VVS Report Assignment 1

Verificação e Validação de Software

2019/20

André Nicolau 47880

Marta Carvalho 48349

Pedro Costa 55115

Conteúdo

1 Introdução

Este relatório enquadra-se na realização do projecto da disciplina Verificação e Validação de Software cujo objectivo principal é fazer a preparação de um conjunto de testes unitários com o intuito de verificar a implementação de uma estrutura de dados Ternary Search Trie(TST). O objectivo da realização deste projecto assenta na aplicação dos conhecimentos adquiridos na disciplina supra referida.
O relatório foi dividido em duas partes. Na primeira são apresentados os testes realizados para cada critério aplicado. Apesar de um dos critérios, instruction coverage, ser relativo a toda a classe TST, todos os outros critérios são apenas aplicados num dos métodos da mesma classe, o longestprefixof() ou put().Na segunda parte é abordado o PIT, utilizado para validar os testes unitários implementados.

2 Coverage Criteria

Tendo em vista a aplicação de critérios na base de coverage criteria, aplicámos vários testes de acordo com a respectiva cobertura e requisitos inerentes. Nestes testes foram incluídos os testes realizados a todos os métodos públicos presentes na classe TST.

2.1 Instruction Coverage

De forma a cobrir todas as linhas de código de todos os métodos públicos da classe TST, faremos nesta secção vários testes que estarão divididos em secções, sendo que cada secção representa um método.

2.1.1 size()

Código 1: size()

```
public int size() {
    return n; /* I1 */
}
```

Test case	Expected Values	IC
test1	4	I1

2.1.2 contains()

Código 2: contains()

```
public boolean contains(String key) {
if (key == null) /* I1 */
    throw new IllegalArgumentException
    ("argument to contains() is null"); /* I2 */
return get(key) != null; /* I3 */
}
```

Test Case	Test Case Values(key)	Expected Values	IC
test1	null	IAE	I1, I2
test2	("odado")	true	I1, I3

2.1.3 get()

Código 3: get()

```
public T get(String key) {
    if (key == null) /* I1 */
        throw new IllegalArgumentException
        ("calls get() with null argument"); /*I2*/
    if (key.length() == 0) /* I3 */
            throw new IllegalArgumentException
            ("key must have length >= 1"); /*I4*/
    Node<T> x = get(root, key, 0); /* I5 */
    if (x == null) /* I6 */
            return null; /* I7 */
    return x.val; /* I8 */
}
```

Test case	Test case values	Expected Values	IC
test1	null	IAE	I1, I2
test2	(”)	IAE	I1, I3, I4
test3	("dado")	3	I1, I3, I5, I6, I8
test4	("adeus")	null	I1, I3, I5, I6, I7

2.1.4 put()

Código 4: put()

```
public void put(String key, T val) {
    if (key == null) /* I1 */
        throw new IllegalArgumentException
        ("calls put() with null key"); /* I2 */
    if (!contains(key)) /* I3 */
            n++; /* I4 */
    root = put(root, key, val, 0); /* I5 */
}
```

Test case	Test case values (key)	Expected values	IC
test1	(null, 2)	IAE	I1, I2
test2	("bye", 2)	2	I1, I3, I4, I5

2.1.5 longestPrefixOf()

Código 5: longestPrefixOf()

```
    public String longestPrefixOf(String query) {
        if (query == null) /* I1 */
            throw new IllegalArgumentException
            ("calls␣longestPrefixOf()
␣␣␣␣␣␣␣␣␣␣␣␣␣with␣null␣argument"); /* I2 */
        if (query.length() == 0) /* I3 */
                return null; /* I4 */
        int length = 0; /* I5 */
        Node<T> x = root; /* I6 */
        int i = 0; /* I7 */
        while (x != null /* I8 */ &&
        i < query.length() /* I9 */) {
            char c = query.charAt(i); /* I10 */
            if (c < x.c) /* I11 */
                x = x.left; /* I12 */
            else if (c > x.c) /* I13 */
                x = x.right; /* I14 */
            else {
                i++; /* I15 */
                if (x.val != null) /* I16 */
                        length = i; /* I17 */
                x = x.mid; /* I18 */
            }
        }
        return query.substring(0, length); /* I19 */
    }
```

Test	Test Case Values (query)	Expected Values	IC
test1	null	IAE	I1, I2
test2	(”)	null	I1, I3, I4
test3	("dardo")	"dardo"	I1, I3, I5, I6, I7, I8, I9, I10, I11, I12, I13 ,I14, I15, I16, I17, I18, I19

