Bruno Gabriel Andrade Ramires da Silva Jota

Nº 43277

Licenciatura em Engenharia Informática

Método de tratamento de *tokens* para a identificação de *concerns* em código *MATLAB*

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
Maio, 2019

Orientador: Miguel Jorge Tavares Pessoa Monteiro
Professor auxiliar
Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Coorientador: Nuno Marques
Professor auxiliar
Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Constituição do júri

Presidente: Professor Doutor Francisco de Moura e Castro Ascensão de Azevedo, FCT
Vogais: Professor Doutor Luis Manuel Pereira Sales Cavique Santos, Universidade Aberta
Professor Doutor Miguel Jorge Tavares Pessoa Monteiro, FCT

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Método de tratamento de *tokens* para a identificação de *concerns* em código *MATLAB*

À minha mãe e falecidos avós

Agradecimentos

Os meus primeiros agradecimentos destinam-se aos meus orientadores, o professor Miguel Monteiro e o professor Nuno Marques, ambos da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa. Sem o apoio incondicional deles não seria possível ultrapassar de forma rápida e com sucesso as várias barreiras que foram surgindo no desenvolvimento desta tese. Agradecer também a disponibilidade com que responderam aos (imensos) emails que lhes enviei, mesmo com a grande carga horária que lhes é atribuída.

O meu maior agradecimento vai, sem dúvida, para a minha mãe. Ao longo destes anos foi o meu grande suporte e a pessoa que me motivou mais para terminar este curso de forma meritória. Sei do imenso esforço que foi e é feito da sua parte para que eu e os meus irmãos usufruamos das melhores competências e capacidades. Agradecer ao meu padrasto e irmãos pelos bons momentos que me fizeram ter ao longo deste percurso, fazendo-me muitas vezes abstrair do "peso" da faculdade.

Um agradecimento bastante especial à minha namorada que me apoiou desde o primeiro momento, tanto em circunstâncias boas como em menos boas. Foi a pessoa que mais conselhos me deu sempre em prol do que é melhor para mim. Agradecer também a sua disponibilidade e boa disposição que me incentivou e permitiu que ao longo destes anos terminasse cada etapa junto dela.

Foram muitas as pessoas que me acompanharam nesta longa caminhada, desde familiares a amigos. A presença deles ajudou-me, de uma ou outra forma, a perceber que cada um deles é importante na minha vida e que nada se consegue sozinho.

Por último, e não menos importante, gostaria de agradecer aos meus avós maternos, que, apesar de já não estarem fisicamente presentes, foram e continuam a ser um grande incentivo na minha vida. Como tal, sei do quão orgulhosos estariam se cá estivessem visto que um dos maiores desejos deles era ver os seus netos diplomados.

Resumo

Em engenharia de *software*, o termo *concern* é descrito como uma qualquer abstração, conceito ou conjunto consistente de responsabilidades que gostaríamos de localizar no seu próprio módulo. No entanto, os *paradigmas* de programação têm bastante dificuldade em modularizar os *concerns* em sistemas de *software*. A presença de *concerns* ao longo de vários sistemas pode ser detetada através de sintomas a que esses *concerns* dão origem no código, com destaque para *concerns* não-modularizados. Este trabalho contribui para a maturação duma técnica destinada a detetar e analisar a presença de *concerns* em sistemas *MATLAB*, baseada no mapeamento de grupos de *tokens* a *concerns* específicos.

A tese a ser desenvolvida pretende validar e melhorar essa técnica. Para tal, esta tese emprega métodos da linguística computacional, dos quais se destaca a informação mútua pontual, que permitem validar e refinar os grupos de *tokens* relacionados com determinados *concerns*. A análise dos resultados deste tipo de métricas, recolhidas com base num repositório de código *MATLAB* específico, pretendem analisar a proximidade entre *tokens* através das suas ocorrências e coocorrências nesse mesmo repositório. A análise e comparação entre estas métricas poderá indicar alterações relativamente às relações existentes entre *tokens* e *concerns* sobre a abordagem existente. Dessa forma, pretende-se que este trabalho valide essa abordagem fazendo a correspondência indicada entre *concerns* e *tokens* para a identificação de *concerns* no código *MATLAB*.

Palavras-chave: *MATLAB*, modularidade, *concerns* transversais, *token*, métricas de associação de palavras, informação mútua pontual

Abstract

In software engineering, a concern is any abstraction, concept or consistent set of responsibilities that we would like to locate in its own module. However, programming paradigms have a lot of difficulties in modularizing concerns in software systems. The presence of concerns along various systems can be detected through the symptoms that these concerns produces in the code, with emphasis on non-modularized concerns. This work contributes to a maturation technique to detect and analyze the presence of concerns in MATLAB systems, based on the mapping of groups of tokens to specific concerns.

The aim of the current thesis is to validate and improve this approach. For such, this thesis applies computational linguistics methods, highlighting the pointwise mutual information, which allows to validate and refine the groups of tokens related to certain concerns. The analysis of the results of this type of metrics, collected based on a specific repository of MATLAB code, intends to analyze the proximity between tokens through their occurrences and co-occurrences in that same repository. The analysis and comparison between these metrics may indicate changes in relation to the existing associations between tokens and concerns about the existing approach. For this reason, this work intends to validate the approach above-mentioned by making the indicated match between concerns and tokens for the identification of concerns in the MATLAB code.

Keywords: MATLAB, modularity, crosscutting concerns, token, word association techniques, pointwise mutual information

Índice

ÍNDICE DE FIGURAS

Índice de Tabelas

Índice de Equações

1. Introdução

Este documento é uma dissertação escrita para o mestrado em engenharia informática na Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa. Esta tese pretende dar continuidade a alguns trabalhos de investigação realizados anteriormente com foco no estudo da modularidade em sistemas *MATLAB* [17] [18] [19] [22].

1.1. Separação de *concerns*

Um *concern*, em engenharia de *software*, é definido como um conjunto consistente de responsabilidades que gostaríamos de localizar no seu próprio módulo para que seja possível a sua compreensão e evolução. A separação de *concerns* pretende separar um programa computacional em secções distintas de modo a que cada secção aborde/corresponda a um *concern* específico. Quando a separação de *concerns* é bem feita dá origem a um programa modular. No entanto, a separação de tais *concerns* não se apresenta como uma tarefa fácil e, muitas das vezes, essa separação não é facilmente visível ao longo de um programa. Por seu lado, o princípio da responsabilidade única indica que uma função, classe ou método deve fazer apenas uma coisa, ou seja, fazer aquilo que lhe é associado, bem, e que não faça mais nada [68].

A vantagem de se realizar a separação de *concerns* é a possibilidade de se simplificar o desenvolvimento e manutenção de um programa. Quando os *concerns* são bem separados, várias secções individuais podem ser reutilizadas, bem como desenvolvidas e atualizadas independentemente. Assim, a capacidade de melhorar ou modificar posteriormente uma secção do código sem precisar de conhecer os detalhes de outras secções é bastante vantajosa [68].

O *software* moderno é bastante complexo e essa complexidade tem vindo a aumentar. Uma das formas tradicionais dos programadores lidarem com a complexidade é através de *paradigmas*, ou seja, de princípios e regras que prescrevem como decompor um sistema em unidades modulares e como essas unidades são relacionadas e compostas. O *paradigma* de programação dominante é a programação orientada a objetos. Este *paradigma* torna-se interessante porque as unidades de decomposição que ele suporta (classes) são módulos completos que se ajustam bem aos domínios e implementação de *concerns* [20].

1.2. Contexto geral desta tese

O principal objetivo do trabalho onde esta tese se insere é identificar a presença de *concerns* (quer sejam modularizados ou não) em código *MATLAB*. A identificação de *concerns* utiliza, até ao momento, uma abordagem baseada em *tokens* que permite a sua localização em sistemas *MATLAB*, existindo um mapeamento entre *concerns* e *tokens* que facilita a deteção dos *concerns* (ver secção 2.3).

1.2.1. Problema: Limitações no suporte à modularidade em sistemas *MATLAB*

As linguagens de programação para manipulação de dados e matrizes são uma ferramenta essencial para análise de dados, como é o caso de *MATLAB*. No entanto, este tipo de linguagens são, muitas vezes, não-estruturadas e com grande perda de modularidade.

Os *concerns* que não se encontram modularizados e estão espalhados por trechos de código dificultando a sua própria localização dá-se o nome de *concerns* transversais (*crosscutting concerns* - *CCCs*). O termo *crosscutting* (transversal) é usado para descrever esses *concerns* porque o seu código atravessa a estrutura modular do sistema de *software* [17] [22], mesmo em situações que os programadores usem as melhores práticas de *design* e estilo de programação.

A Figura 1.1 mostra uma função em *MATLAB* onde é bem visível a presença de vários *concerns* ao longo das linhas de código (a partir das várias cores identificadas). A presença desses *concerns* é detetada através de nomes de funções que estão claramente associados a *concerns* a partir de uma tabela já existente que faz o mapeamento entre *concerns* e nomes de funções (ver Tabela 2.2 da página 20).

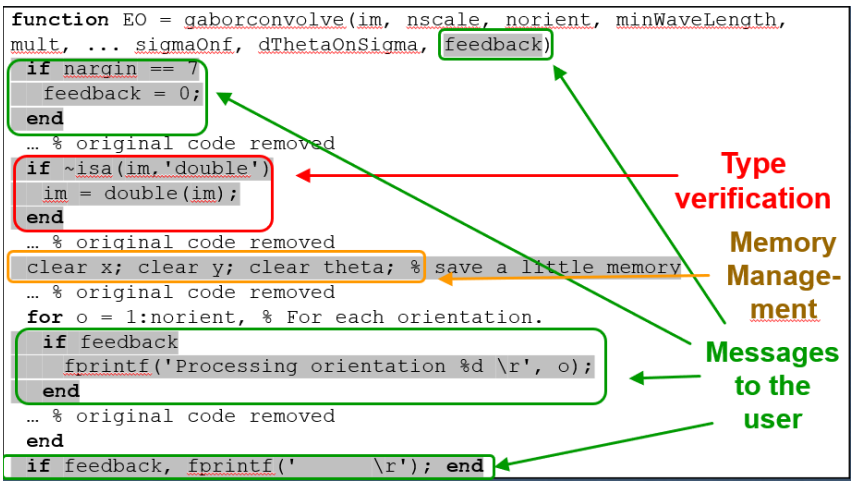


Figura 1.1: Identificação de *concerns* numa função *MATLAB*

Até dada altura, a maior parte dos trabalhos de pesquisa sobre deteção de *concerns* não-modularizados focava-se, sobretudo, em sistemas orientados a objetos (OO) e em C, deixando de parte linguagens como *MATLAB* [17]. A modularidade suportada em *MATLAB* é menos sofisticada comparada com as linguagens OO, visto que os seus módulos são principalmente ficheiros e funções *MATLAB*.

A identificação de *concerns* em sistemas *MATLAB* tem sido alvo de vários estudos ao longo dos últimos anos. Uma das técnicas que foi desenvolvida e que ajuda nessa deteção é uma abordagem que utiliza *tokens* (ver definição de *token* na secção 2.3.2) para a localização de *concerns* (ver abordagem na secção 2.3.3). Assim, existe uma correspondência entre *tokens* e *concerns* (para a abordagem proposta) que é representada através de uma tabela (ver Tabela 2.2 da página 20) e que será analisada e validada neste trabalho por apresentar novas oportunidades de melhoria [19] [22].

1.2.2. Modularidade em linguagens orientadas a objetos

Os *paradigmas* da programação têm bastantes insuficiências naquilo que é a capacidade de incluir todos os *concerns* em módulos separados [14]. Esta limitação deve-se ao fato de cada *paradigma* de programação fornecer um único critério para decompor um sistema de *software* [32].

Sempre que um ficheiro de código possui mais do que um *concern*, então esse ficheiro, além de possuir o *concern* principal, possui também um ou mais *concerns* secundários (que neste caso correspondem a *CCCs* em *MATLAB*). A presença de tais *concerns* origina sintomas indesejáveis no código.

A Figura 1.2 ilustra um exemplo numa linguagem OO (*java*) onde é notória a presença de *CCCs* neste tipo de linguagens. Neste caso, o *concern* secundário é a apresentação gráfica de dados, ou seja, um *concern* de exibição. A presença de tal *concern* no código indica que esse *concern* é não-modularizado tornando-se difícil desacoplar esse *concern* do código. Existem duas classes (*Point* e *Line*) que implementam o conceito abstrato representado pelo tipo *FigureElement*. As duas classes têm operações que permitem alterar os valores dos elementos da figura. A classe *Display*, que também está presente na Figura 1.2, é uma funcionalidade que fornece uma apresentação gráfica dos elementos da figura.

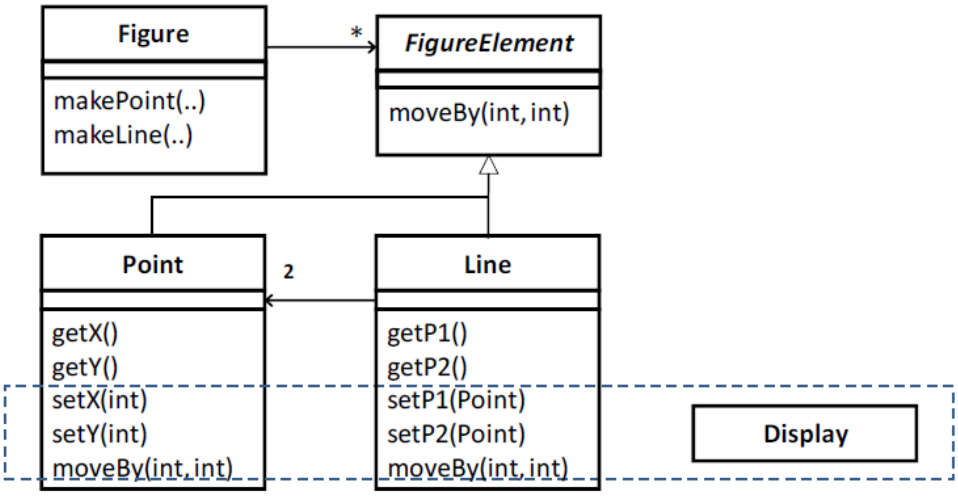


Figura 1.2: Diagrama de classes para justificar *CCCs* em *java* [19]

O grande problema encontrado neste exemplo passa pela atualização das unidades de decomposição (classes *Point* e *Line*). Sempre que existem alterações nos elementos da figura, essas alterações devem ser propagadas para a classe *Display*. A Figura 1.3 demonstra que o *concern* secundário se encontra espalhado ao longo das linhas de código (*concern* não-modularizado) dificultando a tarefa de desacoplar esse *concern*. Caso este *concern* estivesse modularizado, acoplar/desacoplar o *concern Display* estaria significativamente facilitado. A Figura 1.3 mostra, destacado a cinza, o código relacionado com o *concern* secundário de exibição, além da presença do *concern* principal do módulo. A classe seguinte pertence à classe *Point* embora outras classes apresentassem também *CCCs* ao longo do próprio código.

```
public class Point implements FigureElement {
        private int _x, _y;
        private Display _display;

        public Point (int x, int y) {
                _x = x;
                _y = y;
        }
        public Point (int x, int y, Display display) {
                this(x,y);
                setDisplay (display);
        }
        public void setX (int x) {
                _x = x;
                _display.update (this);
        }
        public void setY (int y) {
                _y = y;
                _display.update (this);
        }
        public void setDisplay (Display display) {
                _display = display;
        }
        public void moveBy (int dx, int dy) {
                _x += dx;
                _y += dy;
                _display.update (this);
        }
}
```

Figura 1.3: Exemplo de *CCCs* em *java* [19]

O exemplo dado demonstra a presença de *concerns* transversais em sistemas mais generalizados, como é o caso de *java*. Estes problemas também se verificam em sistemas *MATLAB* (ver secção 2.3.1) embora com características diferentes [19]. No caso de *MATLAB*, as características da sua linguagem diferem das linguagens tradicionais OO fazendo com que haja diferenças na forma como se estuda a presença de *CCCs* em ambas as linguagens. Em *MATLAB*, as suas principais componentes de decomposição são predominantemente funções e grupos de funções. As funções e grupos de funções em *MATLAB* utilizam um mecanismo menos eficaz para modularizar *concerns* do que os módulos de classe em linguagens OO. Esta realidade faz com que os *concerns* não estejam tão bem organizados em sistemas *MATLAB* e, portanto, torna-se mais complicado lidar com a presença de *CCCs* visto que o suporte dado por esta linguagem é menos sofisticado do que o das linguagens OO. Além disso, a utilização de *MATLAB* difere das linguagens OO o que pode originar sintomas de código diferentes [19].

1.3. Objetivos

Nesta tese, pretende-se validar e refinar a abordagem baseada em *tokens* através do estudo das diversas relações *token*/*token* e *concern*/*token*. Serão abordados diversos métodos e ferramentas já existentes. Assim, são propostos como objetivos concretos:

1 - Estudar de que forma métodos da linguística computacional como a informação mútua pontual (ver secção 2.4.1) e outras métricas de associação de palavras (ver secção 2.4.2) podem auxiliar na validação/maturação da abordagem baseada em *tokens* de uma forma semi-automática consoante a coocorrência de *tokens*;

2 - Estudar a coocorrência entre *tokens* e palavras reservadas *MATLAB* através do mapeamento entre *tokens* e *concerns* específicos (usando as métricas de associação de palavras mencionadas no ponto 1). Esta análise terá como auxílio exemplos concretos de código *MATLAB* para que se perceba em que contextos esses pares são utilizados;

3 - Contribuir para melhorar o conhecimento existente da representação dos diversos *concerns* no repositório de código *MATLAB* usado para este trabalho, nomeadamente, se o reconhecimento de *concerns* requer estruturas mais elaboradas do que *tokens* individuais.

1.4. Estrutura da tese

O restante documento está organizado da seguinte maneira:

- **Capítulo 2: Deteção de *concerns* em *MATLAB*** aborda algumas características da linguagem *MATLAB*, estabelece o papel que certas métricas de *software* podem desempenhar neste trabalho, contextualiza a deteção de *concerns* em código *MATLAB* e, por último, são analisadas as métricas que serão exploradas durante a realização da tese;

- **Capítulo 3: Ferramenta para extração e processamento de métricas** esclarece alguns detalhes e ajustes feitos acerca da ferramenta usada para obtenção dos resultados. Este capítulo explica também o processo que se realiza com as métricas para validação da tabela de *tokens*;

- **Capítulo 4 e 5: Validação** fazem uma análise aos resultados obtidos sobre as métricas utilizadas;

- **Capítulo 6: Conclusões e trabalho futuro** apresenta uma síntese do trabalho realizado, incluindo resultados, contribuições e trabalho futuro;

- **Anexos** onde são apresentados alguns resultados da validação e código *java* relacionado com a ferramenta do capítulo 3.

2. Deteção de *concerns* em *MATLAB*

MATLAB (MATrix LABoratory) é uma linguagem de programação usada por múltiplas comunidades científicas, desde a engenharia até à pesquisa. *MATLAB* está presente nas mais variadas áreas como é o caso da computação, processamento de sinal e imagem, sistemas de controlo, entre outros [18]. Na prática, verifica-se que muitos dos utilizadores de *MATLAB* não têm a sua formação base em programação e, como tal, o seu foco enquanto desenvolvem código não é centrado na estrutura do mesmo (causando deficiências nas características do código) [8]. Como consequência desse fato, o uso desta linguagem pelos mais diversos utilizadores de cada área faz com que nem sempre se recorram às melhores práticas de organização modular dos programas. Essas diferenças que existem entre os vários desenvolvedores em *MATLAB* determinam a importância com que cada um deles atribui aos aspetos e limitações no suporte à modularidade dada pela própria linguagem [8].

2.1. Visão geral da linguagem *MATLAB*

Esta secção trata de analisar a linguagem *MATLAB* numa vertente que remete para o problema a desenvolver nesta tese. Será descrito com algum detalhe aquilo que é a sintaxe da linguagem *MATLAB*, sendo que essa sintaxe é composta por funções, *scripts*, variáveis, matrizes, operadores e gráficos. Além disso, veremos outras definições particulares desta linguagem, nomeadamente, as unidades modulares como *m-files* e *toolboxes*. Esta análise recorre a alguns exemplos de possíveis padrões sintáticos presentes em *MATLAB* que poderão fornecer informações relevantes para o estudo da relação entre nomes de funções.

2.1.1. *M-files*

MATLAB permite ao programador escrever dois tipo de *m-files*: ficheiros de funções e ficheiros de *scripts*. O nome de ambos os tipos de ficheiros deve terminar com a extensão *.m* para que sejam sempre compatíveis com *MATLAB* [8]. Na prática, os *m-files* são usados para organizar um programa *MATLAB* em módulos. Nesta tese, a análise particular dos *m-files* é indiferenciada, ou seja, os vários estudos a realizar são feitos a um nível global do repositório utilizado (ver repositório na secção 2.3.4).

2.1.2. Funções

As funções em *MATLAB* permitem receber tantos *inputs* quantos aqueles que desejamos. Por consequência, também permitem uma infinidade de *outputs*. Para que o nome da função seja válido é necessário que esse comece com um carater alfabético, e pode conter números, letras ou *underscores* [45]. Todos os ficheiros estudados neste trabalho são, maioritariamente, ficheiros de funções. As funções em *MATLAB* fazem parte do *m-file*, estruturando o ficheiro e, como tal, desempenham um papel fundamental no estudo da modularidade neste trabalho. Na Figura 2.1 é apresentada uma função relativamente simples para que se perceba melhor como se estruturam os seus elementos:

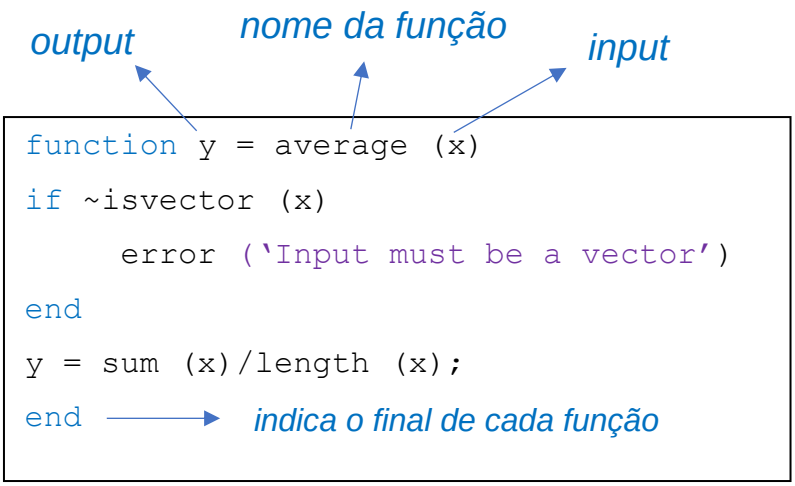


Figura 2.1: Exemplo de uma função em *MATLAB* com apenas um *output* (y) [45]

A Figura 2.1 demonstra uma estrutura de controlo através da validação do argumento x. A palavra reservada *if* aparece muitas vezes em código *MATLAB* relacionada com nomes de funções, como *nargin* ou mesmo *isvector*, podendo ser uma referência interessante para o estudo de relações entre nomes de funções. Esta análise torna-se relativamente interessante para o estudo da tabela de *tokens* da página 20 dada as possíveis relações que a palavra reservada *if* pode ter com alguns dos *concerns*.

2.1.3. *Scripts*

Scripts podem ser vistos como um conjunto de operações que são executadas sequencialmente de forma a correr um dado programa (equivale, de certa forma, a funções ou métodos *main* em outras linguagens). Operações essas que contêm estados *MATLAB*, variáveis, operadores ou chamadas a funções. Ficheiros de *script* não aceitam *inputs* nem devolvem *outputs* [61]. Como *scripts* correspondem a *m-files* sem funções, eles são ignorados no decorrer desta tese.

Este tipo de ficheiros são relativamente triviais visto que os comandos que se encontram num *script* correspondem exatamente àquilo que teríamos que escrever na linha de comandos. O exemplo da Figura 2.2 mostra um *script* que executa o cálculo da área de um triângulo:

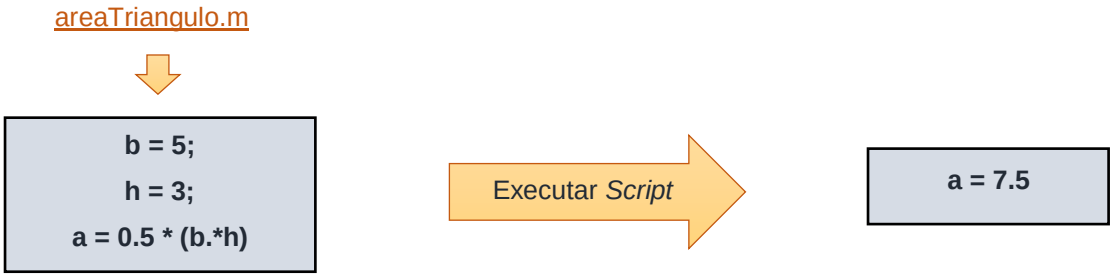


Figura 2.2: Esquema simplista da execução de *script* em *MATLAB*

O desenvolvimento de programas *MATLAB* promove um estilo de programação em que a organização de *scripts* e funções é relativamente não estruturada e não modular. No entanto, *scripts* em *MATLAB* são muito menos estruturados do que as funções e, portanto, apresentam mais desafios do que as próprias funções. Como os *scripts* são muito menos modulares do que as funções, uma das abordagens é fazer a transformação de *scripts* para funções de modo a mitigar aquilo que é a falta de modularidade - *Refactoring* [24].

