



## THIRD CYCLE OF ACADEMIC STUDIES – DOCTORAL STUDIES

### DOCTORAL DISSERTATION TOPIC

## **RECOMMENDER SYSTEMS FOR SECURE SOFTWARE ENGINEERING**

Candidate:

MSc. Astrit Desku

Mentor:

Prof. Assoc. Dr. Bujar Raufi

Tetovo, July 2022



# Abstract

Recommender Systems are software tools that can assist developers with a wide range of activities, from reusing codes to suggest developers what to do during development processes.

Our intention in this thesis is to model a system that will help software developers to use recommended codes in their projects. In this way, we think that their projects will be more secure while using validated codes.

As the final result of this thesis is a model of software application that is able to produce recommender codes for the software developers, by their asks.

The model consists of three main parts: dataset, recommender engine, and user interface for delivering recommendations. The dataset is built using a control flow graph structure, from data gathering from 557 GitHub repositories. The recommender engine is built by using two options: Cosine Similarity and SQL Semantic Search approach. On the user interface part of the model, the user can choose which of these two options would want to use for producing recommendations.

In this thesis, we performed the evaluation of the recommendations in order to increase the reliability of the user in recommended codes. The evaluation is measured in two ways: statistical evaluation and by user experience.

## Abstrakt

Sistemet e rekomanduesve janë mjete softuerike që mund të ndihmojnë zhvilluesit me një gamë të gjerë aktivitetesth, nga ripërdorimi i kodeve deri tek sugjerimi i zhvilluesve se çfarë të bëjnë gjatë proceseve të zhvillimit.

Synimi ynë në këtë tezë është të modelojmë një sistem që do të ndihmojë zhvilluesit e softuerit të përdorin kodet e rekomanduara në projektet e tyre. Në këtë mënyrë mendojmë se projektet e tyre do të jenë më të sigurta gjatë përdorimit të kodeve të vërtetuara.

Si rezultat përfundimtar i kësaj teze është një model i aplikacionit softuerik që është në gjendje të prodhojë kode rekomanduese për zhvilluesit e softuerit, sipas kërkesave të tyre.

Modeli përbëhet nga tre pjesë kryesore: grupi i të dhënave, motori i rekomanduesit dhe ndërfaqja e përdoruesit për dhënien e rekomandimeve.

Të dhënat janë ndërtuar duke përdorur një strukturë grafi të rrjedhës së kontrollit, nga grumbullimi i të dhënave nga 557 depo të GitHub.

Motori rekomandues është ndërtuar duke përdorur dy opsione: Ngjashmëria e kosinuset dhe SQL kërkimi semantik.

Nga pjesa e ndërfaqes së përdoruesit të modelit, përdoruesi mund të zgjedhë se cilin nga këto dy opsione dëshiron të përdorë për të prodhuar rekomandime.

Në këtë punim, ne kemi bërë vlerësimin e rekomandimeve për të rritur besueshmërinë e përdoruesit në kodet e rekomanduara. Vlerësimi bëhet në dy mënyra: vlerësimi statistikor dhe nga përvoja e përdoruesit.

## Апстракт

Системите за препораки се софтверски алатки кои можат да им помогнат на програмерите со широк опсег на активности, од повторна употреба на кодови до сугерирање на програмерите што да прават за време на развојните процеси.

Нашата намера во оваа теза е да моделираме систем кој ќе им помогне на развивачите на софтвер да ги користат препорачаните кодови во нивните проекти. На овој начин, мислиме дека нивните проекти ќе бидат побезбедни при користење на потврдени кодови.

Како краен резултат на оваа теза е модел на софтверска апликација која е во состојба да произведе препорачани кодови за развивачите на софтвер, по нивно барање.

Моделот се состои од три главни делови: база на податоци, мотор за препораки и кориснички интерфејс за доставување препораки. Базата на податоци е изградена со помош на структура на графикон за контролен тек, од собирање податоци од 557 складишта на GitHub. Моторот за препораки е изграден со користење на две опции: Косинусна сличност и пристап на семантичко пребарување на SQL. На делот од корисничкиот интерфејс на моделот, корисникот може да избере која од овие две опции би сакал да ја користи за давање препораки.

Во оваа теза извршивме евалуација на препораките со цел да ја зголемиме веродостојноста на корисникот во препорачаните кодови. Евалуацијата се мери на два начина: статистичка евалуација и според корисничкото искуство.



I hereby declare that my Dissertation

**Recommender Systems for Secure Software Engineering**

has been written entirely by myself. This work has not previously been submitted for a degree or diploma in any university.

The research was carried out at the SEEU under the supervision of my advisor Prof. Assoc. Dr. Bujar Raufi and the thesis working group including Prof. Assoc. Dr. Artan Luma and Prof. Assoc. Dr. Besnik Selimi

Astrit Desku

---



.....

## Proofreading Declaration

To: Whom it may concern  
Subject: Proofreading confirmation

Dear Sir/Madam,

I, Bardha Qirezi, hereby declare, that I have proofread the Ph.D. Thesis titled **"RECOMMENDER SYSTEMS FOR SECURE SOFTWARE ENGINEERING"**. The thesis meets the criteria for being defended and published.

Sincerely Yours

Bardha Qirezi

04/07/2022



*Court interpreter for English, Spanish and Albanian languages appointed  
with the decision of the District Court of Pristina of October 9, 2010.*

**Tel: +383 (0) 44 14 88 88, email: [bardha.qirezi@gmail.com](mailto:bardha.qirezi@gmail.com)**

**Edutask Academic Services: [info@edu-task.com](mailto:info@edu-task.com)**

[www.edu-task.com](http://www.edu-task.com)  
Sheshi Nëna Terezë, 34/7 K3  
10 000 Prishtinë, Kosovë  
+38345379040



## Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Assoc. Dr. Bujar Raufi, and the thesis working group including Prof. Assoc. Dr. Artan Luma and Prof. Assoc. Dr. Besnik Selimi.

Their continuous support, advice, and encouragement have made it possible to finish this thesis. I am especially grateful for their understanding in discussing challenges in the context of this thesis and for their advice on how to look at problems from different dimensions.

I am deeply indebted to my family Drita, Sidrit, and Dielli for their endless patience and love.

## Publications

The results presented in this thesis were published in the following papers

- Desku, A., Raufi, B., Luma, A., & Selimi, B. (2022). Recommender Model for Secure Software Engineering Using Cosine Similarity Measures. In International Journal of Engineering and Advanced Technology (Vol. 11, Issue 5, pp. 144-148). Blue Eyes Intelligence Engineering and Sciences Engineering and Sciences Publication - BEIESP. doi: 10.35940/ijeat.e3628.0611522
- Desku, A., Raufi, B., Luma, A., & Selimi, B. (2022). Recommender System for Software Engineering using SQL Semantic Search. In International Journal of Engineering and Advanced Technology (Vol. 11, Issue 4, pp. 119–122). Blue Eyes Intelligence Engineering and Sciences Engineering and Sciences Publication - BEIESP. <https://doi.org/10.35940/ijeat.d3494.0411422>
- Astrit Desku, Bujar Raufi, Artan Luma, Besnik Selimi. Cosine Similarity through Control Flow Graphs For Secure Software Engineering. *Proc. of the 7<sup>th</sup> International Conference on Engineering and Emerging Technologies (ICEET) 27-28 October 2021, Istanbul, Turkey*
- Astrit Desku, Bujar Raufi, Besnik Selimi, Artan Luma. CODE VECTORIZATION THROUGH CONTROL FLOW GRAPHS FOR SECURE SOFTWARE ENGINEERING. *4th International Scientific Conference on Business and Information Technologies ISCBIT 2020, September 17-18, 2020 in Skopje/Tetovo, North Macedonia*
- Astrit Desku, Mira Mezini. (2018) Recommender System for Secure Software Engineering. *12<sup>th</sup> South East European Doctoral Student Conference; May 2018 Thessaloniki Greece.*



## List of Figures

Figure 1: Proposed conceptual model and solution workflow [2] .....	7
Figure 2: A potential structure of a recommender system [10] .....	18
Figure 3: RSSE architecture .....	21
Figure 4: RSSE Processes .....	21
Figure 5: Examples of induced and non-induced subgraphs .....	25
Figure 6: Symbols in a CFG [19] .....	26
Figure 7: The CFG process .....	26
Figure 8: A control flow graph extracted from a simple example code snippet [20] .....	27
Figure 9: Data Mining techniques used mostly in Recommender Systems .....	31
Figure 10: <a href="https://blog.knoldus.com/first-interaction-artificial-neural-network/">https://blog.knoldus.com/first-interaction-artificial-neural-network/</a> .....	32
Figure 11: SVM boundary decisions [30] .....	34
Figure 12: A high-level process for text mining [52] .....	39
Figure 13: Building a Matrix of Terms from Unstructured Raw Text [52] .....	41
Figure 14: Algorithm used to collect research papers .....	44
Figure 15: Groups of Datasets for RSSE .....	46
Figure 16: The overview of approaches presented in [22] .....	54
Figure 17: Structure of eRose [88] .....	56
Figure 18: Strathcona user interface: (a) recommendation query; results in (b) a structural overview and (c) highlighted source code; and (d) rationale for recommendation [95] .....	58
Figure 19 : Major tasks of data pre-processing [27] .....	66
Figure 20: Dataset creating stages .....	68
Figure 21: regex pattern to identify method header .....	70
Figure 22: Algorithm for finding methods of class .....	71
Figure 23: Control Flow Graph for <i>Calculation()</i> method .....	73
Figure 24: Given C# example .....	75
Figure 25: Transforming in CFG structure .....	76
Figure 26: The own algorithm for creating CFG structure by code .....	77
Figure 27: List of statements .....	78

Figure 28: Extending If conditional statement.....	79
Figure 29: Extending If –Else conditional statement .....	79
Figure 30: Main part of algorithm for calculating TF-IDF.....	83
Figure 31: Snippet of Datsset1 of TF-IDF calculation .....	84
Figure 32: Snippet of Datsset2 of TF-IDF calculation .....	85
Figure 33: Vectors meaning .....	87
Figure 34: Main part of algorithm for calculating CSI .....	88
Figure 35: Taxonomy of recommendation engines [53].....	93
Figure 36: Cosine similarity index calculation [109] .....	95
Figure 37: Recommendation model using cosine similarity .....	96
Figure 38: Relationship between datasets.....	97
Figure 39: Results after applying cosine similarity .....	98
Figure 40: Results after applying semantic search .....	101
Figure 41: User interface for presenting recommendations .....	105
Figure 42: Performance measure for recommendation using Cosine Similarity .....	112
Figure 43: Performance measure for recommendation using Semantic Search.....	113
Figure 44: Performance measure for subjective evaluation of recommendation using Cosine Similarity .....	114
Figure 45: Performance measure for subjective evaluation of recommendation using Semantic Search.....	115
Figure 46: t-Test calculation of first group.....	117
Figure 47: t-Test calculation of the second group .....	118
Figure 48: t-Test calculation of the third group.....	118

List of Tables

Table 1: Publication grouped by recommendation task from literature review .....45

Table 2: New structure of dataset ..... 70

Table 3: CSI for Dataset1 and Dataset2 .....88

## List of Abbreviations

Abbreviations	Explanation
RSSE	Recommender Systems for Software Engineering
IDE	Integrated Development Environments
API	Application Programming Interfaces
CFG	Control Flow Graph
TF-IDF	Term Frequency–Inverse Document Frequency
CSI	Cosine Similarity Index
ML	Machine Learning
regex	Regular Expression
TDM	Term Document Matrix
AST	Abstract Syntax Trees
H-AST	Heterogeneous - Abstract Syntax Trees
CSCC	Context Sensitive Code Completion
DMMC	Detecting Missing Method Calls
XML	Extensible Markup Language
JSON	Java Script Object Notation
kNN	k - Nearest Neighbour Classifier
ANN	Artificial Neural Network
SVM	Support Vector Machine

## Web Links

- <https://github.com/>
- <https://stackoverflow.com/>
- <http://promise.site.uottawa.ca/SERepository/>
- <https://blog.knoldus.com/first-interaction-artificial-neural-network>
- <https://sourceforge.net/>
- <https://www.reddit.com/r/recommenders/>
- <http://ghtorrent.org/>
- <https://www.openhub.net/>
- <http://lsmr.cs.ucalgary.ca/strathcona>
- <https://scikit-learn.org/>
- <https://pypi.org/project/pyodbc/>
- <https://blog.knoldus.com/first-interaction-artificial-neural-network/>
- <https://www.tiobe.com/tiobe-index>





# CONTENTS

<b>PROLOGUE</b>	<b>1</b>
<b>1. INTRODUCTION</b>	<b>2</b>
1.1 Motivation	3
1.2 Problem	5
1.3 Objectives	6
1.4 Hypothesis	8
1.5 Research Questions	9
1.6 Thesis contribution	10
1.7 Methodology	11
1.8 Thesis outline	13
<b>FUNDAMENTALS AND RELATED WORK</b>	<b>15</b>
<b>2. FUNDAMENTALS</b>	<b>16</b>
2.1 Recommender Systems in Software Engineering	17
2.2 Control Flow Graphs	24
2.2.1 Basic definitions and notations	24
2.2.2 CFG visualization aspects	26
2.3 Data Mining Algorithms	28
2.3.1 Supervised models	28
2.3.2 Unsupervised models	29
2.4 Data Mining in Recommender Systems	30
2.4.1 Classification	31
2.4.2 Cluster Analysis	35
2.4.3 Association Rule Mining	37
2.5 Text Mining	39
2.5.1 Term Frequency - Inverse Document Frequency	40
<b>3. RELATED WORK</b>	<b>42</b>
3.1 Introduction	43
3.2 Mechanism to Collect Data	46
3.2.1 Source Code	47
3.2.2 Source Code under Development	47
3.2.3 Meta Data	48
3.2.4 Interactions	49
3.3 Recommendation Engine to Analyse Data and Generate Recommendations	51
3.4 User Interface to Deliver Recommendations	55
3.4.1 Guiding Software Changes	55
3.4.2 Code Search	57
3.5 Chapter Summary	60

<b>MECHANISM TO COLLECT DATA</b>	<b>61</b>
<b>4. DATASET CREATION</b>	<b>62</b>
4.1 Introduction	63
4.2 Data Pre-processing	65
4.3 Dataset Creation Methodology	68
4.3.1 Repository Selection	68
4.3.2 Transferring repositories to SQL database	69
4.3.3 Extension of Dataset	70
4.4 CFG Structure Dataset	73
4.4.1 Implementing CFG structure	75
4.4.2 How algorithm works	78
4.4.3 Generating Control Flow	79
4.4.4 Dataset with CFG Structure	80
4.5 TF-IDF Calculation	81
4.5.1 TF-IDF calculation algorithm	82
4.6 Cosine Similarity Index Calculation	86
4.6.1 Applications of Cosine Similarity	87
4.7 Chapter Summary	89
<b>APPLICATION AND EVALUATION</b>	<b>90</b>
<b>5. RECOMMENDER ENGINE</b>	<b>91</b>
5.1 Introduction	92
5.1.1 Types of Recommendation Engines	92
5.2 Recommendation using Cosine Similarity	95
5.2.1 Implementation	96
5.2.2 How algorithm works	98
5.3 Recommendation using Semantic Similarity	99
5.3.1 Implementation	100
5.4 Presenting Recommendations	102
5.4.1 Understandability	102
5.4.2 Transparency	103
5.4.3 Accessibility	103
5.4.4 Trust	103
5.4.5 Distraction	103
5.4.6 Implementation	104
5.5 Chapter Summary	106
<b>6. EVALUATION</b>	<b>107</b>
6.1 Introduction	108
6.2 Performance measures	111
6.3 Subjective Evaluation	114

6.4 t-Test	116
6.5 Chapter Summary	119
<b>CONCLUSION AND FUTURE WORK</b>	<b>120</b>
<b>7. CONCLUSION</b>	<b>121</b>
7.1 Discussion	123
7.1.1 Related Work	123
7.1.2 Dataset Creation	124
7.1.3 Application and Evaluation	125
7.2 Future work	127
<b>SUMMARY</b>	<b>129</b>
<b>8. REFERENCES</b>	<b>131</b>



Part I

# Prologue

# 1

# Introduction

## 1.1 Motivation

Software developers have always used tools to perform their work. In the earliest days of the discipline, the tools provided essential compilation and assembly functionality. Then came tools and environments that increasingly provided sophisticated data about the software under development. Nowadays, the systematic and large-scale accumulation of software engineering data opened up new opportunities to create tools or APIs (Application Programming Interfaces) that infer information estimated to be helpful to developers in a given context.

Software development can be challenging because of the large information spaces that developers must navigate. Without assistance, developers can become bogged down and spend a disproportionate amount of their time seeking the information at the expense of other value-producing tasks.