2.1.6 keys()

Código 6: keys()

```
    public Iterable<String> keys() {
        Queue<String> queue =
        new LinkedList<>(); /* I1 */
        collect(root, new StringBuilder(),
        queue); /* I2 */
        return queue; /* I3 */
    }
```

Test case	Expected values	IC
test1	("ola","odado","dado","dador","dardo","palha","pala","porta")	I1, I2, I3

2.1.7 keysWithPrefix()

Código 7: keysWithPrefix()

```
    public Iterable<String> keysWithPrefix
    (String prefix) {
        if (prefix == null) /* I1 */
            throw new IllegalArgumentException
            ("calls␣keysWithPrefix()
␣␣␣␣␣␣␣␣␣␣␣␣with␣null␣argument"); /* I2 */
        Queue<String> queue =
        new LinkedList<>(); /* I3 */
        Node<T> x = get(root, prefix, 0); /* I4 */
        if (x == null) /* I5 */
                return queue; /* I6 */
        if (x.val != null) /* I7 */
                queue.add(prefix); /* I8 */
        collect(x.mid, new StringBuilder(prefix),
        queue); /* I9 */
        return queue; /* I10 */
    }
```

Test Case	Test case Values (prefix)	Expected values	IC
test1	null	IAE	I1, I2
test2	("ol")	”	I1, I3, I4, I5, I7, I9, I10
test3	("aka")	[]	I1, I3, I4, I5, I6
test4	("dado")	"dado","dador","dadora"	I1, I3, I4, I5, I7, I8, I9, I10

2.1.8 keysThatMatch()

Código 8: keysThatMatch()

```
public Iterable<String>
keysThatMatch(String pattern) {
    Queue<String> queue =
    new LinkedList<>(); /* I1 */
    collect(root, new StringBuilder(), 0,
    pattern, queue); /* I2 */
    return queue; /* I3 */
}
```

Test Case	Test Case Values (pattern)	Expected Values	IC
test1	("d...")	"dado", "data", "dita"	I1, I2, I3

2.1.9 equals()

Código 9: equals()

```
@Override
    public boolean equals(Object obj) {
            if (this == obj) /* I1 */
                    return true; /* I2 */
            if (obj == null) /* I3 */
                    return false; /* I4 */
            if (getClass() !=
            obj.getClass()) /* I5 */
                    return false; /* I6 */

            TST other = (TST) obj; /* I7 */

            if (this.size() !=
            other.size()) { /* I8 */
                    return false; /* I9 */
            }

            Iterator<String> s = this.keys()
            .iterator(); /* I10 */
            String next; /* I11 */
            while (s.hasNext()) { /* I12 */
                    next = s.next(); /* I13 */
                    if (!other
                    .contains(next) /* I14 */ &&
                    !other.get(next)
                    .equals(get(next))){ /* I15 */
                     return false; /* I16 */
                    }
            }

            return true; /* I17 */
    }
```

Test Case	Test Case Values (obj)	Expected Values	IC
test1	("ola", "odado", "dado")	true	I1 I2
test2	("ola", "odado", "dado")	false	I1, I2, I3
test3	("ola", "odado", "dado", "key")	false	I1, I3, I5, I6
test4	("ola", "odado", "dado")	false	I1, I3, I5, I7, I8, I9
test5	("ola", "odado", "dado")	false	I1, I3, I5, I7, I8, I10, I11, I12, I13, I14, I15, I16
test6	("ola", "odado", "dado")	true	I1, I3, I5, I7, I8, I10, I11, I12, I13, I14, I15, I17

2.1.10 delete()

Para o teste desta função usamos o método get() para comprovar se a chave dada como input (e o respetivo valor) existem ou não na tabela.

Código 10: delete()

```
public void delete(String key) {

    if (key == null /* I1 */ ||
    key.length() == 0 /* I2 */) {
            throw new
            IllegalArgumentException(); /* I3 */
    }

    if (!contains(key)) { /* I4 */
            throw new
            IllegalArgumentException
            ("Nao existe para apagar"); /* I5 */
    }

    n--; /* I6 */

    deleteAux(root,key,0); /* I7 */

}
```

Test Case	Test Case Values (key)	Expected Values	IC
test1	(”)	IAE	I1, I2, I3
test2	("zeca")	IAE	I1, I2, I4, I5
test3	("oleado")	null	I1, I2, I4, I6, I7

2.2 Graph Coverage na função longestPrefixOf()

Para a aplicação de critérios tendo como base os grafos, relativo ao método longestPrefixOf(), começaremos por construir o grafo correspondente. De forma a garantir que o grafo não tornará uma dimensão indesejada, faremos os possíveis para a manter reduzida, agregando operações homónimas no mesmo modo.[1].