Para converter um *script* para uma função é necessário determinar todos os *inputs* e *outputs* que devem funcionar para todas as chamadas do *script* assim como manter o comportamento do programa depois de todas as conversões serem realizadas [24].

2.1.4. *Toolbox*

Uma *toolbox* corresponde a um conjunto de ficheiros *MATLAB* (*m-files*, dados, exemplos, documentação e *apps*) contidos numa pasta separada de outras *toolboxes*. Depois da *toolbox* estar criada, *MATLAB* gera um único ficheiro de instalação com a extensão *.mltbx* que permite a sua instalação. Existe um conjunto de *toolboxes MATLAB* que são pastas e bibliotecas do sistema. O repositório estudado organiza os *m-files* em *toolboxes.* Nesta tese, são particularmente importantes as funções definidas nas *toolboxes* de sistema *MATLAB,* pois funcionam como um elemento uniformizador, possibilitando a identificação de *tokens standard* utilizados por todos os utilizadores de *MATLAB*.

2.1.5. Matrizes

Estrutura de dados que permite armazenar qualquer tipo de dados, sejam esses numéricos ou *strings*, ou até mesmo os dois tipos juntos [36]. Pode, contudo, ser declarado vazio e não possuir qualquer conteúdo. A Figura 2.3 mostra algumas formas de declatar matrizes:

```
% criação de matriz com tamanho 5
a = [1 2 3 4 5]
% matriz de 0's com tamanho 3 por 3
b = zeros(3,3)
% matriz de 1's com tamanho 4 por 2
c = ones(4,2)
```

Figura 2.3: Dec*lara*ção de matrizes em *MATLAB* [56] [66]

A partir da Figura 2.3 temos a perceção do conceito de alocar memória, nomeadamente, através das funções *zeros* e *ones* aqui preenchidas e que foram anteriormente inicializadas. Este fato é bastante relevante por nos transmitir a ideia de correspondência entre essas funções e o conceito de alocação de memória, que, mais uma vez, torna-se extremamente importante para este trabalho.

2.1.6. Variáveis

As variáveis em *MATLAB* são, por defeito, matrizes. Não é necessário definir o tipo da variável antes de atribuir um dado valor a essa variável. Para criar uma nova variável basta atribuir o nome da variável seguida do símbolo igual (=) e, por último, o valor que queremos dar a essa variável [37]. Na Figura 2.4 são dados alguns exemplos de criação de variáveis:

```
% atribuição de valor numérico
x = 6;
% atribuição de string
s = 'String'('String' é também um vetor)
% criação de uma matriz 3 por 3
M = [1 2 3; 4 5 6; 7 8 9]
% atribuição à variável F de uma função
F = myFunction()
```

Figura 2.4: Criação de variáveis em *MATLAB* [37]

As variáveis podem ser globais ou locais. Normalmente são locais, ou seja, podem apenas ser acedidas na função onde foram declaradas. Quando as variáveis são globais significa que são declaradas apenas uma vez e todas as funções que a chamam partilham uma cópia única da variável [46]. Para esta tese, as variáveis são apenas elementos estruturantes, que não ajudam na identificação de *concerns* no código. Na secção 2.3.2 podemos ver quais os elementos utilizados para realizar essa identificação.

2.1.7. Operadores

Operadores são símbolos que dizem ao compilador para executar uma certa operação matemática ou lógica. Embora o *MATLAB* esteja, sobretudo, designado para operar sobre matrizes e *arrays*, esses operadores conseguem operar também com valores escalares e não escalares. Estes operadores e operações elementares podem ser divididos em cinco grupos [57]:

- **Operações aritméticas** - existem dois tipos de operações: operações de *arrays* e de matrizes. No primeiro caso são executadas operações elemento a elemento enquanto no caso das matrizes são seguidas as regras da álgebra linear [35];

- **Operações relacionais** - realizam comparações elemento a elemento entre *arrays*. Os resultados fornecidos por esses operadores são um *array* lógico, mostrando onde essas comparações são verdadeiras [34];

- **Operações lógicas** - devolvem valores lógicos (0 caso seja falso e 1 caso seja verdadeiro) para mostrar se a condição a ser testada foi aceite ou rejeitada;

- **Operações conjuntas** - usadas para realizar junções, uniões e interseções entre dois *arrays*;

- **Operações *Bit-Wise*** - usadas para atualizar, deslocar ou comparar um valor específico num *array*.

Tal como as variáveis em *MATLAB*, os operadores são uma parte interessante da linguagem *MATLAB*, mas, como é visto na secção 2.3.2, não são utilizadas neste trabalho para identificar *concerns* em sistemas *MATLAB*.

2.1.8. Gráficos

A linguagem *MATLAB* tornou-se bastante popular pela possibilidade de se criar gráficos de uma forma rápida a partir de matrizes [21]. A Figura 2.5 apresenta um *script* para a criação de um gráfico em código *MATLAB*. Este exemplo recorre a *toolboxes* de sistema *MATLAB* que possui nomes de funções já dispersos pelas várias pastas da biblioteca do sistema. Esta apresentação é muito comum neste tipo de linguagem, destacando-se alguns nomes de funções que ocorrem frequentemente próximos para este tipo de visualização de dados. Atendendo ao código da Figura 2.5, verificamos a presença das funções *plot* e *title*, assim como de *xlabel* e *ylabel* [12] [60]. Sempre que uma destas funções é chamada, é muito provável que as restantes apareçam junto dela, embora existam outras, como *figure* e *legend*, que também se associam facilmente às restantes. A proximidade entre nomes de funções de *toolboxes* de sistema *MATLAB* ajudam a pensar em possíveis padrões sintáticos que, neste trabalho, poderão ser fundamentais para a exploração de novas relações entre funções.

```
yhat = x * betahat;
resid = y - yhat;
plot (x2, resid)
title ('Graph Title')
xlabel ('Time')
ylabel ('Residual')
print -dpdf 'test.pdf'
```

Figura 2.5: *Script* para criação de gráficos em *MATLAB* [12]

2.2. Métricas em engenharia de *software*

Sempre que um projeto de *software* está a ser desenvolvido, muitas das atividades inerentes ao mesmo levam à necessidade de quantificar aquilo que está sendo desenvolvido. Como tal, a utilização de métricas mostra-se particularmente apropriada (ou essencial) para uma abordagem prudente em engenharia de *software*. Uma boa métrica é aquela que permite a construção de indicadores que facilitam a tomada de decisão sem que a sua confiabilidade seja posta em causa [9].

A aplicação da engenharia de *software* descreve-se como uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, operação e manutenção de *software* [7]. Se cada uma das atividades a desenvolver for controlada à medida que o *software* é especificado, projetado, construído e mantido é menos provável que surjam problemas inesperados.

Muitos dos desenvolvedores de *software* fazem a medição das características do *software* para terem uma noção se os requisitos são consistentes e completos, se o *design* é de alta qualidade e se o código está pronto para ser testado. Assim, o uso de métricas indica potenciais efeitos de código mal estruturado e organizado dificultando a estabilidade do *design*. Código com uma estrutura deficiente pode, por exemplo, ter origem em alterações feitas num módulo que se propagaram para outras partes.

2.2.1. Métricas de *concern*

Para além das métricas que existem ao nível do *software*, recentemente, foram definidas métricas para quantificar propriedades dos *concerns*. Estas métricas referem-se a *concerns* (por oposição a métricas de modularidade) pelo facto de capturarem informação sobre *concerns* independentemente destes se encontrarem presentes em um ou mais módulos [11].

Assim, existe uma forte necessidade de se perceber os padrões comuns de *concerns* transversais, mecanismos para detetar e quantificar esses padrões de *concerns* transversais em projetos de *software* e estudos empíricos que relacionem os padrões específicos de *concerns* transversais com a estabilidade do *design* do projeto [11].

Qualquer sistema está sujeito a modificações ao longo do tempo o que implica que tenham que haver alterações nesse sentido. Essas alterações consomem tempo, por vezes são difíceis de se fazer e, além disso, induzem em erro. Para tal, alguns autores propõem o uso de métricas para quantificar a estabilidade do *design* do sistema. As métricas de *concerns* podem ser usadas para quantificar a dispersão de um *concern* sobre os módulos do sistema [11]. Em seguida, serão dados exemplos de métricas de *concern* propostos por alguns autores.

Sant' Anna et al. propõe a existência de três métricas de *concern* através da difusão de um *concern* sobre componentes (*CDC*), operações (*CDO*) e linhas de código (*CDLoc*) [27]. A difusão sobre componentes e operações quantificam o grau de dispersão de um *concern* em diferentes níveis de granularidade. *CDC* conta o número de classes e interfaces relacionadas a

um *concern* enquanto *CDO* conta o número de métodos e construtores para um dado *concern*. Pelo contrário, a dispersão de um *concern* sobre *LoC* (linhas de código) mede o grau do sintoma emaranhado de um *concern*. Como tal, conta o número de "*concern switches*" por cada *concern* através das linhas de código. Para ilustrar esta métrica de *concern*, a Figura 2.6 mostra o código fonte da classe *AbstractController*. Essa figura indica que há quatro "*concern switches*" entre a cadeia de responsabilidade e outros *concerns* na classe *AbstractController*. Portanto, nessa classe, o valor da métrica *CDLoc* para a cadeia de responsabilidade é quatro.

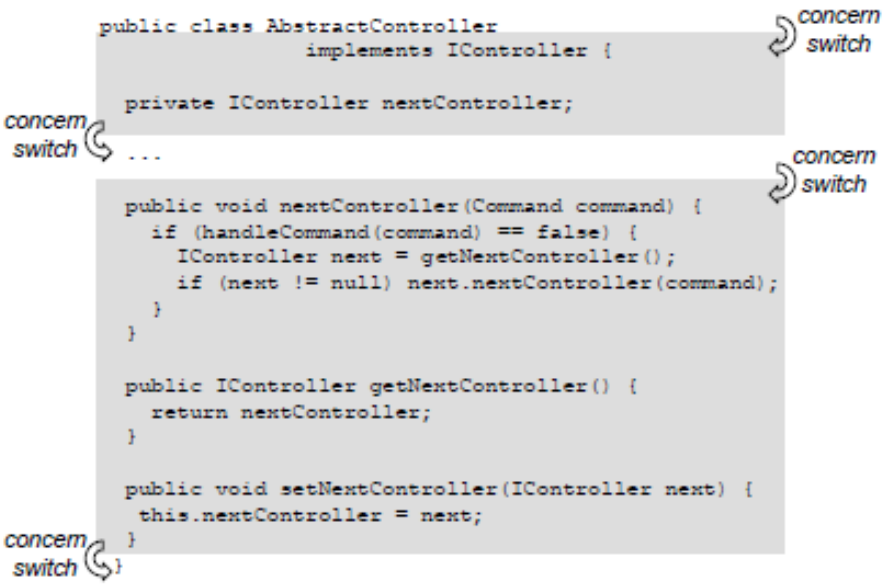


Figura 2.6: Projeção da cadeia de responsabilidade na classe *AbstractController* [11]

Ducasse, *Girba* e *Kuhn* elaboraram uma métrica chamada *Mapa Distribuído* para poderem analisar as propriedades de um sistema [10]. Com base na métrica que desenvolveram, enumeram quatro métricas de *concerns* que são exploradas a partir da ilustração de um exemplo (Figura 2.7) que se baseia na presença de caixas e grandes retângulos que representam, respetivamente, a estrutura do programa e as classes. Neste caso, vemos as propriedades como *concerns*.

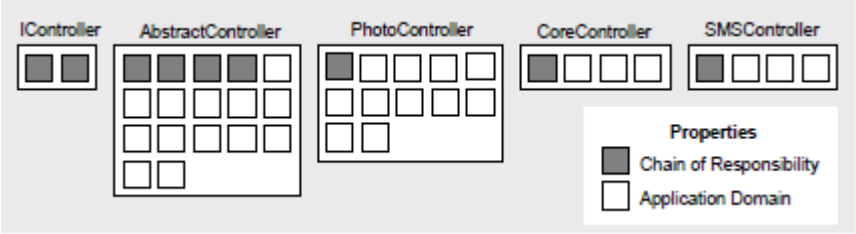


Figura 2.7: *Mapa distribuído* que mostra a propriedade "Cadeia de responsabilidade" [11]

Os retângulos mais pequenos que se encontram inseridos em cada um dos retângulos maiores são os métodos e atributos que estão relacionados com cada uma das propriedades. Neste caso, os quadrados preenchidos a cinzento dizem respeito a métodos e atributos que implementam a propriedade "Cadeia de Responsabilidade". As quatro métricas de *concerns* são: *size* (contagem do número de pequenos quadrados associados com a propriedade), *touch* (contagem do tamanho relativo dado em termos de percentagem de pequenos quadrados que realizam uma propriedade), *spread* (contagem do número de classes que contêm quadrados preenchidos) e *focus* (quantifica a proximidade entre uma classe específica e a propriedade).

Wong, *Gokhale* e *Horgam* introduziram três métricas de *concern*: disparidade, concentração e dedicação [33]. A disparidade mede a quantidade de blocos que se relacionam com uma *feature* num componente específico. Uma *feature* corresponde a uma funcionalidade exercida por um determinado *input* e um bloco é uma sequência de instruções consecutivas tal que, se um elemento é executado, então todos o são. A concentração e dedicação (também definidas em termos de blocos) quantificam quanto uma *feature* está concentrada num componente e quanto um componente é dedicado a uma *feature*. A disparidade tem em conta a concentração e dedicação, ou seja, quanto mais blocos num componente ou numa *feature* (não em ambos), maior a disparidade entre eles [11].

Com base nas três métricas de *concern* definidas anteriormente, *Eaddy*, *Aho* e *Murphy* apresentaram duas variantes para a concentração e dedicação [15]. Os autores propuseram usar o termo *concern* em vez de *feature* e linhas de código em vez de blocos. Assim, a concentração é definida como o quociente de *LoC* num componente destacando um *concern* pelo total de *LoC* no sistema. Da mesma forma, a métrica de dedicação é definida pelo quociente de *LoC* num componente definindo um *concern* pelo número de *LoC* do componente. Noutros trabalhos, *Eaddy et al*. apresentaram outra métrica - *LOCC* - que conta o número total de linhas de código que contribuem para a implementação de um *concern* [11].

2.3. Deteção de *concerns* em código *MATLAB*

Aspect mining é o processo de identificar e localizar *CCCs* como módulos latentes em código fonte, o qual já foi devidamente explorado no que respeita a linguagens OO [4] [13] [29]. A*spect mining* em sistemas *MATLAB* não se encontra tão desenvolvido e, como tal, esta secção mostra a deteção de *concerns* ao nível de *MATLAB* e outros aspetos importantes que ajudam a perceber melhor a identificação de *CCCs* [17] [19].

Em primeiro lugar, é feita uma análise à presença de *CCCs* em sistemas *MATLAB* para que possamos comparar e contextualizar com a presença desses mesmos *concerns* em sistemas mais gerais, como é o caso de linguagens OO. Em seguida, é desenvolvido o conceito

de *token* e uma abordagem baseada em *tokens* que justifica grande parte do trabalho a realizar. É feita uma referência ao repositório de código que serve de base para este projeto e, por último, são analisados os trabalhos realizados anteriormente neste contexto.

2.3.1. Exemplo de *concern* transversal em *MATLAB*

Em *MATLAB*, a presença de *CCCs* é notória quando uma determinada função inclui código que pode ser atribuído a mais do que um *concern*. Assim, esses *concerns* têm uma natureza secundária e encontram-se na função além do *concern* principal [19]. A presença de *CCCs* no código origina dois tipos de sintomas no código: emaranhado (*tangling*) e dispersão (*scattering*). O primeiro pode ser identificado quando cada função individual ou ficheiro *MATLAB* contém código que está relacionado com mais do que um *concern*. O sintoma de dispersão corresponde à representação de *concerns* não-modularizados em forma de pequenos trechos de código, espalhados por várias funções e/ou ficheiros *MATLAB* em vez de estarem localizados num único módulo. Ambos os sintomas dificultam a compreensão e manutenção do código e levantam obstáculos quanto à reutilização do mesmo [19].

Para ilustrar a presença de *CCCs* no código *MATLAB* é apresentado o exemplo da Figura 2.8 onde temos duas implementações diferentes de uma função (neste caso, "*Discrete Fourier Transform*") sendo que na parte superior se encontra uma versão "limpa" dessa função apenas com a presença de um *concern* (não existindo quaisquer *concerns* adicionais), enquanto na parte inferior da figura temos uma versão mais complexa que além de ter o *concern* principal tem outro *concern*, tornando o código "poluído". As funções presentes neste exemplo (*quantize* e *quantizer*) são fornecidas através da *toolbox* '*filter design*' e têm como objetivo forçar uma representação de baixo nível dos valores representados, como, por exemplo, *doubles* em representação de ponto fixo. Assim, o *concern* secundário é detetado através das ocorrências dessas funções ao longo das linhas de código, fazendo com que a segunda função seja um exemplo de código emaranhado. A presença dessas funções adicionais indicam que o *concern* presente na segunda figura diz respeito ao *concern* de especialização do tipo de dados ("*data type specialisation*" - a identificação deste *concern* pode ser vista em maior detalhe na secção 2.3.3) [5] [18].

```
function [y] = dft(x)
y = zeros(size(x));
N = length(x);
t = (0:N-1)/N;
for k = 1:N
      y (k) = sum (x.*exp(-j*2*pi*(k-1)*t));
end
```

```
function [y] = dft_specialized(x)
y = zeros(size(x));
N = length(x);
T = (0:N-1)/N;
quant1 = quantizer('fixed', 'floor', 'wrap', [18 16]);
t = quantize(quant1, t);
quant2 = quantizer('fixed', 'floor', 'wrap', [23 20]);
pi_fix = quantize(quant2, pi);
quant3 = quantizer('fixed', 'floor', 'wrap', [20 8]);
quant4 = quantizer('fixed', 'floor', 'wrap', [23 10]);
quant5 = quantizer('fixed', 'floor', 'wrap', [24 10]);
quant6 = quantizer('fixed', 'floor', 'wrap', [26 12]);
quant7 = quantizer('fixed', 'floor', 'wrap', [28 14]);
quant8 = quantizer('fixed', 'floor', 'wrap', [32 16]);
for k = 1:N
  v1 = quantize(quant3, (k-1)*t);
  v2 = quantize(quant4, pi_fix*v1);
  v3 = quantize(quant5, -j*2*v2);
  v4 = quantize(quant6, exp(v3));
  v5 = quantize(quant7, x.*v4);
  y(k) = quantize(quant8, sum(v5));
end
```

Figura 2.8: Exemplo de código emaranhado presente numa função [18]

É neste sentido que se torna importante desenvolver métodos para a deteção automática ou semi-automática de *CCCs* que possam no futuro propiciar atividades de reengenharia que melhorem a modularidade do sistema.

2.3.2. Tokens

Ao longo dos últimos anos, a comunidade científica portuguesa tem realizado algum trabalho relacionado com a modularidade em *MATLAB*, contribuindo com alguns resultados interessantes, nomeadamente através dos *papers* [18] e [19].

No contexto da linguagem *MATLAB*, *tokens* são os elementos lexicais extraídos a partir de um ficheiro de código com recurso a uma ferramenta de análise léxica. A Tabela 2.1 apresenta a categorização dos *tokens*:

Elementos lexicais	Tokens
keywords	break, case, catch, classdef, continue, else, elseif, end, for, function, global, if, otherwise, parfor, persistent, nargin, return, spmd, switch, try, while, entre outros
Operadores lógicos	&, &&, |, ||, ~, =, ==, ~=, <, <=, >, >=
Operadores aritméticos	+, -, *, /, """
Símbolos	(,), {, }, [,], :=, ;, ,, ., :, ^, \\, @

Tabela 2.1: Categorização dos *tokens*

Os nomes de funções, especialmente das bibliotecas *MATLAB*, a que damos o nome de *tokens*-palavra, compreendem os *tokens* mais utilizados porque normalmente são comuns a todos os sistemas *MATLAB*, fornecendo, assim, uma medida de garantia uniforme relativa a todos os sistemas. Os nomes de funções são, sem dúvida, o indicador mais confiável visto que o número de ocorrências de um dado nome de uma função ajuda a perceber a existência de sintomas no código.

Os *tokens* que correspondem a código que esteja comentado e todos os *tokens* que não sejam *keywords* como, por exemplo, símbolos e literais (incluindo *strings*) devem ser descartados. Os nomes de variáveis também são excluídos devido à sua especificidade nas funções ou no sistema, que dependem da aplicação e do programador em causa (não conseguiríamos garantir uma utilização uniforme desses nomes porque provavelmente significariam coisas diferentes). Além disso, os utilizadores de *MATLAB* costumam definir nomes de variáveis bastante curtos (por vezes, apenas com uma letra) fazendo com que estes nomes não nos deem qualquer informação.

Os *tokens* são agrupados em classes de equivalência (quando apresentam pelo menos uma característica em comum) sendo que cada classe faz a correspondência a um *concern*. Para que um *token* pertença a um *concern* é imprescindível que esse *token* seja informativo. Se pensarmos em palavras reservadas como '*if*' ou '*while*' que ocorrem inúmeras vezes ao longo de um ficheiro, notamos que elas não nos conseguem dar qualquer tipo de informação. Como tal, nem todos os *tokens* podem ser associados a *concerns* [17] [22].

2.3.3. Abordagem baseada em *tokens* para deteção de *concerns*

Com base no conceito de *token*, desenvolveu-se uma abordagem que se baseia na análise de métricas de *concern* (ver secção 2.2.1) facilitando a captura de informações sobre os *concerns* em código *MATLAB*. Estas métricas de *concern* tornam-se interessantes quando se pretende realizar tarefas para identificar e localizar *concerns*. A principal unidade de informação são, precisamente, os *tokens*. Neste sentido, cada *concern* é associado a um grupo de *tokens* e, além disso, os padrões de ocorrência de tais *tokens* são usados para detetar a presença do *concern* correspondente. Um *token* só pode estar associado a um *concern*, no máximo [17].

O mapeamento entre *concerns* e nomes de funções (*tokens*-palavra) deu origem à Tabela 2.2, que foi originalmente desenvolvida por um perito em *MATLAB*. As métricas usadas são as seguintes: *LoC* (linhas de código), que conta o número de linhas de código que não são nem comentários nem linhas em branco; e densidade de *tokens*, que faz o cálculo, por cada ficheiro *MATLAB*, do número de ocorrências de um determinado conjunto de *tokens*-palavra a dividir pelo número de *LoC* (esta métrica começou a ser explorada em [22]). Esta última métrica representa o rácio de *tokens*-palavra por *LoC* para um determinado *concern*. O mapeamento da Tabela 2.2 deu origem a onze instanciações da métrica de densidade de *tokens*.

A Tabela 2.2 mostra a correspondência entre *tokens* e *concerns*. Como foi visto na secção 2.1, existe uma relação de proximidade entre várias funções em *MATLAB* que justificam a presença dessas funções (*tokens*-palavra) para a mesma entrada da tabela. Se pensarmos no exemplo visto na secção 2.1.8, onde *plot* e *title* aparecem frequentemente próximos um do outro, a Tabela 2.2 mostra que eles estão relacionados com o mesmo *concern* (*visualisation*). A palavra reservada *if*, apesar de não pertencer a nenhuma entrada da tabela (por não ser informativa), possui um nível de relevância para alguns *concerns* por aparecer muitas vezes juntamente com eles, destacando-se os *concerns verification of function arguments and return values*, *data type specialisation* e *data type verification*. Estas relações de proximidade entre funções e palavras reservadas *MATLAB* mostram-se muito motivadoras por poderem ajudar na procura de novos *tokens* a partir das relações já conhecidas, com recurso a padrões sintáticos.

Ao longo do tempo, a tabela de *tokens* tem sido sujeita a várias modificações superficiais no sentido de se perceber qual aquela que melhor carateriza a relação entre *concerns* e *tokens*. Já houve subdivisão de um *concern*, alterações na localização de cada *token* e criação de novos *concerns* de forma a evitar que existam demasiados *tokens* por *concern* [22].

Todas estas observações bem como a Tabela 2.2, que relaciona um grupo de *tokens* a um *concern* específico, têm por base o trabalho que levou à tese de Bruno Palma [22] e à publicação dos *papers* [17] e [19].