One way to help developers during daily development activities is reusing code from the previous project that they have developed or reuse code from projects developed by other developers. Today, there are online platforms (like Stack Overflow<sup>1</sup>. GitHub<sup>2</sup> etc.) in which developer posts their projects or snippets code from projects; these codes, therefore can be used by other developers in their projects. In most cases, these codes are used in the worst behaviour, copied from source and pasted in their projects. This behaviour can be risky reusing code in this way, so from 1.3 million Android applications analysed in [1], 15.4 % contained security-related code snippets from Stack Overflow. Out of these 97.9 % contain at least one insecure code snippet.

Nowadays, the most modern tools developers use for software development is an integrated feature for code completion. Code completion is a sophisticated technique that helps developers during software development. It is an integral part of modern Integrated

---

<sup>1</sup> Stack Overflow is a question and answer site for professional and enthusiast programmers: <https://stackoverflow.com/>

<sup>2</sup> GitHub is a global company that provides hosting for software development version control: <https://github.com/>



Development Environments (IDEs). Developers often use it to explore Application Programming Interfaces (APIs). It is also useful to reduce the required amount of typing and to help avoid typos. Traditional code completion systems propose all type of correct methods to the developer. Such a list is often very long, with many irrelevant items. In prior work, more intelligent code completion systems have been proposed to reduce the list of proposed methods to relevant items [2]. It is supposed that by using code completion, developers can make more minor mistakes by using syntax algorithms.

In most cases, large-scale software reuse can be achieved using frameworks and libraries, whose functionalities are consumed through APIs. In the development process, using an API can be as simple as calling a function, but it is often much more difficult in practice. The flexibility offered by large APIs translates into sophisticated interface structures that must be accessed by combining interface elements into usage patterns and taking into account constraints and specialised knowledge about the behaviour of the API [3] [4].

Recommender Systems for Software Engineering (RSSEs) are software tools that can assist developers with a wide range of activities, from reusing codes to suggesting what to do during development [5]. These tools can be used also to guide and recommend a developer for next activities, based on codes writes from other developers.

## 1.2 Problem

While security is now available to everyone, and it can protect private information from attackers, we still frequently hear about major data leakages, many of which are due to improper use of security functionalities.

In most software applications, it is necessary to write secure code. Most programmers, however, do not have the expertise required to use algorithms or APIs that enable code security, so they seek help through various forums or platforms. The pieces of code they find in these forums or platforms which is then simply copy and pasted into their application code, without much analysis of that part of the code they are putting in their software application. In this way, from the security point of view, they endanger their entire application.

This can happen because many application developers are not security experts. Even though high-quality security APIs are widely available, programmers often select the wrong algorithms or misuse APIs due to a lack of understanding [6].

Many studies have analysed the usability of security. Clark and Goodspeed [7] and Whitten and Tygar [8] explore misuses of cryptography components from the end-user's point of view. Their focus is on the usability of cryptographic systems which are targeted at end-users, but require far more cryptography knowledge than what can be expected from an ordinary user. Others analyse misuses of cryptography APIs by application developers. In [9] show that over 83% of the vulnerabilities they analysed from the CVE database were due to misuses of cryptography libraries while only 17% were caused by implementation bugs in the cryptography libraries themselves.

## 1.3 Objectives

The most frequent problem during development is that application developers typically lack security expertise. In contrast, security libraries embody highly specialised knowledge that they fail to expose to clients at the appropriate level of abstraction [6].

For this reason, one approach is to combine techniques and concepts from both software engineering and program analysis.

In this context, the main idea of this thesis is to apply existing techniques for mining software repositories specifically to ensure that security is built-in during software development. In this order, the Mining techniques would extract proper usage patterns of cryptographic components. The latter can either suggest developers what to do during development or discover anomalies when the code being developed does not match extracted patterns.

As a final work of this research, it is a solution that helps developers select the relevant security components to use and automatically generate the required code with the correct application calls for them. Also, reduce the adoption barrier for non-security experts by guiding developers to find proper solutions for their security needs.

A particular feature of code maps into corresponding code and a usage protocol describing API restrictions. By composing, the user's code features, the solution would automatically synthesise a secure code and a usage protocol corresponding to the selected usage scenario.

Our proposed solution allows developers through interactive dialog will send code to Analysis Engine, which will analyse the code based on code collection repository. This is also presented in the Figure 1 below, in which is illustrates an overview of our proposed solution.

Developers through interactive dialog will send code to Analysis Engine, which will analyse the code based on code collection repository.

The code collection repository is a dataset with snippet codes downloaded from GitHub platform and structured based on Control Flow Graph (CFG) structure.

After that, the application will be able to generate recommended code as a suggestion for developer. It will depend from developers if they want incorporate suggestions code in their project.

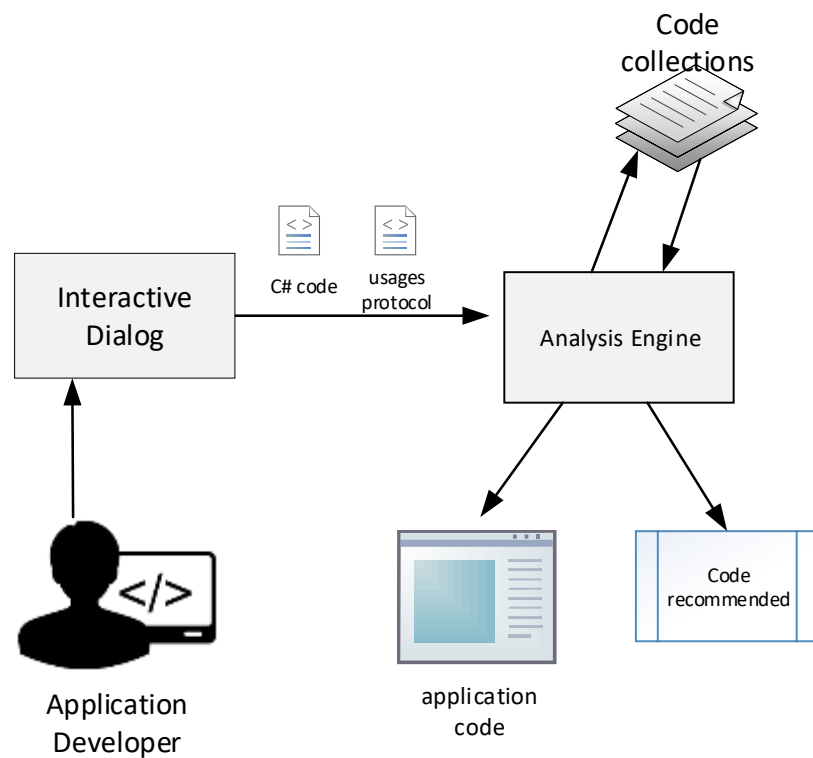


Figure 1: Proposed conceptual model and solution workflow [2]

## 1.4 Hypothesis

In this doctoral thesis, we will concentrate on the following hypothesis:

- I. Creating a recommender system for software development, which will suggest appropriate actions to software developers to automatically ensure that their software is more secure is possible and effective in improving software security.
- II. Automatic generation and validation of code will help in reducing the adoption barrier for non-security experts.
- III. Automatic code recommendation can be achieved based on code patterns extracted from existing codebases.

## 1.5 Research Questions

Research questions that supports hypotheses are presented below

1. Is it possible to explain the discovered code issues in a way that is useful to the developer and assist in resolving them?
2. How can we ensure that the recommendation model learns from the correct code?
3. How can we reduce the adoption barrier for non-security experts by guiding developers to find proper code solutions for their security needs?
4. How can we extract knowledge from code repositories?
5. Which similarity measures are more appropriate for comparing code patterns?
6. How effective are code patterns similarity measures in the context of detection of security anomalies during development?

## 1.6 Thesis contribution

The outcome of this thesis is a solution that helps developers selecting the relevant security components to use and automatically generate the required code with the correct application calls for them.

Thesis also contributes toward reducing the adoption barrier for non-security experts by guiding developers to find proper solutions for their security needs.

Another goal would be to suggest to developers what to do during development and where they need to use security components. Another important issue that is to address with this solution is discovering anomalies during development security issues, by non-security developer's experts.

## 1.7 Methodology

The main purpose of this research is to create a software application or an API, which developers will use as a tool for code suggestions.

In fundamental, our software application solution will compare the code taken from developers with codes that are written by other developers or from the same developer but in earlier projects.

So the code that a software developer gets from software platforms or forums will be able to validate before he puts it in his software application.

This solution enables this validation to be done by codes which have been previously written by other experts in the field and who have placed their solutions on more reliable programming platforms.

Code validation or static analysis of this work is done by using machine learning algorithms, which is determined in the later stages of the work.

In order to arrive at the main goal of this research, the following steps are outlined:

- The first step is collecting data from other projects. For this purpose, we use GitHub an online platform from which data is downloaded.
- The next step is the pre-processing of data from datasets, in order that in our datasets to have the only relevant data that have meaning for our purpose.
- Using Control Flow Graph techniques for code pattern analysis and model generation for code recommendation using machine learning.
- Creating a user interface that communicates with developers, at the same time being able to analyse this code with codes in the dataset, rendering it capable to generate recommendations.

Based on the literature review and potential gaps found, as we emphasize in the first hypothesis, our work will be focused on creating an application as a recommender system for software development.



This application will have three main functionalities:

- Gathering data and creating the dataset
- Using static analysis for comparing input data from users with data gathered in dataset
- Bringing recommendations to users by a user interface

During the static analyses, it is necessary to use adequate Mining techniques to find the proper usage of cryptographic components based on extracted patterns. To choose an adequate Mining algorithm in this phase, we have to analyse them using experimental data. Based on the results found, we would continue researching with an algorithm with better performance.

## 1.8 Thesis outline

Our research plan is divided into several phases accompanied by publications in conferences and journals.

Next, we present the structure of our thesis documentation, which is divided in five parts and seven chapters.

Part I - Prologue, includes 1st chapter: Introduction

Part II - Fundamentals and Related Work, includes 2nd and 3rd chapters: Fundamentals, and Related Work

Part III - Mechanism to collect data, includes 4th chapter: Dataset Creation

Part IV - Application and Evaluation, includes 5th and 6th chapters: Recommendation Engine, and Evaluation

Part V- Conclusion, includes 7th and 8th chapters: Discussion and Future Work, and Summary

The first chapter discusses the importance of recommender systems in software engineering, specifically in secure software engineering. Also, we introduce the problems that have motivated us for this research. This chapter presents the hypotheses and scientific questions of our topic that will finally get answers based on accurate results.

In the second chapter are presented shortly methodologies and techniques that will be used in this work. There will be three main technologies; recommender systems, control flows graphs, and data mining algorithms.

The third chapter contains the latest achievements in the fields of recommender systems in software engineering. Therefore, it will be a state-of-the-art literature review in recommender systems. In this chapter, we also outline datasets used in previous works from different researchers that can potentially be used in this work.

The fourth chapter presents the dataset used for our work.

The fifth chapter includes implementing the software application that is thought with this work.

The sixth chapter contains evaluation of the results generated by the application.

Chapters seven and eight include conclusion and work that is recommended for the future.

Given that this topic is of great importance, we will finally give the recommendations to future researchers to continue work for the future.

## Part II

# Fundamentals and Related Work

# 2

# Fundamentals

This chapter provides information about the main concepts and techniques that we are using in this work; otherwise, in chapter 3, we will present related works in this domain that are also used as motivation for our work.

## 2.1 Recommender Systems in Software Engineering

Many works in the software engineering area propose tools that are built for developers, so understanding the real needs of these target users is a crucial task.

During the review of dozens of papers and other related materials, we found that there are many definitions about the formulation of Recommendation System in Software Engineering (RSSE), but the most valid definition is from [5]. They define RSSE as “a software application that provides information items estimated to be valuable for a software engineering task in a given context.”

RSSEs match this definition in their aim to support developers in decision making. RSSEs help developers find the correct code, API, and human experts in information spaces comprising a system’s code base, libraries, bug reports, version history, and other documentation.

RSSEs also relate their output to a user’s interests. Developers can express their interests explicitly through a direct query or implicitly through actions that the RSSE factors into its recommendations. This distinction highlights a general challenge for recommendation systems, how to establish context, which could include all relevant information about the user, their working environment, and the project or task status at the time of the recommendation.

In general, in the process of recommendation, the user does some actions with a software application, which are processed by various components. Multiple components may process some inputs. There are four components by which the structure of a recommender system consists: analysis of user behaviour, collaborative filtering, analysis of ratings and direct query. Each of these components then creates a user model, which is used for the prediction of the preference of all objects. Some models, like collaborative filtering, use the information about

other users or other additional information. All these models are then combined together to provide most precise recommendation for user [10]

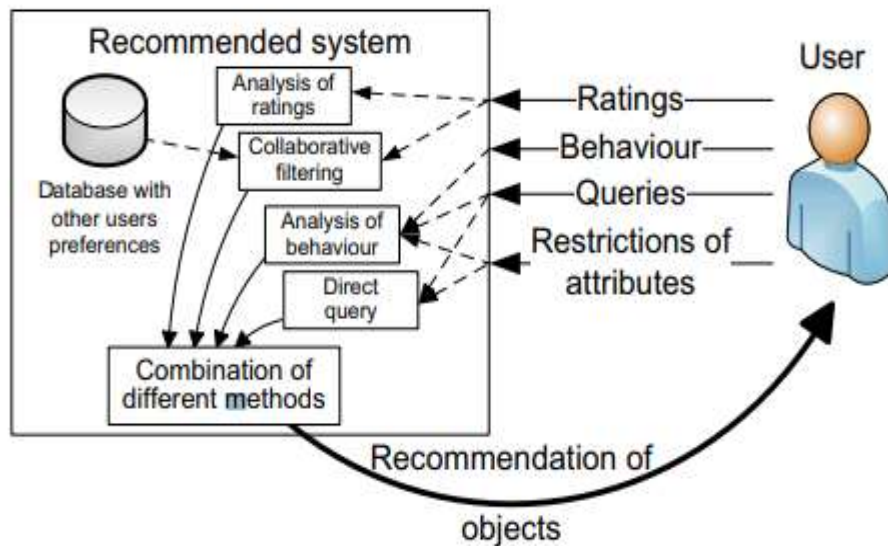


Figure 2: A potential structure of a recommender system [10]

In recommendation systems for traditional domains, the context is typically established through a user profile, which can consist of any combination of user-specified and learned characteristics. However, the context for RSSEs includes the comparatively rich range of activities associated with software development tasks. When a software developer wants help finding where to look next when exploring source code, a recommendation system can establish several parameters, such as what the developer already knows and which parts of the source code are related to their needs.

The RSSEs can be very useful tools for assisting software developers, whereby RSSEs must provide information relevant to their problem and useful to them. The recommendations must be both situation- and user-specific. Sometimes a recommendation is valuable because the developer is aware of the need or all the risks it has, otherwise it is very valuable simply because it confirms the developer's doubts.

Based on [11], [5] an RSSE might need to provide or infer all the following aspects as part of the context:

- the user’s characteristics, such as job description, expertise level, prior work, and social network
- the kind of task being conducted, such as adding new features, debugging or optimising.
- the task’s specific characteristics, such as edited code, viewed code, or code dependencies and
- the user’s past actions or those of the user’s peers, such as artefacts viewed and artefacts explicitly recommended.

Considering a number of papers that we explored for this work and the software application that was as outcome of these papers, it is acceptable to conclude that in the near future, the RSSEs are ready to become part of industrial software developers’ toolboxes. Research prototypes are quickly maturing, tools are being released, and first-generation systems are being re-implemented in different environments [12] [11] [5].

While developers are using different tools for their daily programming activities and they are facing huge programming codes, recommendation systems can support developers in a large number of different information items that can be recommended, some of which are presented in the following [13] [14]:

#### ***Source code within a project***

Recommenders can help developers navigate the source code of their project, for example, by attempting to guess the areas of the project’s source code a developer might need, or want, to look at.

#### ***Reusable source code***

Other recommenders in software engineering attempt to help users discover the API elements (such as classes, functions, or scripts) that can help complete a task.

#### ***Code examples***

In some cases, a developer may know which source code or API elements are required to complete a task but may ignore how to employ them correctly. As a complement to reading



textual documentation, recommendation systems can also provide code examples that illustrate the use of the code elements of interest.

### ***Issue reports***

Much knowledge about a software project can reside in its issue database. When working on a piece of code or attempting to solve a problem, recommendation systems can discover related issue reports.

### ***Tools, commands, and operations***

Large software development environments are getting increasingly complex, and many open-source software development tools and plug-ins are unbounded. Recommendation systems can help developers and other software engineers by recommending tools, commands, and actions that should solve their problems or increase their efficiency.

### ***People***

In some situations, recommendation systems can help find the best person to assign a task to or the expert to contact to answer a question.

