Hokify

Relatório Final

Métodos Formais de Engenharia de Software

Mestrado Integrado em Engenharia Informática e Computação

Fabiola Figueira da Silva - 201502850
Pedro Filipe Agrela Faria - 201406992

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

3 de Janeiro de 2018

Conteúdo

1- Descrição do projeto

1.1. Descrição informal do sistema

Neste projeto pretende-se modelar um sistema que possibilite a criação de CV's e trabalhos, com o objetivo de facilitar a pesquisa de ofertas de emprego para utilizadores cujos atributos satisfaçam os requisitos do trabalho, sendo estes atributos a localidade, escolaridade, interesses ou skills.

No sistema Hokify é possível:

- Listar, adicionar e remover Utilizadores
- Listar, adicionar e remover Trabalhos
- Adicionar ou remover Skills e Interesses dos Utilizadores e dos Trabalhos
- Realizar pesquisas de trabalhos pelo nome, país, localidade, escolaridade, interesses e ou skills
- Realizar pesquisas de utilizadores pelo nome, escolaridade, interesses e ou skills
- Pesquisar possíveis trabalhos para determinado utilizador
- Pesquisar possíveis utilizadores para determinado trabalho

1.2. Lista de requisitos

Requisitos	Prioridade	Descrição
R01	Obrigatória	Adicionar utilizadores
R02	Obrigatória	Listar utilizadores
R03	Obrigatória	Editar utilizadores
R04	Obrigatória	Remover utilizadores
R05	Obrigatória	Obter informações do utilizador
R06	Obrigatória	Adicionar interesses ao utilizador

R07	Obrigatória	Listar interesses do utilizador
R08	Obrigatória	Editar interesses do utilizador
R09	Obrigatória	Remover interesses do utilizador
R10	Obrigatória	Adicionar skills ao utilizador
R11	Obrigatória	Listar skills do utilizador
R12	Obrigatória	Editar skills do utilizador
R13	Obrigatória	Remover skills do utilizador
R14	Obrigatória	Pesquisar utilizadores pela escolaridade mínima
R15	Obrigatória	Pesquisar utilizadores pelos interesses
R16	Obrigatória	Pesquisar utilizadores pelas skills
R17	Obrigatória	Pesquisar utilizadores que podem candidatar-se para determinado trabalho com base na escolaridade, skills e interesses
R18	Obrigatória	Adicionar trabalho
R19	Obrigatória	Listar trabalhos
R20	Obrigatória	Editar trabalho
R21	Obrigatória	Remover trabalho
R22	Obrigatória	Obter informações do trabalho
R23	Obrigatória	Adicionar interesses necessários para obter o trabalho
R24	Obrigatória	Listar interesses necessários para obter o trabalho
R25	Obrigatória	Editar interesses necessários para obter o trabalho
R26	Obrigatória	Remover interesses necessários para obter o trabalho
R27	Obrigatória	Adicionar skills necessárias para obter o trabalho
R28	Obrigatória	Listar skills necessárias para obter o trabalho
R29	Obrigatória	Editar skills necessárias para obter o trabalho
R30	Obrigatória	Remover skills necessárias para obter o trabalho

R31	Obrigatória	Pesquisar trabalhos pela escolaridade mínima
R32	Obrigatória	Pesquisar trabalhos pelos interesses necessárias
R33	Obrigatória	Pesquisar trabalhos pelas skills necessárias
R34	Obrigatória	Pesquisar trabalhos que um determinado utilizador pode candidatar-se com base na escolaridade, skills e interesses

