**Complexity Metrics Miguel Agostinho (60677)**

Uma imagem com mesa

Descrição gerada automaticamente

Uma imagem com mesa

Descrição gerada automaticamente



Obs: Para esta métrica optamos por agrupá-la por packages pois, caso contrário ficaria muito extenso.

**Cyclomatic Complexity** - A Cyclomatic Complexity consiste no número de decisões mais 1, sendo este número de decisões if, if … else, switch , for loop, while loop

**v(G)avg** - A Average Cyclomatic Complexity é a complexidade ciclomática por função de um ficheiro/package. Portanto, esta é a soma das complexidades ciclomáticas de todas as funções a dividir pelo número de funções presentes nesse mesmo ficheiro/package.

**v(G)tot** - A Total Cyclomatic Complexity é a soma das complexidades ciclomáticas de funções presentes num ficheiro/package.

**Pontos problemáticos:**

Nesta métrica podemos observar que na package net.sourceforge.ganttproject.gui.options.model obtemos um valor extremo de Cyclomatic Complexity sendo esse valor 12 para **v(G)avg** ou seja, a média da complexidade de cada função nesta package é 12. Sendo este um valor elevado deveríamos rever o código desta package para tentar reduzir esta mesma complexidade pois, funções com grande cyclomatic complexity são difíceis de entender e também de obter cobertura total do código em testes unitários.

Por outro lado encontramos também outros extremos como por exemplo na package net.sourceforge.ganttproject.gui pois conseguimos observar que o valor de **v(G)tot** é 805 o que significa que a Cyclomatic Complexity tem um valor total de 805 nesta package mas, sendo o valor de **v(G)avg** baixo quando comparado com outras packages.

**Relação com code smells:**

Como explicado acima, o valor desta métrica depende do número de decisões, portanto, sempre que temos um número elevado isto pode significar que podemos encontrar uma grande sequência de if’s o que representará um code smell “Long Method” (Um método que contém muitas linhas de código).

Reviews

—————————————————————————————————————

Francisco Silveira 60816

Primeiramente, penso que o colega tenha feito um excelente trabalho na explicação de cada parâmetro, pois para além de explicar o significado de cada um ainda deu breves exemplos bastante esclarecedores. Concordo com os code smells do meu colega pois com esta métrica conseguimos facilmente identificar “Long Method”’s assim como o excesso de if’s.

—————————————————————————————————————

**MARTIN PACKAGING METRICS**

**RAFAEL COSTA (60441)**Uma imagem com mesa

Descrição gerada automaticamente

Uma imagem com mesa

Descrição gerada automaticamente

**Acoplamentos aferentes (Afferent couplings - Ca):**

O número de classes em outros pacotes que dependem de classes dentro de um pacote é um indicador da responsabilidade do pacote.

**Acoplamentos eferentes (Efferent couplings - Ce):**

O número de classes em outros pacotes dos quais as classes em um pacote dependem é um indicador da dependência de pacotes externos.

**Abstração (A):**

A relação entre o número de classes abstratas (e interfaces) e o número total de classes no pacote analisado.

O intervalo para esta métrica varia entre 0 e 1, sendo A=0 a indicação de que o pacote é completamente concreto e A=1 a indicação de que o pacote é completamente abstrato.

**Instabilidade (I):**

A relação entre o acoplamento eferente (Ce) e o acoplamento total (Ce + Ca) tal que I = Ce / (Ce + Ca).

Esta métrica é um indicador da resiliência do pacote à mudança.

O intervalo para esta métrica varia entre 0 e 1, sendo I=0 a indicação de que o pacote é completamente estável e I=1 a indicação de que o pacote é completamente instável.

**Distância da sequência principal (D):**

A distância perpendicular de um pacote da linha idealizada A + I = 1. D é calculado como o módulo de ( A + I - 1) ou seja |A + I - 1|.

Esta métrica é um indicador do equilíbrio do pacote entre abstração e estabilidade. Um pacote diretamente na sequência principal é perfeitamente equilibrado em relação à sua abstração e estabilidade, ou seja ,os **pacotes ideais** são completamente **abstratos e estáveis (I=0, A=1)** ou completamente **concretos e instáveis (I=1, A=0)**.

O intervalo para essa métrica varia entre 0 e 1, com D=0 indicando um pacote que coincide com a sequência principal e D=1 indicando um pacote que está o mais longe possível da sequência principal.

**Pontos problemáticos:**

A **instabilidade dos pacotes é visível** em todos os pacotes presentes na tabela acima representada como se pode ver através da coluna ***I*** e contudo através da coluna ***D*** a existência de pacotes ideais é elevada e muitos pacotes apresentam valores de D próximos ao que seria o valor idealizado 1, uma vez que apesar de todos os pacotes serem instáveis muitos são concretos, o que fazem com que o indicador D seja quase ideal em muitos casos. Apenas no caso do pacote net.sourceforge.ganttproject.gui.options.model o valor do indicador D apresentado é 0, ou seja não é ideal.

**Code Smells:**

Uma imagem com mesa

Descrição gerada automaticamenteComo explicado acima, os valores de **Afferent couplings (Ca)** e **Efferent couplings (Ce)**, são relativos ao número de classes que dependem ou estão dependentes de outras classes de outros pacotes, e daí, encontramos relação com o Code Smell Inappropriate Intimacy, uma vez que este é referente ao uso de campos e métodos de outras classes.

**MOOD Metrics Guilherme Abrantes (60971)**

Uma imagem com mesa

Descrição gerada automaticamente

As **mood metrics** são constituídas por 6 métricas diferentes: **MHF**, **AHF**, **MIF**, **AIF**, **PF** e **CF**.

**AHF e MHF são métricas relacionadas com a visibilidade.**

AHF  = 1 - Attributes Visible;

MHF = 1 - Methods Visible;

**MHF** tem em conta o número de métodos visíveis  no nosso projeto podemos ver um MHF de 45,86% o que indica que existem quase tantos métodos públicos como privados o que pode resultar em vários métodos especializados que não podem ser reutilizados (45,86% é um número relativamente alto quando comparado com outros projetos).