So, as RSSEs are software tools explicitly introduced to help software development teams and stakeholders deal with information seeking and decision making, the RSSE architecture consists of three main functionalities [15] [13] [16]:

1. A data collection mechanism to collect development-process data and artefacts in a data model
2. A recommendation engine to analyse the data model and generate recommendations
3. A user interfaces to trigger the recommendation cycle and present its results.

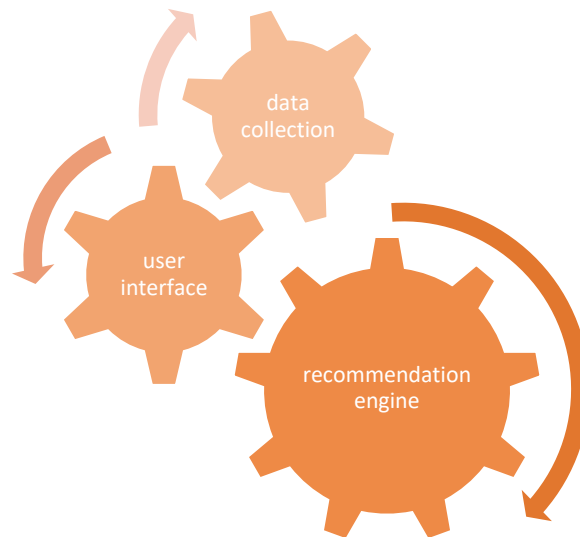


Figure 3: RSSE architecture

Although dozens of RSSEs have been built to provide some of the recommendation functionality mentioned above, there is no unique architecture that we can find during this work. The variety in RSSE architectures is likely a consequence of the fact that most RSSEs work with a dominant data source and are therefore engineered to closely integrate with that data source. In the [13] they propose the main processes for an architecture of RSSE, which are described below.



Figure 4: RSSE Processes

### ***Data pre-processing***

In software engineering, much of the pre-processing effort is required to turn raw character data into a sufficiently interpreted format. For example, source code must be parsed, commits must be aggregated, and software must be abstracted into dependency graphs. This effort is usually needed in addition to more traditional pre-processing tasks such as detecting outliers and replacing missing values.

### ***Capturing context***

While in traditional domains, such as e-commerce, recommendations are heavily dependent on user profiles, in software engineering, recommendation is usually the task-centric orientation. The task context represents all information about the task to which the recommendation system has access to produce recommendations.

In many cases, a task context will consist of a partial view of the solution to the task: for example, some source code that a developer has written, has an element in the code that a user has selected, or has an issue report that a user is reading.

Context can also be specified explicitly, in which case the definition of the context becomes fused with that of a query in a traditional information retrieval system.

In any case, capturing the context of a task to produce recommendations involves somewhat of a paradox: the more precise the information available about the task is, the more accurate the recommendations can be, but the less likely the user can be expected to need recommendations. Similarly, a user in great need of guidance may not be able to provide enough information to the system to obtain usable recommendations. For this reason, recommendation systems must consider that task contexts will generally be incomplete and noisy.

### ***Producing recommendations***

Recommendation algorithms can be executed once pre-processed data and a sufficient amount of task context are available. Here the variety of recommendation strategies is only bounded by the problem space and the creativity of the system designer. However, we note that the traditional recommendation algorithms commonly known as collaborative filtering are seldom used to produce recommendations in software engineering.

### ***Presenting the recommendations***

In its simplest form, presenting a recommendation boils down to listing items of potential interest functions, classes, code examples, issue reports, and so on. Related to the issue of presentation, however, lies the related question of explanation: why was an item recommended? The answer to this question is often a summary of the recommendation strategy: “average rating,” “customers who bought this item also bought,” etc.

In software engineering, the conceptual distance between a recommendation algorithm and the domain familiar to the user is often much larger than in other domains. For example, if a code example is recommended to a user because it matches part of the user's current working code, how can this matching be summarized? The absence of a universal concept such as ratings means that for each new type of recommendation, the question of explanation must be revisited.

## 2.2 Control Flow Graphs

A Control Flow Graph (CFG) - provides a method of representing the decision points and the flow of control within a piece of code, so it is just like a flow chart except that it only shows decisions.

A control flow graph is produced by looking only at the statements affecting the flow of control. In the graph, the statements are represented as nodes and the control flow between the statements is represented as edges. If sequences of unconditional statements appear in the program fragment, then they are illustrated as one single node, because execution of the first statement of the sequence guarantees that all following statements will be executed [17] [18].

### 2.2.1 Basic definitions and notations

- A **graph**  $G = (V, E)$  of size  $N$  is defined by a finite set of vertices  $V = \{1, \dots, N\}$  and a set of edges  $E \subset V \times V$ . Each graph can be represented by a square **adjacency matrix**  $A$  of size  $|V| \times |V|$ , where  $A_{ij}$  is equal to one if there is an edge between vertex  $i$  and vertex  $j$ , and zero otherwise.
- In **weighted graphs**, edges have associated labels (weights). Weights are usually real numbers. Unweighted graphs are described by binary adjacency matrices.
- $G$  is called an **undirected graph** if and only if  $A_{ij}^G = A_{ji}^G$ .
- $G' = (V', E')$  is a **subgraph** of graph  $G$ , if  $V' \subset V$  and  $E' \subseteq V' \times V' \cap E$ .  $G'$  is called an **induced** subgraph of  $G$  if  $E' = V' \times V' \cap E$ .  $G'$ .
- A **matching** or an alignment of two graphs is a mapping between the vertices of two graphs

$$f: V^G \rightarrow V^H \quad (1)$$

- If graphs have the same number of vertices  $N$  and  $f$  is a bijection, then such a matching is called **one-to-one**. A one-to-one matching can be encoded by a **permutation matrix**. The set of permutation matrices is defined as follows

$$P = \{P \in \{0,1\}^{N \times N} : P\mathbf{1}_N = \mathbf{1}_N, P^T \mathbf{1}_N = \mathbf{1}_N\} \quad (2)$$

where  $\mathbf{1}_N$  is a column vector with  $N$  ones.

- Two graph  $G$  and  $H$  are called **isomorphic** if and only if there exists a one-to-one mapping  $f: G \rightarrow H$  such that  $(i,j) \in E^G \leftrightarrow (f(i),f(j)) \in E^H$

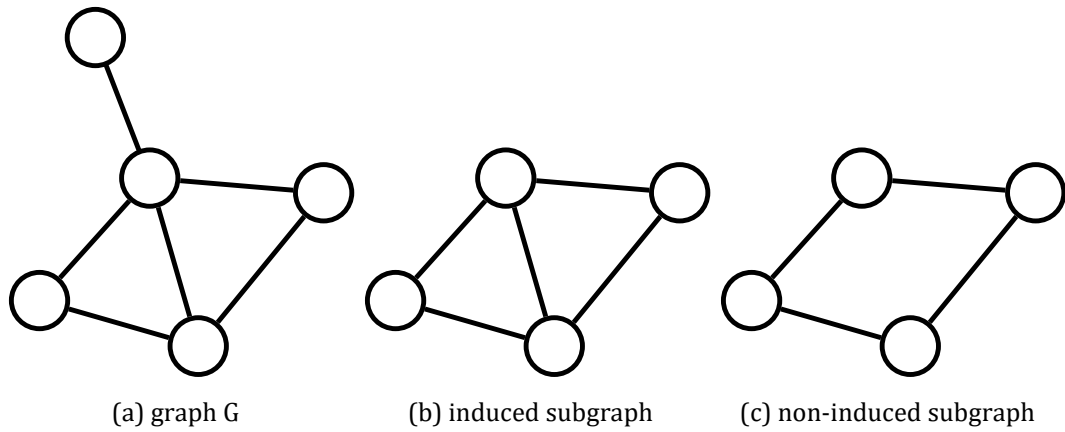


Figure 5: Examples of induced and non-induced subgraphs

### 2.2.2 CFG visualization aspects

In aspect of visualization, a CFG is a graphical representation of a program unit. There are only three symbols that are used to construct a CFG [19] :

- A rectangle represents a sequential
- Decision point
- Merge point

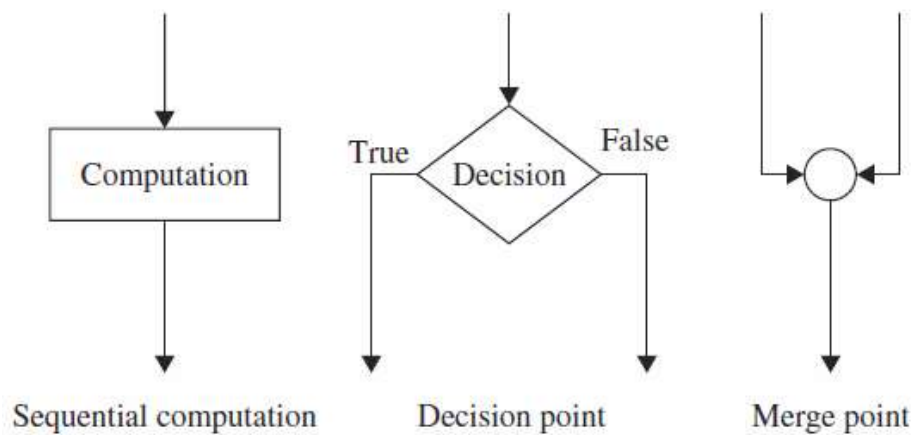


Figure 6: Symbols in a CFG [19]

The CFG is used as a graph for representation of the different paths a program can take during execution. Where each node is a basic block - a unit of code with no branching behaviour and the edges of the graph connect basic blocks that can flow into each other during execution and occur at control flow operations [20].

Transitioning from snippet code to representation as a graph is a process in three phases, as it is presented in the figure below.

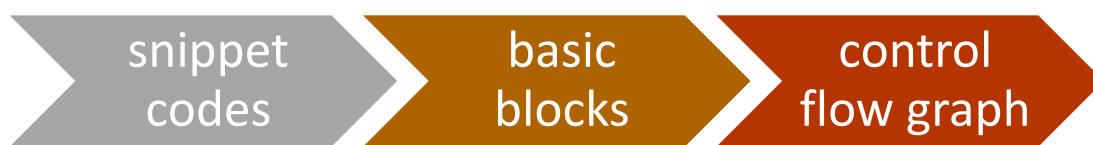


Figure 7: The CFG process

Otherwise, in [21] [22], a control flow graph (CFG) is described as a directed graph,  $G = (V, E)$ , where  $V$  is the set of vertices  $\{v_1, v_2, \dots, v_n\}$  and  $E$  is the set of directed edges  $\{(v_i, v_j), (v_k, v_l), \dots\}$ .

In CFGs, each vertex represents a basic block that is a linear sequence of program instructions having one entry point (the first instruction executed) and one exit point (the last instruction executed) and the directed edges show control flow paths.

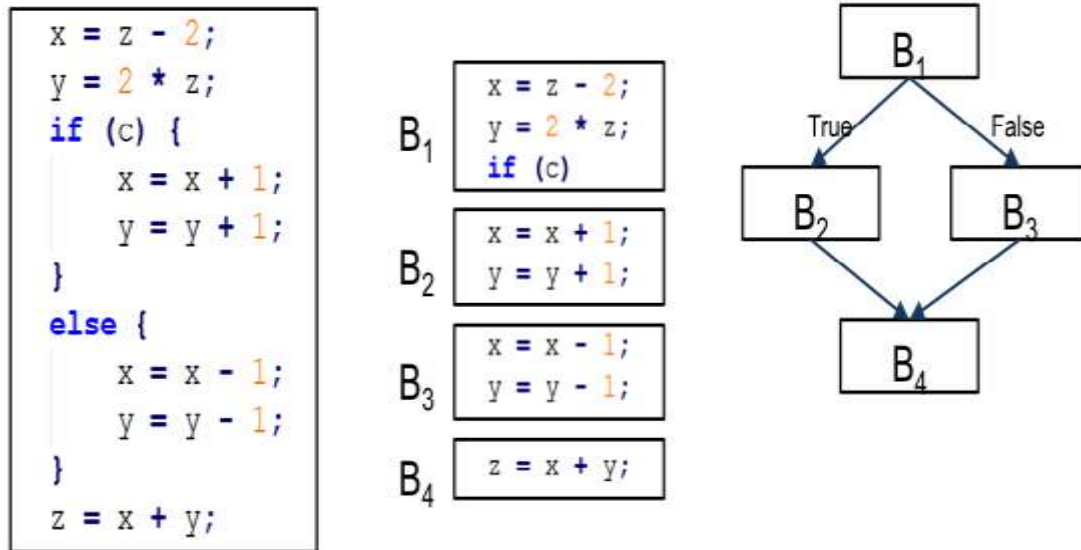


Figure 8: A control flow graph extracted from a simple example code snippet [20]

Since semantic errors are revealed while programs are running, analysing the execution flows of the assembly instructions may be helpful for distinguishing faulty patterns from no-fault ones.

Based on recent research, it has been proved that CFGs are successfully applied to various problems in software testing, including malware analysis [23], [24], and software plagiarism [25], [26].

In the next chapter we present more work from other researchers in the sense of how they have used CFGs in their research; otherwise, in this work, we will attempt to use it as an approach to recommender system.



## 2.3 Data Mining Algorithms

Nowadays, we are conscious that we live in a world where vast amounts of data are collected daily [27].

Increment of data from time to time has made it necessary to apply Data Mining as a field to produce results as accurate and timely as possible.

Data Mining as a technique is very useful in varieties of applications, as a process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems. [28] [27].

The term Data Mining refers to a broad spectrum of mathematical modelling techniques and software tools used to find patterns in data and use these to build models.

In [27], Data Mining is defined as an interdisciplinary subfield of computer science and statistics with an overall goal to extract information, with intelligent methods, from a data set and transform the information into a comprehensible structure for further use.

Data Mining uses models to find knowledge from data; these models consist of a set of rules, equations, or complex functions that can be used to identify valuable data patterns, understand, and predict behaviours. In general, data mining models can be grouped in different ways; in [29] they are grouped into two main classes according to their goal, supervised models and unsupervised models.

### 2.3.1 Supervised models

In supervised models, the goal is to predict an event or estimate the values of continuous attributes. In these models, there are input fields or attributes and an output or target field. Input fields are also called predictors because they are used by the model to identify a prediction function for the output field. These models are further categorised into classification and estimation models: [28] [27]

- **Classification:** In these models, the target groups or classes are known from the start. The goal is to classify the cases into these predefined groups; in other words, to predict

an event. The generated model can be used as a scoring engine for assigning new cases to the predefined classes. It also estimates a propensity score for each case. The propensity score denotes the likelihood of occurrence of the target group or event.

- **Estimation:** These models are similar to classification models but with one major difference. They are used to predict the value of a continuous field based on the observed values of the input attributes.

### 2.3.2 Unsupervised models

In unsupervised models, there is no output field, just inputs. The pattern recognition is undirected, that means it is not guided by a specific target attribute. The goal of such models is to uncover data patterns in the set of input fields. Unsupervised models include: [28] [27]

- **Cluster models:** In these models, the groups are not known in advance. Instead, we want the algorithms to analyse the input data patterns and identify the natural groupings of records or cases. When new cases are scored by the generated cluster model, they are assigned to one of the revealed clusters.
- **Association models:** These models also belong to the class of unsupervised modelling. They do not involve direct prediction of a single field. All the fields involved have a double role since they act as inputs and outputs simultaneously. Association models detect associations between discrete events, products, or attributes. Sequence models detect associations over time.

Each of the models described shortly above consists of a number of algorithms that are used in different applications. In the next section, we will present some of the algorithms that are more relevant for recommender systems purposes.

## 2.4 Data Mining in Recommender Systems

In the context of recommender systems, the term data mining is used to describe the collection of analysis techniques used to infer recommendation rules or build recommendation models from large data sets. Recommender systems that incorporate data mining techniques make their recommendations using knowledge learned from the actions and attributes of users [28].

Many different algorithmic approaches have been applied to the basic problem of making accurate and efficient recommender systems. The earliest “recommender systems” were content filtering systems designed to fight information overload in textual domains. These were often based on traditional information-filtering and information retrieval systems. Recommender systems that incorporate information retrieval methods are frequently used to satisfy ephemeral needs (short-lived, often one-time needs) from relatively static databases. For example, requesting a recommendation for a book preparing a sibling for a new child in the family [28]. Conversely, recommender systems that incorporate information filtering methods are frequently used to satisfy persistent information (long-lived, often frequent, and specific) needs from relatively stable databases in domains with a rapid turnover or frequent additions.

In the next section, we will present data mining techniques that are used in many recommender applications. Based on the literature review [30], [31], [32], in the following figure are shown the classification of data mining methods for recommender systems.

### 2.4.1 Classification

A classifier is a mapping between a feature space and a label space, where the features represent characteristics of the elements to classify and the labels represent the classes. There are many types of classifiers, but in general, as mentioned in the sections above, there are two types of classifications: supervised and unsupervised classification.

