NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

DIOGO DAVID SOUSA ALMEIDA

Bachelor in Computer Science

A CHARACTERIZATION STUDY OF MATLAB AND CODING ANTI-PATTERNS

LANGUAGE CONSTRUCTS AND THEIR IMPORTANCE IN MATLAB

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
March, 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

A CHARACTERIZATION STUDY OF MATLAB AND CODING ANTI-PATTERNS

LANGUAGE CONSTRUCTS AND THEIR, IMPORTANCE IN MATLAB

DIOGO DAVID SOUSA ALMEIDA

Bachelor in Computer Science

Adviser: Miguel Pessoa Monteiro
Assistant Professor, NOVA University Lisbon

Co-adviser: Nuno Miguel Cavalheiro Marques
Assistant Professor, NOVA University Lisbon

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
March, 2023

A Characterization Study of MATLAB and Coding Anti-Patterns

To and for family and friends.

Acknowledgements

It might be a bit too difficult to properly write a thank you text/note to all of those who helped during the elaboration of this thesis. However I will do my best to provide some kind words.

The people who guided me the most during the elaboration were definitely Dr. Miguel Monteiro and Dr. Nuno Marques. Who together worked in order to provide me the language construct idea/definition. Other guidance was obviously given to me by them, however in my opinion, that was the most important one.

This thesis, which was fun to make, obviously would not be possible without years of work and structuring, of the Program of Computer Science, done by everyone working in FCT/UNL and more specifically in the Departamento de Informática. So to every worker/professor/student that put their efforts into making this college a better place, thank you.

And finally to my family and friends who, by giving me small or big emotional support, where always by my side insisting that I should carry on doing what I like. This thank you is not a professional thank you, so I do not know how to express my felling in a small text, as well. So I will try to thank the favor by standing by your side, in the good moments, but specially in the bad ones, hopefully for many years.

"You cannot teach a man anything; you can only help him discover it in himself." (Galileo)

Abstract

This thesis is an analysis of the MATLAB language. The analysis performs a characterization of the language and detects some specific coding anti-patterns.

The work for this thesis builds on previous research on techniques for concern location in MATLAB code bases. It is based on a SQLite database that contains all the lexical tokens from a given repository of MATLAB systems. Previous research studied the use of this database for representing the MATLAB repository and supporting higher-level concepts, as well as its detection in specific MATLAB files. The viability of this approach was demonstrated in work prior to this thesis using SQL queries over the database, demonstrating that code blocks can be used to characterize the language and describe some coding anti-patterns. The present thesis extends previous work by extending the concept of code blocks. It incorporates that notion into the broader concept of language constructs. Support for language constructs in the SQLite platform are a contribution of this thesis. The results were derived from a repository of over 450 000 MATLAB files. Work on this repository also serves as a MATLAB characterization study.

Keywords: Software Quality, Concerns, Language Constructs, Coding Anti-Patterns, MATLAB, Object Oriented Features, Schizophrenic Functions, Code Blocks, Characterization Study, Data Base, SQLite

Resumo

Esta tese é uma análise da linguagem MATLAB. A análise realiza uma caracterização da linguagem e detecta alguns anti-padrões de codificação específicos.

O trabalho para esta tese baseia-se em pesquisas anteriores sobre técnicas de localização de concerns em bases de código MATLAB. É baseado numa base de dados SQLite que contém todos os tokens lexicais de um determinado repositório de sistemas MATLAB. Investigações anteriores estudaram o uso desta base de dados para representar o repositório MATLAB e dar suporte a conceitos de alto nível, bem como sua detecção em ficheiros MATLAB específicos. A viabilidade desta abordagem foi demonstrada em trabalhos anteriores a esta tese utilizando consultas SQL à base de dados, demonstrando que os blocos de código podem ser utilizados para caracterizar a linguagem e descrever alguns anti-padrões de codificação. A presente tese estende trabalhos anteriores ao estender o conceito de blocos de código. Esta tese incorpora essa noção no conceito mais amplo de construções de linguagem. O suporte para construções de linguagem na plataforma SQLite é uma contribuição desta tese. Os resultados foram derivados de um repositório de mais de 450 000 ficheiros MATLAB. O trabalho neste repositório também serve como um estudo de caracterização do MATLAB.

Palavras-chave: Qualidade de Software, Concerns, Construções de Linguagem, Anti-Padrões de Codificação, MATLAB, Funcionalidades Orientadas aos Objectos, Funções Esquizofrênicas, Blocos de Código, Estudo de Caracterização, Base de Dados, SQLite

Contents

List of Figures

List of Tables

List of Listings

Glossary

Code Tangling Tangling is how entangled the system is, so in other words, what is the degree to which the parts of the system are mixed with each other. 20, 21, 22

Code Scattering Scattering is when the source text, of a specific functionality, is distributed among some parts of the system. 20, 21

code block A code block is a piece of source text identified by it's keyword, like "if", and that same code block ends with the keyword "end", in the case of MATLAB. 3, 24, 27, 36, 37

Coding Anti-Pattern In software, anti-pattern is a term that describes how NOT to solve recurring problems in your code. Anti-patterns are considered bad software design, and are usually ineffective or obscure fixes. [6] 1, 3

concern A concern is any abstraction, concept or consistent set of responsibilities that we would like to localize in its own module [25]. xii, 1, 2, 3, 12, 13, 20, 21, 22, 23, 24, 26, 27, 29, 36, 37, 38, 40, 45, 56, 57, 59, 60, 62, 68, 69, 70, 71, 75, 76

Crosscutting Concern Crosscutting concerns are aspects of a program that affect other concerns. These concerns often cannot be cleanly decomposed from the rest of the system in both the design and implementation, and can result in either scattering (code duplication), tangling (significant dependencies between systems), or both. [35] 3, 23, 26

Language Construct A Language Construct is a syntactically correct part of a programming language that is formed by one or more lexical tokens, that have a specific functionality and that is not a function. 1, 2, 3, 4, 6, 8, 12, 13, 14, 24, 36, 37, 40, 42, 43

schizophrenic function Schizophrenic functions are functions that hold several behaviours in their implementation, but the execution flow varies according to the arguments that they are called with. 3, 14, 15, 17, 37, 45, 61, 62, 64, 65, 71, 72, 74, 77

script A Script is a mfile that does not contain a single function 7

token Tokens are the lexical elements extracted from a code file by means of some lexical analyzer tool.[4] 2, 12, 13, 14, 21, 22, 23, 24, 26, 27, 28, 29, 40, 41, 45, 46, 59, 63, 70, 75, 76, 77

Acronyms

CBLC	Code Block Language Construct xiv, 13, 14, 24, 27, 31, 33, 36, 45, 46, 49, 50, 54, 55, 56, 57, 59, 67, 68, 69, 74
CCC	Crosscutting Concern 20, 21, 23, 24
CCCExplorer	CrossCutting Concern Explorer 24
ERD	Entity Relationship Diagram xii, 34
LC	Language Construct 37, 45, 46, 56, 57, 59, 60, 68, 69, 70, 73, 75, 76, 78
LoC	Lines of Code xiv, 55, 68, 69
MATLAB	Matrix Laboratory 1
non-CBLC	Non-Code Block Language Construct 12, 13, 31, 35, 37, 40, 41, 43, 57, 59, 76
OO	Object Oriented 2, 3, 4, 8, 45, 46, 48, 64, 66, 67, 72, 73, 74, 75
OOL	Object Oriented Language 67
PMI	Pointwise Mutual Information 56, 57, 59, 69, 70, 73

1 Introduction

This thesis is about an analysis made over a MATLAB repository. This analysis is based on Language Constructs. The conclusions from the analysis are a characterization study of the MATLAB language and the findings of some more common Coding Anti-Patterns.

1.1 Context

This thesis fits in a series of other Master Thesis, that are all related to a technique of locating concerns in Matrix Laboratory (MATLAB) source text, or are characterization studies of the MATLAB language, like the one performed by E. Reis [26]. This technique was developed, tested and explored in several publications [3] [23] [24]. Tokens from the MATLAB language are used in this method to locate concerns. More about this technique in Chapter 3. This method is currently only applicable to the MATLAB programming language, but it is intended to be further developed for additional programming languages, such as R and Phyton.

- **Using a Repository of MATLAB files**

  In order to test their methods and access the results, the majority of technical studies that focus on scientific advancements in the identification and localization of concerns use a repository. With the assistance of Professors, Master students and even some Bachelor students, NOVALINCS at DI NOVA has been developing and expanding the MATLAB repository [4][29][30].

- **Generalization of Results**

  Utilizing a sizable repository is intended to enable generalization of the findings to the community, in this case the MATLAB community. The results can be more broadly generalized to the population the larger the repository. Because of this, the larger the repository, the more trustworthy the results of the analysis will be. It is wise to bear this in mind, and as a result, there are plans to expand the repository in the future.

- **Repository Growth**

  A smaller repository than what we currently have was used for the majority of studies done using this technique. The results obtained using this new large repository can, generally speaking, serve to characterize the majority of the MATLAB community since we now have a repository of over 450 000 mfiles, compared to when we only had about 65 000 mfiles.

- **Main Focus of this Thesis**

  This thesis focuses primarily on concerns and MATLAB Language Constructs. For more information on concerns, see Chapter 3, and for more on Language Constructs, see Chapter 2.

- **Background on Concern Location**

  There is potential for making scientific advancements in this field because finding problems for OOLs is well developed [23], but MATLAB is still underdeveloped. The approach to identifying concerns also needs to be rethought and carefully planned because the language (MATLAB) differs so much from OOLs due to its "specific characteristics and different typical uses" [23]. In other words, it needs to be radically different and completely original. A token-based approach to identifying concerns in MATLAB is explored by Monteiro *et al.* [23]. By locating a token on a MATLAB source text file, the authors of this paper propose to identify various types of concerns.

- **Language Construct Location**

  By identifying lexical tokens, Language Constructs can also be located using a similar token approach. The relationship between Language Constructs and lexical tokens is displayed in the Table 2.3.

- **Token Location Similarities**

  Using these two techniques, tokens related to concerns and tokens related to Language Constructs can be identified. It is possible to see the relationships between these two types of tokens in MATLAB source text files by using these two types of tokens. By extension, discovering the co-relationships between concerns and Language Constructs. More about these relations in Chapter 2.

- **Characterization Study**

  The purpose of this thesis is to provide a characterization study of the MATLAB language using LCs and concerns. This research includes information on the following topics: the Object Oriented (OO) features in the MATLAB language, the frequency of structures used per mfile and the complexity of mfiles, the relationships between

LCs and concerns, and schizophrenic functions (more on schizophrenic functions in Chapter 2).

- **OO Features Motivation**

  This idea originated with Eduardo Reis [26], who wrote in his Master Thesis that the OO features in the MATLAB language are not widely used, citing his MATLAB community inquiries as evidence. It is thought prudent to verify this situation in a large repository to supplement his work.

1.2 Problem and Motivation

Some studies on crosscutting concerns and concerns in MATLAB have been conducted, but this technical/scientific work could be expanded for this language, according to Monteiro *et. al.* [23]. Check out Chapter 3 for more information on crosscutting concerns. With more developed scientific information about this situation, MATLAB language developers can further improve their future systems so that they comply with accurate programming practices. According to Cardoso *et. al.* [3], the positive consequences of improving information about concerns and crosscutting concerns could lead to more maintainable MATLAB systems.

J. Barrulas [2] conducted a study that outlined Coding Anti-Patterns that used code blocks. The code blocks that J. Barrulas uses are also Language Constructs, there is more information on code blocks in Chapter 2. In order to improve the chances of producing better results, it is intended in this thesis to use Language Constructs that are wider range of analysis criteria than code blocks, and a larger repository.

These problems must be addressed because the MATLAB community could benefit from more maintainable systems if these situations were better understood.

1.3 Objectives and Contributions

The primary goal of this thesis is to present a thorough characterization study. This study will be conducted using a sizable repository of MATLAB source text files, resulting in accurate and objective data. This study can be used to learn more about the MATLAB community and the language itself. A programmer could select MATLAB over another programming language for the development of a system with the help of this characterization study, because it provides trustworthy information on how the community prefers to use the language.

Finding coding anti-patterns is another goal of this thesis. Language Constructs are studied to try to identify MATLAB coding anti-patterns. Finding coding anti-patterns can aid in the development of more maintainable and efficient systems. The characterization study is used to identify these coding anti-patterns. This study uncovered some

commonly used programming practices, some of which can be classified as coding anti-patterns.

The Research Questions of this thesis are:

- **RQ1:** What is the scope of the concrete use of OO features in MATLAB?
- **RQ2:** What are the most used LCs in MATLAB, and how frequently in each mfile?
- **RQ3:** What are the most frequent concerns/LCs relations?
- **RQ4:** What is the average degree of complexity of mfiles, in MATLAB?
- **RQ5:** Are schizophrenic functions so frequent/probable in a repository of 450 000 mfiles as in a 65 000 repository?
- **RQ6:** Can schizophrenic functions be related to the OO paradigm?

1.4 Document Structure

This document is a report for this thesis and it is structured as follows:

- **Chapter 1** - Introduction - A brief introduction to this thesis' document.
- **Chapter 2** - The MATLAB Language - A description about the MATLAB language, story, owner and it's features. It also mentions a very important part of this thesis that are MATLAB's Language Constructs. Within the Language Constructs section, Section 2.3, it is provided some details on how they will be analysed.
- **Chapter 3** - Token Approach - This chapter explains the approach that is in the origin of this and the other master thesis related to this one.
- **Chapter 4** - CCCExplorer: Token Extraction Tool - This chapter further informs on the tool that is used to apply the approach of the previous chapter.
- **Chapter 5** - CCCExplorer's Database - This chapter describes the specifics of the database that was produced by the tool discussed in the previous chapter and used to identify coding anti-patterns and produce the characterization study.
- **Chapter 6** - Implementation - This chapter describes all practical work done before the analysis.
- **Chapter 7** - Repository Analysis - This chapter provides all the results that originated from the analysis.
- **Chapter 8** - Results Discussion - In this chapter it is described all the conclusion that can be withdrawn from the analysis' results.

- **Chapter 9** - Conclusions and Future Work - In this chapter there is all the conclusions from this thesis and some future work possibilities.

2

The MATLAB Language

This chapter explains in some details the most relevant part of the MATLAB language. But most importantly it provides information about what are and how Language Constructs are analysed in this thesis.

2.1 Overview of MATLAB

This chapter was written to clarify the reader about some of the MATLAB's most important features. It is written this way in order to be applied to this thesis, specially in Section 2.3. This section further explains what are Language Constructs, and how they will be analysed. The Table 2.3, contributes with the list of Language Constructs and the keyword that identifies each of them. The Table 2.3 was made specifically for this thesis. This table originated from online research [37] [38] [36] [32], and also from the work of J.Barrulas [2].

- **The MATLAB Language**

  The name MATLAB is derived from MATrix LABoratory. This language and related libraries belong to MathWorks Inc. This way being a proprietary programming language. This company only allows some toolboxes to be available after payment, this way having profits. As the name implies, MATrix LABoratory, MATLAB, is based on matrix calculations along with numeric calculations. It also contains signals processing, visualization and algorithm development. Since software creators can develop fast in MATLAB it is often used to make prototypes. However it can be very good to make prototypes, it can be rather challenging to develop very complex source text, and the results of these more complex systems are also not top level [9].

- **MathWorks Inc.**

  MathWorks Inc. was founded in 1984, and they have been profitable since, 1 billion USD in profits in 2019, they have customers in over 185 countries. They have more than 4 million MATLAB users worldwide, 625,000 contributors, and employ over 5000 people in 33 offices worldwide. The MATLAB language together with Simulink

have originated over 2100 books in 26 different languages. This programming environment is also one of the key elements in the teaching of computer science, and gets this title for being lectured in over 6500 colleges and universities [12].

2.2 MATLAB's Syntax

This section contains a presentation of the MATLAB language, in a way that the reader can be better informed and better guided to understand the rest of this thesis.

2.2.1 MATLAB's Basics

In the most overall view of the MATLAB's way of functioning we have two main 'objects' that store information on MATLAB systems. These are MATLAB files, known as mfiles, and toolboxes, that can store many mfiles.

- **Mfiles**

  Mfiles contain all the source text that a MATLAB system contains. Toolboxes have inside them several mfiles, in other words they create groups of mfiles so that the system is better organized, and therefor more maintainable. Since mfiles contain all the source text of the system and there are several types of source text, they can be of several types. Some of those most important types to this thesis are: Scripts, Classes and non-Scripts mfiles.

- **Scripts and the 'Function' Block**

  To start explaining what a script is in MATLAB is it must first be explained what is a function block. A function block in MATLAB is the block that stores a single function inside it. This block can be identified by the key word 'function'. So an mfile may have one or several functions (function blocks) inside it, or even zero functions.

- **Scripts**

  A script is a mfiles that is mostly used as a main of a system, or in the case of a small system can even contain the entire system. Scripts are identified as scripts when the mfile does not contain a single function block, in other words when the mfile does not have functions, it is just source text.

- **Non-Script Mfiles**

  Safe to assume that the non-script mfiles need to be different from the scripts. The way that they differ is in the amount of function blocks in the given mfile. These non-script mfiles contain at least one function block.

- **Mfiles with Classes**

  The mfiles that are classes, or contain classes, are named this way because they contain at least one object class in them. The MATLAB language was not created with OO features, so all the OO features that are mentioned in this thesis are rather recent when compared with the full age of the language. This meaning that these sort of features are not yet widely used, as it is possible to be seen in the later chapters of this thesis.

2.2.2 Variables, Matrices and Arrays

Variables in MATLAB do not have a defined type. So, in a single program run of a MATLAB system, a variable can have several values of different types. Every variable in MATLAB is a matrix, this is demonstrated in Table 2.1, on the lines Numeric Value and String Value. When one single variable is declared they can be accessed by using the origin point of a matrix ([0][0]). Within numeric calculations, each number value starts, by default, as a double [17]

Table 2.1: MATLAB example of Variables, Matrices and Arrays.

Type	Source Text
Numeric Value	x = 3;
	x[0][0] = 3;
String Value	myText = 'This is an example';
	myText[0][0] = 'This is an example';
Array (4 elements)	a = [5 6 7 8];
Matrix (3 by 3)	m = [9 10 11; 12 13 14; 15 16 17];

2.2.3 Operators

There are several types of operations in MATLAB. The example for Relational Operations is in Table 2.2. Arithmetic operations are depicted in Listing 2.1. There are also Logical Operations, these are better explained in Section 2.3, as belonging to Language Constructs entirely. Other type of operations are Set and Bitwise operations [34].

```
1  % Adding
2  A + B;
3  plus(A,B);
4
5  % Subtracting
6  A - B;
7  minus(A,B);
8
9  % Multiplying
10 A.*B;  times(A,B);      %Arrays
11 A * B; mtimes(A,B);     %Matrices
12 A * 3; times(A,3);      %Numbers
```

```
13
14 % Right Division (Divides A by B)
15 A./B;  rdivide(A,B);    %Arrays
16 A / B; mrdivide(A,B);   %Matrices
17 3 / A; rdivide(3,A);    %Numbers
18
19 % Left Division (Divides B by A)
20 A.\B;  ldivide(A,B);    %Arrays
21 A \ B; mldivide(A,B);   %Matrices
22 3 \ A; ldivide(3,A);    %Numbers
23
24 % Exponential
25 A.^B; power(A,B);       %Arrays
26 A^B:  mpower(A,B);      %Matrices
27 A^3;  power(A,3);       %Numbers
28
29 % Transpose
30 A.'; transpose(A);      %Arrays
31 A';  transpose(A);      %Matrices
```

Listing 2.1: MATLAB example of Arithmetic Operations

Table 2.2: MATLAB example of Relational Operations.

Type of Relational Operation	**Source Text**
Equal to	A == 3;
	eq(A,3);
Not Equal to	A ~= 3;
	ne(A,3);
Less than	A < B;
	lt(A,B);
Less or Equal to	A <= C;
	le(A,C);
Greater Than	A > D;
	gt(A,D);
Greater or Equal to	A >= E;
	ge(A,E);
Array Equal to another Array	isequal(A,B, ...)
Array Equal to another Array (while dealing with NaN)	isequaln(A,B, ...)

2.2.4 Object Oriented Features

There are several object oriented features implemented in MATLAB. The ones that are most discussed in this thesis are inheritance, methods and visibility configurations for methods and classes, like private, public and protected.

2.2.4.1 Classes Declaration

Each class in MATALB must be declared with the keyword 'classdef'. This is a special keyword that always symbolizes the declaration of a class. Following this keyword must be the name of the class.

- **Property**

  Afterwards another set of instructions begins, this set is named properties. This block properties also as a special keyword named 'properties', and it is inside this block of source text that all properties related to the class must be declared. It is also inside this properties block that variables are declared. This variables may have types, and even restrictions, like the one seen with the variable 'Value' in Listing 2.2 [18].

- **Property Attributes**

  Inside this block many types of attributes can be defined, however no specific type of attribute needs to be specified to declare a class. Because all types of attributes have a default value that is used when it is not specified. The most relevant attributes to this thesis are the visibility attributes: Access, GetAccess and SetAccess.