**AHF** tem em conta o número de atributos visíveis, sendo que o nosso projeto possui um AHF de 89,63% acho um número bastante aceitável onde a maioria dos atributos são privados.

**MIF e AIF são métricas relacionadas com a herança de métodos.**

MIF = inherited methods / total methods available in classes

AIF = inherited attributes / total attributes available in classes

Podemos então constatar um **MIF** de 0 que pode estar a ser causado por as classes que herdam métodos dos pais estarem a redefinir todos os métodos ou então acrescentar novos, este MIF é considerado bastante mau.

O **AIF** deste projeto é de 68,73% o que sugere que bastantes atributos são herdados pelas classes filhas, o que pode não ser o ideal.

**PF Polymorphism Factor:**

PF = overrides / sum for each class(new methods \* descendants)

A **PF** indica o grau de redefinição de métodos durante a herança de classes, podemos então observar um PF de 100% o que indica que nos estamos a dar override a tudo o que condiz com o facto do MIF ser 0%.

**CF Coupling Factor:**

CF = Actual couplings / Maximum possible couplings

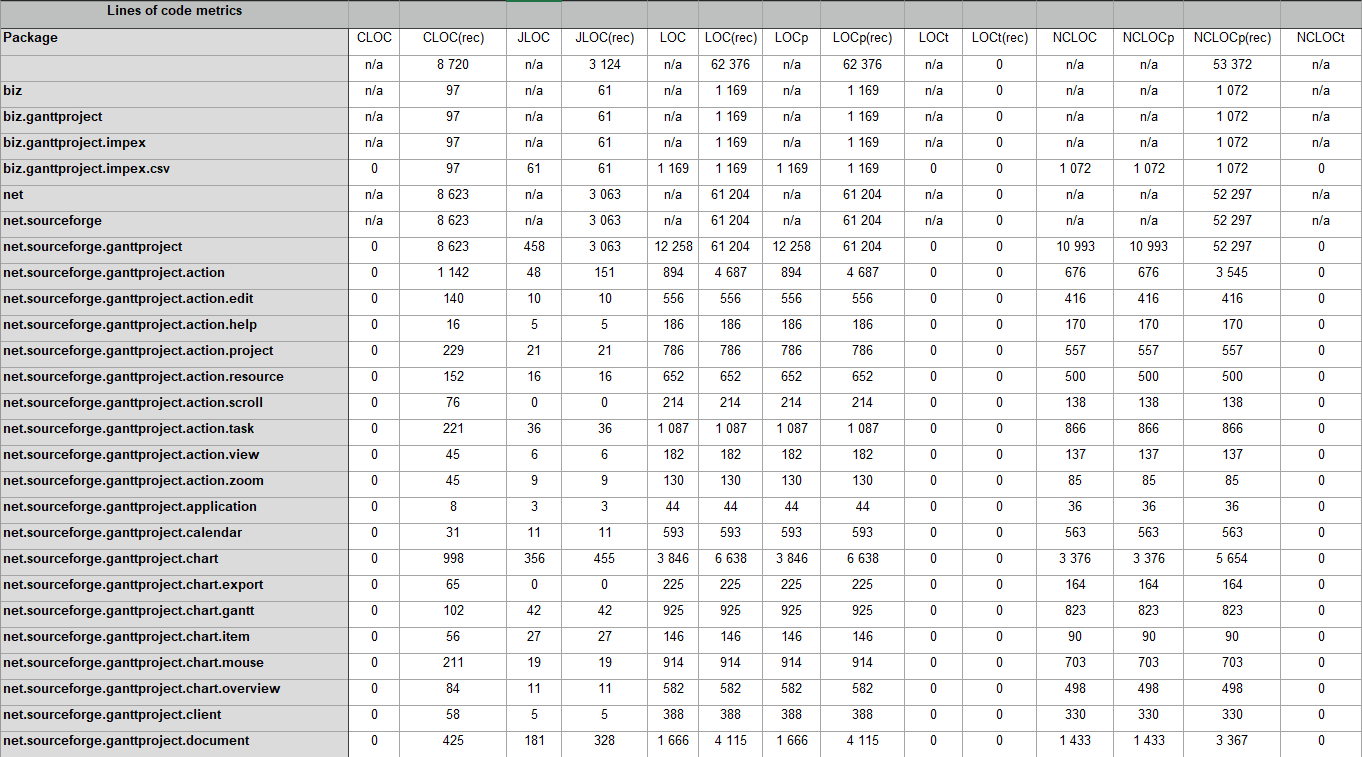
CF mede as copulações que existem, para existir uma população de A para B, A tem de chamar métodos ou variáveis que existem em B, neste projeto este valor é de 0,20% o que indica que quase não existem classes copuladas.

