*
- *Trip*
- *Umer*
- *User*
- *Van*
- *Vehicle*

No total, em todas estas classes, foram gerados **372** testes.

2.1.2 Manuais

Além dos testes gerados pelo *EvoSuit*, decidimos fazer uma analise pessoal ao código do projeto. Com a nossa experiência no ramo de criação de software, analisamos todas as classes e procuramos encontrar possíveis falhas que não tenham sido identificadas pelos testes anteriores. Concentramos a nossa atenção nas classes cujos testes foram gerados em menor quantidade, ou seja, todos os os *Comparators* e assim como, a classe *Car*.

Estes testes foram colocados nas classes geradas anteriormente e foram comentados com a devida anotação de testes manuais.

2.2 Deteção de Erros

2.2.1 Automáticos

Dada por terminada a criação de testes unitários para o projeto, prosseguimos para execução dos testes com o objetivo de encontrar erros no código fornecido para o efeito.

Dos 372 testes criados, constatamos que **3** falham. Estas falhas encontram-se nas classes **UMER** e **Company**.

No primeiro caso, ocorre no **test25** e é possível verificar que quando pedimos ao programa para realizar uma viagem através do método **newTrip** ocorre o erro especificado na imagem a seguir.

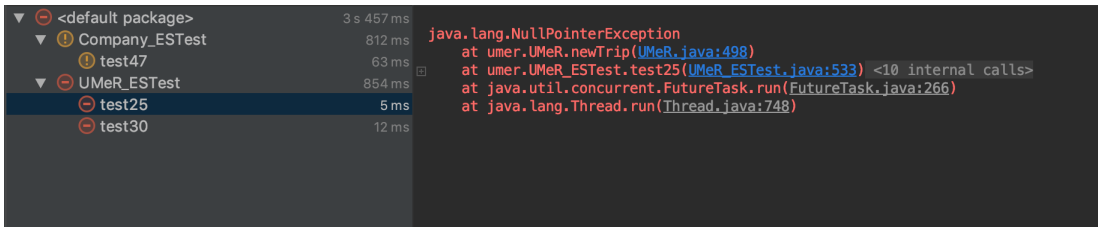

Figure 1: Erro no test25

Este erro ocorre porque o programador não protegeu os objetos que manipula do facto de poderem estar com o conteúdo vazio. Quando se tenta aceder aos campos de um objeto vazio acontece esta anomalia.

Ainda na classe *Umer*, constatamos que o **test30** provoca o mesmo tipo de erro assim que é executado. Quando o teste pede ao programa para calcular o tempo real de uma viagem através do método **realTime**, este termina a sua execução na linha 458. Mais uma vez, o programador não protegeu o acesso a objetos vazios provocando um *Null Pointer Exception*.

Na classe *Company*, foi possível detetar outro erro, desta feita no **test47**. O erro gerado foi o seguinte:

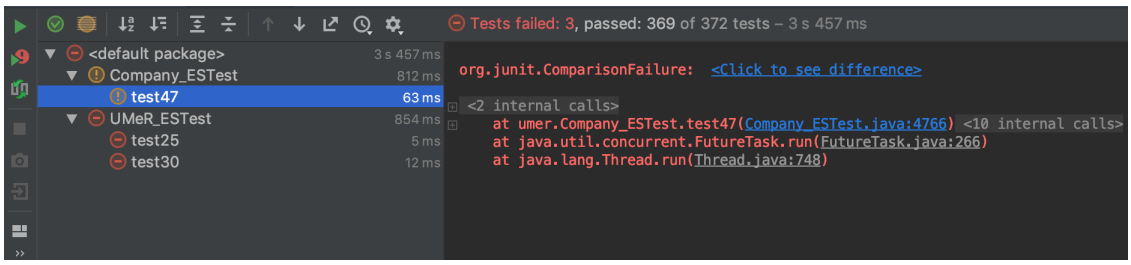

Figure 2: Erro no test47

Quando fazemos a analise pormenorizada do erro, podemos ver esta falha:

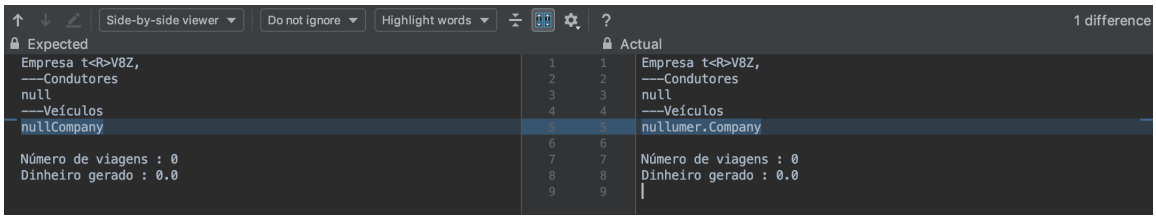

Figure 3: Falha do test47

Conforme está elaborado o teste, este esperava que o atributo a designar a empresa de um objeto *veiculo* fosse do tipo nulo. No entanto, o programa retorna a ação de tentar aceder ao campo *empresa* de um objeto nulo. Por esta razão, o programa parava a sua execução.

2.2.2 Manuais

Na execução dos testes manuais, podemos verificar que alguns executam de acordo com o previsto. No entanto, na classe *Car* conseguimos detetar um erro.

No método *calculateTraffic* observamos uma anomalia, uma vez que os valores que estavam previstos, não correspondem aos que foram calculados.

Desta forma, a função encontra-se mal definida.

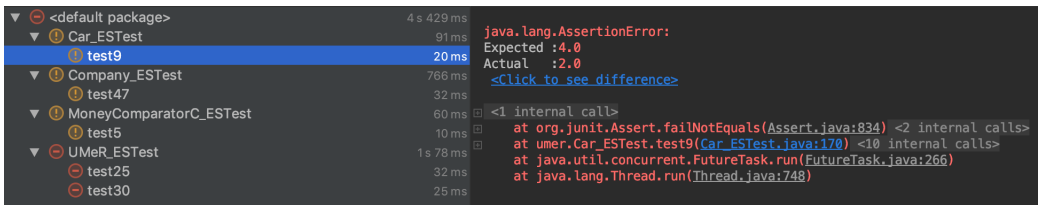

Figure 4: Falha do test9

2.3 Cobertura do código

Depois da verificação da fiabilidade do código com os testes unitários, seguimos para verificação da quantidade de código que é analisada por esses testes unitários. Para tal, recorremos à ferramenta **JUNIT** e podemos ver o resultado obtido na figura seguinte.

Element	Class, %	Method, %	Line, %
ATS	0% (0/1)	0% (0/3)	0% (0/24)
ATSBaseListener	0% (0/1)	0% (0/28)	0% (0/28)
ATSBaseVisitor	0% (0/1)	0% (0/12)	0% (0/12)
ATSLexer	0% (0/1)	0% (0/11)	0% (0/35)
ATSParser	0% (0/13)	0% (0/153)	0% (0/929)
Bike	100% (1/1)	100% (4/4)	75% (12/16)
Car	100% (1/1)	100% (5/5)	68% (15/22)
Client	100% (1/1)	100% (15/15)	90% (38/42)
Company	100% (1/1)	95% (23/24)	97% (86/88)
CustomProbabilisticDistribution	100% (1/1)	100% (3/3)	100% (13/13)
DeviationComparator	100% (1/1)	100% (1/1)	100% (4/4)
Driver	100% (1/1)	100% (24/24)	91% (64/70)
GUI	0% (0/1)	0% (0/72)	0% (0/924)
Helicopter	100% (1/1)	100% (4/4)	75% (12/16)
MoneyComparatorC	100% (1/1)	100% (1/1)	100% (4/4)
MoneyComparatorD	100% (1/1)	100% (1/1)	100% (4/4)
RatingComparator	100% (1/1)	100% (1/1)	100% (4/4)
Trip	100% (1/1)	100% (26/26)	100% (75/75)
UMeR	100% (1/1)	100% (51/51)	68% (191/279)
User	100% (1/1)	95% (22/23)	92% (58/63)
Van	100% (1/1)	100% (4/4)	75% (12/16)
Vehicle	100% (1/1)	100% (31/31)	93% (106/113)

Figure 5: Cobertura do código s/Testes Manuais

Como podemos verificar na figura 4, as classes *ATS, ATSBaseListener, ATSBaseVisitor, ATSLexer, ATSParser* e *GUI* não foram consideradas na cobertura dos testes pela razão já referida acima.