- **Property Attributes: Visibility**

  GetAccess is used when the programmer want to specify the visibility in terms of the ability to read information from the class. SetAccess is used to specify the visibility in terms of the ability to write or edit information stored in the class. Access is the junction of both. When Access is used both GetAccess and SetAccess are configured to the same value. These values can be: Public, Private, Protected or a list of classes that can have Access, GetAccess or SetAccess set to public. If not mentioned Public, the default value, is applied [19].

- **More than 1 Property per Class**

  One aspect to keep in mind is that the properties block can be defined several times in a single class. Because it's purpose is to define how variables (properties in MATLAB) can be read, edited or both, and for this purpose the MATLAB creators decided that the developer could chose to give different authorizations to different variables (properties) in the same class [19].

- **Methods**

  After the properties block usually it is declared the methods block, where all methods of the class are declared and written. These methods like most programming languages also have arguments, in which the first argument is the object in which

the method is being declared [20]. Also important to notice that there can be several methods blocks, with different values to their attributes. This could result for example in private and public methods co-existing in the same class.

- **Method Attributes: Visibility**

  This block also contains attributes, these are declared between curved brackets next to the 'methods' special keyword. The most important method attribute for this thesis is Access, that works the same way as for classes. Note that in methods GetAccess and SetAccess do not exist [21].

- **Class Example**

  In Listing 2.2, we can see a simple example of a class being declared, with the 'classdef', 'properties' and 'methods' special MATLAB keywords. As it is possible to see in Table 2.3, all this blocks are also code block language constructs. This is better explained further ahead in this Chapter.

```
classdef SimpleClass
properties
  Value {mustBeNumeric}
end
methods
  function R = roundOff(object)
    R = round([object.Value],2);
  end
  function R = DivideBy(object,n)
    R = [object.Value] / n;
  end
end
end
```

Listing 2.2: MATLAB example of Class Declaration [7]

2.2.4.2 Inheritance

Inheritance is declared in MATLAB resorting to the '<' keyword. The subclass is named before the keyword and the super class afterwards.

In Listing 2.3 we can see an example of inheritance, private class, declaration of variables and public method.

```
classdef MyMap < matlab.mixin.indexing.RedefinesParen
properties (Access = private)
   Keys (:,1) string
   Values (:,1) cell
end
%The private properties Keys and Values are n-by-1 vectors.
methods (Static, Access = public)
   function obj = empty(varargin)
```

```
9         if nargin == 0
10           obj = MyMap(string.empty(0,1),cell.empty(0,1));
11           return;
12        end
13     end
14 ...
```

Listing 2.3: MATLAB example of Class Inheritance [13]

2.3 Analysis of MATLAB's Language Constructs

A definition to Language Construct is: "A language construct is a syntactically allowable part of a program that may be formed from one or more lexical keywords in accordance with the rules of a programming language. The term Language Construct is often used as a synonym for control structure" [36]. However a Language Construct can be a 'while' it can never be a function like 'sin()' or 'add()' [32].

2.3.1 Non Code Block Language Constructs

Every language construct present in the Table 2.3, is a Non-Code Block Language Construct (non-CBLC), except the language constructs that belong to: Control Flow, Object Oriented Features, and Subroutines-Methods.

- **Analysis with non-CBLC**

  These Language Constructs do not encapsulate any source text. For this reason a suggested method of analysis is to analyse the same line of source text where it appears. In order to be the most specific possible in the analysis of the relation Language Construct/concern.

- **Analysis with non-CBLC: Example**

  Some examples of co-occurrences of these types of Language Constructs with concern tokens are in: Listing 2.4, Listing 2.5, Listing 2.6. These listings are shown in this section to demonstrate how these LCs can relate to concern tokens. And show a practical example of the LCs relating to the concern tokens, and this way showing a source text example of how the analysis described in Chapter 7 can be made.

- **Analysis with non-CBLC: Concern Tokens**

  The Figure 3.3, provides the concern/token relationships. These relationships represent a token that can indicate the presence of a concern in MATLAB. The concern tokens used in the Listings mentioned above are present in the table of this figure. More about this mapping in the next chapter.

- The Listing 2.4, shows the occurrence of the '&&' Language Construct with the 'isinteger' and 'isfloat' concern tokens.
- The Listing 2.5, shows the occurrence of the '=' Language Construct with the 'parfor' concern token. This specific concern token represents both the parallelization concern and the code block language construct. But in this case is being analysed as a concern token.
- The Listing 2.6, shows the occurrence of the '*' Language Construct with the 'fprintf' concern token.

```
x=5
y= 2.0
if isinteger(x) && isfloat(y)
    x=6
end
```

Listing 2.4: Example of AND(wsc) and Identification of Numeric Types Token

```
n = 200;
A = 500;
a = zeros(1,n);
parfor i = 1:n
    a(i) = max(abs(eig(rand(A))));
end
```

Listing 2.5: Example of Assig. of Values and Parallelization Token [14]

```
fprintf('%0.f\n',1000*pi)
```

Listing 2.6: Example of Multiplication and File I/O Token [15]

2.3.2 Code Block Language Constructs

The code block language constructs (CBLC) present in the Table 2.3 are: Control Flow, Object Oriented Features, and Subroutines-Methods.

- **Identification of a CBLC**

  On these cases the Language Construct encapsulates a part of the source text of the system. One general characteristic of these code blocks is their composition in at least two keywords. In example, when an 'If' is declared it starts with the 'if' keyword and always ends with the 'end' keyword. So everything in between belongs to this 'If' code block.

- **CBLC or non-CBLC**

As it is possible to understand there are only two possibilities for Language Constructs, either they encapsulate source text or they do not. Because the only possibilities are: they are either a single keyword or a group of keywords with source text in between the first and the final keyword. In this later case they are nearly all composed by two keywords, the main keyword and the 'end' keyword, and some source text is left between these two keywords. So every language construct is either a code block language construct or a non-code block language construct.

- **CBLC Examples**

  Some examples of co-occurrences of code block language constructs with concern tokens are in: Listing 2.7, Listing 2.8 and Listing 2.9. These listings are shown in this section to demonstrate how these LCs can relate to concern tokens.The Listing 2.8 is also the case of a schizophrenic function, more about schizophrenic functions further ahead in this chapter.

- The Listing 2.7, shows the occurrence of the 'if' Language Construct with the 'disp' concern token.
- The Listing 2.8, shows the occurrence of the 'if' and 'switch' Language Constructs with the 'nargin' concern token.
- The Listing 2.9, shows the occurrence of 'try/catch' Language Constructs with the 'error' concern token.

```
reply = input('Would you like to see an echo? (y/n): ','s');
if strcmp(reply,'y')
  disp(reply)
end
```

Listing 2.7: Example of If and Console Messages Token [16]

```
function result = getLonAxis(mGridCoordinatesObj)
     result = [];
     if nargin < 1, help(mfilename), return , end
     try
        switch nargin
            case 1
                result=squeeze(mGridCoordinatesObj.myCoordID.getLonAxis());
            otherwise , error("MATLAB:mGridCoordinates:getLonAxis:Nargin",...
     "Incorrect number of arguments");
        end
     catch %gets the last error generated
        err = lasterror();
        disp(err.message);
     end
end
```

Listing 2.8: Example of If, Switch and Verification of Function Arguments Token [2]

```
function out=nnProc(spd, trq, time, nnFuncs , outputNames , outputUnits)
 try
    spd=makeRowArray(spd);
 catch
    error("[nnProc.m]: spd array is not correct --please check inputs and try
    again")
 end
 try
    trq=makeRowArray(trq);
 catch
    error("[nnProc.m]: trq array is not correct --please check inputs and try
    again")
 end
 try
    time=makeRowArray(time);
 catch
    error("[nnProc.m]: time array is not correct --please check inputs and try
     again")
 end
 ...
```

Listing 2.9: Example of Try/Catch and Console Messages Token [2]

2.4 Schizophrenic Functions

Monteiro *et. al.* found schizophrenic functions during their work [24].

- **Schizophrenic Behaviour**

  These functions have a schizophrenic behaviour because they run different pieces of source text according to the number of input arguments. In MATLAB this situation is verified when using the combination of 'nargin' + 'if' or 'switch'. The programmer developed this function in the following way: the first 'if' checks if the function has a certain number of arguments, the second 'else if' checks if it has another number of arguments, and so on until the last 'else' checks for a number of arguments different from all the above 'ifs'. Some examples of this situation are in Listing 2.10, Listing 2.8 and Figure 2.1. These functions are called schizophrenic because they executed different pieces of source code on different 'runs' (depending on the number of arguments), like they have schizophrenia (mental illness).

```
...
argNumber = 0
if nargin == 1
```

```
4     argNumber = 1;
5 else if nargin == 2
6     argNumber = 2;
7 else
8     argNumber = 3;
9 ...
```

Listing 2.10: MATLAB example of schizophrenic function

```
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
```

[ ] – *Verification of function arguments and return value;* [ ] – *Data type specialization;* [ ] – *Memory allocation/ deallocation;*

Figure 2.1: NemoStep function. An example of Schizophrenic Functions[24]

2.4.1 Frequency of these Practices

The analysis made on 'if+nargin' and 'switch+nargin', seen in Figure 2.2, was made by J.Barrulas [2].

- **Frequency**

  There are 12.962 m-files with 'if+nargin' and 500 with 'switch+nargin', so about 20% of the repository has schizophrenia symptoms. The repository size was about 65 000 m-files. A severely large part of the repository has schizophrenia symptoms, and therefore they are very common.

- **Re-Validation**

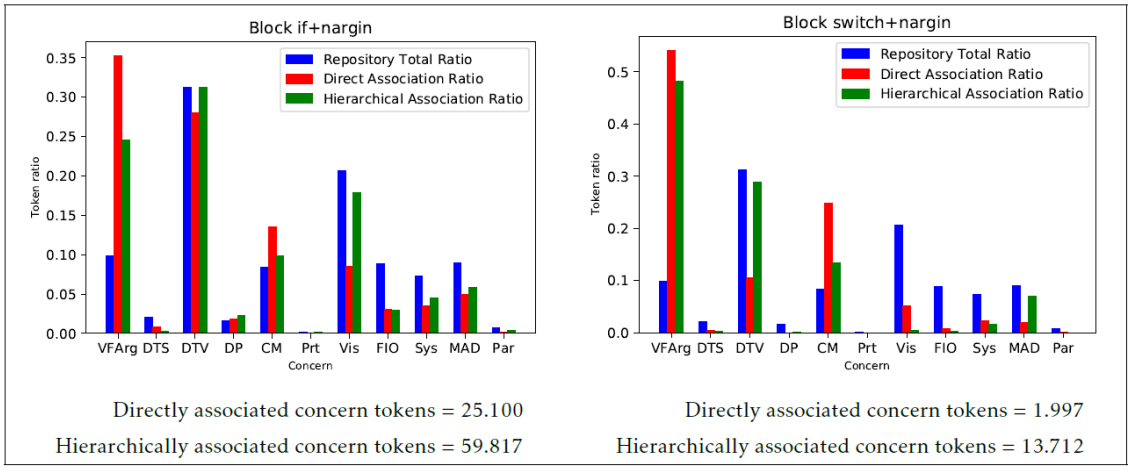

Figure 2.2: Token Ration vs Concern- 'if+nargin' and 'switch+nargin' [2]

Being this common it is a good idea to re-validate this schizophrenic functions in a bigger repository, in order to have a more firm validation. This re-validation is in Chapter 7.

Table 2.3: Language Constructs by Lexical Keyword.

Language Construct	Sub-Language Construct	Specific Language Construct	Lexical Keyword
Control Flow		If	if
		While	while
		For	for
		Switch	switch
		Try	try
		Catch	catch
		SPMD	spmd
		Parfor	parfor
		Main	beginning of file
Object Oriented Features		Class Defining	classdef
		Class Properties	properties
		Class Methods	methods
		Class Events	events
		Class Enumerations	enumeration
Boolean Algebra		AND	A&B
		OR	A|B
		AND (with short--circuiting)	A&&B
		OR (with short--circuiting)	A||B
		NOT	~A
Operators	Assignment of values to a variable		Y=5
	Unary	Unary Minus	C=-A
		Unary Plus	C=+A
	Relational	Equal to	A==3
		Not Equal to	A~=3
		Less than	A<B
		Less or Equal to	A=<B
		Greater than	A>B
		Greater or Equal than	A>=B
	Arithmetic	Plus	A+3
		Minus	A-3
		Multiply	A*B
		Multiply (Array)	A.*B
		Right Division	A /B
		Right Division (Array)	A./B
		Left Division	A\B
		Left Division (Array)	A.\B
		Exponential	A^B
		Exponential (Array)	A.^B
		Transpose (Array)	A.'
		Transpose (Matrix)	A'
Subroutines		Methods	function
		Process Terminating Functions	quit
			exit

3

Token Approach

The first Section in this Chapter references previous work necessary to discover the Token Approach. The second Section explains in some detail what is the Token Approach.

3.1 Discovery of the Token Approach

This section explains the history necessary to understand how the Token Approach was invented by Monteiro *et. al.* [3], during many years of improvements.

3.1.1 What is Modularity

There is a field in computer science, that studies a way of decomposing software systems into smaller and independent parts. The smaller and independent parts are called modules.

- **Module Definition**

  "A module is a software component or part of a program that contains one or more routines. One or more independently developed modules make up a program. An enterprise-level software application may contain several different modules, and each module serves unique and separate business operations" [33].

- **Modularity Definition**

  - "Modularity is the degree to which a system's components may be separated and recombined" [2].

  - "Broadly speaking, modularity is the degree to which a system's components may be separated and recombined, often with the benefit of flexibility and variety in use. The concept of modularity is used primarily to reduce complexity by breaking a system into varying degrees of interdependence and independence across and hide the complexity of each part behind an abstraction and interface" [39].

- **Modularity Importance**

Modularity is important to help make a system more maintainable, and a more maintainable system has several positives sides, like being cheaper to maintain and being easier to understand by programmers that never saw the system before and must evolve it.

3.1.2 Concerns

- **Concern Definition**

  "A concern is a concept or cohesive set of functionalities that, along with more concerns, define the overall behaviour of a system" [2]. In other words, a concern is a concept, big or small, of a system. For example, in MATLAB, a indicator of a concern can be the 'disp' function. This function has the concept of printing in the user's console whatever the developer has programmed that function to print.

- **Concern Relation with Modularity**

  The relation between modularity and concerns is that each smaller and independent part called modules, mentioned in the section above, ideally would contain a single concern. It is this separation of concerns in each module that is studied by the previously mentioned field of software engineering with the goal of obtaining more maintainable systems.

- **Concerns and Crosscutting Concerns**

  Each concern must be located (ideally) in only one module, without cutting across the modular composition of the system. However when this does not occur, and the concern is spread in several modules, we have, on that piece of source code, what we call a Crosscutting Concern (CCC).

3.1.3 Crosscutting Concerns

- **Crosscutting Concern Definition**

  "Crosscutting concerns are aspects of a program that affect several modules, without the possibility of being encapsulated in any of them. These concerns often cannot be cleanly decomposed from the rest of the system in both the design and implementation, and can result in either Code Scattering (code duplication), Code Tangling (significant dependencies between systems), or both" [35].

- **Crosscutting Concern Importance for this Thesis**

  The CCC concept is not very important to this thesis on it's own. However the study done on this subject is very valuable and contributes significantly to this thesis from a theoretical perspective. So it was decided that it would be useful to give some short explanations about this part of software engineering. Those studies that were read during the time elapsed ofthis thesis explain many concepts about CCCs.

- **Crosscutting Concern Important Concepts**

  One important concept is that CCCs can originate *Code Scattering* and *Code Tangling* [35]. Code Scattering is when the source text of a certain functionality is spread through the system, through several files or functions. In other words when the same functionality is spread through several modules. Code Tangling is when the code that belongs to several different functionalities is found in the same module.

- **Crosscutting Concern Importance for the Token Approach**

  Other important aspects that can be extracted from the readings done during the course of this thesis, is all the information related to the location of concerns and CCC produced by Monteiro *et. al.* [3] [23] [24]. The method used by the previously mentioned authors is the Token Approach.

  These information served as the basis for the analysis in this thesis and, consequently, for any conclusions that might be drawn from it.

3.2 What is the Token Approach

In simple terms the Token Approach is a method of identifying some concerns in MATLAB resorting to tokens. These Approach on it's own has the objective of identifying some MATLAB concerns resorting to tokens, however has it is possible to see further ahead on this chapter, the identification of some concerns in MATLAB can lead to several other fields in the Computer Science world.

- **Token Approach defined by the Inventors**

  "The approach is based on the analysis concern metrics, which capture information about concerns, whether they are modularised or not. This is in contrast to traditional modularity metrics, which capture information on the modules themselves. concern metrics are particularly suitable for supporting concern mining tasks. The main information unit are the tokens, i.e., the lexical elements extracted from a code file by means of some lexical analyzer tool" [4].

- **Token Approach: Basics**

  These authors followed the hypotheses that some tokens, specific groups of tokens, could be related to some specific concerns. In other words, they believed that some groups of tokens, composed by distinct tokens and never repeated, could be related to distinct concerns. So each group of tokens could be related to a single concern. The relation mentioned above is identification. So they believed that one specific group of tokens could identify, or at least partially, one single concern.

- **Token Approach: Usable Tokens**

The groups of tokens identified by the authors were composed by functions tokens.

"Function names, particularly from standard MATLAB libraries, comprise the most useful tokens because they are common to most or all MATLAB systems, thus providing a measure of guarantee that the technique will operate uniformly in most MATLAB systems" [4].

- **Token Approach: Function Tokens**

  Other reason they used these tokens and not others, is the high natural relation between function tokens and concerns. Because a concern is a concept of system, as it is written in the previous section, and each function represents a concept of the system. This way since each token represents a function, each function represents a concept, and each concept being a concern, they believed that following this natural sequence of representations each token cloud indicate the presence of a concern.

3.2.1 Location of Concerns Motivational Example

The example presented in Figure 3.1 and Figure 3.2 is relevant, because it clearly demonstrates how the Token Approach can be made, and also why it can be important. This example uses two versions of the same functions, one simple, and another more complex where some concerns can be identified. Therefore the Token Approach can be applied in order to identify those concerns.

- **Example Explanation**

  In the figures Figure 3.1 and Figure 3.2 we can see two versions of the same function. In Figure 3.1 we can see the simpler version of the Discrete Fourier Transform (DFT) function programmed in MATLAB. In Figure 3.2 it is represented the same function, however it has the possibility of dealing with Fixed-Point representations.

- **Second Figure Concern Tokens**

  In this second figure we can realize that there are two concerns in the same function. One being the main one, the DFT main functionality, and the other one represented by the functions 'quantize' and 'quantizer', in green and yellow. This last concern was added to deal with the Fixed-Point representations.

  The example shows the identification of a concern, Data type handling - Specialization, resorting to the tokens 'quantize' and 'quantizer'. This mapping can be seen in Figure 3.3.

- **Second Figure Symptom**

  The biggest symptom present in Figure 3.2 is Code Tangling. This is considered a symptom because the function 'quantize' and 'quantizer' are present 8 times each,

in just 22 Lines of Code, and in the presence of the main concern of the function, the DFT main functionality.

- **Function Tokens are Indicators of Concerns**

  Now it is possible to understand better how and why the functions 'quantize' and 'quantizer' are indicators of the presence of a concern. Keep in mind that they are indicators of the presence of a concerns and that they can be identified by their MATLAB token using the Token Approach.

- **Another Example**

  Another concern location example is in Figure 2.1, in this example each concern has it's own color. A conclusion about concerns and crosscutting concerns to withdraw from these two examples, is that several concerns can be present in the same function in just a few lines of source text.

```
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
```

Figure 3.1: Simpler DFT [5]

```
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
```

Figure 3.2: Complex DFT [5]

3.2.2 Locating Concerns

Locating concerns and CCCs, has it is written on Section 1.1, is not yet very developed for the MATLAB language. However several studies have been done by some scientists [3] [23] [24], in order to achieve scientific advancements on this area of expertise of the Computer Science field.

- **Concern Token Mapping**

  However many conclusions can be extracted from these author's works, one relevant for this thesis is surely the final table elaborated by B. Jota [10]. This table is presented in the Figure 3.3. This table was first developed by Monteiro *et al.* [24]. However due to some more years of work and the help of other scientist/students

this table was further developed to the table that is presented on Figure 3.3. This table shows the mapping between the concern tokens and the concerns. In this table each concern token may indicate the presence of the associated concern.

- **Token Approach Importance for this Thesis**

  The usage of lexical tokens to identify some concerns is very important to this thesis because Language Constructs can also be identified by resorting to lexical tokens, as it is depicted on Table 2.3. So with some adaptions to the tool used to identify concerns and CCCs, that tool can also be used to identify Language Constructs. This is true because that same tool, *CrossCutting Concern Explorer (CCCExplorer)*, has already been extended in a similar purpose. It was extended by J.Barrulas [2], in order to add code block functionalities (CBLC). The next Chapter will provide and develop further information about *CCCExplorer*.