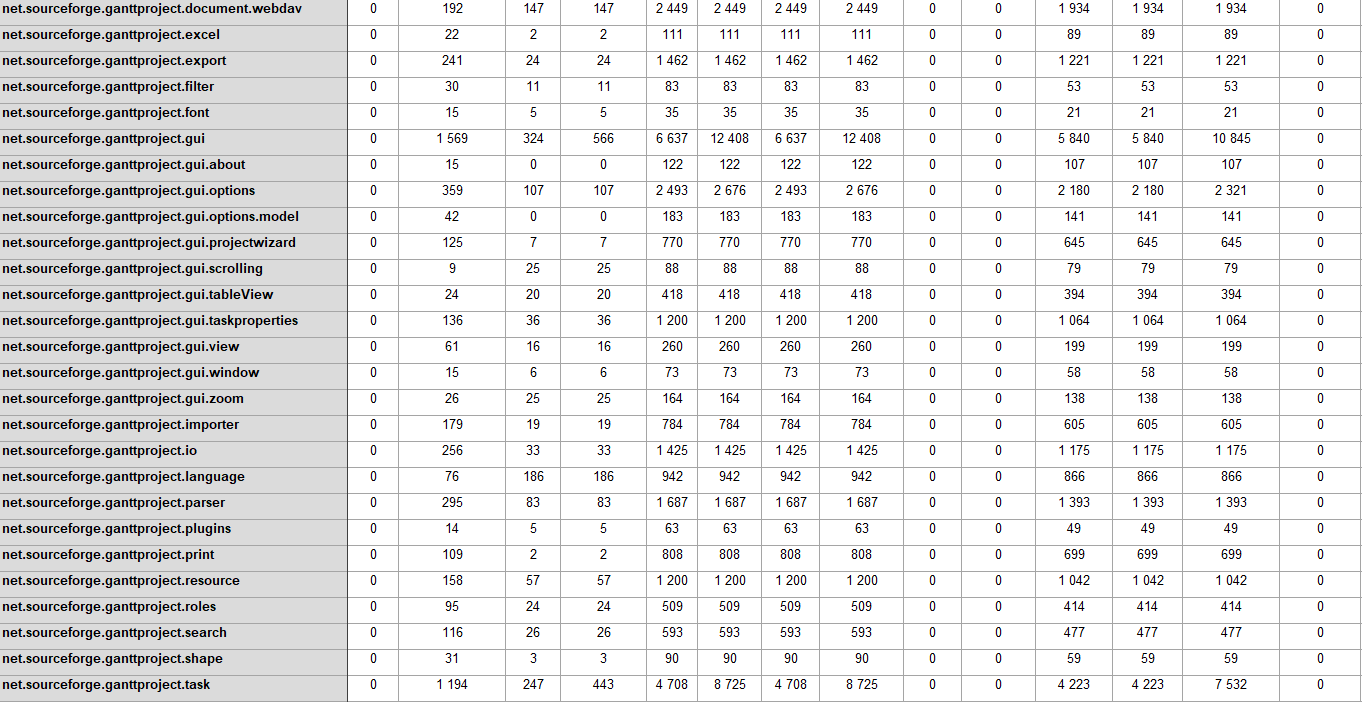
**Pontos problemáticos e code smells:**

Sobre os pontos problemáticos temos o PF e o MIF que se completam e indicam que grande parte dos métodos são redefinidos, o que leva às classes filhas a não aproveitarem os métodos que herdam dos pais, o que pode levar ao code Smell Divergent Change, como não existem métodos comuns sempre que queremos adicionar algo temos de reescrever todos os métodos que já existem.

**Uma imagem com texto

Descrição gerada automaticamente**

**Lines of Code Metrics Daniel Eugénio (59797)**



Uma imagem com parede, lotes, vários

Descrição gerada automaticamente

Obs: Para esta métrica optamos por agrupá-la por packages pois, caso contrário ficaria muito extenso.

**Alguns constituintes da métrica Lines of Code:**

**LOC - Lines Of Code**

* Todas linhas de código, incluindo comentários

**NCLOC - Non-Comment Lines of Code**

* Linhas de código excluindo os comentários

**LOCp - Lines of product code**

* Linhas de código “eficaz” (source code). Exclui comentários e linhas em branco.

**Pontos problemáticos:**

Esta métrica é difícil de avaliar, visto ser mais eficaz quando avaliada ao nível da classe, no entanto, avaliar um projeto desta dimensão à classe demoraria muito tempo, pelo que achámos melhor ver os valores ao nível da package. A nível da package, a package com mais linhas é a package “mãe”, pelo que se justifica o valor de linhas apresentado, por esta ter várias packages em si.







Aqui estão dois exemplos, ao nível da classe, de valores que estão fora do normal para a média da aplicação, o que pode implicar dificuldades na leitura e interpretação destas classes e que podiam ser divididas em classes mais pequenas.

**Code Smells**

Apesar de difícil de avaliar, esta métrica pode ser extremamente útil para identificar code smells como Large Class ou Duplicate Code, no caso de uma classe se destacar das outras no número de linhas. Outro Code Smell é o No Comments, detetável em classes onde o número de linhas seja igual, independentemente do parâmetro ser LOC ou NCLOC.

Em relação à análise do meu colega, concordo que esta métrica é difícil de ser avaliada no formato de packages, em relação aos code smells concordo que possa ajudar a identificar os code smells referidos mas também o code smell “Comments” que é identificado quando uma classe tem excesso de comentários.

Reviews

—————————————————————————————————————Miguel Agostinho 60677

Em relação à análise do meu colega, concordo que esta métrica é difícil de ser avaliada no formato de packages, em relação aos code smells concordo que possa ajudar a identificar os code smells referidos mas também o code smell “Comments” que é identificado quando uma classe tem excesso de comentários.

—————————————————————————————————————

**Dependency Metrics Francisco Silveira (60816)**

Uma imagem com texto, documento

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, documento

Descrição gerada automaticamente

**Dependency Metrics**

Uma métrica dependente é uma métrica em que suas variáveis de influência são explicitadas. Um exemplo é a escalabilidade da taxa de transferência em relação ao número de clientes que acessam uma conexão de dados. Aqui, uma métrica (a taxa de transferência) é fornecida em

dependência de outra variável (o número de clientes).

**Cyclic:**

* número de dependências cyclic por package. Esta ocorre quando um package A depende de outro package B, e o B também depende do A.

**PDcy:**

* número de dependências por package,ou seja, por quantos packages está dependente o package em questão.

**PDpt:**

* número de packages dependentes,ou seja,quantos packages estão dependentes do package em questão.

**PDpt\*:**

* número de packages dependentes transitivamente. Considerando três packages: A, B,C. Se C for dependente de B e B for dependente de A então C é uma dependência transitiva para A.

**Pontos Problemáticos:**

Como podemos ver nas colunas “Cyclic” e “PDpt\*"Os valores apresentados são praticamente os mesmos, ou seja, não conseguimos observar algo em concreto apenas que estes packages dependem quase todos uns dos outros.

