

# INSPECTING NEW DATA STRUCTURES FOR STATIC ANALYSIS

by  
CONSTANTINOS MAKRIS  
URN: 6544731



A dissertation submitted in partial fulfilment of the  
requirements for the award of

BACHELOR OF SCIENCE IN COMPUTING AND INFORMATION  
TECHNOLOGY

May 2021

Department of Computer Science  
University of Surrey  
Guildford GU2 7XH

Supervised by: Dr Santanu Dash

I declare that this dissertation is my own work and that the work of others is acknowledged and indicated by explicit references.

Constantinos Makris  
May 2021

© Copyright Constantinos Makris, May 2021

# Acknowledgements

Firstly, I would like to thank my family for supporting me throughout this difficult and important year, as none of this would have been possible without their support. I would like to also thank my supervisor Dr Santanu Dash for the opportunity to work on this project, but also for his guidance and support over the year. Finally, I want to show my gratitude to his PhD student Constantin Cezar (Costin) Petrescu for his support and assistance during this project.

# Abstract

The continually rising size of modern software projects has led to long and computationally expensive Static Analyses. Many developers need Static Analysis to keep a correct and secure code, but its resources' costly execution is a drawback of the tool. A more efficient and optimized process would be a great advancement for the analysis.

This research project introduces a solution for the optimization of Static Analysis and identifies its overall capabilities and potential success. It uses a C++ compiler pass for the LLVM compiler to find how relations between parent and child parts of code are able to spot the most important parts of a program. This is done by breaking down the code into smaller sections, identifying relations between the parts, and analyzing them one by one. The comparison between a parent and child is done by applying information metrics to their natural language and programming identifiers, to discover their similarity.

Throughout this research process, an interesting and promising project has been developed. A deeper analysis of some of the cases led to the conclusion that the information provided from the parts of the code used is insufficient to be able to make important decisions based on them. Although a slightly different approach is suggested for Future Work, that has a great potential for success.

# Contents

<b>1</b>	<b>Introduction</b>	<b>13</b>
1.1	Background and Overview . . . . .	13
1.1.1	Static Analysis . . . . .	14
1.2	Aims and Objectives of the project . . . . .	14
1.3	Success Criteria . . . . .	15
1.4	Structure of the Report . . . . .	15
<b>2</b>	<b>Literature Review</b>	<b>17</b>
2.1	Specific Problem . . . . .	17
2.2	Importance . . . . .	18
2.3	Relevant Work . . . . .	18
2.3.1	Natural Language . . . . .	18
2.3.2	Identifier Names . . . . .	18
2.4	Key technologies . . . . .	19
2.4.1	Python . . . . .	19
2.4.2	JSON . . . . .	19
2.4.3	LLVM pass . . . . .	20
2.5	Basic Blocks . . . . .	20
2.6	Bitcode . . . . .	20

2.7 Levenshtein Distance . . . . .	21
<b>3 System Requirements and Specification</b>	<b>22</b>
3.1 System Requirements . . . . .	22
3.1.1 Functional Requirements . . . . .	22
3.1.2 Non-Functional Requirements . . . . .	23
3.2 Specification . . . . .	23
3.3 Planning . . . . .	24
3.3.1 Gantt chart breakdown . . . . .	24
3.4 Software Development Life Cycle . . . . .	24
3.4.1 Models . . . . .	26
3.4.1.1 Waterfall . . . . .	26
3.4.1.2 Iterative . . . . .	26
3.4.1.3 Agile . . . . .	26
3.4.2 Chosen model . . . . .	27
3.5 Feasibility Study . . . . .	27
3.5.1 Technical Feasibility . . . . .	27
3.5.2 Operational Feasibility . . . . .	28
3.5.3 Economic Feasibility . . . . .	28
3.5.4 Legal Feasibility . . . . .	28
3.5.5 Schedule Feasibility . . . . .	28
<b>4 System Design and Implementation</b>	<b>29</b>
4.1 Chosen Approach . . . . .	29
4.2 Boost Library data . . . . .	30
4.3 Extraction of Basic Blocks . . . . .	30
4.3.1 LLVM pass functionality . . . . .	30

## Contents

---

4.3.2	JSON file format . . . . .	31
4.4	Data Analysis tool . . . . .	32
4.4.1	Processing the data . . . . .	32
4.4.2	Encoding . . . . .	32
4.4.3	Levenshtein Distance and Token Count . . . . .	33
4.4.4	Visualising the results . . . . .	33
4.5	Evaluation Process . . . . .	34
4.5.1	Outliers Selection . . . . .	34
4.5.1.1	Process . . . . .	34
4.5.2	Sampling . . . . .	36
4.5.3	Module separation . . . . .	36
<b>5</b>	<b>Testing and Evaluation</b>	<b>37</b>
5.1	Quantitative Analysis . . . . .	37
5.2	Qualitative Analysis . . . . .	38
5.2.1	Case 1 . . . . .	39
5.2.2	Case 2 . . . . .	40
5.2.3	Case 3 . . . . .	41
5.3	Results Discussion . . . . .	42
<b>6</b>	<b>Conclusions and Future Work</b>	<b>43</b>
6.1	Conclusions . . . . .	43
6.2	Future Work . . . . .	44
<b>7</b>	<b>Appendix</b>	<b>45</b>
7.1	Statement of Ethics . . . . .	45
7.2	Public Interest - Do no harm: Ethical . . . . .	45

## Contents

---

7.3	Informed Consent: Legal . . . . .	45
7.4	Confidentiality of Data: Legal . . . . .	46
7.5	Social Responsibility: Social . . . . .	46
7.6	Professional Competency and Integrity . . . . .	46

# List of Figures

2.1 Basic Blocks identification example . . . . .	20
3.1 Gantt chart timetable . . . . .	25
4.1 Data Extraction Process . . . . .	30
4.2 JSON file structure . . . . .	31
4.3 An example of a call in its first format . . . . .	32
4.4 An example of a call in its encoded format . . . . .	32
4.5 An example of a call with its Levenshtein Distance and Parent Tokens Count . .	33
4.6 Scatter plot of all the cases . . . . .	34
4.7 Outliers JSON file structure . . . . .	35
4.8 Sample cases in CSV format . . . . .	36
5.1 'log' module calls scatter . . . . .	39
5.2 'filesystem' module calls scatter . . . . .	39
5.3 'locale' module calls scatter . . . . .	39
5.4 Scatter of all the rest modules . . . . .	39
5.5 First case call . . . . .	39
5.6 Case 1 lines of code . . . . .	40
5.7 Second case call . . . . .	40
5.8 Case 2 lines of code . . . . .	41

## List of Figures

---

5.9	Third case call . . . . .	41
5.10	Case 3 lines of code . . . . .	41

# List of Tables

5.1 Boost Library Data. Displays the library modules along with the number of calls and the outlier cases they contain . . . . .	38
---	----

# Abbreviations

BB	Basic Block
SDLC	Software Development Life Cycle
BCS	British Computer Society
JSON	Java Script Object Notation
CSV	Comma Separated Values file

# Chapter 1

## Introduction

### 1.1 Background and Overview

As technology is advancing, software code is a critical bit for the digitization of it. Every day, more and more people are using various programming languages to create programs for their jobs or for personal use. There are unlimited languages, tools, and technologies whose task is to assist the programmer in the software developing process. The main challenge and difficulty that a developer will face during the development of software is not the implementation of it, but its correctness, security, and maintainability. A programmer can be supported in these challenges by the use of a Static Code Analysis tool. Such tool can help the developer create error-less, secure, and maintainable software as explained later in this chapter.