Concern	Tokens
Verification of function arguments and return values	nargchk, nargin, nargout, nargoutchk, varargin, varargout
Data type specialisation	fi, fimath, int16, int32, int64, int8, quantize, quantizer, sfi, single, ufi, uint16, uint32, uint64, uint8, double
Data type verification	cast, class, intmax, intmin, isa, isboolean, iscell, ischar, iscolumn, isempty, isfi, isfield, isfimath, isfixed, isfloat, isinf, isinfinite, isinteger, islogical, isnan, isnumeric, isobject, isquantizer, isreal, isrow, isscalar, isstr, isstruct, isvector, length, ndims, numel, range, realmax, realmin, size, typecast, wordlength
Dynamic properties	eval, evalc, evalin, inline, feval
Console messages	annotation, assert, disp, display, error, last, lastwarn
Printing	orient, print, printdlg, printopt
Visualisation	aaxes, axis, box, cla, clabel, clf, close, datacursormode, datetick, errorbar, figure, figurepalette, fplot, gca, gcbf, gcbo, gco, getframe, gplot, grid, gtext, hist, histogram, hold, imfinfo, ishold, legend, line, loglog, mesh, meshgrid, newplot, pan, plot, plot3, plotbrowser, plotedit, plottools, plotyy, polar, propertyeditor, rectangle, reset, rgbplot, scatter, semilogx, semilogy, showplottool, subplot, surf, texlabel, text, title, xlabel, ylabel, zlabel, zoom, set, rotate, rotate3d, imformats, imread, imwrite, movie, image, frame2im, im2frame, VideoReader, VideoWriter
File I/O	diary, fgetl, fgets, fileformats, fopen, fprintf, fread, fscanf, fwrite, hgload, hgsave, load, save, saveas, uisave
System	ans, echo, exist, inmem, input, inputname, inputParser, isglobal, iskeyword, isvarname, mexext, mfilename, namelengthmax, pcode, symvar, who, whos, systems, slist, where, loadlibrary, mex, calllib, libisloaded, unloadlibrary, libfunctionsview, onCleanup, clearvars, rehash, pack, memory, clear, addtodate, now, weekday, date, calendar, dbcont, dbquit, dbstop, dbmex, ba, bafter, break, ebreak, nanbreak, rbreak, tbreak, xbreak, zcbreak, wait, stop, run, batch, spmd, pause, step, next, mislocked, mlock, munlock, clock, cputime, etime, start, startat, tic, timerfind, timerfindall, toc, profile, profsave
Memory allocation/deallocation	delete, global, persistent, zeros, ones
Parallelisation	cancel, codistributed, codistributor, createParallelJob, createTask, defaultParallelConfig, demote, destroy, detupForParallelExecution, dfeval, dfevalasync, distributed, gather, gcat, gop, gplus, gpuArray, gpuDevice, gpuDeviceCount, importParallelConfig, isreplicated, jobStartup, labBarrier, labBroadcast, labindex, labProbe, matlabpool, mpiLibConf, mpiprofile, mpiSettings, numlabs, parfor, pctconfig, pctRunDeployedCleanup, pctRunOnAll, pload, pmode, poolStartup, promote, psave, redistribute, resume, sparse, submit, subsasgn, subsref, taskFinish, taskStartup

Tabela 2.2: Relação entre *concerns* e *tokens* aquando do início desta tese [17] [19]

Verifica-se que a métrica de densidade de *tokens* parametriza o *concern*, ou seja, o conjunto específico de *tokens* considerados. O mapeamento da Tabela 2.2 deu origem a onze instanciações da densidade de *tokens*.

A Figura 2.9 mostra o conteúdo de um *m-file* (exceto as linhas em branco e os comentários), cujas métricas se destacam por alguns *concerns*. Se assumirmos que o número de linhas de código é 9 (para o exemplo de código da Figura 2.9), a métrica para o *concern verification of function arguments and return values* é de 0.2/*LoC*, o *concern data type specialisation* é de 1.0/*LoC* e o *concern memory allocation/deallocation* é de 0.5/*LoC*.

```
function fired = nemoStep(fstim, istim_nidx, istim_current)
    if nargin < 1
        fired = nemo_mex(uint32(12), uint32(zeros(1, 0)), uint32(zeros(1, 0)), zeros(1, 0));
    elseif nargin < 2
        fired = nemo_mex(uint32(12), uint32(fstim), uint32(zeros(1, 0)), zeros(1, 0));
    else
        fired = nemo_mex(uint32(12), uint32(fstim), uint32(istim_nidx), istim_current);
    end
end
```

Figura 2.9: Ficheiro *MATLAB* com a presença de 3 *concerns* para o cálculo da métrica de densidade de *tokens* [17]

Legenda:

- *Concern verification of function arguments and return values*
- *Concern data type specialisation*
- *Concern memory allocation/deallocation*

A densidade de *tokens* é suficiente para permitir a deteção automática de *concerns*. No entanto, esta métrica não é adequada para fornecer uma visão ampla de um grande repositório e fornecer um panorama de todos os *concerns* da Figura 2.9 e das suas várias coocorrências [17].

Como exemplo ilustrativo, apresenta-se o *concern verification of function arguments and return values* para fornecer uma visão mais detalhada acerca dos seus *tokens* associados. Os *tokens* que correspondem a este *concern* pertencem a funções especiais, suportadas pelo *MATLAB*, que são usadas para validar argumentos de *input* e *output*. Todas as funções deste *concern* estão listadas na Tabela 2.3, com uma breve descrição para cada uma delas. As definições de cada função ajudam a perceber eventuais relações que existam entre cada par de funções, algo que se torna bastante interessante para este trabalho.

Token	Descrição
nargchk	nargchk (minArgs, maxArgs) valida o número de argumentos de *input* na chamada para a função que está atualmente em execução. nargchk lança um erro se o número de *inputs* especificados na chamada for menor que minArgs ou maior que maxArgs. Se o número de *inputs* estiver entre minArgs e maxArgs (inclusive), então nargchk não fará nada [52]
nargin	Devolve o número de argumentos de *input* fornecidos na chamada para a função que está atualmente em execução [53]
nargout	Devolve o número de argumentos de *output* fornecidos na chamada para a função que está atualmente em execução [54]
nargoutchk	nargoutchk (minArgs, maxArgs) valida o número de argumentos de *output* na chamada para a função que está atualmente em execução. nargoutchk lança um erro se o número de *outputs* especificados na chamada for menor que minArgs ou maior que maxArgs. Se o número de *outputs* estiver entre minArgs e maxArgs (inclusive), então nargoutchk não fará nada [55]
varargin	Variável de *input* numa definição de função que permite que a função aceite qualquer número de argumentos de *input* [64]
varargout	Variável de *output* numa definição de função que permite que a função aceite qualquer número de argumentos de *output* [65]

Tabela 2.3: Funções do *concern verification of function arguments and return values*

2.3.4. Repositório *MATLAB* usado neste trabalho

Para realizar a validação da abordagem proposta é necessário a utilização de um repositório de código *MATLAB* bastante grande com a presença dos mais variados *tokens* e, consequentemente, de *concerns*. A comparação e utilização de vários *m-files* ajudará a validar e refinar a tabela de *tokens* porque cada ficheiro *MATLAB* apresenta condições específicas de código.

O repositório a ser tratado para a validação da tabela foi utilizado inicialmente por Bispo e Cardoso para testar o compilador *MATLAB* [2] embora tenha sido posteriormente utilizado noutros trabalhos [17] [22], compreendendo 35.193 *m-files* organizados por *toolboxes* e cobrindo vários domínios de aplicação. Após algumas observações, o repositório ficou composto por 34.409 *m-files*, descartando-se alguns devido a certos *m-files* terem apenas linhas de comentário. Alguns *m-files* foram removidos por possuírem menos de cinco linhas de código não se apresentando como um caso interessante para aplicar a métrica da densidade de *tokens* (resultando em pouco mais de 30.000 *m-files*). De acordo com os dados recolhidos, apenas uma pequena percentagem dos *m-files* do repositório não contém nenhum dos *concerns* da Tabela 2.2 [17].

O trabalho desenvolvido por António Relvas [25] facilita a forma como são realizadas algumas pesquisas (validações) através de um repositório implementado com toda a estrutura do código *MATLAB* também usado neste trabalho.

2.3.5. Trabalhos anteriores

O desenvolvimento desta tese tem por base o trabalho realizado anteriormente [17] [18] [22]. No contexto do trabalho para a sua tese de mestrado [22], Bruno Palma utilizou a métrica dos Mapas Auto-Organizados (*Self Organization Map - SOM*) que permitiu visualizar *concerns* de forma eficiente (baseado em métricas) e detetar sintomas de *CCCs* em grandes repositórios de código, verificando-se a presença de código emaranhado. Produziram-se ainda mais duas métricas: distância léxica e contagem de pares (ver secção 3.1.2). Ambas as métricas são também baseadas na abordagem falada anteriormente sendo que não foram utilizadas durante [22] porque não foram úteis nem completamente exploradas. Concluiu-se que a métrica da densidade de *tokens* (aplicada sobre a abordagem proposta na secção 2.3.3) é suficiente para permitir a deteção automática de *concerns*. Contudo, se aplicada em larga escala, o uso dessa métrica por si só não é adequada para fornecer uma visão global de um repositório grande e, mais ainda, de todos os seus *tokens* e das suas várias coocorrências.

Na sequência do paper [18] surgiram novas oportunidades de trabalho no que se refere à abordagem baseada em *tokens* e à tabela de *concerns*/*tokens*. Na tese de mestrado de Bruno Palma [22] realizaram-se algumas mudanças progressivas relativamente à tabela de *tokens*

embora existam alguns detalhes ao nível do *token* que poderão ser revistos: verificar quais aqueles que ocorrem vezes suficientes a ponto de se manterem na tabela; aqueles que ocorrem frequentemente, mas não estão na tabela; identificar *tokens* que são transversais a vários *concerns* e verificar quais os *tokens* da tabela que nunca ocorrem. De um modo geral, as várias versões da tabela de *tokens* mencionadas em [22] não estão devidamente validadas, o que pode dificultar a forma como se localizam os *concerns* a partir dos *tokens*.

2.4. Análise das métricas a utilizar

Neste capítulo são estudadas algumas das abordagens/métricas que são consideradas e adaptadas no contexto desta tese.

2.4.1. Informação Mútua Pontual - Associação de palavras

A teoria da informação estuda a quantificação, armazenamento e comunicação da informação. Em 1949, o matemático *Claude Shannon et al*. publicou o livro "The *Mathematical Theory of Communication"* [28] onde apresenta um modelo linear de comunicação que visa a precisão e a eficácia do fluxo informativo. O autor procura estudar a comunicação em qualquer âm*bit*o e não se cingir apenas às áreas de engenharia.

A medida chave em teoria da informação é a *entropia*. A *entropia* quantifica a quantidade de incerteza envolvida no valor de uma variável aleatória. Dado o exemplo de uma moeda "justa" com probabilidades iguais de sair cara ou coroa, esta fornece menos informação (menos *entropia*) do que calcular a probabilidade de sair uma face de um dado (com 6 faces) [31].

A unidade utilizada para quantificar informação é o *bit*:

$$n = \log_2 m \, [bits]$$

Equação 2.1: Quantificação da informação através de *bits* [31]

n bits é a informação necessária para se escolher entre *m* alternativas equiprováveis, ou 1 *bit* é a quantidade de informação necessária para escolher entre duas alternativas também equiprováveis. Geralmente, a *entropia* da informação é a quantidade média de informação transmitida por um evento, ao considerar todos os resultados possíveis.

Uma das medidas importantes em teoria da informação é a informação mútua pontual (IMP) [6] que fornece características interessantes para estudar a associação de palavras (no nosso caso, *tokens*). Com a utilização da IMP pretende-se perceber a relação que existe entre cada par de *tokens* da Tabela 2.2 da página 20 e, além disso, a relação entre os *concerns* e *tokens*. A IMP é uma medida de informação comum entre as variáveis aleatórias [3].

O termo "associação de palavras" é usado muito particularmente na área da linguística computacional. A ideia é perceber a relação que existe entre cada duas palavras, não pelo significado de cada uma delas, mas sim pela ocorrência com que as palavras se apresentam juntas (num dado repositório de palavras). Se pensarmos na palavra "*nurse"*, a maior parte das pessoas responde mais rapidamente do que o normal a essa palavra se "*nurse"* estiver a seguir à palavra "*doctor"*. Sendo assim, e tendo em conta o paper de *Church et al.* [6], será feita uma descrição estatística com base num repositório grande de palavras de forma a percebermos possíveis relações que possam existir entre cada duas palavras.

É neste sentido que *Church et al.* utiliza a IMP para calcular o valor entre cada duas palavras consoante as probabilidades de elas aparecerem individualmente e em conjunto num dado repositório de palavras não estruturadas.

Informalmente, a IMP compara a probabilidade de observar x e y juntos no repositório (probabilidade conjunta) com a multiplicação das probabilidades individuais de cada uma das palavras, x e y. Se existe uma relação muito grande entre duas palavras, então a probabilidade conjunta será muito maior que a multiplicação das suas probabilidades independentes e, consequentemente, a IMP entre elas será muito superior a zero. Pelo contrário, se x e y têm uma distribuição complementar, então P(x,y) será muito menor que P(x) * P(y) fazendo com que a IMP seja altamente negativa. Por último, se P(x,y) apresenta valores semelhantes à multiplicação de P(x) com P(y), então conclui-se que a IMP será perto de zero, não existindo uma relação interessante entre as duas palavras.

Assim, se duas palavras, x e y, têm probabilidade individuais P(x) e P(y), respetivamente, e probabilidade conjunta P(x,y), então a sua IMP, I(x,y), é definida por:

$$I(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

Equação 2.2: IMP entre duas palavras [6]

A Tabela 2.4 calcula a IMP entre cada duas palavras que se seguem consecutivamente. Os dados utilizados têm por base um repositório de cerca de quinze milhões de palavras, que é utilizado em [6].

Por definição, f(x) e f(y) correspondem ao número de ocorrências da palavra x e y no *corpus* do texto, respetivamente, enquanto f(x,y) e f(y,x) corresponde ao número de vezes que x é seguido por y ou vice-versa.

x	f(x)	y	f(y)	f(x,y)	I(x,y)
honorary	111	**doctor**	621	12	**11.3**
doctors	1105	**dentists**	44	8	**11.3**
doctors	1105	**nurses**	241	30	**10.7**
doctors	1105	**treating**	154	8	**9.4**
examined	275	**doctor**	621	6	**9.0**
doctors	1105	**treat**	317	11	**8.9**
doctor	621	**bills**	1407	25	**8.7**
doctor	621	**visits**	350	6	**8.7**
doctors	1105	**hospitals**	676	19	**8.6**
nurses	241	**doctors**	1105	6	**8.4**

Tabela 2.4: Cálculo da IMP entre duas palavras [6]

A Tabela 2.4 apresenta a negrito as duas palavras que estão a ser analisadas pela IMP e, também, o valor da IMP entre elas (na última coluna). Através da tabela apresentada podemos notar que todas as palavras apresentam uma relação bastante interessante com a palavra "*doctor"* visto que as informações mútuas pontuais são claramente positivas. O valor apresentado da IMP sobre cada um dos pares tem por base as contagens da Tabela 2.4 que não estão a negrito e que são fornecidas sobre a equação da IMP.

A métrica da IMP pode claramente ser transposta para uma vertente que tem como base a abordagem em *tokens*, onde cada palavra será vista como um *token*. A ordem pela qual os *tokens* se encontram não é indiferente para a IMP porque dois *tokens* em duas posições distintas podem fornecer informações diferentes e fundamentais no estudo das suas relações. As relações existentes entre cada par de *tokens* e entre cada *concern* e *tokens* poderão fornecer bons indicadores que ajudarão a justificar possíveis alterações que possam ser feitas à tabela de *tokens* (Tabela 2.2 da página 20).

2.4.2. Outras métricas de associação de palavras

Uma primeira abordagem descrita sobre a IMP ajuda a perceber possíveis relações entre *tokens* e *concerns* sobre a tabela de *tokens*. A análise feita propõe-se ser interessante por mostrar de uma forma direta quais os valores da IMP entre cada par de *tokens*, ainda que os valores apresentados por essa métrica sejam algo subjetivos por não existir um valor máximo, ou "teto", para os valores. Embora a IMP seja uma métrica bastante confiável para prever a coocorrência de *tokens* num repositório, foram estudadas e analisadas outras métricas que serviram de base comparativa com a IMP. Essas métricas apresentam apenas valores entre 0 e 1 e prometem, por isso, ser mais escaláveis comparativamente com a IMP. De seguida, são mostradas as restantes métricas utilizadas [30]:

- A função ***cosseno***, tal como a IMP, recebe duas palavras (ou *tokens*) e executa a divisão entre a frequência absoluta de x e y no repositório e a raíz da multiplicação entre as frequências de x e y:

$$\cos(x,y) = \frac{f(x,y)}{\sqrt{f(x) * f(y)}}$$

Equação 2.3: Função *cosseno* entre duas palavras

- A função ***dice*** calcula a relação entre duas palavras/*tokens* (x e y) a partir da divisão da frequência absoluta de x e y no repositório multiplicado por dois sobre a soma das frequências individuais de cada uma das palavras, x e y:

$$dice\ (x,y) = \frac{2 * f(x,y)}{f(x) + f(y)}$$

Equação 2.4: Função *dice* entre duas palavras

- A função ***SCP*** (*Symmetrical Conditional Probability*) corresponde a $\cos^2$ e é uma medida de probabilidade condicional simétrica que testa a coocorrência entre as palavras/*tokens* x e y (ao longo de um repositório) tomando as probabilidades condicionais de cada um elevado ao quadrado sobre a multiplicação de ambos os termos:

$$scp\ (x,y) = \frac{f(x,y)^2}{f(x) * f(y)}$$

Equação 2.5: Função *SCP* entre duas palavras

2.4.3. *Word2vec*

Word2vec é uma rede neural desenvolvida por *Mikolov et al.* [16] que calcula representações de palavras como vetores tendo vindo a ganhar bastante popularidade na linguística computacional. As representações de palavras distribuídas num espaço vetorial ajudam os algoritmos de aprendizagem a alcançar um melhor desempenho em tarefas de processamento de linguagem natural, agrupando as palavras semelhantes. *Mikolov* introduziu o modelo *skip-gram* que é um método eficiente para aprender representações vetoriais de alta qualidade de palavras a partir de grandes quantidades de dados de texto não estruturado. Ao contrário da maior parte das arquiteturas de rede neural usadas anteriormente para aprender vetores de palavras, o treino no modelo *skip-gram* (Figura 2.10) não envolve multiplicações de matrizes densas. Isso torna o treino extremamente eficiente em comparação com abordagens anteriores [26]: a nova implementação otimizada permite a uma única máquina poder treinar mais de 100 bilhões de palavras num dia [16]. No entanto, esta abordagem precisa de imensos dados para que se possa mostrar fiável e poder ter um uso devidamente aceitável, algo que é limitado pelo número insuficiente de dados a que temos acesso no repositório *MATLAB*.

As representações de palavras em *word2vec* são muito interessantes porque os vetores codificam explicitamente muitas regularidades e padrões linguísticos. Surpreendentemente, muitos desses padrões podem ser representados como traduções lineares. Por exemplo, o resultado de um vetor *vec* ("Madrid") -> *vec* ("Espanha") + *vec* ("França") está mais próximo de *vec* ("Paris") do que qualquer outro vetor de palavras [16].

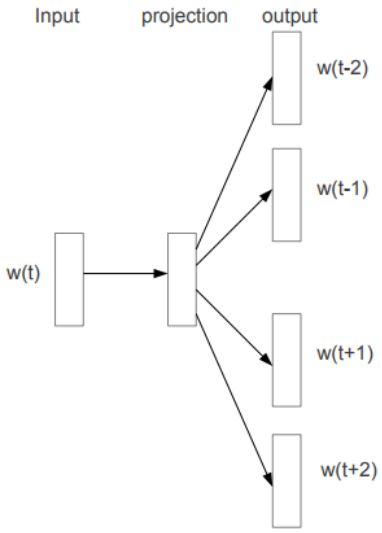


Figura 2.10: Arquitetura do modelo *skip-gram* [16]

2.4.4. *LARA*

Até ao momento, a utilização de ferramentas para extração de métricas a partir de *m-files* tem sido limitada. *CCCExplorer* é uma dessas ferramentas (ver secção 3.1) que foi desenvolvida por alguns autores do artigo [19] com base nalguns conceitos desse *paper*. No entanto, esta ferramenta apresenta alguma especificidade (suportando apenas a linguagem *MATLAB*) e é nesse sentido que, em seguida, se apresenta uma outra ferramenta que obtém resultados similares de uma forma mais eficaz.

LARA é uma linguagem específica de domínio (*Domain-Specific Language* - *DSL*), inspirada nos conceitos da programação orientada a aspetos (*Aspect-Oriented Programming* - *AOP* [14]), para estratégias de programação muitas vezes relacionadas com *concerns* não-modularizados através de transformações, monitoramento e instrumentação de código. *LARA* explora a ideia de que é possível ter uma única linguagem de programação, independentemente da linguagem de destino, capaz de selecionar pontos de interesse e expressar transformações no código-fonte. Ao contrário de muitas abordagens de *AOP*, a abordagem baseada em *LARA* foi projetada para ser, tanto quanto possível, indiferente à linguagem de destino e para especificar inserções e transformações de código para qualquer código de destino suportado. Ao usar o código *LARA* para transformar o código-fonte de *input* numa linguagem de destino específica coneta-se o modelo da linguagem e a representação do código de destino, por exemplo, numa *Abstract Syntax Tree* (*AST* - é uma representação em árvore da estrutura sintática abstrata do código-fonte de uma linguagem de programação [67]) [23].

Adicionalmente àquilo que o *CCCExplorer* permite realizar com a extração de métricas, *LARA* vai para além dessa funcionalidade permitindo fazer uma melhor exploração sobre o código e, sobretudo, explorar diversas linguagens que vão para além do *MATLAB*. Contudo, esta ferramenta não foi usada neste trabalho porque o *CCCExplorer* fornecia uma melhor documentação quando comparado com o *LARA* e, além disso, a única linguagem analisada foi *MATLAB* (a qual o *CCCExplorer* explora suficientemente bem para o trabalho a ser realizado).

A linguagem *LARA* permite a exploração de código *MATLAB* na representação em AST do código e, consequentemente, permite o desenvolvimento de padrões para a deteção de *concerns*. A partir de certos padrões compostos por *tokens* (e não só), é possível, através do *LARA*, identificar *concerns* e, se possível, encontrar novos *tokens*.

3. Ferramenta para extração e processamento de métricas

Este capítulo esclarece o que esta ferramenta permite fazer e algumas alterações que foram feitas nela tendo em conta o contexto desta tese. De forma a se poder extrair métricas e analisá-las foi utilizada e integrada neste projeto uma aplicação baseada em *java*: *CCCExplorer*.

3.1. *CCCExplorer*

CCCExplorer foi inicialmente desenvolvida com base no trabalho apresentado no CoRTA [19], por alguns autores desse artigo. Esse *software* aceita *m-files* como ficheiros de *input* (Figura 3.1), processa-os, decompondo o seu conteúdo em *tokens* (Figura 3.2) e executa o cálculo de várias métricas.

A Figura 3.2 mostra o *output* gerado pela ferramenta *CCCExplorer* para o exemplo de código recebido na Figura 3.1. A Figura 3.2 apresenta todos os *tokens* presentes no ficheiro de código *MATLAB* da Figura 3.1, além do nome do ficheiro, da linha associada e, sobretudo, da associação que existe entre cada *token* e a sua *tag*. A Tabela 3.1 mostra o significado de cada uma dessas *tags*:

Tag	Significado
ARG (Argument)	Palavra usada como parâmetro formal do cabeçalho de uma função
FHD (Function Header)	Palavra usada como nome de uma função, no cabeçalho da definição de uma nova função
FUN (Function)	Todas as palavras encontradas que não foram classificadas com nenhuma outra etiqueta. São, assim, assumidas como nomes de funções
KEY (Keyword)	Palavra reservada de *MATLAB*
NWD (Non-Keyword)	*Token* que não é uma palavra
VAR (Variable)	Palavra usada no lado esquerdo de uma atribuição e, portanto, é identificada como sendo uma variável

Tabela 3.1: Significado de cada *tag* associada aos *tokens*

```
N=length(seq);
cf=zeros(N,N);
disp('Forward counts...');
for i=1:N   %for each position p in the sequence
    %disp(['position ' num2str(i)]);
    for j=1:i
          cf(i,j)=length(strfind(seq,seq( (i-j+1):i) ));
    end
end
seq=seq(end:-1:1);
cb=zeros(N,N);
disp('Backward counts...');
for i=1:N   %for each position p in the sequence
    %disp(['position ' num2str(i)]);
    for j=1:i
          cb(i,j)=length(strfind(seq,seq( (i-j+1):i) ));
    end
end
cb=cb(end:-1:1,:);
```

Figura 3.1: Código *MATLAB* para análise de *tokens*

```
MATLAB-Bioinformatics count__repeat.m 18
  VAR:N FUN:function ARG:seq ARG:cf FUN:zeros VAR:N VAR:N FUN:disp
  KEY:for VAR:i VAR:N KEY:for VAR:j VAR:i ARG:cf VAR:i VAR:j
  FUN:length FUN:strfind ARG:seq ARG:seq VAR:i VAR:j VAR:i KEY:end
MATLAB-Bioinformatics count__repeat.m 18
  KEY:end
MATLAB-Bioinformatics count__repeat.m 18
  ARG:seq ARG:seq KEY:end
MATLAB-Bioinformatics count__repeat.m 18
  ARG:cb FUN:zeros VAR:N VAR:N FUN:disp KEY:for BAR:i VAR:N KEY:for
  VAR:j VAR:i ARG:cb VAR:i VAR:j FUN:length FUN:strfind ARG:seq ARG:seq
  VAR:i VAR:j VAR:i KEY:end
MATLAB-Bioinformatics count__repeat.m 18
  KEY:end
MATLAB-Bioinformatics count__repeat.m 18
  ARG:cb ARG:cb KEY:end
```

Figura 3.2: Ficheiro *.tk* (Token) com as tags e respetivos *tokens*

Nesse contexto, o termo *token* é usado para se referir a símbolos (por exemplo, '=', '+'), a nomes de funções (por exemplo, *nargin*, *error*) ou a palavras reservadas do *MATLAB* (por exemplo, *while*, *for*). Todas essas métricas são direta ou indiretamente baseadas em *tokens*, para dar suporte à abordagem descrita na secção 2.3.3. Algumas das métricas são baseadas diretamente em *tokens*, uma vez que são meras contagens de *tokens*. As métricas que são indiretamente baseadas em *tokens* usam um raciocínio mais elaborado, como a métrica da densidade de *tokens*, que, no fundo, é uma contagem de *token* normalizada pelas suas linhas de código. Essas métricas são calculadas em vários contextos: para *m-files* individuais, para todos os *m-files* numa *toolbox* e todos os *m-files* em coleções de *toolboxes*. Os resultados são normalmente apresentados em formato *.csv* e armazenados em estruturas complexas [22]. Para este projeto, usa-se esta ferramenta para extrair dados de métricas de um repositório de código *MATLAB*.