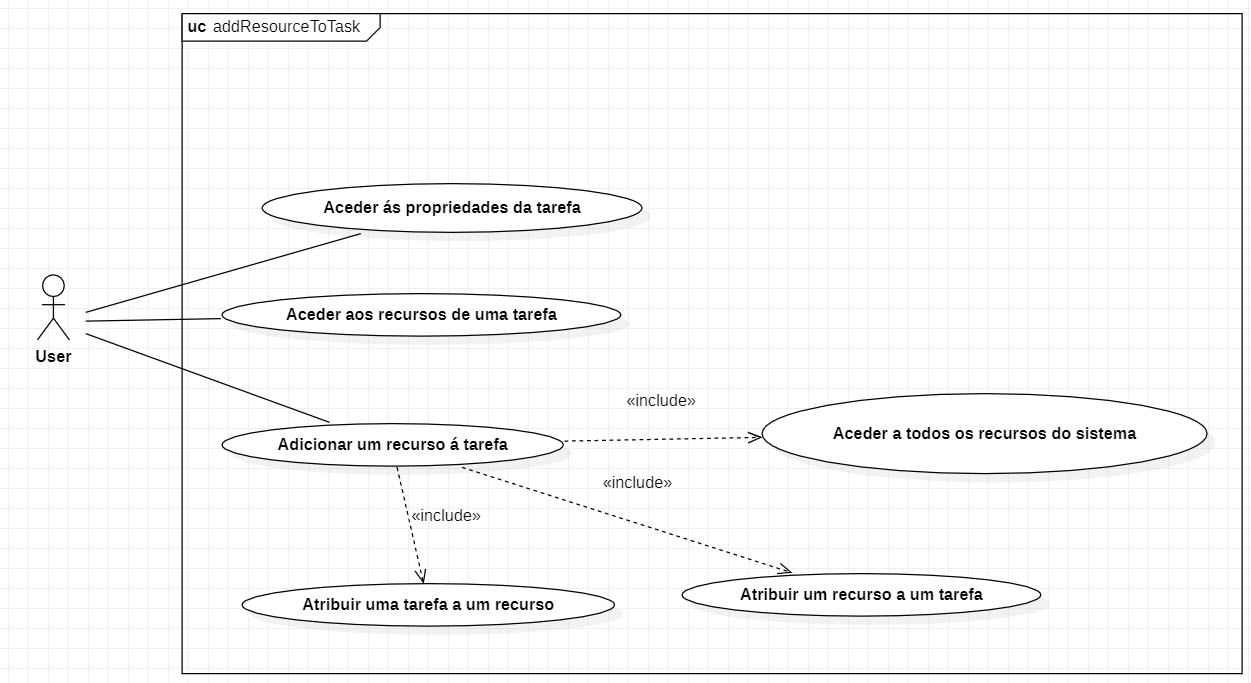
**Code Smells:**

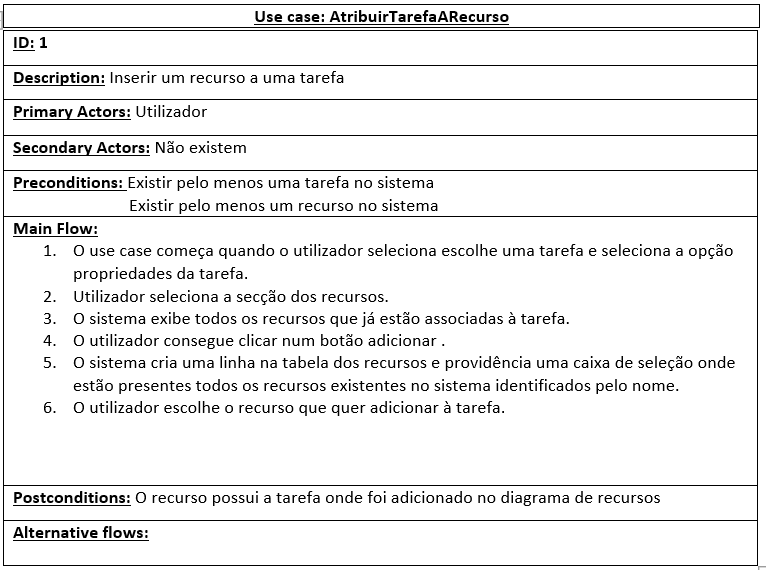
Esta métrica pode ajudar-nos a identificar vários tipos de code smells como por exemplo o Shotgun Surgery no caso de querermos alterar algo simples no programa mas temos que alterar várias classes para o fazermos. Outro code smell que podemos identificar é o Feature Envy no caso de existir um método que está mais preocupado em manipular dados de outra classe em vez da sua. Estes code smells podem ser identificados observando as suas dependências.No caso do Feature Envy podemos observá lo se existir uma classe com poucas dependências e muitos dependentes.

Atenção estes exemplos podem ocorrer (ou ocorrer de outra maneira dependendo do programa em questão) e não existir nenhum code smell apenas são algumas maneiras de podermos identificá-los.