A concerning matter about Static Analysis is the size of projects that is constantly growing. The computational cost that is needed to perform a Static Analysis on a large project is tremendous. An example is Boost C++ library which is an open-source project whose source code size is about 850Mb. To analyse projects as big as this, a huge computational cost and time are needed. This is the challenge that we want to face; to minimize the computational time that is required to perform Static Analysis on software.

The idea is to improve its time to analyse software. This will be done by decreasing the amount of code that the tool will be analyzing. This is supported by the fact that not all the code needs to be analysed. Some parts of the code may be less important to what the software does than other parts. Such code can be excluded from the Static Analysis execution so that only the parts that are the most important are analysed. Moreover, such findings may not only be relevant

## 1.2. Aims and Objectives of the project

---

to the Static Analysis, but also to the developers of software that are looking to refactor their code.

### 1.1.1 Static Analysis

Static Analysis is a method of debugging code without executing it. The source code is analysed for parts that are associated with errors and vulnerabilities. If any are spotted in the code, they are reported back to the developer. The analysis is performed before the testing phase of the Software Development Life Cycle. It can analyse every single path of software in contrast with testing that only checks some parts of the code.

This way of analyzing programs serves multiple roles in the development process. It ensures code quality to the developer by reporting any issue with the code. It also provides the big data that are required to improve the development process. Moreover, it eases the machinations of the DevOps automated feedback loop.

Static Code Analysis follows a sequence of tasks to analyse a piece of software. At first, it gets the code and adds it to an intermediary model. Then, it runs analysers on the model, and finally, it reports any vulnerabilities or warnings found.

## 1.2 Aims and Objectives of the project

The general aim of this project is to implement and investigate a software code analysis tool. The tool will be responsible for the collection of data from the source code, their analysis, and the presentation of the results. The research project will test and evaluate this approach to discover and analyse the findings on the topic and prove its potential success. The purpose of this project is to minimize the Static Analysis execution time.

The objectives of the project are:

- To create a tool that extracts the tokens (variable names, functions etc.) and store them based on the part of the code in which they belong to, from a C++ program.
- To create an analyser that accepts these data and finds the similarity between different parts of the code and identifies the less important parts of it.
- To find parts of the analysed code that are more important than the rest of the program.

### **1.3. Success Criteria**

---

- The visualisation of the findings and plotting of useful graphs.
- To find a method to test the correctness and efficiency of the overall software.

## **1.3 Success Criteria**

When this project is done, its success will be determined from some high level criteria that need to be evaluated. These are some requirements originated from the project's aims and objectives that the need to be completed in order to call it a success.

- A functioning implementation of a tool that extracts tokens from a C++ program.
- A functioning implementation of a software that gets the data and analyses them.
- Correct visualisation and presentation of the findings using graphs and tables

## **1.4 Structure of the Report**

- Chapter 1: Introduction

This chapter gives a general overview of the project and the report as well as some background knowledge on the topic.

- Chapter 2: Literature Review

Defines the specific problem along with the general topic of the project and relevant work.

- Chapter 3: System Requirements and Specification

Defining the requirements and specification of the project and conducting feasibility study.

- Chapter 4: System Design and Implementation

Details how the project is designed and implemented.

## 1.4. Structure of the Report

- Chapter 5: Testing and Evaluation

The results of the software testing and evaluation of the project

- Chapter 6: Conclusions and Future Work

The conclusions of this project and future work that could be done on this project or overall topic.

- Chapter 7: Appendix

Statement of Ethics along with Legal, Social, Ethical and Professional issues.

# Chapter 2

## Literature Review

This chapter's scope is to introduce the problem that this project aims to face and the importance of the project. Moreover, it will present some relevant work that was done on the topic and detail the key technologies that were used to implement the software.

### 2.1 Specific Problem

The problem that this project's implementation aims to face, was also the motivation to face this challenge. Making even a small contribution to the software engineering community is something inspiring that drives the completion of this project.

The huge sizes and the growth rates of modern software engineering projects make the Static Analysis function even harder. The computational resources and time needed to analyse a large project are massive. This is because Static Analysis goes through every single path in the code that analyses. This is not ideal, as it is unnecessary to analyse some parts of the code.

This project will attempt to create a tool that identifies those parts so that they could be excluded from Static Analysis. This solution will decrease the computing resources needed for a Static Analysis, without affecting the important parts of the software.

## 2.2 Importance

This project is important because it is something that was not applied before. Using Information Theory Metrics to filter the code that will be analysed from Static Analysis is a new method for this tool. It has a great chance of helping the developer to understand the most important or less important parts of his project, which is something not done yet with this approach. The potential for this method to be successful and used further by programmers makes it important and interesting to work with.

## 2.3 Relevant Work

### 2.3.1 Natural Language

For the paper 'On the Naturalness of Software' (Hindle, Barr, Su, Gabel & Devanbu 2012), they began by the conjecture that because software is created by humans, it includes a naturalness. If that was correct, the code was also likely to be repetitive and predictable, just like natural language is. The idea was to analyse the repetitive patterns and use them to create tools that would assist software engineering. They used the n-gram model which is a probabilistic language model used to predict the next item in a sequence and they have shown that code is in fact even more repetitive than natural language. To give an example of the model, they developed a Java code completion engine, that performed better than the Eclipse's code completion engine. This research proves the direct relationship between code and natural language, and that the code identifiers can give a lot of information about the code. This conclusion gives potential to our research, as the code identifiers are the key information of the project.

### 2.3.2 Identifier Names

In their paper 'Deepbugs: A learning approach to name-based bug detection' (Pradel & Sen 2018), Pradel and Sen stated that many popular static analysis tools ignore the identifier names. This leads to the problem that those bug detection tools, miss obvious bugs that a human can identify by reading the code. Their approach to face this problem was to create a machine learning-based framework, called DeepBugs. The tool extracts correct and incorrect training examples from code and trains the model after applying a simple transformation to create more

data. The model is then tested on code that the model did not train with to find its success rate. When DeepBugs was evaluated by training and testing with 68 million lines of JavaScript code, it performed with 89% to 95% accuracy. This indicates the great success of the approach. This project's greatest achievement is that it has proven that code identifier names are something really important in a code that cannot be ignored by analysis tools. Based on this finding, our project also uses natural language identifiers to make new observations in the code.

## 2.4 Key technologies

In this section, the key technologies that are used in this project are introduced and it is detailed why they are chosen to serve each purpose.