3.1.1. Estrutura dos pacotes

CCCExplorer é estruturado numa hierarquia de pacotes e subpacotes onde os seus componentes são armazenados. No primeiro nível da estrutura de pacotes deste sistema temos as três principais raízes que levam a outros subpacotes:

1. ***src*** - raíz de origem do sistema;
2. ***test*** - contém uma bateria de casos de teste que testa as diferentes funcionalidades do sistema;
3. ***main*** - contém exemplos de classes com métodos principais que são implementados para extrair dados métricos e imprimi-los num ficheiro *csv*.

A raíz *src* é onde as principais classes do sistema estão localizadas. É composto pelos seguintes subpacotes:

- ***cccfinder*** - contém as métricas a serem processadas e as funcionalidades necessárias para processar o código *MATLAB*;
  - ***metrics*** - contém as métricas a serem processadas noutras classes;
  - ***tool*** - contém todas as funcionalidades essenciais para analisar *m-files* e extrair dados métricos.

- ***token_classifier*** - contém os componentes essenciais usados para classificação de *token*;

- ***lexical_analyzer*** - contém o componente do analisador léxico;

- ***token*_table** - contém o componente da tabela de *tokens*;

- ***util*** - contém outros utilitários (como, por exemplo, barra de exibição de progresso e gerador csv).

3.1.2. Métricas suportadas

A presente tese baseia-se num conjunto de métricas, algumas das quais exploradas por Bruno Palma [22] e que são aplicadas a ficheiros *MATLAB* para uma posterior análise dos seus resultados. Em seguida, é mostrada uma lista com várias dessas métricas para análise de dados extraídos:

- **Contagem de *token*** *(Token Count)* - número total de ocorrências de um dado *token* num *m-file* ou numa *m-function*;

- **Contagem de pares** (*Pair Count*) - número total de ocorrências para um par de quaisquer *tokens* (por exemplo, número de instâncias em que um *token* '*if*' é seguido por um *token* '*nargin*') num *m-file* ou numa *m-function*. Uma ocorrência é detetada se o segundo *token* do par é localizado imediatamente depois do primeiro *token* do par ser detetado;

- **Linhas de código** *(Lines of Code - LoC)* - por cada *m-file*, é a contagem de todas as linhas de código que não sejam exclusivamente de comentário nem em branco;

- **Contagem de *m-files*** (*M-file Count*) - número total de *m-files* no repositório fornecido ao sistema para análise;

- **Contagem de ficheiros com *token* específico** *(Files With Token Count - FWTC)* - número total de *m-files* em que foi detetada pelo menos uma ocorrência do *token*;

- **Soma da contagem de *token* específico** *(Sum Token Count - STC)* - somatório de todas as contagens para um dado *token*;

- **Contagem máxima de *tokens*** *(Maximum Token Count - MTC)* - número máximo das contagens de todos os *tokens*, em qualquer *m-file* do repositório;

- **Distância léxica** (*Lexical Distance*) - número de ocorrências de pares de *tokens* X e Y com uma distância de N ou menos *tokens* entre eles. Esta métrica pode ser calculada tanto para *m-files* como para *m-functions*. Esta métrica foi implementada em [22];

- **Densidade de *tokens*** (*Token Density*) - para um dado *m-file* ou *m-function*, é a contagem total das instâncias de determinado conjunto de *tokens* (entrada da tabela de *concerns*/*tokens*), contando cada ocorrência como um, normalizada em função de *LoC*. Esta métrica representa o rácio de *tokens* por linha de código para um dado *concern*. Esta métrica foi a mais explorada em [17] e [22].

3.2. Adaptação da ferramenta

Inicialmente, foram realizados alguns testes acerca da IMP que obrigaram à implementação de várias classes *java.* Essas classes foram desenvolvidas em conjugação com o pacote *CCCExplorer* já existente, para que fosse possível estudar e comparar as várias relações de coocorrência entre cada par de *tokens* ao nível da tabela de *tokens*.

Para que fosse possível estudar cada uma das métricas, a ferramenta sofreu algumas modificações no sentido de gerar um *output* apenas com os *tokens* presentes nos ficheiros *MATLAB*, espaçados entre si e sem a presença das *tags*. Assim, a Figura 3.3 ilustra as alterações que foram realizadas sobre o *output* da Figura 3.2 e que facilitam a análise de todo o repositório de *tokens* sobre as métricas a utilizar.

```
N function cf cb count_repeat seq N
length seq cf zeros N N disp
for i N for j i cf i j
length strfind seq seq i j i end
end
seq seq end
cb zeros N N disp for i N for
j i cb i j length strfind seq seq
i j i end
end
cb cb end
```

Figura 3.3: *Output* modificado da ferramenta *CCCExplorer*

A Figura 3.3 corresponde a uma pequena amostra do repositório de *tokens* que se pretende estudar e validar (no total, são cerca de sete milhões de *tokens*). Esta amostra pretende clarificar aquilo que será o repositório total a ser analisado, sendo que o que aqui foi feito para um pedaço de código *MATLAB* (Figura 3.1) será abrangido a todo o repositório de código *MATLAB*.

A implementação das várias métricas descritas nas secções 2.4.1 e 2.4.2 foram

desenvolvidas numa nova classe - *relatedWords* - inserida no pacote *src/cccfinder/metrics* (essa classe pode ser vista em maior detalhe na secção Anexo Código). Essa classe tem acesso a todo o repositório *MATLAB* que se pretende analisar, permitindo fazer as contagens de *tokens* e pares de *tokens* necessárias para cada métrica. Para que essas contagens sejam facilitadas, inicialmente criou-se dois *HashMap* para preencher com todas as contagens dos *tokens* e dos pares de *tokens* do repositório, sendo que o primeiro campo destina-se à inserção de uma *String* (correspondendo ao *token* ou ao par) enquanto o segundo campo corresponde à contagem efetuada sobre o repositório. A partir dos dois *HashMap*, a tarefa de implementar cada uma das métricas é facilitada porque todas as contagens já estão efetuadas e devolver um valor a partir do *HashMap* é relativamente rápido. Após isso, seguiu-se com rigor cada uma das métricas, desenvolvendo-as consoante as suas especificidades e recorrendo a cada um dos *HashMap* sempre que era necessário devolver um valor qualquer de uma contagem ou probabilidade.

A maioria das classes criadas dizem respeito ao estudo da tabela de *tokens* e foram desenvolvidas na raíz *test*, permitindo gerar um *output* em formato *.html*. Os subpacotes da raíz *test* recebem como *input* um conjunto de *tokens* que será depois apresentado em forma de tabela com todos os resultados obtidos entre eles (sobre o repositório de *tokens*) para cada uma das métricas. Essas tabelas são apresentadas em maior detalhe no capítulo 4.

3.3. Método para a identificação de *concerns*

Os ajustes realizados sobre a ferramenta pretendem fazer o cálculo de várias métricas sobre pares de *tokens*, quer pertençam ao mesmo *concern* ou não. Existindo, até ao momento, a Tabela 2.2 da página 20 que permite, a partir de um conjunto de *tokens*, identificar *concerns* específicos, o método aqui apresentado permite validar e melhorar essa identificação, a partir da análise de coocorrência entre dois *tokens*.

Numa primeira instância, serão descartados os *tokens* que não estão presentes no repositório de *tokens*, visto que a sua ausência não permite identificar qualquer tipo de *concern*. Essa remoção é realizada com o auxílio da ferramenta *CCCExplorer*, que alerta o utilizador da ausência de um *token* aquando do estudo de cada par de *tokens* para cada *concern*. Consoante o repositório que se está a utilizar, o número de *tokens* descartados pode ser bastante variado, permitindo ao utilizador "limpar" um conjunto de *tokens* que nada favorece a abordagem a estudar. Além disso, algumas métricas não respondem de forma adequada quando o número de determinado *token* é relativamente pequeno num grande repositório, os quais deverão também ser descartados.

Após a remoção de tais *tokens*, pretende-se estudar os *tokens* ao nível de um *concern*, percebendo qual o conjunto de *tokens* que mais se adequa para identificar um *concern*. Essa validação é realizada a partir do estudo de cada par de *tokens* de um *concern*, verificando-se quais os valores mais elevados de cada métrica para cada par do conjunto de *tokens*. No capítulo 4 verifica-se que mesmo pares com valores baixos nas métricas podem ser informativos.

Uma segunda abordagem pretende estudar os *tokens* a um nível global da tabela, ou seja, estudar pares de *tokens* que pertençam a *concerns* diferentes. Os valores mais elevados entre alguns pares pode indiciar novas mudanças na tabela de *tokens* e, assim, propiciar melhorias no que à identificação de *concerns* diz respeito. Esta análise merece algum cuidado porque um *token* pode coocorrer várias vezes com *concerns* diferentes, dificultando a identificação de *concerns* ao longo do código *MATLAB*.

Por último, pretende-se encontrar novos *concerns* a partir do estudo de coocorrência entre *tokens* e palavras reservadas de *MATLAB*. Alguns dos pares encontrados podem fornecer informações interessantes acerca de conceitos de código *MATLAB* e de novos padrões sintáticos encontrados na linguagem.

A validação destas abordagens é feita nos capítulos 4 e 5, com uma breve discussão no final de cada um deles para que se perceba eventuais validações e melhorias que possam ser realizadas.

4. Associação entre *tokens*-palavra

Este capítulo valida e refina as entradas da tabela de *tokens* a partir do estudo e análise de várias métricas de associação de *tokens*-palavra mencionadas nas secções 2.4.1 e 2.4.2. Com base nos resultados obtidos a partir dessas métricas, a tabela de *tokens* sofrerá algumas propostas de modificações, que serão devidamente justificadas nas secções seguintes.

4.1. Informação mútua pontual para um *concern*

A IMP ganha especial interesse por ser trivial, ou seja, a implementação da sua fórmula em *java* é relativamente rápida e fácil de se fazer e, além disso, apresenta resultados motivadores. Essas características contribuíram para que se estudasse, num primeiro momento, a forma como alguns *tokens* se relacionavam para uma determinada entrada da tabela de *tokens*. Assim, estudou-se a relação de coocorrência entre *tokens* para um concern específico, o *concern verification of function arguments and return values*, onde foram analisados os resultados obtidos e retiradas algumas notas relativamente a melhorias que poderiam ser feitas para facilitar a forma como se tratava dos dados. A escolha deste *concern* deveu-se, sobretudo, à pequena quantidade de *tokens* que tinha (*nargchk*, *nargin*, *nargout*, *nargoutchk*, *varargin*, *varargout*) e, por outro lado, às boas perspetivas de coocorrências que existiam entre alguns deles no repositório.

Com base nos seis *tokens* presentes neste *concern*, produziu-se a Tabela 4.1 que mostra a IMP e as coocorrências dos pares de *tokens* (entre parêntesis), destacando cada um deles com a cor correspondente à sua IMP:

Tokens	nargchk	nargin	nargout	nargoutchk	varargin	varargout
nargchk	-	8.2 (826)	7.1 (88)	0 (0)	0 (0)	0 (0)
nargin	0 (0)	-	2.3 (59)	0 (0)	0.5 (122)	1.9 (79)
nargout	0 (0)	0.04 (12)	-	0 (0)	0 (4)	7.7 (1064)
nargoutchk	0 (0)	0 (0)	10.4 (366)	-	0 (0)	0 (0)
varargin	0 (1)	-1.04 (43)	-1.4 (8)	2.9 (16)	-	2.6 (229)
varargout	0 (0)	0 (0)	6.6 (494)	0 (0)	-0.6 (25)	-

Tabela 4.1: Exemplo de *output* através da implementação em *java* da IMP

Legenda:

- valor de IMP negativo ou 0, inclusivé
- valor de IMP entre 0 e 1, inclusivé
- valor de IMP superior a 1

A partir da Tabela 4.1 é possível fazer uma seleção daqueles pares de *tokens* que mais se destacam. A IMP não permite classificar os pares como sendo “bons” e “maus” visto que não existe um valor limite para essa definição. O método usado ao longo da tese teve por base a comparação de valores para perceber quais aqueles pares que mais se destacavam para um conjunto de pares a ser estudado. Dessa maneira, eliminou-se aqueles pares que tinham um valor de IMP menor que um (para este caso), filtrando apenas os que estão sombreados a verde. Essa filtragem permite criar uma lista com todos os pares de *tokens* ordenados de forma descendente pela IMP, o que facilita a visualização dos dados. Na Tabela 4.1 verifica-se que não faz sentido estudar a coocorrência de um par com o mesmo *token* e, além disso, a tabela define uma ordem para os pares, sendo que aqueles que se encontram na primeira coluna correspondem ao primeiro *token* e aqueles que estão na primeira linha correspondem ao segundo *token*. Essa ordem pela qual os *tokens* se encontram é importante porque a ordem com que eles se apresentam podem dar-nos informações diferentes no código.

Para que exista uma contagem de um par de *tokens* no repositório é necessário que não exista nenhum outro *token*-palavra entre esses dois *tokens*, ou seja, o par a ser estudado tem que estar contíguo no repositório de *tokens*.

O par *nargoutchk-nargout* foi aquele que maior valor de IMP apresentou (10.4) com uma coocorrência no repositório de 366. De salientar que o par *nargoutchk-nargout* e *nargout-nargoutchk* são pares diferentes, como referido anteriormente. Quando a IMP entre dois *tokens* é considerada elevada num certo *concern* indica que esses *tokens* apresentam um grau de proximidade muito relevante e, como tal, evidenciam a possibilidade de estarem presentes na mesma entrada da tabela.

As figuras seguintes ilustram um exemplo da coocorrência dos *tokens nargoutchk* e *nargout* ao longo de código *MATLAB*:

```
error (nargchk(1,2,nargin))
error (nargoutchk(0,1,nargout))
if nargin == 1
      Opt = DClab.DCOptions;
end
```

Figura 4.1: Uso das funções *nargoutchk* e *nargout* em *MATLAB*

```
function bool = isempty (obj)
error (nargoutchk(0,1,nargout))
error (nargchk(1,1,nargin))
```

Figura 4.2: Uso das funções *nargoutchk* e *nargout* em *MATLAB*

As Figuras 4.1 e 4.2 mostram dois casos similares em que os *tokens nargoutchk* e *nargout* são usados em conjunto. A análise sobre vários *m-files* provou que na maior parte das vezes em que esses dois *tokens* aparecem relacionados dizem respeito a casos muito semelhantes com aqueles aqui expostos. A função *nargoutchk* (*minArgs*, *maxArgs*, *numArgs*) valida o valor de *numArgs*, e retorna uma mensagem de erro se o *numArgs* é menor que o *minArgs* ou maior que o *maxArgs* [55]. Esse *numArgs* corresponde, precisamente, ao *token nargout*, que é o número de argumentos de *output* da função [54].

Além dessa forte relação, percebe-se que todos os *tokens* desta entrada têm pelo menos uma relação de coocorrência com algum dos outros *tokens*, indicando uma boa relação entre todos os *tokens* deste *concern*.

De uma forma geral, o aspeto da Tabela 4.1 mostra que existem bons indicadores para os *tokens* presentes, dada a quantidade de valores elevados entre alguns deles. Além disso, esta tabela é aquela que melhor resultados apresenta para a IMP, tendo em conta o número de *tokens* que existem e as suas coocorrências.

Em maior detalhe, verifica-se que todos os *tokens* que se apresentam na primeira coluna da Tabela 4.1 possuem pelo menos um valor de IMP superior a um com qualquer outro *token*. O mesmo não se passa com alguns *tokens* que se apresentam na primeira linha da mesma tabela, em que os *tokens nargchk* e *varargin* não têm nenhuma coocorrência de interesse com nenhum outro *token*.

Ainda que esta tabela mostre alguns resultados interessantes, outras experiências com outros *concerns* mostraram que existem muitos *tokens* na mesma entrada da tabela de *tokens* que não possuem qualquer relação com outros *tokens* dessa entrada. Contudo, mesmo *tokens* que não apresentem relações de interesse com *tokens* do mesmo *concern* podem ser informativos no sentido em que dois *tokens* do mesmo *concern* podem realizar as mesmas operações (os quais nunca poderão ser usados simultaneamente). Estas conclusões preliminares serão detalhadas nas secções seguintes com algumas melhorias no *output* das tabelas relativas às métricas e uma análise mais profunda sobre cada um dos *concerns*.

4.2. Melhorias na análise a efetuar

Os primeiros testes realizados com a IMP fizeram perceber a necessidade de se realizar algumas melhorias ao nível da obtenção e análise de resultados. Dessa maneira, houve alterações na tabela de *output* da IMP e aplicou-se para cada *concern* cada uma das métricas apresentadas nas secções 2.4.1 e 2.4.2 (permitindo comparar os diversos resultados). Assim, esta secção faz um planeamento das estratégias que serão efetuadas de forma a que a tabela de *tokens* seja devidamente validada e melhorada consoante os resultados que se forem obtendo.

Uma das primeiras modificações efetuadas foi a alteração do *output* gerado pela IMP (o qual seria também aplicado sobre as restantes métricas). Como foi referido na secção 4.1, o

output tinha o formato de uma tabela com cores distintas mostrando as relações entre os vários pares de *tokens* (ver Tabela 4.1). Contudo, esse *output* produzia poucos *tokens*, os quais eram facilmente observáveis a "olho nu". Por outro lado, alguns *concerns* possuem imensos *tokens*, dificultando a forma como se extrai os resultados obtidos. Dessa forma, foram tomadas algumas decisões relativamente ao *output* gerado pelas métricas de maneira a que fosse possível fazer uma análise cuidada mesmo quando o número de *tokens* é bastante elevado. A alternativa foi optar por uma lista com todos os pares de *tokens* (de um *concern*), seguidos da sua IMP (superior a 0) ordenados de forma descendente. Assim, os pares de *tokens* que indicam ter boas relações de proximidade aparecem no topo da lista e permitem uma comparação mais facilitada entre eles.

Quando foram realizadas as primeiras experiências com a IMP notou-se que muitos dos *tokens* da tabela de *tokens* não apareciam no repositório de código *MATLAB*. Tais *tokens* foram descartados para a aplicação de cada uma das métricas. Ainda que se tenham descartados muitos *tokens* após essa verificação, também foram descartados aqueles *tokens* que apareciam menos de seis vezes no repositório, visto que não se mostravam informativos sobre nenhuma das métricas a aplicar. De salientar que os *tokens* descartados poderão ter contagens completamente diferentes quando aplicados sobre outro repositório.

Após a implementação das várias métricas em *java*, o objetivo é gerar o *output* de cada uma delas e analisar os resultados, estudando e comparando alguns pares de *tokens* que se destacam entre as várias métricas. O estudo realizado incide sobre duas vertentes:

- Estudar a relação *token-token* para um *concern* específico;
- Estudar a relação *token-concern* ao longo da tabela de *tokens*.

As diferentes análises que se fazem nesta secção pretendem preparar as decisões que se vão tomar consoante os resultados que serão apresentados por cada uma das métricas, sendo que a IMP será a métrica base para analisar os resultados, embora as restantes sejam cuidadosamente justificadas quando necessário e anexadas na secção Anexos. Como tal, tudo o que será descrito nesta secção generaliza potenciais alterações na tabela de *tokens* sem que se entre em detalhes sobre os valores obtidos a partir de cada uma das métricas.

Inicialmente, o estudo das métricas foca-se em cada um dos *concerns*, de forma a que seja possível verificar a relação entre os *tokens* para uma determinada entrada da tabela de *tokens*. Este estudo pretende fazer uma comparação entre o *output* gerado por cada uma das métricas e, assim, verificar quais aqueles *tokens* e pares de *tokens* que ganham destaque sobre cada uma delas (tendo a IMP um peso maior). O objetivo é perceber quais os *tokens* que coocorrem mais para uma dada entrada da tabela de *tokens* e aqueles que, por outro lado, não possuem relações de interesse com nenhum dos outros *tokens* dessa entrada. Logicamente que existirão *tokens* na mesma entrada da tabela que poderão não ter coocorrências entre eles visto que ou se utiliza um *token* ou outro. Esta análise mais particular permite, desde logo, perceber quais os *concerns* que possuem *tokens* que se relacionam facilmente, aqueles que têm *tokens*

que praticamente não possuem relações com nenhum *token* desse mesmo *concern* e, por último, aqueles *concerns* que têm dois conjuntos de *tokens* (os que se relacionam e os que não se relacionam).

Após a análise de cada *concern*, a IMP irá estudar as relações dos *tokens* a um nível geral da tabela de *tokens*, ou seja, perceber quais os *tokens* que coocorrem bastantes vezes ou possuem um valor de IMP relativamente alto com outros *concerns*. No entanto, esta análise é dificultada pelo fato de um *token* poder possuir uma forte relação com vários *tokens* de *concerns* distintos, dando a entender que poderá ser um *token* que aparece em vários contextos e a sua posição na tabela de *tokens* não seja assim tão clara. Além disso, também podem existir *tokens* que possuem não só boas relações com os *tokens* do seu *concern*, mas também com *tokens* de um *concern* específico. Neste caso, é necessário perceber qual o contexto em código *MATLAB* da utilização desse *token* para que seja possível definir uma entrada em que ele possa ser inserido.

A análise de cada um dos estudos anteriores poderá desde logo trazer algumas conclusões acerca da posição de alguns *tokens* na tabela de *tokens*. Contudo, existem muitas funções que necessitarão de uma outra análise, ou seja, perceber o seu contexto no código *MATLAB* visto que as métricas poderão não ser suficientes para se chegar a uma conclusão.

4.3. *Tokens* descartados da tabela

O código *MATLAB* disponível para o estudo das várias métricas permite identificar cerca de sete milhões de *tokens*-palavra (incluindo nomes de funções que não pertencem à tabela de *tokens*). Contudo, este número mostrou-se relativamente pequeno por não conseguir abranger todos os *tokens* da tabela de *tokens* e por apresentar, muitas das vezes, contagens pequenas sobre as ocorrências e coocorrências de certos *tokens*. Assim, o trabalho desenvolvido nesta tese está limitado pelo repositório de *tokens* e quaisquer resultados que se obtenham em função de cada métrica de associação de *tokens*-palavra é sempre restringido por essas limitações. Um maior repositório de código *MATLAB* poderia permitir um maior número de resultados obtidos e, por consequência, a obtenção de mais conclusões acerca das relações dos *tokens* na tabela.

A Tabela 4.2 lista, para cada *concern*, os *tokens* que foram descartados por se apresentarem no repositório com frequências demasiados baixas (menos de seis vezes):

Concern	Tokens descartados
Verification of function arguments and return values	-
Data type specialisation	fimath, sfi, ufi
Data type verification	iscolumn, isfi, isfimath, isfixed, isinfinite, isquantizer, wordlength
Dynamic properties	-
Console messages	-
Printing	printopt
Visualisation	aaxes, figurepalette, gtext, plotbrowser, plotedit, plottools, propertyeditor, rgbplot, showplottool, texlabel, imformats, im2frame, VideoReader, VideoWriter
File I/O	fileformats
System	inmem, inputparser, isglobal, iskeyword, namelengthmax, pcode, libfunctionsview, onCleanup, clearvars, weekday, calendar, dbcont, dbquit, dbmex, bafter, ebreak, nanbreak, rbreak, tbreak, zcbreak, munlock, startat, timerfindall, profsave
Memory allocation/deallocation	-
Parallelisation	createParallelJob, createTask, defaultParallelConfig, demote, destroy, detupForParallelExecution, dfeval, dfevalasync, distributed, gcat, gop, gplus, gpuArray, gpuDevice, gpuDeviceCount, importParallelConfig, isreplicated, jobStartup, labBarrier, labBroadcast, labProbe, mpiLibConf, mpiprofile, mpiSettings, pctconfig, pctRunDeployedCleanup, pctRunOnAll, pload, poolStartup, psave, redistribute, submit, taskFinish, taskStartup

Tabela 4.2: *Tokens* descartados da tabela de *tokens* por se apresentarem muito poucas vezes no repositório *MATLAB*

A Tabela 4.2 mostra que o *concern parallelisation* é aquele que mais *tokens* descarta por serem *tokens* que são usados em contextos muito particulares de código *MATLAB* e, como tal, não são usados frequentemente como outros *tokens* da tabela o são.

A filtragem feita a partir dos *tokens* descartados permite eliminar muitos *tokens* da tabela o que facilita, posteriormente, a forma como os dados obtidos são analisados. O fato de se descartar estes *tokens* faz com que muitos pares de *tokens* não apareçam no *output* das métricas e, por isso, não "inundem" os resultados obtidos.

4.4. Análise das métricas sobre os *concerns*

Definido aquilo que se pretende realizar com cada uma das métricas ao longo do repositório de *tokens*, será agora feita uma análise bastante detalhada acerca de cada um dos *concerns* de forma a que possamos tirar algumas conclusões acerca da presença de cada um dos *tokens* na tabela. A métrica que será tida em consideração será a IMP por, ao longo dos vários testes, ter mostrado resultados motivadores sempre que existia boas coocorrências entre *tokens*. As restantes métricas servem como um apoio comparativo para a análise da IMP, embora sejam mencionadas sempre que houver necessidade e, além disso, serão apresentadas em anexo. As tabelas em anexo apresentam duas cores distintas: verde, que corresponde a pares que aparecem mais de dez vezes no repositório e vermelho, aqueles que não têm contagens superiores a dez.

