Testes de caixa preta

Test design Specification

Version 2.0

24/11/2019

|  |  |
| --- | --- |
| Grupo\_TP1\_Q: | |
| Rui Bessa | 8150521 |
| Miguel Guimarães | 8150520 |
| José Fernando | 8150498 |

**Histórico de Versões**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version #** | **Implemented**  **By** | **Revision**  **Date** | **Approved**  **By** | **Approval**  **Date** | **Reason** |
| 1.0 | *Grupo TP1 Q* | 12/11/2019 | *Grupo TP1 Q* | *Grupo* *TP1 Q* | 1º versão |
| 2.0 | *Grupo TP1 Q* | 24/11/2019 | *Grupo TP1 Q* | *Grupo TP1 Q* | Entrega |

Tabela de Conteúdos

[1. Introdução 4](#_Toc25524956)

[1.1. Identificador do documento 4](#_Toc25524957)

[1.2. Âmbito 4](#_Toc25524958)

[1.3. Repositório 4](#_Toc25524959)

[1.4. Referências 4](#_Toc25524960)

[1.5 Glossário 5](#_Toc25524961)

[2. Features/Itens a testar 6](#_Toc25524962)

[3. Detalhes da abordagem aos testes 8](#_Toc25524963)

[4. Identificação dos Testes 9](#_Toc25524964)

[5. Critérios de passagem ou falha das features 16](#_Toc25524965)

# 1. Introdução

## 1.1. Identificador do documento

Test\_Case\_Specification\_TP1\_Q

## 1.2. Âmbito

Este relatório foi elaborado para a disciplina de Engenharia de Software II, e tem como objetivo apresentar e explicar os testes de software, caixa preta, realizados pelo grupo “TP1 Q” no primeiro trabalho prático. Pretende-se, também, apresentar a documentação detalhada de todas as definições, planos, abordagens efetuadas e os resultados retornados.

O trabalho desenvolvido passou por realizar os testes unitários de software fornecido pelo docente da disciplina. Este software, consiste na realização de um teste, ou seja, adicionar perguntas a uma coleção (Test) para formar um conjunto de questões. Estas podem ser de três tipos, questões numéricas, questões de verdadeiro ou falso e questões de múltipla escolha.

Para a realização dos testes, foi necessário planear e testar com testes de caixa preta, equivalência e limites, para os métodos addQuestion, numberQuestions, removeQuestion, isComplete, correctAnswers, incorrectAnswers, correctAnswerPecentage, incorrectAnswerPecentage, evaluateAnswer e answer.

Serão de seguida, apresentados todos os dados para o desenvolvimento dos testes de software, tais como, inputs para cada método, outputs esperados e os obtidos, e uma avaliação feita pelos membros do grupo e em alguns casos, acrescentadas observações para possíveis resoluções ou melhorias de problemas encontrados.

## 1.3. Repositório

A hiperligação para o repositório do gitlab, onde foi efetuado o desenvolvimento deste trabalho, é o seguinte:

<http://gitlab.estg.ipp.pt/8150521/esii_tp1_q.git>

## 1.4. Referências

* Slides disponibilizados na plataforma moodle da unidade curricular ESII;
* IEEE Standard for Software Unit Testing;
* S.W.E.B.O.K.;

## 1.5 Glossário

ECP (equivalency class partitioning) – este método procura definir um caso de teste com o intuito de descobrir classes com erros, reduzindo assim o número tota de casos de teste a serem desenvolvidos.

Uma classe de equivalência representa um conjunto de estados válidos para condições de entrada.

BVA (boundary value analysis) – esta análise leva à escolha de casos de teste que põem à prova os valores limites. Serve de complemento aos testes de partição de equivalência. Em vez de selecionar qualquer elemento de uma classe de equivalência, o BVA leva à seleção de casos de testes nos extremos da classe.