### 2.4.1 Python

Python programming language (*About Python* 2021) is the most popular programming language today, overtaking Java in 2018 and now owning almost 30% of the google searching share between programming languages according to PYPL (Carbonnelle 2020). It is an easy language for someone that wants to learn it or to transfer to it from another programming language. It is a dynamically typed language which means that the interpreter does type checking only when the program runs. Moreover, it has a huge online community that would support anybody who has any kind of issues or questions. Its thousands of libraries are a huge benefit because they support the developer with their functionalities and they make some tasks very simple. Finally and most importantly, it is open-source software that anybody can use for free.

### 2.4.2 JSON

Java Script Object Notation(JSON) (*Introducing JSON* 2021) is a data interchange format that is human-readable and easily parsed by computers. It is a language-independent text format that was derived from the JavaScript programming language, but many languages are today able to read from and write to a JSON file. It is built on two structures; a collection of key-value pairs and an ordered list of values.

### 2.4.3 LLVM pass

The LLVM pass is a compiler pass of the LLVM system that is a subclass of the Pass class and is where the compiler's most interesting parts exist. The passes can perform transformations and optimizations that make up the compiler and they build the analysis results that are used by these transformations. They are a structuring technique for compiler code.

## 2.5 Basic Blocks

Basic Blocks(BB) are sequences of code that execute one after the other and together form the whole program. A basic block is identified if there is a leader in the code. A leader could be either the first line of the code, or if there is a conditional or unconditional goto statement, or if it immediately follows a goto or conditional goto statement. I have used Basic Blocks to separate the program into small blocks in order to analyse it. The following code shows how Basic Blocks are located in code.

Figure 2.1: Basic Blocks identification example

```
1) x=0           //Leader 1 - First Statement
2) y=0
3) x = x * 2    //Leader 2 - Target of the 5Th statement
4) y = y / 4
5) if x < y goto(3)
6) y = y + 1    //Leader 3 (Immediately following Conditional goto statement)
```

This example code consists of 3 Basic Blocks. Lines 1-2, 3-4, and 5-6

## 2.6 Bitcode

Bitcode is program code that is compiled from a high-level (source code) into low-level code, that the computer processor will understand. It is usually executed by a virtual machine or

compiled again into machine code that the computer processor understands.

The key characteristic that Bitcode has and interests us is the ability to be read both by us and the LLVM pass. It still contains the tokens of the code in their natural language so we are able to grab those and use them for further analysis.

## 2.7 Levenshtein Distance

The metric that Vladimir Levenshtein considered in 1965, was named Levenshtein Distance after him. It measures the similarity between two sequences of data by calculating an edit distance. It is the minimum number of single-character or single-bit edits needed, for the set to look like the other one. A characteristic of the metric is that it can compare sets of different sizes. The edits that are performed and counted in the metric can be a substitution, deletion, or insertion.

Below, a few examples of Levenshtein distance are presented.

Example 1:

```
home -> hose  
[1 substitution (m -> s)]  
Levenshtein Distance = 1
```

Example 2:

```
immortal -> portal  
[2 deletions (i, m), 1 substitution (m -> p)]  
Levenshtein Distance = 3
```

Example 3:

```
march -> variety  
[2 insertions (t, y), 3 substitutions ( m -> v, c -> i, h -> e)]  
Levenshtein Distance = 5
```

# Chapter 3

# System Requirements and Specification

This chapter's role is to provide an overview of the project's scope. It will be detailing the requirements and specifications of the system along with the software engineering methods that were applied. The plan that was created to guide this project will also be detailed.

## 3.1 System Requirements

This section is going to detail the requirements of the project. Their purpose is to ensure the completion and success of the project. There are two types of requirements, Functional and non-Functional.

### 3.1.1 Functional Requirements

These requirements describe the functionalities and features of the software from the user's perspective. They need to be followed and incorporated into the project. This tool performs a specific task, so there are only a few Functional Requirements. Moreover, this is a research project looking into a new approach and validating its feasibility, thus there is no user interaction, meaning that the functional requirements are limited.

- The extraction software should process the submitted source code, and output a file directory with the data in JSON format.
- The analysis tool should read the data from all the folders in the directory and input them to the system.

### **3.2. Specification**

---

- The system should analyse the data and output the results
- The graphs created should appear on the screen.

#### **3.1.2 Non-Functional Requirements**

The non-functional requirements detail how the software should respond to certain functionalities.

- After the execution of the LLVM pass, the created files should be available in the directory that the user set.
- When the data analysis happens, the results should be presented in the command line, and the graphs should appear on the screen.
- The user has to be able to edit the source code of the tool to set the appropriate settings.
- The tool must not store any sensitive data, apart from the data stored on the users' machine.

## **3.2 Specification**

The project's specification details the software and hardware requirements that a computer needs in order to run the software.

- Operating System
  - Windows: 7 or newer
  - MAC: OS X v10.7 or higher
  - Linux: Ubuntu

Hardware Specifications:

- Screen
- Mouse and Keyboard

### 3.3 Planning

The planning of the project is an important part that I followed from the beginning of the year. It provided the timeline for the different parts of the project. It includes the project planning phase, the implementation phase, and the final report phase.

A Gantt chart is used as shown in Figure 3.1 to organize the schedule. This method is chosen because it is a clear and simple way to keep track of tasks along with their planned duration and their actual duration. It also provides the percentage of each task's completion which helps keep on track with the timetable.