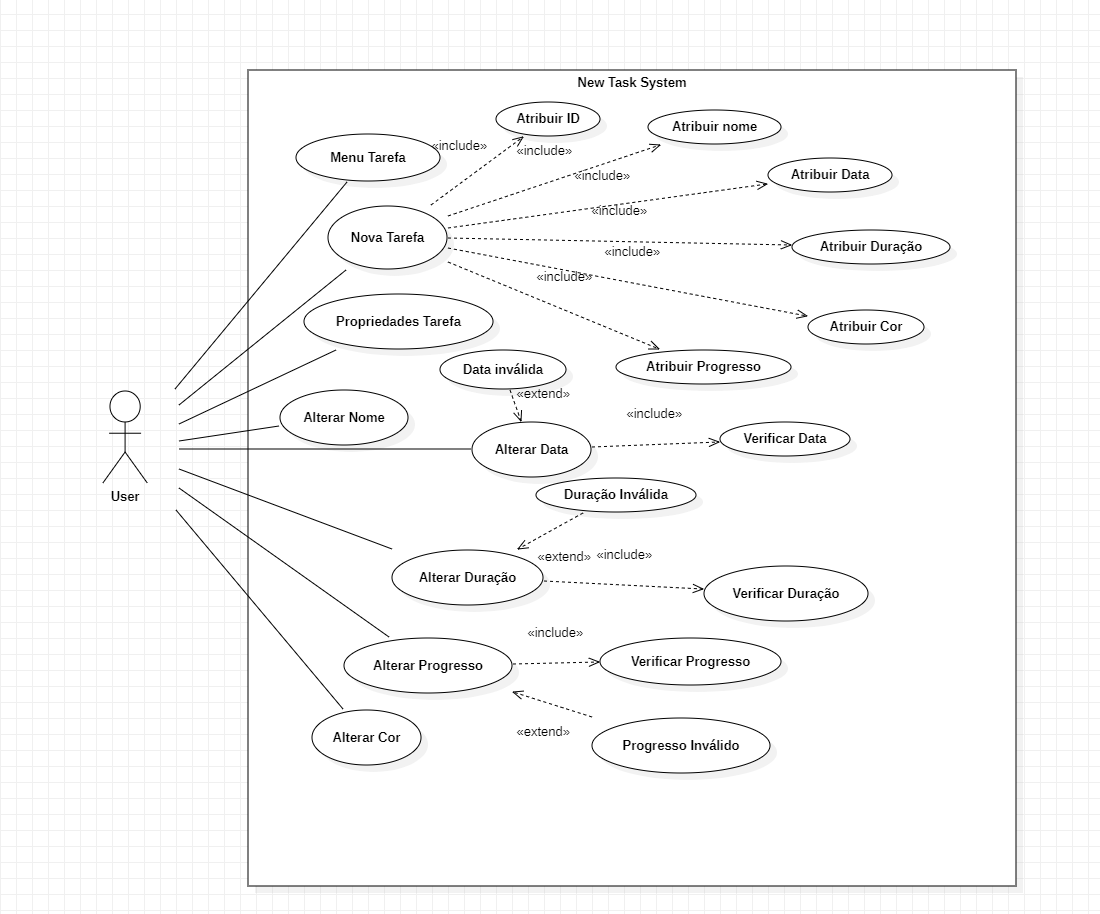
Uma imagem com mesa

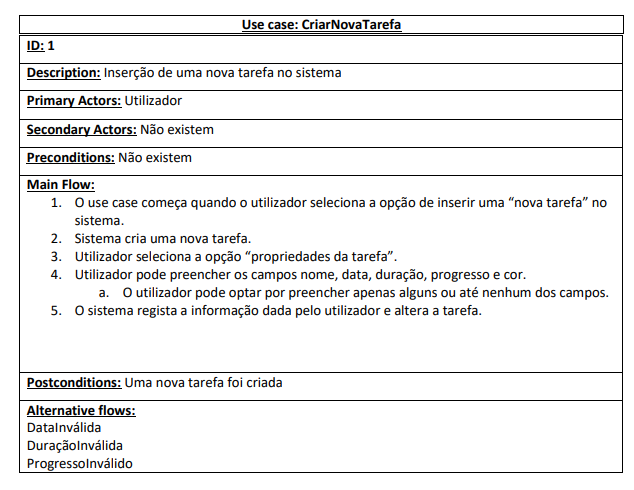
Descrição gerada automaticamente

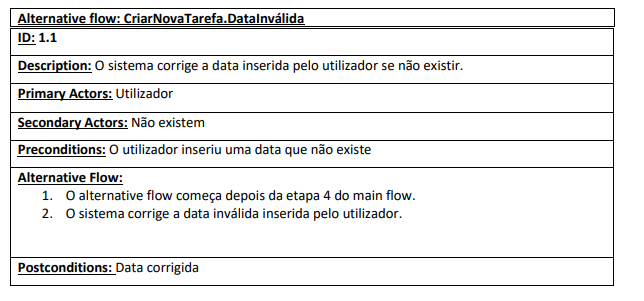
**Use Case Guilherme Abrantes (60971)**

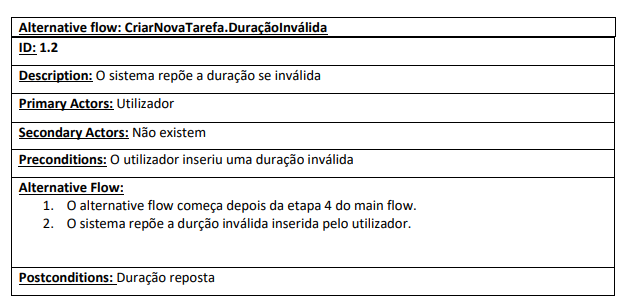


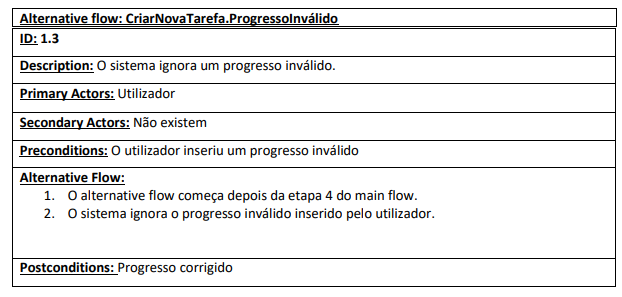
**Use Case Miguel Agostinho (60677)**



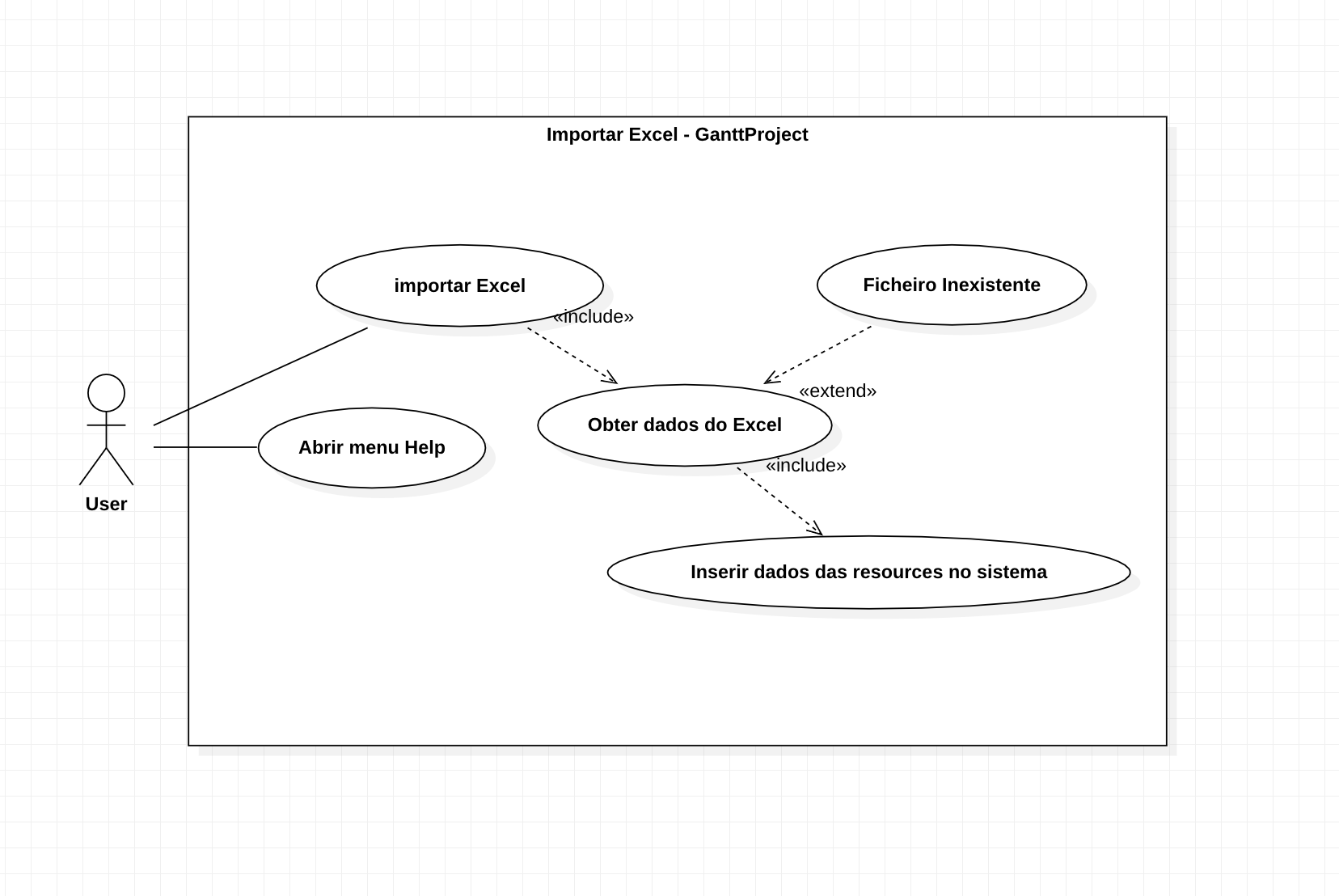


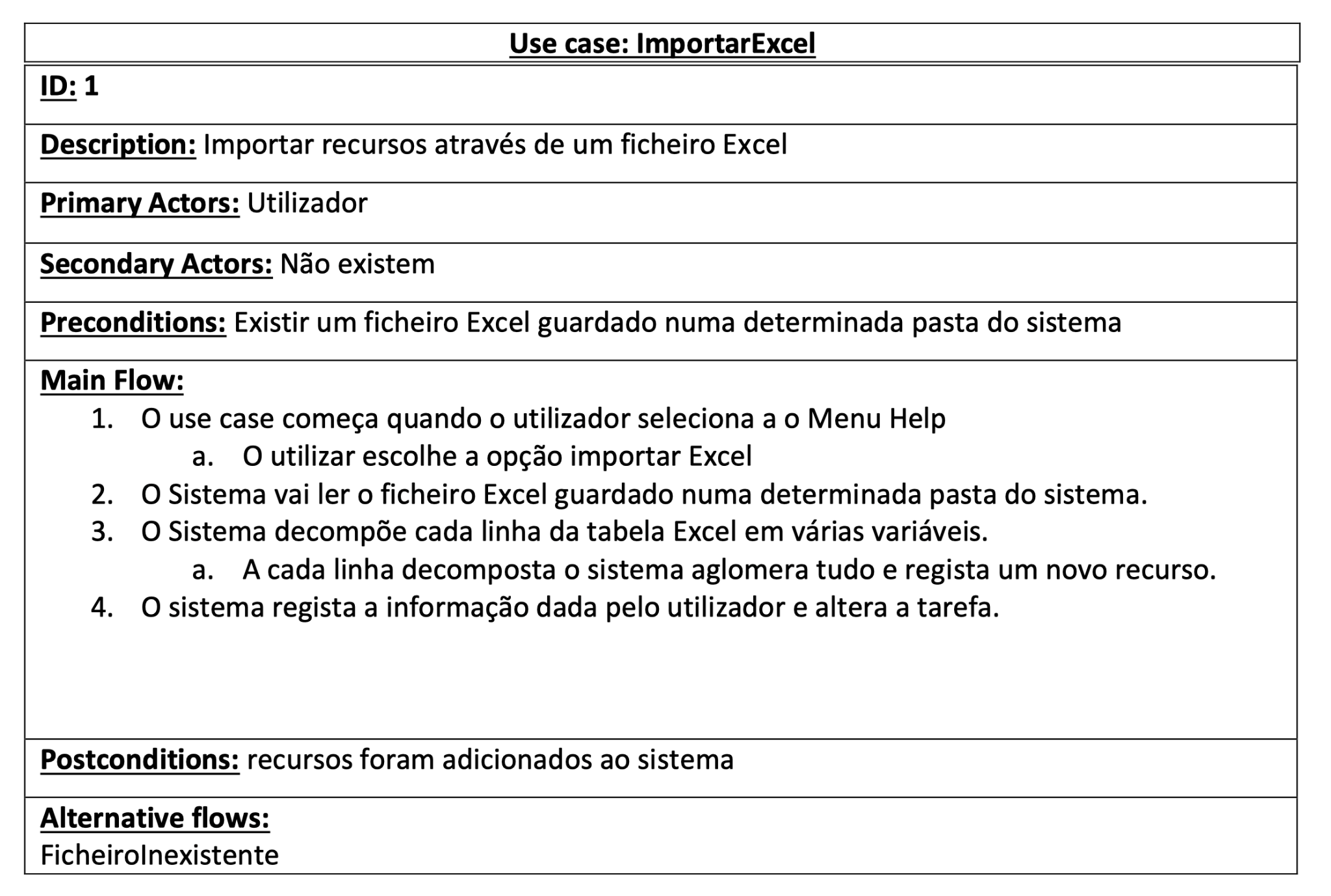


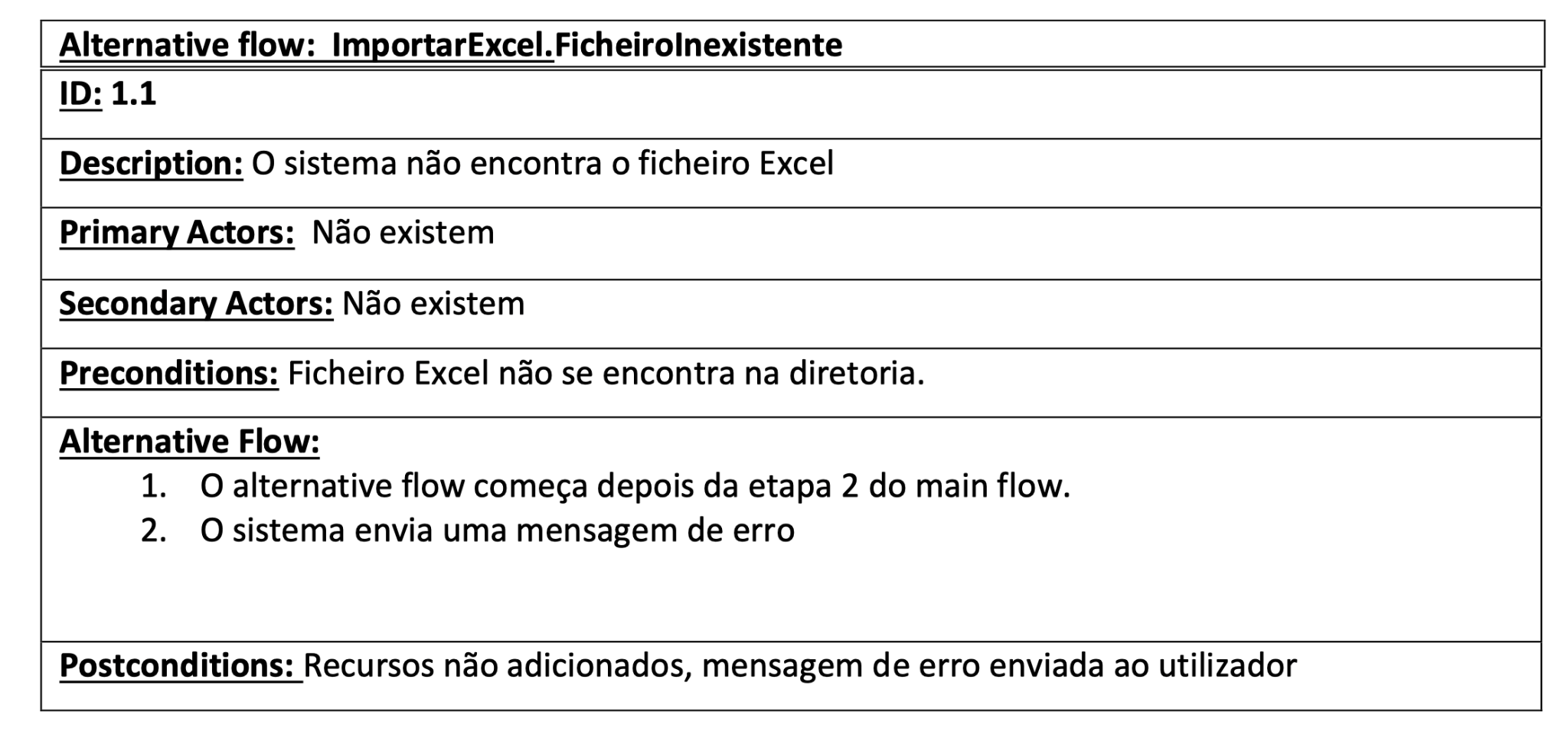




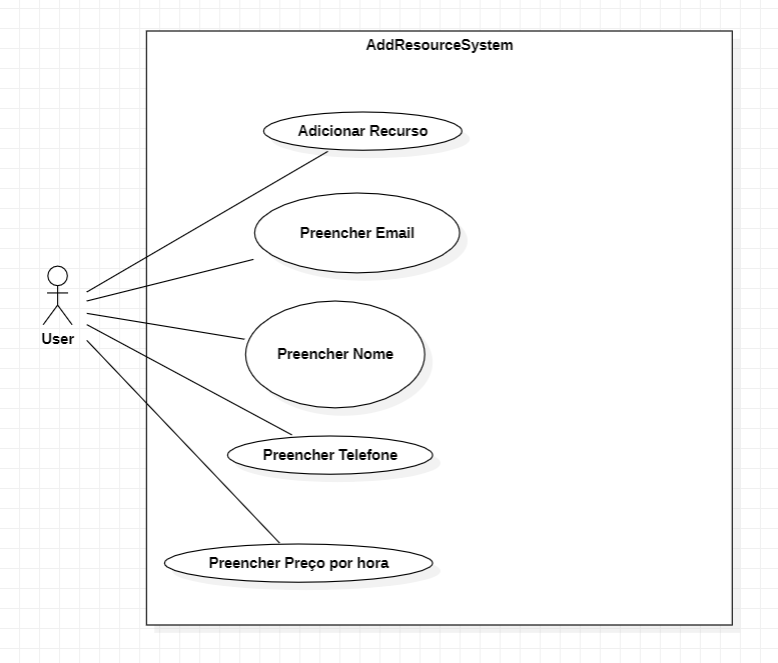