Tabela 1 - Lista de requisitos

2- UML

2.1. Use Case Model

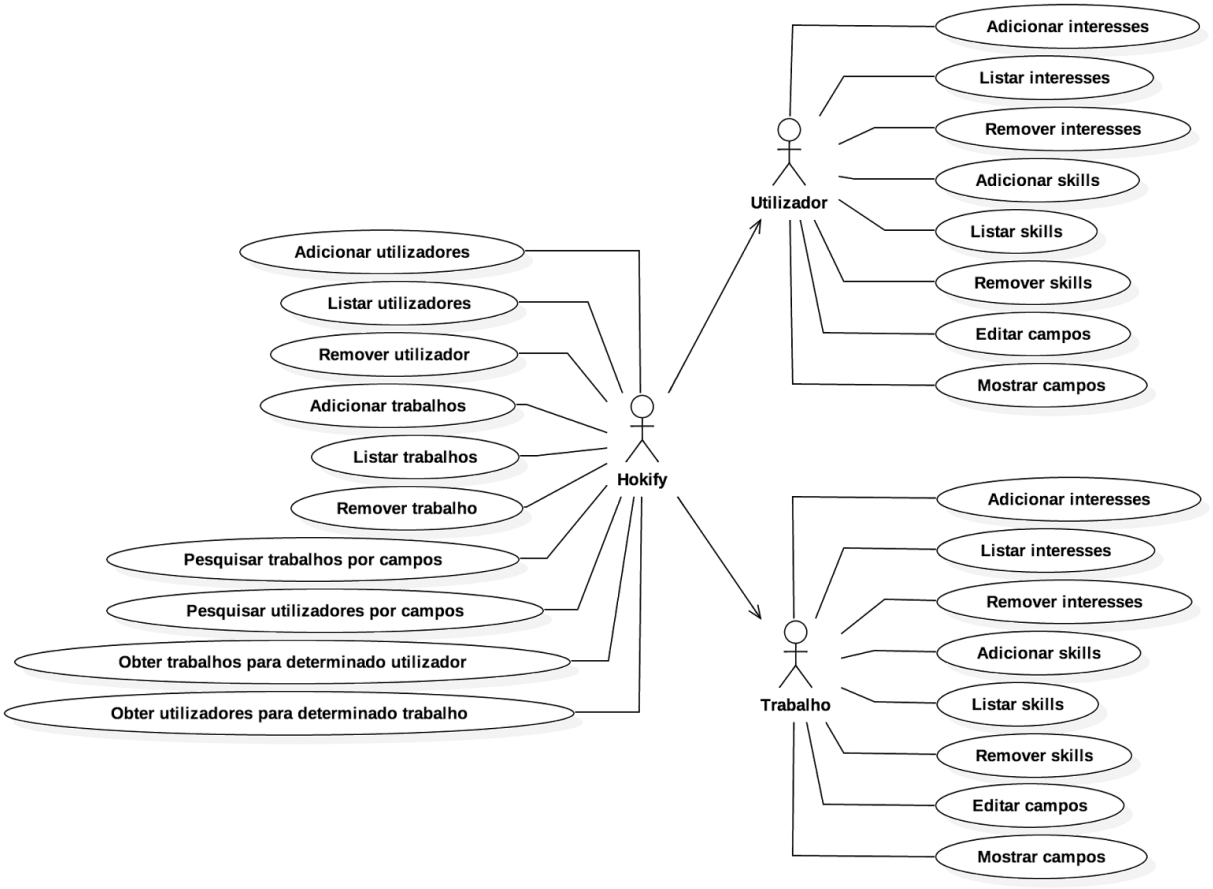

Ilustração 1 - Use Case Model

2.2. Class Model

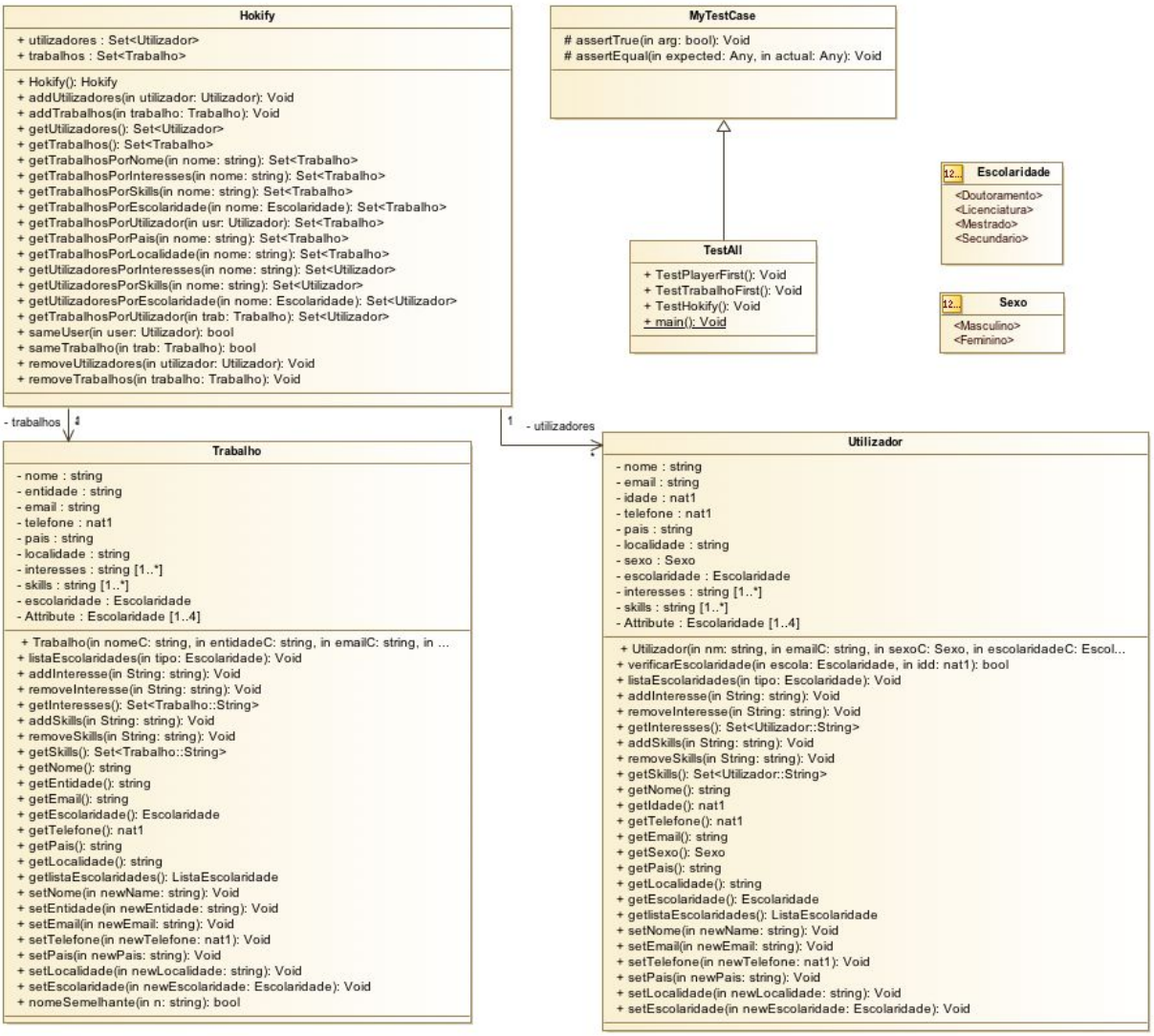

Ilustração 2 - Class Model

Classes	Descrição
Utilizador	Classe utilizador, que contém campos que o define, como o nome, idade, escolaridade, áreas de interesse e skills.
Trabalho	Classe trabalho, que contém campos que o define, como a entidade, a escolaridade mínima, áreas de interesse e skills.
Hokify	Classe principal, onde irá conter os utilizadores e trabalhos a fim de fazer a gestão dos mesmos.

myTestCase	Superclasse para testar as classes; define assertEquals e assertTrue.
TestCall	Define os cenários de teste/uso e testa casos para o Hokify.

Tabela 2 - Descrição das classes

3. Modelo Formal VDM++

3.1. Classe Hokify

3.1.1. Descrição mais detalhada dos métodos

Nome	Adicionar utilizador
Descrição	Adicionar um novo utilizador a lista de utilizadores armazenados no sistema Hokify
Pré-condições	O utilizador não pode ser duplicado (os campos e-mail e telemóvel devem ser únicos)
Pós-condições	O utilizador deve constar na lista de utilizadores
Passos	1. Adicionar um utilizador na classe Hokify criada
Exceções	O utilizador já pertencer à lista de utilizadores adicionados.

Tabela 3 - Descrição do método addUtilizadores

Nome	Listar utilizador
Descrição	Obter utilizadores registados na lista de utilizadores
Pré-condições	Nenhuma pré-condição
Pós-condições	Nenhuma pós-condição
Passos	Sem passos
Exceções	Sem exceções

Tabela 4 - Descrição do método getUtilizadores

Nome	Remover utilizador
Descrição	Remover determinado utilizador da lista de utilizadores

Pré-condições	O utilizador deve constar na lista de utilizadores
Pós-condições	O utilizador não pode existir na lista de utilizadores
Passos	1. Adicionar um utilizador na classe Hokify criada 2. Inserir o utilizador a remover
Exceções	O utilizador não pertencer à lista de utilizadores adicionados.

Tabela 5 - Descrição do método removeUtilizadores

Nome	Adicionar trabalho
Descrição	Adicionar um novo trabalho a lista de trabalhos armazenados no sistema Hokify
Pré-condições	O trabalho não pode ser duplicado (os campos e-mails, nome e entidade devem ser únicos)
Pós-condições	O trabalho deve constar na lista de trabalhos
Passos	1. Adicionar um trabalho na classe Hokify criada
Exceções	O trabalho já pertencer à lista de trabalhos adicionados.

Tabela 6 - Descrição do método addTrabalhos

Nome	Listar trabalho
Descrição	Obter trabalhos registados na lista de trabalhos
Pré-condições	Nenhuma pré-condição
Pós-condições	Nenhuma pós-condição
Passos	Sem passos
Exceções	Sem exceções

Tabela 7 - Descrição do método getTrabalhos

Nome	Remover trabalho
Descrição	Remover determinado trabalho da lista de trabalhos
Pré-condições	O trabalho deve constar na lista de trabalhos da aplicação

Pós-condições	O trabalho não pode existir na lista de trabalhos
Passos	1. Adicionar um trabalho na classe Hokify criada 2. Inserir o trabalho a remover
Exceções	O trabalho não pertencer à lista de trabalhos da aplicação

Tabela 8 - Descrição do método removeTrabalhos

Nome	Pesquisar trabalhos por campos
Descrição	Obter uma lista de trabalhos que contenham o valor do campo introduzido, (é possível pesquisar pelos campos: nome, skills, interesses e escolaridade mínima, país e localidade)
Pré-condições	A dimensão da opção escolhida deve ser maior que zero Ou A opção escolhida ter que estar definida
Pós-condições	Nenhuma pós-condição
Passos	1. Adicionar um trabalho na classe Hokify criada 2. Pesquisar pelo trabalho
Exceções	O tamanho da string pesquisa ser inferior a 1 O trabalho não pertencer à lista de trabalhos da aplicação

Tabela 9 - Descrição dos métodos getTrabalhoPorNome, getTrabalhosPorInteresses, getTrabalhosPorSkills, getTrabalhosPorEscolaridade, getTrabalhosPorUtilizador, getTrabalhosPorPais e getTrabalhosPorLocalidade

Nome	Pesquisar utilizadores por campos
Descrição	Obter uma lista de utilizadores que contenham o valor do campo introduzido, (é possível pesquisar pelos campos: interesses, skills, escolaridade)
Pré-condições	A dimensão da opção escolhida deve ser maior que zero Ou A opção escolhida ter que estar definida
Pós-condições	Nenhuma pós-condição
Passos	1. Adicionar um utilizador na classe Hokify criada 2. Pesquisar pelo utilizador
Exceções	O tamanho da string pesquisa ser inferior a 1 O utilizador não pertencer à lista de utilizadores da aplicação

Tabela 10 - Descrição dos métodos getUtilizadoresPorInteresses, getUtilizadoresPorSkills, getUtilizadoresPorEscolaridade, getTrabalhosPorUtilizador

Nome	Obter trabalhos para determinado utilizador
Descrição	Obter lista de trabalhos para o qual determinado utilizador satisfaz as exigências mínimas do trabalho (verifica a escolaridade, skills e interesses)
Pré-condições	O utilizador tem que estar definido
Pós-condições	Nenhuma pós-condição
Passos	1. Adicionar utilizadores na classe Hokify criada 2. Adicionar trabalhos na classe Hokify criada 3. Inserir o utilizador a pesquisar
Exceções	O utilizador não estar definido

Tabela 11 - Descrição do método getTrabalhosPorUtilizador

Nome	Obter utilizadores para determinado trabalho
Descrição	Obter lista de utilizadores que satisfazem as exigências mínimas de determinado trabalho (verifica a escolaridade, skills e interesses)
Pré-condições	O trabalho tem que estar definido
Pós-condições	Nenhuma pós-condição
Passos	1. Adicionar utilizadores na classe Hokify criada 2. Adicionar trabalhos na classe Hokify criada 3. Inserir o trabalho a pesquisar
Exceções	O trabalho não estar definido