In supervised classification, a set of labels or categories is known in advance and we have a set of labelled examples, which constitute a training set. In unsupervised classification, the labels or categories are unknown in advance and the task accordingly to some criteria, organize the elements at hand without the use of labelled training set [30].

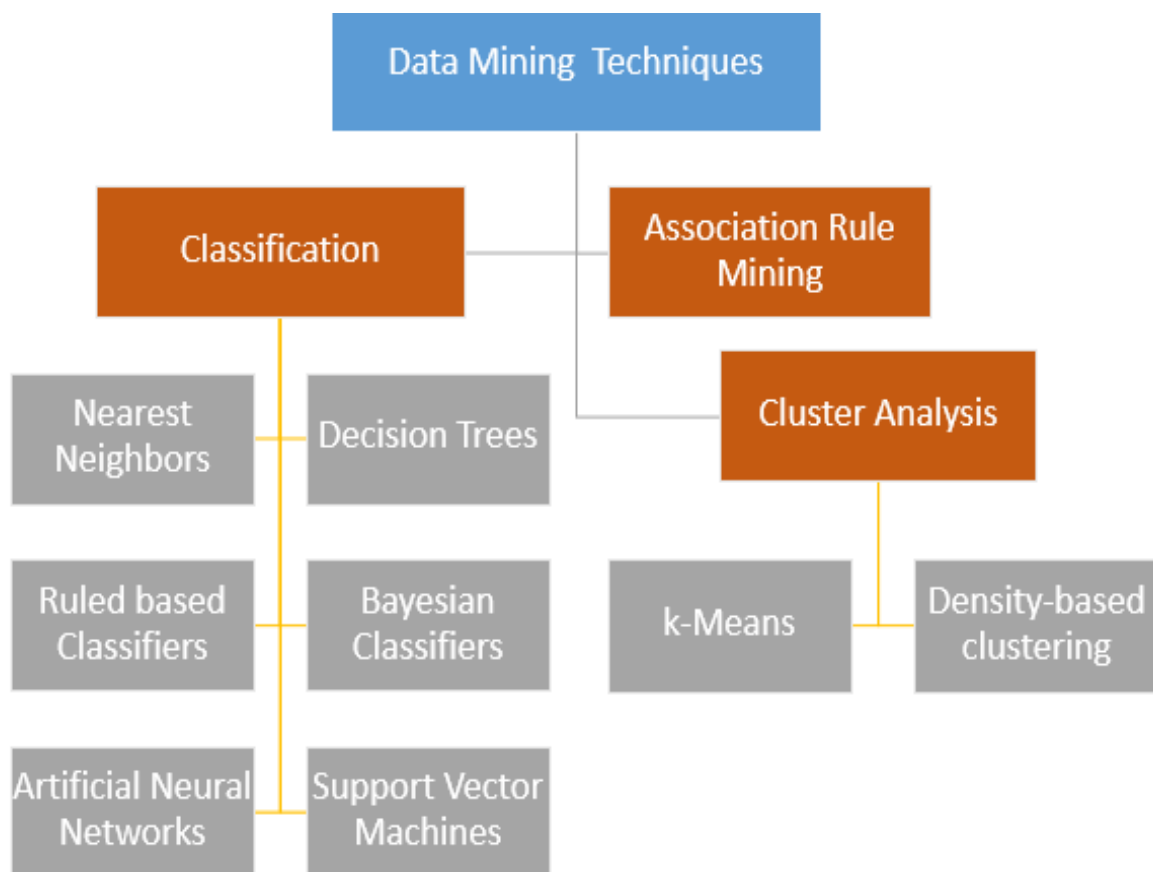


Figure 9: Data Mining techniques used mostly in Recommender Systems

## Nearest Neighbours

Instance-based classifiers work by storing training records and using them to predict the class label of unseen cases. A trivial example is the so-called rote-learner.

This classifier memorizes the entire training set and classifies only if the attributes of the new record match one of the training examples exactly. A more elaborate, and far more popular, instance-based classifier is the k-nearest neighbour classifier (kNN) [33].

## Decision Trees

Decision trees [34] [35] are classifiers on a target attribute in the form of a tree structure. The observations to classify are composed of attributes and their target value. The nodes of the tree can be: a) decision nodes, in these nodes a single attribute-value is tested to determine to which branch of the subtree applies. Or b) leaf nodes which indicate the value of the target attribute. There are many algorithms for decision tree induction: Hunts Algorithm, CART, ID3, C4.5, SLIQ, SPRINT to mention the most common [30].

## Artificial Neural Networks

An Artificial Neural Network (ANN) [36] is an assembly of inter-connected nodes and weighted links that is inspired in the architecture of the biological brain. Nodes in an ANN are called neurons as an analogy with biological neurons. These simple functional units are composed into networks that have the ability.

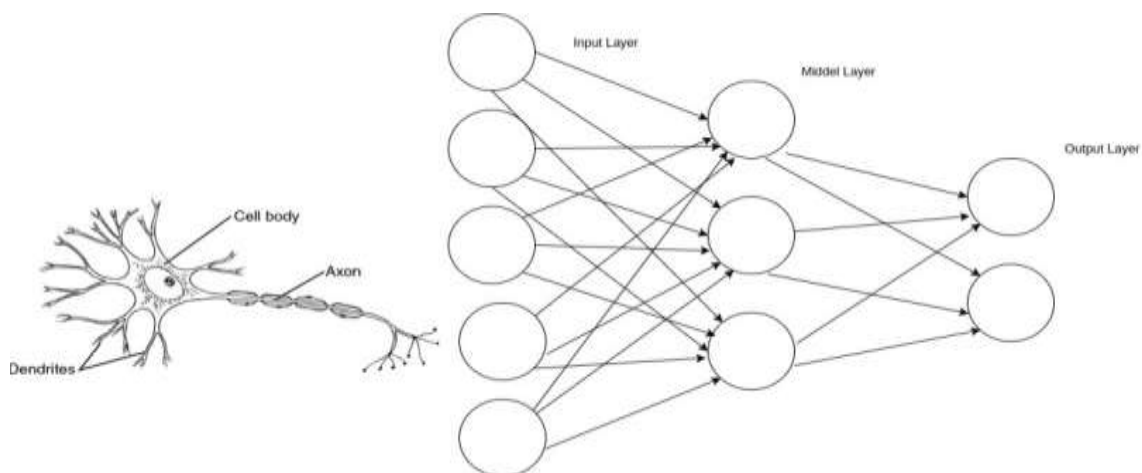


Figure 10: <https://blog.knoldus.com/first-interaction-artificial-neural-network/>

In [37] the Neural Network has been recognized as a promising technique for anomaly detection because the intrusion detector should ideally recognize not only previously observed attacks but also future unseen attacks. While in [38] are presented two advantages of these techniques for anomaly detection in recommender systems:

- Ability to generalize from limited, noisy and incomplete data.
- Has the potential to recognize future unseen patterns.

### **Bayesian Classifiers**

A Bayesian classifier [39] is a probabilistic framework for solving classification problems. It is based on the definition of conditional probability and the Bayes theorem.

Studies comparing classification algorithms have found a simple Bayesian classifier known as the naïve Bayesian classifier to be comparable in performance with decision tree and selected neural network classifiers. Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases. Naive Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called class conditional independence. It is made to simplify the computations involved and, in this sense, is considered “naive.” [27] [30].

In [40] is mention that this technique is generally used for intrusion detection in combination with statistical schemes, a procedure that yields several advantages, including the capability of encoding interdependencies between variables and of predicting events, as well as the ability to incorporate both prior knowledge and data.

While in [38] are present two advantages of this technique for anomaly detection in recommender systems:

- Encodes probabilistic relationships among the variables of interest.
- Ability to incorporate both prior knowledge and data

## Support Vector Machines

The goal of a Support Vector Machine - SVM classifier [41] is to find a linear hyperplane or decision boundary that separates the data in such a way that the margin is maximized. For instance, if we look at a two class separation problem in two dimensions like the one illustrated in figure below, we can easily observe that there are many possible boundary lines to separate the two classes. Each boundary has an associated margin. The rationale behind SVM's is that if we choose the one that maximizes the margin we are less likely to misclassify unknown items in the future.

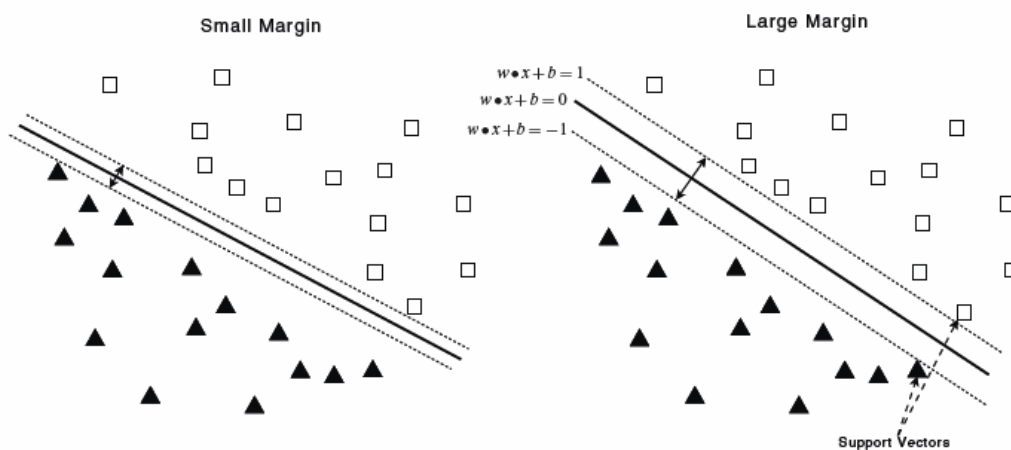


Figure 11: SVM boundary decisions [30]

### 2.4.2 Cluster Analysis

Clustering [42], also referred to as unsupervised learning, consists of assigning items to groups so that the items in the same groups are more similar than items in different groups. The goal is to discover natural or meaningful groups that exist in the data. There are two main categories of clustering algorithms: hierarchical and partitional.

Partitional clustering algorithms divide data items into non-overlapping clusters such that each data item is in exactly one cluster. Hierarchical clustering algorithms successively cluster items within found clusters, producing a set of nested cluster organized as a hierarchical tree.

#### k-Means

k-Means [30] clustering is a partitioning method. The function partitions the data set of  $N$  items into  $k$  disjoint subsets  $S_j$  that contain  $N_j$  items so that they are as close to each other as possible according a given distance measure. Each cluster in the partition is defined by its  $N_j$  members and by its centroid  $\lambda_j$ . The centroid for each cluster is the point to which the sum of distances from all items in that cluster is minimized.

The basic k-means is an extremely simple and efficient algorithm.

The algorithm works by randomly selecting  $k$  centroids. Then all items are assigned to the cluster whose centroid is the closest to them. The new cluster centroid needs to be updated to account for the items who have been added or removed from the cluster and the membership of the items to the cluster updated. This operation continues until there are no further items that change their cluster membership.

Most of the convergence to the final partition takes place during the first iterations of the algorithm, and therefore, the stopping condition is often changed to “until relatively few points change clusters” in order to improve efficiency [43].



## **Density-based clustering**

Density-based clustering [44] algorithms such as DBSCAN work by building up on the definition of density as the number of points within a specified radius.

DBSCAN, for instance, defines three kinds of points: core points are those that have more than a specified number of neighbours within a given distance; border points have fewer than the specified number but belong to a core point neighbourhood; and noise points are those that are neither core or border.

The algorithm iteratively removes noise points and performs clustering on the remaining points.

### 2.4.3 Association Rule Mining

Association Rule Mining focuses on finding rules that will predict the occurrence of an item based on the occurrences of other items in a transaction.

The fact that two items are found to be related through means of co-occurrence but not causality.

In general, association rule mining can be viewed as a two-step process [27]:

1. Find all frequent item sets: by definition, each of these item sets will occur at least as frequently as a predetermined minimum support count, min sup.
2. Generate strong association rules from the frequent item sets: by definition,

In [45] association rules is mention as one of the best-known examples of data mining in recommender systems. Item-to-item correlation recommender applications usually use current interest rather than long-term history.

Association rules have been used for many years in merchandising [28] , both to analyse patterns of preference across products, and to recommend products to consumers based on other products they have selected. An association rule expresses the relationship that one product is often purchased along with other products [46].

In addition to use in commerce, association rules have become powerful tools in recommendation applications in the domain of knowledge management. Such systems attempt to predict which web page or document can be most useful to a user. As [47] writes “The problem of finding Web pages visited together is similar to finding associations among item sets in transaction databases. Once transactions have been identified, each of them could represent a basket, and each web resource an item.” Systems built on this approach have been demonstrated to produce both high accuracy and precision in the coverage of documents recommended [48].

There are many recommender systems that base in association rule, in the following we will present some of them based on our research. So, Michail presented in [49], the use of

association rule mining to document typical API usage. In their study presented [50] Li and Zhou also use association rule mining for PR-Miner. They learn the rules on item sets of program elements to automatically extract general programming rules [2].

Another study includes association rule, is conducted by [51] where is presented the FrUIT who is a Framework Understanding Tool integrated into Eclipse that applies association rule mining on the class level.

As is described in their paper, they use three kinds of properties to learn the rules for an example class: all method calls existing in that class, the list of extended classes, and the list of overridden methods. The approach is evaluated using three case studies with rules learned for the SWT framework from code shipped together with the Eclipse IDE.

## 2.5 Text Mining

Text mining is related to patterns and discovering new knowledge by applying techniques not on ordered data, but on unstructured natural language.

This constitutes the vast area of text and web mining. A cleaned and organized table consisting of rows and columns of data was fed as input to an algorithm. The output from the algorithm was a model that could then be used to predict outcomes from a new data set or to find patterns in data. But are the same techniques applicable to extract patterns and predict outcomes when the input data looks like normal written communication [52].

Text mining, more than any other technique within data mining, fits the mining metaphor. Traditionally, mining refers to the process of separating dirt from valuable metal and in the case of text mining, it is an attempt to separate valuable keywords from a mass of other words and use them to identify meaningful patterns or make predictions [52].

The fundamental step in text mining involves converting text into semi structured data. Once the unstructured text is converted into semi-structured data, there is nothing to stop one from applying any of the analytics techniques to classify, cluster, and predict. The unstructured text needs to be converted into a semi-structured data set so that patterns can be found and even better, models could be trained to detect patterns in new and unseen text [53] [52].

The chart in Figure 12 identifies the main steps in this process at a high level. Each of the main processes will now be examined in detail and some necessary terminology and concepts will be introduced. But before these processes are described, a few core ideas essential to text analytics will need to be defined.

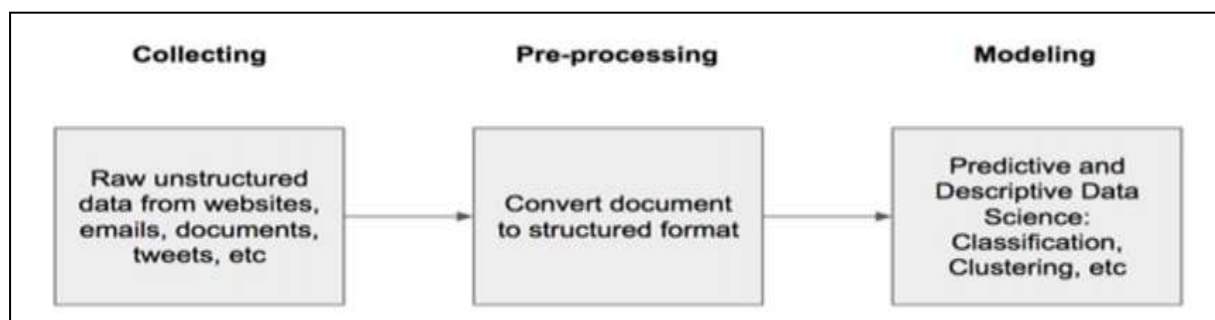


Figure 12: A high-level process for text mining [52]

### 2.5.1 Term Frequency - Inverse Document Frequency

Term Frequency–Inverse Document Frequency (TF-IDF), is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. [54]. It is often used as a weighting factor in searches of information retrieval, text mining, and user modelling.

The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

The TF-IDF in the collection with N documents is computed in following steps.

First is computed the frequency of the word in the document representation by the following formula

$$TF_{ji} = \frac{f_{ij}}{\max_k f_{kj}} \quad (3)$$

- Where  $f_{ij}$  is the frequency of term (word)  $i$  in document  $j$

The term frequency of term  $i$  in document  $j$  is  $f_{ij}$  normalized by dividing it by the maximum number of occurrences of any term in the same document. Thus, the most frequent term in document  $j$  gets a TF of 1, and other terms get fractions as their term frequency for this document.