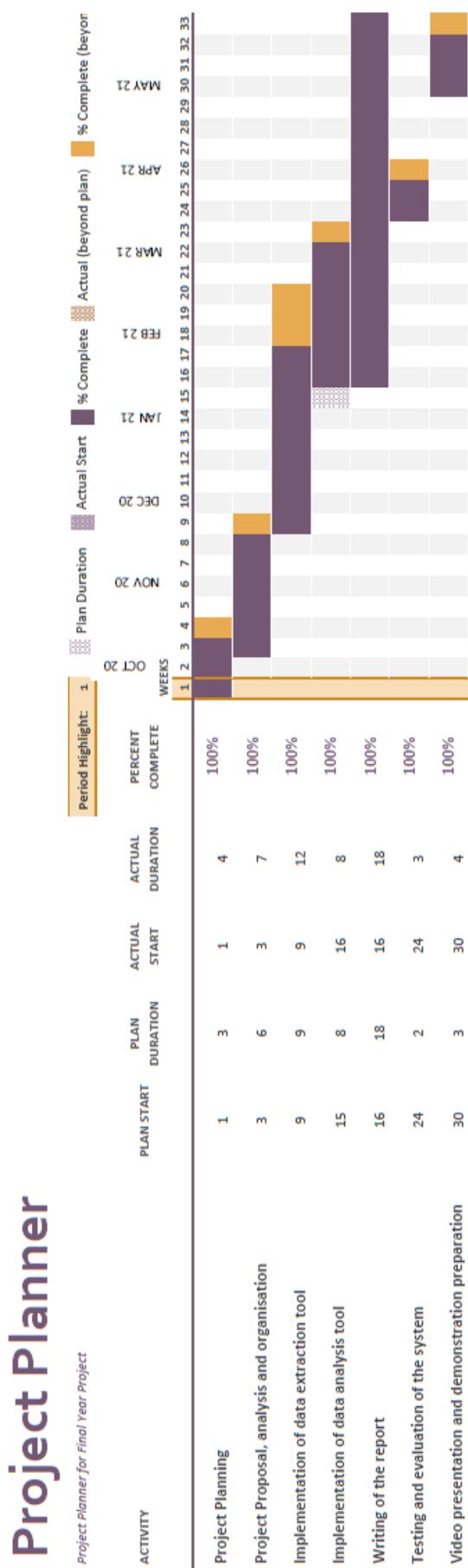
#### 3.3.1 Gantt chart breakdown

The project plan is divided into 7 sections, specified as Activities in the figure. The first activity is "Project Planning" which was responsible for finding the main idea and transforming it into an actual project plan. The next activity is "Project Proposal, Analysis, and Organization" whose aim was to propose the project, create the project plan, set the aims, objectives, and requirements of this project. Following is the "Implementation of Data Extraction Tool" section. Its responsibility was to complete the implementation of the software that extracts the data and adds them to the appropriate folders. After that, the "Implementation of Data Analysis Tool" whose aim was to provide the software that performs the data analysis and provides the results. Then, the longest activity of the plan "Writing of the Report", which was responsible for creating this project's report. The next step was "Testing and Evaluation of the System", whose responsibility was to test the implementations of the project and evaluate their results. Finally, the "Video Presentation and Demonstration Preparation" activity, with the aim to plan and create the video presentation along with the demonstration of the software.

### 3.4 Software Development Life Cycle

Software Development Life Cycle is the process used by software developers to design, implement and test a software project. Its aim is to lower the development costs while improving the quality of the project and lowering the production time. It consists of 6 phases that form the

Figure 3.1: Gantt chart timetable



### **3.4. Software Development Life Cycle**

---

Life Cycle. The first phase is the planning and analysis of the requirements. Then the definition of the project's requirements. The following phase is the design of the project and then the development. The final two stages are the project testing and the deployment in the market.

#### **3.4.1 Models**

There are various Software Development Life Cycle models that each one of them has a different approach to the Life Cycle. At the beginning of this project, a model had to be chosen to be followed for the development of the project.

Below, the most common Software Development Life Cycle models are detailed, and it is specified which one is used for this project and why.

##### **3.4.1.1 Waterfall**

It is considered the oldest SDLC model and one of the basics. It has a straightforward approach and the name is self-explanatory. Every phase needs to finish to begin the next one as it acts like its input. The phases are, "Requirement Analysis", "System Design", "Implementation", "Testing", "Deployment" and "Maintenance".

##### **3.4.1.2 Iterative**

This model has the advantage that it provides a basic version of the project at an early stage of development. This is because it urges the development of a basic version at the first "iteration" and then at every iteration, more functionalities are added until the final state is acquired. It lets the developers understand the risks, requirements, and challenges at an early stage of the development.

##### **3.4.1.3 Agile**

This modern SDLC model is a combination of two basic models, the "Iterative" and the "Incremental" models. Its main objective is to deliver a basic-form project fast and to adapt to the different changes and requirements of the project. It creates small incremental parts of the project and completes them through iterations. After every iteration, the result is presented to the customer if there is one so that changes or transformations can be made.

### 3.4.2 Chosen model

The model chosen to be followed for the Software Development Life Cycle of this project is the "Agile" model. This is because of the meetings with the Supervisor of this project. After every building iteration of the project, regular meetings were arranged to report the current state of the software, and discuss any updates or issues. Although not all the parts could be implemented straight from the beginning, whenever a task has begun, it was added into the iteration and updated at every stage. Moreover, the "Agile" approach was adopted because of its realistic approach to software development and the flexibility that it provided during the iterations.

## 3.5 Feasibility Study

In Software Engineering, Feasibility Study is an analysis conducted before the system implementation to check whether the development of the proposed system is feasible and beneficial. In addition, it helps in the identification of the risks that could appear during the software development of the specific project. It consists of various types, which will be detailed in the following subsections.

### 3.5.1 Technical Feasibility

During the planning phase of this project, I decided what technologies I would use to implement the software part. For the tool that would extract the data, I chose to implement it using an LLVM pass. I was not familiar with this technology, but I trusted my learning skills enough to add them as part of my project. Furthermore, I decided to store that data in JSON format. This is because I am familiar with it and it is also a well-documented and easy file format. To get the data and analyse it, I used Python programming language. It is a technology that I have been using and I can code in it, but it also has many functionalities and a large online community that would support me throughout the development of the project.

I made these choices in order to maximize the technical feasibility of the project and lower the risk of not being able to implement the final software. Moreover, the meetings and support of my supervisor also helped minimize the risk of the technical part of the project.

### **3.5.2 Operational Feasibility**

The aim of this project was to be completed and tested about its effectiveness and efficiency. It is not planned to be kept as a usable open-source tool before any future work, thus it has no Operational Feasibility issues.

### **3.5.3 Economic Feasibility**

This project is only developed by me and only free open-source software is used. I am also not using any special equipment apart from my laptop. This makes the project cost-free so it is Economically Feasible.

### **3.5.4 Legal Feasibility**

The finished product of what I am implementing will be a tool available for anyone to use for their convenience to analyse their personal data. Thus, there are no legal implications about data misuse. Also, the project is ethical as it is only implemented to support the use of it and does not harm anyone.

### **3.5.5 Schedule Feasibility**

The early planning and the deadlines set for the different objectives of the project, made it Schedule Feasible. I also did not set more requirements and objectives than what I believed I could complete in time so that I will be done on time for the deadline.

## Chapter 4

# System Design and Implementation

### 4.1 Chosen Approach

The project's objective is to extract the data, analyse them through their natural language identifiers and return some meaningful results. The dataset chosen to be used is Boost C++ Library. It is chosen because it is a popular and a widely used C++ library that is easy to work with. To get the specific data that were necessary for the data analysis, a specific process is followed. Firstly, the LLVM C++ compiler is used to create the bitcode of the source code. Then, an LLVM pass is implemented that goes through the bitcode and extracts the data needed. Afterwards, the LLVM pass outputs the data and prints them with a specific format in a JSON file as Figure 4.2 illustrates. The data include information about each Basic Block of the code, such as where it belongs, its' identifiers and which Basic Blocks it calls and from which it is called by, if any exist. The data are structured this way because of the easy manipulation and clarity of the data when they are structured in a JSON format. The key-value structure makes it efficient for the analysis software to access the data.

Following, Python is chosen as the programming language to implement the analysis software. This is due to its various libraries that would support the analysis but also because of its great features to analyse and visualize data. For the analysis of the data, and the comparison of the Basic Blocks, Levenshtein Distance is chosen, as it works efficiently on sets of data, which fits the project's implementation.

Figure 4.1: Data Extraction Process



## 4.2 Boost Library data

For the implementation of this project, Boost Library data were chosen to assist the design of the system and also to test and evaluate the system. It is an open-source, peer-reviewed C++ library with many useful utility components for C++ developers. The library was selected because it is a C++ library that has a modified build system. It fits the purpose because it allows the implementation of the LLVM pass and therefore the ability to extract C++ code.