Concern	Sub-concern		Tokens
1. Verification of function arguments and return values			nargchk, nargin, nargout, nargoutchk, varargin, varargout
Data type handling	2. Specialization		double, fi, int8, int16, int32, int64, quantize, quantizer, single, uint8, uint16, uint32, uint64
	3. 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
4. Dynamic properties			eval, evalc, evalin, feval, inline
5. Console messages			annotation, assert, disp, display, error, last, lastwarn
6. Printing			orient, print, printdlg
7. Visualization	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
8. File I/O			diary, fgetl, fgets, fopen, fprintf, fread, fscanf, fwrite, hgload, hgsave, load, save, saveas, uisave
9. 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
10. Memory allocation / deallocation			delete, global, ones, persistent, zeros
11. Parallelisation			cancel, codistributed, codistributor, gather, labindex, labProbe, matlabpool, numlabs, parfor, pload, pmode, promote, resume, sparse, spmd, subsasgn, subsref

Figure 3.3: Concern Token Mapping

4

CCCExplorer: Token Extraction Tool

In this chapter there is a description about the tool that applies the Token Approach in an automatic way. This description includes background, input, output and a general overview of the system.

4.1 What is CCCExplorer

CCCExplorer is a tool written in Java. It was developed by Monteiro et al. [23] as a concern detection and token-based aspect mining tool for MATLAB systems [2]. CCCExplorer is a tool that applies the approach explained in the last chapter in an automatic way. The 'CCC' in CCCExplorer stands for Crosscutting Concern.

- **Automation**

  CCCExplorer is a standard java system, with src, main and test folders. Currently it does not use a specific project automation such as Ant, Maven or Gradle. It uses Java for this purpose.

- **Tokens**

  This system generates sequences of tokens (lexical elements, like words, punctuation and grammar rules), and from these sequences of tokens it allows the computation of metrics related to the source text analysed [2].

- **Extensions**

  There was a need to extend the system with more features. All of these improvements and feature additions were made over a long period of time by various individuals.

4.2 Background

This java tool was first proposed by Monteiro *et al.* [3]. Later, B.Jota [10] enhanced the original table to become the one shown in Figure 3.3 and added support for it in CCCExplorer. Making CCCExplorer more complete in terms of concerns identification and location. After that, Relvas [28] edited it to include the intelligent repository features [29]. These features allow CCCExplorer to generate 'Insert' SQL commands, this way making it easier and faster to generate a SQLite database. It supported code blocks (CBLC) following one change made by J.Barrulas [2]. So, it was possible to query the generated database to determine the most prevalent concerns inside each code block.

4.3 Overview

The ability of CCCExplorer to create SQL files with insert statements, which are used to create databases and then be used in the analysis presented in Chapter 7, is the main focus of this thesis. This ability is obtained by using 'MainGenSQL.java'. With a broad perspective, this main performs all the parsing and processing of a repository of mfiles and writes SQL files with insert statements based on the information about the identifiable tokens found in those mfiles.

- **Information Splitting**

  CCCExplorer splits the information by line, mfile and toolboxes, in order to have all the information stored relating to the tokens exact location in the repository. This information is stored in direct order, in other words, the order of occurrence of each token in the mfile is never lost by CCCExplorer.

- **Parser**

  The way CCCExplorer does this transformation from lines of code to insert statements, mostly occurs in the parser, and it's auxiliary class. The parser reads each token in the mfiles and stores them in iterators. Each iterator as a specific function, like storing each usable line of source text, or each token that is a language construct.

- **Parsing**

  When parsing a mfile first the system stores the name of the toolbox, and then the name of the mfile, and adds these names to the respective iterators. Afterwards it starts to process the mfile line by line.

- **Parsing: Line by Line**

  Each line looks for the presence of language constructs, variable names, and concerns. These details are saved in the appropriate iterator. CCCExplorer's mechanism for identifying these tokens is based on enumerations. This enumeration

stores all tokens that must be identified, such as language constructs or concern tokens. When the parser finds a token that belongs to one of these enumerations, it stores it in the appropriate iterator.

- **Parsing: Statistics**

  These tokens have specific classes, specially the concern tokens. The enumerations relate to objects that besides the keywords of each token contains statistics and other numeric values related to the frequency and relevance of that same token in the repository.

- **Parsing: Whole Lines of Source Code**

  The parser also stores whole lines of source text, commented and non-commented, however it does not store blank lines. These lines are saved in order to reconstruct the mfile if necessary, using queries on the database after it has been populated.

- **Generation of SQL Files with 'Inserts'**

  The system will begin producing the SQL files after analyzing the mfile. The iterators that the parser had previously filled are used for this generation. Each insert in the SQL files is created by applying algorithms to the above mentioned iterators.

4.4 Input

The input of CCCExplorer is essentially a repository. There must be toolboxes and mfiles inside each toolbox in this repository. A repository is used as an input because each mfile contains the tokens that CCCExplorer needs to parse.

- **Input Structure**

  The input structure is a very straightforward repository layout. This layout must include a single folder containing the entire repository. Because CCCExplorer begins by reading this folder. There must be several toolboxes (folders) in this single folder. These folders contain entire MATLAB systems; each folder may contain more than one system. Each toolbox must contain the source code for the system(s). This system source code may include multiple folders or, in some cases, only mfiles.

- **New Toolboxes in the Repository**

  One peculiar feature of CCCExplorer's input and run is that the system must be re-executed every time new mfiles are added to the repository. It is not possible to simply run one toolbox and add the results to the previous repository results due to the system's design. This is obviously disadvantageous because running the system on a large repository may take several days. The Section 9.2 addresses this situation in order to provide one solution.

4.5 The Output in SQL

The sql files CCCExplorer generates are:

- block_types.sql, this file contains the insert instructions to then populate the static table 'block_types'. This table contains all code block types indexed.
- concerns.sql, this file contains the insert instructions to then populate the static table 'concerns'. This table contains all concern types indexed.
- blocks_mfiles.sql, this file contains the insert instructions to then populate the table 'blocks_mfiles'. This table contains all the code blocks indexed, that the CCCExplorer found in the entire repository.
- lines_comments.sql, this file contains the insert instructions to then populate the table 'lines_comments'. This table contains all commented lines that CCCExplorer analysed.
- lines_mfiles.sql, this file contains the insert instructions to then populate the table 'lines_mfiles'. This table contains all non commented lines that CCCExplorer analysed.
- mfiles.sql, this file contains the insert instructions to then populate the table 'mfiles'. This table contains the list of MATLAB files that CCCExplorer analysed.
- tokens.sql, this file contains the insert instructions to then populate the table 'tokens'. This table contains the list of all different tokens that CCCExplorer found while it was analysing the repository.
- toolboxes.sql, this file contains the insert instructions to then populate the table 'toolboxes'. This table contains the list of all MATLAB toolboxes that CCCExplorer analysed, in the format of string. So in other words, if a folder is inside another folder it would be like this: Folder1\ Folder2.
- lc_types.sql, this file contains the insert instructions to then populate the static table 'lc_types'. This table contains all language constructs types indexed (New Table).
- lc_mfiles.sql, this file contains the insert instructions to then populate the table 'lc_mfiles'. This table contains all the language constructs indexed that the CCCExplorer found (New Table).
- lines_tokens.sql, this file contains the insert instructions to then populate the table 'lines_tokens'. This table contains the list of each combination of line and token (Modified Table). The modification to this table was addiction of the 'lc_type' as a column.

To conclude the explanations about *CCCExplorer*, it is good to have in mind that a database can be easily generated by the end of the system's run. More about the database generated by CCCExplorer in the next Chapter.

5

CCCExplorer's Database

This chapter exposes the most important aspects of the database that can be generated from CCCExplorer. Namely: entities, relations, how it is useful, proofs of usefulness, and queries that are useful for this thesis and why.

5.1 What is CCCExplorer's Database

CCCExplorer creates this sqlite database from a repository of mfiles. In other words, if we use CCCExplorer to retrieve sql files containing insert statements from a specified repository of mfiles, these sql files will be used to populate this database. The database must first be created, empty, and then populated with the sql files from CCCExplorer.

- **Database Schema**

  The database schema in is Listing I.1.

- **Repository Used**

  In the next chapter it is explained the origin of the repository used as input for CCCExplorer, see Section 6.4.

- **Database Usefulness**

  Further on in this chapter, it is explained why and how the database is useful. However, in a nutshell, this database can be useful for analyzing a large repository of mfiles for certain characteristics. To be present in the database, these characteristics must be present in CCCExplorer. Concerns, non-CBLC, CBLC, variable names, and function names are currently available in CCCExplorer.

  If someone wants to analyze a large repository of mfiles for some of these characteristics, this database can make the work much easier.

id_block	block_type	parent_block	id_mfile
Filter	Filter	Filter	Filter
0	1	-1	0
1	2	0	0
2	5	1	0
3	3	2	0

Figure 5.1: Excerpt of block_mfiles table

5.2 Elements of the Database

The database created contains several tables and some key features that deserve to be explained, and are explained next.

5.2.1 Entities

- 'block_types' table contains the types possible for each Language Construct that is a code block, and that were identified by J.Barrulas [2]. This table contains an internal database identifier and the name of that type, for example 'if', 'for', 'class defining', etc.

- 'block_mfiles' contains a list of all occurrences of language constructs that are code blocks, and these are also indexed by an internal database identifier. Each code block identified is represented by: ID, Type_ID, parent and mfile_ID. The graphical example is in Figure 5.1.

- **Block_mfiles Parent Block Explanation**

  In the Figure 5.1, we can see some example of language constructs that are code blocks and how they are represented in the database. One of the tuples has ID = 2 and is of type 5. If we check the block_type table we can see that 5 is the identifier of 'for'. This code block is present in the mfile that contains the ID 0 in the database.

  We can also see a code block with the ID = 3, that is of type 3, this means that it is an 'if'. It's parent is the code block above mentioned, with ID = 2. This means that we have an 'if' inside a 'for'. For a better understanding, see the excerpt of the mfile with index 0 in the database, Listing 5.1, which contains the mentioned 'for' and 'if'.

```
1 ...
2 j = 1;
3 for i = 1:popsize
4     if chromosome_rate(i,1) ==1
```

```
5           index_cro(j,1) = i;
6           j = j+1;
7       end
8   end
9   ...
```

Listing 5.1: MATLAB excerpt of crossover.m (id_mfile = 0)

- 'lc_type' and 'lc_mfiles' tables have a very similar behaviour to the above mentioned tables, 'block_type' and 'block_mfiles'. With a single difference, the 'parent' field. This is reflected in the table 'lc_mfiles', that does not contain this field, because, has it is seen in Chapter 2, the language constructs that are not code blocks do not encapsulate any source text, therefor they can not encapsulate any other language constructs. For this reason the parent field would be very irrelevant.

- 'toolboxes' contains a simple list of all toolboxes analysed by CCCExplorer indexed by an internal database ID.

- 'mfiles' contains a list of all mfiles analysed by CCCExplorer, with their respective internal database ID, and with the ID of the toolbox they belong to.

- 'concerns' contains a simple list of all concerns that CCCExplorer can identify, also indexed with an internal database ID.

- 'tokens' contains the list of all different tokens found by CCCExplorer while analysing the repository. Each different token has an internal database ID, the 'string' of the token, and the ID of the concern that CCCExplorer considers it must belong to.

- 'lines_comments' is not used in this thesis, however it is provided a short description. Each comment has an internal database ID, the ID of the mfile in which it was found, the number of the line of that mfile, and finally, the actual 'string' of the line.

- 'lines_mfiles' has the exact same structure has the above mentioned table, however the information stored in this table is not related to commented lines but is related to non-commented lines.

- Finally and most importantly, the table 'lines_tokens'. In simple terms this table stores every occurrence of each token in each line.

  In more specific and technical terms, the first column is the ID of the line in which the tokens appear. The second is the ID of the CBLC in which the token is encapsulated. The third field is the ID of the type of non-block language construct; if the tokens do not fall into this category, a 0 is stored. Fourth, the token's ID in the table 'tokens'. The fifth and sixth numbers are the starting and ending columns of the token's text in the mfile line.

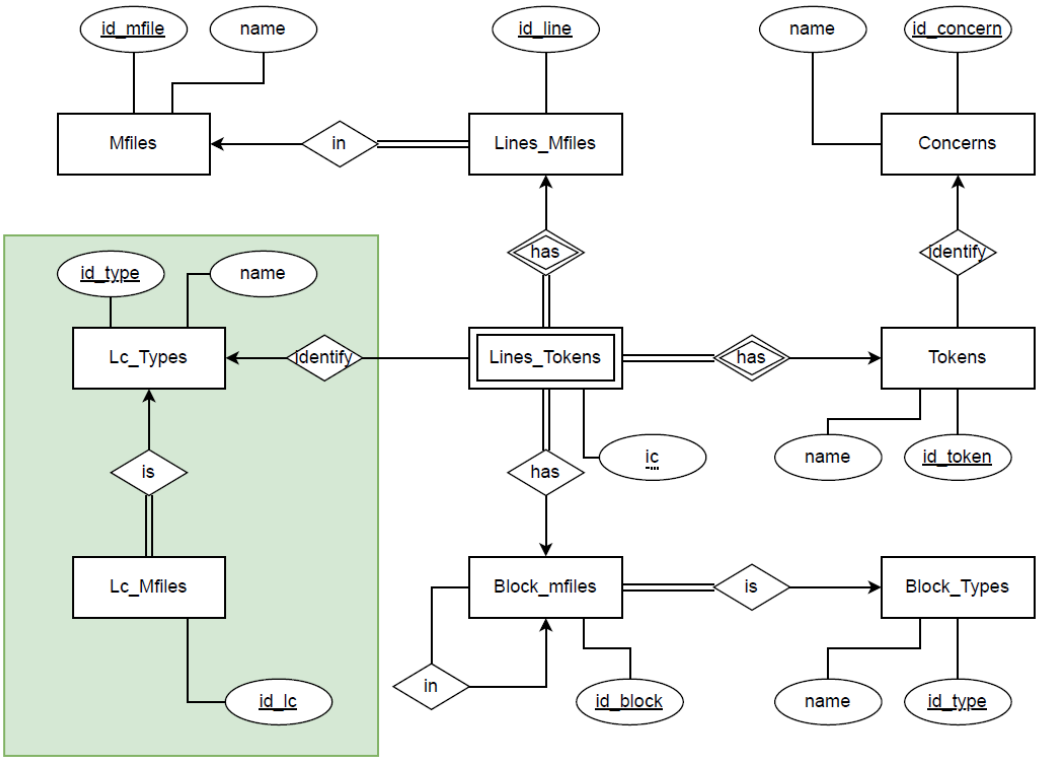

Figure 5.2: ERD for the new Database (in green is the part created for this thesis) (original design from SEKE [29])(adapted from J.Barrulas [2])

The table's key is (idLine; idToken; startingColumn). If there is a token in a line, CCCExplorer will generate an insert instruction. Even if the token appears twice in the same line, it will generate two insert instructions because the starting column is different.

- **Important Detail**

  One aspect to have in mind about the database CCCExplorer generates is that CCCExploer is designed to always have a code block in each tuple of 'lines_tokens'. Because it always starts the parsing of a mfile by identifying the 'main' block type in the beginning. This code block language construct mostly represents the beginning of the mfile, with the purpose of always having a token related to one code block language construct. Therefore each token found by CCCExplorer and present in the table 'lines_tokens', is at least related to the 'main' code block of the mfile it belongs to.

- **Database ERD**

  The database ERD can be seen in Figure 5.2.

5.2.2 Relations

- 'Mfiles - Lines_Mfiles' are connected by the primary key of 'Mfiles', 'id_mfile'. This primary key is present in 'Lines_Mfiles' because each line belongs to a single mfile. So in 'Lines_Mfiles' the 'id_mfile' is present in each tuple representing the mfile present in 'Mfiles'.

- A similar situations occurs in 'Lines_Mfiles - Lines_Tokens', but with the 'id_line'. Each token present in 'Lines_Tokens' is present in a single line of source code, therefore the primary key of 'Lines_Mfiles', 'id_line', is present in each tuple of 'Lines_Tokens'.

- 'Tokens - Concerns' are connected by the primary key of 'Concerns', 'id_concern'. Because in this database each token must have at least one concern, even if it is a variable name. There are special "concerns" in the database for these type of situations.

- 'Lines_Tokens - Tokens' are connected by the primary key of 'Tokens'. Since each different token is only stored once in the table 'Tokens', each token occurrence in 'Lines_Tokens' (each tuple), has present the primary key of that same token in the table 'Tokens'.

- 'Lines_Tokens - Block_Mfiles' are connected by the primary key of 'Block_Mfiles'. CCCExplorer is designed to always identify tokens inside a code block. So each token occurrence in 'Lines_Tokens' has the primary key of the code block in which it is encapsulated('id_block').

- 'Blocks_Mfiles - Block_Types' are connected by the primary key of 'Block_Types', 'id_type'. This is due to each code block having a specific type that is present in 'Block_Types'.

- 'Lines_Tokens - Lc_Mfiles' are connect by the primary key of 'Lc_Types', 'id_type'. When a token occurrence in 'Lines_Tokens' is a Non Code Block Language Construct, the type of the non-CBLC is stored in 'Lines_Tokens' else a 0 is stored.

- 'Lc_Mfiles - Lc_Types' are connected by the primary key of 'Lc_Types'. Because each occurrence of a non-CBLC has a specific typing.

5.3 Proofs of Usefulness

This database can be used to perform analysis on a large collection of MATLAB mfiles. The results of each analysis are the most useful information that can be derived from it. This database organizes an entire repository of mfiles in SQLite tables, making analysis over that same repository far more convenient. This situation demonstrates that the

database's usefulness is dependent on how it is used. The more useful the analysis performed on the database and the more conclusive the results obtained from that analysis, the more useful is the database.

5.3.1 Analysis

The analysis already made over the database generated by CCCExplorer were not numerous. However there is the analysis on code blocks and schizophrenic functions performed by J.Barrulas [2].

- **Code Blocks**

  As it can be seen in Table 2.3, code blocks are also Language Constructs, more specifically CBLC. J.Barrulas [2] analysed code blocks using CCCExplorer. He firstly understood and edited CCCExplorer to support code block functionalities. Then upon the extension of CCCExplorer the generated database was able to be queried in order to find what are the most common concerns in each code block.

  The method he used to finalize his studies about code block was to present bar charts with information of the most common concern types in each code block. For example Figure 5.3. On this image we can see the amount of each type of concerns that are inside the 'While' code block, in a repository with over 65 000 m-files. Right now we have a much bigger repository with over 450 000 m-files, so it is believable that more conclusive studies can be made resorting to this newer and larger repository.

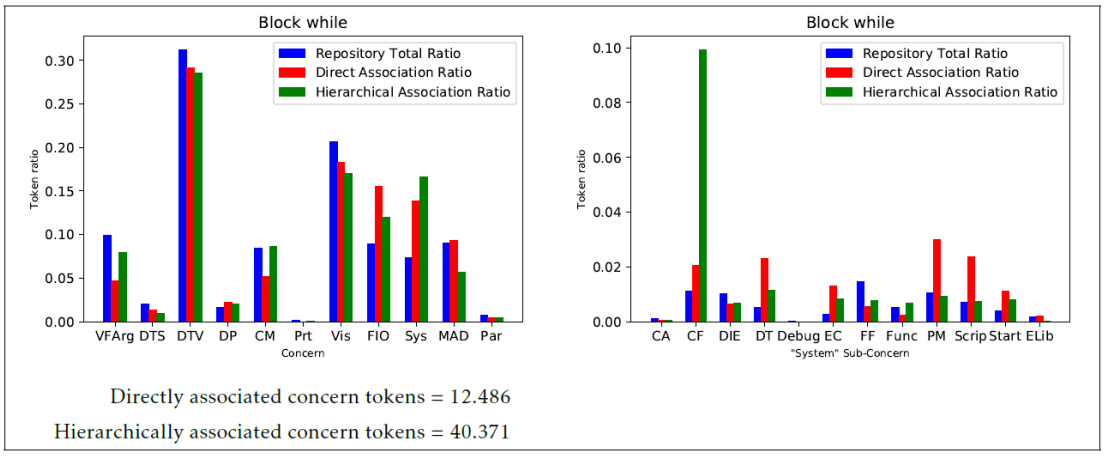

Figure 5.3: Bar Chart showing the repository, While code block and it's relation to concerns [2]

- **Schizophrenic Functions**

  Another analysis that was already made is the one mentioned in Section 2.4. This analysis has lead to the conclusion that schizophrenic functions are common and deserve to be further investigated. Once again proving the usefulness of the database.

5.3.2 Queries

The most important queries for this thesis are the ones related to the code blocks and to schizophrenic functions. Because they can relate highly to Language Constructs and also because they may serve as a base for future programming conducts analysis (coding anti-patterns). However some changes are necessary, the core concept of each query remains nearly intact. This queries were valid on the older version of the Database, some names have been changed, see Section 6.2. In order to maintain compatibility with the older version of the Database, views were created with the older names.

- **Repository Content Query**

  This query, Listing 5.2, is important to answer RQ2, and to have some further guidance on what direction to follow when doing the analysis. It will provide information about what are the most common Language Constructs in the repository (when also applicable to non-CBLC).