The next step is to measure IDF, which is defined by the following formula.

$$IDF_i = \log_2(N/n_i) \quad (4)$$

- Where term  $i$  appears in  $n_i$  of the N documents in the collection

Finally, the TF-IDF score for term  $i$  in document  $j$  is then defined to be  $TF_{ji} \times IDF_i$ . The terms with the highest TF-IDF score are often the terms that best characterize the topic of the document.

## Terminology

Consider the following two sentences: “This is a book on data mining” and “This book describes data mining and text mining using RapidMiner.”

Suppose the objective is to perform a comparison between them, or a similarity mapping. For this purpose, each sentence is one unit of text that needs to be analysed. These two sentences could be embedded in an email message, in two separate web pages, in two different text files, or else they could be two sentences in the same text file.

In the text mining context, each sentence is considered a distinct document. Furthermore, in the simplest case, words are separated by a special character: a *blank space*. Each word is called a token, and the process of discretizing words within a document is called tokenization.

For the purpose here, each sentence can be considered a separate document, although what is considered an individual document may depend upon the context.

- Document 1- This is a book on data mining
- Document 2 -This book describes data mining and text mining using RapidMiner

Some form of structure can be imposed on this raw data by creating a matrix where the columns consist of all the tokens found in the two documents and the cells of the matrix are the counts of the number of times a token appears, as shown in figure below

	This	is	a	book	on	data	mining	describes	text	rapidminer	and	using
Document 1	1	1	1	1	1	1	1	0	0	0	0	0
Document 2	1	0	0	1	0	1	2	1	1	1	1	1

Figure 13: Building a Matrix of Terms from Unstructured Raw Text [52]

The table in figure above is called a document vector or term document matrix (TDM) and is the cornerstone of the pre-processing required for text mining

3

# Related Work

The work presented in this work touches many related research areas. In this chapter, we will present a wide selection of previous works to establish the necessary background and to build the foundations for the work presented later.

## 3.1 Introduction

Nowadays, many studies conducted in this area have resulted with Application Programming Interfaces (API) or tools that can be integrated on development platforms.

In modern software development, APIs are a means to encapsulate responsibilities and facilitate code reuse, and they are widely used in statically typed programming languages. Typical RSSE approaches apply static analyses to extract information about the usage of these APIs from the source code of many projects. Some RSSE techniques learn general patterns from source code by treating it as text or on the syntax level [55].

To identify related work, we consulted a recent comprehensive survey such as [56] [3] [57] [15] [58] and the RSSE book [13]. We added several popular RSSE and recent publications at premier software engineering conferences. In the following sections, we will present surveyed recommenders grouped by their recommendation task and give a brief introduction to each paper or technique.

There are consulted more than 130 papers; we found that from them around 100 are relevant for our research in broader sense, and only 42 are in the scope of our research.

In the Figure 14, below it was presented the algorithm how we collect research papers. We have used four sources to search for papers: ACM Digital Library, IEEEExplore, Springer Verlag and Google Scholar.

There are used four phases to classifying papers. In the beginning phase, we collected all papers by searching with keywords: Recommender Systems, Machine Learning, Software Engineering, Data Mining Techniques, and Control Flow Graph.

In the first phase, articles are selected based on their titles, in the second phase are eliminated duplicate papers, and in the final phase, papers are classified by their abstracts.



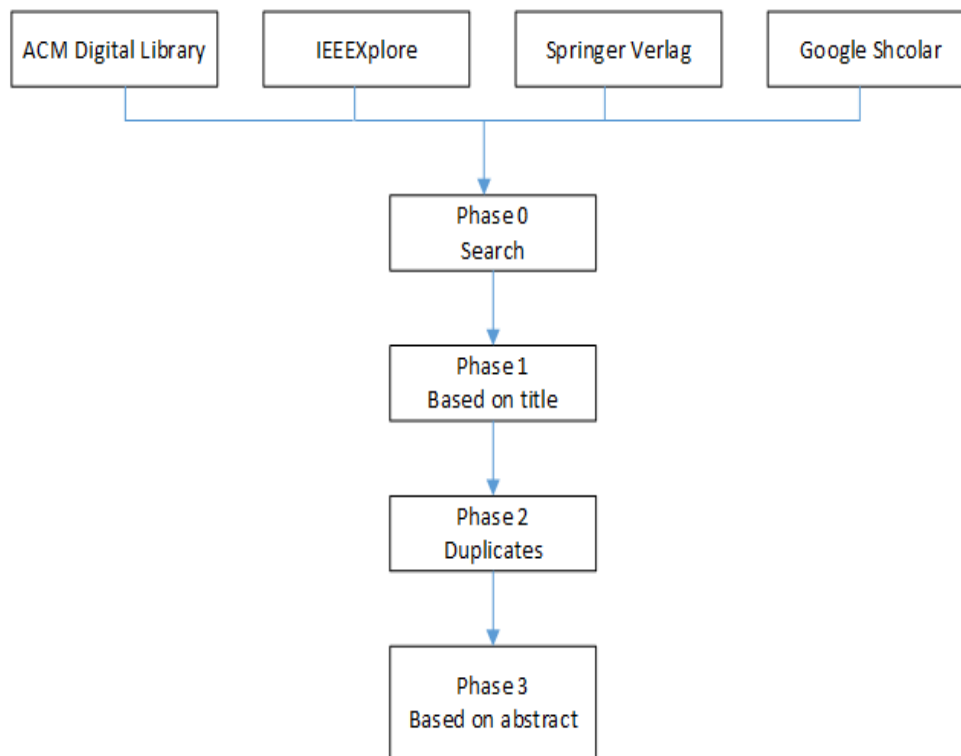


Figure 14: Algorithm used to collect research papers

Based on the literature review, we have grouped the literature in three main groups

- Mechanism to collect data,
- Recommendation engine to analyse data and generate recommendations,
- A user interface to deliver recommendations.

In the table below are presented publications grouped by recommendation task and their sub categories.

Otherwise, in the following sections, we will describe in detail each of them.

Recommendation Task	Categories	Publication
Mechanism to Collect Data	Source Code	[60], [61], [62], [63]
	Source Code Under Development	[64], [65], [66], [67]
	Meta Data	[68], [69]
	Interactions	[2], [70], [71], [72], [73]
Recommendation Engine to Analyse Data and Generate Recommendations		[74], [2], [76], [77], [78], [79], [80], [81], [82], [83], [6], [84], [85], [86], [87], [20], [22]
User Interface to Deliver Recommendations	Guiding Software Changes	[88], [89], [90], [91], [92]
	Code Search	[93], [94], [95], [96], [97]

Table 1: Publication grouped by recommendation task from literature review

## 3.2 Mechanism to Collect Data

While for researchers in different field of studies, data is very important part of their research, for researchers in recommender systems finding reliable data or dataset is one of the biggest challenges.

Nowadays, versioned source code is available in many public repositories of open-source projects; it is much harder to get access to more detailed change information or to activities that describe them in IDE (Integrated Developer Environment) development process.

Once the essential properties of a target domain have been identified, extensive datasets can be generated through simulation [59]. Unfortunately, the IDE development process is very complex and not yet fully understood to be able to simulate it. As a result, it is required to create appropriate datasets.

The mechanism for collecting data is one of the important aspects of recommender systems. In the following paragraphs, we will present publications that deal with collecting data.

Based on literature review for this work, datasets for recommender systems can be grouped in four main categories. In the following of this section, we will present these datasets and a short description of each of them specifically.

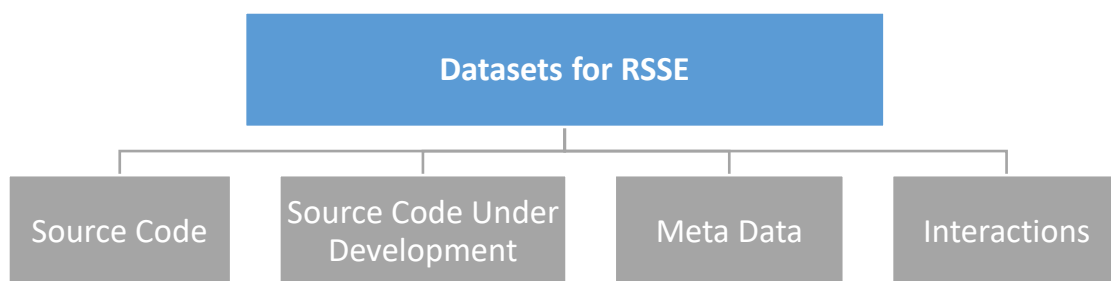


Figure 15: Groups of Datasets for RSSE

### 3.2.1 Source Code

Several approaches make source code available for research in large and stable datasets. The PROMISE<sup>3</sup> repository presented in [60] was an early advance to provide a platform to share datasets and tools used in mining studies.

The QUALITASCORPUS [61] is a curated collection of open-source JAVA source code for empirical research. The corpus contains dependencies for most contained projects and describes how the missing dependencies can be installed.

In [62] authors extract a dataset for usages of API methods and annotations from a large number of GitHub repositories. From all libraries that have been referenced in the repositories, the authors select the five most popular open-source projects that are reasonably large and actively developed.

The BOA project, presented firstly in [63] provides an infrastructure for writing analyses with an ultra-large-scale repository. The project provides curated data sets that contain source code from many projects hosted on GitHub (about 7.8M projects) and SourceForge<sup>4</sup> (about 700K projects). The source code is stored in a custom Abstract Syntax Trees (AST) format that supports most JAVA constructs.

### 3.2.2 Source Code under Development

Changes extracted from the commit history of repositories are coarse-grained and not representative of actual source code evolution [64].

Other means are required to capture intermediate states of source code to find and analyse the problems that programmers face during development. Several tools exist that can record source code that changes during regular coding activities to enable studies of the evolution of the source code [65] .

---

<sup>3</sup> <http://promise.site.uottawa.ca/SERepository/>

<sup>4</sup> <https://sourceforge.net/>

In [66], the authors report that 80% of these snapshots can be compiled; these works usually capture immediate states on save, losing many intermediate edit steps in between.

A different form of incomplete source code still under development can be found on the question and answer sites like Stack Overflow or REDDIT<sup>5</sup>. These sites have become an important data source for empirical research on software engineering.

While existing research mainly focused on the textual parts of the posts, programmers ask many questions about a specific coding problem when they are stuck with their solution.

Their questions often contain incomplete code snippets, which are completed or rewritten by the community.

In [67] a solution is presented with an island grammar that can be used to parse Stack Overflow posts into Heterogeneous Abstract Syntax Trees (H-AST).

The grammar supports JAVA, XML, JSON, stack traces, and text fragments and can be used to transform released Stack Overflow data dumps.

### **3.2.3 Meta Data**

Another kind of dataset exists that does not focus on source code, but on the meta data that describes projects. In the context of this work, these datasets are interesting as an additional data source that can be integrated into analyses to enrich existing data.

In [68] presented OSSMOLE, which was one of the first advances to create a high-quality database of FLOSS project information for research. The authors achieve this through publishing standard analyses that enable replication of results and through facilitating reuse of analysis scripts by others.

GHTorrent<sup>6</sup> as presented in [69] is an effort to make the vast amount of development activities on GitHub available for research. The project stores the development events of GitHub repositories, for example, activities that include push, fork, or branch operations.

---

<sup>5</sup> <https://www.reddit.com/r/recommenders/>

<sup>6</sup> <http://ghtorrent.org/>

While openHUB <sup>7</sup> is a website that curates' meta data for a large number of open-source projects. The provided data consists of general meta-data such as repository URLs, main programming language, and license, and of several metrics regarding the source code, the activities of contributors, and historical data regarding the evolution of the project. The database can be accessed through a REST-based API.

### **3.2.4 Interactions**

One of the earliest and most extensive datasets of developer interactions is the public dataset of the ECLIPSE USAGE DATA COLLECTOR. This project intended to provide a means for plugin developers to analyse which functionality of ECLIPSE was actually being used by their users. To this end, the dataset contains a log of executed commands and activated windows grouped by user [2].

In [70] presented a recommender system called RASCAL. It is a recommender agent that tracks usage histories of a group of developers to recommend to an individual developer components that are expected to be needed by that developer. Further they introduce a content-based filtering technique for ordering the set of recommended software components and present a comparative analysis of applying this technique to a number of collaborative filtering algorithms.

In [71] presented the BLAZE tool that can be installed in VISUAL STUDIO 2010 to track which commands are invoked by the developer in the IDE. In [72] authors declare that they deployed it within ABB INC. and tracked activities of almost 200 developers, covering more than 30K hours of active development time.

In [73] proposed DFLOW to capture information about in-IDE activities. To this end, they capture source-code changes on a structural level (method rename), windows that are being interacted with, and the layout space occupied by open windows. Their dataset covers 750 hours of development work of 17 developers.

---

<sup>7</sup> <https://www.openhub.net/>

In the papers presented above, different aspects and approaches to data collection mechanisms have been highlighted, which are grouped into several categories depending on the collection methods.

The problems with above-mentioned approaches are that many of them focus on collecting metadata about codes from various repositories. In this aspect, a gap that can be identified is the lack of automatic suggestion from the code structure itself.

The method proposed in our approach delves deeper in the code structure by analysing and representing the code type through a control-flow graph, which can be used for generating a dataset for recommendation purposes.

### 3.3 Recommendation Engine to Analyse Data and Generate Recommendations

Recommendation engine and data analyses or in literature can also be found as static analyses are the most important part of recommender systems. There are many studies that treat this part, in the following will be some of them based on the literature reviewed.

In [74] authors present a recommender systems which generate recommendation by asking from users, this approach is called a call recommendation, which was later extended in [2]. The underlying approach in both cases identifies object instances in an intra-class analysis and extracts all method invocations on each instance, as well as a description of the surrounding source code.

In [75] proposed a recommender system called Precise, which focuses on parameter call sites. They extract several features from the structural context that describe the method invocation and its parameters: the called method, the enclosing method, methods that are called on the receiver, and methods that are called on the parameter.

In [76] mainly use identifiers to predict method calls. They extract all identifiers used in the source code and split names on camel-case humps. Even though control structures are not part of their model, they consider control structure keywords in the tokenization. They apply text-mining techniques such as stemming or removing stop words to unify the collected tokens.

In [77] authors implemented a tool that learns correct API usage from interactions of developers in their IDE. When code completion is triggered in the IDE, they extract features from the structural context around the trigger point, including the type, definition, enclosing statement, expression type, and enclosing method.

In [78] they solve the task of call recommendation by mapping it to a text-mining problem. They model sequences of method calls as sentences. Their main idea is to reduce the problem of code completion to a natural-language processing problem of predicting probabilities of sentences. They designed a simple and scalable static analysis that extracts sequences of



method calls from a large codebase, and indexes these into a statistical language model. We then employ the language model to find the highest ranked sentences and use them to synthesise a code completion.

In [79] presented a Context-Sensitive Code Completion (CSCC) system that is built on a simple and efficient algorithm. The approach tokenizes source code by traversing an AST. Tokens will be created for JAVA keywords, types, and method names. CSCC is context-sensitive in that it uses new sources of information as the context of a target method call. CSCC indexes method calls in code examples by their contexts. To recommend completion proposals, CSCC ranks candidate methods by the similarities between their contexts and the context of the target call.

Programmers often rely on documentation when learning about an API, but manually created documentation is hard to maintain and is thus often incomplete or outdated. Automated documentation generators try to solve this problem by mining source code repositories. They extract information that describes how to use an API and create documentation that can be consulted by developers to understand a system better. In the following paragraphs will be presented some papers that deal with documentation.

In [80] proposed CODEWEB, a tool that can be used to identify “reuse relationships” in source code. The developer can consult these tuples to learn about the correct API usage.

In [81] presented JAVA RULE FINDER, a tool that infers rules about correct usage of a framework directly from its source code. The rules are prepared in a textual format and serve as browsable documentation for developers.

In [82] authors propose a system that automatically generates the documentation of an API method. Instead of analysing the implemented behaviour in the method, they look at callers of the method and extract descriptive information from there.

In [83] proposed PROMPTER, a tool that proposes relevant postings on STACKOVERFLOW to developers while working in the IDE. It extracts the current programming context from the source code under edit. The context contains fully qualified names for types and methods, the source code of the enclosing element at the edit location, and a list of all types and methods used in the enclosing element that are defined outside of the project.

In [6], authors have presented a software product line called OpenCCE, which in reality is an Eclipse plugin. This solution combines the advantages of documentation and program analysis with ease of use and availability. In fact, the solution separates API users from the domain knowledge required to understand these APIs through an expert system.

The OpenCCE intend to guide developers through selecting the relevant cryptographic components to use, automatically generates the required code with the correct API calls for them and analyses the final program to ensure that no threats have been introduced during initial development or program evolution. As was noted in this paper, the main goal of OpenCCE is not to detect intentional malicious code but to avoid unintentional mistakes by non-expert developers.