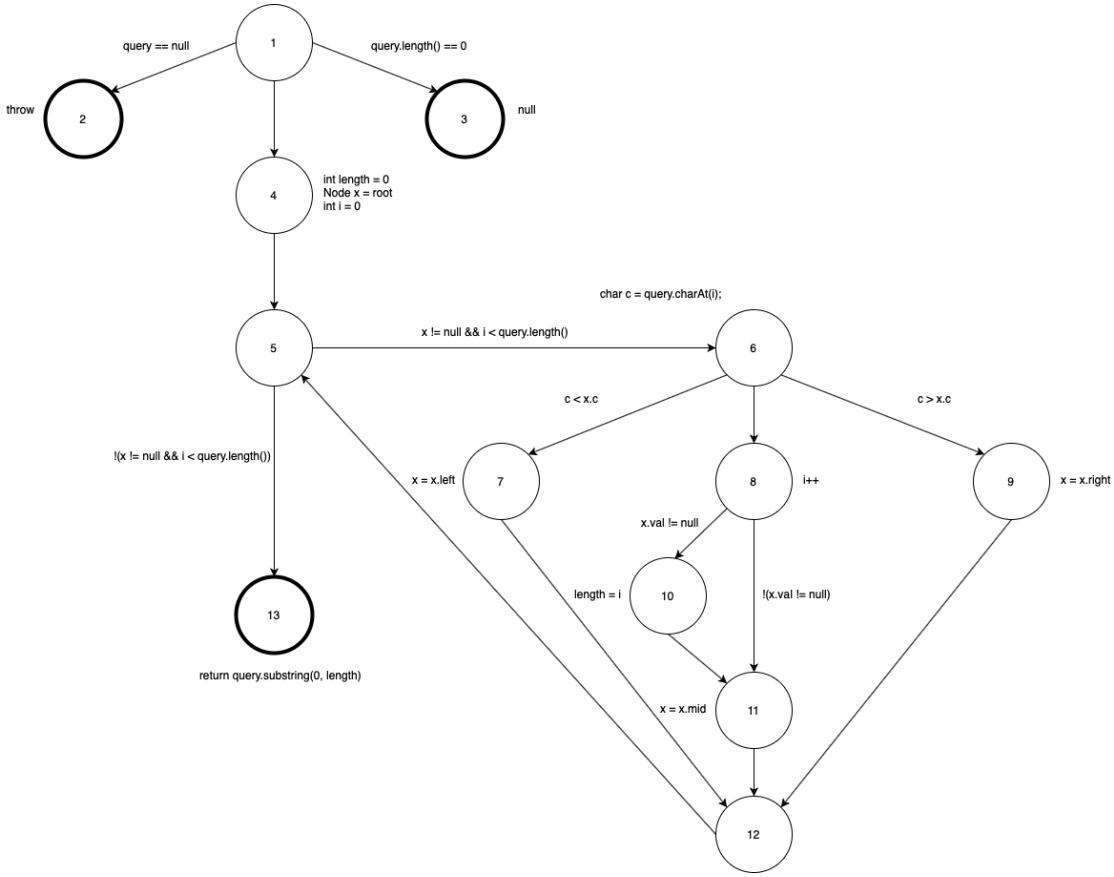

Figura 1: Grafo que foi considerada para a realização dos testes do longestPrefixOf()

Neste grafo podemos verificar que temos três nós terminais (2, 3 e 13) e um nó inicial (1). Os nós terminais correspondem às instruções *return* e *thows*.

Para executar os testes foram adicionados vários dados para se poder proceder aos testes. A escolha dos dados foi considerada para que a estrutura fique suficientemente rica em casos que divergem entre si com o objectivo de serem feitos testes mais benéficos.

2.2.1 Edge Coverage

Neste critério, os testes produzidos têm como objectivo cobrir todas as arestas do grafo. Como tal, foi construída uma tabela para identificar os casos de teste.