Tabela 12 - Descrição do método getTrabalhosPorUtilizador

3.1.2. Código e respetiva cobertura

```
class Hokify
types
-- TODO Define types here
        public String = seq of char;
        public Utilizadores = set of Utilizador;
        public Trabalhos = set of Trabalho;
```

```
        public Escolaridade = <Secundario> | <Licenciatura> | <Mestrado> |
<Doutoramento>;

values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
        private utilizadores: set of Utilizador := {};
        private trabalhos: set of Trabalho := {};

operations
-- TODO Define operations here

        --Construtor
        public Hokify: () ==> Hokify
        Hokify()==(return self)
        post utilizadores = {} and
        trabalhos = {};


        --Adicionar Utilizadores
        public addUtilizadores: Utilizador ==> ()
        addUtilizadores(utilizador) == utilizadores := utilizadores union {utilizador}
        pre sameUser(utilizador)
        post utilizadores = utilizadores~ union {utilizador};

        -- Remover Utilizadores
        public removeUtilizadores: Utilizador ==> ()
        removeUtilizadores(utilizador) == utilizadores := utilizadores \ {utilizador}
        pre not sameUser(utilizador)
        post utilizadores = utilizadores~ \ {utilizador};

        --Adicionar Trabalhos
        public addTrabalhos: Trabalho ==> ()
        addTrabalhos(trabalho) == trabalhos := trabalhos union {trabalho}
        pre sameTrabalho(trabalho)
        post trabalhos = trabalhos~ union {trabalho};

        -- Remover Trabalhos
        public removeTrabalhos: Trabalho ==> ()
        removeTrabalhos(trabalho) == trabalhos := trabalhos \ {trabalho}
        pre not sameTrabalho(trabalho)
        post trabalhos = trabalhos~ \ {trabalho};
```

```
-- Retorna os utilizadores
public pure getUtilizadores : () ==> set of Utilizador
getUtilizadores() ==
(
return utilizadores;
);

-- Retorna os trabalhos
public pure getTrabalhos : () ==> set of Trabalho
getTrabalhos() ==
(
return trabalhos;
);

-- Retorna os trabalhos por nome
public pure getTrabalhosPorNome: seq of char ==> set of Trabalho
getTrabalhosPorNome(nome) == (return {trabalhos | trabalhos in set trabalhos &
trabalhos.nomeSemelhante(nome)})
pre len nome > 0;

--Retorna os trabalhos por interesses
public pure getTrabalhosPorInteresses: seq of char ==> set of Trabalho
getTrabalhosPorInteresses(nome) == (
dcl results: set of Trabalho := {};
for all tr in set trabalhos do
if nome in set tr.getInteresses() then
        results := results union {tr};
return results;
)
pre len nome > 0;

--Retorna os trabalhos por skills
public pure getTrabalhosPorSkills: seq of char ==> set of Trabalho
getTrabalhosPorSkills(nome) == (
dcl results: set of Trabalho := {};
for all tr in set trabalhos do
if nome in set tr.getSkills() then
        results := results union {tr};
return results;
)
pre len nome > 0;
```

```
--Retorna os trabalhos por Escolaridade
public pure getTrabalhosPorEscolaridade: Escolaridade ==> set of Trabalho
getTrabalhosPorEscolaridade(nome) == (
dcl results: set of Trabalho := {};
for all tr in set trabalhos do
if nome in set tr.getlistaEscolaridades() then
        results := results union {tr};
return results;
)
pre nome <> undefined;

--Retorna os trabalhos por Utilizador (Escolaridade,Skills,interesses)
public pure getTrabalhosPorUtilizador: Utilizador ==> set of Trabalho
getTrabalhosPorUtilizador(usr) == (
dcl results_escolaridade: set of Trabalho := {};
dcl results_skills: set of Trabalho := {};
dcl results_interesses: set of Trabalho := {};
dcl trabalhos_temp: set of Trabalho := {};
results_escolaridade := getTrabalhosPorEscolaridade(usr.getEscolaridade());

for all skill in set usr.getSkills() do
trabalhos_temp := getTrabalhosPorSkills(skill);
for all skill_temp in set trabalhos_temp do
        if skill_temp not in set results_skills then
        results_skills := results_skills union {skill_temp};

for all interesse in set usr.getInteresses() do
trabalhos_temp := getTrabalhosPorInteresses(interesse);
for all interesse_temp in set trabalhos_temp do
        if interesse_temp not in set results_interesses then
        results_interesses := results_interesses union {interesse_temp};
return (results_escolaridade inter results_skills inter results_interesses);
)
pre usr <> undefined;

--Retorna os trabalhos por pais
public pure getTrabalhosPorPais: seq of char ==> set of Trabalho
getTrabalhosPorPais(nome) == (
dcl results: set of Trabalho := {};
for all tr in set trabalhos do
if nome = tr.getPais() then
        results := results union {tr};
return results;
```

```
)
pre len nome > 0;

--Retorna os trabalhos por localidade
public pure getTrabalhosPorLocalidade: seq of char ==> set of Trabalho
getTrabalhosPorLocalidade(nome) == (
dcl results: set of Trabalho := {};
for all tr in set trabalhos do
if nome = tr.getLocalidade() then
        results := results union {tr};
return results;
)
pre len nome > 0;

--Retorna os utilizadres por interesses
public pure getUtilizadoresPorInteresses: seq of char ==> set of Utilizador
getUtilizadoresPorInteresses(nome) == (
dcl results: set of Utilizador := {};
for all tr in set utilizadores do
if nome in set tr.getInteresses() then
        results := results union {tr};
return results;
)
pre len nome > 0;

--Retorna os utilizadres por skills
public pure getUtilizadoresPorSkills: seq of char ==> set of Utilizador
getUtilizadoresPorSkills(nome) == (
dcl results: set of Utilizador := {};
for all tr in set utilizadores do
if nome in set tr.getSkills() then
        results := results union {tr};
return results;
)
pre len nome > 0;

--Retorna os utilizadres por Escolaridade
public pure getUtilizadoresPorEscolaridade: Escolaridade ==> set of Utilizador
getUtilizadoresPorEscolaridade(nome) == (
dcl results: set of Utilizador := {};
for all tr in set utilizadores do
if nome in set tr.getlistaEscolaridades() then
        results := results union {tr};
```

```
return results;
)
pre nome <> undefined;

--Retorna os utilizadores por Trabalhos (Escolaridade,Skills,interesses)
public pure getTrabalhosPorUtilizador: Trabalho ==> set of Utilizador
getTrabalhosPorUtilizador(trab) == (
dcl results_escolaridade: set of Utilizador := {};
dcl results_skills: set of Utilizador := {};
dcl results_interesses: set of Utilizador := {};
dcl utilizadores_temp: set of Utilizador := {};
results_escolaridade := getUtilizadoresPorEscolaridade(trab.getEscolaridade());

for all skill in set trab.getSkills() do
utilizadores_temp := getUtilizadoresPorSkills(skill);
for all skill_temp in set utilizadores_temp do
        if skill_temp not in set results_skills then
        results_skills := results_skills union {skill_temp};

for all interesse in set trab.getInteresses() do
utilizadores_temp := getUtilizadoresPorInteresses(interesse);
for all interesse_temp in set utilizadores_temp do
        if interesse_temp not in set results_interesses then
        results_interesses := results_interesses union {interesse_temp};

return (results_escolaridade inter results_skills inter results_interesses);
)
pre trab <> undefined;

-- Verifica se o utilizador existe por email ou telefone
public pure sameUser: Utilizador ==> bool
sameUser(user) ==(
for all u in set utilizadores do
if (u.getEmail() = user.getEmail() or
        u.getTelefone() = user.getTelefone()) then
        return false;
return true;
)
pre user <> undefined;

-- Verifica se o trabalho existe por email ou nome ou entidade
public pure sameTrabalho: Trabalho ==> bool
sameTrabalho(trab) ==(
```

```
        for all u in set trabalhos do
        if (u.getEmail() = trab.getEmail() or
                u.getNome() = trab.getNome() or
                u.getEntidade() = trab.getEntidade()) then
                return false;
        return true;
        )
        pre trab <> undefined;

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Hokify
```

3.2. Classe Utilizador

3.2.1. Descrição mais detalhada dos métodos e construtor

Nome	Utilizador
Descrição	Construtor da classe Utilizador
Pré-condições	Verificar se o utilizador tem idade mínima para a escolaridade introduzida e o utilizador tem que ser maior de idade
Pós-condições	A lista de interesses e a lista de skils têm que estar vazias, e todos os restantes campos serem guardados corretamente
Passos	1. Criar um utilizador em específico
Exceções	A idade ser inferior ao valor da variável *idadeMinima* A idade ser superior à escolaridade inserida

Tabela 13 - Descrição do contrutor da classe Utilizador

Nome	Listar interesses
Descrição	Obter lista de interesses de determinado utilizador
Pré-condições	Nenhuma pré-condição
Pós-condições	Nenhuma pós-condição
Passos	1. Criar um utilizador em específico 2. Adicionar interesses 3. Listar interesses
Exceções	Sem exceções

Tabela 14 - Descrição do método getInteresses

Nome	Adicionar interesses
Descrição	Permite adicionar interesses ao utilizador
Pré-condições	O interesse não pode ser duplicado
Pós-condições	O interesse adicionado tem que pertencer a lista dos interesses do utilizador

Passos	1. Criar um utilizador em específico 2. Adicionar interesses
Exceções	O interesse já pertencer a lista de interesses do utilizador