## 4.3 Extraction of Basic Blocks

As discussed earlier in the project, the LLVM pass is chosen to extract the BBs from the source code. Its main task is to go through the bitcode produced by the compiler and extract the information needed for us. It provides the following information:

- The current BB name
- The name of the function that it belongs to
- The lines that starts and ends in the source code
- Any functions that the BB is called from, if there are any
- Any functions that the BB calls, if there are any
- A list of the natural language tokens collected from the BB

### 4.3.1 LLVM pass functionality

As figure 4.1 demonstrates, the pass scans the code by splitting it into small parts. Firstly, it detects the different functions of the program. For each function, it gets the Basic Blocks one by one. From the Basic Blocks, it extracts the function that it belongs to and its parent and

children if they exist. Then, it goes through all the Basic Block's code as it goes through each of the instructions and it extracts any tokens that may exist. A token could be a variable name, a loop, an if statement, etc. All those data are being written to a JSON file that each one is linked to a function.

### 4.3.2 JSON file format

In Figure 4.2 we can see a part of the JSON file that contains the information from the function called "is\_backslash". The BB that is starting on line 3 has the name "entry". The "function" key on line 4 specifies the function that the BB belongs to and after that it shows the lines that the BB's start and finish. Following are the important data that we also need for the analysis. If the BB has children, they appear after the key "next\_bb\_NUM\_name". Also, if they have a parent, as seen in the next BB on line 26, it appears with the key "pred\_bb\_NUM\_name". Finally and most importantly come the tokens of the current BB in an array.

Figure 4.2: JSON file structure

```

is_backslash.json
1 [
2 {
3     "current_bb": "entry",
4     "function": "is_backslash",
5     "loc_end_line": 55,
6     "loc_start_line": 0,
7     "next_bb_0_func": "is_backslash",
8     "next_bb_0_name": "return",
9     "next_bb_1_func": "is_backslash",
10    "next_bb_1_name": "if.then",
11    "next_bb_2_func": "is_backslash",
12    "next_bb_2_name": "land.lhs.true",
13    "variables_old": [
14        "p",
15        "return",
16        "land.lhs.true"
17    ],
18 },
19 {
20     "current_bb": "if.then",
21     "function": "is_backslash",
22     "loc_end_line": 57,
23     "loc_start_line": 56,
24     "next_bb_0_func": "is_backslash",
25     "next_bb_0_name": "return",
26     "pred_bb_0_func": "is_backslash",
27     "pred_bb_0_name": "entry",
28     "variables_old": [
29         "len",
30         "return"
31     ],
32 }

```

## 4.4 Data Analysis tool

The main part of the implementation is the tool that reads and analyses the data, and it is implemented using the Python programming language. The python scripts, do all the functionalities in various steps. The steps are going to be detailed below.

### 4.4.1 Processing the data

As the data are already in a directory separated into folders for each function, the script goes through the folders and picks the functions one by one. JSON's python library is the key here as it is responsible for reading the JSON files and gives us the ability to interact with them. Then, a loop through all the data will allow us to gather the parts we need. The data that is acquired from the JSON files is the name of the calling Basic Block, the name of the called Basic Block, and each one's tokens. An example of an entry in this data structure is:

Figure 4.3: An example of a call in its first format

```
[‘regerrorW’, ‘if.then5’, [‘and1’, ‘idxprom’, ‘arrayidx’], [‘add74’, ‘buf_size’]]
```

### 4.4.2 Encoding

To analyse the Natural Language of the data, the system Encodes them and gives each token a specific key that represents it. This is done because it makes the Levenshtein Distance analysis more efficient and simple. All the tokens are added in a dictionary that contains every individual token that is in the data and they are then represented with a serial number. When the data are encoded, they have the following structure:

Figure 4.4: An example of a call in its encoded format

```
[‘regerrorW’, ‘if.then5’, [3976, 1516, 1365], [5656, 5657]]
```

#### 4.4.3 Levenshtein Distance and Token Count

This is the main part of the tool, which uses the data to produce some metrics that are going to help us understand more about the data. Firstly, Levenshtein Distance is calculated between the two sets of the encoded tokens. This is done by the 'textdistance' [ ] library's Levenshtein Distance calculator. It finds the minimum number of edits that need to be done so that the two sets of encoded tokens become the same. Next, it calculates the number of tokens of the calling function as it is used in the plotting of the graphs. After this step, the data look like the following structure:

Figure 4.5: An example of a call with its Levenshtein Distance and Parent Tokens Count

```
[‘regerrorW’, ‘if.then5’, 3, 3]
```

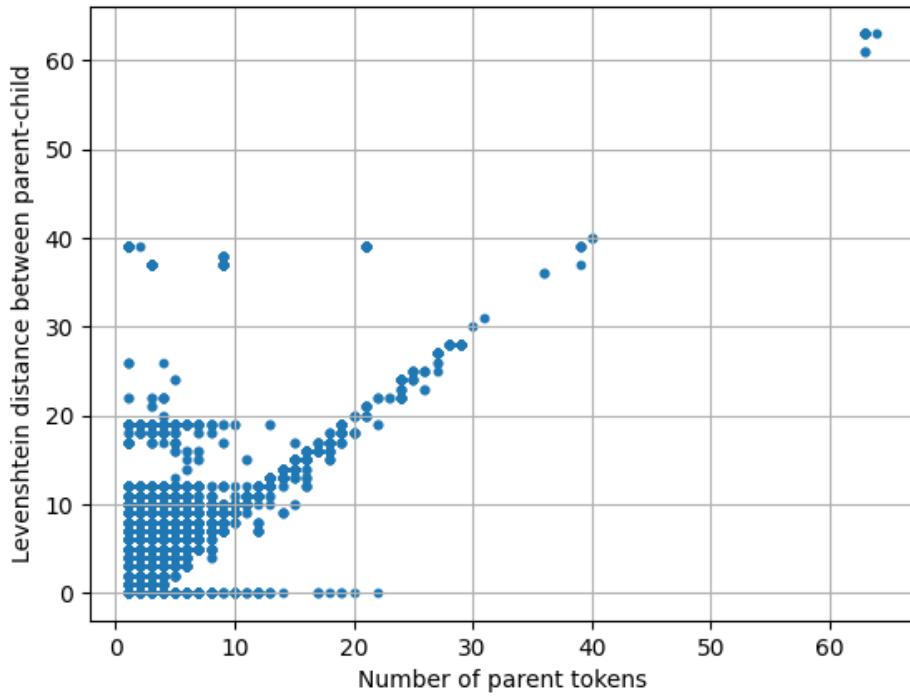
#### 4.4.4 Visualising the results

The process that happens before this step, prepares the data to be illustrated so that it becomes more clear and easy to understand. Every 'call' that is extracted from the system, is added to a scatter plot as a dot that indicates a call.

The graph that is selected to illustrate the data consists of the x-axis which represents the 'number of the parent tokens' and the y-axis which represents the 'Levenshtein Distance between the parent and child'. The number of the parent tokens are simply how many tokens are extracted from the calling BB and the Levenshtein Distance is the distance between the tokens of the calling BB and the called BB.