Anomaly detection approaches learn characteristics of a typical correct program. The underlying models are then used to detect deviations from this established norm. Based on the literature review, some approach with defect and anomaly detection are presented.

In [84] authors propose a Detecting Missing Method Calls (DMMC) to detect missing method calls. They extract object usages that describe how an object instance is used. They encode the type of the object, the enclosing method, and all calls on it.

In [85] authors present PR-MINER, another detector for missing method calls. The tool extracts facts from a method such as the type of variable declarations, variable names, assignments, and calls and then uses a prefixed strategy to prevent name collisions in different scoping levels.

In [86] authors present an approach to build finite-state-automatons that describe valid protocols for using a specific type. The approach is based on their own framework that can be used to mine method sequences [87].

In [20] authors present a data-driven approach to vulnerability detection using machine learning, specifically applied to C and C++ programs. During the build process, they extract features at two levels of granularity. At the function level, they extract the Control Flow Graph (CFG) of the function. Within the control flow graph, they extract features about the operations happening in each basic block and the definition and use of variables.

In [22] present an end-to-end model for solving software defect prediction, one of the most difficult tasks in the field of software engineering. By applying precise representations, Control Flow Graphs and a graphical deep neural network, the model deeply explores programs' behaviour to detect faulty source code from others. The authors declare that their method improves the accuracies from 4.08% to 15.49% in comparison with the feature-based approach and from 1.2% to 12.39% in comparison with the tree-based techniques.

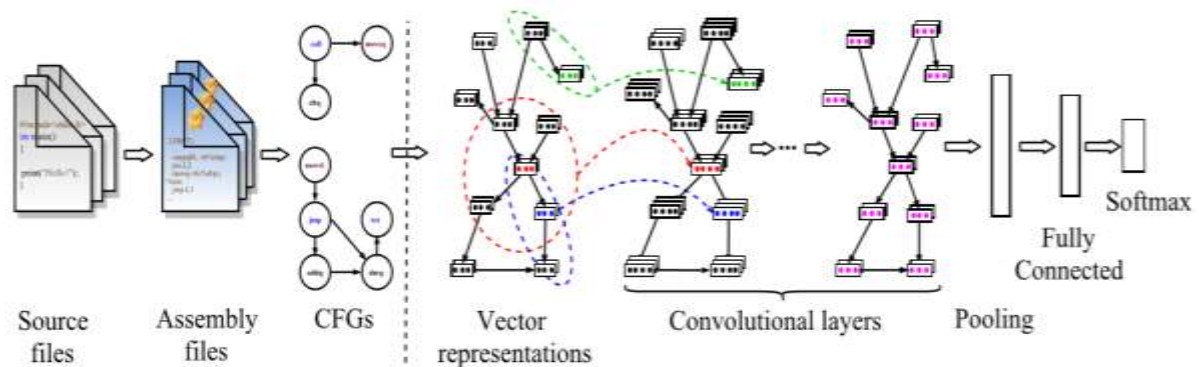


Figure 16: The overview of approaches presented in [22]

Many of the overhead analysed papers focus their attempts on building recommendations from simplistic code analysis. The focus is given either to the perspective of method invocations used in the codes subject of the analysis, either they imply a direct natural language processing techniques to gain insights from the code and later perform some machine learning technique for generating recommendations, or they extract limited features from the code, such as functions to generate the control-flow-graph (CFG) from the code for recommendation purposes.

The analysed above-mentioned papers failed to address the holistic approach towards code analysis, i.e. the programming codes are often too complex to be analysed from a single aspect alone. Our approach aims to use more holistic methods for code analysis by using Control Flow Graph techniques for code pattern analysis. In this way, a system will recommend a more effective and appropriate action for software developers to automatically ensure that their software is more secure.

## 3.4 User Interface to Deliver Recommendations

The user interface is the part by which users interact with recommender systems. Interaction can be implemented in different ways; some solutions are implemented via API or plug-in, which are integrated into existing development tools like Eclipse or Visual Studio, whilst some solutions are integrated which includes gathering data, static analyses and delivering recommendations to users.

The following will be presenting some solutions that are grouped in two groups based on solution approach.

### 3.4.1 Guiding Software Changes

In many online book shopping web sites or shopping for something else, by the time when we make our choice, we can encounter recommendations of the form, “Customers who bought this also bought....”. Such suggestions stem from purchase history. Buying two or more things together establishes a relationship between them, that web sites use to create recommendations.

In [88], present the eRose plug-in for the Eclipse integrated development environment (IDE) that realizes a similar feature for software development by mining past changes from version archives. This feature tracks changed elements and updates recommendations in a view after every save operation. For example, if a developer wants to add a new preference to the Eclipse IDE and so changes `fKeys[]` and `initDefaults()`, eRose would recommend “Change `plugin.properties`” because all developers who changed the Eclipse code did so in the past. The figure below illustrates the eRose system architecture.

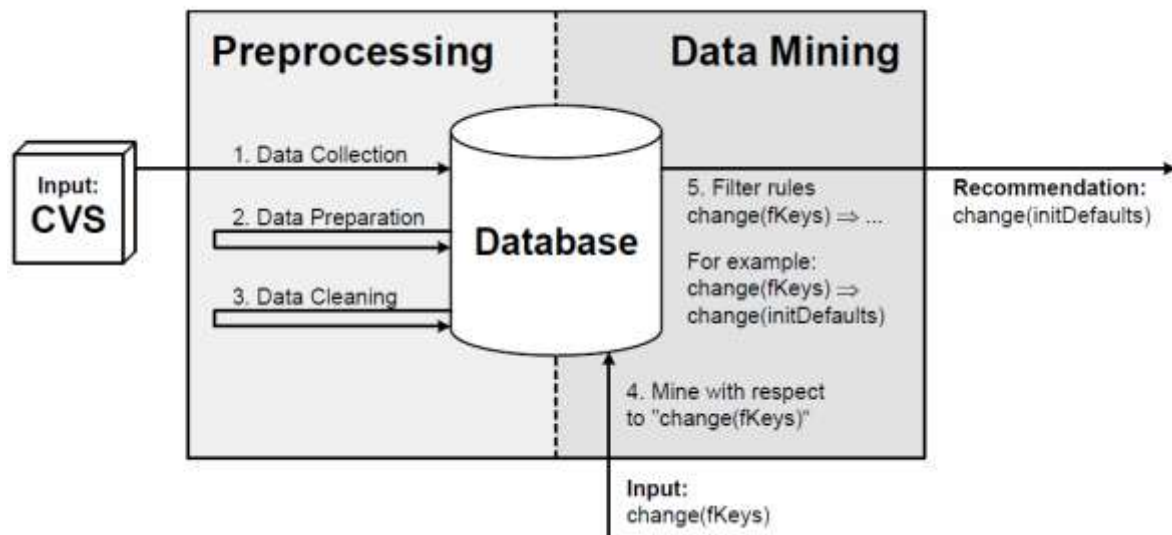


Figure 17: Structure of eRose [88]

In [89], a system called GROUMINER was developed to find patterns in API usages from source code repositories. Their approach is based on the syntax tree; they do not consider resolved types. The patterns, called Groums, contain ordered information about method calls, object declarations, and control points. In their follow-up work, in [90] they propose the GRAPACC snippet recommender that uses these patterns. In addition to extracting the graph-based features from the Groums of the code under edit, they also tokenize the code to create token-based features to support un-parsable code.

PROSPECTOR presented in [91] and PARSEWEB presented in [92] are two recommenders that propose ordered sequences that involve different API types. Both recommenders suggest call sequences that show the developer how to get from one API type to another.

### 3.4.2 Code Search

Code search is quite similar to snippet recommendations. The difference is that proposals point to existing examples that were observed in repositories or the local workspace instead of making probabilistic recommendations. The proposals cannot be directly integrated into the current editor, but it will point to source code that the developer can use to understand the correct usage of the API in question.

In [93] the authors present MAPO, a code search tool that mines method sequences from API usage examples. Methods calls include constructor calls, static and non-static calls, as well as casts.

In [94] the authors present the code search tool STRATHCONA for the ECLIPSE IDE. The tool retrieves relevant source code examples to help developers use frameworks effectively. When a developer wants to find how to do any specific code can highlight the partially complete code, and the context and ask Strathcona for similar examples. Strathcona<sup>8</sup> uses PostgreSQL queries to search for occurrences of each fact in a code repository.

Next, it uses a set of heuristics to decide on the best examples, which it orders according to how many heuristics select them. It returns the top 10 examples, displaying them in two formats: a structural overview diagram and highlighted source code to show similarities to the developer's partially complete code.

Developers can also view a rationale for a proposed example [95]. The STRATHCOMA User Interface and the recommendation process is depicted in the figure below.

---

<sup>8</sup> Strathcona prototype: <http://lsmr.cs.ualgary.ca/strathcona>.

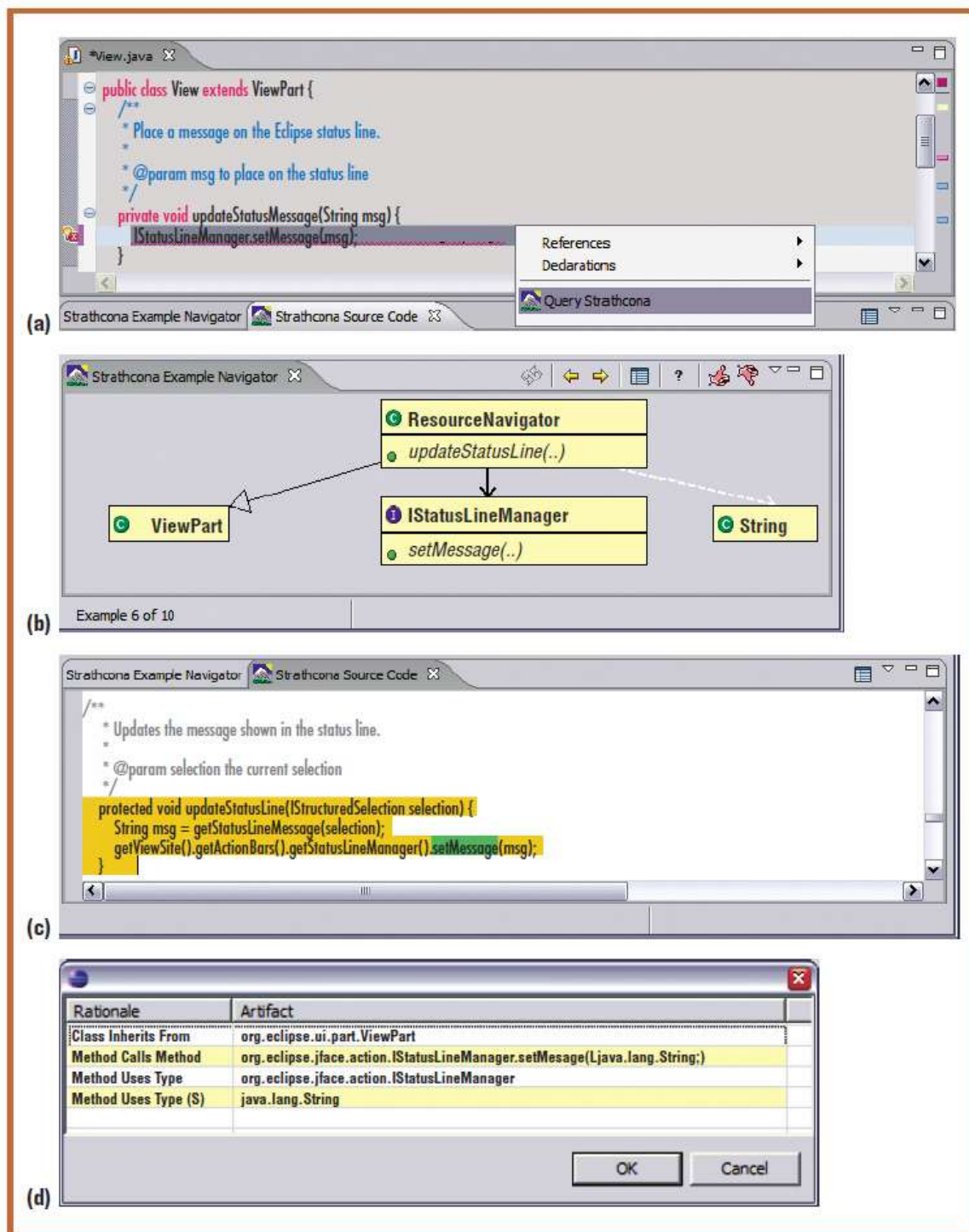


Figure 18: Strathcona user interface: (a) recommendation query; results in (b) a structural overview and (c) highlighted source code; and (d) rationale for recommendation [95].

In [96], a system called MUSE was proposed. It represents a tool that provides developers with examples of how to use a particular method. The tool requires the source code of the target API as well as a list of its clients. Whenever a call to one of the target API public methods is observed in the client code, an intra-procedural backward slice is created that shows how to get to this call.

In [97] presents CODECONJURER that supports developers by searching for working code that follows a specified UML-like syntax. The involved static analysis is based on the structural context only.

In almost all of the papers mentioned above, there is a lack of attention given to Control Flow Graphs (CFG) for code pattern extraction.

CFGs are well-founded and described in software testing. However, as the literature review conducted by the authors suggest that very little attention has been given to CFGs for code pattern analysis that can be used for generating datasets which furthermore can be utilised in Machine Learning for code recommendation. As a result, this research path is worth exploring.



## 3.5 Chapter Summary

Based on the literature consulted during this work, the recommendation systems for software engineering (RSSEs) are defined as software tools introduced specifically to help software development teams and stakeholders deal with information seeking and decision-making.

They comprise three main components, which are [95]:

- Mechanism to collect data,
- Recommendation engine to analyse data and generate recommendations,
- A user interface to deliver recommendations.

In this work, we have presented past work from different researchers in the field of recommender systems, different approaches to the use of data mining algorithms for static analysis and different tools that are developed as an outcome of these researches and which now are using by developers during coding processes.

Based in literature review, in the table below are presented publication grouped by recommendation task and theirs sub categories.

In the last section of this work we presented some datasets that can be used for system recommenders. Some of them are online and are free to use.

For our work, we have downloaded data from two platforms; Stack Overflow and GitHub. While our work will be concentrated on recommender systems for securing software engineering, we downloaded only projects and codes that are related to cryptography.

As an initial proof-of-concept, we have collected and created two datasets.

A dataset with data from Stack Overflow, which contains about 5,660 rows of data and another, is a dataset with data from GitHub, which contains about 25,250 data rows.

The data in two datasets mentioned before contains only cryptography codes, because implementation of recommender application that will be an outcome of this work will be dedicated to developers that are non-expert of cryptography [31].

## Part III

# Mechanism to Collect Data

# 4

## Dataset Creation

## 4.1 Introduction

Recommender systems for software engineering (RSSE) have several important elements in them, and one of the most essential elements are datasets, which are used as inputs.

For researchers in recommender systems finding reliable data or dataset is one of the biggest challenges.

Although source-codes are available in many public repositories of open-source projects, it is difficult to find an appropriate dataset that can be directly used as a source for a recommender system.

Nowadays there are various source-code based RSSE's that have been developed in several areas or fields, such as: useful for code compilation, code search or code snippet mining. For all these types of systems, a useful data source must be obtained [98].

Some of them have been discussed in section 3.1 of related work. Also, a recurring challenge for new RSSE is to find suitable projects that can be used as input for their static analyses that extract information. There are various sources from which sketchy program/code data can be obtained, and such software engineering data repositories are GitHub, StackOverflow, SourceForge etc, which make it easy to find vast amounts of source code that can be used for the above research.

However, open-source repositories cannot be directly used in analyses because some effort is required to prepare them, i.e., resolving code dependencies and making the code compile. Additionally, it is necessary to think of how the results of a batch analysis of many repositories are stored so that it is easy to access them in a structured way. Therefore, using and analysing a large number of open-source projects is usually non-trivial.

These online platforms contain valuable research data and can also be used by novice to experienced software developers to put code snippets in their projects. In both cases, these data cannot be directly used as source, because some effort is required in order to prepare them.

Instead of spending time assembling a dataset and making all sources compile, researchers can simply use a curated dataset for their research. Reusing an existing dataset has the added benefit of improving the comparability and reproducibility of the results.

Various website analyses and literature review have pointed out that there are much more datasets for Java compared to datasets in C#, and matters gets even more difficult since there is a lack of datasets in the Control Flow Graph (CFG) structure.

According to the TIOBE index [99], C# is among the most commonly used programming languages. Our objective is to provide these datasets, which can be valuable data sources for C# researchers. This also simultaneously and effectively opens the opportunity to study the effect of the C# programming language on RSSE.

The method proposed in our approach delves deeper into the code structure by analysing and representing the code type through a Control Flow Graph, which can be used for generating a dataset for recommendation purposes.