Tabela 15 - Descrição do método addInteresses

Nome	Remover interesses
Descrição	Remover interesse da lista de interesses
Pré-condições	O interesse tem que existir na lista de interesses do utilizador
Pós-condições	O interesse não deve existir na lista de interesses do utilizador
Passos	1. Criar um utilizador em específico 2. Adicionar interesses 3. Remover um interesse
Exceções	O interesse não pertencer a lista de interesses do utilizador

Tabela 16 - Descrição do método removeInteresses

Nome	Listar skills
Descrição	Obter lista de skills de determinado utilizador
Pré-condições	Nenhuma pré-condição
Pós-condições	Nenhuma pós-condição
Passos	1. Criar um utilizador em específico 2. Adicionar skills 3. Listar skills
Exceções	Sem exceções

Tabela 17 - Descrição do método getSkills

Nome	Adicionar skills
Descrição	Adicionar uma nova skills a lista de skills
Pré-condições	A skill não pode ser duplicada
Pós-condições	A skill adicionada tem que pertencer a lista dos skills do utilizador

Passos	1. Criar um utilizador em específico 2. Adicionar skills
Exceções	O skill já pertencer a lista de skills do utilizador

Tabela 18 - Descrição do método addSkills

Nome	Remover skills
Descrição	Remover uma skills da lista de skills
Pré-condições	A skills tem que existir na lista de skills do utilizador
Pós-condições	A skills não deve existir na lista de skills do utilizador
Passos	1. Criar um utilizador em específico 2. Adicionar skills 3. Remover um skill
Exceções	O skill não pertencer a lista de skills do utilizador

Tabela 19 - Descrição do método removeSkills

Nome	Get campo
Descrição	Obter valor de determinado campo pertencente ao utilizador (Nome, Idade, Telefone, E-mail, Sexo, País, Localidade, Escolaridade ou Data de nascimento)
Pré-condições	Nenhuma pré-condição
Pós-condições	Nenhuma pós-condição
Passos	1. Criar um utilizador em específico 2. Fazer a chamada ao método pretendido
Exceções	Sem exceções

Tabela 20 - Descrição dos métodos getNome, getIdade, getTelefone, getEmail, getSexo, getPais, getLocalidade, getEscolaridade, getListaEscolaridade e getDataNascimento

Nome	Editar campos
Descrição	Editar cada campo do utilizador individualmente pertencente ao utilizador (Nome, telefone, email, pais, localidade ou escolaridade)
Pré-condições	O novo valor do campo não pode ser indefinido

Pós-condições	O respetivo campo deve ser igual ao valor recebido
Passos	1. Criar um utilizador em específico 2. Fazer a chamada ao método pretendido com os novos campos inseridos
Exceções	O novo valor ser indefinido

Tabela 21 - Descrição dos métodos setNome, setTelefone, setEmail, setPais, setLocalidade e, setEscolaridade

3.2.2. Código e respetiva cobertura

```
class Utilizador
types
-- TODO Define types here
        public String = seq of char;
        public Sexo = <Masculino> | <Feminino>;
        public Escolaridade = <Secundario> | <Licenciatura> | <Mestrado> |
<Doutoramento>;
        public ListaEscolaridade = set of Escolaridade;
        public Interesses = set of String;
        public Skills = set of String;
        public Date :: year : nat month: nat1 day : nat
        inv mk_Date(y,m,d) == m <= 12 and d <= DaysOfMonth(m,y);

values
-- TODO Define values here
        public idadeMinima = 18;

instance variables
-- TODO Define instance variables here
        private nome: seq of char;
        private email: seq of char;
        private sexo: Sexo;
        private escolaridade: Escolaridade;
        private idade: nat1;
        private telefone: nat1;
        private pais: seq of char;
        private localidade: seq of char;
        private interesses: set of String := {};
        private skills: set of String := {};
        private listaEscolaridade: ListaEscolaridade := {};
        private dataNascimento : Date;
        inv idade >= idadeMinima;
```

```
operations
-- TODO Define operations here

        --Construtor
        public Utilizador: seq of char * seq of char * Sexo * Escolaridade * seq of char *
seq of char * nat1 * nat1 * nat * nat1 * nat1 ==> Utilizador

Utilizador(nm,emailC,sexoC,escolaridadeC,paisC,localidadeC,idadeC,telefoneC,year,month,
day) == (
        nome := nm;
        email := emailC;
        sexo := sexoC;
        escolaridade := escolaridadeC;
        listaEscolaridades(escolaridade);
        idade := idadeC;
        telefone := telefoneC;
        pais := paisC;
        localidade := localidadeC;
        dataNascimento := mk_Date(year, month, day);
        return self
        )
        pre verificarEscolaridade(escolaridadeC, idadeC) and idadeC >= idadeMinima
        post interesses = {} and
        skills = {} and
        nome = nm and
        email = emailC and
        sexo = sexoC and
        idade = idadeC and
        telefone = telefoneC and
        pais = paisC and
        localidade = localidadeC;

        -- Verifica se o utilizador tem idade para a escolaridade
        public pure verificarEscolaridade: Escolaridade * nat1 ==> bool
        verificarEscolaridade(escola, idd)==(
        if (escola = <Secundario> and idd >= 17) then(
        return true;
        )elseif(escola = <Licenciatura> and idd > 20) then(
        return true;
        )elseif(escola = <Mestrado> and idd > 22) then(
        return true;
        )elseif(escola = <Doutoramento> and idd > 24) then(
```

```
return true;
)else
return false;
)pre escolaridade <> undefined;

-- Adiciona uma lista com as escolaridades no qual possui
public listaEscolaridades: Escolaridade ==> ()
listaEscolaridades(tipo)==(
if tipo = <Doutoramento> then (
listaEscolaridade := listaEscolaridade union {<Secundario>};
listaEscolaridade := listaEscolaridade union {<Licenciatura>};
listaEscolaridade := listaEscolaridade union {<Mestrado>};
listaEscolaridade := listaEscolaridade union {<Doutoramento>};
)elseif tipo = <Mestrado> then(
listaEscolaridade := listaEscolaridade union {<Secundario>};
listaEscolaridade := listaEscolaridade union {<Licenciatura>};
listaEscolaridade := listaEscolaridade union {<Mestrado>};
)elseif tipo = <Licenciatura> then(
listaEscolaridade := listaEscolaridade union {<Secundario>};
listaEscolaridade := listaEscolaridade union {<Licenciatura>};
)else(
listaEscolaridade := listaEscolaridade union {<Secundario>};
);
)pre tipo <> undefined;

-- Adicionar interesses
public addInteresse: seq of char ==> ()
addInteresse(String) == interesses := interesses union {String}
pre String not in set interesses
post interesses = interesses~ union {String};

-- Remover interesses
public removeInteresse: seq of char ==> ()
removeInteresse(String) == interesses := interesses \ {String}
pre String in set interesses
post interesses = interesses~ \ {String};

-- Retorna os interesses
public pure getInteresses : () ==> set of String
getInteresses() ==
(
return interesses;
);
```

```
-- Adicionar skills
public addSkills: seq of char ==> ()
addSkills(String) == skills := skills union {String}
pre String not in set skills
post skills = skills~ union {String};

-- Remover skills
public removeSkills: seq of char ==> ()
removeSkills(String) == skills := skills \ {String}
pre String in set skills
post skills = skills~ \ {String};

-- Retorna as skills
public pure getSkills : () ==> set of String
getSkills() ==
(
return skills;
);

-- Retorna o nome
public pure getNome : () ==> seq of char
getNome() ==
(
return nome;
);

-- Retorna a idade
public pure getIdade : () ==> nat1
getIdade() ==
(
return idade;
);

-- Retorna o telefone
public pure getTelefone : () ==> nat1
getTelefone() ==
(
return telefone;
);

-- Retorna o email
public pure getEmail : () ==> seq of char
```

```
getEmail() ==
(
return email;
);

-- Retorna o sexo
public pure getSexo : () ==> Sexo
getSexo() ==
(
return sexo;
);

-- Retorna o pais
public pure getPais : () ==> seq of char
getPais() ==
(
return pais;
);

-- Retorna a localidade
public pure getLocalidade : () ==> seq of char
getLocalidade() ==
(
return localidade;
);

-- Retorna a escolaridade
public pure getEscolaridade : () ==> Escolaridade
getEscolaridade() ==
(
return escolaridade;
);

-- Retorna o listaEscolaridades
public pure getlistaEscolaridades : () ==> ListaEscolaridade
getlistaEscolaridades() ==
(
return listaEscolaridade;
);

-- Retorna a data de nascimento
public pure getDataNasccimento : () ==> Date
getDataNasccimento() ==
```

```
(
return dataNascimento;
);

-- Editar Nome
public setNome: seq of char ==> ()
setNome(newName) == nome := newName
pre newName <> undefined
post nome = newName;

-- Editar Email
public setEmail: seq of char ==> ()
setEmail(newEmail) == email := newEmail
pre newEmail <> undefined
post email = newEmail;

-- Editar Telefone
public setTelefone: nat1 ==> ()
setTelefone(newTelefone) == telefone := newTelefone
pre newTelefone <> undefined
post telefone = newTelefone;

-- Editar Pais
public setPais: seq of char ==> ()
setPais(newPais) == pais := newPais
pre newPais <> undefined
post pais = newPais;

-- Editar Localidade
public setLocalidade: seq of char ==> ()
setLocalidade(newLocalidade) == localidade := newLocalidade
pre newLocalidade <> undefined
post localidade = newLocalidade;

-- Editar Escolaridade
public setEscolaridade: Escolaridade ==> ()
setEscolaridade(newEscolaridade) == escolaridade := newEscolaridade
pre newEscolaridade <> undefined
post escolaridade = newEscolaridade;
```

functions

-- TODO Define functiones here

```
-- Retorna o número de dias do mês num dado ano
        public static DaysOfMonth(month,year : nat1) r : nat1 == (
                if month = 1 or month = 3 or month = 5 or month = 7 or month = 8 or month =
10 or month = 12 then
                31
                else if month = 2 and ((year mod 4 = 0 and year mod 100 <> 0) or year mod
400 = 0) then
                29
                else if month = 2 then
                28
                else
                30
        )

traces
-- TODO Define Combinatorial Test Traces here

end Utilizador
```