  Because the result of this query could be a list of LCs ordered from most common to least common, it is also important to provide some additional meaning to the answers of the following RQs. Because if one of the next RQs' results is at the top of this ordered list, it means the result is even more meaningful because the Language Construct (LC) in question is very common.

```
1 select id_type , count(id_block)
2 from blocks_mfiles_OLD inner join block_types_OLD on id_type =
      id_block_type
3 group by id_type;
```

Listing 5.2: SQL Query Repository Content adapted from J. Barrulas[2]

- **Relation Language Construct/Concern Query**

  This type of queries, Listing 5.3, are important to find what are the most common concerns related to each Language Construct. They will help answer RQ3. "id_type = 9", the 9 has the purpose of representing try/catch code blocks by being it's identifier. "id_concern = 6", the 6 has the purpose of representing CM concerns by being it's identifier. When non-CBLC features are added to the database, changing these numbers allow to find the relationships between each Language Construct and each concern type. This has been shown in Section 2.3.

```
1 select distinct id_mfile
2 from blocks_mfiles_OLD inner join lines_tokens_OLD using(id_block) inner
      join tokens using(id_token)
3 where id_type = 9 and id_concern = 6
```

Listing 5.3: SQL Query Relation Language Construct/Concern adapted from J. Barrulas[2]

- **Relation Language Construct/Concern Token Query**

  In case there is a high correlation to a concern, there may be the need to go in to further details. The way of doing this is to find relations not to the hole concern, but to the several specific concern tokens. The query example is in Listing 5.4. It will also help answer RQ3.

  To then access the intensity and the relevance of each relation (a percentage), it will be necessary to query the database about how many of those specific tokens, or total amount of tokens of that same concern are in the database. In the case of this query it is being analysed the 'nargin' (1125) and 'if' (3) relationship.

```
1 select count(id_block)
2 from blocks_mfiles_OLD inner join lines_tokens_OLD using(id_block)
3 where id_type = 3 and id_token = 1125
```

Listing 5.4: SQL Query Relation Language Construct/Concern Token adapted from J. Barrulas[2]

- **Schizophrenic Functions**

  The most important queries to re-validate schizophrenic functions are: Listing 5.5, Listing 5.6, Listing 5.7. These queries are testing for the 'if+nargin' relation. However similar should be also done to the 'switch+nargin' relationship. These queries are partial fulfillment of RQ5 and RQ6.

  The query, Listing 5.7, uses hierarchically associated blocks. These type of associations are explained in Section 5.2.

```
1 select count(distinct id_mfile)
2 from blocks_mfiles_OLD inner join lines_tokens_OLD using(id_block)
3 where id_token = 1125 and id_type = 3
```

Listing 5.5: SQL Query Number of files with 'if+nargin' adapted from J. Barrulas[2]

```
1 select count(*)
2 from (
3 select distinct id_block from blocks_mfiles_OLD inner join
      lines_tokens_OLD
4 using(id_block)
5 where id_token = 1125 and id_type = 3
6 )
7 inner join lines_tokens_OLD using(id_block) inner join tokens using(
      id_token)
8 where id_concern between 1 and 35;
```

Listing 5.6: SQL Query Number of directly associated concern tokens to 'if+nargin' adapted from J. Barrulas[2]

```
with parents as
(
    select id_block , id_type , id_parent_block
    from (
    select distinct id_block , id_type , id_parent_block
    from blocks_mfiles_OLD inner join lines_tokens_OLD using(id_block)
    where id_token = 1125 and id_type = 3
    )
union all
    select b.id_block , b.id_type , b.id_parent_block
    from blocks_mfiles_OLD b inner join parents p on b.id_parent_block =
    p.id_block
)
select id_concern , count(id_token)
from parents inner join lines_tokens_OLD using(id_block) inner join
    tokens
using(id_token)
group by id_concern
```

Listing 5.7: SQL Query Number of hierarchically associated concern tokens to ’if+nargin” adapted from J. Barrulas[2]

6

Implementation

This and the following chapters aim to further develop the main topic of this thesis. This chapter, specifically, advances with information regarding all the implementation and necessary means to follow trough with the analysis. This information is about CCCExplorer, the modifications done to it. It is mentioned the database that can be generated by CCCExplorer and the modifications done to it. The final section is about the repository used to generate the DB.

6.1 Features Added to the Extraction Tool

The main features added by the author to CCCExplorer were the non code block Language Constructs features. These features mostly generate all the information necessary to then proceed with the analysis about the relations of Language Constructs with concerns. Mostly these information is about the indexing of each Language Construct, its typing, and their exact location in each mfile of the repository analysed. In order to apply these changes some classes had to be modified and others had to be created.

- **New Classes**

  The classes 'LC.java' and 'LCType.java' have been added to the system. These classes serve to create objects to be stored in iterators. These iterators are used to store and output tokens in the SQL files.

- **Enumerations and Switch Cases**

  Enumerations and Switch Cases with all non-CBLC have been added to the following classes: 'Token.java', 'Parser.java', 'LexicalElem.java' and 'LexicalAnalyser.java'.

- **Class: LexicalAnalyser.java**

  The 'parser.java' contains the parser of the system. This parser uses an auxiliary class to do the parsing, this class is named 'LexicalAnalyser.java'. It is in this auxiliary class that each char of an mfile is analysed, stored in memory and selected by their role in the system.

- **Method: analyseLine**

  The auxiliary class has a method named 'analyseLine', and has the name implies, analyses each line of a mfile. In the beginning of the implementation it was necessary to add all non-CBLC tokens to several classes, due to the construction of the system. One of those classes was this auxiliary class and exactly in this extensive method. It was difficult to modify this method due to having several lines of code, and also because it is not a trivial method. This method has several token cases that range between a one char token to two char tokens, from special cases like the apostrophe token, that can be the beginning of a string or the transpose of a matrix, but also includes cases like numbers, digits, and names of variables and functions. The changes made to it were to modify the source text in order to add all non-CBLCs to the memory of CCCExplorer. Theses changes required a lot of understanding of the previous version of the source text, because they had to be added in the exact place where they were suppose to be in order to produce an output suitable for analysis.

- **Class: SQLProducer.java**

  Another big change to the system was to add the non-CBLCs to the output of the system, in the class 'SQLProducer.java'. The files 'lc_type.sql' and 'lc_mfiles.sql' were added, and another, the 'lines_tokens.sql', was modified by adding the column 'lc_type' to it.

- **Method: 'processParser'**

  This class also includes a sizable method called 'processParser', which contains some difficult-to-understand algorithms, but it was necessary to comprehend and study these algorithms in order to move forward with their modification and get the desired output from them. The use of a "hard-coded" integer in this method to represent the number of characters in the path where the repository is stored on the computer is another instance. This was a challenging situation to handle because the programming was not very elegant.

- **Class: MainGenSql.java**

  The main file (MainGenSql) was used during the course of this thesis, and was altered. It was added new sql files: the 'lc_types.sql' file and the 'lc_mfiles.sql' file.

- **Transactions**

  While the source text that produced the output was being extended it also added the SQLite keywords 'Begin' and 'Commit' to the beginning and end of each file, respectively. This adding of keywords was necessary in order for the database to execute the 'inserts' in the same transaction. If this was not the case the database would take several weeks to be populated. The other option was to add these keywords by hand, however it was necessary to scroll through millions of lines of 'inserts', specially in the file 'lines_tokens.sql'.

- **Biggest Challenges**

  During all the modification to the source text of the system, the biggest challenge was definitely to understand how the system operated from the first reading of a mfile until the production of the output. It took several weeks, and lectures, to completely understand how the modifications should be made, in order to produce the desired output.

  In terms of programming the biggest challenge was to modify the above mentioned method's algorithm, in the auxiliary class of the parser, due to being a very complex method.

- **Good Programming Practices**

  All the above mentioned tough criticism to the system are being investigated in a parallel thesis to this one [8], also oriented by professor Miguel Monteiro.

  It is taught in the informatics community that good programming practices are very important to the maintainability of a system, so that a system could be easily modified and understood by other programmers in the future. This was proven to be true due to the working experience on this system.

6.2 Improvements to the Database

In terms of new tables and attributes the tables 'lc_type' and 'lc_mfiles' were created, and the attribute 'lc_type' was added to the 'lines_tokens' table. The biggest improvements to the database, and not new information related to Language Constructs, were made in order to have a faster, more efficient and also a more maintainable analysis. These changes are the following:

- **Create Table Files**

  The sql files that contained the create table instructions were created by hand to then generate an empty database, these schema is in Listing I.1. These schema used to have different names for the same attribute in different tables. This was fixed.

- **Transactions**

  The addiction of transactions to the output of CCCExplorer, these transactions make the filling of an empty database much faster, in more specific units, weeks faster.

- **Indexes**

  These indexes make nearly every query faster. Being the most substantial change in performance the index that makes the cartesian product of the 'lines_tokens' table faster. Several indexes were created in order to make the queries used in the analysis faster. Some indexes only made those queries faster by minutes, others possibly made those queries faster by days, or even weeks. These indexes were

mostly all created in the 'lines_tokens' table. This table is by far the longest table in the database, having 294 652 847 entries. With so many entries, and being the most important table for the analysis it was extremely necessary to make these indexes. The indexes created are in Listing I.2.

- **Views**

  Views were created in order to have simpler and more maintainable queries. The views created are in Listing I.3. These views are non-materialized views because SQLite does not support this feature.

6.3 Validation of New Features

The new features added were validated by hand and with the use of a testing system.

- **Testing System**

  The above mentioned testing system checked if all the Language Constructs were correctly identified, if they were by order, and also if the DB was coherent with itself. In this case if the table 'lc_mfiles' was coherent with the table 'lines_tokens', in other words, if the order and typing of each Language Construct was coherent in both table. In this test about 200 mfiles were used.

- **Testing by Hand**

  The testing by hand was made resorting to about 50 mfiles. The system was ran, and afterwards the output was saved. On another front, results were made by hand, for the same set of mfiles. Then both results (CCCExplorer's output and the results made by hand) were compared to check for correctness.

- **Result**

  The system passed all tests, when taken in consideration one error.

- **Error**

  This error is related to the identification of strings. When there are many apostrophes starting a string, allowed by MATLAB but not taken into consideration in CCCExplorer, the system does not correctly identify it as a single string. In these type of scenarios it can identify LCs inside the string. This situation, while not easily identifiable in previous versions of CCCExplorer, was identified during this thesis, after some years, with the addiction of non-CBLCs to the system and some testing.

- **Error Contributions**

  A list of occurrences of this error was made and delivered it to a college that is investigating this situation [8].

6.4 Repository used to Generate the DB

The repository was originally designed to test CCCExplorer, in the context of Cardoso *et. al.* [3]. At this time a small amount of mfiles and toolboxes were gathered to make a small repository.

- **First Iteration of a large Repository**

  Afterwards Monteiro and Marques [24], continued their work in the MATLAB language, and with time there was a need for a bigger repository. With this need emerged a ~65 000 mfile repository from websites like SourceForge and GitHub.

- **Second Iteration of a large Repository**

  After some years Monteiro and Marques [24] still need a bigger sample of the MATLAB language in order to better analyse the overall community. They asked for the help of a student in 2020, João Santos [30]. This student downloaded from the web a repository of ~400 000 mfiles.

- **Contribution**

  At the start of this thesis both repositories where checked, they were different. In fact, completely different. All mfiles and toolboxes were different in both repositories. This way it was decided to join both repositories, and resulted a ~450 000 mfile repository.

- **Newer Repository Characteristics**

  About this latter repository it is a good idea to give the reader a general idea about the most superficial details on the repository. This way the reader can have the ability to criticise on his/her own the relevance of all phases of the analysis. It is possible to do this by comparing the results of each analysis with the general details of the entire repository.

  The repository has 452 983 different mfiles, and 11528 different toolboxes. Each toolbox has an average of 39,3 mfiles in it, with a standard deviation of 211.

7

Repository Analysis

This chapter aims to provide a characterizations study of the MATLAB language, using the largest and most recently obtained repository mentioned in Section 6.4. This characterization study is concerned with: OO Systems, the most frequently used LCs, the relationships between LCs and concerns, and the relationships of LCs to each concern token, a re-validation of schizophrenic functions, and an analysis of the relationship between schizophrenic functions and the OO paradigm.

7.1 Analysis on Object Oriented Systems

The SQL queries in Listing I.4 were used to answer questions like which are the most common OO features.

The analysed categories in this section are the following:

The first category is OO features (OO mfiles), the second category is inheritance (inheritance mfiles), which is a subset of the first, and the third category involves inheritance and libraries (inheritance mfiles related to libraries), which is a subset of the second group. The inheritance group comprises all classes that are sub-classes and are declared as such, while the third group contains all classes that are sub-classes of a MATLAB library.

- **'classdef' CBLC**

  The keyword 'classdef' is used in source text files to express OO features in MATLAB. This keyword in MATLAB represents the beginning of a class, as seen in Chapter 2. All OO features must be contained within this CBLC, that starts with the token 'classdef'. So, if it was possible to determine the number of these keywords/CBLCs in the repository, it would also be possible to begin analyzing the frequency of the OO features in MATLAB.

- **OO features**

  Since it was possible, it was also possible to determine the number of mfiles that contained OO features, the number of toolboxes that also contained OO features,

as well as the mean and standard deviation of the number of mfiles that contained OO features in their toolboxes.

- **Inheritance**

  The same analysis can be applied to the use of inheritance. Inheritance is represented by the character '<' in front of 'classdef' [1], as stated in Chapter 2. Therefore the analysis is extremely similar because the less symbol is an LC and the specific keyword 'classdef' is a token. In this manner, the same analysis can be used, but with the '<' symbol taken into account.

- **Visibility**

  How many property blocks and method blocks have visibility set to Public, Private, Protected, or are set by a List was a different factor to consider when analyzing OO features. These results were obtained by running a query that revealed the frequency with which the tokens Access, GetAccess, and SetAccess were directly followed by Public, Private, or Protected within the CBLCs class properties or class methods. The Chapter 2 includes an explanation of these attributes.

  It was also necessary to calculate the number of defaults and the number of times it was set by a List. More information on these values further ahead in this section.

7.1.1 Number of Classes in the Repository

In terms of the number of classes, the results are in Table 7.1. These Results are discussed in the next chapter.

- **Figure 7.1 Illustration**

  The graph in Figure 7.1 depicts the most common distributions of the number of OO mfiles per toolbox. The first bar from the left, for instance, reveals that there are 647 toolboxes in the repository with exactly one OO mfile.

- **Figure 7.1 Relevant Details**

  Increasing the number of OO mfiles in the toolbox causes a decrease in frequency, which is followed by a stabilization, as shown in the graph. It is evident that, in terms of the number of OO mfiles, the less complex toolboxes are more prevalent than the more complex ones.

Table 7.1: Number of Classes Results

Result	Value
Mfiles with Classes (% Repository)	14156 (3,13%)
'classdef' ocurrences	14193
Average classes per mfile	~1
Toolboxes with Classes (% Repository)	1424 (12,35%)
Average OO mfiles per OO toolboxes	9,941
Standard Deviation OO mfiles per OO toolboxes	32,845

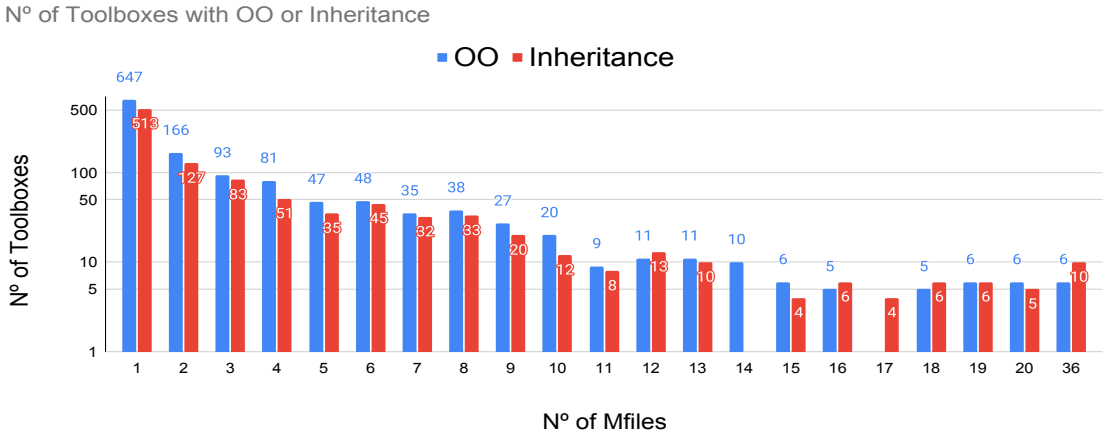

Figure 7.1: (Y axis) Nº of Tooboxes (logarithmic scale) vs (X axis) Nº of OO mfiles (Blue) and Inheritance mfiles (Red). Each bar (Blue) represents the number of toolboxes containing a particular number of OO mfiles. Each bar (Red) shows the number of toolboxes containing a particular number of Inheritance mfiles.

7.1.2 Inheritance Results

Based on how inheritance is used, the results are depicted in Table 7.2.

- **Figure 7.1 Relevant Details**

  The graph in Figure 7.1 shows a decrease in frequency as the number of inheritance mfiles in the toolbox increases, followed by almost a stabilization. As it is also seen in the blue bars (OO mfiles).

Table 7.2: Inheritance Results

Result	Value
Mfiles with Inheritance (% Repository)	11833 (2,61%)
Inheritance ocurrences	11862
Average Inheritance classes per mfile	~1
Toolboxes with Inheritance (% Repository)	1152 (9,99%)
Average Inheritance mfiles per Inheritance toolboxes	10,272
Standard Deviation Inheritance mfiles per Inheritance toolboxes	29,426

- **Inheritance and OO**

Remember that since inheritance is an OO feature, the analysis of OO features includes all mfiles with inheritance features as well. Therefore, it was decided to conduct a combined analysis of the two. These results are in Table 7.3. These Results are discussed in the next sub-section.

Table 7.3: Combined Analysis of OO with Inheritance

Result	**Value**
Mfiles with OO and not Inheritance (% Repository)	2323 (0,51%)
Toolboxes with OO and not Inheritance (% Repository)	272 (2,36%)
Mfiles with Inheritance (% OO mfiles)	(83,59%)
Toolboxes with Inheritance (% OO toolboxes)	(80,90%)

7.1.3 Analysis on Inheritance and Libraries

In Table 7.3 it´s observable that 83% of all OO mfiles have inheritance. This means that almost every mfile with OO features also had inheritance features. Therefore, it was decided to conduct additional research to determine the reason behind the abundance of subclasses declared in OO mfiles.

After some web and repository investigation, it was revealed that MATLAB sub-classes can also be sub-classes of libraries in addition to being sub-classes of another class in the system. Therefore, it was decided to investigate the proportion of these classes that were importing from libraries rather than from other classes that the programmer had developed.

- **How this Analysis was made**

  It was acknowledged that almost all of these "imports" were produced using paths. The same as in the case of Listing 2.3, these paths were separated by a period '.'. As a result, a query was made to determine whether the tokens 'classdef', '<' and '.' were on the same line.

  After conducting searches online and in the repository, it was revealed that the super-class 'handle' is frequently used in the repository [22]. Unlike the majority of others, such as Listing 2.3, this library does not use '.'. Thus, all inheritance results related to the library's 'handle' were added to the results of the previous query, both queries are in Listing I.4. These results are in Table 7.4. These Results are discussed in the next chapter.

- **Figure 7.2 Relevant Details**

  As the number of mfiles with inheritance related to libraries in each toolbox increases, a decline in frequency is also visible in the graph in Figure 7.2. The less complex toolboxes are once again more prevalent than the more complex ones, as can be seen. Once more, there is a huge disparity between the simplest possibility

Table 7.4: Inheritance Related to Libraries Results

Result	Value
Mfiles with Inheritance related to libraries (% Repository)	6504 (1,44%)
Toolboxes with Inheritance related to libraries (% Repository)	970 (8,41%)
Average Inheritance mfiles related to libraries per toolbox	6,7
Standard Deviation Inheritance mfiles related to libraries per toolbox	16,85
Mfiles with Inheritance related to libraries (% OO mfiles) (% Inheritance mfiles)	(45,95%) (54,96%)
Toolboxes with Inheritance related to libraries (% OO toolboxes) (% Inheritance toolboxes)	(68,12%) (84,20%)

and the next, marginally more complex possibility (toolboxes with one and two mfiles).

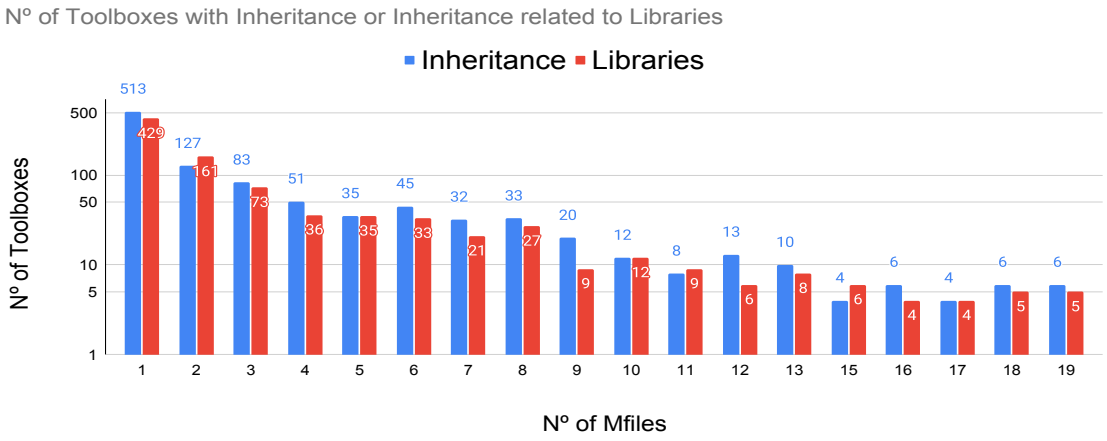

Figure 7.2: (Y axis) Nº of Tooboxes (logarithmic scale) vs (X axis) Nº of Inheritance Mfiles (Blue) and Inheritance related to libraries Mfiles (Red). Each bar shows how many toolboxes contain a specific number of inheritance mfiles (blue) or mfiles with inheritance pertaining to libraries (Red).