Test	Test Case Values	Expected Value	Test Path
test1	(null)	IAE	[1,2]
test2	("ola")	null	[1,3]
test3	("ditad")	"dita"	[1,4,5,6,7,12,5,6,8,11,12,5,6,9,12,5,6,8,11,12,5,6,8,11,12,5,6,8,10,11,12,5,6,8,11,12,5,13]

2.2.2 Prime Path Coverage

Em primeiro lugar, terá de se identificar todos os prime paths presentes no grafo em análise. Os requisitos encontrados são os seguintes:

[1,2]	[1,3]	[1,4,5,13]	[7,12,5,6,9]	[12,5,6,9,12]
[12,5,6,7,12]	[9,12,5,6,7]	[9,12,5,6,9]	[7,12,5,6,7]	[5,6,7,12,5]
[6,9,12,5,6]	[6,9,12,5,13]	[5,6,9,12,5]	[6,7,12,5,13]	[6,7,12,5,6]
[8,11,12,5,6,9]	[8,11,12,5,6,7]	[8,11,12,5,6,8]	[12,5,6,8,11,12]	[9,12,5,6,8,11]
[11,12,5,6,8,11]	[1,4,5,6,9,12]	[1,4,5,6,7,12]	[7,12,5,6,8,11]	[6,8,11,12,5,13]
[5,6,8,11,12,5]	[6,8,11,12,5,6]	[8,10,11,12,5,6,9]	[8,10,11,12,5,6,8]	[10,11,12,5,6,8,10]
[12,5,6,8,10,11,12]	[9,12,5,6,8,10,11]	[11,12,5,6,8,10,11]	[8,10,11,12,5,6,7]	[1,4,5,6,8,11,12]
[5,6,8,10,11,12,5]	[7,12,5,6,8,10,11]	[6,8,10,11,12,5,13]	[6,8,10,11,12,5,6]	[1,4,5,6,8,10,11,12]

Em segundo lugar, o objectivo é identificar os testes que irão testar cada prime path. Sabendo que cada teste pode englobar mais do que um path.
Os testes a serem realizados são os seguintes:

Test	Test Case Values	Expected Value	Test Paths	Prime Paths Covered
test1	(null)	IAE	[1,2]	[1,2]
test2	(”)	null	[1,3]	[1,3]
test3*	("abc")	”	[1,4,5,13]	[1,4,5,13]
test4	("zoo")	,,	[1,4,5,6,9,12,5,6,9,12,5,13]	[1,4,5,6,9,12], [5,6,9,12,5], [6,9,12,5,13], [6,9,12,5,6],[9,12,5,6,9], [12,5,6,9,12]
test5	("ol")	”	[1,4,5,6,8,11,12,5,13]	[1,4,5,6,8,11,12], [5,6,8,11,12,5], [6,8,11,12,5,13]
test6	("pa")	,,	[1,4,5,6,9,12,5,6,8,11,12,5,13]	[5,6,8,11,12,5], [6,8,11,12,5,13], [1,4,5,6,9,12], [9,12,5,6,8,11], [12,5,6,8,11,12], [5,6,9,12,5], [6,9,12,5,6]
test7	("ou")	,,	[1,4,5,6,8,11,12,5,6,9,12,5,13]	1,4,5,6,8,11,12], [6,8,11,12,5,6], [5,6,8,11,12,5], [8,11,12,5,6,9], [5,6,9,12,5], [6,9,12,5,13], [12,5,6,9,12]
test8	("d")	"d"	1,4,5,6,7,12,5,6,8,10,11,12,5,13]	[6,8,10,11,12,5,13], [7,12,5,6,8,10,11], [5,6,8,10,11,12,5], [12,5,6,8,10,11,12], [1,4,5,6,7,12], [6,7,12,5,6], [5,6,7,12,5]
test9	("obra")	"o"	[1,4,5,6,8,10,11,12,5,6,7,12,5,13]	[1,4,5,6,8,10,11,12], [6,8,10,11,12,5,6], [5,6,8,10,11,12,5], [8,10,11,12,5,6,7], [6,7,12,5,13], [5,6,7,12,5], [12,5,6,7,12]
test10	("paralelo")	"p"	[1,4,5,6,9,12,5,6,8,10,11,12,5,13]	[6,8,10,11,12,5,13], [5,6,8,10,11,12,5], [9,12,5,6,8,10,11], [12,5,6,8,10,11,12], [1,4,5,6,9,12], [5,6,9,12,5], [6,9,12,5,6]
test11	("ova")	"o"	[1,4,5,6,8,10,11,12,5,6,9,12,5,13]	[1,4,5,6,8,10,11,12], [6,8,10,11,12,5,6], [5,6,8,10,11,12,5], [8,10,11,12,5,6,9], [5,6,9,12,5], [6,9,12,5,13], [12,5,6,9,12]
test12	("fato")	()	[1,4,5,6,7,12,5,6,9,12,5,6,7,12,5,13]	[1,4,5,6,7,12], [6,7,12,5,6], [6,7,12,5,13], [5,6,9,12,5], [6,9,12,5,6], [5,6,7,12,5], [9,12,5,6,7], [12,5,6,7,12], [12,5,6,9,12], [7,12,5,6,9]
test13	("ola")	"ola"	[1,4,5,6,8,10,11,12,5,6,8,10,11,12,5,6,8,10,11,12,5,13]	[1,4,5,6,8,10,11,12], [6,8,10,11,12,5,6], [6,8,10,11,12,5,13], [5,6,8,10,11,12,5], [11,12,5,6,8,10,11], [12,5,6,8,10,11,12], [10,11,12,5,6,8,10], [8,10,11,12,5,6,8]
test14	("daa")	"da"	[1,4,5,6,7,12,5,6,8,11,12,5,6,8,11,12,5,6,7,12,5,6,7,12,5,13]	[6,8,11,12,5,6], [5,6,8,11,12,5], [7,12,5,6,8,11], [1,4,5,6,7,12], [11,12,5,6,8,11], [12,5,6,8,11,12], [8,11,12,5,6,8], [8,11,12,5,6,7], [6,7,12,5,6], [6,7,12,5,13], [5,6,7,12,5], [7,12,5,6,7], [12,5,6,7,12]