In this section, we will present a dataset in the structure of a Control Flow Graph (CFG) for secure cryptographic codes written in C#.

The dataset has been compiled from the data obtained by GitHub repositories and contains 19,590 lines of code extracted from 557 GitHub repositories.

The dataset created with CFG structure contains 1,191,340 lines of code.

The dataset is primarily meant for use in our research as source-based on RSSE and to answer questions that target correct usage of Application Programming Interfaces (API), but it could also be used in related areas, such as in software testing for anomaly detection.

## 4.2 Data Pre-processing

Data pre-processing is considered the most critical phase of analysing data. The data collection is usually a loosely controlled process, resulting in out-of-range values, e.g., impossible data combinations (e.g., Gender: Male; Pregnant: Yes), missing values, etc. Analysing data that has not been carefully screened for such problems can produce misleading results [100].

Data pre-processing includes activities such as data preparation, compounded by integration, cleaning, normalisation and transformation of data. Also, data reduction tasks such as feature selection, instance selection, discretization, etc. are part of data pre-processing.

The result expected after reliable chaining of data pre-processing tasks is a final dataset, which can be considered correct and useful for further processing.

The process of data processing is the key factor in increasing data quality. Data quality comprises of many factors, such as: accuracy, completeness, consistency, timeliness, believability, and interpretability [27].

Major tasks in data processing include data cleaning, data integration, data reduction and data transformation. The following section will shortly present the meaning of each of these tasks, which are explained widely in [27].

### **Data Cleaning**

Real-world data tend to be incomplete, noisy, and inconsistent. Data cleaning (or data cleansing) routines attempt to fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data.

### **Data Integration**

In many literatures data integration has the meaning of merging of data from multiple data stores. Careful integration can help reduce and avoid redundancies and inconsistencies in the resulting data set. This can help improve the accuracy and speed of the subsequent data mining process.

## Data Reduction

The data set will likely be huge. Complex data analysis and mining on huge amounts of data can take a long time, making such analysis impractical or infeasible. Data reduction techniques can be applied to obtain a reduced representation of the data set that is much smaller in volume, yet closely maintains the integrity of the original data. That is, mining on the reduced data set should be more efficient yet produce the same analytical results.

## Data Transformation

In data transformation, the data are transformed and consolidated into forms appropriate for mining. Normalization, data discretization, and concept hierarchy generation are forms of data transformation. Discretization and concept hierarchy generation can be useful, where raw data values for attributes are replaced by ranges or higher conceptual levels. For example, raw values for age may be replaced by higher-level concepts, such as youth, adult, or senior. Figure 19 below summarizes the data pre-processing steps described above.

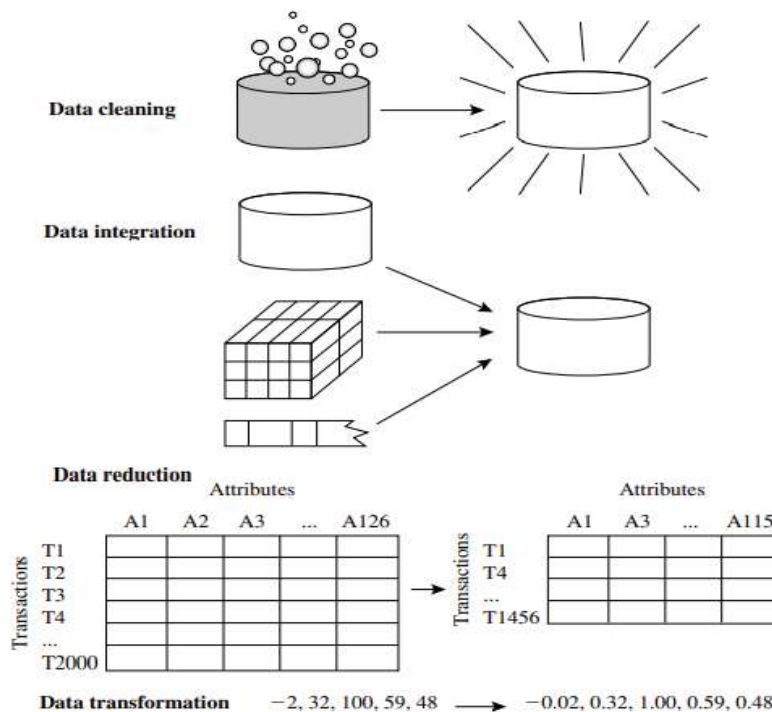


Figure 19 : Major tasks of data pre-processing [27]

As a conclusion for this section, the data pre-processing techniques can improve data quality, thereby helping to improve the accuracy and efficiency of the subsequent mining process. Data pre-processing is an important step in the knowledge discovery process, because quality decisions must be based on quality data. Detecting data anomalies, rectifying them early, and reducing the data to be analysed can lead to huge payoffs for decision making.



## 4.3 Dataset Creation Methodology

The process of creating the dataset has to go through several stages.

Initially, the GitHub API was used to find the required repositories. Since there are multiple repositories, the search was done by filtering only the repositories containing code in C#; from these, only the repositories containing cryptographic codes in C# were filtered.

The selected repositories are then saved as files on disk, whereby one entire GitHub repository is saved as one folder on disk. It has been observed that a repository can have more than one document.

For further processing, data are transferred to SQL database.

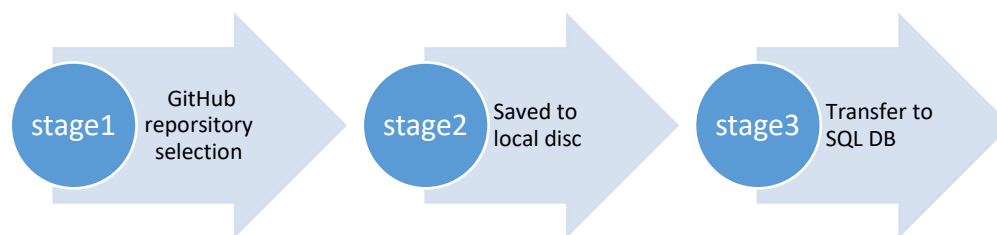


Figure 20: Dataset creating stages

The Figure 20 above illustrates main stages which have been used to create dataset.

### 4.3.1 Repository Selection

One of the most challenges is to select relevant repositories, because if the selected repository are not correct than recommendation also will not recommend correct output. Based on these prerequisites we should have to choose trusted repositories or mature repositories.

As was mentioned in before paragraphs, today, there exists considerable number of platforms in which developers collaborate with each other in different way. In some of them they upload their solutions with source code, such are GitHub, BITBUCKET, SOURCEFORGE etc. while some platforms serve for questions and answers between developers, and out of those the most popular is Stack Overflow.

For our research we decided to use GitHub as code resource to create dataset for next step of research work. Because data in GitHub are stored in repositories, we used a GitHub API to

download repositories. To ensure that repositories are quite trusted or mature enough, we skipped repositories with less than five commits.

Since this thesis will treat the aspect of code security in software engineering, therefore we selected only repositories with cryptography code and written in C#.

### **Statistics**

After selection there are 557 repositories that are downloaded from GitHub.

These repositories have more than 19,590 classes.

And in total there are more than 1,225,135 methods of classes

### **Saving to local disc**

To be more efficient for the future work, we have chosen to save data initially in the local disc. In this way each repository downloaded from GitHub was stored in single folder. Because of repositories includes more than one class, in the folder of each repository might have more than one document or file.

## **4.3.2 Transferring repositories to SQL database**

Transferring repositories to the SQL Database is the next step in creating the initial dataset.

First, in SQL DB a table structure with three columns was created.

- ID – represent an incremental integer to identify the unique document.
- Repository name - represent the name of the folder, in which are stored repository from GitHub.
- Class Name – represent classes for each repository.

After that, a simple .NET solution was developed, by which it can transfer repositories to SQL DB.

This activity included removing comments from class code and duplicates on the table rows.

The data in this table will be referred to as Dataset1 in the following sections.

### 4.3.3 Extension of Dataset

One of the primary intentions of the work on this thesis is to bring as a recommendation the complete method of the class to developer users. So, from the Dataset1 presented in the above section, a modification was required.

The idea is to divide methods of the class and to earn a new dataset when except of the classes are method of classes too.

New structure of the dataset is presented in the table below.

Class	Method Header	Method Body	Full method
-------	---------------	-------------	-------------

Table 2: New structure of dataset

In order to create the new structure of dataset presented above, we have implemented our own algorithm with a simple solution in .NET, by which will be able to divide methods for each class

#### How algorithm works

The first step is to identify the header of the method. We will use a pattern based on regular expression or regex for method header identifying.

The **regex** pattern is defined as it is shown below.

```
string pattern =  
"(public|private|static|protected|abstract|native|synchronized)+([a-zA-Z0-9<>._? ,\\[\\] ]+)(<:(\\s([a-zA-Z0-9_]+)))? +([a-zA-Z0-9_]+) *\\s*([a-zA-Z0-9<>\\[\\] ._? =, \\n]*\\s) *([a-zA-Z0-9_ ,\\n]*) *\\s*{"
```

Figure 21: regex pattern to identify method header

A regular expression (regex) is a pattern that the regular expression engine attempts to match in input text. A pattern consists of one or more character literals, operators, or constructs.

The extensive pattern-matching notation of regular expressions enables you to quickly parse large amounts of text to:

- Find specific character patterns.
- Validate text to ensure that it matches a predefined pattern (such as an email address).
- Extract, edit, replace, or delete text substrings.
- Add extracted strings to a collection in order to generate a report.

The Figure 22 below presents the main part of the algorithm, which will be able to find each method from the given class as an input parameter.

Each method is saved in database in the dataset structure as presented in the above section.

```

Regex expression = new Regex(pattern);
var myCode = Regex.Replace(code, "\n|\r|\t", "");
var results = expression.Matches(myCode);
foreach (Match match in results)
{
    try
    {
        int index = match.Index;
        int length = match.Length;
        string value = match.Value;
        string myHeader = value.Replace("@{", string.Empty);

        int OpeningBraceIndex = index + length;
        int startPosition = OpeningBraceIndex + 1;
        int endPosition = startPosition;
        int bracesToBalance = 1;

        while (true)
        {
            string myChar = myCode[endPosition].ToString();
            if (myCode[endPosition].ToString() == "{")
            {
                bracesToBalance++;
            }
            else if (myCode[endPosition].ToString() == "}")
            {
                bracesToBalance--;
                if (bracesToBalance == 0)
                {
                    break;
                }
            }
            endPosition++;
        }

        string myBody = myCode.Substring(startPosition - 1, endPosition - (startPosition - 1));
        string fullMethod = myHeader + " { " + myBody + " }";
    }
}

```

Figure 22: Algorithm for finding methods of class

The algorithm consists by two loops or iterations. The outer loop is used for iterating methods of the class, whereas the inner loop is used to find the method's body.

The flow of the algorithm goes through next steps as follows:

- First is identifying the method's header, which is done by using the regex pattern defined in the section above.
- After that are defined variables to measure the length and index of the snippet code.
- The algorithm uses curly braces to identify starting and ending points of the method body. So, the algorithm use a variable to hold the balance of the counting curly braces, in our case, ***braceToBalance***. (see figure above, Figure 22).
- When the algorithm detects an open curly brace in that time, the variable increase by one, when the algorithm detects close curly brace, the variable decrease by one.
- When the value of the variable is equal to zero, this is a flag for the algorithm to know that it is the end of the method and
- Continue searching for the next method from the input parameter.

## 4.4 CFG Structure Dataset

The goal of our approach in this research is to use more holistic methods for code analysis by using Control Flow Graph techniques for code pattern analysis. In this way, a system will recommend a more effective and appropriate action for software developers to automatically ensure that their software is more secure.

So, for the following work it is needed to create a structured dataset based on Control Flow Graphs rules.

As mentioned in the above section, a control flow graph (CFG) is a graph  $G = (N, E)$ , where the nodes are marked as  $N$  and represent statements of the procedure and the edges marked as  $E$  represent the transfer of the control between two statements [101].

Using CFG will be able to track the flow of execution and variable states. A node in a CFG represents a block of code which will always run sequentially. Edges in a CFG represent possible paths of execution.

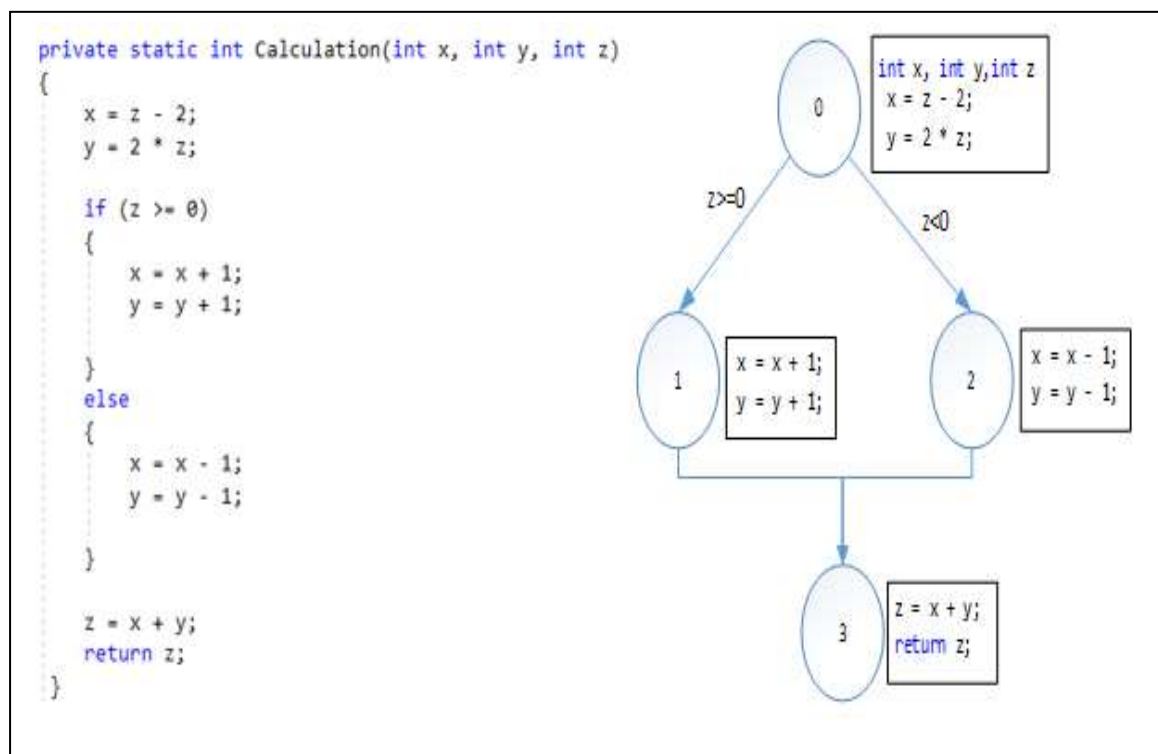


Figure 23: Control Flow Graph for *Calculation()* method

To explain in more detail how the Control Flow graph works, let's get an example with a method from a program with C# syntax. Let us name this method *Calculation()*, which has three input parameters, as it is presented in the figure above.

The flow of execution of *Calculation()* method can be represented as provided in the figure above. When ***Calculation()*** executes, the program always takes three parameters as input (int x, int y and z int). This is denoted by node 0, with an arrow going into it.

From node 0, there are two possible paths based in the value of parameter z. If  $z \geq 0$ , the label for edge (0, 1), then the program will increase the values of variables x and y by 1, as in node 1. Otherwise, if the value of z is negative  $z < 0$ , then the transition from node 0 to node 2 occurs. In this case, the program will decrease the values of variables x and y by 1, as in node 2. At the end, the program will calculate the value of variable z and return that value, as in node 3. Node 3 is the final node or exit point.

By this example, it can be seen that CFGs allow us to represent the program's complete execution flow, as well as data flow.

In the literature review, it was mentioned that CFGs are successfully applied to various problems in software testing including malware analysis [23], [24], or software plagiarism [25], [26].

CFGs are appropriate as well in the process of vulnerability detection. In [20] the authors present a data-driven approach to vulnerability detection using machine learning, specifically applied to C and C++ programs.

During the build process, they extract features at two levels of granularity. At the function level, they extract the Control Flow Graph (CFG) of the function. Within the control flow graph, they extract features about the operations happening in each basic block and the definition and use of variables.

In [22] presents a solution for software defect prediction by applying precise representations of Control Flow Graphs and a graphical deep neural network; the model deeply explores programs' behaviour to detect faulty source code from others.

The authors declare that their method improves the accuracies from 4.08% to 15.49% in comparison with the feature-based approach and from 1.2% to 12.39% in comparison with the tree-based approaches.

The next section presents an implementation developed for this research to create a dataset with a control flow graph structure.