7.1.4 Visibility of Properties Blocks

This subsection contains the results for the visibility of class property blocks, which are found in the Table 7.5. These Results are discussed in the next chapter.

- The average number of 'property' CBLCs per class: 1,43.

- **Set by List Calculation**

  The total number of times Access, GetAccess, and SetAccess occurred was calculated using a query, and all specific results (Public, Private and Protected) were subtracted to leave the number of times they were set to public in the list of system-related classes.

- **Defaults (SetAccess and GetAccess) Calculation**

  Using the Venn Diagram in Figure 7.3 is the easiest way to explain the default values calculation. The Figure 7.3 was added to help make the following explanations

easier to understand and illustrate the proportions between 'GetAccess' and 'SetAccess'. It is possible to see in the venn that 'GetAccess' was used 173 times without 'SetAccess'. If 'GetAccess' was specified and 'SetAccess' was not, means that 'SetAccess' was set to default. Therefore 'SetAccess' was set to deafult 173 times. On the opposite position of the Venn we have that 'SetAccess' was used 2347 times without 'GetAccess'. Following the same line of thought, 'GetAccess' was set to default 2347 times.

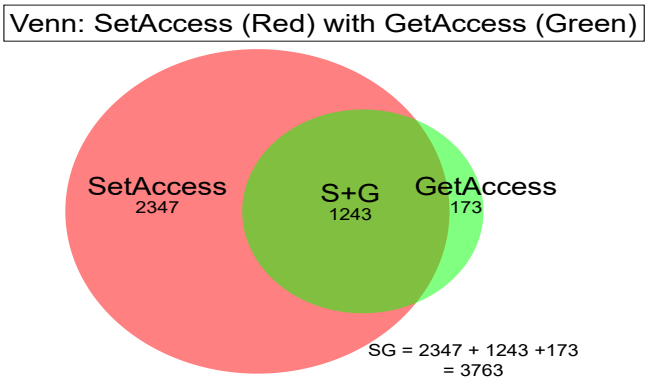

Figure 7.3: Venn Diagram with SetAccess and GetAccess

- **General Default Calculation**

  The general default was calculated by subtracting SG (in the writing in Figure 7.3) and the total amount of times Access was specified from the total number of property CBLCs.

- **Total in Table 7.5**

  Total, in Table 7.5, represents the total number of times an attribute was specified, therefore it is the sum of the number of times the attribute was set to public, private, protected, or list.

7.1.5 Visibility of Methods Blocks

In terms of visibility, methods are simpler than properties because they lack GetAccess and SetAccess, as stated in Chapter 2. But while it was simpler, the process for getting the analysis's findings was very similar to the 'property' CBLCs analysis. These results are in Table 7.6. These Results are discussed in the next chapter.

- The average number of 'method' CBLCs per class: 1,85.

Table 7.5: Class Properties Results. This table shows the attribute, its associated value, the number of properties that have that value, and the percentage of the total repository represented by that value.

Property	Value	#Ocurrences	% Of Total Blocks
Access	Public	683	3,369
Access	Private	1160	5,722
Access	Protected	389	1,919
Access	List	166	0,819
Access	Total	2398	11,830
GetAccess	Public	600	2,960
GetAccess	Private	308	1,519
GetAccess	Protected	112	0,553
GetAccess	List	396	1,954
GetAccess	Total	1416	6,985
GetAccess	Default	2347	11,578
SetAccess	Public	189	0,932
SetAccess	Private	1923	9,486
SetAccess	Protected	879	4,336
SetAccess	List	599	2,955
SetAccess	Total	3590	17,710
SetAccess	Default	173	0,853
General	Default	14110	69,607

The total value in the second column is another feature of the Table 7.6, this value has the same meaning as in Table 7.5.

Table 7.6: Methods Visibility Results. This table has the same design as Table 7.5.

Property	Value	#Occurrences	% Of Total Blocks
Access	Public	1474	5,626
Access	Private	1259	4,805
Access	Protected	1671	6,378
Access	List	454	1,733
Access	Total	4858	18,541
Access	Default	21343	81,459

7.2 Characterization of Language Constructs in the Repository

The primary purpose of this section is to identify the most prevalent LCs used by MATLAB developers in order to characterize the MATLAB language. The list of all LC occurrences in the repository is displayed in the Table 7.7, this table is analysed in the next chapter. Other factors to consider are Scripts, and the complexity of mfiles. The queries used in these section are in Listing I.5.

- **Scripts**

  Another factor to consider was whether the MATLAB language was mostly used as scripts or if each mfile contained several 'function' CBLCs. A query was created for this purpose that first counted the number of 'function' CBLCs in each mfile. The number of mfiles was then counted for each different value of the number of CBLCs per mfile.

- **Manual Search**

  Another query was created to obtain the names of the mfiles and their respective toolboxes in order to perform a manual analysis and determine whether the mfiles that contained zero 'function' CBLCs were scripts, data files, or any other type of file, such as documentation.

- **Other Purposes Found**

  With these queries created, it was preferable to put them to some other good use. Another useful application discovered was to perform a similar analysis on other CBLCs. If, While, and For were chosen to investigate whether or not these seemingly common LCs are widely used, and if so, how frequently.

7.2.1 Most Frequent LCs Results

This section result's are depicted in Table 7.7.

7.2.2 Function Blocks Analysis Results

A query was run to determine the number of mfiles containing a specific number of 'function' CBLCs.

- **Figure 7.4 Explanation**

  The top 24 results, are in Figure 7.4. In order to facilitate reading and understanding of the graph, the Y axis has a logarithmic scale. The first bar from the left, in example, shows that 72 436 mfiles do not contain a single 'function' CBLC. The second bar from the left demonstrates that 311 265 mfiles contain only a single 'function' CBLC, and so on.

Table 7.7: The Most Commonly Used Language Constructs in the Repository. The first column of this table contains the name of the LC. A second with the is token that identifies it. Finally, the total number of occurrences in the repository. This table is arranged in descending order of the number of occurrences.

LC (CB and Non-CB)	**Token**	**Nº of Occurrences**
Assignment of values to a variable	=	15306007
Minus	-	3829187
Multiply	*	3060552
Plus	+	2796020
If	if	2329568
Right Division	/	1259158
Transpose (Matrix)	’	975568
Equal to	==	817430
For	for	764382
Function	function	742359
NOT	~	707822
Multiply (Array)	.*	571984
Exponential	^	552184
Main	beginning of file	452983
Greater than	>	439258
Less than	<	416042
Exponential (Array)	.^	289783
AND wsc	&&	276386
Right Division (Array)	./	236888
Not Equal to	~=	185010
AND	&	176383
OR wsc	||	175345
Less or Equal to	<=	129471
Greater or Equal than	>=	104504
Left Division	\	94758
Switch-case	switch	89645
OR	|	82249
While	while	65851
Try-catch	try	65832
Transpose (Array)	.’	44126
Class methods	methods	26201
Class properties	properties	20271
Class defining	classdef	14193
Class events	events	11258
Parfor	parfor	7522
Process Terminating Func (exit)	exit	1174
Left Division (Array)	.\	1103
Process Terminating Func (quit)	quit	786
Spmd	spmd	486
Class enumerations	enumeration	198

- **Figure 7.4 Relevant Details**

  This graph clearly shows that there are many mfiles with 0 'function' CBLCs, scripts, and even more with just one. When the complexity of the mfile is increased, in this case the number of 'function' CBLCs, a clear descendent path in the number of mfiles can be seen after the second bar.

  The query used in this section of the analysis does not take into account 'function' CBLCs that are methods. This was necessary because each 'method' CBLC can have multiple 'function' CBLC, and the results would vary greatly.

- **Scripts Manual Analysis**

  The repository was manually searched for cases where the mfile contained 0 occurrences of the 'function' CBLC, and it was discovered that the majority of mfiles were scripts, with only a small number of data files and nearly 0 documentation files.

- **Average**

  The repository had an average of 1,69 'function' CBLCs per mfile. In this average only non-OO mfiles were analysed.

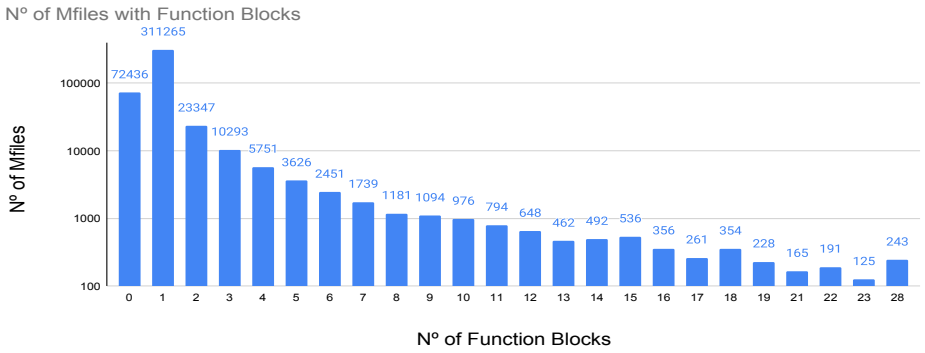

Figure 7.4: (Y axis) Nº of Mfiles (logarithmic scale) vs (X axis) Nº of Function Blocks. Each bar in this graph represents the number of mfiles containing a specific number of function blocks.

- **Mfiles with 1 'function' CBLC**

  Because so many mfiles had only one 'function' CBLC, it was decided to delve deeper into the complexity of these mfiles. These mfiles were also chosen because they have limited modularization options. There are only two options: either the source text is modularized inside the 'function' or outside.

- **Table 7.8 Illustration**

  As an illustration, the first line of the Table 7.8 states that there are 95 562 mfiles with only one CBLC 'function' and between three and twelve lines of code. Only non-commented lines are counted in this table.

- **Table 7.8 Relevant Details**

  It's interesting to see how many LoC there are in many of these mfiles; it's interesting because it begins to make sense of how the MATLAB community uses the language (few structures and high LoC).

Table 7.8: Nº of mfiles with one 'function' CBLC per LoC partition (only non-commented lines). The first column is the number of LoC partitions, and the second is the number of mfiles that contain only one 'function' CBLC. Each line indicates the number of mfiles with a single 'function' CBLC and a particular number of LoC.

LoC Partitions	Nº of Mfiles
SUM 3-12	94933
SUM 13-22	61250
SUM 23-32	39259
SUM 33-42	24565
SUM 43-52	17050
SUM 53+	208643

7.2.3 Results of the If, While and For Blocks Analysis

The top 24 results, are in Figure 7.5, Figure 7.6 and Figure 7.7. In order to facilitate reading of the graph, the Y axis has a logarithmic scale. These graphs have a very similar design to the Figure 7.4.

- **If**

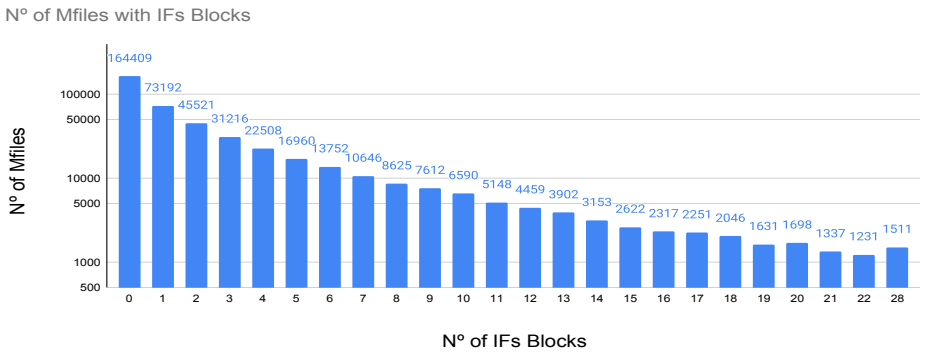

Figure 7.5: (Y axis) Nº of Mfiles (logarithmic scale) vs (X axis) Nº of Ifs Blocks

- **While**

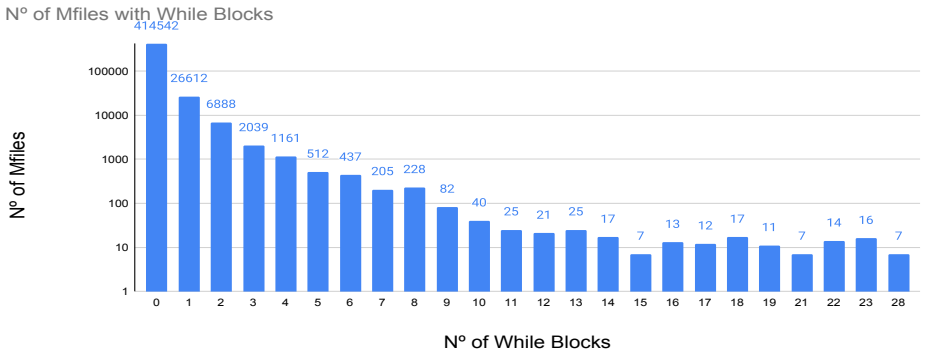

Figure 7.6: (Y axis) Nº of Mfiles (logarithmic scale) vs (X axis) Nº of While Blocks

- **For**

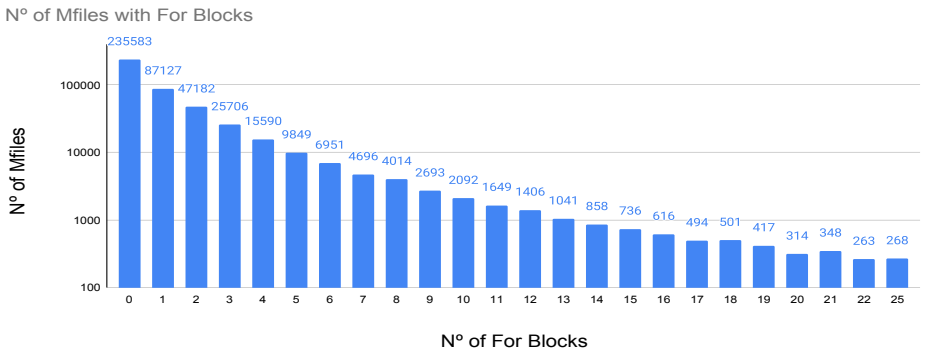

Figure 7.7: (Y axis) Nº of Mfiles (logarithmic scale) vs (X axis) Nº of For Blocks

- **Relevant Details about the Graphs (Figure 7.5, Figure 7.6 and Figure 7.7)**

  When analyzing from the less complex mfiles to the more complex ones, a descendent path of the number of mfiles is clearly visible in this graph. As a result, it is possible to conclude that more complex mfiles are less common than simpler ones, in terms of the number of these CBLCs per mfile.

7.3 Language Constructs and Concerns Relations

It is studied in this section which are the most relevant relations between LCs and concerns found when analysing the database. In this stage, Pointwise Mutual Information is used to compare all types of relations.

7.3.1 Pointwise Mutual Information

In this thesis, Pointwise Mutual Information (PMI) is used to determine how relevant a relationship between an LC and a concern. PMI can be calculated trough a mathematical

formula, this formula is in 7.1.

Any relation with a PMI value greater than 1 has some or high relevance, the higher the PMI value, the more relevant the relation is. When the PMI is between 1 and -1, the relevance is minimal, and when it is less than -1, the relation is relevant but because it is uncommon. A PMI of 0 means that the co-relation never occurs.

$$pmi(x;y) = \log_2 \frac{p(x,y)}{p(x)p(y)} \tag{7.1}$$

Equation 7.1: Pointwise Mutual Information (PMI) formula [40]

7.3.2 Most Frequent Relations Concerns\LCs

The first step in elaborating this section of the analysis was to query the database using the queries in Listing I.6. For each possible combination, this query returned the number of relationships between a LC and a concern.

- **Frequencies Matrix**

  The results had to be sorted in descending order after calculating the PMI of each relationship, with the most pertinent results (in terms of PMI) appearing at the top.

  A matrix containing each result from the queries was created to calculate the PMI. In this matrix, each column was a concern and each line was a LC, both CBLCs and non-CBLCs.

- **PMI Matrix**

  Then, using the formula in Equation 7.1, it was simple to create a new matrix with all of the PMI values, because all frequencies were in the previous matrix.

7.3.3 Results of the Relations LCs with Concerns

The results for this section are shown in the tables Table 7.9 and Table 7.10, with the most pertinent results — those with positive and negative PMI values — at the top of each table. For ease of reading, the tables are arranged by concerns.

Table 7.9: Most Relevant Positive PMI Results of the Relations Between Concerns and Language Constructs. The 'rank' of the relationship is indicated in the first column of both tables, a rank was created by sorting the PMI values from highest to lowest. The name of the LC is listed in the second column, so the information in each line of the results refers to both that LC and the concern listed directly above it in the table. The third column contains the f(x,y) frequency, which shows how many times the relationship in question has occurred. The frequency of the LC in concern/LCs relations given by F(x). The frequency of the concern in concern/LCs relations is expressed as F(y). The relation's PMI value is displayed in the final column.

#	LCName	f(x,y)	f(x) (LC)	f(y) (Concern)	PMI
Concern: System - Entering comands					
1º	Process Terminating Func (quit)	12.0	48.0	36062.0	6.778
6º	Process Terminating Func (exit)	6.0	119.0	36062.0	4.468
Concern: Parallelization					
2º	Spmd	904.0	1110.0	140071.0	6.524
4º	Parfor	7568.0	13289.0	140071.0	6.008
20º	Process Terminating Func (quit)	4.0	48.0	140071.0	3.235
Concern: System - Using external libraries					
3º	Left Division (Array)	83.0	664.0	28962.0	6.094
8º	Left Division	498.0	14800.0	28962.0	4.201
10º	Right Division (Array)	1188.0	40656.0	28962.0	3.997
17º	Right Division	4528.0	232338.0	28962.0	3.413
Concern: Printing					
5º	Left Division	706.0	14800.0	24728.0	4.933
12º	Left Division (Array)	15.0	664.0	24728.0	3.854
Concern: System - Code analysis					
7º	Left Division (Array)	21.0	664.0	23476.0	4.415
Concern: System – Functions					
9º	Transpose (Array)	1315.0	17497.0	73404.0	4.019
Concern: Console messages					
11º	Class enumerations	126.0	153.0	901447.0	3.854
Concern: Visualization – Images					
13º	Process Terminating Func (quit)	4.0	48.0	105827.0	3.640
Concern: System – Scripts					
14º	Process Terminating Func (quit)	4.0	48.0	106477.0	3.631
Concern: Data type specialization					
15º	Class defining	126.0	233.0	707885.0	3.596
Concern: Visualization - 2D and 3D plots - Polar plots					
16º	Exponential (Array)	42.0	24236.0	2492.0	3.461
Concern: File I/O					
18º	Left Division (Array)	436.0	664.0	1039896.0	3.321
21º	Left Division	8591.0	14800.0	1039896.0	3.143
Concern: Visualization - 2D and 3D plots - Contour plots					
19º	Transpose (Matrix)	206.0	352807.0	965.0	3.260

Table 7.10: Most Relevant Negative PMI Results of the Relations Between Concerns and Language Constructs. This table has the same design as the previous one.

#	LCName	f(x,y)	f(x) (LC)	f(y) (Concern)	PMI
Concern: Verification of function arguments and return values					
1º	Main	287.0	874851.0	1596753.0	-8.263
Concern: System - Data import and export					
2º	Greater or Equal than	1.0	40652.0	100716.0	-8.014
5º	Less or Equal to	2.0	35315.0	100716.0	-6.811
6º	Greater than	12.0	203914.0	100716.0	-6.755
9º	Parfor	1.0	13289.0	100716.0	-6.401
Concern: Visualization - Graphic Objects – Programming/output					
3º	OR wsc	22.0	226779.0	207258.0	-7.075
Concern: Visualization - 2D and 3D plots - Data distribution plots					
4º	OR wsc	6.0	226779.0	50113.0	-6.902
Concern: System - Files and folders					
7º	Parfor	2.0	13289.0	244556.0	-6.681
12º	Exponential	4.0	18656.0	244556.0	-6.170
Concern: File I/O					
8º	OR wsc	166.0	226779.0	1039896.0	-6.487
Concern: Memory allocation/deallocation					
10º	AND wsc	420.0	309725.0	1674610.0	-6.285
Concern: System - Using external libraries					
11º	Greater than	5.0	203914.0	28962.0	-6.220

7.4 Language Constructs and their Relation to Concern Tokens

The previous section's positive PMI results, Sub-Section 7.3.3, are being studied in greater detail in this section of the analysis. Due to the fact that each concern in MATLAB can be represented by a number of tokens, it is also possible to investigate the relationships between each of these tokens and the LCs.