3.3. Classe Trabalho

3.3.1. Descrição mais detalhada dos métodos e construtor

Nome	Trabalho
Descrição	Construtor da classe Trabalho
Pré-condições	Nenhuma pré-condição
Pós-condições	A lista de interesses e a lista de skils têm que estar vazias e todos os restantes campos serem guardados corretamente
Passos	1. Criar um trabalho em específico
Exceções	Sem exceções

Tabela 22 - Descrição do contrutor da classe Trabalho

Nome	Listar interesses
Descrição	Obter lista de interesses para determinado trabalho
Pré-condições	Nenhuma pré-condição
Pós-condições	Nenhuma pós-condição
Passos	1. Criar um trabalho em específico 2. Adicionar skills 3. Listar skills
Exceções	Sem exceções

Tabela 23 - Descrição do método getInteresses

Nome	Adicionar interesses
Descrição	Permite adicionar interesses ao trabalho
Pré-condições	O interesse não pode ser duplicado
Pós-condições	O interesse adicionado tem que pertencer a lista dos interesses do trabalho
Passos	1. Criar um trabalho em específico 2. Adicionar interesses
Exceções	O interesse já pertencer a lista de interesses do trabalho

Tabela 24 - Descrição do método addInteresse

Nome	Remover interesses
Descrição	Remover interesse da lista de interesses
Pré-condições	O interesse tem que existir na lista de interesses do trabalho
Pós-condições	O interesse não deve existir na lista de interesses do trabalho
Passos	1. Criar um trabalho em específico 2. Adicionar interesses 3. Remover um interesse
Exceções	O interesse não pertencer a lista de interesses do trabalho

Tabela 25 - Descrição do método removeInteresse

Nome	Listar skills
Descrição	Obter lista de skills de determinado trabalho
Pré-condições	Nenhuma pré-condição
Pós-condições	Nenhuma pós-condição
Passos	1. Criar um trabalho em específico 2. Adicionar skills 3. Listar skills
Exceções	Sem exceções

Tabela 26 - Descrição do método getSkills

Nome	Adicionar skills
Descrição	Adicionar uma nova skills a lista de skills
Pré-condições	A skill não pode ser duplicada
Pós-condições	A skill adicionada tem que pertencer a lista dos skills do trabalho
Passos	1. Criar um trabalho em específico 2. Adicionar skills
Exceções	O skill já pertencer a lista de skills do trabalho

Tabela 27 - Descrição do método addSkills

Nome	Remover skills
Descrição	Remover uma skills da lista de skills
Pré-condições	A skills tem que existir na lista de skills do trabalho
Pós-condições	A skills não deve existir na lista de skills do trabalho
Passos	1. Criar um trabalho em específico 2. Adicionar skills 3. Remover um skill
Exceções	O skill não pertencer a lista de skills do trabalho

Tabela 28 - Descrição do método removeSkills

Nome	Get campo
Descrição	Obter valor de determinado campo pertencente ao trabalho (Nome, Entidade, Telefone, E-mail, Escolaridade, País eou Localidade)
Pré-condições	Nenhuma pré-condição
Pós-condições	Nenhuma pós-condição
Passos	1. Criar um trabalho em específico 2. Fazer a chamada ao método pretendido
Exceções	Sem exceções

Tabela 29 - Descrição dos métodos getNome, getEntidade, getTelefone, getEmail, getEscolaridade, getlistaEscolaridades, getPais e getLocalidade

Nome	Editar campos
Descrição	Editar cada campo do trabalho individualmente pertencente ao trabalho (Nome, Entidade, E-mail, Telefone, País, Localidade ou Escolaridade)
Pré-condições	O novo valor do campo não pode ser indefinido
Pós-condições	O respetivo campo deve ser igual ao valor recebido
Passos	1. Criar um utilizador em específico 2. Fazer a chamada ao método pretendido com os novos campos inseridos
Exceções	O novo valor ser indefinido

Tabela 30 - Descrição dos métodos setNome, setEntidade, setEmail, setTelefone, setPais, setLocalidade e setEscolaridade

3.3.2. Código e respetiva cobertura

```
class Trabalho
types
-- TODO Define types here
	public String = seq of char;
	public Escolaridade = <Secundario> | <Licenciatura> | <Mestrado> |
<Doutoramento>;
	public ListaEscolaridade = set of Escolaridade;
	public Interesses = set of String;
	public Skills = set of String;

values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
	private nome: seq of char;
	private entidade: seq of char;
	private email: seq of char;
	private escolaridade: Escolaridade;
	private telefone: nat1;
	private pais: seq of char;
	private localidade: seq of char;
	private interesses: set of String := {};
	private skills: set of String := {};
	private listaEscolaridade: ListaEscolaridade := {};

operations
-- TODO Define operations here

	--Construtor
	public Trabalho: seq of char * seq of char * seq of char * Escolaridade * nat1 *
seq of char * seq of char ==> Trabalho
	Trabalho(nomeC,entidadeC,emailC,escolaridadeC,telefoneC,paisC,localidadeC) == (
	nome := nomeC;
	entidade := entidadeC;
	email := emailC;
	escolaridade := escolaridadeC;
	listaEscolaridades(escolaridade);
	telefone := telefoneC;
```

```
pais := paisC;
localidade := localidadeC;
return self;
)
post interesses = {} and
skills = {} and
nome = nomeC and
entidade = entidadeC and
email = emailC and
escolaridade = escolaridadeC and
telefone = telefoneC and
pais = paisC and
localidade = localidadeC;

-- Adiciona uma lista com as escolaridades no qual possui
public listaEscolaridades: Escolaridade ==> ()
listaEscolaridades(tipo)==(
if tipo = <Secundario> then (
listaEscolaridade := listaEscolaridade union {<Secundario>};
listaEscolaridade := listaEscolaridade union {<Licenciatura>};
listaEscolaridade := listaEscolaridade union {<Mestrado>};
listaEscolaridade := listaEscolaridade union {<Doutoramento>};
)elseif tipo = <Licenciatura> then(
listaEscolaridade := listaEscolaridade union {<Licenciatura>};
listaEscolaridade := listaEscolaridade union {<Mestrado>};
listaEscolaridade := listaEscolaridade union {<Doutoramento>};
)elseif tipo = <Mestrado> then(
listaEscolaridade := listaEscolaridade union {<Mestrado>};
listaEscolaridade := listaEscolaridade union {<Doutoramento>};
)else(
listaEscolaridade := listaEscolaridade union {<Doutoramento>};
););

-- Adicionar interesses
public addInteresse: seq of char ==> ()
addInteresse(String) == interesses := interesses union {String}
pre String not in set interesses
post interesses = interesses~ union {String};

-- Remover interesses
public removeInteresse: seq of char ==> ()
removeInteresse(String) == interesses := interesses \ {String}
pre String in set interesses
```

```
post interesses = interesses~ \ {String};

-- Retorna os interesses
public pure getInteresses : () ==> set of String
getInteresses() ==
(
return interesses;
);

-- Adicionar skills
public addSkills: seq of char ==> ()
addSkills(String) == skills := skills union {String}
pre String not in set skills
post skills = skills~ union {String};

-- Remover skills
public removeSkills: seq of char ==> ()
removeSkills(String) == skills := skills \ {String}
pre String in set skills
post skills = skills~ \ {String};

-- Retorna as skills
public pure getSkills : () ==> set of String
getSkills() ==
(
return skills;
);

-- Retorna o nome
public pure getNome : () ==> seq of char
getNome() ==
(
return nome;
);
-- Retorna o entidade
public pure getEntidade : () ==> seq of char
getEntidade() ==
(
return entidade;
);
-- Retorna o email
public pure getEmail : () ==> seq of char
getEmail() ==
```

```
(
return email;
);
-- Retorna o escolaridade
public pure getEscolaridade : () ==> Escolaridade
getEscolaridade() ==
(
return escolaridade;
);
-- Retorna o telefone
public pure getTelefone : () ==> nat1
getTelefone() ==
(
return telefone;
);
-- Retorna o pais
public pure getPais : () ==> seq of char
getPais() ==
(
return pais;
);
-- Retorna o localidade
public pure getLocalidade : () ==> seq of char
getLocalidade() ==
(
return localidade;
);
-- Retorna o listaEscolaridades
public pure getlistaEscolaridades : () ==> ListaEscolaridade
getlistaEscolaridades() ==
(
return listaEscolaridade;
);

-- Editar Nome
public setNome: seq of char ==> ()
setNome(newName) == nome := newName
pre newName <> undefined
post nome = newName;

-- Editar Entidade
public setEntidade: seq of char ==> ()
setEntidade(newEntidade) == entidade := newEntidade
```

```
pre newEntidade <> undefined
post entidade = newEntidade;

-- Editar Email
public setEmail: seq of char ==> ()
setEmail(newEmail) == email := newEmail
pre newEmail <> undefined
post email = newEmail;

-- Editar Telefone
public setTelefone: nat1 ==> ()
setTelefone(newTelefone) == telefone := newTelefone
pre newTelefone <> undefined
post telefone = newTelefone;

-- Editar Pais
public setPais: seq of char ==> ()
setPais(newPais) == pais := newPais
pre newPais <> undefined
post pais = newPais;

-- Editar Localidade
public setLocalidade: seq of char ==> ()
setLocalidade(newLocalidade) == localidade := newLocalidade
pre newLocalidade <> undefined
post localidade = newLocalidade;

-- Editar Escolaridade
public setEscolaridade: Escolaridade ==> ()
setEscolaridade(newEscolaridade) == escolaridade := newEscolaridade
pre newEscolaridade <> undefined
post escolaridade = newEscolaridade;

public pure nomeSemelhante: seq of char ==> bool
nomeSemelhante(n) == (
dcl nameS: seq of char := nome;
dcl found: bool := false;

while len nameS >= len n and not found do (
found := true;

for index = 1 to len n do
        if found and n(index) <> nameS(index) then (
```

```
            found := false;
            );

        if found then
            return true
        else (
            nameS := tl nameS;
            found := false;
        );
        );

        return false;
        )
        pre len n > 0;

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Trabalho
```