As tabelas resultantes da IMP da secção seguinte são mais elaboradas do que aquelas que estão em anexos, apresentando quatro colunas mencionando os pares de *tokens* que se está a estudar (limite de cinco pares por *concern*), o seu rácio ao longo do repositório (com as contagens individuais de cada um dos *tokens* no repositório), as suas coocorrências e, por fim, o valor da IMP. O segundo e terceiro campos são importantes porque ajudam a perceber se o número de vezes que dois *tokens* aparecem (individualmente ou em conjunto) reforça ou não o valor resultante da IMP.

4.4.1. Análise sobre cada *concern*

Esta subsecção apresenta, sempre que possível, as tabelas de *output* para cada um dos *concerns* da tabela de *tokens*. Existem casos em que a IMP não obtém qualquer resultado dada as poucas coocorrências entre os *tokens* do mesmo *concern*. Após a ilustração das tabelas é feita uma descrição acerca dos resultados obtidos sobre possíveis alterações que se possam efetuar na tabela de *tokens*, embora esta subsecção sirva essencialmente para analisar de uma maneira eficaz quais aqueles *tokens* que se relacionam melhor para um *concern* específico. Após a análise sobre os *concerns* espera-se chegar a algumas conclusões sobre a posição de alguns

tokens na tabela e com algumas motivações para a aplicação da IMP ao nível de toda a tabela.

O estudo da IMP sobre o *concern verification of function arguments and return values* mostrou, desde logo, algumas relações que prometiam ser interessantes (ver secção 4.1). Uma nova experiência com algumas melhorias no *output* assim como a sua abordagem sobre outras métricas torna possível retirar algumas conclusões acerca da presença dos *tokens* deste *concern*. O *concern verification of function arguments and return values* apresentou resultados bastante interessantes entre os *tokens* dessa entrada, como é possível verificar na Tabela 4.3 (nos anexos A1 existe uma versão estendida desta tabela assim como as tabelas das restantes métricas):

Tokens	Rácio	Coocorrências	IMP
nargoutchk - nargout	$\frac{426}{4327}$	366	10.4
nargchk - nargin	$\frac{1026}{18589}$	826	8.2
nargout - varargout	$\frac{4327}{8013}$	1064	7.7
nargchk - nargout	$\frac{1026}{4327}$	88	7.1
varargout - nargout	$\frac{8013}{4327}$	494	6.6

Tabela 4.3: Informação mútua pontual para o *concern verification of function arguments and return values*

Numa primeira visualização sobre os dados obtidos verifica-se que todos os pares de *tokens* coocorrem no repositório com valores altos, o que facilita a forma como devem ser analisados. Além disso, verifica-se que estes *tokens* possuem frequências individuais altas quando comparado com outros *tokens* de outros *concerns*. Com o *output* de cada uma das métricas é possível verificar que o par *nargoutchk-nargout* aparece destacado em todas elas, apenas não se apresentando em primeiro lugar na função *dice* (tornando a relação destes dois *tokens* bastante forte). A IMP de cada par também é bastante elevada.

O único *token* que não aparece nesta tabela é o *token varargin*, o qual poderia indiciar que deveria estar presente numa outra entrada da tabela de *tokens*. Contudo, quando analisamos a tabela estendida da IMP (ver anexo A1.1) assim como as tabelas das restantes métricas (ver anexos A1.2, A1.3 e A1.4) verifica-se facilmente que esse *token* já aparece nessas tabelas, ao contrário daquilo que se passa na Tabela 4.3.

Também o par *nargchk-nargin* se apresenta com valores relativamente altos e com destaque em todas as métricas. O único *token* que não se destaca tanto como os outros, mas

também não deve ser descartado é o *varargout*, que ainda assim apresenta, juntamente com o *token nargout*, o valor mais elevado para a função *dice*.

Desta forma, e dada a potencialidade deste *concern* (o único que apresenta relações de coocorrência para todos os *tokens*), faz sentido manter os *tokens* deste *concern* e não movê-los para nenhuma outra entrada da tabela.

O *concern data type specialisation* e *data type verification* apresentam resultados interessantes para a IMP, mas, por outro lado, as coocorrências entre *tokens* no repositório são baixas, como mostram as Tabelas 4.4 e 4.5. Tanto numa tabela como noutra existem alguns *tokens* que se relacionam melhor do que outros, mas o fato de muitos deles serem utilizados em com propósitos ou fins similares no código *MATLAB* faz com que nem todos possam ser utilizados em conjunto e, consequentemente, serem apresentados nas várias métricas.

Tokens	Rácio	Coocorrências	IMP
uint32 - uint64	$\frac{277}{44}$	5	11.5
double - uint64	$\frac{3215}{44}$	5	7.9
double - int16	$\frac{3215}{285}$	20	7.2
uint32 - double	$\frac{277}{3215}$	8	6.0
double - int32	$\frac{8013}{3215}$	11	5.7

Tabela 4.4: Informação mútua pontual para o *concern data type specialisation*

Tokens	Rácio	Coocorrências	IMP
intmax - class	$\frac{33}{2287}$	5	8.8
cast - class	$\frac{60}{2287}$	5	8.0
isstr - range	$\frac{1100}{1807}$	10	5.1
isempty - class	$\frac{35640}{2287}$	25	1.08
length - class	$\frac{52026}{2287}$	36	1.06

Tabela 4.5: Informação mútua pontual para o *concern data type verification*

As Tabelas 4.4 e 4.5 chamam a atenção pela presença de pares de *tokens* que coocorrem menos de dez vezes ao longo do repositório. Apesar da Tabela 4.4 apresentar valores mais interessantes relativamente à IMP, a verdade é que essa tabela representa a totalidade da tabela obtida a partir da IMP. Por outro lado, a Tabela 4.5 é maior (ver anexo A3.1) mas apresenta, a partir da quarta linha, valores para a IMP relativamente baixos quando comparada com as primeiras três linhas.

O *concern data type specialisation* mostra que o *token uint64* apresenta-se com algum destaque em todas as métricas relacionando-se com os *tokens uint32* e *double*. Contudo, as suas coocorrências aparecem muito poucas vezes no repositório. Por outro lado, o par *double-int16* também se destaca (sobretudo na IMP) e com valores de coocorrência um pouco superiores a dez (embora continue a ser um valor baixo para o repositório em questão). Este fato faz com que estes *tokens* sejam aqueles que melhores ligações possuam nesta entrada da tabela de *tokens*. Todas as tabelas resultantes das métricas são bastante parecidas fazendo com que os pares de *tokens* sejam praticamente os mesmos em todas elas (ver anexos A2). A partir da Tabela 4.4 verifica-se que o *token double* é aquele que melhor se relaciona com os restantes *tokens* dessa entrada da tabela.

O *concern data type verification* destaca o *token class* como aparecendo em várias relações de coocorrência com outros *tokens* desse *concern*. Além do *token class*, também os *tokens intmin*, *intmax* e *cast* são destacados por cada uma das métricas, assim como o par *isstr-range* o é. Tal como o *concern data type specialisation*, também este apresenta dois tipos de *tokens*: aqueles que se relacionam e aqueles que não coocorrem com nenhum outro *token* deste *concern*. Esse fato deve-se, sobretudo, ao fato de alguns dos *tokens* do *concern data type verification* serem usados em contextos muito semelhantes do código *MATLAB*, os quais poderão constituir alternativas que não faz sentido usar simultaneamente.

Ambos os *concerns* pertencem a *data type*, sendo que o primeiro é *specialisation* e o segundo *verification*. Na atual versão da tabela de *tokens* existe uma separação entre ambos, os quais poderiam ser unidos pertencendo ambos os *concerns* a *data type*. Assim, pode existir uma ramificação dos dois *concerns* a partir de *data type* sendo que o *concern data type verification* pode ser dividido ainda em vários sub-*concerns*. Dessa maneira, o *concern data type verification* fica dividido da seguinte forma:

- **Identification and numeric types**
  - cast, class, intmax, intmin, isa, isboolean, iscell, ischar, isfloat, isinf, isinteger, islogical, isnan, isnumeric, isobject, isreal, isstr, isstruct, realmax, realmin, typecast;

- **Matrices and arrays**
  - isempty, isfield, isrow, isscalar, isvector, length, ndims, numel, range, size.

Concern	*Tokens*
Data type specialisation	fi, fimath, int16, int32, int64, int8, quantize, quantizer, sfi, single, ufi, uint16, uint32, uint64, uint8, double
Data type verification	cast, class, intmax, intmin, isa, isboolean, iscell, ischar, iscolumn, isempty, isfi, isfield, isfimath, isfixed, isfloat, isinf, isinfinite, isinteger, islogical, isnan, isnumeric, isobject, isquantizer, isreal, isrow, isscalar, isstr, isstruct, isvector, length, ndims, numel, range, realmax, realmin, size, typecast, wordlength

Tabela 4.6: Parte da atual tabela de *tokens* com os *concerns data type specialisation* e *verification*

Concern	*Sub-concerns*		*Tokens*
Data type	Specialisation		double, fi, int8, int16, int32, int64, quantize, quantizer, single, uint8, uint16, uint32, uint64
	Verification	Identification and numeric types	cast, class, intmax, intmin, isa, isboolean, iscell, ischar, isfloat, isinf, isinteger, islogical, isnan, isnumeric, isobject, isreal, isstr, isstruct, realmax, realmin, typecast
		Matrices and arrays	isempty, isfield, isrow, isscalar, isvector, length, ndims, numel, range, size

Tabela 4.7: Nova entrada da tabela de *tokens* para os *concern data type specialisation* e *verification*

A Tabela 4.6 representa a atual entrada da tabela de *tokens* para os *concerns data type specialisation* e *verification*, com uma separação entre ambos indicando que se trata de dois *concerns* diferentes. A Tabela 4.7 une os dois *concerns*, havendo uma posterior sub-divisão dos dois *concerns* a partir de *data type*. Esta sub-divisão pretende estruturar melhor a presença dos *tokens* na tabela para que cada um deles se posicione melhor sobre a sua função no código *MATLAB*. Assim, os sub-*concerns* funcionam como uma "ponte" entre os *concerns* e os *tokens*.

A nova coluna da tabela (referente a sub-*concerns*) foi construída com base no *site MathWorks* [47], que possui um manual com cada uma das funções/*tokens* num mapeamento entre essas funções e o nome que é dado aos sub-*concerns*. A divisão que foi aqui feita será realizada mais à frente sobre *concerns* que alberguem uma grande quantidade de *tokens*.

Ainda que os *concerns* anteriores mostrem boas relações de coocorrência entre *tokens*, também existem casos em que a IMP não apresentou resultados (ou mostrou resultados fracos) entre *tokens* de alguns *concerns*. Trata-se do caso dos *concerns dynamic properties*, *console messages*, *printing*, *memory allocation/deallocation* e *parallelisation* (sendo que estes últimos três não apresentam quaisquer resultados para a IMP). O fato destes *concerns* possuírem fracas coocorrências deve-se, naturalmente, ao contexto em que os seus *tokens* são utilizados, ou seja, em contextos muito semelhantes no código. Visto que muitos deles produzem o mesmo efeito no código, é pouco provável que sejam usados em conjunto. Apesar disso, algumas métricas (exceto a IMP) apresentam alguns valores sobre pares de *tokens* quando a própria IMP não apresenta qualquer resultado. Isso acontece devido às especificidades de cada uma das outras métricas e, sobretudo, por conseguirem obter resultados demasiado baixos (algo que a IMP ignora).

O *concern dynamic properties* (*eval*, *evalc*, *evalin*, *feval*, *inline*) executa e avalia expressões em *MATLAB*. Por exemplo, T = *evalc* (expressão) é o mesmo que *eval* (expressão) exceto que qualquer coisa que poderia normalmente ser escrito para a janela de comandos (exceto para mensagens de erro), é capturado e devolvido num *array* de carateres T [41][42]. Logicamente que dificilmente estas duas funções serão encontradas juntas ao longo do repositório, pelo que a sua IMP será provavelmente sempre nula. A Tabela 4.8 apresenta os resultados sobre este *concern* para a IMP (nos anexos A4 encontram-se os resultados das restantes métricas).

Tokens	Rácio	Coocorrências	IMP
evalin - eval	$\frac{3255}{6426}$	8	1.4

Tabela 4.8: Informação mútua pontual para o *concern dynamic properties*

Tanto a IMP como as restantes métricas apresentam coocorrências entre *tokens* para este *concern* que não ultrapassam as dez contagens no repositório. Aliás, o par *evalin-eval* é aquele que melhor relação possui e só tem apenas uma contagem de oito ao longo de todo o repositório. Isso mostra que *tokens* que executam as mesmas funcionalidades no código *MATLAB* (ou funcionalidades semelhantes) nunca irão apresentar (muitos) resultados para a IMP.

Os próximos parágrafos continuam a fazer uma análise acerca dos *concerns* que apresentaram resultados fracos para cada uma das métricas estudadas.

O *concern console messages* (*annotation*, *assert*, *disp*, *display*, *error*, *last*, *lastwarn*) exibe mensagens e informações para o utilizador. A Tabela 4.9 ilustra os resultados obtidos sobre a IMP para o *concern* em questão.

Tokens	Rácio	Coocorrências	IMP
display - disp	$\frac{1731}{22364}$	47	3.1
disp - error	$\frac{22364}{24484}$	212	1.4

Tabela 4.9: Informação mútua pontual para o *concern console messages*

A Tabela 4.9 mostra, claramente, a presença de três *tokens* que se relacionam de forma positiva sobre o *concern console messages*, com frequências relativamente elevadas. Também as restantes métricas (ver anexos A5) chamam a atenção pelas primeiras posições se referirem aos pares *display-disp* e *disp-error*, ainda que o par *error-disp* seja destacado por cada uma delas com uma frequência no repositório de 68. Numa primeira análise é possível referir que estes três *tokens* (*disp*, *error* e *display*) se devem manter juntos sobre este *concern*. Ainda que esses *tokens* sejam os únicos que aparentam manter relações de proximidade neste *concern* (apesar dos resultados apresentados não exibirem valores tão altos quando comparados com outros *concerns*), sabe-se que a maioria dos *tokens* presentes aqui não iriam apresentar quaisquer resultados, derivado do uso que requerem ao longo do código *MATLAB*.

Por exemplo, o *token last* devolve a última exceção não identificada [48] enquanto o *token lastwarn* devolve a última mensagem de aviso [49]. Dadas as características muito específicas de cada um desses *tokens* consoante as necessidades do utilizador, verifica-se que o uso dos dois simultaneamente é muito pouco provável. O mesmo acontece com as restantes funções do *concern console messages*.

O *concern printing* (*orient*, *print*, *printdlg*) possui *tokens* que imprimem e guardam figuras, sendo que são apenas alguns detalhes que distinguem as diferentes funções. Este *concern* é aquele que menos *tokens* tem ao longo da tabela de *tokens* e, dado o contexto em que as três funções são utilizadas, era de esperar que a IMP não obtivesse qualquer resultado sobre este *concern*. Nos anexos A6 é possível verificar que as restantes métricas conseguiram gerar uma tabela apenas com uma linha, referente ao par *orient-print* (que apenas coocorre uma vez ao longo de todo o repositório). Mais uma vez, a IMP ignorou um resultado aparentemente fraco (ao contrário das restantes) e, daí, esta métrica ser utilizada como justificação de possíveis validações e melhorias na tabela de *tokens*.

Tal como o *concern console messages*, também a IMP não apresentou qualquer *output* para o *concern memory allocation/deallocation*. Ainda que todas as outras métricas o tenham feito (ver anexos A7), é notória a quantidade de valores muito baixos que são apresentados, indicando baixas frequências para os pares no repositório. O *concern memory allocation/deallocation* (*delete*, *global*, *persistente*, *zeros*, *ones*) declara variáveis e inicializa *arrays*, sendo que nunca é possível utilizar, por exemplo, as funções *global* e *persistent*

juntamente. No caso da primeira, se várias funções declararem um nome de variável específico como global, todas partilharão uma única cópia dessa variável e, sempre que houver uma mudança de valor dessa variável (em qualquer função), é visível para todas as funções que a declararam como global [46]. Por seu lado, a função *persistent* declara variáveis como persistentes que são locais para a função na qual elas são declaradas [59]. As definições de cada uma das funções mostra que o uso de uma delas implica o não uso da outra e, consequentemente, estas duas funções nunca serão realçadas por nenhuma das métricas estudadas (o mesmo se passa com os restantes *tokens*).

O *concern parallelisation* possui mais *tokens* do que qualquer um dos quatro *concerns* mencionados anteriormente, mas, por outro lado, a IMP não destaca nenhum par ao longo desse *concern*. Os *tokens* presentes neste *concern*, tal como o próprio nome do *concern* indica, são funções utilizadas em contextos de paralelização e, por esse motivo, estão na posição certa da tabela e não podem ser encontrados em conjunto no código *MATLAB*. Com base nas restantes métricas estudadas (ver anexos A8), nota-se que o par *labindex-numlabs* é o único que aparece em cada uma delas, embora a frequência deste par ao longo do repositório seja apenas de dois. Verifica-se uma fraca conexão entre os *tokens* deste *concern*, embora o seu resultado não seja completamente surpreendente. Dessa forma, e dado o contexto em que essas funções de paralelização são utilizadas, os *tokens* presentes neste *concern* devem lá permanecer.

O *concern file I/O* apresenta resultados relativamente mais interessantes para a IMP quando comparado com os cinco *concerns* anteriores. Ainda assim, verifica-se na Tabela 4.10 que apenas quatro dos catorze *tokens* são salientados pela IMP (nos anexos A9 estão os resultados das restantes métricas).

Tokens	Rácio	Coocorrências	IMP
fopen - fprintf	$\frac{3275}{29192}$	79	2.5
fprintf - save	$\frac{29192}{2047}$	15	0.8
fprintf - load	$\frac{29192}{3936}$	19	0.2

Tabela 4.10: Informação mútua pontual para o *concern file I/O*

O *token fprintf* possui relações de coocorrência em todas as métricas com os *tokens fopen*, *save* e *load*. Aliás, tendo em conta as frequências conjuntas no repositório, as três relações aqui obtidas são as únicas que prometem ser comprometedoras para o *concern* em questão. Contudo, os valores da IMP para cada par são baixos e as suas frequências conjuntas

também, tendo em conta as vezes que cada *token* aparece individualmente no repositório. Nos anexos A9 é possível notar, a partir das restantes métricas, que existem mais relações de coocorrência entre alguns *tokens* deste *concern*, embora as suas frequências conjuntas no repositório sejam demasiado baixas (ao ponto de serem descartadas pela IMP).

Este *concern* possui *tokens* que realizam operações sobre ficheiros. Mais uma vez, são *tokens* que raramente são utilizados simultaneamente no código, mas que estão na posição correta da tabela, por serem chamados sempre que se pretende realizar operações específicas sobre ficheiros. Por exemplo, as funções *fgetl* e *fgets* leem ambas uma linha de um ficheiro, com a diferença que a primeira função remove os carateres da nova linha [43] enquanto a segunda função mantém-nos [44].

Os últimos dois *concerns* a serem estudados dizem respeito a *visualisation* e *system*. Estes *concerns* são aqueles que mais *tokens* possuem e, além disso, aqueles que melhores resultados apresentam entre os *tokens* do seu *concern* (com coocorrências bastante interessantes). Tal como acontece nos *concerns* de *data type specialisation* e *visualisation*, também estes têm relações fortes entre alguns *tokens* e *tokens* que não se corelacionam com nenhum dos *tokens* do próprio *concern*. A Tabela 4.11 mostra os resultados obtidos sobre o *concern visualisation* para a IMP.

Tokens	Rácio	Coocorrências	IMP
cla - reset	$\frac{568}{374}$	42	10.5
clf - reset	$\frac{1722}{374}$	96	10.0
ylabel - zlabel	$\frac{4206}{339}$	206	10.0
xlabel - ylabel	$\frac{4290}{4206}$	2435	9.9
datetick - title	$\frac{184}{7157}$	92	9.0

Tabela 4.11: Informação mútua pontual para o *concern visualisation*

A Tabela 4.11 exibe resultados muito motivadores nomeadamente quando olhamos para a coluna da IMP. Os valores aí apresentados são muito elevados quando comparados com a maior parte dos *concerns* da tabela de *tokens* e, além disso, a Tabela 4.11 corresponde a uma pequena parte do resultado obtido (nos anexos A10.1 está uma extensão desta tabela ainda que não corresponda ao tamanho total obtido). A extensão desta tabela continua a apresentar

resultados altos para a IMP e muitos dos *tokens* do *concern visualisation* aparecem nessa extensão. Além da IMP, as restantes métricas também apresentaram bons resultados, nomeadamente naquilo que são as coocorrências de pares no repositório (como indicam os imensos valores a verde nos anexos A10.2, A10.3 e A10.4 para cada uma das três métricas).

O *concern visualisation* diz respeito a operações sobre gráficos e, dada a grande quantidade de *tokens* que este *concern* possui foi possível criar sub-*concerns* sobre o *concern visualisation* (semelhantemente áquilo que foi feito com o *concern data type verification*). A sub-divisão realizada teve por base o site *MathWorks* [47]. Assim, temos:

- **2D and 3D Plots**
  - Animation: getframe, movie;
  - Contour plots: clabel;
  - Data distribution plots: hist, histogram, scatter;
  - Line plots: errorbar, fplot, gplot, loglog, plot, plot3, semilogx, semilogy;
  - Polar plots: polar;
  - Surfaces, volumes and polygons: mesh, meshgrid, surf;

- **Formatting and annotation**
  - Axes appearance: axis, box, datetick, figure, grid, plotyy, subplot;
  - Titles and labels: legend, line, rectangle, text, title, xlabel, ylabel, zlabel;

- **Graphics objects**
  - Graphics object identification: gca, gcbf, gcbo, gco;
  - Graphics object properties: reset, set;
  - Graphics object programming/output: cla, clf, close, hold, ishold, newplot;

- **Images**
  - frame2im, image, iminfo, imread, imwrite;

- **Visual exploration**
  - datacursormode, pan, rotate, rotate3d, zoom.

As Tabelas 4.12 e 4.13 correspondem, respetivamente, à atual entrada da tabela de *tokens* para o *concern visualisation* e a nova entrada proposta com base na sub-divisão feita anteriormente.

Concern	Tokens
Visualisation	aaxes, axis, box, cla, clabel, clf, close, datacursormode, datetick, errorbar, figure, figurepalette, fplot, gca, gcbf, gcbo, gco, getframe, gplot, grid, gtext, hist, histogram, hold, imfinfo, ishold, legend, line, loglog, mesh, meshgrid, newplot, pan, plot, plot3, plotbrowser, plotedit, plottools, plotyy, polar, propertyeditor, rectangle, reset, rgbplot, scatter, semilogx, semilogy, showplottool, subplot, surf, texlabel, text, title, xlabel, ylabel, zlabel, zoom, set, rotate, rotate3d, imformats, imread, imwrite, movie, image, frame2im, im2frame, VideoReader, VideoWriter

Tabela 4.12: Parte da atual tabela de *tokens* com o *concern visualisation*

Concern	Sub-concerns		Tokens
Visualisation	2D and 3D plots	Animation	getframe, movie
		Contour plots	clabel
		Data distribution plots	hist, histogram, scatter
		Line plots	errorbar, fplot, gplot, loglog, plot, plot3, semilogx, semilogy
		Polar plots	polar
		Surfaces, volumes and polygons	mesh, meshgrid, surf
	Formatting and Annotation	Axes appearance	axis, box, datetick, figure, grid, plotyy, subplot
		Titles and labels	legend, line, rectangle, text, title, xlabel, ylabel, zlabel
	Graphic Objects	Identification	gca, gcbf, gcbo, gco
		Properties	reset, set
		Programming/output	cla, clf, close, hold, ishold, newplot
	Images		frame2im, image, iminfo, imread, imwrite
	Visual exploration		datacursormode, pan, rotate, rotate3d, zoom

Tabela 4.13: Nova entrada da tabela de *tokens* para o *concern visualisation*

A Tabela 4.14 ilustra os resultados apresentados para o *concern system*, com base na IMP. Tal como o *concern visualisation*, também o *concern system* apresenta valores igualmente elevados para cada par apresentado na tabela (nos anexos A11.1 é possível visualizar a tabela da IMP para este *concern* na íntegra).

Tokens	Rácio	Coocorrências	IMP
unloadlibrary - loadlibrary	$\frac{20}{18}$	6	16.8
loadlibrary - calllib	$\frac{18}{352}$	11	13.6
etime - clock	$\frac{154}{480}$	95	13.1
pause - echo	$\frac{1539}{201}$	25	9.13
toc - tic	$\frac{1263}{1163}$	118	9.12

Tabela 4.14: Informação mútua pontual para o *concern system*

Comparativamente ao *concern visualisation*, as métricas apresentam valores altos para muitos dos pares do *concern system*. Ainda assim, muitos deles não se apresentam no repositório com frequências altas, tal como indicam os muitos pares a vermelho em anexos (ver anexos A11.2, A11.3 e A11.4).

Tendo em conta a quantidade de *tokens* que existe para este *concern* e o número de resultados que foram produzidos a partir de cada uma das métricas, a análise a ser feita coincide muito com aquela que se faz para o *concern visualisation*. Ainda assim, o *concern visualisation* apresentou uma maior extensão da tabela da IMP e, além, disso, as coocorrências dos pares no repositório são bastante mais altas para esse *concern* do que para o *concern system*.