- **Queries**

  The queries used in this section are in Listing I.7. Both the non-CBLC and the CBLC queries are very similar to their counterparts in Section 7.3, but they filter the results to just one concern. For each relation that was identified in Section 7.3, it was necessary to repeat these queries with appropriate values (LC and concern).

- **Queries' Results**

  The results were then expanded, and a percentage was determined. This percentage is calculated by dividing the value of the number of tokens that are related to that LC by the total number of concern tokens of that token's concern. This percentage reveals the importance of those particular relationships to the whole concern.

7.4.1 Results of the Relation LCs with Concern Tokens

The Table 7.11 displays the results for this section. For ease of reading, the table is arranged by concerns. The table is also arranged according to LCs, so within each concern the table is first arranged according to LCs and only then by rank.

Table 7.11: Most Common Relations Between Concern Tokens and Language Constructs. The rank is the first column in the table and it displays the order, sorted by the percentage in the last column. The relation's LC is shown in the second column. The name of the concern's token, appears in the third column. TotalRelations displays the co-occurrence rates of the concern token and the LC. The total number of instances of the concern is shown in the fifth column. The final column displays the proportion of the number of relationships being examined divided by the total number of instances of the concern.

#	LCName	nameToken	TotalRelations	TotalConcernsInDB	Percentage
Concern: System - Using external libraries					
1º	Right Division	mex	4491.0	11269.0	39.852%
14º	Right Division	calllib	25.0	11269.0	0.221%
20º	Right Division	mexext	12.0	11269.0	0.106%
4º	Right Division (Array)	mex	1188.0	11269.0	10.542%
7º	Left Division	mex	493.0	11269.0	4.374%
24º	Left Division	mexext	5.0	11269.0	0.044%
10º	Left Division (Array)	mex	83.0	11269.0	0.736%
Concern: Visualization - 2D and 3D plots - Contour plots					
2º	Transpose (Matrix)	clabel	206.0	546.0	37.728%
Concern: Parallelization					
3º	Parfor	parfor	7522.0	53182.0	14.143%
23º	Parfor	sparse	39.0	53182.0	0.073%
38º	Parfor	gather	3.0	53182.0	0.005%
40º	Parfor	labindex	2.0	53182.0	0.003%
41º	Parfor	numlabs	2.0	53182.0	0.003%
9º	Spmd	spmd	486.0	53182.0	0.913%
13º	Spmd	labindex	248.0	53182.0	0.466%
16º	Spmd	numlabs	85.0	53182.0	0.159%
18º	Spmd	codistributed	77.0	53182.0	0.144%
33º	Spmd	gather	5.0	53182.0	0.009%
42º	Spmd	sparse	2.0	53182.0	0.003%
49º	Spmd	codistributor	1.0	53182.0	0.002%
36º	Process Terminating Func (quit)	cancel	4.0	53182.0	0.007%
Concern: Printing					
5º	Left Division	print	706.0	13829.0	5.105%
19º	Left Division (Array)	print	15.0	13829.0	0.108%
Concern: Visualization - 2D and 3D plots - Polar plots					
6º	Exponential (Array)	polar	42.0	940.0	4.468%
Concern: System – Functions					
8º	Transpose (Array)	mfilename	1253.0	39354.0	3.184%
17º	Transpose (Array)	inputname	61.0	39354.0	0.155%
45º	Transpose (Array)	symvar	1.0	39354.0	0.003%

Concern: File I/O					
11º	Left Division	fprintf	4038.0	732050.0	0.551%
12º	Left Division	load	3802.0	732050.0	0.519%
22º	Left Division	save	572.0	732050.0	0.078%
31º	Left Division	fopen	98.0	732050.0	0.013%
35º	Left Division	diary	58.0	732050.0	0.008%
44º	Left Division	saveas	20.0	732050.0	0.003%
54º	Left Division	fgetl	3.0	732050.0	0.0004%
25º	Left Division (Array)	fprintf	294.0	732050.0	0.040%
30º	Left Division (Array)	load	126.0	732050.0	0.017%
46º	Left Division (Array)	save	16.0	732050.0	0.002%
Concern: System - Code analysis					
15º	Left Division (Array)	run	21.0	10486.0	0.200%
Concern: System - Entering comands					
21º	Process Terminating Func (quit)	stop	12.0	14059.0	0.085%
27º	Process Terminating Func (exit)	stop	3.0	14059.0	0.021%
28º	Process Terminating Func (exit)	ans	3.0	14059.0	0.021%
Concern: Data type specialization					
26º	Class defining	uint8	70.0	242949.0	0.029%
32º	Class defining	uint32	25.0	242949.0	0.010%
39º	Class defining	double	13.0	242949.0	0.005%
43º	Class defining	int8	9.0	242949.0	0.004%
48º	Class defining	int32	5.0	242949.0	0.002%
50º	Class defining	int16	1.0	242949.0	0.0004%
51º	Class defining	single	1.0	242949.0	0.0004%
52º	Class defining	uint16	1.0	242949.0	0.0004%
53º	Class defining	uint64	1.0	242949.0	0.0004%
Concern: Console messages					
29º	Class enumerations	error	126.0	667008.0	0.019%
Concern: System – Scripts					
34º	Process Terminating Func (quit)	input	4.0	45788.0	0.009%
Concern: Visualization – Images					
37º	Process Terminating Func (quit)	imread	3.0	47928.0	0.006%
47º	Process Terminating Func (quit)	image	1.0	47928.0	0.002%

7.5 Validation of Schizophrenic Functions

It is necessary to re-validate schizophrenic functions because J.Barrulas [2] found that 20% of the repository he examined had schizophrenia symptoms. Plus he examined a smaller repository (about 65 000 mfiles), and with the current bigger repository (about 450 000 mfiles) a more accurate re-validation can be made, see Section 2.4.

7.5.1 Frequency of Schizophrenic Functions

The queries used here were adapted from J.Barrulas [2], these queries are shown in Listing I.8.

- **1st Query**

  The first query identifies the number of different mfiles that have schizophrenia symptoms. This query, as well as the following ones, must be used with once with 'if+nargin' and again with 'switch+nargin'.

- **2nd Query**

  The amount of 'if+nargin' combinations that relate to other concern tokens is returned by the second query in Listing I.8. With this query it is possible to determine the average number of concern tokens that schizophrenic functions relate to.

- **3rd Query**

  This query provides a similar result, but it considers parent blocks, see Section 5.3.

7.5.2 Schizophrenic Functions Results

These results obtained from this section are discussed in the next Chapter.

- **1st and 2nd Queries**

  The results obtained from running the 1st and 2nd queries are in Table 7.12.

Table 7.12: Results obtained from the 1st and 2nd Queries.

Result	**Value**
Mfiles with 'if+nargin'	96056
Toolboxes with 'if+nargin'	5678
Average mfiles with 'if+nargin' per toolbox	16,9
Relations concern tokens with 'if+nargin'	374869
Distinct 'if+nargin'	203845
Average 'if+nargin' per concern token	1,84
Mfiles with 'switch+nargin'	2962
Toolboxes with 'switch+nargin'	686
Average mfiles with 'switch+nargin' per toolbox	4,3
Relations concern tokens with 'switch+nargin'	16674
Distinct 'switch+nargin'	3758
Average 'switch+nargin' per concern token	4,44
Mfiles with schizophrenic functions (% Repository)	98221 (21,70%)
Toolboxes with schizophrenic functions (% Repository)	5799 (50,00%)

- **3rd Query**

  The number of hierarchically associated concern tokens to 'if+nargin' and 'switch+nargin' blocks are in Table 7.13. These results are discussed in the next Chapter.

Table 7.13: Number of hierarchically associated concern tokens to 'if+nargin' and 'switch+nargin blocks'. The first column is the name of the concern. The second is the number of times the concern relates to 'if+nargin'. The third is the number of times the concern relates to 'switch+nargin' and the rank of the concern in the 'switch' relations. With this rank it is possible to see if a concerns relates similarly, with 'ifs' and 'switches'.

Concern	**If**	**Switch**
Verification of function arguments and return values	322267	29387 -1#
Data type verification - Matrices and arrays	161622	10197 -2#
Console messages	64445	7831 -3#
Memory allocation/deallocation	30284	2286 -5#
Data type verification - Identification and numeric types	29580	3391 -4#
File I/O	19282	325 -7#
Visualization - Graphic Objects - Properties	16254	625 -6#
Dynamic properties	10185	118 -15#
System - Functions	8561	249 -8#
Visualization - Formatting and Annotation - Titles and labels	8148	138 -12#
Data type specialization	6382	200 -10#
Visualization - Formatting and Annotation - Axes appearance	5807	91 -16#
Visualization - Graphic Objects - Identification	5438	165 -11#
Visualization - 2D and 3D plots - Line plots	3624	38 -20#
Visualization - Graphic Objects - Programming/output	3252	38 -20#
Parallelization	2453	73 -18#
System - Files and folders	2427	81 -17#
System - Control flow	2331	127 -14#
System - Dates and time	1812	62 -19#
System - Data import and export	1779	222 -9#
System - Scripts	1466	19 -23#
Visualization - Images	1335	12 -25#
System - Performance and memory	890	8 -26#
Printing	580	NaN
System - Startup	566	130 -13#
Visualization - 2D and 3D plots - Surfaces, volumes and polygons	562	8 -26#
System - Entering comands	390	25 -22#

Visualization - 2D and 3D plots - Data distribution plots	245	NaN
System - Using external libraries	209	1 -28#
Visualization - Visual exploration	133	13 -24#
System - Code analysis	83	NaN
Visualization - 2D and 3D plots - Animation	54	NaN
System - Debugging	15	NaN
Visualization - 2D and 3D plots - Contour plots	11	NaN
Visualization - 2D and 3D plots - Polar plots	11	NaN

7.6 Schizophrenic Functions and the OO Paradigm

When analyzing the MATLAB language in terms of OO and schizophrenic functions, it's interesting to think whether or not these concepts are related in any way. There are two ways: - mfiles with schizophrenic functions are frequently OO mfiles; - OO mfiles are frequently mfiles with schizophrenic functions. The queries in Listing I.9 were used for this analysis.

7.6.1 Schizophrenic Functions and the OO Paradigm Results

The findings from the analysis of the relationship between schizophrenic functions and the OO paradigm can be found in Table 7.14.

Table 7.14: Relation Results: Schizophrenic Functions with the OO Paradigm

Result	Value
OO mfiles with schizophrenic funtions (% Repository) (% OO mfiles)	3783 (0,84%) (26,70%)
Toolboxes contained OO mfiles with schizophrenic funtions (% Repository) (% OO toolboxes)	766 (6,64%) (53,80%)
OO mfiles with schizophrenic funtions (Percentage of mfiles with schizo. Funcs.)	3,85%
Toolboxes contained OO mfiles with schizophrenic funtions (Percentage of mfiles with schizo. Funcs.)	13,20%
Average OO mfiles with schizophrenic funtions per Toolbox	4,94
Standard Deviation OO mfiles with schizophrenic funtions per Toolbox	10,86

- **Possible Relevant Relation**

  Because schizophrenic symptoms were present in more than one-fourth of all OO mfiles, a significant correlation can be seen (- mfiles with schizophrenic functions are frequently OO mfiles).

- **No Relevant Relation**

  OO mfiles are 3.85% of mfiles with schizophrenic functions (- OO mfiles are frequently mfiles with schizophrenic functions), this is not relevant due to a low value.

- **Graph**

  The decrease in frequency as complexity rises can once again be seen, Figure 7.8.

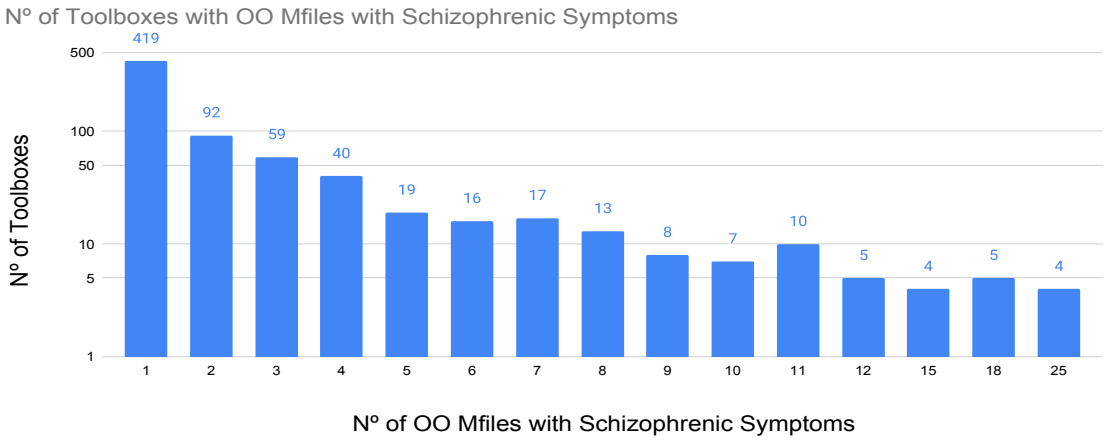

Figure 7.8: (Y axis) Nº of Toolboxes (logarithmic scale) vs (X axis) Nº of OO mfiles with schizophrenic symptoms. Each bar shows the number of toolboxes containing a specific number of OO mfiles with schizophrenic symptoms.

7.6.2 Schizophrenic Functions in OO

Due to the fact that the relationship mentioned above (- mfiles with schizophrenic functions are frequently OO mfiles;) was deemed significant, it was decided to look further and compare the frequencies of schizophrenic functions in OO mfiles with the frequencies in non-OO mfiles (mfiles without OO features).

- **Result table**

  The frequency of schizophrenic functions in OO toolboxes and mfiles is compared to the frequency in non-OO in the Table 7.15. It should be noted that a toolbox is counted in both columns if it contains both OO and non-OO mfiles.

Table 7.15: Comparative Analysis between OO paradigm and Schizophrenic Functions

	OO	**NON-OO**
Nº Mfiles	14156	438827
Nº Toolboxes	1424	11272
Mfiles with Schizophrenia	3783 (26,7%)	94438 (21,5%)
Toolboxes with Schizophrenia	766 (53,7%)	5525 (49%)
Mean	4,93	17
Standard Deviation	10,86	78,7

- **Unrelated Relevant Details**

  There are only 11 272 toolboxes in the repository that have mfiles that are not OO. This indicates that 256 toolboxes only contain OO mfiles.

8

Results Discussion

In a more in-depth reading of each of the most significant results, this chapter discusses the findings that emerged from the analysis of the previous chapter.

8.1 Object Oriented Systems

The results discussed in this section are the ones presented in Section 7.1, and they will help answer RQ1.

- **Classes and Inheritance**

  From the collected data it is possible to observe that while OO features are not frequently used in MATLAB, they are present in about 12% of toolboxes. This indicates that some programmers are familiar with OO features in MATLAB. Additionally, those who are familiar with it frequently use it, since each OO toolbox (a toolbox with at least one OO mfile) contains on average nearly 10 OO mfiles. A standard deviation of 33 suggests that the number of OO mfiles per toolbox typically deviates greatly from the mean. The fact that most toolboxes only contain one OO mfile, as it can be seen in Figure 7.1, greatly contributes to a high standard deviation.

  Since it might be assumed that all inheritance mfiles are also OO mfiles but not all OO mfiles are inheritance mfiles, inheritance mfiles are used even less frequently than OO mfiles. However, the results of inheritance mfiles are similar to the previous results (OO mfiles). Therefore the analysis of the results is the same.

- **Inheritance related to Libraries**

  The study found that only 0.5% of mfiles used OO features without inheritance, while 83.6% of OO mfiles used inheritance. Further research revealed that inheritance in MATLAB is commonly used to import libraries. Additionally, 46% of all OO mfiles and 55% of all inheritance mfiles were related to libraries. The analysis drawn from this results is that OO features are not frequently used in MATLAB, and when they are, they are "mostly" used for importing libraries.

- **'Property' CBLC: Default Visibility Option**

  The Table 7.5 shows that 'Public' is used frequently. Leaving the access field empty is the more accurate option as programmers tend to use more sophisticated OO features. However, access is still set to 'public' many times. 'GetAccess' has a similar situation. 'SetAccess' has a higher number of public values than default values. It would be advantageous for all programmers to know that leaving a field empty has the same effect as making it public, which would make the code easier to read and less intimidating to readers. A 400 LoC class does not look much more readable by leaving one field empty. But it does help, and every little bit counts when writing source code, especially when all improvements are considered together.

- **'Property' CBLC: Protected and List Visibility Options**

  The study found that Private was the most commonly used visibility in the repository other than 'Public' and Default, but it is uncertain whether Protected or List should have been used. Checking this would require a significant software quality analysis in several systems. The MATLAB language offers the ability for classes to only be visible to a specific List of other classes, which can improve overall software quality and maintainability, because all classes that have access to the class in question can be better organized. Further discussion on the low usage of 'Protected' and 'List' can be found in the Future Work section, see Section 9.2.

- **Venn**

  From the Venn diagram in Figure 7.3, the most obvious analysis is that 'GetAccess' is nearly always used together with 'SetAccess'. 'SetAccess' on the other hand, is used more times alone than together with 'GetAccess'.

- **'Method' CBLC**

  Regarding the 'method' CBLC, it is once again evident that the 'public' is being overused. 'Protected', however, has a greater representation. Lists are once more underutilized.

- **OO features in MATLAB**

  Despite the addition of OO features to MATLAB, programmers do not use them as extensively as in regular Object Oriented Languages (OOLs). Instead, MATLAB users tend to favor using the language in its original design without the OO features. The OO features are mainly utilized to support the language's original purpose. This analysis was drawn based on manual inspection that found that non-OO mfiles tend to instantiate OO classes more often than OO mfiles.

8.2 Most Common Language Constructs in the Database

The results discussed in this section are the ones presented in Section 7.2, and they will help answer RQ2 and RQ4.

- **Mathematical Algorithms**

  It is clear that MATLAB is a programming language that is frequently used for calculations. Because the top-used LC, in Table 7.7, are all used in calculations, despite the fact that some of them can also be used in a variety of other ways.

  The LCs 'if', 'for' and 'function' frequency, which are also at the top, in Table 7.7, suggests that this language is frequently used in the creation of algorithms.

  It is possible to infer from the two facts above that MATLAB is primarily used in numerical algorithms. A manual check revealed this to be true.

- **Parallelization**

  The parallelization is not widely used, neither the 'parfor' or the 'spmd', as seen in Table 7.7. This means that most algorithms designed in this language are linear, and not parallel.

- **Process Terminating Lcs**

  Most algorithms created in this language do not need to be terminated in the middle of their "run" because process termination functions are not frequently used, as shown in Table 7.7.

- **Mfiles with only One 'function' CBLC**

  There are typically two common types of these mfiles: with a low LoC, or with a high LoC.

  The analysis of the repository revealed that a large number of mfiles with a single 'function' CBLC had an unreasonable amount of lines of code, as seen in Table 7.8, indicating that they may have too many concerns. The example provided in the Chapter 3 suggests that this situation is problematic, as there may be even more concerns in 53+ LoC. The options for modularization are limited to either inside or outside of the function, and therefore the chances of each concern being located in its own module are extremely low. Instead of having everything in the same 'function' CBLC, developers should aim to modularize their system by dividing each functionality into its own module, as argued in Chapter 3, which would make their system easier to maintain and comprehend.

- **Maintainability Motivation**

  The risk of modifying an older system or code file is high, especially if the code is too long or complex. It is advisable to avoid using mfiles with a single 'function' CBLC and too many LoC to prevent further challenges.

  Mfiles tend to be less common as they become more complex, as demonstrated by several analyses including the 'function' CBLC analysis, as well as the 'if', 'while', and 'for' analyses described in Section 7.1 and Section 7.2. Mfiles with low LoC can serve as a useful guide for improving system maintainability, as they are likely to be better modularized and easier to read. However, increasing the LoC can worsen the maintainability of an mfile with few structures, as previously discussed.

8.3 Language Constructs and Concerns Relations

The results discussed in this section are the ones presented in Section 7.3, and they will help answer RQ3.

- **Process Terminating Functions**

  The Process Terminating Functions have the highest PMI values in Table 7.9. This is because there aren't many in the repository and consequently, there aren't many that relate to concerns. However the PMI value tells us that they have an high occurrence with entering commands.

- **Parallelization**

  Due to their nature as parallelization features, 'SPMD' and 'Parfor' have a clearly high incidence with parallelization concern tokens. This confirms the validity of the tokens used to identify the parallelization concern.

  Due to the fact that there are so few instances of 'Quit' throughout the entire repository, it also occurs frequently with parallelization tokens, as seen in Table 7.9.

- **CCCExplorer's LC Identification Error**

  When analyzing the concerns "System - Using external libraries" and "File I/O", it is abundantly clear that the Division LCs are used in the paths. This happens because the CCCExplorer error mentioned earlier causes the LCs to be identified inside strings.

- **'classdef' CBLC**

  One of the most intriguing top PMI values in the analysis is the 'classdef' CBLC strong relation to the concern 'Data type specialization'. This happens because when one class is instantiated, the integer type must be changed. The low number of classes in the repository contributes to this high value as well.