Assim podemos verificar que a percentagem de classes cobertas é de 100% e a percentagem de métodos cobertos é de 99%, em que esta última não corresponde ao pretendido 100% devido aos erros referidos em cima. Quanto à percentagem de linhas cobertas pelos testes, esta é de 89%, o que consideramos ser um bom valor para a cobertura dos testes definidos para este projeto.

Com a criação dos testes manuais, conseguimos aumentar a cobertura do código. Desta forma, resultados obtidos foram:

Element	Class, %	Method, %	Line, %
ATS	0% (0/1)	0% (0/3)	0% (0/24)
ATSBaseListener	0% (0/1)	0% (0/28)	0% (0/28)
ATSBaseVisitor	0% (0/1)	0% (0/12)	0% (0/12)
ATSLexer	0% (0/1)	0% (0/11)	0% (0/35)
ATSParser	0% (0/13)	0% (0/153)	0% (0/929)
Bike	100% (1/1)	100% (4/4)	75% (12/16)
Car	100% (1/1)	100% (5/5)	81% (18/22)
Client	100% (1/1)	100% (15/15)	90% (38/42)
Company	100% (1/1)	95% (23/24)	97% (86/88)
CustomProbabilisticDistribution	100% (1/1)	100% (3/3)	100% (13/13)
DeviationComparator	100% (1/1)	100% (1/1)	100% (4/4)
Driver	100% (1/1)	100% (24/24)	91% (64/70)
GUI	0% (0/1)	0% (0/72)	0% (0/924)
Helicopter	100% (1/1)	100% (4/4)	75% (12/16)
MoneyComparatorC	100% (1/1)	100% (1/1)	100% (4/4)
MoneyComparatorD	100% (1/1)	100% (1/1)	100% (4/4)
RatingComparator	100% (1/1)	100% (1/1)	100% (4/4)
Trip	100% (1/1)	100% (26/26)	100% (75/75)
UMeR	100% (1/1)	100% (51/51)	68% (191/2...
User	100% (1/1)	95% (22/23)	92% (58/63)
Van	100% (1/1)	100% (4/4)	75% (12/16)
Vehicle	100% (1/1)	100% (31/31)	93% (106/1...

Figure 6: Cobertura do código c/Testes Manuais

2.4 Mutações do código

Após a verificação de cobertura dos testes efetuados procedemos à execução do *PIT* para ver quais os efeitos de mutações no código deste projeto.

Antes de passar aos teste de mutação tivemos de remover todos os testes gerados que não estavam corretos.

Após a remoção dos testes que identificavam os erros do código, passamos às várias tentativas de execução do *plugin*, no qual verificamos que este não concluía, ou seja, o programa entrava em ciclo infinito e não terminava a sua execução. Assim, podemos ver na figura 5 os erros na execução e como tal, existe a possibilidade de os testes que foram criados terem problemas.

```
21:18:13 PIT >> WARNING : Minion exited abnormally due to TIMED_OUT <1 internal call> <1 internal call>
21:18:22 PIT >> WARNING : Minion exited abnormally due to TIMED_OUT <1 internal call>
21:18:31 PIT >> WARNING : Minion exited abnormally due to TIMED_OUT
21:18:40 PIT >> WARNING : Minion exited abnormally due to TIMED_OUT <1 internal call> <1 internal call>
21:18:49 PIT >> WARNING : Minion exited abnormally due to TIMED_OUT <1 internal call>
21:18:58 PIT >> WARNING : Minion exited abnormally due to TIMED_OUT <1 internal call>
21:19:07 PIT >> WARNING : Minion exited abnormally due to TIMED_OUT <1 internal call>
21:19:15 PIT >> WARNING : Minion exited abnormally due to TIMED_OUT
21:19:24 PIT >> WARNING : Minion exited abnormally due to TIMED_OUT <1 internal call>
21:19:33 PIT >> WARNING : Minion exited abnormally due to TIMED_OUT <1 internal call>
21:19:42 PIT >> WARNING : Minion exited abnormally due to TIMED_OUT <1 internal call>

Process finished with exit code 137 (interrupted by signal 9: SIGKILL)
Open report in browser
```