Mais uma vez, a enorme quantidade de *tokens* que este *concern* possui permitiu gerar sub-divisões que facilitam a forma como os *tokens* são tratados. Assim, temos:

- **Code analysis**
  - run, timerfind;
- **Control flow**
  - break, next, pause;
- **Data import and export**
  - clear, who, whos;

- **Dates and time**
  - addtodate, clock, date, etime, now;
- **Debugging**
  - dbstop, echo;
- **Entering commands**
  - ans, slist, stop;
- **Files and folders**
  - exist, rehash;
- **Functions**
  - inputname, isvarname, mfilename, mislocked, mlock;
- **Performance and memory**
  - cputime, memory, pack, profile, tic, toc;
- **Scripts**
  - batch, input;
- **Startup**
  - start;
- **Using external libraries**
  - calllib, libisloaded, loadlibrary, mex, mexext, unloadlibrary.

A sub-divisão aqui realizada permite alterar a entrada da tabela de *tokens* do *concern system* e, assim, criar sub-*concerns*. A sub-divisão realizada não permitiu a alocação dos *tokens ba*, *where* e *systems* em nenhum dos sub-*concerns* pela dificuldade em encontrar uma definição de cada uma dessas funções, que não existia no site MathWorks [47].

As Tabelas 4.15 e 4.16 correspondem à atual e nova proposta da entrada da tabela de *tokens*, respetivamente, para o *concern system*.

Concern	Tokens
System	ans, echo, exist, inmem, input, inputname, inputParser, isglobal, iskeyword, isvarname, mexext, mfilename, namelengthmax, pcode, symvar, who, whos, systems, slist, where, loadlibrary, mex, calllib, libisloaded, unloadlibrary, libfunctionsview, onCleanup, clearvars, rehash, pack, memory, clear, addtodate, now, weekday, date, calendar, dbcont, dbquit, dbstop, dbmex, ba, bafter, break, ebreak, nanbreak, rbreak, tbreak, xbreak, zcbreak, wait, stop, run, batch, spmd, pause, step, next, mislocked, mlock, munlock, clock, cputime, etime, start, startat, tic, timerfind, timerfindall, toc, profile, profsave

Tabela 4.15: Parte da atual tabela de *tokens* com o *concern system*

Concern	Sub-concerns	Tokens
System	Code analysis	run, timerfind
	Control flow	break, next, pause, wait, xbreak
	Data import and export	clear, who, whos
	Dates and time	addtodate, clock, date, etime, now, step
	Debugging	dbstop, echo
	Entering commands	ans, slist, stop
	Files and folders	exist, rehash
	Functions	inputname, isvarname, mfilename, mislocked, mlock, symvar
	Performance and memory	cputime, memory, pack, profile, tic, toc
	Scripts	batch, input
	Startup	start
	Using external libraries	calllib, libisloaded, loadlibrary, mex, mexext, unloadlibrary

Tabela 4.16: Nova entrada da tabela de *tokens* para o *concern system*

A Tabela 4.16 mostra um número mais elevado de sub-*concerns* relativamente ao *concern visualisation* devido a este *concern* possuir uma grande variedade de *tokens* com definições muito particulares que exercem no código *MATLAB* especificidades que diferem muito um dos outros, fazendo com que não seja possível agrupar demasiados *tokens* num só sub-*concern*.

4.4.2. Análise sobre todos os *concerns*

O estudo das relações entre *tokens* para a mesma entrada da tabela (ou seja, para um *concern* específico) torna-se interessante na medida em que permite perceber quais os *tokens* de um *concern* que se corelacionam com regularidade e aqueles que não coocorrem. Como foi possível verificar anteriormente, existem muitos *tokens* que não têm qualquer tipo de relação com nenhum outro *token* da entrada onde eles estão inseridos. Esse fato motivou o estudo das relações a outro nível, ou seja, permitir estudar as várias relações entre *tokens* sobre todos os *concerns*, ajudando a verificar quais aqueles que possuem coocorrências mais interessantes com outros *concerns* do que aqueles onde eles estão inseridos. Contudo, esse estudo requer alguma atenção visto que alguns *tokens* no mesmo *concern* podem não coocorrer dada as suas características semelhantes no código (como foi mencionado na subsecção anterior). Derivado disso, a análise realizada nesta subsecção vai para além dos dados quantitativos obtidos e, como tal, será feita uma descrição de cada função para que se perceba até que ponto será possível mover um *token* para outra entrada da tabela de *tokens*.

A Tabela 4.17 corresponde ao *output* da IMP sobre todas as relações entre todos os *concerns*, exceto para *tokens* que pertençam ao mesmo *concern* (que foi realizado na secção anterior). A tabela seguinte corresponde a uma pequena amostra daquilo que foram os resultados obtidos, e aqueles que são aqui apresentados têm por base a sua IMP e, sobretudo, as suas coocorrências ao longo do repositório:

Tokens	Rácio	Coocorrências	IMP
matlabpool - spmd	$\frac{70}{9}$	6	16
error - nargchk	$\frac{24484}{1026}$	1005	8
zeros - size	$\frac{23672}{43792}$	4106	4.8
ones - size	$\frac{12099}{43792}$	2053	4.7
nargin - error	$\frac{18589}{24484}$	1725	4.7
ones - length	$\frac{12099}{52026}$	1316	3.9
zeros - length	$\frac{23672}{52026}$	1944	3.4

Tabela 4.17: Informação mútua pontual sobre toda a tabela de *tokens*

Numa primeira visualização sobre os dados obtidos verifica-se que o primeiro par apresentado (*matlabpool-spmd*) apresenta um valor de IMP muito elevado quando comparado com os restantes. Contudo, a coocorrência do par no repositório é apenas de seis, ainda que o *token spmd* só apareça nove vezes (indicando que, sempre que *spmd* aparece no código, este encontra-se quase sempre relacionado com o *token matlabpool*). Nenhum desses *tokens* se destaca nos seus *concerns*, ou seja, não aparentam qualquer relação com nenhum outro *token* nas suas entradas da tabela de *tokens*. Essa relação mostrou desde cedo bons indícios acerca da mudança de um dos *tokens* para o mesmo *concern* do outro. Tendo em conta a definição de cada uma das funções, sabe-se que *matlabpool* inicia um agrupamento paralelo de *workers* [51], ao que *spmd* (*single program*, *multiple data*) executa código em paralelo nesses *workers* [62]. Dessa forma, faz sentido mover o *token spmd* para o *concern parallelisation* (*concern* esse onde está localizado o *token matlabpool*) e, assim, sair do seu *concern* atual (nesse caso, do *concern system*). A Figura 4.3 ilustra um exemplo do uso de *matlabpool* e *spmd* em conjunto:

```
  matlabpool('open','local',PTV_Data.parprocessors);
        matlabpool close
        matlabpool('open','local',PTV_Data.parprocessors);
if str2double(PTV_Data.par) && matlabpool('size')>1
        spmd
matlabpool('close')
        matlabpool('open','local',PTV_Data.parprocessors);
            matlabpool close
            matlabpool('open','local',PTV_Data.parprocessors);
if str2double(PTV_Data.par) && matlabpool('size')>1
          spmd
matlabpool close
```

Figura 4.3: Uso das funções *matlabpool* e *spmd* em conjunto no código *MATLAB*

Além do par *matlabpool-spmd*, também os pares *zeros-size*, *ones-size*, *zeros-length* e *ones-length* apresentam valores interessantes de IMP, assim como frequências altas no repositório (ao contrário do par *matlabpool-spmd*). Os *tokens zeros* e *ones* pertencem ao *concern memory allocation/deallocation* e os *tokens size* e *length* pertencem ao *concern data type verification*. O fato de dois *tokens* pertencerem a um *concern* e outros dois *tokens* a outro *concern* poderia motivar a modificação de dois deles para o outro *concern*, juntando assim todos eles. O que acontece é que os *tokens zeros* e *ones* criam matrizes de 0's e 1's, respetivamente, enquanto os *tokens size* e *length* devolvem o tamanho e o maior comprimento de uma matriz/*array*, respetivamente. O uso dessas funções simultaneamente tem como objetivo a criação de matrizes ou *arrays* com um tamanho específico, como mostram os exemplos das Figuras 4.4 e 4.5:

```
Y = zeros (size(A,1), size(B,2), 'single');
for y = 1 : size(A,1)
  for x = 1 : size(B,2)
    for pos = 1 : size(A,2)
```

Figura 4.4: Uso das funções *zeros* e *size* em conjunto no código *MATLAB*

```
magx = mag * ones (1, length(mag));
magy = ones (length(mag),1) * mag;
```

Figura 4.5: Uso das funções *ones* e *length* em conjunto no código *MATLAB*

Os *tokens zeros* e *ones* possuem algumas coocorrências com outros *tokens* do seu *concern*, indicando que, a haver uma mudança de *tokens* nas tabelas, diriam respeito aos *tokens size* e *length*. No entanto, pesquisas realizadas sobre cada um dos quatro *tokens* mostrou que eles são utilizados em vários contextos de código *MATLAB*, em situações que diferem muito daquelas que foram mostradas nas figuras anteriores (sobretudo para os *tokens size* e *length*). Esse fato prova que se devem manter os *tokens* nas suas entradas da tabela, não só pelo seu significado, mas, sobretudo, pelo contexto em que são chamados no código.

Os pares *error-nargchk* e *nargin-error* também apresentam boas coocorrências ao longo do repositório, com frequências de 1005 e 1725, respetivamente. Mais uma vez, esta análise sugere uma boa relação entre esses *tokens*, embora cada um deles possua outras boas relações com os *tokens* pertencentes aos *concerns* onde eles estão inseridos. Estes dois pares são relativamente semelhantes dado que os *tokens nargchk* e *nargin* pertencem ao *concern verification of function arguments and return values* e o *token error* (presente nos dois pares) pertence ao *concern console messages*. O *token error* gera um erro e exibe uma mensagem, sendo que no primeiro caso serve para validar se o número de argumentos de *input* é o correto enquanto no segundo caso é exibida uma mensagem de erro sempre que o número de argumentos testado na condição não é satisfeito. As Figuras 4.6 e 4.7 ilustram cada um dos casos:

```
error (nargchk(3, 3, nargin)) % check number of input args is correct
error ('Invalid ma por topol struct.');
```

Figura 4.6: Uso das funções *error* e *nargchk* em conjunto no código *MATLAB*

```
if nargin > 2
      error ('Too many arguments');
```

Figura 4.7: Uso das funções *nargin* e *error* em conjunto no código *MATLAB*

No entanto, a análise sobre todos os *tokens* ao nível da tabela indicou imensos resultados (nomeadamente a partir da IMP) os quais terão que ser estendidos se se pretende fazer uma análise em maior detalhe acerca das várias relações que existem entre muitos *tokens* da tabela. Os pares salientados nesta subsecção dizem respeito, nomeadamente, a casos que coocorrem demasiadas vezes no código *MATLAB* e, como tal, mostram sinais de interesse quando comparados com outros pares.

4.5. Discussão

O estudo e análise das diversas métricas de associação de *tokens*-palavra permitiu validar a tabela de *tokens*. Inicialmente, descartou-se uma grande quantidade de *tokens* por não apresentarem resultados interessantes sobre cada uma das métricas utilizadas ou, por outro lado, nem sequer aparecerem no repositório que se está a estudar.

A partir do estudo de cada um dos *concerns* foi possível verificar quais aqueles que mantinham melhores relações entre os seus *tokens* e aqueles que não mantinham relações de coocorrência tão fortes. Mesmo estes últimos, muitas vezes, correspondiam a *tokens* que, para o mesmo *concern*, possuem um conceito semelhante no código *MATLAB* fazendo com que nunca possam ser usados simultaneamente. A parte mais interessante deste capítulo passa pela inserção de sub-*concerns*, facilitando a forma como o utilizador identifica um *token* na tabela de *tokens*. A nova estrutura com que alguns *concerns* ficaram serve para “arrumar” melhor os *tokens* e organizar a tabela sem misturar demasiadamente cada conjunto de tokens.

Numa última análise, pretendia-se estudar os *tokens* ao nível dos *concerns* em geral, de forma a verificar eventuais mudanças de *tokens* de posição na tabela. Esse estudo levou à mudança do *token spmd* para o *concern parallelisation*, sem que tivesse havido qualquer outra mudança. Mesmo existindo relações de coocorrência forte entre *tokens* de *concerns* distintos, a verdade é que muitos deles aparecem em contextos diferentes no código e, como tal, nunca sabemos ao certo qual a posição ideal na tabela que cada um desses *tokens* deve tomar.

5. Associação entre *tokens* e palavras reservadas

Para além daquilo que foi analisado no capítulo anterior entre *tokens* e *concerns*, também foi estudada a relação entre *tokens* e palavras reservadas. Essas palavras dizem respeito a palavras que aparecem frequentemente no código *MATLAB*, como é o caso de palavras de condição ou de ciclos, e que podem ter relações interessantes com alguns *tokens* da tabela. A ocorrência de *tokens* com tais palavras e com contagens significativas poderão ser bastante informativos, com a possibilidade de se identificar e criar novos *concerns*.

5.1. Tabela que relaciona *tokens* e palavras reservadas

As contagens das relações entre as mais variadas palavras reservadas *MATLAB* e os diferentes *tokens* da tabela de *tokens* foram realizadas, novamente, com o auxílio da ferramenta *CCCExplorer*. Da mesma forma que se preparou toda a análise para as relações entre *tokens* e *concerns*, também neste caso se optou por uma abordagem similar. Contudo, a única métrica aqui utilizada foi a IMP por apresentar valores bastante elevados no topo da tabela do *output* resultante e, além disso, por estas relações apresentarem imensos resultados o que dificultaria muito a comparação entre as diferentes métricas.

A Tabela 5.1 corresponde ao *output* dos vinte maiores valores da IMP para pares que opõem palavras reservadas de *MATLAB* e *tokens* da tabela de *tokens*. A tabela está ordenada de forma descendente pelo valor da IMP de cada par e a negrito estão destacadas todas as palavras reservadas.

A Tabela 5.1 mostra que, ao longo do repositório, existem contagens bastante interessantes de alguns pares e valores da IMP claramente positivos. Como tal, a secção 5.2 selecciona aqueles pares que apresentam contagens conjuntas muito elevadas no repositório e, por outro lado, alguns pares que se apresentam poucas vezes, mas que tenham uma IMP alta e aparentam, numa primeira visualização, bons indicadores para estarem presentes juntamente no código.

Token e palavra reservada	Contagens do par no repositório	Informação mútua pontual
otherwise - error	2488	7.1
try - matlabpool	6	6.4
evalc **- try**	6	6.3
lastwarn **- switch**	8	6.2
elseif - isfloat	10	5.9
elseif - isnumeric	415	5.6
try - delete	153	5.6
disp **- try**	1001	5.4
catch - disp	875	5.3
if - libisloaded	8	5.2
elseif - iscell	273	5.2
elseif - isinteger	6	5.1
elseif - isa	343	5.0
elseif - isstruct	208	5.0
elseif - islogical	31	5.0
if - mislocked	5	5.0
int32 **- elseif**	53	4.9
elseif - ischar	427	4.9
if - nargin	15196	4.9
tic **- while**	25	4.9

Tabela 5.1: Informação mútua pontual entre *tokens* e palavras reservadas *MATLAB*

5.2. Análise das relações entre *tokens* e palavras reservadas

As palavras reservadas usadas a partir do *MATLAB* não são *tokens* porque aparecem demasiadas vezes no código *MATLAB* e em contextos bem distintos. No entanto, ao relacionarem-se com os *tokens* da tabela de *tokens* podem dar bons indícios acerca da presença de certos *concerns*.

A análise realizada sobre a Tabela 5.1 terá como auxílio exemplos concretos de código *MATLAB* onde esses pares se encontram, de modo a que se entenda melhor o contexto onde eles são utilizados. Além disso, e sempre que necessário, serão colocados na secção Anexos B outros exemplos de código mencionando casos semelhantes que estão a ser analisados.

A contagem no repositório do par *if-nargin* destaca-se facilmente das restantes por se apresentar mais de quinze mil vezes ao longo de todo o repositório. A função *nargin* devolve o número de argumentos de *input* para a função que está em execução [53], enquanto a palavra reservada *if* testa e verifica esse número, impondo algumas restrições consoante aquilo que foi passado. A Figura 5.1 mostra um exemplo de parte de uma função em que se utiliza a palavra reservada *if* e o *token nargin*. Por norma, este par costuma ser apresentado no inicío de cada função, para verificar o que foi recebido como *input*. Neste caso, a função seguinte devolve todos os nomes de ficheiros (*fn*) e diretorias (*dn*) sobre uma diretoria específica (*direct*) e com a extensão pretendida (*opt*). Contudo, a função verifica aquilo que foi recebido como argumento (através da função *nargin*), assume a diretoria raíz (caso não seja especificada nenhuma) e também qualquer extensão se o número de argumentos recebidos for inferior a dois.

```
function [fn,dn] = getfnames(direc,opt)

if (nargin<1 | isempty(direc)),
    direc = '.';
end

if nargin<2 | isempty(opt),
    opt = [];
end
```

Figura 5.1: Utilização da palavra reservada *if* e do *token nargin* em conjunto no código *MATLAB*

A Figura 5.2 mostra um outro exemplo de uma função que verifica se o número de argumentos de *input* é zero e, se for o caso, exibe uma mensagem informando que o primeiro argumento da função é obrigatório.

```
function [pyr,pind] = buildLpyr(im, ht, filt1, filt2, edges)

if (nargin < 1)
  error('First argument (IM) is required');
end
```

Figura 5.2: Utilização da palavra reservada *if* e do *token nargin* em conjunto no código *MATLAB*

A presença deste par em alguns exemplos de código *MATLAB* (mais exemplos em anexos B1) é utilizado com o intuito de se perceber aquilo que foi recebido como *input* na função, correndo-se, então, alguns estados, garantido que os parâmetros sejam devidamente preenchidos, consoante as especificidades de cada função.

O par *otherwise-error* ocupa a primeira posição da tabela com uma frequência conjunta no repositório de 2488 (a segunda mais elevada). Este par aparenta ter uma boa coocorrência ao longo do repositório, apresentando um valor de IMP relativamente interessante. A palavra *otherwise* é opcional e executa um estado apenas quando nenhum outro caso é verdadeiro (fazendo com que seja chamado sempre que os outros casos sejam ignorados) [58]. O *token error*, tal como o próprio nome indica, gera um erro e exibe uma mensagem, informando que nenhum outro caso foi executado [40]. Depois da análise individual de cada palavra percebe-se que *otherwise* e *error* possuem uma relação de proximidade visto que uma mensagem de erro é exibida sempre que nenhum outro caso é executado (fazendo com que *otherwise* o seja). A Figura 5.3 mostra um pedaço de código relativo à construção de um círculo, onde, neste caso, é especificado o tipo de vetor que se pretende: de duas ou três dimensões. Se, por acaso, for definido qualquer outro valor que não seja dois nem três, será exibida uma mensagem de erro informando que o vetor só poderá ser de duas ou três dimensões.

```
switch size(v,2)
 case 2,
  x = xx + v(1); y = yy + v(2);
  z = patch(x,y,c);
 case 3,
  x = xx + v(1); y = yy + v(2) - yy; z = yy + v(3);
  z = patch(x,y,z,c);
 otherwise error('vector must be two or three-dimensional')
```

Figura 5.3: Utilização da palavra reservada *otherwise* e do *token error* em conjunto no código *MATLAB*

Verificou-se que o par *otherwise-error* aparece, na maior parte dos casos, relacionado no final de um estado *switch* (mais exemplos em anexos B2), exibindo uma mensagem de erro caso os valores de *case* não correspondam a nenhum daqueles que foi dado como *input*.

Ainda que existam pares com contagens aparentemente desinteressantes na Tabela 5.1, importa perceber, através de exemplos de código *MATLAB*, porque estes pares aparecem destacados nos vinte melhores resultados desta métrica. Assim, o par *try-matlabpool* merece ser destacado, porque, apesar de aparecer apenas seis vezes ao longo de todo o repositório, mostra um valor relativamente elevado para a IMP. Esse par permite explorar as características da programação paralela em *MATLAB*, tentando executá-lo num número diferente de máquinas. A Figura 5.4 pretende inicializar um conjunto de *workers* que irão correr código *MATLAB*:

```
try
     matlabpool('open','local',PTV_Data.parprocessors);
catch
     try
          matlabpool close
          matlabpool('open','local',PTV_Data.parprocessors);
```

Figura 5.4: Utilização da palavra reservada *try* e do *token matlabpool* em conjunto no código *MATLAB*

O valor da IMP é bastante elevado porque *matlabpool* aparece poucas vezes ao longo do repositório, mas sempre que aparece, está frequentemente relacionado com a palavra reservada *try*.

A relação *try-delete*, com uma frequência conjunta de 153 no repositório, pretende apagar ficheiros ou objetos [38]. Tal como foi especificado anteriormente, a palavra *try* tenta fazer a remoção de tais elementos sem que haja erros e, caso hajam, serão executadas as instruções do bloco *catch*. Este par apenas apaga ficheiros e objetos sendo que a palavra *try* apenas é usada antevendo que algo possa correr mal e, então, terá que se executar alguns comandos caso isso aconteça. Podemos notar que a Tabela 5.1 apresenta outro par que facilmente se relaciona com este: *catch-disp*. Este par, com uma frequência de 875 no repositório de *tokens*, exibe uma mensagem sempre que as instruções em *try* não são satisfeitas [39].

A Figura 5.5 mostra um exemplo de uma função *MATLAB* que pretende remover um objeto x, com recurso às palavras *try* e *delete*. A utilização da palavra reservada *try* juntamente com o *token delete* tem como objetivo apagar esse objeto, embora seja desencadeada uma mensagem de alerta se não se conseguir proceder à remoção de tal objeto (a partir da utilização de *catch* com o *token disp*).

```
function remove_found(x)
        try
            delete([x]);
        catch
            disp([x ' could not be deleted'])
        end
    end
```

Figura 5.5: Utilização das palavras reservadas *try* e *catch* e dos *tokens* *delete* e *disp* em conjunto no código *MATLAB*

A utilização destes dois pares no código *MATLAB* é muitas vezes encontrado em situações que se assemelham à figura anterior. Algumas pesquisas sobre o código *MATLAB* mostrou que as palavras reservadas *try* e *catch*, juntamente com o *token disp* são bastante usuais. O que se verifica é que, ao invés de se utilizar a função *delete*, recorre-se a outro tipo de funções consoante as necessidades do utilizador. A Figura 5.6 mostra mais um caso recorrente do exemplo anterior, com o uso do *token print* no lugar de *delete*:

```
disp('saving plot of density as a function of position
with N variable in png and pdf format')

try
    print('-dpng',[seq_name,'_N_variable.png'])
    print('-dpdf',[seq_name,'_N_variable.pdf'])
catch
    disp(lasterr)
end
```

Figura 5.6: Utilização das palavras reservadas *try* e *catch* e dos *tokens print* e *disp* em conjunto no código *MATLAB*

Verifica-se que muitos dos pares presentes na Tabela 5.1 dizem respeito a pares que opõem palavras de condição (*if*, *elseif* ou *else*) com nomes de funções (neste caso, *tokens*) que se iniciem com '*is'*. A presença desses pares na Tabela 5.1 é utilizada em casos que se pretende testar cada uma das funções que se seguem à palavra reservada de condição. Quando acontece o oposto (ou seja, a palavra reservada de condição aparece depois do *token*) torna-se mais complicado prever o porquê desses pares aparecerem na Tabela 5.1.

Os pares *elseif-isfloat* e *elseif-isnumeric* fazem a avaliação de cada uma das funções: *isfloat* e *isnumeric* (com frequências de 10 e 415, respetivamente). Estes dois pares possuem uma IMP relativamente alta dada a sua presença no topo da Tabela 5.1. De fato, a palavra *elseif*, tendo em conta a análise que está a ser feita, poderia ser manipulada no repositório para *if*, visto que não existe qualquer diferença. O mesmo se passa com os casos *for* e *while*, em que ambos têm a mesma definição e são tratados como ciclos. Tal como indica, a palavra reservada *elseif* testa uma expressão e executa um grupo de estados se ela for verdadeira. Já as funções *isfloat* e *isnumeric* determinam se o *input* é uma matriz de ponto flutuante ou numérica, respetivamente. Logicamente que estas duas relações são muito próximas visto que o objetivo do uso de *elseif* é testar cada uma das funções.