- **Negative Values of PMI**

  The negative PMI values were evaluated in this study because they were considered as having some value because at least one programmer saw value in that relation. However, values with 0 PMI were not evaluated because no one saw value in that relation. The most intriguing finding in the second table of results in Table 7.10 was the low correlation between 'parfor' and the concern "System - Data import and export". This suggests that parallelization is not widely used for data import and export, possibly because MATLAB's parallelization is not widely used.

8.4 Language Constructs and their Relation to each Concern Token

The results discussed in this section are the ones presented in Section 7.4, and they will help answer RQ3.

- **CCCExplorer's LC Identification Error**

  With the concerns: "System - Using external libraries", "Printing" and "File I/O" it is clear that the right and left division are still the most frequently used LC. Therefore, it is reasonable to infer that they are frequently used in paths. This situation is evident in the analysis, but only as a result of the apostrophe error in CCCExplorer and incorrect identification of LCs. Due to this result the apostrophe error was identified and it is now being corrected.

- **Re-Validation of Concern Tokens**

  The 'clabel' token is found to be confirmed to belong to the concern it was first thought to belong to, and the Transpose(Matrix) LC is prevalent when 'clabel' is used, indicating that one of the matrices used as arguments for the concern token is often transposed.

  The parallelization concern tokens are also re-validated due to new evidence showing their frequent use with parallelization LCs.

- **'classdef' CBLC**

  It is possible to see that not many concern tokens are used with 'classdef' in the "Data type specialization" concern. With this knowledge, it is possible to analyse the results in order to find that the low usage of the 'classdef' LC was the cause of the high PMI in the previous phase.

8.5 Validation of Schizophrenic Functions

The results discussed in this section are the ones presented in Section 7.5, and they will help answer RQ5.

- **Schizophrenic Functions are Even More Frequent than First Believed**

  On this phase, it is possible to observe that this repository contains more instances of schizophrenic functions than the repository of J.Barrulas [2]. He discovered that while the repository he examined contained a percentage of roughly 20%, this analysis contains a percentage of almost 22%. This also indicates that, as would be expected, the results of a study conducted are slightly altered by a larger repository. However in this specific case the bigger repository was not necessary, since the results are very similar.

- **Relation to Concern Tokens**

  It is also visible that 'switch' relate more to concern tokens than 'if'. This is accurate because the CCCExplorer's 'if' block is handled similarly to a 'switch'. This meaning that the first 'if' in the chain and all subsequent 'elses' and 'elseifs' are the same block. Just like a 'switch' and all it's 'cases'. This finding that schizophrenic 'switches' are more closely related to concerns than to schizophrenic 'ifs', as seen in Table 7.12, is quite intriguing. Statistics cannot be used to demonstrate why this is the case. This leads me to think of an idea for an analysis. This idea is presented in the next chapter, more specifically in the Future Work section, see Section 9.2.

- **Hierarchical Relations**

  The two concerns most related hierarchically to schizophrenic 'ifs' are identified as 'Verification of function arguments and return values' and 'Data type verification - Matrices and Arrays', implying that verification often occur near schizophrenic functions. Additionally, the concerns most related to schizophrenic 'switches' are the same as those related to schizophrenic 'ifs', so the analysis about schizophrenic 'ifs' apply to schizophrenic functions in general. The "Memory allocation/deallocation" concern is also frequently used with schizophrenic functions, indicating that variables' memory is deallocated/allocated based on the number of arguments passed to the functions.

8.6 Schizophrenic Functions and the OO Paradigm

The results discussed in this section are the ones presented in Section 7.6, and they will help answer RQ6.

- **Schizophrenic Function in OO vs non-OO**

  The Section 7.6 ends with Table 7.15, that compares the frequency of schizophrenic functions in OO mfiles and toolboxes to the frequency of schizophrenic functions in non-OO mfiles and toolboxes.

  In this table, the frequency of schizophrenic functions is higher in OO than in non-OO. This is because the proportion of OO mfiles containing schizophrenic functions

is 5% higher than the proportion of non-OO mfiles containing schizophrenic functions. The same is true for toolboxes, which also have a 5% increase.

- **Mean and Standard Deviation**

  Only toolboxes with at least one mfile containing schizophrenic symptoms are included in the mean and standard deviation. This explains why non-OO has a higher mean and st.dv. but a lower percentage of frequency. It's because schizophrenic functions are used in a more "aggressive" manner in non-OO.

- **No Relationship Between Schizophrenic Function and the OO Paradigm**

  However, observing that there is a significant relationship between schizophrenic functions and the OO paradigm is impossible. Because the results of the OO/non-OO comparison are different in practice, but too similar to draw a firm conclusion.

9 Conclusions and Future Work

This chapter contains the thesis's final conclusions as well as recommendations for future work.

9.1 Conclusions

To conclude this thesis, it may be best to start by providing brief responses to the research questions that motivated the analysis presented in this thesis.

- **RQ1:** What is the scope of the concrete use of OO features in MATLAB?

  - OO feaures are infrequently used in MATLAB, as seen in Table 7.1. When OO features are used in MATLAB, it is typically to import from libraries, as seen in Table 7.4. The visibility settings 'Protected' and 'List' are rarely used, as seen in Table 7.5 and Table 7.6.

- **RQ2:** What are the most used LCs in MATLAB, and how frequently in each mfile?

  - In MATLAB, the LCs that are most closely related to calculations are also the most common, as seen in Table 7.7. According to average usage, '=' is used 33,8 times per mfile, '-' 8,5 times, '*' 6,74 times, and '+' 6,2 times.

- **RQ3:** What are the most frequent concerns/LCs relations?

  - The most common relationships, in terms of high PMI values, are those that relate parallelization LCs ('spdm' and 'parfor') with parallelization concern, as seen in Table 7.9.

- **RQ4:** What is the average degree of complexity of mfiles, in MATLAB?

  - Mfiles typically contain many lines of code but only a few structures, as seen in Table 7.8, Figure 7.4, Figure 7.5, Figure 7.6 and Figure 7.7.

- **RQ5:** Are schizophrenic functions so frequent/probable in a repository of 450 000 mfiles as in a 65 000 repository?

- The 450 000 mfile repository produces results that show a slightly higher frequency of schizophrenic functions, as seen in Table 7.12.

- **RQ6:** Can schizophrenic functions be related to the OO paradigm?

  - The OO paradigm and schizophrenic functions do not significantly co-relate to one another, as seen in Table 7.15.

Besides answering the Research Questions another objective of this thesis was to find some coding anti-patterns. The coding anti-patterns found with the analysis present in this thesis are the following:

- The 'Public' option seems to be overused in both 'property' and 'methods' visibility options,
- The option to define the visibility by a list is underused, in both 'property' and 'methods' CBLCs,
- Generally speaking, mfiles should be better modularized, since they have too many lines of code and few structures on average,
- Schizophrenic functions are even more common that what was first believed,
- Overall paralelization is not often used. Since parallel algorithms are faster than linear ones in the same task [31] [11], they should be more used,
- Import and Export of data is nearly never parallelized, parallelization in this case could obviously make these actions faster,
- Using OO features to import from libraries is common, this is a grey area because the computer science community does not view this as the primary option for using OO features, even though it could be a good thing because it makes it easier to import libraries and use their features.

Additional general conclusions can be drawn after succinctly responding to the objective and research questions, and they are as follows:

- **OO features**

  Finally, it should be noted that the MATALB community does not make extensive use of OO or inheritance features. Nevertheless, some people who use them do so in a sizable quantity.

- **Mathematical Algorithms**

  Additionally, it can be said that the language is applied to mathematical algorithms, as its designers intended. It is taught to computer scientist that designers of informatics systems occasionally create and design features to be used in a particular

way. The community, however, later employs those same features for other uses. The features that were once intended to be used in a specific way are therefore not often used in that particular way. All of this serves to support the assertion that it is advantageous for the MATLAB language to be primarily used in the manner in which its designers originally intended.

- **Schizophrenic Functions**

  They are a slightly more frequent than what was first thought. This provides some more motivation to carry out additional research on this topic, especially if it is worthwhile to find a general method of dissecting each of these functions into a number of non-schizophrenic functions, more on this topic in the next section.

- **LCs Concern Relations**

  Regarding the relations between concerns and LCs, this thesis has assisted in the re-validation of concern tokens that have been under investigation for a number of years [23] [24] [3] [10]. This is significant because these tokens were never "physically" re-validated before. There was no extensive repository used to build the table. However, it was created with the assistance of several scientists, who used their knowledge of MATALB to build the initial iteration of this table, which was later improved. Because it was verified in a sizable repository, the scientists can be certain that their table, or at least a portion of it, is accurate. This provides the researchers looking into concerns in MATLAB with yet another reassuring boost in motivation for their work.

- **Characterization Study**

  This thesis made a contribution by characterizing the way the MATLAB community uses the language. This characterization study can have the following practical applications:

  - Considering how the community uses MATLAB the most, can help people decide on MATLAB as their preferred language for developing a system. In example if someone would need a language good for mathematical algorithms and with an alternative way of using libraries (many libraries), MATLAB could be a good choice.

  This may also help to offer additional guidance on how to evolve the MATLAB language.

  - Due to the infrequent use of the recently introduced OO features, it may be preferable to also evolve MATLAB's OO features in order to improve library support.

  - A section on system maintainability and modularization could be added to the MATLAB website's main page.

  - MathWorks should possibly promote the calculation features of MATLAB even more.

- **Contribution to the Overall Project over the MATLAB Language**

  One contribution to the overall work of all members involved in this analysis over the years, is the extension to CCCExplorer and the Database with non-CBLC features. This contribution has been proven to be useful with all the conclusions that can be extracted from the analysis.

- **Ideas for the Future**

  With this analysis, several ideas were resolved. One was that certain LCs relate highly to certain concerns, sometimes even more specifically to certain concern tokens. This is significant because in the future, as a result of this analysis, less desirable programming practices (coding anti-patterns) might be discovered. The section on LCs and concerns relations should be used in future studies to pinpoint problematic MATLAB programming practices.

  Additionally, this thesis and it's main ideas can be applied to future research examining this and other programming languages. The following section goes into more detail about this.

9.2 Future Work

Some future work possibilities are the following:

- The apostrophe error that CCCExplorer makes, first explained in Chapter 4, is something that needs to be fixed in the future because it can lead to poor identification of LCs and concern tokens. But this is already being looked into [8].

- Another error with CCCExplorer is that it double stores the apostrophe when storing complete lines of code. This needs to be fixed as well.

- The names of the OO classes need to be stored in memory so that we can know how many times they were instantiated when performing an analysis. This is one area where CCCExplorer's parser may need some work.

- The elimination of the requirement to restart CCCExplorer from scratch each time the repository expands is a further feature that CCCExplorer may find useful. Currently, CCCExplorer must be restarted from scratch if even a single new mfile is added to the repository and needs to be analyzed along with the rest of the repository. This issue needs to be resolved because running CCCExplorer with 452 000 mfiles on a typical laptop took about 3 days. Use of plain text files to store the finishing state of CCCExplorer after a "run" would be one potential fix for this. It only required importing the txt files and applying the most recent finishing state when new mfiles were added to the repository. One method of using these text files would be to convert the objects in CCCExplorer to JSON, then write the JSONs

on the text files. Read the JSONs and create/fill the objects with the JSONs when CCCExplorer is restarted.

- Another future work is the study about schizophrenic functions mentioned in the previous section, the discovery of a general method to dissect a schizophrenic function. It is believable that it would be better for this study to select a number of small systems that each have one function of this kind and then attempt to re-write those systems so that the functions become non-schizophrenic. Then trying to identify a pattern that all re-wrote systems share when a few small systems were redesigned. Finally finding —or attempting to find— a general method of rearranging this kind of functions.

- Regarding the idea enunciated in Section 8.5, about concern tokens relating more frequently to schizophrenic 'switchs' rather than schizophrenic 'ifs', the future work idea about this subject is presented next.

  First, it is believable that almost all junior programmers use 'ifs' rather than 'switches. Since the 'switch' is practically a more complex form of a 'if', even though there is no evidence to support this assertion, it is a reasonable theory.

  Second, because concern tokens typically also have complex functions and very specific functionalities, more experienced programmers use them more often. Given that concerns are being identified by these more complex tokens than the typical programming, it is also possible that more seasoned professionals use these concern tokens more frequently.

  These two beliefs lead to the conclusion that schizophrenic 'switchs' are more closely related to concerns than schizophrenic 'ifs', since more experienced programmers use both concerns and 'switchs' more frequently. Additionally, the correlation is significantly higher because more experienced programmers use both of these features more frequently. Of course, there is no evidence that this is the case in reality. Afterwards in this section, there is more information on the topic of conducting an analysis that takes programmers' experience into account, and that could be used to clarify this situation, by comparing the case of the more season professional to the less experienced ones.

- Perhaps the reason why the two features—Protected and defining the visibility by a List (in both 'properties' and 'methods')—are not more widely used is that junior prgomammers are unaware that they exist or these features were never sufficiently explained to junior programmers. This situation requires further investigation because the evidence for this theory is unclear since we lack the resources to clarify it. If this is the case, some simple but helpful advice should be given. The "art" of programming is constantly evolving and changing. In this way, a programmer can never settle for just knowing the fundamentals or what they mostly remembered

from their studies. For the best possible source code development, programmers should constantly strive to further advance their abilities and knowledge.

Even though the evidence for this theory is unclear, it is still a good idea for junior programmers to be aware of this advise. An analysis including the experience of programmers is presented next. This analysis could also be used to clarify this theory, by comparing the results of the more experienced professionals with the junior programmers.

- Last but not least, a line of potential follow-up works came up. This body of work is a continuation of Reis *et.al* [27]. According to a study that was included in this paper, MATLAB and its clones have some differences between more and less experienced users [27].

The body of future works is as follows:

1. Making an online request for MATLAB files is a suggestion for starting some future work. Asking about the level of expertise that programmers had at the time the mfiles were written. Distribute the mfiles according to the level of expertise, in example experience time in MATLAB. This way having a characterized repository. In each group of mfiles, run CCCExplorer before populating a database. Repeat the queries made throughout this thesis for each database, then examine and contrast the results. In this manner, we could contrast the variations in programming approaches based on the level of experience of the programmers.

2. Another task that should be completed in the future is to review the coding anti-patterns that can be identified through this analysis and that are widespread. It is recommended that all relationships, not just the ones that are the most prevalent, be analysed in order to complete this future work. Because some coding anti-patterns might be used frequently but they might not be among the most prevalent.

3. The use of this type of analysis with other programming languages is yet another project for the future. It would be necessary to adapt CCCExplorer, the LC table, and even the concern table to the new programming languages. Numerous Master's students would need to work on this for many years, as well as many different people. This work would need to be carefully planned out and divided into stages, like master thesis. These phases could include: the necessary rework for CCCExplorer, an updated concern table for the new language, assembling a sufficiently large repository, the analysis using the new LC table, followed by a review of the analysis to find coding anti-patterns that might be somewhat widespread.

4. Applying the previous future work to repositories that are divided by the programmers' level of experience at the time the mfiles were created is another future task. The results of this analysis could be used to compare the coding anti-patterns used

by programmers with various levels of experience. Which are more prevalent, and which are more severe? Is it the obvious outcome that more experienced programmers adhere better to good programming practices? Or do the younger programmers still remember the best practices, whereas the more seasoned ones write source code in their own unique style due to experience?

5. Future works in this area would allow for the analysis of various programming languages in terms of:

   - how are the languages used (a characterization study),

   - what and how frequently are coding anti-patterns used,

   - and the ability to divide both parts according to the experience of the programmers.

   - After that, a comparison study would be possible, comparing each part according to various levels of experience.

This master's thesis was one of many other master's theses, as one can infer from this section. It was also possible to identify a future course of action, or at least a course of action to be considered. This course of action could reveal great potential for the programming community but also great "working" possibilities to future students and the involved professors. All of this was only possible thanks to the advancements made by all of the involved students and professors over the years.

Bibliography

[1] *Acervolima POO Matlab*. https://acervolima.com/programacao-orientada-a-objetos-oops-no-matlab/. Accessed: 2022-9-23 (cit. on p. 46).

[2] J. Barrulas. "Analysis of Code Blocks for Concern Detection in MATLAB Systems". MSc thesis. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2019 (cit. on pp. xvi, 3, 6, 15, 16, 19, 20, 24, 26, 27, 32, 34, 36–39, 61, 71).

[3] J. Cardoso, J. Fernandes, and M. Monteiro. "Adding aspect-oriented features to MATLAB". In: *Workshop on Software Engineering Properties of Languages and Aspect Technologies (SPLAT 2006), within the 5th International Conference on Aspect-Oriented Software Development (AOSD 2006)*. Bona, Germany, Mar. 2006 (cit. on pp. 1, 3, 19, 21, 23, 27, 44, 75).

[4] N. Cavalheiro Marques, M. P. Monteiro, and B. Silva. "Analysis of a token density metric for concern detection in Matlab sources using UbiSOM". English. In: *Expert Systems* 35.4 (Aug. 2018). ISSN: 0266-4720. DOI: 10.1111/exsy.12306 (cit. on pp. xix, 1, 21, 22).

[5] K. Duarte. "Limitations in the Support to Modularity in MATLAB: a Survey-based Empirical Study". MSc thesis. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2017 (cit. on p. 23).

[6] *Free Code Camp Anti-Pattern*. https://www.freecodecamp.org/news/antipatterns-to-avoid-in-code/. Accessed: 2023-6-7 (cit. on p. xviii).

[7] *GeeksForGeeks Information on OO in Matlab*. https://www.geeksforgeeks.org/classes-and-object-in-matlab/. Accessed: 2022-11-13 (cit. on pp. xvi, 11).

[8] A. Grilo. "A case study of Reengineering of the CCCExplorer Tool". unpublished MSc thesis. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2023 (cit. on pp. 42, 43, 76).

[9] L. J. Hendren. "Typing aspects for MATLAB". In: *Proceedings of the sixth annual workshop on Domain-specific aspect languages, DSAL '11, Porto de Galinhas, Brazil, March 21, 2011*. Ed. by T. Dinkelaker, J. Noyé, and É. Tanter. ACM, 2011, pp. 13–18.

DOI: 10.1145/1960496.1960501. URL: https://doi.org/10.1145/1960496.1960501 (cit. on p. 6).

[10] B. Jota. "Métodos para o tratamento de tokens na identificação de concerns em código MATLAB". MSc thesis. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2019 (cit. on pp. 23, 27, 75).

[11] L. Mak. "Are parallel machines always faster than sequential machines?" In: *STACS 94*. Ed. by P. Enjalbert, E. W. Mayr, and K. W. Wagner. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 137–148. ISBN: 978-3-540-48332-8 (cit. on p. 74).

[12] *MathWorks Company Fact Sheet*. https://uk.mathworks.com/content/dam/mathworks/handout/2020-company-factsheet-8-5x11-8282v20.pdf. Accessed: 2021-12-8 (cit. on p. 7).

[13] *MathWorks Documentation*. https://www.mathworks.com/help/matlab/matlab_oop/customize-parentheses-indexing-for-mapping-class.html. Accessed: 2022-11-13 (cit. on pp. xvi, 12).

[14] *MathWorks Documentation*. https://www.mathworks.com/help/parallel-computing/parfor.html. Accessed: 2022-2-7 (cit. on pp. xvi, 13).

[15] *MathWorks Documentation*. https://www.mathworks.com/matlabcentral/answers/178473-fprintf-function-and-how-to-use-it. Accessed: 2022-2-7 (cit. on pp. xvi, 13).

[16] *MathWorks Documentation*. https://www.mathworks.com/help/matlab/ref/if.html. Accessed: 2022-2-7 (cit. on pp. xvi, 14).

[17] *MathWorks Documentation*. https://www.mathworks.com/help/matlab/numeric-types.html. Accessed: 2022-5-29 (cit. on p. 8).

[18] *MathWorks Documentation*. https://www.mathworks.com/help/matlab/matlab_oop/example-representing-structured-data.html. Accessed: 2022-11-13 (cit. on p. 10).

[19] *MathWorks Documentation*. https://www.mathworks.com/help/matlab/matlab_oop/property-attributes.html. Accessed: 2022-11-13 (cit. on p. 10).

[20] *MathWorks Documentation*. https://www.mathworks.com/help/matlab/matlab_oop/specifying-methods-and-functions.html. Accessed: 2022-11-13 (cit. on p. 11).

[21] *MathWorks Documentation*. https://www.mathworks.com/help/matlab/matlab_oop/method-attributes.html. Accessed: 2022-11-13 (cit. on p. 11).

[22] *MathWorks Handle Library*. https://www.mathworks.com/help/matlab/ref/handle-class.html. Accessed: 2023-1-3 (cit. on p. 48).

[23] M. Monteiro. "Identification and Characterization of Crosscutting Concerns in MATLAB Systems". In: *Conference on Compilers, Programming Languages, Related Technologies and Applications (CoRTA 2010)*. Conference date: 01-01-2010. Jan. 2010, pp. 1–12 (cit. on pp. 1–3, 21, 23, 26, 75).