4. Validação do Modelo

Os testes de seguida apresentados foram executados com sucesso na totalidade:

4.1. Classe Hokify

Function or operation	Line	Coverage	Calls
Hokify	20	100.0%	4
addTrabalhos	31	100.0%	3
addUtilizadores	25	100.0%	12
getTrabalhos	44	100.0%	4
getTrabalhosPorEscolaridade	76	100.0%	6
getTrabalhosPorInteresses	56	100.0%	7
getTrabalhosPorLocalidade	119	100.0%	2
getTrabalhosPorNome	51	100.0%	3
getTrabalhosPorPais	109	100.0%	3
getTrabalhosPorSkills	66	100.0%	7
getTrabalhosPorUtilizador	86	100.0%	2
getUtilizadores	37	100.0%	4
getUtilizadoresPorEscolaridade	149	100.0%	6
getUtilizadoresPorInteresses	129	100.0%	7
getUtilizadoresPorSkills	139	100.0%	7
removeInteresse	33	100.0%	1
removeTrabalhos	45	100.0%	1
removeUtilizadores	33	100.0%	1
sameTrabalho	193	100.0%	5
sameUser	183	100.0%	5
Hokify.vdmpp		100.0%	90

Tabela 31 - Cobertura na classe Hokify

4.2. Classe Utilizador

Function or operation	Line	Coverage	Calls
Utilizador	32	100.0%	20
addInteresse	75	100.0%	56
addSkills	94	100.0%	56
getEmail	134	100.0%	44
getEscolaridade	162	100.0%	17
getIdade	120	100.0%	4
getInteresses	87	100.0%	100
getLocalidade	155	100.0%	8
getNome	113	100.0%	8
getPais	148	100.0%	8
getSexo	141	100.0%	4
getSkills	106	100.0%	100
getTelefone	127	100.0%	34
getlistaEscolaridades	169	100.0%	72
listaEscolaridades	55	100.0%	4
removeInteresse	81	100.0%	8
removeSkills	100	100.0%	8
setEmail	182	100.0%	4
setEscolaridade	206	100.0%	5
setLocalidade	200	100.0%	4
setNome	176	100.0%	8
setPais	194	100.0%	4
setTelefone	188	100.0%	4
verificarEscolaridade	56	100.0%	22
Utilizador.vdmpp		100.0%	602

Tabela 32 - Cobertura na classe Utilizador

4.3. Classe Trabalho

Function or operation	Line	Coverage	Calls
Trabalho	29	100.0%	20
addInteresse	70	100.0%	16
addSkills	89	100.0%	16
getEmail	120	100.0%	44
getEntidade	114	100.0%	30
getEscolaridade	126	100.0%	16
getInteresses	82	100.0%	100
getLocalidade	144	100.0%	32
getNome	108	100.0%	34
getPais	138	100.0%	20
getSkills	101	100.0%	100
getTelefone	132	100.0%	8
getlistaEscolaridades	150	100.0%	72
listaEscolaridades	51	100.0%	9
nomeSemelhante	198	100.0%	36
removeInteresse	76	100.0%	4
removeSkills	95	100.0%	4
setEmail	169	100.0%	4
setEntidade	163	100.0%	4
setEscolaridade	193	100.0%	4
setLocalidade	187	100.0%	4
setNome	157	100.0%	4
setPais	181	100.0%	4
setTelefone	175	100.0%	4
Trabalho.vdmpp		100.0%	589

Tabela 33 - Cobertura na classe Trabalho

4.4. Classe myTestCase

Function or operation	Line	Coverage	Calls
assertEqual	20	38.8%	0
assertTrue	12	100.0%	26
MyTestCase.vdmpp		45.0%	26

Tabela 34 - Cobertura na classe MyTestCase

4.4.1. Código e respetiva cobertura

```
class MyTestCase
/*
  Superclass for test classes, simpler but more practical than VDMUnit`TestCase.
  For proper use, you have to do: New -> Add VDM Library -> IO.
  JPF, FEUP, MFES, 2014/15.
*/

operations

        -- Simulates assertion checking by reducing it to pre-condition checking.
        -- If 'arg' does not hold, a pre-condition violation will be signaled.
        protected assertTrue: bool ==> ()
        assertTrue(arg) ==
        return
        pre arg;

        -- Simulates assertion checking by reducing it to post-condition checking.
        -- If values are not equal, prints a message in the console and generates
        -- a post-conditions violation.
        protected assertEqual: ? * ? ==> ()
        assertEqual(expected, actual) ==
        if expected <> actual then (
        IO`print("Actual value (");
        IO`print(actual);
        IO`print(") different from expected (");
        IO`print(expected);
        IO`println(")\n")
```

```
        )
        post expected = actual

end MyTestCase
```

4.5. Classe TestAll

Function or operation	Line	Coverage	Calls
TestHokify	102	100.0%	3
TestPlayerFirst	10	100.0%	4
TestTrabalhoFirst	59	100.0%	4
main	206	100.0%	1
TestAll.vdmpp		100.0%	12

