Análise e Teste de Software
Trabalho Prático

-

Mestrado em Engenharia Informática
Universidade do Minho
Relatório

Grupo nº3

PG41091	Nelson José Dias Teixeira
PG41081	José Alberto Martins Boticas
PG41094	Pedro Rafael Paiva Moura
A80499	Moisés Manuel Borba Roriz Ramires

25 de Novembro de 2019

Resumo

No ano lectivo 2018/2019, no contexto da disciplina de Programação Orientada a Objectos (POO) leccionada no Departamento de Informática da Universidade do Minho, os alunos tiveram de desenvolver em grupo uma aplicação Java, denominada por *UmCarroJá*, para gerir um serviço de aluguer de veículos particulares pela internet. No contexto da disciplina de Análise e Teste de Software (ATS) pretende-se que neste projeto se apliquem técnicas de análise e teste de software, estudadas nas aulas, de modo a analisar a qualidade de duas das soluções desenvolvidas pelos alunos de POO.

Conteúdo

Capítulo 1

Introdução

Neste projeto foi-nos proposto a realização de várias tarefas de forma a analisar a qualidade das duas soluções desenvolvidas pelos alunos de POO no ano lectivo de 2018/2019. Entre estas tarefas destacam-se as seguintes:

1. Analisar a qualidade do código fonte dos sistemas de *software*. Nesta análise identificam-se *bad smells* no código fonte e o seu *technical debt*;
2. Aplicar *refactorings* de modo a eliminar os *bad smells* encontrados e deste modo reduzir (se possível eliminar) o *technical debt*;
3. Testar o *software* de modo a ter mais garantias que ele cumpre os requisitos do enunciado da aplicação *UmCarroJá*;
4. Gerar *inputs* aleatórios para a aplicação *UmCarroJá* que simulem execuções reais (tal como foi fornecido em POO);
5. Analisar a performance (tempo de execução e consumo de energia) das versões iniciais do *software* (i.e., com *smells*) e as obtidas depois de eliminados os *smells*.

Os cinco pontos mencionados acima foram agrupados em quatro tarefas finais, cada uma das quais com uma percentagem na avaliação final do trabalho prático. As abordagens tomadas pelo grupo sobre cada uma destas tarefas serão expostas nos capítulos seguintes deste relatório. De salientar que também existem tarefas extras que complementam cada uma das tarefas referidas anteriormente.

Capítulo 2

Análise e Especificação

2.1 Tarefa 1 - Qualidade do código fonte

Nesta etapa, tal como o nome indica, será feita a análise da qualidade do código fonte da aplicação *UmCarroJá* desenvolvido pelos alunos de POO. Como tal, através da ferramenta *Sonarqube*, serão indicados o número de erros no código (*bugs*), vulnerabilidades, *code smells* e o respetivo *technical debt*. Para além destas, existe uma **tarefa extra** que consiste em definir regras adicionais na ferramenta *Sonarqube* para encontrar *red smells* (ou qualquer outro *smell* não suportado pelo mesmo) na aplicação desenvolvida.

2.1.1 Versão 1 - *demo1*

Na primeira versão desenvolvida pelos alunos de POO, *demo1*, foi possível observar alguns erros no código fonte e bastantes *code smells*. Apresenta-se de seguida, por categorias, a análise qualitativa desta mesma implementação.

2.1.1.1 Fiabilidade

Nesta secção da análise qualitativa da aplicação desenvolvida observam-se e identificam-se unicamente os erros (*bugs*) presentes no código fonte. Como tal, após verificar a informação existente na ferramenta *Sonarqube*, os elementos deste grupo depararam-se, essencialmente, com quatro tipos de erros. Entre eles destacam-se:

1. a implementação do método *equals()* numa determinada classe sobrepõe a predefinida, pelo que também deve ser codificado o método *hashCode()*;
2. o método *equals()* presente numa determinada classe necessita de ser sobreposto à implementação predefinida ou, simplesmente, renomeado;
3. o objeto *Random* presente numa determinada classe deve ser reutilizado;
4. o objeto *ObjectOutputStream* deve ser fechado através de uma clausula *try-catch-finally*.

Na totalidade existem cerca de catorze erros no código fonte. Quanto à severidade destes erros, existem dois de tipo *blocker*, dois de tipo *critical*, um de tipo *major* e 9 de tipo *minor*.

De salientar que, apesar da existência de alguns erros presentes nesta implementação, estes são facilmente corrigíveis.

2.1.1.2 Segurança

Ao nível da segurança, esta implementação apresenta apenas uma vulnerabilidade cujo grau de severidade é do tipo *minor*. A ferramenta *Sonarqube*, por forma a eliminar esta mesma, sugere encapsular a amostragem de um determinado erro através de um objeto *LOGGER*.

2.1.1.3 Manutenção

2.1.1.4 Cobertura

2.1.1.5 Duplicação de código

2.1.2 Versão 2 - *demo2*

2.1.2.1 Fiabilidade

Nesta secção da análise qualitativa da aplicação desenvolvida observam-se os erros no código fonte.

2.1.2.2 Segurança

2.1.2.3 Manutenção

2.1.2.4 Cobertura

2.1.2.5 Duplicação de código

2.2 Tarefa 2 - *Refactoring*

2.2.1 Versão 1 - *demo1*

2.2.2 Versão 2 - *demo2*

2.3 Tarefa 3 - Teste da aplicação

2.3.1 Versão 1 - *demo1*

2.3.2 Versão 2 - *demo2*

2.4 Tarefa 4 - Análise de desempenho

2.4.1 Versão 1 - *demo1*

2.4.2 Versão 2 - *demo2*

Capítulo 3

Conclusão

Apêndice A

Observações

Durante a análise da primeira tarefa associada a este trabalho prático observaram-se, na ferramenta ***Sonarqube***, variados tipos de severidade de erros. Como tal, apresentam-se de seguida, de forma mais detalhada, os mesmos:

- ***blocker***:
  Erro com alta probabilidade de afetar o comportamento da aplicação em produção. O código deve ser corrigido imediatamente.
- ***critical***:
  Erro com baixa probabilidade de afetar o comportamento da aplicação em produção ou um problema que representa uma falha de segurança. O código deve ser examinado imediatamente.
- ***major***:
  Falha que afeta a qualidade do código e que pode ter um impacto significativo na produtividade do programador: blocos duplicados, parâmetros não utilizados, entre outros.
- ***minor***:
  Falha que afeta a qualidade do código e que pode ter um impacto minorativo na produtividade do programador: linhas de código longas, entre outros.