[24] M. Monteiro et al. "Toward a token-based approach to concern detection in MATLAB sources". English. In: *Progress in Artificial Intelligence - 18th EPIA Conference on Artificial Intelligence, EPIA 2017, Proceedings*. Ed. by Z. Vale et al. Vol. 10423 LNAI. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 18th EPIA Conference on Artificial Intelligence, EPIA 2017 ; Conference date: 05-09-2017 Through 08-09-2017. Germany: Springer Verlag, 2017, pp. 573–584. ISBN: 978-331965339-6. DOI: 10.1007/978-3-319-65340-2_47 (cit. on pp. 1, 15, 16, 21, 23, 44, 75).

[25] B. Palma. "A Tool for Mining Concerns in MATLAB Code Repositories". MSc thesis. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2017 (cit. on p. xviii).

[26] E. Reis. "Surveying communities of users of MATLAB and clone languages". MSc thesis. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2021 (cit. on pp. 1, 3).

[27] E. Reis, C. Gralha, and M. Monteiro. "Surveying communities of users of MATLAB and clone languages". English. In: *Journal of Computer Languages* 73 (Dec. 2022). Funding Information: This work is supported by NOVA LINCS, Portugal (UIDB/04516/2020). Publisher Copyright: © 2022 The Author(s). ISSN: 2590-1184. DOI: 10.1016/j.cola.2022.101170 (cit. on p. 78).

[28] A. Relvas. "Uma Interface Web para Comparação de Métricas Utilizando Mapas Auto-Organizados". MSc thesis. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2019 (cit. on p. 27).

[29] A. Relvas et al. "An annotated repository for MATLAB code". English. In: *Proceedings - SEKE 2019: 31st International Conference on Software Engineering and Knowledge Engineering*. Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE. 31st International Conference on Software Engineering and Knowledge Engineering, SEKE 2019 ; Conference date: 10-07-2019 Through 12-07-2019. Knowledge Systems Institute Graduate School, July 2019, pp. 497–502. DOI: 10.18293/SEKE2019-137 (cit. on pp. 1, 27, 34).

[30] J. Santos, N. Marques, and M. P. Monteiro. "Final Report for APDC Course". In: *APDC Course at Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa* (2020) (cit. on pp. 1, 44).

[31] S. Sharma. "Performance Analysis of Parallel Algorithms on Multi-core System using OpenMP". In: *International Journal of Computer Science, Engineering and Information Technology* 2 (Oct. 2012), pp. 55–64. DOI: 10.5121/ijcseit.2012.2506 (cit. on p. 74).

[32] *Stackoverflow Language Construct*. https://stackoverflow.com/questions/10057524/what-does-language-construct-mean. Accessed: 2022-2-15 (cit. on pp. 6, 12).

[33] *Technopedia Module*. https://www.techopedia.com/definition/3843/module. Accessed: 2022-2-12 (cit. on p. 19).

[34] *Tutorials Point MATLAB Operators*. https://www.tutorialspoint.com/matlab/matlab_operators.html. Accessed: 2022-1-10 (cit. on p. 8).

[35] *Wikipedia Crosscutting Concern*. https://en.wikipedia.org/wiki/Cross-cutting_concern. Accessed: 2022-2-16 (cit. on pp. xviii, 20, 21).

[36] *Wikipedia Language Construct*. https://en.wikipedia.org/wiki/Language_construct. Accessed: 2022-2-15 (cit. on pp. 6, 12).

[37] *Wikipedia Language Construct Category*. https://en.wikipedia.org/wiki/Category:Programming_constructs. Accessed: 2022-2-15 (cit. on p. 6).

[38] *Wikipedia Language Construct Different Programing Languages*. https://en.wikipedia.org/wiki/Comparison_of_programming_languages_(basic_instructions). Accessed: 2022-2-15 (cit. on p. 6).

[39] *Wikipedia Modularity*. https://en.wikipedia.org/wiki/Modularity. Accessed: 2022-12-21 (cit. on p. 19).

[40] *Wikipedia Pointwise Mutual Information*. https://en.wikipedia.org/wiki/Pointwise_mutual_information. Accessed: 2022-9-18 (cit. on p. 57).

I

SQL Queries and Schema

Listing I.1: Schema for the Database

```
create table block_types
(id_block_type integer constraint block_types_pk primary key,
name_block_type text,
descriptiom_block_type text);

create table blocks_mfiles
(id_block integer constraint blocks_mfiles_pk primary key,
block_type integer references block_types,
parent_block integer references blocks_mfiles,
id_mfile integer);

create table concerns
(id_concern integer constraint concerns_pk primary key,
name_concern text,
description_concern text);

create table lc_types
(id_lc_type integer constraint lc_types_pk primary key,
name_lc_type text,
description_lc_type text);

create table lcs_mfiles
(lc_id integer constraint lcs_mfiles_pk primary key,
id_lc_type integer references lc_types,
id_mfile integer references mfiles);

create table lines_comments
(id_comment text constraint lines_comments_pk primary key,
id_mfile integer references mfiles,
line integer,
code text);

create table lines_mfiles
(id_line integer constraint lines_mfiles_pk primary key,
```

```
line integer,
code text,
id_mfile integer references mfiles);

create table lines_tokens
(id_line integer references lines_mfiles,
id_block integer references blocks_mfiles,
id_lc_type integer references lc_types,
id_token integer references tokens,
initial_column integer,
final_column integer,
constraint lines_tokens_pk primary key (id_line, initial_column, id_token));

create table mfiles
(id_mfile integer constraint mfiles_pk primary key,
id_toolbox integer constraint mfiles_toolboxes_id_toolbox_fk references
    toolboxes (id_toolbox),
name_mfile text);

create table tokens
(id_token integer constraint tokens_pk primary key,
id_concern integer references concerns,
name_token text,
desc_token text);

create table toolboxes
(id_toolbox integer constraint toolboxes_pk primary key,
name_toolbox text);
```

Listing I.2: Indexes

```
CREATE INDEX "id_block_id_line_lines_tokens" ON "lines_tokens" (
  "id_block"  ASC,
  "id_line" ASC
)

CREATE INDEX "id_block_id_token_lines_tokens" ON "lines_tokens" (
  "id_block"  ASC,
  "id_token"  ASC
)

CREATE INDEX "id_lc_type_lines_tokens" ON "lines_tokens" (
  "id_lc_type"  ASC
)

CREATE INDEX "id_lct_id_line_lines_tokens" ON "lines_tokens" (
  "id_lc_type"  ASC,
  "id_line" ASC
)

```

```
CREATE INDEX "id_line_id_lct_id_token_lines_tokens" ON "lines_tokens" (
  "id_line" ASC,
  "id_lc_type"  ASC,
  "id_token"  ASC
)

CREATE INDEX "id_token_lines_tokens" ON "lines_tokens" (
  "id_token"  ASC
)
```

Listing I.3: Views

```
CREATE VIEW "00_mfiles_idmfile_idtoolbox" AS select id_toolbox, id_mfile
from(
select DISTINCT id_line
from lines_tokens INDEXED BY id_line_id_lct_id_token_lines_tokens
where id_token = 3381) t1
natural inner join lines_mfiles natural inner join mfiles

CREATE VIEW id_line_all_lines_related_to_main_in_one_function_mfiles
AS
select Distinct id_line
from
one_function_CBLC_mfiles_id_mfile
natural inner join lines_mfiles
natural inner join lines_tokens INDEXED BY id_block_id_line_lines_tokens
natural inner join blocks_mfiles
where block_type = 1

CREATE VIEW "one_function_CBLC_mfiles_id_mfile" AS select id_mfile
from
  (
  select id_mfile, count(id_block) as b
  from
    (
    select id_block, id_mfile
    from blocks_mfiles
    WHERE block_type = 2 and id_mfile NOT IN (select id_mfile
                          from 00_mfiles_idmfile_idtoolbox)
    )t1
  group by id_mfile
  )t2

where b = 1


CREATE VIEW "schizo_idmfile_idtoolbox" AS select id_toolbox, id_mfile
from blocks_mfiles inner join
    lines_tokens INDEXED BY id_block_id_token_lines_tokens using(id_block)
    natural inner join mfiles
```

```
where block_type = 3 and id_token = 197
UNION
select id_toolbox, id_mfile
from blocks_mfiles inner join
    lines_tokens INDEXED BY id_block_id_token_lines_tokens using(id_block)
    natural inner join mfiles
where block_type = 6 and id_token = 197
```

Listing I.4: Queries used in the Analysis of OO Systems

```
select id_token
from tokens
where name_token = 'classdef'
group by name_token

--id_token = 3381--

--OO features--
--Amount of OO mfiles
select count(DISTINCT id_mfile)
from(
select DISTINCT id_line
from lines_tokens INDEXED BY id_line_id_lct_id_token_lines_tokens
where id_token = 3381) t1
natural inner join lines_mfiles;

--Amount of mfiles in the repository to calculate the percentage
select count(*)
from mfiles;

--Amount of OO toolboxes
select count(DISTINCT id_toolbox)
from(
select DISTINCT id_mfile
from(
select DISTINCT id_line
from lines_tokens INDEXED BY id_line_id_lct_id_token_lines_tokens
where id_token = 3381) t1
natural inner join lines_mfiles) t2
NATURAL inner join mfiles;

--Amount of OO mfiles in each toolbox to calculate mean and std.dv.
select DISTINCT id_toolbox, name_toolbox, count (*)
from(
select DISTINCT id_mfile
from(
select DISTINCT id_line
from lines_tokens INDEXED BY id_line_id_lct_id_token_lines_tokens
where id_token = 3381) t1
natural inner join lines_mfiles) t2
```

```
NATURAL inner join mfiles natural inner join toolboxes
group by id_toolbox;

--Amount of toolboxes in the repository to calculate the percentage
select count(*)
from toolboxes;

--INHERITANCE--
--Amount of inheritance mfiles
select count(DISTINCT id_mfile)
from(
select a.id_line as id_line
from lines_tokens as a INDEXED BY id_line_id_lct_id_token_lines_tokens
     ,lines_tokens as b INDEXED BY id_line_id_lct_id_token_lines_tokens
where a.id_lc_type = 11 and b.id_token = 3381 and a.id_line = b.id_line
) t1
natural inner join lines_mfiles;

--Amount of inheritance toolboxes
select count(DISTINCT id_toolbox)
from(
select DISTINCT id_mfile
from(
select a.id_line as id_line
from lines_tokens as a INDEXED BY id_line_id_lct_id_token_lines_tokens
     ,lines_tokens as b INDEXED BY id_line_id_lct_id_token_lines_tokens
where a.id_lc_type = 11 and b.id_token = 3381 and a.id_line = b.id_line
) t1
natural inner join lines_mfiles) t2
NATURAL inner join mfiles;

--Amount of inheritance mfiles in each toolbox to calculate mean and std.dv.
select DISTINCT id_toolbox, name_toolbox, count (*)
from(
select DISTINCT id_mfile
from(
select a.id_line as id_line
from lines_tokens as a INDEXED BY id_line_id_lct_id_token_lines_tokens
     ,lines_tokens as b INDEXED BY id_line_id_lct_id_token_lines_tokens
where a.id_lc_type = 11 and b.id_token = 3381 and a.id_line = b.id_line
) t1
natural inner join lines_mfiles) t2
NATURAL inner join mfiles natural inner join toolboxes
group by id_toolbox;

--INHERITANCE AND LIBRARIES
select id_token
from tokens
where name_token = '.'
group by name_token;
```

```
--270

--inheritance mfiles related to libraries with '.'
select DISTINCT name_toolbox, name_mfile
from(
select DISTINCT id_mfile
from
(select DISTINCT id_line
from
(select DISTINCT a.id_line as id_line
from lines_tokens as a INDEXED BY id_line_id_lct_id_token_lines_tokens
     ,lines_tokens as b INDEXED BY id_line_id_lct_id_token_lines_tokens
where a.id_lc_type = 11 and b.id_token = 3381 and a.id_line = b.id_line
) t1
NATURAL inner join lines_tokens INDEXED BY
    id_line_id_lct_id_token_lines_tokens
NATURAL inner join tokens
where id_token = 270
)t3
natural inner join lines_mfiles) t2
NATURAL inner join mfiles natural inner join toolboxes

--inheritance mfiles related to the 'handle' library
select DISTINCT name_toolbox, name_mfile
from(
select DISTINCT id_mfile
from
(select DISTINCT id_line
from
(select DISTINCT a.id_line as id_line
from lines_tokens as a INDEXED BY id_line_id_lct_id_token_lines_tokens
     ,lines_tokens as b INDEXED BY id_line_id_lct_id_token_lines_tokens
where a.id_lc_type = 11 and b.id_token = 3381 and a.id_line = b.id_line
) t1
NATURAL inner join lines_tokens INDEXED BY
    id_line_id_lct_id_token_lines_tokens
NATURAL inner join tokens
where name_token = 'handle'
)t3
natural inner join lines_mfiles) t2
NATURAL inner join mfiles natural inner join toolboxes



----------METHODS and PROPERTIES BLOCKS
--Access/GetAccess/SetAccess followed by Public/Private/Protected
select  count(*) -- code, id_line --
from lines_mfiles
natural inner join blocks_mfiles
natural inner join
```

```
(
select a.id_line , a.id_block
from lines_tokens as a,
     lines_tokens as b
where a.id_line = b.id_line
and a.id_token = 3384 -- Access / GetAccess / SetAccess
and b.id_token = 3539 -- Public / Private / Protected
and a.ROWID = (b.ROWID - 2)
) t1
where block_type = 12;

--Total ACCESS
select count(*) -- code, id_line --
from lines_mfiles
natural inner join blocks_mfiles
natural inner join
(
select a.id_line , a.id_block
from lines_tokens as a
where  a.id_token = 3384 --Access / GetAccess / SetAccess
) t1
where block_type = 12;

--GetAccess and SetAccess used together
select count(*) -- code, id_line --
from lines_mfiles
natural inner join blocks_mfiles
natural inner join
(
select a.id_line , a.id_block
from lines_tokens as a,
     lines_tokens as b
where a.id_line = b.id_line
and a.id_token = 3433 -- GetAccess
and b.id_token = 3435 -- SetAccess
) t1
where block_type = 11;
```

Listing I.5: Queries used in the Analysis of the Most Commun LCs

```
--Most Prevalent LCs
select name_lc_type, description_lc_type, count(id_lc_type)
from lcs_mfiles natural inner join lc_types
group by id_lc_type
union all
select name_block_type, id_block_type , count(id_block)
from blocks_mfiles inner join block_types on block_type = id_block_type
group by block_type;

------------------SCRIPTS OR FUNCTIONS---------------
```

```
--Number of 'function' blocks per mfiles
with parents as
(
  select distinct id_block
  from blocks_mfiles
  where block_type = 12
)
select b, count(*) as c
from
  (
  select id_mfile, count(id_block) as b
  from
    (
    select id_block, id_mfile
    from blocks_mfiles
    WHERE block_type = 2 and NOT EXISTS (SELECT *
                        FROM parents p
                        WHERE p.id_block = parent_block)
    )t1
  group by id_mfile
  )t2
GROUP by b


--Count for 0 occurrences
SELECT count(*)
from
(
select id_mfile
from mfiles
EXCEPT
select id_mfile
from blocks_mfiles
WHERE block_type = 5
)t1

---MANUAL ANALYSIS
select name_mfile, name_toolbox
from toolboxes
natural inner join mfiles
NATURAL inner join
(
select id_mfile
from mfiles
EXCEPT
select id_mfile
from blocks_mfiles
WHERE block_type = 2
)t1
```

Listing I.6: Queries used in the Analysis of Relations LCs/Concerns

```
--Amount of Concerns in DB--
select name_concern, id_concern, count(*)
from lines_tokens natural inner join tokens natural inner join concerns
group by id_concern

--Relations LCs(Code Blocks)/Concerns--
select name_block_type, block_type, id_concern, count(*)
from blocks_mfiles inner join block_types on block_type = id_block_type
inner join lines_tokens using(id_block)
inner join tokens using(id_token)
group by block_type, id_concern

--Relations LCs(Non-Code Blocks)/Concerns--
select a_id_lc_type, id_concern, count(*)
from
(
select a.id_line as a_id_line, b.id_line as b_id_line,
a.id_lc_type as a_id_lc_type,
b.id_token as b_id_token
from lines_tokens as a INDEXED BY id_lct_id_line_lines_tokens
     ,lines_tokens as b INDEXED BY id_line_id_lct_id_token_lines_tokens
) t1
inner join tokens on id_token = t1.b_id_token
where t1.a_id_line = t1.b_id_line and a_id_lc_type = 21
group by a_id_lc_type, id_concern;
```

Listing I.7: Queries used in the Analysis of Relations LCs/Tokens of a Specific Concern

```
--Amount of tokens for each concern--
select id_concern, id_token, name_token, count(*)
from lines_tokens natural inner join tokens
where id_concern between 1 and 35
group by id_token
order by id_concern;

--Relations LCs(Code Blocks)/Tokens of a Specific Concern--
select name_block_type, block_type, id_concern, name_token, id_token, count(*)
from blocks_mfiles inner join block_types on block_type = id_block_type
inner join lines_tokens using(id_block)
inner join tokens using(id_token)
where id_concern = 35 and block_type = 7
group by block_type, id_token

--Relations LCs(Non-Code Blocks)/Tokens of a Specific Concern--
select name_lc_type ,a_id_lc_type, id_concern, name_token, id_token, count(*)
from
(
select a.id_line as a_id_line, b.id_line as b_id_line,
```

```
a.id_lc_type as a_id_lc_type,
b.id_token as b_id_token
from lines_tokens as a INDEXED BY id_lct_id_line_lines_tokens
     ,lines_tokens as b INDEXED BY id_line_id_lct_id_token_lines_tokens
where a.id_lc_type = 3 AND a.id_line = b.id_line
) t1
inner join tokens on id_token = t1.b_id_token inner join lc_types on
    id_lc_type = a_id_lc_type
where id_concern = 4
group by a_id_lc_type, id_token;
```

Listing I.8: Queries used in the Analysis of Schizophrenic Functions

```
select id_token, name_token
from tokens
where name_token like 'nargin'
--197,nargin


--distinct mfile ids that contain at least 1 if block that has at least 1
    nargin inside.
--197 nargin           3 if   6 switch
select count(distinct id_mfile)
from blocks_mfiles inner join lines_tokens INDEXED BY
    id_block_id_token_lines_tokens using(id_block)
where block_type = 3 and id_token = 197;


--Relations of if blocks that have at least 1 nargin and relate to concern
    tokens of the concern ids between 1 and 35.
select count(*)
from (
select distinct id_block from blocks_mfiles inner join lines_tokens INDEXED BY
     id_block_id_token_lines_tokens using(id_block)
where block_type = 3 and id_token = 197
)
inner join lines_tokens INDEXED BY id_block_id_token_lines_tokens using(
    id_block) inner join tokens using(id_token)
where id_concern between 1 and 35;


--Number of hierarchically associated concern tokens to switch+nargin blocks
with parents as
(
    select id_block , block_type , parent_block
    from (
    select distinct id_block , block_type , parent_block
    from blocks_mfiles inner join
        lines_tokens INDEXED BY id_block_id_token_lines_tokens using(id_block)
    where id_token = 197 and block_type = 6
```

```
    )
union all
    select b.id_block , b.block_type , b.parent_block
    from blocks_mfiles b inner join parents p on b.parent_block = p.id_block
)
select id_concern , count(id_token)
from parents inner join lines_tokens INDEXED BY id_block_id_token_lines_tokens
     using(id_block) inner join tokens using(id_token)
where id_concern between 1 and 35
group by id_concern;
```

Listing I.9: Queries used in the Analysis of Schizophrenic Functions Relation with the OO paradigm

```
--Creation of View with mfiles with schizophrenia
select id_toolbox, id_mfile
from blocks_mfiles inner join
    lines_tokens INDEXED BY id_block_id_token_lines_tokens using(id_block)
    natural inner join mfiles
where block_type = 3 and id_token = 197
UNION
select id_toolbox, id_mfile
from blocks_mfiles inner join
    lines_tokens INDEXED BY id_block_id_token_lines_tokens using(id_block)
    natural inner join mfiles
where block_type = 6 and id_token = 197;

--Creation of View with OO mfiles
select id_toolbox, id_mfile
from(
select DISTINCT id_line
from lines_tokens INDEXED BY id_line_id_lct_id_token_lines_tokens
where id_token = 3381) t1
natural inner join lines_mfiles natural inner join mfiles;

select count(*)
from schizo_idmfile_idtoolbox;

select count(DISTINCT id_toolbox)
from schizo_idmfile_idtoolbox;

--Schizophrenic functions in OO
select DISTINCT id_toolbox, count (*)
from(
select DISTINCT id_toolbox, id_mfile
from schizo_idmfile_idtoolbox
where id_mfile IN (select id_mfile
                   from OO_mfiles_idmfile_idtoolbox))
group by id_toolbox;

```

```
37 --Schizophrenic functions not in OO
38 select DISTINCT id_toolbox, count (*)
39 from(
40 select DISTINCT id_toolbox, id_mfile
41 from schizo_idmfile_idtoolbox
42 where id_mfile NOT IN (select id_mfile
43                        from OO_mfiles_idmfile_idtoolbox))
44 group by id_toolbox;
```