The visualisation is done, to help us understand the abnormalities in the cases. The Levenshtein Distance between Basic Blocks and the parent tokens count visualisation, will cause the cases that are interesting to us to be detected. Such cases are the ones that the child is not similar to the parent, which later in the evaluation we name as 'Outlier cases'. Figure 4.6 is the result of plotting all the calls from the Boost library data.

Figure 4.6: Scatter plot of all the cases



## 4.5 Evaluation Process

In this section, it is detailed how the system that evaluates the results of the project is implemented. Its purpose is to gather all the data and select a sample of some outliers that occur to analyse them individually.

### 4.5.1 Outliers Selection

For this part of the project, the system is modified to select only the cases that are 'Outliers'. Its purpose is to separate the normal cases from the cases that we believe are 'interesting', so that we can sample them and analyse them further. What defines an 'Outlier' is the high number of a call's Levenshtein Distance or a case that lies away from the  $x=y$  line on the graph.

#### 4.5.1.1 Process

The process followed to select the outliers, is almost identical to the implementation process that is discussed earlier, until a certain point. Its main difference is that instead of presenting all the results in the output and on the scatter plot, it filters the data and prepares them for

Figure 4.7: Outliers JSON file structure

```
10930      {
10931          "callingFunction": "create",
10932          "calledFunction": "invoke.cont",
10933          "distance": 13,
10934          "parentTokensCount": 4,
10935          "function_name": "threadsafe_queue.cpp",
10936          "module_name": "log"
10937      },
10938      {
10939          "callingFunction": "create",
10940          "calledFunction": "lpad",
10941          "distance": 13,
10942          "parentTokensCount": 4,
10943          "function_name": "threadsafe_queue.cpp",
10944          "module_name": "log"
10945      },
```

the sampling.

The filtering of the data, goes through of identification of the outliers which is based on the metrics of the tool. The Levenshtein Distance between the parent and the child Basic Blocks and the Parent Tokens Count, are used for this purpose. For the Levenshtein Distance, the threshold that is set is to be a minimum of 9 between the two sets, and for the number of parents' tokens to be at least 4. This way, the cases with not enough information will be ignored in the outlier selection.

For easier manipulation and sampling of the outliers, they are added to a file in a new JSON format, as shown in Figure 4.7. The figure displays two different examples of entries in the file. Every entry contains the names of the parent and child Basic Blocks, 'callingFunction' and 'calledFunction', the Levenshtein Distance between of them, with the key 'distance', and the count of the parent tokens with the key 'parentTokensCount'. In addition to these values, the function ('function\_name') and the module ('module\_name') that the call belongs to are included, as they are needed for a different part of the analysis.

### 4.5.2 Sampling

After the outliers selection part, the sampling needed to be made in order to pick random cases from the outliers to investigate. For this task, a Python library is used, which randomly picks the number of cases that are set to. To find the number of samples that are needed, the central limit theorem (Anderson 2010) is used. For 1882 outliers cases that exist, to achieve 90% confidence, we need at least 238 samples to investigate.

The system randomly picks this number of cases and it adds them to a CSV file 4.8 which allows them to be saved and previewed in a tabular format. This way, the data are more accessible and easier to understand and read.

Figure 4.8: Sample cases in CSV format

A	B	C	D	E	F	G
	callingFunction	calledFunction	distance	parentTokensCount	function_name	module_name
1	callingFunction	calledFunction	9	5	operations.cpp	filesystem
2	1236 absolute	lpad13	12	13	parser_utils.cpp	log
3	559 translate_escape_sequences	if.end282	12	12	localization_backend.cpp	locale
4	126 impl	for.body_lr.ph	12	19	basic_oarchive.cpp	serialization
5	734 save_pointer	lpad101	19	9	format_parser.cpp	log
6	1041 parse_format<char>	lpad48	9	9	systlog_backend.cpp	log
7	1627 set_local_address	lpad	9	4	message.cpp	locale
8	156 load_file	invoke.cont45	11	5	basic_iarchive.cpp	serialization
9	1685 load_pointer	invoke.cont80	9	6	options_description.cpp	program_options
10	501 find_nothrow	for.body	11	27	dump_ssse3.cpp	log
11	1493 dump_data_char_ssse3	for.body37.i	27	9	format_parser.cpp	log
12	931 parse_format<char>	invoke.cont144	9	13	format_parser.cpp	log
13	842 parse_format<wchar_t>	invoke.cont101	13	11	message.cpp	locale
14	141 runtime_conversion<char>	return	10	4	fileiter.cpp	regex
15	1598 close	delete.notnull	22	12	mo_lambda.cpp	locale
16	362 i4	lpad	11	9	date_time_format_parser.cpp	log
17	1859 parse_format<wchar_t, boost::log::v2_mt>if.else	for.body10	24	24	dump.cpp	log
18	1305 dump_data_generic<char32_t>	invoke.cont86	9	4	format_parser.cpp	log
19	838 parse_format<wchar_t>	land.lhs.true	9	11	fileiter.cpp	regex
20	1592 lock	lpad84	9	4	format_parser.cpp	log
21	908 parse_format<char>	invoke.cont14	20	21	cpu_timer.cpp	timer
22	1416 show_time	cleanup36	13	13	code_conversion.cpp	log
23	1798 code_convert<char32_t, char, std::literals::cout>	invoke.cont2	13	13	format_parser.cpp	log
24	874 parse_format<char>	if.then	11	4	options_description.cpp	program_options
25	461 print	invoke.cont63	39	21	mo_lambda.cpp	locale
26	316 bin_factory	invoke.cont6	9	4	basic_iarchive.cpp	serialization
27	1662 load_pointer	invoke.cont107	11	8	message.cpp	locale
28	176 load_file	land.lhs.true	12	9	message.cpp	locale
29	217 mo_message					

### 4.5.3 Module separation

In order to carry out one part of the testing, the data needed to be separated into large groups. The Modules, which are sets of source code, are selected to define the different groups. A script is implemented to perform this task. It starts by finding the most used modules in the code from a file that contains all the modules and files. It then inputs a specific module and it outputs the number of calls that exist in it, along with a scatter plot with all the calls visualized.

# Chapter 5

## Testing and Evaluation

The testing and evaluation of the project are done using Quantitative and Qualitative analysis. The 'Evaluation Process' that is detailed in the previous chapter, prepares the data for the analyses.

### 5.1 Quantitative Analysis

Table 5 shows the distribution of calls and outlier cases of Boost Library. The total cases are 11949, of which 1882 are outliers. The outliers make the 15.75% of the calls, with the filters that are set. As mentioned in section 4.5.1.1, the filters that determine the outliers from the casts are for the Levenshtein Distance between the tokens of the parent and child BB to be a minimum of 9, and for the number of parent tokens to be at least 4. These filters are set to exclude cases that are not interesting, and also keep for outliers a normal proportion of cases.