As Figuras 5.7 e 5.8 ilustram dois casos do uso de *elseif* com as funções *isfloat* e *isnumeric*, respetivamente:

```
elseif isfloat(str) || isdouble(str)
    % Floating point, so should be a triple in range 0 to 1
    if numel(str)==3
        col = double( str );
        col(col<0) = 0;
        col(col>1) = 1;
    else
        error( 'UIExtras:interpretColor:BadColor', 'Could not interpret
color %s', num2str( str ) );
    end
```

Figura 5.7: Utilização da palavra reservada *elseif* e do *token isfloat* em conjunto no código *MATLAB*

```
if ischar(values), values = cellstr(values);
elseif isnumeric(values), values = num2cell(values);
end
[nrow,ncol] = size(values);
```

Figura 5.8: Utilização da palavra reservada *elseif* e do *token isnumeric* em conjunto no código *MATLAB*

O par *if-libisloaded* apresenta uma frequência conjunta baixa (8) no repositório onde a palavra *if*, como já referido, testa uma expressão. Por outro lado, a função a testar (*libisloaded*) devolve o valor 1 (verdadeiro) se a biblioteca C partilhada está carregada e 0 (falso) caso contrário [50]. Esta relação, ainda que com uma frequência baixa, faz sentido porque testa uma função *MATLAB*. A Figura 5.9 mostra um exemplo em código desse par:

```
if libisloaded('CFS32')==0
    % Switch off R2008a warning for now
    % TODO: Need to generate new cfs.m for future release
    warning('off', 'MATLAB:loadlibrary:OldStyleMfile');
    loadlibrary('CFS32.DLL',@cfs);
    warning('on', 'MATLAB:loadlibrary:OldStyleMfile');
end
```

Figura 5.9: Utilização da palavra reservada *if* e do *token libisloaded* em conjunto no código *MATLAB*

Por último, também o par *tic-while* se mostra interessante nesta análise (com uma frequência de 25 no repositório). A função *tic* inicia um cronómetro para medir o desempenho de uma função, o qual é exibido assim que a função *toc* é chamada [63]. *While* corresponde a um ciclo que executa um grupo de estados enquanto a expressão é verdadeira sendo que para este caso é utilizado como um contador de tempo entre o início do cronómetro e o seu final. Como é natural, o par *while-toc* também aparece no repositório e tem uma IMP de 4.8 (embora a Tabela 5.1 não o apresente por corresponder apenas a uma parte da totalidade da tabela), um valor bastante significativo tendo em conta os valores que são apresentados na Tabela 5.1. A Figura 5.10 ilustra a utilização dos *tokens tic* e *toc* para medir o desempenho de uma função juntamente com a palavra reservada *while*. Para manter a uniformidade, a cor azul corresponde aos *tokens* e a cor amarela à palavra reservada.

```
function img = cam_wait(T,BB)
% Runs cam for T sec. and returns the last frame.

global vid;
tic;
while toc < T
    img = img_alloc(getsnapshot(vid));
    imshow(img.in);
    if exist('BB','var')
        bb_draw(BB,'EdgeColor','y','LineWidth',3);
        %title(toc);
        drawnow;
    end
end
```

Figura 5.10: Utilização da palavra reservada *while* e dos *tokens tic* e *toc* em conjunto no código *MATLAB*

A Tabela 5.1 mostrou as vinte melhores relações entre algumas palavras reservadas de *MATLAB* e os seus *tokens*, embora essa tabela seja muito mais extensa e com outras relações que se possam mostrar também elas interessantes para a análise que se tem vindo a realizar.

5.3. Discussão

O estudo e análise de pares que relacionam *tokens* e palavras reservadas *MATLAB* conseguiu identificar um número bastante elevado de pares e com valores de IMP relativamente altos. Contudo, notou-se, a partir da Tabela 5.1, que existiam muitos pares com contagens baixas no repositório, algo que não sustentava alguns dos pares encontrados. Outro dos problemas foi a ordem pelo qual alguns *tokens* se encontravam, o que dificultava fazer uma análise viável relativamente à forma como eles aparecem no código. Assim, não existiu a possibilidade de se modificar a Tabela 2.2 da página 20 e, consequentemente, de inserir novos *concerns* na tabela de *tokens*.

Ainda assim, este capítulo permitiu analisar um número interessante de pares que aparecem frequentemente no código e que correspondem a padrões sintáticos muito bem identificados em sistemas *MATLAB* (como o caso de *if-nargin*). Esta questão dos padrões sintáticos pode ser devidamente estudada e explorada através da ferramenta *LARA* (ver secção 2.4.4), que parece adequar-se de forma perfeita a estes conceitos. Este tipo de padrões são devidamente destacados e explorados nas teses em realização do João Barrulas [1] e do António Relvas [25], podendo vir a contribuir para uma melhor modularidade dos *concerns* em *MATLAB*.

6. Conclusões e trabalho futuro

6.1. Síntese

A presente tese tinha como principal objetivo fazer a validação e melhoria da tabela de *concerns* e *tokens*, a qual permite identificar *concerns* modularizados e não-modularizados no código *MATLAB*. A validação da tabela foi realizada com recurso a métodos da linguística computacional, com especial destaque para a informação mútua pontual. Essa métrica permitiu verificar que existiam muitos *tokens* da tabela que podiam ser descartados por não possuírem contagens suficientemente interessantes para serem usadas no cálculo de cada uma das métricas utilizadas. O estudo efetuado conseguiu clarificar muitas relações da tabela de *tokens* e validar o posicionamento de muitos *tokens* da tabela. O trabalho efetuado não gerou novos *concerns* nem novos *tokens* que pudessem ser inseridos na tabela de *tokens*, embora ainda haja espaço em trabalho futuro para continuar a análise sobre os resultados obtidos, nomeadamente, podendo permitir eventuais inserções. Além disso, a nova estrutura da tabela (ver Tabela 6.1) pode facilitar essa inclusão dada a forma como os *tokens* se encontram posicionados.

À medida que o trabalho foi sendo desenvolvido percebeu-se que as métricas apresentavam resultados interessantes e, como tal, optou-se por estudar os *tokens* numa outra vertente: a sua relação com palavras reservadas *MATLAB*. Contudo, esse estudo não alterou qualquer característica da tabela de *tokens*, mas conseguiu identificar novos padrões de código *MATLAB* que, explorados numa fase posterior a esta tese, poderão contribuir para o aperfeiçoamento da técnica de deteção de *concerns*.

6.2. Resultados

Todo o trabalho desenvolvido ao longo desta tese tinha como objetivo gerar uma nova versão da tabela de *tokens* (a versão mais recente encontra-se na Tabela 2.2 da página 20). Com base nas várias métricas de associação de *tokens*-palavra, nomeadamente a informação mútua pontual, foi, sobretudo, possível validá-la, mas também melhorá-la. Contudo, a melhoria da tabela foi mais visível em relação à inserção de sub-*concerns* e à ordenação alfabética dos *tokens*, que pretendem facilitar a forma como o utilizador localiza um *token* e quais as suas características no código *MATLAB*. Ainda que essas características não definam completamente aquilo que é o papel de um *token* no código, a criação de sub-*concerns* parece evitar a abstração que existia em outras versões da tabela, onde se notava uma grande “distância” entre o *concern* e os seus *tokens*.

A Tabela 6.1 ilustra a versão resultante do trabalho desenvolvido ao longo desta tese, mantendo-se a alternância de cores claras entre os vários *concerns* e conjunto de *tokens*, de forma a minorar o esforço aquando da sua visualização.

Concern	Sub-concerns		Tokens
Verification of function arguments and return values			nargchk, nargin, nargout, nargoutchk, varargin, varargout
Data type	**Specialisation**		double, fi, int8, int16, int32, int64, quantize, quantizer, single, uint8, uint16, uint32, uint64
	Verification	Identification and numeric types	cast, class, intmax, intmin, isa, isboolean, iscell, ischar, isfloat, isinf, isinteger, islogical, isnan, isnumeric, isobject, isreal, isstr, isstruct, realmax, realmin, typecast
		Matrices and arrays	isempty, isfield, isrow, isscalar, isvector, length, ndims, numel, range, size
Dynamic properties			eval, evalc, evalin, feval, inline
Console messages			annotation, assert, disp, display, error, last, lastwarn
Printing			orient, print, printdlg
Visualisation	**2D and 3D plots**	Animation	getframe, movie
		Contour plots	clabel
		Data distribution plots	hist, histogram, scatter
		Line plots	errorbar, fplot, gplot, loglog, plot, plot3, semilogx, semilogy
		Polar plots	polar
		Surfaces, volumes and polygons	mesh, meshgrid, surf
	Formatting and Annotation	Axes appearance	axis, box, datetick, figure, grid, plotyy, subplot
		Titles and labels	legend, line, rectangle, text, title, xlabel, ylabel, zlabel
	Graphic Objects	Identification	gca, gcbf, gcbo, gco
		Properties	reset, set
		Programming/output	cla, clf, close, hold, ishold, newplot
	Images		frame2im, image, iminfo, imread, imwrite
	Visual exploration		datacursormode, pan, rotate, rotate3d, zoom
File I/O			diary, fgetl, fgets, fopen, fprintf, fread, fscanf, fwrite, hgload, hgsave, load, save, saveas, uisave
System	**Code analysis**		run, timerfind
	Control flow		break, next, pause, wait, xbreak
	Data import and export		clear, who, whos
	Dates and time		addtodate, clock, date, etime, now, step
	Debugging		dbstop, echo
	Entering commands		ans, slist, stop
	Files and folders		exist, rehash
	Functions		inputname, isvarname, mfilename, mislocked, mlock, symvar
	Performance and memory		cputime, memory, pack, profile, tic, toc
	Scripts		batch, input
	Startup		start
	Using external libraries		calllib, libisloaded, loadlibrary, mex, mexext, unloadlibrary
Memory allocation/deallocation			delete, global, ones, persistent, zeros
Parallelisation			cancel, codistributed, codistributor, gather, labindex, labProbe, matlabpool, numlabs, parfor, pload, pmode, promote, resume, sparse, spmd, subsasgn, subsref

Tabela 6.1: Versão mais recente da tabela de *tokens* devidamente validada e melhorada

O estudo e a análise de várias métricas de associação de palavras (ver secções 2.4.1 e 2.4.2) permitiram, essencialmente, validar a posição de cada um dos *tokens* da tabela. A única alteração de posição de *tokens* realizada sobre a tabela deveu-se ao estudo das coocorrências entre *tokens* de *concerns* distintos (ver secção 4.4.2) e resultou na mudança do *token spmd* do *concern system* para o *concern parallelisation*. Essa alteração não foi feita apenas com base no valor de cada uma das métricas, mas também com base nas contagens individuais de cada *token* no repositório de *tokens* e, sobretudo, pela definição de cada uma dessas funções no código *MATLAB*.

A Tabela 6.1 foi o resultado de toda a análise e devidas justificações realizadas no capítulo 4. O que mais se destaca nesta nova versão é a redução no número de *tokens* que consta da tabela e a inserção de sub-*concerns*. A redução de *tokens* pode ser analisada em maior detalhe na secção 4.3, onde foram descartados *tokens* que tinham contagens inferiores a seis ao longo de todo o repositório. A inserção de sub-*concerns* teve origem na análise individual de cada um dos onze *concerns* (ver secção 4.4.1) e serve para estruturar melhor a posição de cada *token* ao longo da tabela. A versão da Tabela 2.2 da página 20 dificultava, por vezes, a procura de um *token* específico por parte de um utilizador, devido à grande quantidade de *tokens* por *concern* (algo que os sub-*concerns* vieram ajudar a resolver). No entanto, a criação de sub-*concerns* dificultou, por vezes, o local onde se deveria inserir certo *token*. Dessa maneira, existiram três *tokens* pertencentes ao *concern system* (*ba*, *where* e *systems*) que foram retirados da tabela por não apresentarem uma posição indicada para estarem. Outra melhoria efetuada na nova versão da tabela diz respeito ao fato dos *tokens* aparecerem ordenados alfabeticamente por *concern* e sub-*concern*.

O estudo da relação de coocorrência entre *tokens* e palavras reservadas *MATLAB* (ver capítulo 5) tinha como objetivo a inserção de novos *concerns* na nova versão da tabela. Contudo, essa análise mostrou pares interessantes no código *MATLAB*, mas não foi possível inserir novos *concerns* com base nesses resultados.

6.3. Contribuições

A execução e análise de diferentes métricas da linguística computacional (informação mútua pontual, *cosseno*, *dice* e *SCP*) permitiram validar e melhorar a tabela de *tokens* e, assim, realizar as seguintes contribuições:

- Utilização de diversas métricas de associação de palavras (ver secções 2.4.1 e 2.4.2) onde se permitiu concluir que a IMP é preferível relativamente às funções *cosseno*, *dice* e *SCP* (*Symmetrical Conditional Probability*). A IMP ignora resultados fracos e que não contribuem para a validação e melhoria da tabela de *tokens* enquanto as restantes métricas obtinham resultados muito baixos quando deveriam ignorá-los;

- Desenvolvimento de um método para análise e comparação entre várias métricas da linguística computacional (sobre um repositório de *tokens*-palavra) que permitiu validar e refinar a tabela de *tokens* na identificação de *concerns* ao longo do código *MATLAB* (ver secção 3.3);

- Remoção de *tokens* que tinham poucas contagens no repositório utilizado e, como tal, não forneciam resultados interessantes para análise (ver secção 4.3);

- As coocorrências entre os *tokens* da tabela de *tokens* permitiram identificar quais os *tokens* que estavam nas posições corretas da tabela e aqueles que eventualmente poderiam ser movidos, ajudando, assim, a agrupar os *tokens* certos para a identificação de *concerns* (ver secção 4.4);

- Validação das posições dos *tokens* relativamente aos seus *concerns* (ver secção 4.4.1);

- Alteração da posição de *tokens* da tabela por apresentarem melhores relações de coocorrência com outros *concerns* do que aqueles onde estavam inseridos (ver secção 4.4.2).

6.4. Trabalho futuro

Tendo o trabalho desenvolvido tido sucesso na realização duma validação e refinamento da tabela de *tokens*, abrem-se novas oportunidades de trabalho futuro, nomeadamente:

- Identificar novos *tokens* que aparecem frequentemente no código *MATLAB,* mas que não aparecem na tabela de *tokens*. Esta melhoria pode ser realizada através de uma contagem individual de todos os *tokens* do repositório ordenados de forma descendente e, a partir daí, verificar aqueles que aparecem em maior destaque (e que não estão na tabela de *tokens*). Depois disso, aplicar a informação mútua pontual entre os *tokens* encontrados e toda a tabela de *tokens*, verificando quais os *tokens* que melhor se relacionam com os *tokens* encontrados;

- Uma das primeiras preocupações que deverá ser tida em consideração como seguimento deste trabalho será o repositório de *tokens* a utilizar. A grande limitação desta tese deveu-se, sobretudo, ao código *MATLAB* disponível, que, apesar dos seus 35 mil *m-files*, não consegue abranger todos os *tokens* da tabela;

- Com base no ponto anterior, torna-se importante estudar os *tokens* da tabela que não foram estudados neste trabalho. Além disso, uma grande extensão do repositório de

tokens poderia permitir obter melhores resultados (mesmo sobre as ocorrências e coocorrências dos *tokens* estudados) visto que o número de vezes que um *token* apareceria no repositório era maior;

- Aplicar as métricas de associação de palavras ao nível de um *m-file* e verificar as relações entre os *tokens* para o mesmo *m-file*. Contrariamente ao que foi feito neste trabalho, em que os *tokens* foram estudados independentemente do *m-file* a que pertenciam;

- Aplicar métricas de associação de palavras sobre *tokens* que não apareçam consecutivamente no repositório (com intervalos de um e dois *tokens* entre eles), comparando com os resultados que foram obtidos neste trabalho ou utilizando uma base de dados relacional contendo o repositório [25];

- Estudar e focar nos sub-*concerns* criados por ser algo recente implementado nesta tese e poder ser ainda melhorado, com a mudança de *tokens* entre sub-*concerns* do mesmo *concern*;

- Utilizar a ferramenta *LARA* para estudar estruturas mais elaboradas, ou seja, padrões sintáticos de *MATLAB* que vão para além das ocorrências e coocorrências de *tokens* e que permitam a deteção de *concerns* a partir desses mesmos padrões.

Bibliografia

[1] J. Barrulas. “Code Patterns for Concern Detection in MATLAB Systems”. MSc preparation report, Universidade Nova de Lisboa, 2019.

[2] J. Bispo e J. Cardoso. “A MATLAB subset to c compiler targeting embedded systems.” *Software: Practice and Experience* 47(2), 2017.

[3] G. Bouma. “Normalized (pointwise) mutual information in collocation extraction”. *Proceedings of GSCL*, 2009, p. 31-40.

[4] M. Bruntink et al. “An evaluation of clone detection techniques for crosscutting concerns.” Em: *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*. IEEE, 2004. p. 200-209.

[5] J. Cardoso, J. Fernandes, M. Monteiro, T. Carvalho e R. Nobre. “Enriching MATLAB with aspect-oriented features for developing embedded systems.” Em: *Journal of Systems Architecture* 59.7 (2013), p. 412-428.

[6] K. Church e P. Hanks. "Word association norms, mutual information, and lexicography." *Computational linguistics*. 1990. p. 22-29.

[7] A. Diniz. “Engenharia de Software”. *Termos Técnicos*, p. 119-120.

[8] K. Duarte. “Limitations in the Support to Modularity in MATLAB: a Survey-based Empirical Study”. MSc Thesis, Universidade Nova de Lisboa, 2017.

[9] K. Duarte e R. Falbo. “Uma ontologia de qualidade de software.” Em: *Workshop de Qualidade de Software, João Pessoa*. 2000. p. 275-285.

[10] S. Ducasse, T. Girba e A. Kuhn. “Distribution map.” Em: *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on*. IEEE, 2006. p. 203-212.

[11] E. Figueiredo. “Concern-oriented heuristic assessment of design stability”. PhD Thesis, Lancaster University, 2009.

[12] J. Frain et al. “MATLAB for Economics and Econometrics A Beginners Guide.” *Trinity College Dublin, Department of Economics*, 2014.

[13] A. Kellens, K. Mens e P. Tonella. “A survey of automated code-level aspect mining techniques.” Em: *Transactions on aspect-oriented software development IV*. Springer, Berlin, Heidelberg, 2007. p. 143-162.

[14] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier e J. Irwin. “Aspect-oriented programming.” Em: *European conference on object-oriented programming*. Springer, Berlin, Heidelberg, 1997. p. 220-242.

[15] E. Lopez-Herrejon e S. Apel. “Measuring and characterizing crosscutting in aspect-based programs: Basic metrics and case studies.” Em: *International Conference on Fundamental Approaches to Software Engineering*. Springer, Berlin, Heidelberg, 2007. p. 423-437.

[16] T. Mikolov, I. Sutskever, K. Chen, G. Corrado e J. Dean. “Distributed representations of words and phrases and their compositionality”. Em: *Advances in neural information processing systems*. 2013. p. 3111-3119.

[17] M. Monteiro et al. "Toward a Token-Based Approach to Concern Detection in MATLAB Sources." Em: *Portuguese Conference on Artificial Intelligence*. Springer, Cham, (EPIA). 2017.

[18] M. Monteiro, J. Cardoso e J. Fernandes. “Adding aspect-oriented features to MATLAB.” Em: *Workshop on Software Engineering Properties of Languages and Aspect Technologies* (SPLAT). 2006. p. 573-584.

[19] M. Monteiro, J. Cardoso e S. Posea. “Identification and characterization of crosscuting softwares in MATLAB systems.” Em: *Conference on Compilers, Programming Languages, Related Technologies and Applications* (CoRTA 2010), Braga, Portugal, 2010. p. 9-10.

[20] M. Monteiro e J. Fernandes. “Aspect-oriented refactoring of Java programs.” Java in Academia and Research, 2011. p. 141-158.

[21] K. Murphy et al. “The bayes net toolbox for MATLAB.” *Computing science and statistics*, 2001. 33.2: p. 1024-1034.

[22] B. Palma. “A Tool for Mining Softwares in MATLAB Code Repositories”. MSc Thesis, Universidade Nova de Lisboa, 2017.

[23] P. Pinto et al. “Aspect composition for multiple target languages using LARA.” *Computer Languages, Systems & Structures*, 2018, 53: p. 1-26.

[24] S. Radpour, L. Hendren e M. Schäfer. “Refactoring MATLAB.” Em: *International Conference on Compiler Construction.* Springer, Berlin, Heidelberg, 2013. p. 224-243.

[25] A. Relvas. “Uma Interface Web para Comparação de Métricas Utilizando Mapas Auto-Organizados”. MSc preparation report, Universidade Nova de Lisboa, 2018.

[26] D. Rumelhart, G. Hintont e R. Williams. “Learning representations by backpropagating errors.” *Nature*, 323(6088): p. 533-536, 1986.

[27] C. Sant’Anna et al. “On the reuse and maintenance of aspect-oriented software: An assessment framework.” Em: *Proceedings of Brazilian symposium on software engineering.* 2003. p. 19-34.

[28] C. Shannon e W. Weaver. "The mathematical theory of communication." *Urbana, IL: University of Illinois Press.* 1949.

[29] D. Shepherd et al. “Using natural language program analysis to locate and understand action-oriented concerns.” Em: *Proceedings of the 6th international conference on Aspect-oriented software development*. ACM, 2007. p. 212-224.

[30] J. Silva e G. Pereira. “A local maxima method and a fair dispersion normalization for extracting multi-word units from corpora.” Em: *Sixth Meeting on Mathematics of Language*. 1999. p. 369-381.

[31] J. Stone. “Information theory: a tutorial introduction.” *Sebtel Press*, 2015.

[32] P. Tarr, H. Ossher, W. Harrison e S. Sutton. “N degrees of separation: multi-dimensional separation of concerns.” Em: *Software Engineering. Proceedings of the 1999 International Conference on*. IEEE. p. 107-119.

[33] W. Wong, S. Gokhale e R. Horgan. “Quantifying the closeness between program components and features.” Em: *Journal of Systems and Software*, 2000, 54.2: p. 87-98.

[34] MATLAB - Array Comparison with Relational Operators https://www.mathworks.com/help/matlab/matlab_prog/array-comparison-with-relational-operators.html. Último acesso: 28-01-2019.

[35] MATLAB - Array vs. Matrix Operations https://www.mathworks.com/help/matlab/matlab_prog/array-vs-matrix-operations.html. Último acesso: 28-01-2019.

[36] MATLAB - Cell Arrays
http://www.mathworks.com/help/*MATLAB*/cell-arrays.html. Último acesso: 17-12-2018.

[37] MATLAB - Create and Edit Variables
http://www.mathworks.com/help/*MATLAB*/*MATLAB*_env/create-and-edit-variables.html. Último acesso: 17-12-2018.

[38] MATLAB - Delete function
https://www.mathworks.com/help/matlab/ref/delete.html. Último acesso: 10-02-2019.

[39] MATLAB - Disp function
https://www.mathworks.com/help/matlab/ref/disp.html. Último acesso: 26-02-2019.

[40] MATLAB - Error function
https://www.mathworks.com/help/matlab/ref/error.html. Último acesso: 26-02-2019.

[41] MATLAB - Eval function
https://www.mathworks.com/help/matlab/ref/eval.html. Último acesso: 07-02-2019.

[42] MATLAB - Evalc function
https://www.mathworks.com/help/matlab/ref/evalc.html. Último acesso: 07-02-2019.

[43] MATLAB - Fget function
https://www.mathworks.com/help/matlab/ref/fgetl.html. Último acesso: 07-02-2019.

[44] MATLAB - Fgets function
https://www.mathworks.com/help/matlab/ref/fgets.html. Último acesso: 07-02-2019.

[45] MATLAB - Function
http://www.mathworks.com/help/*MATLAB*/ref/function.html. Último acesso: 17-12-2018.

[46] MATLAB - Global variable
https://www.mathworks.com/help/matlab/ref/global.html. Último acesso: 08-02-2019.

[47] MATLAB - Function library
https://www.mathworks.com/help/matlab/referencelist.html?type=function&category=index. Último acesso: 11-03-2019.

[48] MATLAB - Last function
https://www.mathworks.com/help/matlab/ref/mexception.last.html. Último acesso: 08-02-2019.

[49] MATLAB - Lastwarn function
https://www.mathworks.com/help/matlab/ref/lastwarn.html. Último acesso: 08-02-2019.

[50] MATLAB - Libisloaded function
https://www.mathworks.com/help/matlab/ref/libisloaded.html. Último acesso: 11-03-2019.

[51] MATLAB - Matlabpool function
https://www.mathworks.com/matlabcentral/answers/128607-matlabpool-open-n-doesn-t-work.
Último acesso: 09-03-2019.

[52] MATLAB - Nargchk function
https://www.mathworks.com/help/matlab/ref/nargchk.html. Último acesso: 12-02-2019.

[53] MATLAB - Nargin function
https://www.mathworks.com/help/matlab/ref/nargin.html. Último acesso: 12-02-2019.

[54] MATLAB - Nargout function
https://www.mathworks.com/help/matlab/ref/nargout.html. Último acesso: 12-02-2019.

[55] MATLAB - Nargoutchk function
https://www.mathworks.com/help/matlab/ref/nargoutchk.html. Último acesso: 12-02-2019.

[56] MATLAB - Ones function
http://www.mathworks.com/help/*MATLAB*/ref/ones.html?searchHighlight=ones&s_tid=doc_srchtitle. Último acesso: 24-02-2019.

[57] MATLAB - Operators and Elementary Operations
https://www.mathworks.com/help/matlab/operators-and-elementary-operations.html.
Último acesso: 28-01-2019.

[58] MATLAB - Otherwise
https://www.mathworks.com/help/matlab/ref/switch.html. Último acesso: 04-03-2019.

[59] MATLAB - Persistent variable
https://www.mathworks.com/help/matlab/ref/persistent.html. Último acesso: 08-02-2019.

[60] MATLAB - Plot function
http://www.mathworks.com/help/*MATLAB*/ref/plot.html. Último acesso: 07-10-2018.

[61] MATLAB - Scripts vs Functions
http://www.mathworks.com/help/*MATLAB*/*MATLAB*_prog/scripts-and-functions.html.
Último acesso: 25-01-2018.

[62] MATLAB - Spmd function
https://www.mathworks.com/help/distcomp/spmd.html. Último acesso: 25-02-2019.

[63] MATLAB - Tic function
https://www.mathworks.com/help/matlab/ref/tic.html. Último acesso: 11-03-2019.

[64] MATLAB - Varargin
https://www.mathworks.com/help/matlab/ref/varargin.html. Último acesso: 12-02-2019.

[65] MATLAB - Varargout
https://www.mathworks.com/help/matlab/ref/varargout.html. Último acesso: 12-02-2019.

[66] MATLAB - Zeros function
http://www.mathworks.com/help/*MATLAB*/ref/zeros.html?searchHighlight=zeros&s_tid=doc_srchtitle. Último acesso: 24-02-2019.