Figure 7: Execução do PIT

3 Testes de Sistema

3.1 Definição de Geradores

Para a criação dos **Testes de Sistema** foi utilizado o *template* de *QuickCheck* para geração de ficheiros de *input*, para tal, foi definido um gerador global **genLogs**, que contém todos os geradores necessários para a criação do ficheiro *log*. Sendo assim foram definidos os seguintes geradores:

- **genLogs**: Este gerador faz uso das funções *replicateM* e *frequency* para gerar os vários inputs pretendidos, chamando os vários geradores que foram definidos.
- **genRegistarCondutor**: Primeiramente definimos um nome e apelido através da lista de nomes e apelidos fornecidos no *template*. De seguida, foi definido o email, recorrendo ao nome de utilizador, e também uma palavra-passe, sendo que estes foram guardados no estado, sob o nome *stcondutores*, para posteriormente serem utilizados. Foi ainda definida uma localidade, um código postal, uma data de nascimento e um identificador de táxi. Foi ainda guardado o email do condutor e identificador de táxi no estado sob o nome *sttaxis*. Por fim, retornamos o que foi definido com a inicialização *registar condutor*.
- **genRegistarCliente**: Análogo ao **genRegistarCondutor**, com a particularidade de definirmos a posição do cliente e não atribuirmos um identificador de táxi, nem este ser guardado no estado. Por fim, retornamos o que foi definido com a inicialização *registar cliente*.
- **genRegistarEmpresa**: Inicialmente, definimos um nome para a empresa e guardamos no estado, sob o nome *stempresas*. De seguida definimos um palavra-passe e retornamos o que foi definido com a inicialização *registar empresa*.
- **genCliente**: Primeiramente, começamos por definir o login do cliente através da função **genLoginCliente**, em que esta acede à lista de clientes presentes no estado e retorna o login com esse cliente. De seguida, definimos um número aleatório e replicamos esse número de vezes a função **genSolicitar**, onde esta gera coordenadas aleatórias. Por fim, retornamos o login, os vários solicitar e o logout.
- **genCondutor**: Inicialmente, começamos por definir o login do condutor através da função **genLoginCondutor**, em que esta acede à lista de condutores presentes no estado e retorna o login com esse condutor. De seguida, definimos um número aleatório e replicamos esse número de vezes as funções **genViajar, genRecusarViagem** com o uso do *frequency*. Por fim, retornamos o login, os vários genViajar, genRecusarViagem e o logout.

- **genRegistarCarrinhaCondutor**: Inicialmente, definimos um condutor através da lista de condutores presente no estado, depois definimos um fator de fiabilidade, uma matricula e as coordenadas. Por fim, retornamos o que foi definido com a inicialização *registar carrinha*.
- **genRegistarCarrinhaEmpresa**: Análogo ao **genRegistarCarrinhaCondutor** mas com a particularidade de agora definirmos a empresa, em vez do condutor, em que a carrinha será registada.
- **genRegistarCarroCondutor**: Análogo ao **genRegistarCarrinhaCondutor** mas com a única diferença de agora retornamos o que foi definido com a inicialização *registar carro*.
- **genRegistarCarroEmpresa**: Análogo ao **genRegistarCarrinhaEmpresa** mas com a única diferença de agora retornamos o que foi definido com a inicialização *registar carro*.
- **genRegistarMotaCondutor**: Análogo ao **genRegistarCarrinhaCondutor** mas com a única diferença de agora retornamos o que foi definido com a inicialização *registar mota*.
- **genRegistarMotaEmpresa**: Análogo ao **genRegistarCarrinhaEmpresa** mas com a única diferença de agora retornamos o que foi definido com a inicialização *registar mota*.
- **genRegistarHelicopteroCondutor**: Análogo ao **genRegistarCarrinhaCondutor** mas com a única diferença de agora retornamos o que foi definido com a inicialização *registar helicoptero*.
- **genRegistarHelicopteroEmpresa**: Análogo ao **genRegistarCarrinhaEmpresa** mas com a única diferença de agora retornamos o que foi definido com a inicialização *registar helicoptero*.