The main observation from Table 5 is that the Calls are not proportional to the Lines of Code which point out the size of the source code. This refers to the fact that each module has a different nature and may or may not contain calls. Although, most of the modules that contain many calls seem to have more outlier cases than those with fewer calls.

Figures 5.1, 5.2, 5.3 and 5.4 present the normal and outlier calls on scatters of the Levenshtein Distance against the parent tokens count. Figure 5.1 shows the scatter plot that represents the module 'log', which has the biggest number of calls. There are two types of calls on the plot;

Table 5.1: Boost Library Data. Displays the library modules along with the number of calls and the outlier cases they contain

Module Name	Lines of Code	Calls	Outlier Cases
log	26453	2963	740
filesystem	11815	2327	144
locale	13256	2191	387
program_options	5774	1867	291
regex	10464	848	54
graph	27840	693	64
serialization	14986	521	144
iostreams	10897	135	12
container	16751	98	9
timer	585	65	11
atomic	2725	58	14
wave	7547	52	4
random	5884	38	2
math	92486	36	0
coroutine	3333	24	0
type_erasure	7229	21	4
nowide	2635	9	2
thread	38412	3	0
<b>TOTAL</b>	<b>383040</b>	<b>11949</b>	<b>1882</b>

the normal calls represented by blue dots and the outlier calls which are red dots. The same applies for Figures 5.2 and 5.3, where they display the plots for modules 'filesystem' and 'locale', the second and third modules with the most calls respectively. Lastly, Figure 5.4 illustrates the calls of the rest of the modules, in the same way as the previous Figures.

## 5.2 Qualitative Analysis

In this section, some calls are being selected to be investigated manually. The Sampling tool that is detailed in Section 4.5.2 of the Evaluation Process, is developed for this specific purpose. As mentioned, it randomly selects 238 samples targeting 90% confidence. The samples are then stored in a file, where we can access them and find some interesting characteristics that determine them as outliers. In the following analysis, we will manually investigate some cases that will determine the project's conclusions.

## 5.2. Qualitative Analysis

---

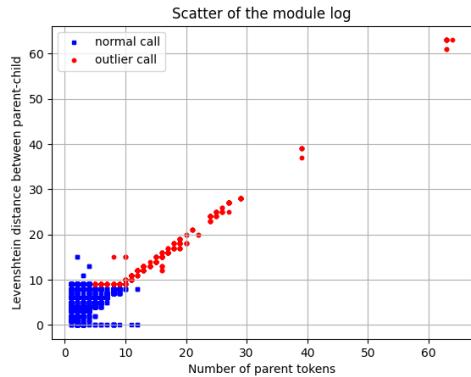


Figure 5.1: 'log' module calls scatter

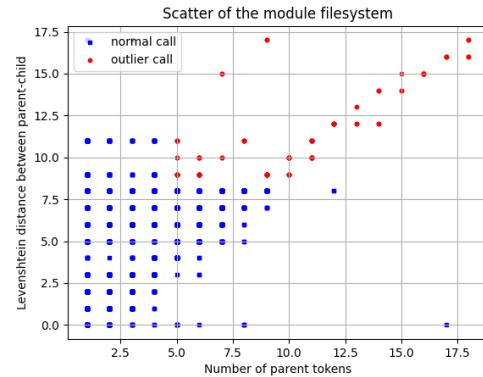


Figure 5.2: 'filesystem' module calls scatter

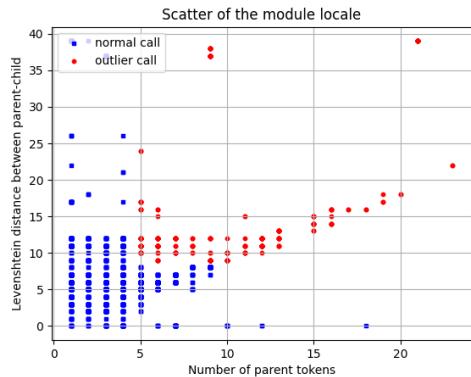


Figure 5.3: 'locale' module calls scatter

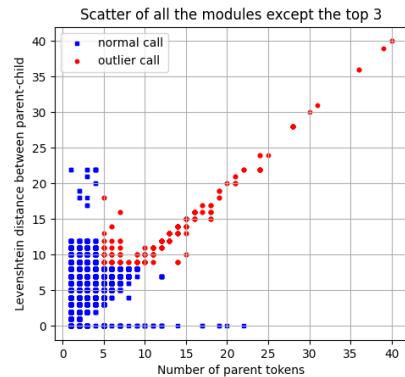


Figure 5.4: Scatter of all the rest modules

### 5.2.1 Case 1

Figure 5.5: First case call

```
{
    "callingBB": "invoke.cont132",
    "calledBB": "lpad135",
    "distance": 9,
    "parentTokensCount": 8,
    "function_name": "basic_iarchive.cpp",
    "module_name": "serialization"
}
```

Figure 5.6: Case 1 lines of code

```

503     // add to list of serialized objects so that we can properly handle
504     // cyclic structures
505     object_id_vector.push_back(aobject(t, cid));
506
507     // remember that the address of these elements could change
508     // when we make another call so don't use the address
509     bpis_ptr->load_object_ptr(
510         ar,
511         t,
512         m_pending.version
513     );
514     object_id_vector[ui].loaded_as_pointer = true;
515 }
516
517 return bpis_ptr;
518 }
```

In Figure 5.6 the source code for the case 5.5 is displayed. The calling Basic Block appears on line 505 and the called Basic Block on lines 509-518. The parent with the child shares some common characteristics. Although, only from the parent, in this case, we are not able to determine whether the child is important to us or no. More data are needed to make such conclusions.

### 5.2.2 Case 2

Figure 5.7: Second case call

```
{
    "callingBB": "sw.bb15",
    "calledBB": "invoke.cont",
    "distance": 9,
    "parentTokensCount": 9,
    "function_name": "std_backend.cpp",
    "module_name": "locale"
}
```

In Figure 5.8 we can see the code for the second case 5.7. The parent Basic Block is the lines 181-182 and the child is on line 183. There is no conditional statement or loop in the code, the flow of the program defines the Basic Blocks. For this case too, the data are insufficient to be

Figure 5.8: Case 2 lines of code

```

181      gnu_gettext::messages_info minf;
182      minf.language = data_.language;
183      minf.country = data_.country;

```

able to make any decisions for the child Basic Block.

### 5.2.3 Case 3

Figure 5.9: Third case call

```
{
    "callingBB": "invoke.cont26",
    "calledBB": "if.then28",
    "distance": 13,
    "parentTokensCount": 13,
    "function_name": "core.cpp",
    "module_name": "log"
}
```

Figure 5.10: Case 3 lines of code

```

734      if (it->get()->try_consume(rec_view))
735      {
736          --end;
737          end->swap(*it);
738          all_locked = false;
739      }

```

Figure 5.10 displays the third case 5.9 of the analysis. We can see the calling Basic Block on line 734 and the called Basic Block on lines 736-738. This call is an if statement that has a condition. If the condition is met, then the parent calls the child, and in that case, the child Basic Block is run. The information provided to us is not enough in this case as well. To make a decision as to whether the child is important to us, we need more information. This applies mainly for the parent's side, but in this case for the child's side too, as an else/else-if statement

could provide some insights for this occasion.