Tabela 35 - Cobertura na classe TestAll

4.5.1. Código e respetiva cobertura

```
class TestAll is subclass of MyTestCase
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
operations
-- TODO Define operations here
public TestPlayer :() ==> ()
TestPlayer() ==
(
        dcl user : Utilizador := new
Utilizador("Pedro","email@email.com",<Masculino>,<Mestrado>,"Portugal","Porto",27,12345
6789,1990,09,08);

        -- Declaracao para testes de um utilizador sem idade minima
```

```
--dcl user2 : Utilizador := new
Utilizador("Pedro2","email2@email.com",<Masculino>,<Secundario>,"Portugal","Porto",17,9
87654321,1990,09,08);

user.addInteresse("Informatica");
user.addInteresse("Cinema");
user.addSkills("Java");
user.addSkills("VDM");

-- Testes simples de verificacao dos dados do utilização - Inserir, editar e eliminar.
assertEqual(user.getNome(),"Pedro");
assertEqual(user.getEmail(),"email@email.com");
assertEqual(user.getSexo(),<Masculino>);
assertEqual(user.getEscolaridade(),<Mestrado>);
assertEqual(user.getPais(),"Portugal");
assertEqual(user.getLocalidade(),"Porto");
assertEqual(user.getIdade(),27);
assertEqual(user.getTelefone(),123456789);
assertEqual(user.getInteresses(),{"Informatica","Cinema"});
assertEqual(user.getSkills(),{"Java","VDM"});
assertEqual(user.getDataNasccimento(),mk_Utilizador`Date(1990, 09, 08));

user.removeInteresse("Cinema");
assertEqual(user.getInteresses(),{"Informatica"});

user.removeSkills("VDM");
assertEqual(user.getSkills(),{"Java"});

user.setNome("Pedro Faria");
user.setEmail("PedroFaria@gmail.com");
user.setPais("Espanha");
user.setLocalidade("Madrid");
user.setEscolaridade(<Secundario>);
user.setTelefone(987654321);
assertEqual(user.getNome(),"Pedro Faria");
assertEqual(user.getEmail(),"PedroFaria@gmail.com");
assertEqual(user.getPais(),"Espanha");
assertEqual(user.getLocalidade(),"Madrid");
assertEqual(user.getEscolaridade(),<Secundario>);
assertEqual(user.getTelefone(),987654321);

--deve falhar pois estamos adicionando um duplicado
--user.addInteresse("Informatica");
```

```
        --user.addSkills("Java");

        --Verifica a escolaridade de um curso
        assertTrue(not (user.verificarEscolaridade(<Secundario>,10)));

        return;
);


public TestTrabalho :() ==> ()
TestTrabalho() ==
(
        dcl trabalho : Trabalho := new Trabalho("Programador de
java","Google","google@google.pt",<Mestrado>,123456789,"Portugal","Porto");
        trabalho.addInteresse("Informatica");
        trabalho.addInteresse("Cinema");
        trabalho.addSkills("Java");
        trabalho.addSkills("VDM");

        -- Testes simples de verificacao dos dados do trabalho - Inserir, editar e eliminar.
        assertEqual(trabalho.getNome(),"Programador de java");
        assertEqual(trabalho.getEntidade(), "Google");
        assertEqual(trabalho.getEmail(),"google@google.pt");
        assertEqual(trabalho.getEscolaridade(),<Mestrado>);
        assertEqual(trabalho.getTelefone(),123456789);
        assertEqual(trabalho.getPais(),"Portugal");
        assertEqual(trabalho.getLocalidade(),"Porto");
        assertEqual(trabalho.getInteresses(),{"Informatica","Cinema"});
        assertEqual(trabalho.getSkills(),{"Java","VDM"});

        trabalho.removeInteresse("Cinema");
        assertEqual(trabalho.getInteresses(),{"Informatica"});

        trabalho.removeSkills("VDM");
        assertEqual(trabalho.getSkills(),{"Java"});

        trabalho.setNome("Programador de C++");
        trabalho.setEntidade("apple");
        trabalho.setEmail("apple@apple.pt");
        trabalho.setPais("Espanha");
        trabalho.setLocalidade("Madrid");
        trabalho.setTelefone(987654321);
        trabalho.setEscolaridade(<Secundario>);
```

```
        assertEqual(trabalho.getNome(),"Programador de C++");
        assertEqual(trabalho.getEmail(),"apple@apple.pt");
        assertEqual(trabalho.getPais(),"Espanha");
        assertEqual(trabalho.getLocalidade(),"Madrid");
        assertEqual(trabalho.getTelefone(),987654321);
        assertEqual(trabalho.getEscolaridade(),<Secundario>);

        return;
);


public TestHokify :() ==> ()
TestHokify() ==
(
        dcl hokify : Hokify := new Hokify();
        dcl utilizador : Utilizador := new
Utilizador("Pedro","email@email.com",<Masculino>,<Licenciatura>,"Portugal","Porto",27,123
456789,1990,10,31);
        dcl utilizador2 : Utilizador := new
Utilizador("Fabiola","gmail@gmail.com",<Feminino>,<Doutoramento>,"Portugal","Lisboa",26
,123123123,1994,02,28);
        dcl utilizador3 : Utilizador := new
Utilizador("Francisca","asd@asd.com",<Feminino>,<Secundario>,"Portugal","Lisboa",26,123
132323,1992,02,29);
        dcl sameuser : Utilizador := new
Utilizador("Pedro","email@email.com",<Masculino>,<Licenciatura>,"Portugal","Porto",27,123
456789,1990,10,31);
        dcl trabalho : Trabalho := new Trabalho("Programador de
java","Google","google@google.pt",<Licenciatura>,123456789,"Portugal","Porto");
        dcl trabalho2 : Trabalho := new Trabalho("Programador de
c++","Apple","apple@apple.pt",<Doutoramento>,4562343434,"Portugal","Lisboa");
        dcl trabalho3 : Trabalho := new Trabalho("Programador de
php","Apple2","apple2@apple.pt",<Secundario>,357864,"Portugal","Funchal");
        dcl sametrabalho : Trabalho := new Trabalho("Programador de
java","Google","google@google.pt",<Licenciatura>,123456789,"Portugal","Porto");

        hokify.addUtilizadores(utilizador);
        hokify.addTrabalhos(trabalho);
        assertEqual(card hokify.getUtilizadores(),1);
        assertEqual(card hokify.getTrabalhos(),1);

        hokify.addUtilizadores(utilizador2);
        hokify.addTrabalhos(trabalho2);
```

```
assertEqual(card hokify.getUtilizadores(),2);
assertEqual(card hokify.getTrabalhos(),2);

hokify.addUtilizadores(utilizador3);
hokify.addTrabalhos(trabalho3);

--Deve falhar pois estamos adicionando um utilizador igual
--hokify.addUtilizadores(sameuser);
--hokify.addTrabalhos(sametrabalho);
assertTrue(not hokify.sameUser(sameuser));
assertTrue(not hokify.sameTrabalho(sametrabalho));

-- Testes para a pesquisa por nome do trabalho em que retorne os trabalhos

assertEqual(hokify.getTrabalhosPorNome("Programador"),{trabalho,trabalho2,trabalho3});
assertEqual(hokify.getTrabalhosPorNome("java"),{trabalho});
assertEqual(hokify.getTrabalhosPorNome("c++"),{trabalho2});

-- Testes para a pesquisa por interesses em que retorne os trabalhos
trabalho.addInteresse("Informatica");
trabalho.addInteresse("Cinema");
assertEqual(hokify.getTrabalhosPorInteresses("Informatica"),{trabalho});
assertEqual(hokify.getTrabalhosPorInteresses("Cinema"),{trabalho});
assertTrue(not (hokify.getTrabalhosPorInteresses("Cinema")={trabalho2}));

-- Testes para a pesquisa por skills em que retorne os trabalhos
trabalho.addSkills("Java");
trabalho.addSkills("VDM");
assertEqual(hokify.getTrabalhosPorSkills("Java"),{trabalho});
assertEqual(hokify.getTrabalhosPorSkills("VDM"),{trabalho});
assertTrue(not (hokify.getTrabalhosPorSkills("Java")={trabalho2}));

-- Testes para a pesquisa por localidade em que retorne os trabalhos
assertEqual(hokify.getTrabalhosPorLocalidade("Porto"),{trabalho});
assertEqual(hokify.getTrabalhosPorLocalidade("Lisboa"),{trabalho2});

-- Testes para a pesquisa por pais em que retorne os trabalhos
assertEqual(hokify.getTrabalhosPorPais("Portugal"),{trabalho,trabalho2,trabalho3});

-- Testes para a escolaridade em que retorne os trabalhos
assertEqual(hokify.getTrabalhosPorEscolaridade(<Secundario>),{trabalho3});

assertEqual(hokify.getTrabalhosPorEscolaridade(<Licenciatura>),{trabalho,trabalho3});
```

```
        assertEqual(hokify.getTrabalhosPorEscolaridade(<Mestrado>),{trabalho,trabalho3});

assertEqual(hokify.getTrabalhosPorEscolaridade(<Doutoramento>),{trabalho,trabalho2,trab
alho3});

        -- Testes para procurar por trabalhos para um utilizador
        utilizador.addInteresse("Informatica");
        utilizador.addInteresse("Cinema");
        utilizador.addSkills("Java");
        utilizador.addSkills("VDM");
        assertEqual(hokify.getTrabalhosPorUtilizador(utilizador),{trabalho});

        utilizador2.addInteresse("Informatica");
        utilizador2.addInteresse("Cinema");
        utilizador2.addSkills("Java");
        utilizador2.addSkills("VDM");
        assertEqual(hokify.getTrabalhosPorUtilizador(utilizador2),{trabalho});

        -- Testes para a pesquisa por interesses em que retorne os utilizadores

assertEqual(hokify.getUtilizadoresPorInteresses("Informatica"),{utilizador,utilizador2});
        utilizador2.removeInteresse("Cinema");
        assertEqual(hokify.getUtilizadoresPorInteresses("Cinema"),{utilizador});
        assertTrue(not (hokify.getUtilizadoresPorInteresses("Cinema")={}));

        -- Testes para a pesquisa por skills em que retorne os utilizadores
        assertEqual(hokify.getUtilizadoresPorSkills("Java"),{utilizador,utilizador2});
        utilizador2.removeSkills("VDM");
        assertEqual(hokify.getUtilizadoresPorSkills("VDM"),{utilizador});
        assertTrue(not (hokify.getUtilizadoresPorSkills("VDM")={}));

        -- Testes para a escolaridade em que retorne os utilizadores

assertEqual(hokify.getUtilizadoresPorEscolaridade(<Secundario>),{utilizador,utilizador2,utili
zador3});

assertEqual(hokify.getUtilizadoresPorEscolaridade(<Licenciatura>),{utilizador,utilizador2});
        assertEqual(hokify.getUtilizadoresPorEscolaridade(<Mestrado>),{utilizador2});
        assertEqual(hokify.getUtilizadoresPorEscolaridade(<Doutoramento>),{utilizador2});

        -- Testes para procurar por utilizadores para um trabalho
        assertEqual(hokify.getTrabalhosPorUtilizador(trabalho),{utilizador});
        utilizador2.addInteresse("Cinema");
```

```
        utilizador2.addSkills("VDM");
        assertEqual(hokify.getTrabalhosPorUtilizador(trabalho),{utilizador,utilizador2});

        assertEqual(card hokify.getUtilizadores(),3);
        assertEqual(card hokify.getTrabalhos(),3);
        hokify.removeUtilizadores(utilizador2);
        hokify.removeTrabalhos(trabalho2);
        assertEqual(card hokify.getUtilizadores(),2);
        assertEqual(card hokify.getTrabalhos(),2);
        return;
);


public static main: () ==> ()
main() ==
(
        IO`print("TestPlayerFirst -> ");
        new TestAll().TestPlayer();
        IO`println("Passed");

        IO`print("TestTrabalhoFirst -> ");
        new TestAll().TestTrabalho();
        IO`println("Passed");

        IO`print("TestHokify -> ");
        new TestAll().TestHokify();
        IO`println("Passed");
);

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end TestAll
```