3.2 Exemplo de um ficheiro de input

Nesta secção, deixamos um pequeno exemplo do ficheiro *log* gerado, através dos geradores acima descritos. Para a geração deste ficheiro *log* basta apenas correr a makefile na diretoria do gerador.

```
registar mota "89-UM-48" 76 (8.9,4.8) empresa "Empresa 84" ;

registar condutor "Ianis_Pereira@mail.google.com" "Ianis Pereira" "8aFS176xk"
"Largo de Sao Jose, 1888, 2525-028 Atouguia da Baleia" 1932-6-23 289 ;

registar carrinha "33-PX-20" 31 (3.3,2.0) "Alice_Jesus@mail.google.com" ;

registar helicoptero "77-PE-93" 52 (7.7,9.3) empresa "Empresa 71" ;

registar condutor "Lara_Pinho@mail.google.com" "Lara Pinho" "fLcDcI97S"
"Rua das Pereiras, 1626, 4615-409 Macieira da Lixa" 1988-7-13 91 ;

registar carro "24-WR-43" 82 (2.4,4.3) "Iriana_Valente@hotmail.com" ;

registar carrinha "08-EN-93" 83 (0.8,9.3) "Dolique_Almeida@mail.google.com" ;

login "Aquila_Pinheiro@hotmail.com" "vkleW1P7n" ;
solicitar (9.2,55.4) ;
solicitar (75.0,16.2) ;
solicitar (82.0,24.9) ;
logout ;

registar carro "84-YI-62" 90 (8.4,6.2) "Violinda_Raposo@gmail.com" ;

login "Toledo_Domingues@sapo.pt" "Y7VU8N8Ma" ;
viajar;
viajar;
viajar;
recusar viagem;
viajar;
logout ;

registar carro "71-IM-58" 36 (7.1,5.8) "Isolino_Teixeira@gmail.com" ;
```

4 Conclusão

Ao longo deste trabalho, fomos estudando diferentes formas de efetuar testes a um projeto que nos é fornecido com o intuito de verificar a sua capacidade e fiabilidade.

Na primeira parte, passamos pela parte de adaptação ás ferramentas de trabalho. Feito todo o *setup* e instalação de todos os *plugins* e dependências do projeto *JAVA*, prosseguimos para a implementação das capacidades das mesmas. Após a criação dos testes unitários, podemos constatar que o resultado obtido é bastante bom, visto que conseguimos detetar algumas falhas do projeto em questão, assim como a cobertura do mesmo é bastante satisfatória, rondando os 90% de linhas código testadas.

De seguida, construimos os testes manuais nos quais foi possível identificar mais erros ao longo do projeto, e ao mesmo tempo, aumentar a cobertura do mesmo.

Apesar da incapacidade da execução do *PIT*, não rotulamos como negativo os testes gerados, sendo que o problema pode estar no próprio código fonte do programa.

Na segunda parte do trabalho, efetuamos os testes ao sistema, gerando ficheiros de input aleatórios. Apesar do projeto estudado estar elaborado na linguagem *Java*, optamos por desenvolver o gerador em *Haskell* com o auxilio da ferramenta *QuickCheck*.

Começamos por elaborar pequenos geradores para os pequenos atributos de cada ação do ficheiro de *log*. Posto isto, construimos geradores maiores para cada ação terminando com o gerador final que agrupava estes todos. No fim, os ficheiros gerados, foram testados com a gramática construida na etapa anterior e constamos que estes eram criados corretamente.

Apesar do trabalho desenvolvido, achamos que futuramente o código fornecido para análise pode ser ainda mais escrutinado, uma vez que, dificilmente conseguimos analisar todos os erros que possivelmente existirão no projeto.