### 5.3 Results Discussion

The quantitative and qualitative analyses performed for this project provided us with enough information to be able to evaluate the results. From the quantitative analysis, we understand that the calls between Basic Blocks are not distributed equally into the project's modules. This has to do with the type of the module and with the fact that more programming languages exist in the project other than C++, to a smaller extent. From the qualitative analysis, we can also see and control the percentage of outliers that our filters produce. This way we were able to control them and keep a reasonable amount of them.

In fact, the most insights about the project came from the qualitative analysis. By investigating a few cases one by one, gave us a good understanding of the final result. There is an issue that appeared throughout this process, which is the limited data that single Basic Blocks provided us with. As this is the main information we need to make the decision as to whether a child's Basic Block is important to us, it needs to be sufficient and reliable. A single line of code cannot give enough information as proven from the analysis. This is not the case for all the calls, but since there are many occurrences, is a flaw of the general project.

# Chapter 6

## Conclusions and Future Work

### 6.1 Conclusions

Throughout this research project, a new set of techniques and technologies were put together to create a system that would potentially help the Static Analysis reduce its workload. To begin with, the LLVM pass provides successfully the data we need for manipulation from any C++ file. Using Python as the main tool for data analysis, was a great choice as it helped us create exactly what the plan was without any important issues. Moreover, Python's libraries made the visualisation and output of results a simple and nice task. Boost Library was also a brilliant selection of data as it provided us with multiple and diverse cases of calls and outliers in different modules.

Moving on to the data that the system was using, the tokens of each Basic Block were a good representation of its content. Comparing two Basic Block's tokens using Levenshtein Distance, turned out to be a successful way to tell how common two Basic Blocks are. This is due to the metrics' ability to compare sets of items rather than one-to-one comparisons. Although, as discussed in the results, the one-to-one comparison of Basic Blocks turned out to be an insufficient way of finding the importance of the child to the parent. As this would have been our criterion on whether the child was important to our code or no, we are unable to correctly make such decisions.

Overall, a really interesting approach has been developed with this system. Many interesting findings came up throughout this project, and new combinations of techniques turned out to

work nicely. This could be a starting point to new research and it could be transformed into a real tool.

## 6.2 Future Work

As the final objective for this project did not go as desired, some future work with the appropriate adjustments could transform this tool into a better version. The idea is that instead of comparing Basic Blocks one-to-one, the comparison of a set of parent Basic Blocks with a child would solve the insufficiency of information needed for the decision. As a larger part of code flow would tell more about the code, it would be easier to tell how important a child is to them.

With some improvement, this project could also become a tool for general use. The final product could be a tool that is applied to your code, analyses it, and using the user's input as a threshold, returns the parts of the code that are less important than the others. The threshold could represent the percentage of data that the user wants to exclude, and it could manipulate the outliers' filters to get the desired amount. Finally, the tool could have the ability to communicate with a Static Analysis tool and coordinate as to which code to exclude from the analysis.

# **Chapter 7**

## **Appendix**

### **7.1 Statement of Ethics**

This project does not access directly sensitive data. Although, it will eventually process other people's programs and data that are sensitive content. Therefore, this project takes ethical, social legal and professional issues into consideration.

### **7.2 Public Interest - Do no harm: Ethical**

The tool created is designed to suggest to the user, which parts of his project are less relevant than others in the project. It is designed to assist programmers when doing Static Analysis of their code. The project is effective, although its results are not to be fully trusted as they are not absolutely reliable. They provide a suggestion that the developer would then assess and do accordingly. In a scenario that the user does exactly what the tool suggests, he could potentially omit an important part of the program to go through Static Analysis. This could cause issues and errors, possibly after the deployment of the project which may lead to harm. Thus, the results of the tool are only to be considered as a suggestion rather than taken for granted.

### **7.3 Informed Consent: Legal**

The project does not involve the active participation of human subjects. Therefore, this section does not apply in the Statement of Ethics.

## 7.4 Confidentiality of Data: Legal

As the project developed is a tool that anybody can use, the users must comply with the Data Protection Act of 2018. If the user is using the tool to process another person's data, he must keep them secure and not accessible from a third party. The user's duty is to keep the data Confidentiality, Integrity, and Availability at all costs.

## 7.5 Social Responsibility: Social

This project, is designed to help the software development community with the analysis of their code. Thus it is fulfilling a civil duty and provides assistance to people that they need it. The correct use of this software can benefit the user by reducing the amount of time that he needs to run his code through Static Analysis.

## 7.6 Professional Competency and Integrity

During the whole process of implementing this project, Professional Competency and Integrity was kept in mind. As BCS indicates, the work that was undertaken was within a logical competence. This project avoids injuring others, and the legislation was clearly understood and complied with, throughout the project's development.

# Bibliography

*About Python* (2021).

URL: <https://www.python.org/>

Anderson, C. J. (2010), *Central Limit Theorem*, American Cancer Society, pp. 1–2.

URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470479216.corpsy0160>

Aouragh, S., Gueddah, H. & Yousfi, A. (2015), ‘Adaptating the levenshtein distance to contextual spelling correction’, *International Journal of Computer Science & Applications* **12**, 127–133.

*Bytecode* (2018).

URL: <https://techterms.com/definition/bytecode>

*Bytecode Definition* (2005).

URL: <https://whatis.techtarget.com/definition/bytecode>

Carbonnelle, P. (2020), ‘Pypl popularity of programming language’.

URL: <https://pypl.github.io/PYPL.html>

Gregor, D. (2010), ‘Clang++ builds boost!’.

URL: <https://blog.llvm.org/2010/05/clang-builds-boost.html>

Hindle, A., Barr, E. T., Su, Z., Gabel, M. & Devanbu, P. (2012), On the naturalness of software, in ‘Proceedings of the 34th International Conference on Software Engineering’, ICSE ’12, IEEE Press, p. 837–847.

*Introducing JSON* (2021).

URL: <https://www.json.org/json-en.html>

## Bibliography

---

- Pradel, M. & Sen, K. (2018), ‘Deepbugs: A learning approach to name-based bug detection’, *Proc. ACM Program. Lang.* **2**(OOPSLA).
- URL: <https://doi.org/10.1145/3276517>
- Satyabrata, J. (2020), ‘Types of feasibility study in software project development’.
- URL: <https://www.geeksforgeeks.org/types-of-feasibility-study-in-software-project-development/>
- SDLC - Overview* (2021).
- URL: [https://www.tutorialspoint.com/sdlc/sdlc\\_overview.htm](https://www.tutorialspoint.com/sdlc/sdlc_overview.htm)
- textdistance* (2021).
- URL: <https://pypi.org/project/textdistance/>
- Writing an LLVM Pass* (2021).
- URL: <https://llvm.org/docs/WritingAnLLVMPass.html#introduction-what-is-a-pass>
- Yujian, L. & Bo, L. (2007), ‘A normalized levenshtein distance metric’, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **29**(6), 1091–1095.