4.6. Resultados

Como é possível verificar pelas tabelas anteriores, o código em VDM++ obteve uma cobertura de 100%. Na seguinte imagem podemos confirmar que todos os testes unitários passaram com sucesso.

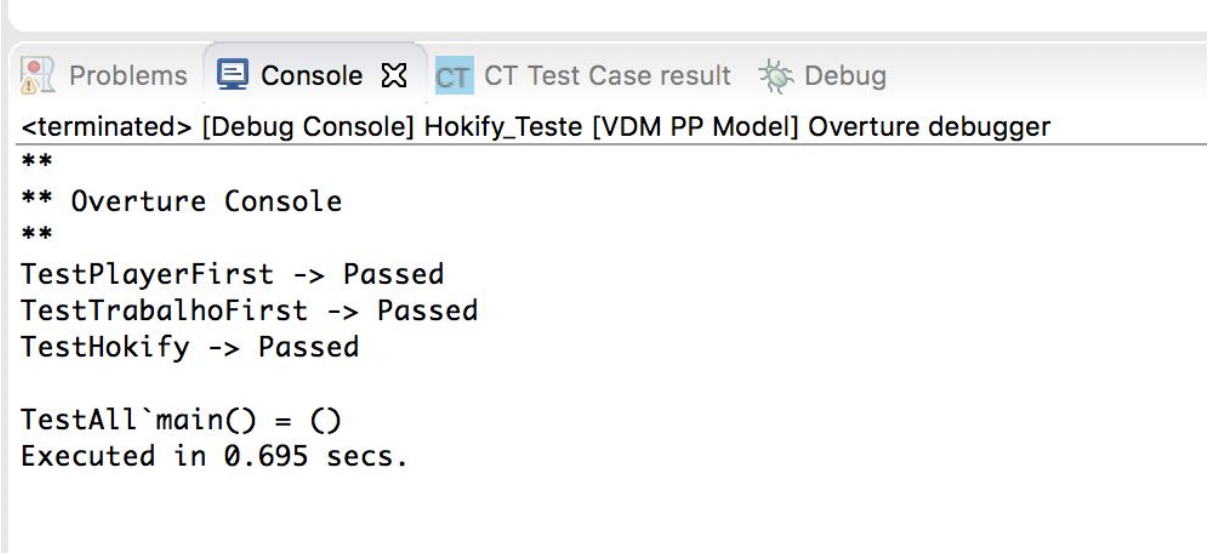

Ilustração 3 - Mensagem de cada teste no Overture

5. Verificação do modelo

5.1. Exemplo de uma verificação de invariante

Um exemplo de invariante é comparação entre a idade do utilizador e da *idadeMinima*, ao criar um novo utilizador com uma idade inferior a *idadeMinima* a consola dá erro na precondição como demonstra a seguinte imagem. Neste caso que estamos analisando, a *idadeMinima* é de 18 anos e foi tentado adicionar um utilizador com uma idade de 17 anos.

Só é possível criar um novo utilizador com idade igual ou superior à *idadeMinima*.

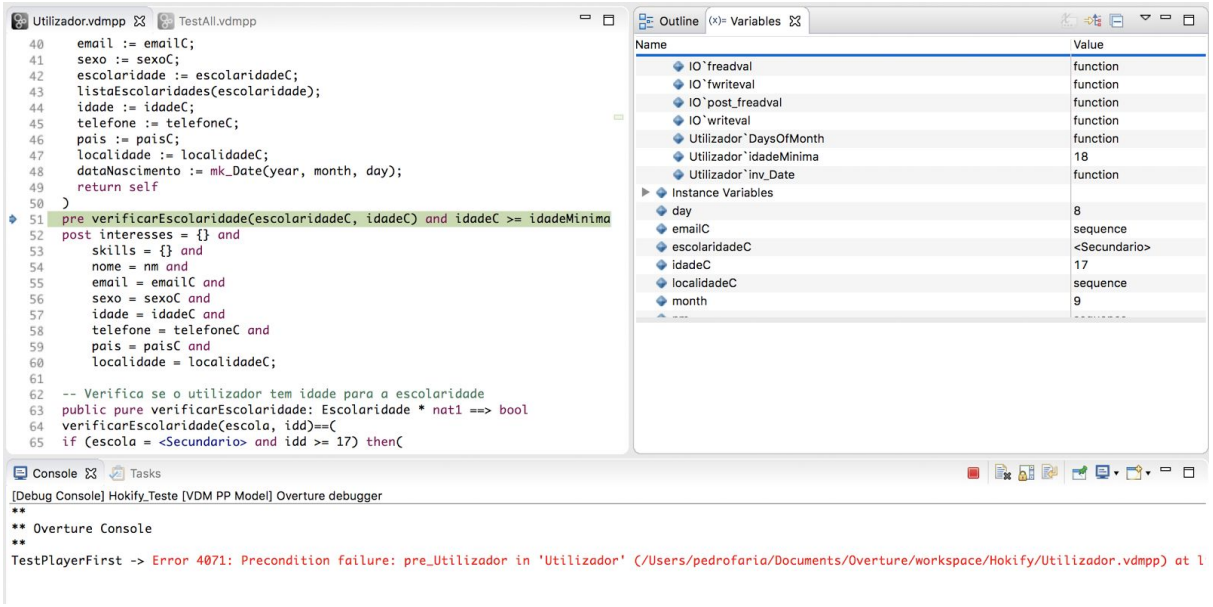

Ilustração 4 - Mensagem da verificação da invariante idade.

6. Geração de código Java

O código foi gerado com sucesso desde o Overture para o código java, sendo apenas necessário mudar o package Hokify.quotes para Hokify_quotes e o nome das chamadas aos métodos desse package.

Ao correr os testes unitários usando o Eclipse com o código já em java, o código foi corrido com sucesso como mostra a seguinte figura:

Ilustração 4 - Mensagem de cada teste executado em java

7- Conclusões

O grupo utilizou o trabalho da melhor maneira para pôr em prática os conceitos lecionados nas aulas. Trabalhamos desde a primeira hora e todos os prazos foram cumpridos.

Em traços muito gerais o projeto desenvolvido por nós simula a aplicação Hokify, onde os utilizadores podem se registar, manipular dados, procurar e adicionar trabalhos para uma comunidade.

Em suma, este projeto elucidou-nos sobre o impacto de uma boa implementação e de como existem ferramentas para verificar a sua correção. Em sistemas críticos esta verificação revela-se essencial.

Concluímos que atingimos assim todos os objetivos que nos tínhamos proposto e que a contribuição dos elementos do grupo foi equitativa.

A criação do projeto e relatório teve um tempo estimado de 35 horas (cerca de 7 horas por dia, durante 5 dias).

8- Referências

[1] http://overturetool.org/download/

[2] https://www.eclipse.org/

[3]https://docs.google.com/document/d/1cf1cn2qbELCMaHZcBut__0dQL9HGnHlk-l7dmM5q5kw/pub

[4] VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014

[5] Apontamentos das aulas teóricas e práticas da unidade curricular Métodos Formais de Engenharia de Software