## 2. Features/Itens a testar

|  |  |  |  |
| --- | --- | --- | --- |
| **Item a testar** | **Descrição** | **Requisitos** | **Responsabilidade** |
| addQuestion | Adicionar uma questão ao teste | Existência de uma coleção de perguntas previamente instanciada | Rui / Fernando |
| removeQuestion | Remover questão de uma determinada posição no teste | Existência de uma coleção de perguntas previamente instanciada | Rui |
| numberQuestions | Retorno o número de questões existentes no teste | Existência de uma coleção de perguntas previamente instanciada | Fernando/ Miguel |
| isComplete | Retorna o valor true se a questão estiver todas respondidas, se não retorna false. | Existência de uma coleção de perguntas previamente instanciada | Fernando / Rui |
| correctAnswers | Retorna uma coleção de respostas corretas do teste | Existência de uma coleção de perguntas previamente instanciada | Miguel / Rui |
| incorrectAnswers | Retorna uma coleção de respostas incorretas do teste | Existência de uma coleção de perguntas previamente instanciada | Miguel |
| correctAnswersPercentage | Retorna uma percentagem de número de questões corretas | Existência de uma coleção de perguntas previamente instanciada | Fernando |
| incorrectAnswersPercentage | Retorna uma percentagem de número de questões incorretas | Existência de uma coleção de perguntas previamente instanciada | Fernando |
| evaluteAnswer | Retorna o valor true, se a resposta do Utilizador for igual a respota correta da questão, se não, retorna false | correct\_answer != null e número de caracteres de correct\_answer > 0 | Miguel |
| answer | Método que atribui a resposta do utilizador à questão | correct\_answer != null e número de caracteres de correct\_answer > 0 | Rui |

## **3. Detalhes da abordagem aos testes**

De forma a assegurar a deteção de erros de software, com o objetivo de alcançar a qualidade de software, seguimos a abordagem de deteção de intervalos e de casos especiais, bem como os limites dos intervalos, através de técnicas conhecidas como "Input Domain-Based Techniques".

Mais concretamente, através das técnicas conhecidas como "Equivalence Class Partitioning" (ECP) e "Boundary Value Analysis" (BVA) para posteriormente detetar casos de teste suficientes que cubram todas os intervalos detetados através de testes JUnit.

Em primeiro lugar realizamos para cada método de forma isolada a tabela ECP para identificação dos vários intervalos possíveis e não possíveis, identificados através de classes válidas e classes inválidas.

Após a tabela ECP, realizamos a tabela BVA para identificar os limites dos vários intervalos identificados anteriormente, de realçar, que é sempre testado o limite inferior, limite inferior menos um, limite inferior mais um, limite máximo, limite máximo menos um e limite máximo mais um. Para além dos limites dos intervalos, na tabela BVA incluimos também os chamados casos especiais, como por exemplo string vazia ("") e string nula (null).

Por último, mas não menos importante, será elaborada uma tabela de testes para cada teste realizado, com os inputs introduzidos, outputs esperados, outputs obtidos e indicação se o teste passou ou não. Bem como observações pelos elementos do grupo em alguns casos em que o teste não passou, para por exemplo, indicar qual foi o motivo da falha.

Com esta abordagem, pretendemos detetar o maior número possível de erros com o menor número de casos de teste.

Estes testes serão executados no IDE IntelliJ usando a ferramenta de testes unitários JUnit, assim como todas as dependências utilizando o Gradle para assegurar que todos os elementos do grupo usam as mesmas versões.

Ao longo do desenvolvimento dos testes foi usado um repositório no GitLab para facilitar o trabalho em grupo e ter registado e controlado as várias versões e alterações do projeto.

## **4. Identificação dos Testes**

De seguida são ilustradas as várias tabelas ECP elaboradas pelo grupo, foi uma para cada método, assim como as respetivas tabelas BVA, que foi uma para cada tabela ECP.

Devido a algumas imagens poderem não estar bastante percetiveis, enviamos em anexo as respetivas tabelas em formato excel.









































## **5. Critérios de passagem ou falha das features**

Como critérios de passagem para os testes definidos anteriormente, considerando se o resultado esperado fosse igual ao output, o teste passava, caso contrario, não passava, assumindo uma falha no método e consequentemente irá ser reportado, para que a correção do método seja realizada. Caso contrário poderá levar a diversas consequências como falha de segurança e performance, assim como erros inesperados.