[67] Wikipédia - Abstract Syntax Tree
https://en.wikipedia.org/wiki/Abstract_syntax_tree. Último acesso: 23-03-2019.

[68] Wikipédia - Separation of concerns
https://en.wikipedia.org/wiki/Separation_of_concerns. Último acesso: 09-08-2018.

Anexos

Anexo A - Resultados obtidos através de métricas de associação de palavras, como a IMP e as funções *cosseno*, *dice* e *SCP*. A cor verde corresponde a pares de *tokens* que se encontram mais de dez vezes no repositório de *tokens* enquanto a cor vermelha indica pares de *tokens* que se encontram menos de dez vezes no repositório.

A1

Tokens	Informacao Mutua Pontual
[nargoutchk___nargout]	10.424176
[nargchk___nargin]	8.2273586
[nargout___varargout]	7.7303415
[nargchk___nargout]	7.0998024
[varargout___nargout]	6.6234263
[varargin___nargoutchk]	2.9784988
[varargin___varargout]	2.5842855
[nargin___nargout]	2.3436673
[nargin___varargout]	1.8758297
[nargin___varargin]	0.4617842
[nargout___nargin]	0.0459868

Tabela A1.1: Informação mútua pontual para o concern verification of function arguments and return values (Tabela total)

Tokens	Cosseno
[nargoutchk___nargout]	0.269577
[nargchk___nargin]	0.189137
[nargout___varargout]	0.180696
[varargout___nargout]	0.083894
[nargchk___nargout]	0.041765
[varargin___varargout]	0.014087
[nargin___nargout]	0.006578
[nargin___varargout]	0.006472
[nargin___varargin]	0.004927
[varargin___nargoutchk]	0.004268

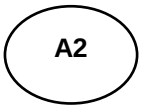
Tabela A1.2: Função *cosseno* para o *concern verification of function arguments and return values* (Tabela parcial)

Tokens	Dice
[nargout___varargout]	0.172447325
[nargoutchk___nargout]	0.154007994
[nargchk___nargin]	0.084221259
[varargout___nargout]	0.080064829
[nargchk___nargout]	0.032878759
[varargin___varargout]	0.011173729
[nargin___varargout]	0.005939403
[nargin___nargout]	0.00514924
[nargin___varargin]	0.004731891
[varargin___nargin]	0.001667797
[varargout___varargin]	0.001219839
[nargout___nargin]	0.001047303

Tabela A1.3: Função *dice* para o *concern verification of function arguments and return values* (Tabela parcial)

Tokens	SCP
[nargoutchk___nargout]	0.072671759
[nargchk___nargin]	0.03577311
[nargout___varargout]	0.032651355
[varargout___nargout]	0.007038366
[nargchk___nargout]	0.001744339
[varargin___varargout]	1.98462E-4
[nargin___nargout]	4.3277E-5
[nargin___varargout]	4.1898E-5
[nargin___varargin]	2.428E-5
[varargin___nargoutchk]	1.8223E-5
[varargin___nargin]	3.016E-6
[varargout___varargin]	2.365E-6
[nargout___nargin]	1.79E-6

Tabela A1.4: Função *SCP* para o *concern verification of function arguments and return values* (Tabela parcial)



Tokens	Cosseno
[uint32___uint64]	0.04529
[double___int16]	0.020893
[double___uint64]	0.013293
[double___int32]	0.009045
[uint32___double]	0.008477
[uint8___double]	0.006823
[uint8___single]	0.005704
[int16___int32]	0.005523
[single___uint32]	0.004429
[int16___double]	0.004178

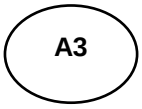
Tabela A2.1: Função *cosseno* para o *concern data type specialisation* (Tabela parcial)

Tokens	Dice
[uint32___uint64]	0.031152647
[double___int16]	0.011428571
[double___int32]	0.005986394
[uint8___single]	0.005698005
[int16___int32]	0.005369127
[uint8___double]	0.005150656
[uint32___double]	0.004581901
[single___uint32]	0.003948667
[int32___single]	0.003344481
[double___uint64]	0.003068425
[uint32___int32]	0.002713704

Tabela A2.2: Função *dice* para o *concern data type specialisation* (Tabela parcial)

Tokens	SCP
[uint32___uint64]	0.002051197
[double___int16]	4.3655E-4
[double___uint64]	1.76728E-4
[double___int32]	8.1817E-5
[uint32___double]	7.1865E-5
[uint8___double]	4.6563E-5
[uint8___single]	3.2543E-5
[int16___int32]	3.0511E-5
[single___uint32]	1.962E-5
[int16___double]	1.7462E-5
[int32___single]	1.1814E-5
[uint32___int32]	7.848E-6

Tabela A2.3: Função *SCP* para o *concern data type specialisation* (Tabela parcial)



Tokens	Informacao Mutua Pontual
[intmax___class]	8.8406305
[cast___class]	7.9781341
[isstr___range]	5.1215967
[isempty___class]	1.085743
[length___class]	1.0660764
[ndims___size]	0.8822752
[isinf___size]	0.8343157
[length___range]	0.8209738
[ndims___length]	0.7592406
[size___class]	0.3926444
[size___range]	0.3914673
[range___length]	0.1429019

Tabela A3.1: Informação mútua pontual para o *concern data type verification* (Tabela total)

Tokens	Cosseno
[isboolean___islogical]	0.047062
[intmin___class]	0.022179
[intmax___class]	0.0182
[cast___class]	0.013497
[isstr___range]	0.007092
[length___size]	0.003666
[length___class]	0.0033
[isempty___class]	0.002769
[length___range]	0.002475
[size___class]	0.001898
[ndims___length]	0.001713
[ndims___size]	0.001711
[size___range]	0.001686
[range___length]	0.001547

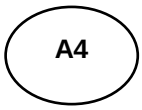
Tabela A3.2: Função *cosseno* para o *concern data type verification* (Tabela parcial)

Tokens	Dice
[isboolean___islogical]	0.015094339
[isstr___range]	0.006879944
[intmax___class]	0.004310344
[cast___class]	0.004260758
[length___size]	0.003652758
[intmin___class]	0.002614379
[length___class]	0.001325649
[isempty___class]	0.001318322
[ndims___isnumeric]	0.001255098
[ndims___isreal]	0.001109877
[length___range]	8.91646E-4
[ndims___numel]	8.64397E-4
[size___class]	8.2467E-4

Tabela A3.3: Função *dice* para o *concern data type verification* (Tabela parcial)

Tokens	SCP
[isboolean___islogical]	0.002214839
[intmin___class]	4.9191E-4
[intmax___class]	3.31253E-4
[cast___class]	1.82189E-4
[isstr___range]	5.0309E-5
[length___size]	1.3441E-5
[length___class]	1.0892E-5
[isempty___class]	7.667E-6
[length___range]	6.126E-6
[size___class]	3.604E-6
[ndims___length]	2.935E-6
[ndims___size]	2.93E-6
[size___range]	2.843E-6
[range___length]	2.393E-6

Tabela A3.4: Função *SCP* para o *concern data type verification* (Tabela parcial)



Tokens	Cosseno
[evalin___evalc]	0.004131
[evalin___eval]	0.001749
[eval___inline]	0.00117
[inline___eval]	5.85E-4
[eval___evalin]	2.18E-4

Tabela A4.1: Função *cosseno* para *o concern dynamic properties* (Tabela total)

Tokens	Dice
[evalin___eval]	0.001652721
[evalin___evalc]	0.001202284
[eval___inline]	5.81395E-4
[inline___eval]	2.90697E-4
[eval___evalin]	2.0659E-4

Tabela A4.2: Função *dice* para *o concern dynamic properties* (Tabela total)

Tokens	SCP
[evalin___evalc]	1.7067E-5
[evalin___eval]	3.059E-6
[eval___inline]	1.371E-6
[inline___eval]	3.42E-7
[eval___evalin]	4.7E-8

Tabela A4.3: Função *SCP* para *o concern dynamic properties* (Tabela total)

A5

Tokens	Cosseno
[disp___error]	0.009059
[display___disp]	0.007553
[lastwarn___error]	0.002913
[error___disp]	0.002905
[last___error]	0.001045
[disp___lastwarn]	7.62E-4
[disp___display]	4.82E-4
[error___display]	4.6E-4
[display___error]	3.07E-4

Tabela A5.1: Função *cosseno* para o *concern console messages* (Tabela total)

Tokens	Dice
[disp___error]	0.009050546
[display___disp]	0.003901224
[error___disp]	0.002903005
[lastwarn___error]	3.25719E-4
[disp___display]	2.49014E-4
[last___error]	2.4174E-4
[error___display]	2.28876E-4
[display___error]	1.52584E-4
[disp___lastwarn]	8.9122E-5

Tabela A5.2: Função *dice* para o *concern console messages* (Tabela total)

Tokens	SCP
[disp___error]	8.208E-5
[display___disp]	5.7062E-5
[lastwarn___error]	8.486E-6
[error___disp]	8.444E-6
[last___error]	1.094E-6
[disp___lastwarn]	5.8E-7
[disp___display]	2.32E-7
[error___display]	2.12E-7
[display___error]	9.4E-8

Tabela A5.3: Função *SCP* para o *concern console messages* (Tabela total)

A6

Tokens	Cosseno
[orient___print]	0.001854

Tabela A6.1: Função *cosseno* para o *concern printing* (Tabela total)

Tokens	Dice
[orient___print]	0.001803426

Tabela A6.2: Função *dice* para o *concern printing* (Tabela total)

Tokens	SCP
[orient___print]	3.439E-6

Tabela A6.3: Função *SCP* para o *concern printing* (Tabela total)

A7

Tokens	Cosseno
[ones___zeros]	9.45E-4
[zeros___ones]	8.86E-4
[global___delete]	2.07E-4
[delete___zeros]	1.17E-4
[zeros___global]	7.4E-5

Tabela A7.1: Função *cosseno* para o *concern memory allocation/deallocation* (Tabela total)

Tokens	Dice
[ones___zeros]	8.94579E-4
[zeros___ones]	8.38668E-4
[global___delete]	1.8716E-4
[delete___zeros]	7.4867E-5
[zeros___global]	6.3865E-5

Tabela A7.2: Função *dice* para o *concern memory allocation/deallocation* (Tabela total)

Tokens	SCP
[ones___zeros]	8.93E-7
[zeros___ones]	7.85E-7
[global___delete]	4.3E-8
[delete___zeros]	1.3E-8
[zeros___global]	5.0E-9

Tabela A7.3: Função *SCP* para o *concern memory allocation/deallocation* (Tabela total)

A8

Tokens	Cosseno
[labindex___numlabs]	0.210818

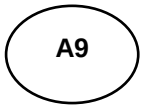
Tabela A8.1: Função *cosseno* para o *concern parallelisation* (Tabela total)

Tokens	Dice
[labindex___numlabs]	0.19047619

Tabela A8.2: Função *dice* para o *concern parallelisation* (Tabela total)

Tokens	SCP
[labindex___numlabs]	0.044444444

Tabela A8.3: Função *SCP* para o *concern parallelisation* (Tabela total)



Tokens	Cosseno
[save___uisave]	0.008353
[fopen___fprintf]	0.008079
[fprintf___save]	0.00194
[fprintf___load]	0.001772
[fopen___fwrite]	0.001003
[load___fprintf]	5.59E-4
[save___fprintf]	3.88E-4
[fopen___fread]	3.71E-4
[load___save]	3.52E-4

Tabela A9.1: Função *cosseno* para o *concern file I/O* (Tabela total)

Tokens	Dice
[fopen___fprintf]	0.004866479
[fprintf___load]	0.001147065
[fopen___fwrite]	9.995E-4
[save___uisave]	9.73709E-4
[fprintf___save]	9.60338E-4
[load___fprintf]	3.62231E-4
[load___save]	3.3428E-4
[fopen___fread]	3.3006E-4
[save___fprintf]	1.92067E-4

Tabela A9.2: Função *dice* para o *concern file I/O* (Tabela total)

Tokens	SCP
[save___uisave]	6.9788E-5
[fopen___fprintf]	6.5279E-5
[fprintf___save]	3.765E-6
[fprintf___load]	3.141E-6
[fopen___fwrite]	1.007E-6
[load___fprintf]	3.13E-7
[save___fprintf]	1.5E-7
[fopen___fread]	1.38E-7
[load___save]	1.24E-7

Tabela A9.3: Função *SCP* para o *concern file I/O* (Tabela total)

A10

Tokens	Informacao Mutua Pontual
[cla___reset]	10.4180131
[clf___reset]	10.0105358
[ylabel___zlabel]	9.9654632
[xlabel___ylabel]	9.8670482
[datetick___title]	8.917198
[figure___clf]	8.7693172
[close___gcbf]	8.3555224
[subplot___errorbar]	8.2282017
[datetick___ylabel]	8.1605421
[datetick___legend]	7.9508478
[ishold___hold]	7.8565597
[ylabel___xlabel]	7.6369329
[axis___image]	7.6244419
[fplot___plot]	7.6133228
[zlabel___title]	7.5564506
[clf___loglog]	7.5262116
[legend___datetick]	7.4914161
[subplot___semilogy]	7.4624845
[clf___surf]	7.3286116
[ylabel___title]	7.307325
[figure___errorbar]	7.1396578
[zoom___grid]	7.1231156
[subplot___plot]	7.1219021
[ylabel___legend]	6.9879722
[clf___semilogx]	6.8597677
[subplot___semilogx]	6.8518899
[figure___semilogy]	6.8478718
[legend___xlabel]	6.6896831
[subplot___cla]	6.6513442
[cla___hold]	6.6449219

Tabela A10.1: Informação mútua pontual para o *concern visualisation* (Tabela parcial)

Tokens	Cosseno
[xlabel___ylabel]	0.573238
[figure___clf]	0.225438
[set___gca]	0.217449
[ylabel___zlabel]	0.172517
[subplot___plot]	0.148052
[ylabel___title]	0.125579
[ylabel___xlabel]	0.122181
[clf___reset]	0.119624
[cla___reset]	0.091125
[axis___image]	0.088069
[datetick___title]	0.08017
[figure___plot]	0.078556
[title___xlabel]	0.07688
[close___gcbf]	0.07599
[figure___hold]	0.070573
[title___subplot]	0.069264

Tabela A10.2: Função *cosseno* para o *concern visualisation* (Tabela parcial)

Tokens	Dice
[xlabel___ylabel]	0.573210922
[figure___clf]	0.176271186
[subplot___plot]	0.136056971
[set___gca]	0.133984701
[ylabel___xlabel]	0.122175141
[ylabel___title]	0.121270791
[clf___reset]	0.091603053
[ylabel___zlabel]	0.090649064
[cla___reset]	0.089171974
[figure___plot]	0.076952052
[axis___image]	0.075325394
[title___xlabel]	0.074429981
[close___gcbf]	0.071116292
[figure___hold]	0.070443666
[title___subplot]	0.067971678
[figure___subplot]	0.047591883

Tabela A10.3: Função *dice* para o *concern visualisation* (Tabela parcial)

Tokens	SCP
[xlabel___ylabel]	0.328602883
[figure___clf]	0.050822669
[set___gca]	0.047284316
[ylabel___zlabel]	0.02976223
[subplot___plot]	0.021919414
[ylabel___title]	0.015770234
[ylabel___xlabel]	0.014928224
[clf___reset]	0.014309936
[cla___reset]	0.008303833
[axis___image]	0.007756209
[datetick___title]	0.006427273
[figure___plot]	0.006171183
[title___xlabel]	0.00591059
[close___gcbf]	0.005774542
[figure___hold]	0.004980634
[title___subplot]	0.004797629

Tabela A10.4: Função *SCP* para o *concern visualisation* (Tabela parcial)

A11

Tokens	Informacao Mutua Pontual
[unloadlibrary___loadlibrary]	16.8154466
[loadlibrary___calllib]	13.5524122
[etime___clock]	13.1185157
[pause___echo]	9.1273642
[toc___tic]	9.118706
[start___stop]	8.8342664
[exist___mexext]	8.2108687
[date___clock]	7.9119951
[etime___toc]	7.7378743
[mexext___exist]	7.5893803
[stop___start]	6.5663332
[clock___start, start___clock]	5.7362582
[step___stop]	5.6663591
[start___step]	5.6307238
[ans___input]	5.0242905
[date___mfilename]	4.6972874
[clear___mex]	4.6395974
[pause___tic]	4.2728524
[stop___toc]	4.220026
[toc___pause]	4.1538489
[step___start]	4.0457613
[pause___clear]	3.7046648
[stop___break]	3.6365873
[toc___clear]	3.5423444
[start___toc]	3.4925129
[mfilename___start]	3.0069773
[exist___stop]	2.3215365
[exist___step]	2.193282
[exist___clear]	0.6762764

Tabela A11.1: Informação mútua pontual para o *concern system* (Tabela total)

Tokens	Cosseno
[mislocked___munlock]	0.547722
[etime___clock]	0.349415
[unloadlibrary___loadlibrary]	0.316227
[loadlibrary___calllib]	0.138192
[start___stop]	0.124765
[toc___tic]	0.097362
[calllib___unloadlibrary]	0.047673
[mex___inmem]	0.045501
[pause___echo]	0.044949
[weekday___date]	0.034544
[exist___mexext]	0.029263
[stop___start]	0.025904
[mexext___exist]	0.019021
[now___spmd]	0.016771
[timerfind___stop]	0.015649

Tabela A11.2: Função *cosseno* para o concern system (Tabela parcial)

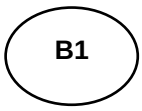
Tokens	Dice
[mislocked___munlock]	0.545454545
[unloadlibrary___loadlibrary]	0.315789473
[etime___clock]	0.299684542
[start___stop]	0.120901639
[toc___tic]	0.097279472
[loadlibrary___calllib]	0.059459459
[pause___echo]	0.028735632
[stop___start]	0.025102459
[calllib___unloadlibrary]	0.021505376
[date___clock]	0.015572858
[start___step]	0.011541632
[step___stop]	0.009719626
[etime___toc]	0.008468595
[clock___start, start___clock]	0.006177076
[unloadlibrary___calllib]	0.005376344

Tabela A11.3: Função *dice* para o *concern system* (Tabela

Tokens	SCP
[mislocked___munlock]	0.3
[etime___clock]	0.12209145
[unloadlibrary___loadlibrary]	0.1
[loadlibrary___calllib]	0.019097222
[start___stop]	0.015566325
[toc___tic]	0.009479402
[calllib___unloadlibrary]	0.002272727
[mex___inmem]	0.002070393
[pause___echo]	0.002020437
[weekday___date]	0.001193317
[exist___mexext]	8.56333E-4
[stop___start]	6.71049E-4
[mexext___exist]	3.618E-4
[now___spmd]	2.81293E-4
[timerfind___stop]	2.44897E-4

Tabela A11.4: Função *SCP* para o *concern system* (Tabela parcial)

Anexo B - Código *MATLAB* que relaciona *tokens* e palavras reservadas.

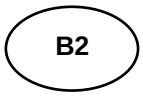



```
function [v,s,d] = firstncut(base_name,rec_num);

if nargin<2 | isempty(rec_num),
   rec_num = 1;
end
```

Figura B1.1: Utilização da palavra reservada *if* e do *token nargin* em conjunto no código *MATLAB*

```
function z = circle(v,r,c,n)
%
%  circle(v,r,n)
%
%   Plot a circle
%
%     v - center (either 2 or 3 dimensional vector)
%     r - radius; default = 1
%     c - color; default = white
%     n - number of plot points; default = 20
%
%   See also TRIANGLE

if nargin < 2, r = 1; end
if nargin < 3, c = 'w'; end
if nargin < 4, n = 20; end
```

Figura B1.2: Utilização da palavra reservada *if* e do *token nargin* em conjunto no código *MATLAB*




```
function [a_afterQ] = Qtest(a,p)
    %%%a--input vector;p--significant value 0.1,0.05,0.01
    if length(a)<3||length(a)>10
        error('Q test is not effective for this data');
        return
    end

    Table = [0.941 0.765 0.642 0.560 0.507 0.468 0.437 0.412;...
     0.970 0.829 0.710 0.625 0.568 0.526 0.493 0.466;...
     0.994 0.926 0.821 0.740 0.680 0.634 0.598 0.568];
switch p
    case 0.1
        value = Table(1,:);
    case 0.05
        value = Table(2,:);
    case 0.01
        value = Table(3,:);
    otherwise
        error('A wrong significant value inputed');
        return;
end
```

Figura B2.1: Utilização da palavra reservada *otherwise* e do *token error* em conjunto no código *MATLAB*

```
switch nargin
    case 1
        nc = mDataset(uri);
        result=nc.getInfo;
     otherwise, error('MATLAB:cf_info:Nargin',...
                'Incorrect number of arguments');
end
%cleanup
nc.close();
clear nc;
```

Figura B2.2: Utilização da palavra reservada *otherwise* e do *token error* em conjunto no código *MATLAB*

Anexo Código - Código *java* desenvolvido para construção de métricas de associação de palavras.

```
public class relatedWords {
	private HashMap<String,Integer> indTokens;
	public double size;
	private HashMap<String, Integer> seq;

	public mutualInfo(String filename) throws IOException{
		size=0.0;
		indTokens = new HashMap<String,Integer>();
		seq =  new HashMap<String, Integer>();
		loadFile(filename);
	}

		private void loadFile(String filename) throws IOException
{
		BufferedReader txt = new BufferedReader(new
FileReader(filename));
		String line = txt.readLine();
		String[] toks = line.split(" ");
		size = toks.length;
		//preenchimento do hashmap com as ocorrencias de tokens
individualmente
		for(int i=0;i<toks.length;i++) {
			indTokens.put(toks[i], 0);
		}
		for(int i=0;i<toks.length;i++) {
			indTokens.put(toks[i], indTokens.get(toks[i])+1);
		}
		//preenchimento do hashmap com as ocorrencias de pares de
tokens consecutivos
		for(int j = 0; j < toks.length-1; j++) {
			String first = toks[j];
			String second = toks[j+1];
			String concat = first.concat(second);
			if(seq.containsKey(concat)) {
				seq.put(concat, seq.get(concat)+1);
			}
			else
				seq.put(concat, 1);
			}
	}

//	devolve numero de determinado token no rep
	public int getFreqToken(String tok) {
		return indTokens.get(tok);
	}

//	devolve numero de sequencia de determinados tokens no rep
	public int getFreqTokenCons(String a, String b) {
		String conc = a.concat(b);
		int val = 0;
		if(seq.containsKey(conc)) {
			val = seq.get(conc).intValue();
		}
		return val;
	}
```

```
	public int getFreqTokenCons2(String str) {
		String[] parts = str.split("___");
		String part1 = parts[0];
		String part2 = parts[1];
		String conc = part1.concat(part2);
		int val = 0;
		if(seq.containsKey(conc)) {
			val = seq.get(conc).intValue();
		}
		return val;
	}

//	devolve a probabilidade de um determinado token no rep.
//	devolve 0 caso esse token se encontre menos de 5x no rep
	public double getProb(String a) throws IOException {
		if(indTokens.get(a)<5) {
			return 0.0;
		}
		else
			return indTokens.get(a)/size;
	}

//	devolve a probabilidade de uma determinada sequencia de tokens
//	devolve 0 caso essa sequencia de tokens apareca menos de 5x no
rep
	public double getProb2(String a,String b) throws IOException {
		String conc = a.concat(b);
		double val = 0.0;
		if(seq.containsKey(conc) && seq.get(conc).intValue()>4) {
			val = seq.get(conc).doubleValue();
		}
		else
			return 0.0;
		double x = val/(size-1);
		return x;
	}

//	calculo da IMP para 2 tokens
	public double getMI(String a,String b) throws IOException {
		double prob = 0;
		double probA = getProb(a);
		double probB = getProb(b);
		double probAB = getProb2(a,b);
		double multAB = probA * probB;
		prob = probAB/multAB;
		return prob;
	}

//	calculo do cosseno para 2 tokens
	public double getCos(String a,String b) throws IOException {
		double cosAB = 0.0;
		double freqA = getFreqToken(a);
		double freqB = getFreqToken(b);
		double freqAB = getFreqTokenCons(a,b);
		double ab = freqA * freqB;
		double sqrtAmultB = Math.sqrt(ab);
		cosAB = (int)((freqAB/sqrtAmultB)*1000000)/1000000.0;
		return cosAB;
	}

//	calculo do dice para 2 tokens
```

```
public double getDice(String a,String b) throws IOException {
            double diceAB = 0.0;
            double freqA = getFreqToken(a);
            double freqB = getFreqToken(b);
            double freqAB = getFreqTokenCons(a,b);
            double twoab = 2 * freqAB;
            double amoreb = freqA + freqB;

            diceAB = (int)((twoab/amoreb)*1000000000)/1000000000.0;
            return diceAB;
      }

//    calculo do SCP para 2 tokens
public double getSCP(String a,String b) throws IOException {
      double scpAB = 0.0;
      double freqA = getFreqToken(a);
      double freqB = getFreqToken(b);
      double freqAB = getFreqTokenCons(a,b);
      double ab = freqA * freqB;
      scpAB =
(int)((((freqAB)*(freqAB))/(ab))*1000000000)/1000000000.0;;
      return scpAB;
}

//    devolve o logaritmo de um determinado valor
      public double getLog(double val) throws IOException {
            return
(int)((Math.log(val)/Math.log(2))*10000000)/10000000.0;
      }

      public boolean hasToken(String a) throws IOException {
            return indTokens.containsKey(a);
      }

      public String[] rep() throws IOException {
            String[] list = indTokens.keySet().toArray(new
String[indTokens.size()]);
            return list;
      }
}
```