(*) Para este teste a estrutura não tem nenhum nó.

De notar que para determinados testes é necessário adicionar novos dados. Esses casos estão descritos no código nos testes: 9, 10, 11 e 13.

2.3 All-Uses Coverage

O All-Uses coverage cobre todos os caminhos que vão desde cada def até cada uso dessa def.

2.3.1 longestPrefixOf()

Considerando todos os usos, identificámos testes que cobrissem cada um deles. Para a sua realização foram criadas duas árvores, ambas inicializadas vazias. Os inputs realizados na árvore em cada teste estão descriminados na coluna "tree insertions", sendo que as que dizem respeito à segunda árvore têm essa indicação, enquanto que as outras têm só o input. Os testes efectuados foram os seguintes:

Test	Test Case Values	Expected Value	Test Paths	Tree insertions
test1	(null)	IAE	[1,2]	
test2	(”)	null	[1,3]	
test3	("ola")	”	[1,4,5,13]	
test4	("m")	”	[1,4,5,6,7,12,5,13]	put("moda"), put("ola")
test5	("o")	”	[1,4,5,6,8,11,12,5,13]	
test6	("v")	”	[1,4,5,6,9,12,5,13]	
test7	("m")	”	[1,4,5,6,7,12,5,6,8,11,12,5,13]	put("j")
test8	("n")	”	[1,4,5,6,7,12,5,6,9,12,5,13]	
test9	("ol")	"ol"	[1,4,5,6,8,11,12,5,6,8,11,12,5,13]	
test10	("ox")	”	[1,4,5,6,8,11,12,5,6,9,12,5,13]	
test11	("v")	”	[1,4,5,6,9,12,5,6,8,11,12,5,13]	put("v")
test12	("x")	”	[1,4,5,6,9,12,5,6,9,12,5,13]	
test13	("o")	"o"	[1,4,5,6,8,10,11,12,5,13]	put("o", 1)
test14	("j")	"j"	[1,4,5,6,7,12,5,6,7,12,5,13]	put("j")
test15	("m")	"m"	[1,4,5,6,7,12,5,6,8,10,11,12,5,13]	2ª arvore: put("m", 2)
test16	("oi")	”	[1,4,5,6,8,11,12,5,6,7,12,5,13]	
test17	("o")	”	[1,4,5,6,8,11,12,5,13]	
test18	("ov")	"ov"	[1,4,5,6,8,11,12,5,6,8,10,11,12,5,13]	2ª arvore: put("v", 2)
test19	("p")	”	[1,4,5,6,9,12,5,6,7,12,5,13]	
test20	("v")	”	[1,4,5,6,9,12,5,6,8,10,11,12,5,13]	2ª arvore

2.4 Logic-based test coverage

Os testes de cobertura logic based cobrem as expressões lógicas. O critério usado foi o Correlated Active Clause Coverage (CACC), já que cobre não só os predicados, mas também as clauses, e sendo que um dos predicados é composto por duas clauses, considerámos que seria pertinente.