**Use Case Rafael Costa (60441)**

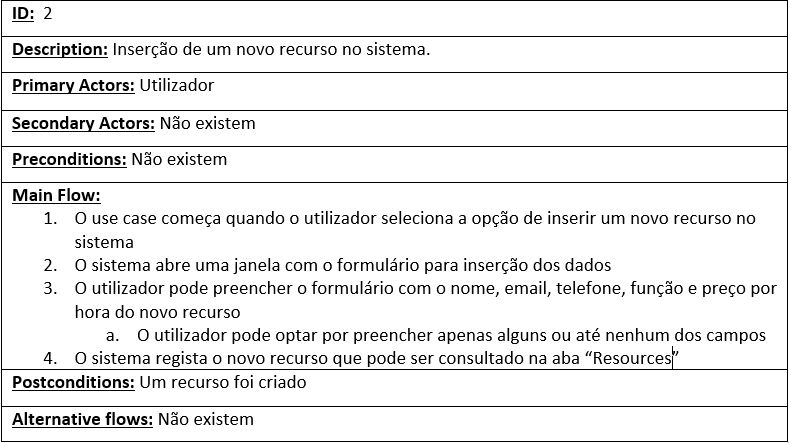


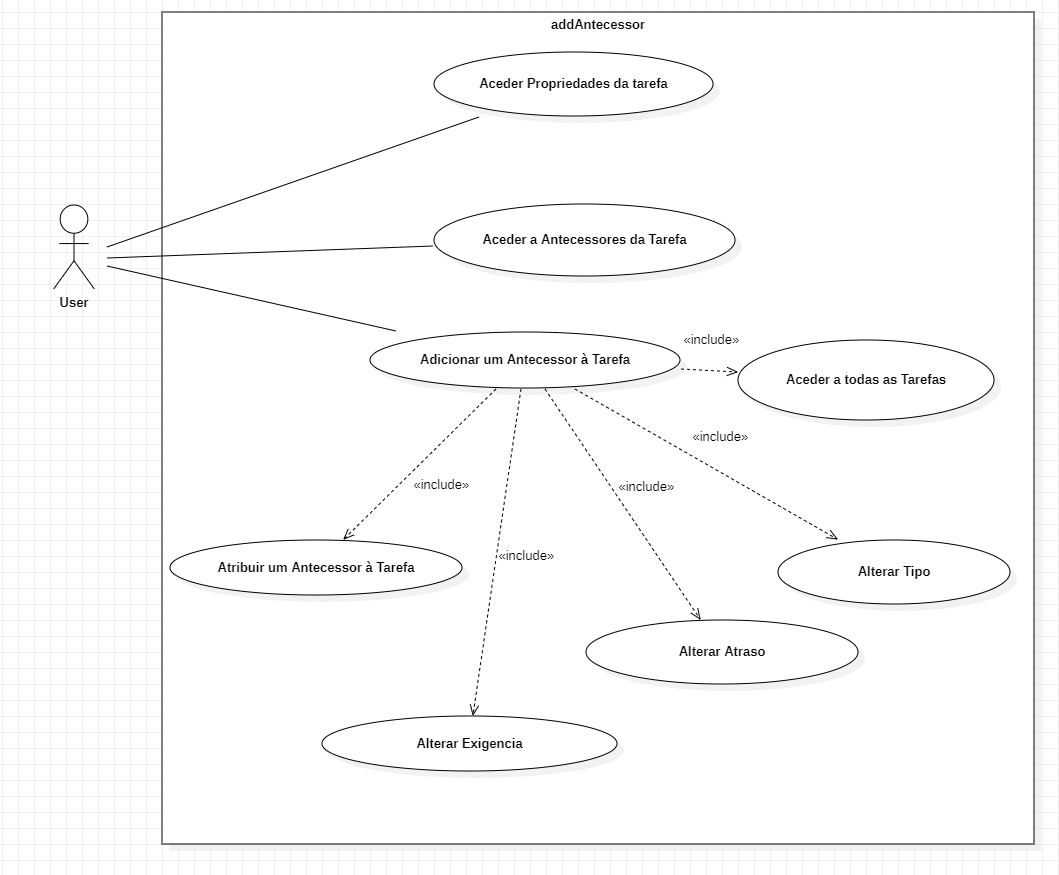


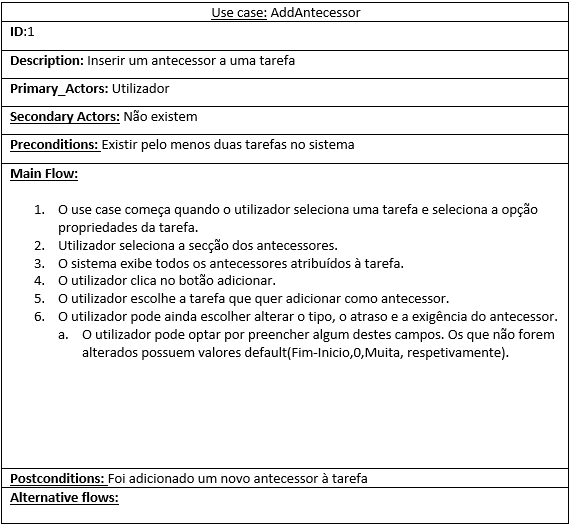


**Use Case Daniel Eugénio (59797)**





**Use Case Francisco Silveira (60816)**



**\**

**User Stories Daniel Eugénio (59797)**

* As a user I want to be able to view all the resources in the system so that I don't add the same person twice.
* As a user I want to be able to insert new columns to the resources so that I can complete their information as I see fit.

**User Stories Rafael Costa (60441)**

* As a user I want to be able to import resources to the system using excel so that It will not be necessary to add a lots of resources by hand
* As a user I want to save the project file so that I can send to my co-workers.

**User Stories Guilherme Abrantes (60971)**

* As a user I want to see what tasks each resource has so I don't waste time going through every task to check.
* As a user I want to zoom out my calendar so that I don't waste time scrolling right to plan my work.

**User Stories Miguel Agostinho (60677)**

* As a user I want to add tasks so that I can manage my project.
* As a user I want to highlight completed tasks so that I can see these tasks better.

**User Stories Francisco Silveira (60816)**

* As a user I want to add predecessors to my tasks so that I can organize which tasks to do first.
* As a user I want to add a resource's days off so you can organize the project without delay.