2.4.1 longestPrefixOf()

Na função considerada identificámos seis predicados diferentes e sete claúsulas, sendo que apenas um dos predicados apresenta duas cláusulas. Os testes efetuados foram os seguintes:

#	Y	Expected	Predicate Values
test1	(null)	IAE	P1
test2	(”)	null	¬p1 & p2
test3	("ola")	”	¬p1 & ¬p2 & ¬p3
test4	("a")	”	¬p1 & ¬p2 & p3 & p4
test5	("r")	”	¬p1 & ¬p2 & p3 & ¬p4, p5
test6	("om")	"om"	p1, ¬p2, p3, ¬p4, ¬p5, p6

2.5 Input State Partitioning

Nesta secção aplicamos o critério de Input State Partitioning usando o Base Choice Coverage. Primeiramente, foram definidos os blocks para cada characteristic. Foram definidos da seguinte forma:

Characteristic 1: Trie already includes the new key
Blocks: T,F
Characteristic 2: Trie already includes some new key prefix
Blocks: T,F
Characteristic 3: Trie is empty
Blocks: T,F

A base choice escolhida foi (F,T,F), em que a primeira posição refere à characteristic 1 e assim sucessivamente. Foi escolhida tal base choice pois é o caso mais interessante no ponto de vista de teste.

Partindo da base choice escolhida, foram feitos os seguintes testes:

test	Characteristic 1	Characteristic 2	Characteristic 3
test1	F	T	F
test2	T	T	F
test3	F	F	F

A aplicação deste critério levou a mais resultados mas esses são inconsistentes. Por exemplo (F,T,T). Neste caso, a estrutura está vazia e tem prefixo da nova key ao mesmo tempo. Assim podemos confirmar a inconsistência desta instância.

3 PIT

Usando o plugin Pitclipse foram feitas as mutações no código geradas a fim de proceder à Mutation coverage. Os resultados foram os seguintes:

	Killed	Survived	No Coverage
Instruction Coverage	113	6	11
Edge Coverage	36	14	80
Prime Path Coverage	39	13	78
All-Uses Coverage	39	14	77
Logic-based Coverage	34	12	84
Base Choice Coverage	35	5	80

Analisando estes resultados, constatámos que o Instruction Coverage se destaca, pois este critério foi usado para testar toda a classe TST<T> enquanto que, os restantes critérios foram aplicados a uma só função da mesma classe (longestPrefixOf() ou put()). O que verificámos foi que muitos dos mutantes que sobreviveram são comuns entre critérios e dizem respeito principalmente aos métodos get() e contains(), assim sendo, como não estão enquadrados são métodos "acessórios"em relação ao método que foi principalmente alvo neste projecto, não constituíram uma grande preocupação. Estes mutantes sobreviventes terão provavelmente ocorrido devido ao uso de funções dentro dos testes, e não propriamente como consequência dos métodos alvo, longestPrefixOf() e put().
Foram feitos novos testes com o objectivo de melhorar o critério Mutation Coverage.
A nível global, no contexto de Mutation Coverage, foi executado o PIT tendo como alvo a bateria de testes completa. Os resultados foram os seguintes:

- Killed - 116
- Survived - 3
- No Coverage - 11

De notar que os mutantes que sobreviveram dizem respeito a mutações que ocorreram em métodos privados da classe a ser testada.

4 Conclusão

Este projecto permitiu-nos a aplicação de conhecimentos adquiridos na disciplina de Verificação e Validação de Software, explorámos os conceitos de testing, utilizando vários critérios de testes. Foi-nos possível concluir que, perante os testes que criámos, não existiram falhas. Assim, acreditamos que conseguimos cumprir os objectivos que nos tinham proposto inicialmente. Através do uso da ferramenta PIT que usámos para validar os nossos testes, pudemos observar que a maioria das mutações não sobreviveram. Assim, podemos concluir que os testes foram consideravelmente bem sucedidos, apenas sobreviveram 3 mutantes.