#### 4.4.1 Implementing CFG structure

Because the approach in this research is to use the CFG structure dataset as a source of data for recommendations, we are implementing a simple .NET solution by which the dataset will be transformed into a dataset with a CFG structure.

For the given example as it is in Figure 24, the solution will transform it in control flow graph structure as it is presented in Figure 25.

```
private static int Calculation(int x, int y, int z)
{
    x = z - 2;
    y = 2 * z;

    if (z >= 0)
    {
        x = x + 1;
        y = y + 1;
    }
    else
    {
        x = x - 1;
        y = y - 1;
    }

    z = x + y;
    return z;
}
```

Figure 24: Given C# example



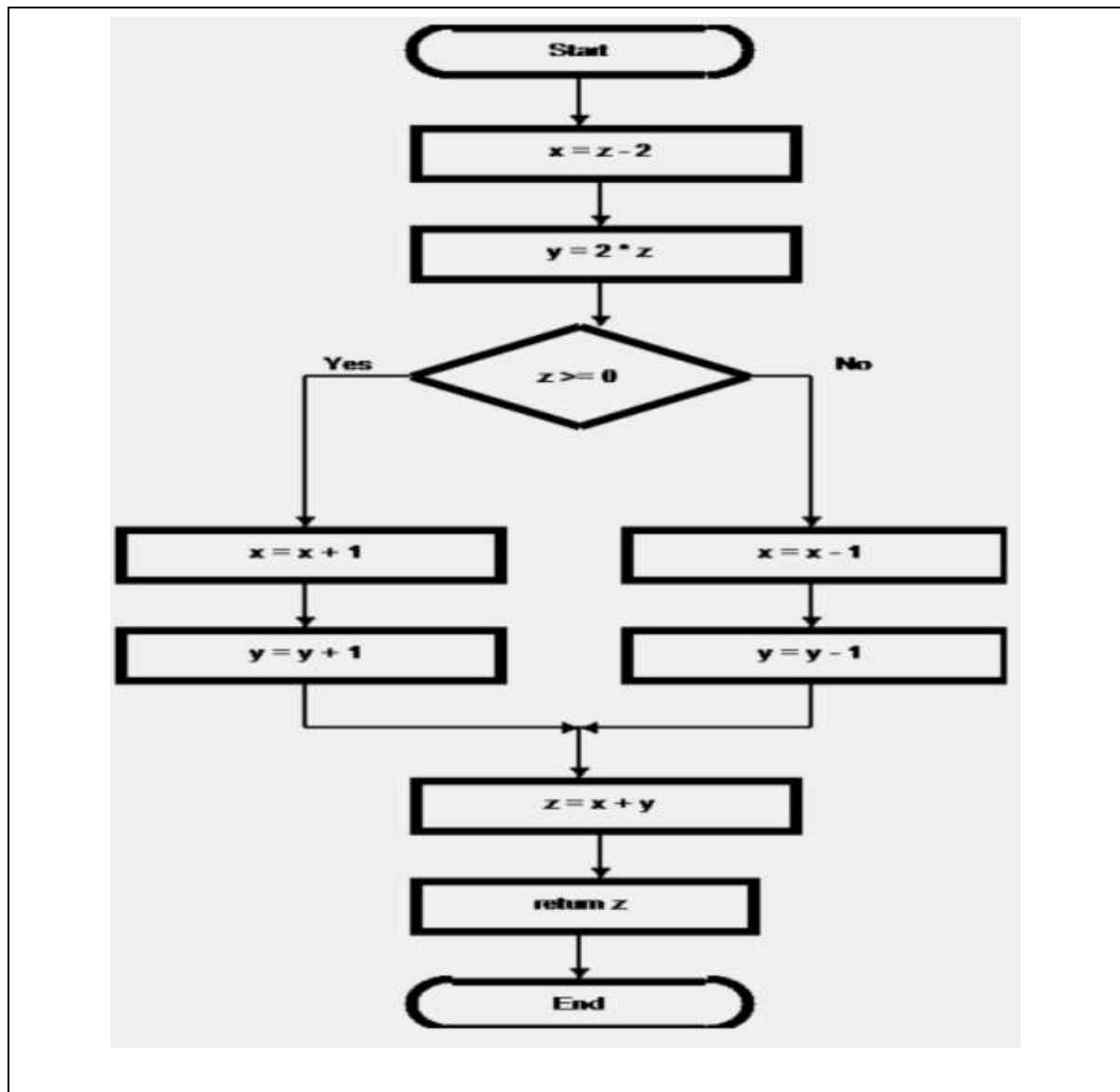


Figure 25: Transforming in CFG structure

The solution presented in the above figure, for a given an example as in Figure 24 will separate source code by blocks. So, each statement is presented as a block in Figure 25.

Every statement in the database will be saved as a new row in the table within the information inside the path. The figure above contains two paths, which it is determined by variable  $z$ .

In Figure 26 below, we outline an algorithm implemented in C# language, by which our solution is able to create dataset in control flow graph structure.

```

public static string[] OperatorSplit(this string str)
{
    List<String> pList = new List<string>();
    int main = 0, inside = 0;
    int prev = 0;
    for (int i = 0; i < str.Length; i++)
    {
        char c = str[i];
        switch (c)
        {
            case '{':
                main ++;
                break;
            case '}':
                main --;
                break;
            case '(':
                inside++;
                break;
            case ')':
                inside--;
                break;
        }
        if (main == 0 && c == ';' && inside == 0)
        {
            if (i - prev > 0)
            {
                string s = str.Substring(prev, i - prev);
                s = s.Trim();
                if (s.Length > 0)
                    pList.Add(s);
            }
            prev = i + 1;
        }
    }
    return pList.ToArray();
}

```

Figure 26: The own algorithm for creating CFG structure by code

The algorithm uses some coding rules from C#.

So, in C# each statement ends with semicolon character ';', the algorithm uses this character as a flag for ending a statement.

The parentheses '(' and ')' are used for two purposes:

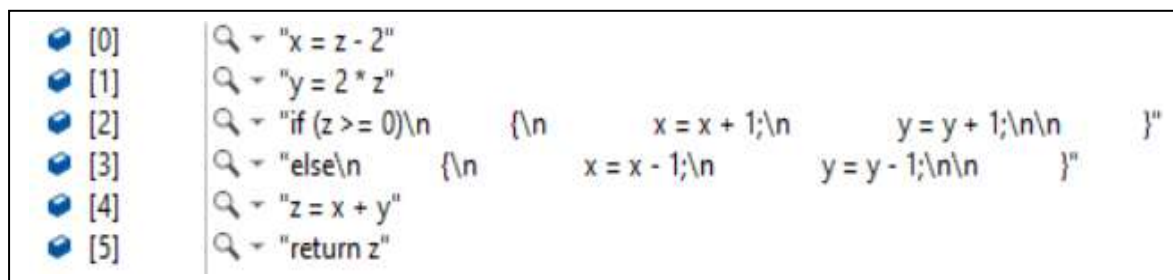
1. to control the order of operations in an expression, and
2. to supply parameters to a constructor or method.

Curly braces '{' and '}' is used to group a set of statement and deceleration. We use them along with loops and conditional statements in order to avoid confusion and to define a clear scope.

Every opening parentheses or brace must have a closing parentheses or brace in order to define a clear boundary.

#### 4.4.2 How algorithm works

The algorithm uses a class method as input parameter, and returns an output list of statements. The figure below presents a list of statements generated from the algorithm for the example from Figure 23.



[0]	Q - "x = z - 2"
[1]	Q - "y = 2 * z"
[2]	Q - "if (z >= 0)\n {\n x = x + 1;\n y = y + 1;\n\n }"
[3]	Q - "else\n {\n x = x - 1;\n y = y - 1;\n\n }"
[4]	Q - "z = x + y"
[5]	Q - "return z"

Figure 27: List of statements

Curly braces are used to identify the body of the method or conditionals statements, whereas the semicolon character ';' use to identify the end statements.

The algorithm's main idea is to count the input parameter's length and to read each character from it. Each time that character reading is ';', the **prev** variable will be increased for one. By using the substring feature (**string s = str.Substring(prev, i - prev)**) of the type of string variables, we can identify the next statement that will add to the return list.

### 4.4.3 Generating Control Flow

To enable the presentation of the code in the CFG structure, it is necessary for the algorithm to know on which side of the control flow to present statements. Based on the variable's value in the conditional statements, the algorithm will be able to add statements on the right side (left or right) that belong to. For these reasons, our solution, we have made extensions for each conditional statement. The figures below (Figure 28 and Figure 29) are presented modifications for *If* and *If - Else* conditional statements.

```
public If(string oprtr)
{
    this.oprtr = oprtr;
    // ++count;
    _operator = new COperator(OperatorType.If, GetCondition(oprtr));
    _operator.LeftText = "Yes";
    _operator.RightText = "No";
    oprtr = oprtr.Substring(oprtr.BlockBeetweenEnd('(', ')')+1);

    oprtr = oprtr.Beetween("{", "}");
    _operator.PLeft = CPPFileAnalyzer.AnalyzeBlock(oprtr);
}
```

Figure 28: Extending If conditional statement

```
public If Else(string o)
{
    o = o.Substring("else ".Length).Trim();
    if (o.StartsWith("if"))
    {
        _operator.PRight = new List<COperator>();
        var i = new If(o);
        _operator.PRight.Add(i.GetOperator());
        return i;
    }
    if(o.StartsWith("{"))
        o = o.Beetween("{", "}");
    _operator.PRight = CPPFileAnalyzer.AnalyzeBlock(o);

    return null;
}
```

Figure 29: Extending If –Else conditional statement

#### 4.4.4 Dataset with CFG Structure

Using the algorithm explained above in a recursive way will create a new dataset with CFG structure. As mentioned in the section above, the algorithm uses as an input parameter a method of class and, as a result, will give a list of statements. This list of statements is saved to DB and in this way a new dataset with CFG structure is created.

This dataset contains more than 1 million rows.

The dataset has a structure with three columns

- ID- contains the Unique line number
- ParentID - which contains the ID of the class from the original dataset
- Value - contains the statement generated by the solution for each method of a class in original Dataset1

This dataset we will refer to as Dataset2 in the following sections.

## 4.5 TF-IDF Calculation

Based on the literature review Term Frequency - Inverse Document Frequency (TF-IDF) is one of the most popular term-weighting schemes today. A survey conducted in 2015 showed that 83% of text-based recommender systems in digital libraries use TF-IDF [102].

Because algorithms of Machine Learning usually deal with numbers, and natural language is text. So, we need to transform that text into numbers, otherwise known as text vectorization. It's a fundamental step in machine learning for analysing data, and different vectorization algorithms will drastically affect the end results, so you need to choose one that will deliver the results you're hoping for.

Once you've transformed words into numbers in a way that machine learning algorithms can understand, the TF-IDF score can be fed to algorithms such as Naive Bayes, Support Vector Machines, Cosine Similarity, etc improving the results of more basic methods like word counts.

Vectorising a document is taking the text and creating one of these vectors, and the numbers of the vectors somehow represent the content of the text. TF-IDF enables us to associate each word in a document with a number that represents how relevant each word is in that document. Then, documents with similar, relevant words will have similar vectors, which is what we are looking for in a machine learning algorithm.

Calculation of TF-IDF in our case, will be used to find frequencies of the word, how many times a word is displayed in all documents in the generated datasets. For that calculation we use a simple algorithm developed in Python by using ***scikit-learn*** library which is known also as ***sklearn***.

### What is scikit-learn

The project was originally started back in 2007 as part of the Google Summer of Code while the first public release was made in early 2010.

scikit-learn<sup>9</sup> is an open source Machine Learning Python package that offers functionality supporting supervised and unsupervised learning. Additionally, it provides tools for model development, selection and evaluation as well as many other utilities including data pre-processing functionality.

More specifically, scikit-learn's main functionality includes classification, regression, clustering, dimensionality reduction, model selection and pre-processing

So, scikit-learn is definitely one of the most commonly used packages when it comes to Machine Learning and Python.

#### 4.5.1 TF-IDF calculation algorithm

In the Figure 30 below is presented part of algorithm used to calculate TF-IDF. The input of data for this algorithm are data from Dataset1 respectively Dataset2. By using *pyodbc*<sup>10</sup> library, the algorithm will be feed with data direct from SQL DB for each of dataset mention earlier.

The algorithm as output will generate a vectorised file document which will be able to use later as input from any Machine Learning (ML) algorithm, in our case Cosine Similarity.

---

<sup>9</sup> <https://scikit-learn.org/>

<sup>10</sup> <https://pypi.org/project/pyodbc/>

```

terms = vectorizer.vocabulary_
idx2terms = {}
for term in terms:
    idx2terms[terms[term]] = term

doc_matrix_str = ''
for term in terms:
    doc_matrix_str += '\t{}'.format(term)
doc_matrix_str += '\n'

for i in range(X.shape[0]):
    docstr = '{}\t'.format(doc_ids[i])
    doc_indices = list(X[i].indices)
    for idxterm in idx2terms:
        if idxterm not in doc_indices:
            docstr += '0.0\t'
        else:
            idx = doc_indices.index(idxterm)
            docstr += '{:.4f}\t'.format(X[i].data[idx])
    docstr += '\n'
    doc_matrix_str += docstr

    if len(doc_matrix_str) > 100000:
        fout_doc.write(doc_matrix_str)
        doc_matrix_str = ''

fout_doc.write(doc_matrix_str)

```

Figure 30: Main part of algorithm for calculating TF-IDF

TF-ID calculation is done for both datasets (Dataset1 and Dataset2). For Dataset1 the TF-IDF was calculated for all documents in dataset, while for Dataset2 a sample of random 1000 rows were used for calculation, because Dataset2 is larger and requires larger computing power which is not necessary at this stage of research.

In the Figure 31 below is presented a snippet from the vector dataset of the Dataset1, when the **x-axis** represents the words, while the **y-axis** represents the id of the document in Dataset1.



	using	system	io	security	cryptography	windows	forms	namespace	netcryptography	class	coder	idisposable	public	delegate	void				
1	0.0259	0.0253	0.0168	0.0155	0.0160	0.0164	0.0181	0.0062	0.0470	0.0067	0.1610	0.0253	0.0334	0.0287	0.0292	0.4608	0.0322	0.0138	0
2	0.2108	0.2058	0.0	0.0	0.0	0.1069	0.1181	0.0407	0.3061	0.0439	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0549	0.0402	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0262	0.0128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0424	0.0207	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0075	0.0110	0.0	0.0090	0.0093	0.0	0.0	0.0036	0.0273	0.0039	0.0	0.0	0.0543	0.0	0.0170	0.0	0.1186	0.0	0.0
7	0.0215	0.0210	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0157	0.0153	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0080	0.1675	0.0	0.0	0.0	0.0	0.0	0.0038	0.0580	0.0249	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0462	0.0113	0.0	0.0	0.0277	0.0	0.0	0.0	0.0111	0.0	0.0120	0.0	0.0	0.0714	0.0	0.0	0.0	0.0957	0.0
11	0.0	0.0667	0.0	0.0	0.0	0.0552	0.0522	0.0030	0.0	0.0032	0.0	0.0	0.0	0.0093	0.0	0.0	0.0	0.0323	0.0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0142	0.0139	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0051	0.0050	0.0	0.0	0.0	0.0	0.0	0.0	0.0050	0.0	0.0107	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15	0.0087	0.0085	0.0	0.0	0.0	0.0	0.0	0.0	0.0042	0.0	0.0045	0.0	0.0	0.0089	0.0	0.0130	0.0	0.0	0.0
16	0.0396	0.0322	0.0	0.0159	0.0163	0.0	0.0	0.0	0.0069	0.0	0.0	0.0	0.0	0.0	0.0	0.0098	0.0	0.0	0.0
17	0.0209	0.0204	0.0	0.0	0.0	0.0	0.0	0.0	0.0201	0.0	0.0435	0.0	0.0	0.1722	0.0	0.0	0.0	0.2080	0.0
18	0.0892	0.0290	0.0	0.0	0.0	0.0	0.0	0.0	0.0143	0.0	0.0155	0.0	0.0	0.1073	0.0	0.0	0.0	0.0444	0.0
19	0.0510	0.0124	0.0	0.0	0.0	0.0	0.0	0.0	0.0123	0.0	0.0133	0.0	0.0	0.0	0.0	0.0	0.0	0.0423	0.0
20	0.0247	0.0241	0.0321	0.0	0.0	0.0	0.0	0.0	0.0119	0.0	0.0129	0.0	0.0	0.0509	0.0	0.0126	0.0	0.0410	0.0
21	0.0338	0.0165	0.0	0.0	0.0	0.0	0.0	0.0	0.0163	0.0	0.0176	0.0	0.0	0.0872	0.0	0.0254	0.0	0.0561	0.0
22	0.0183	0.0089	0.0475	0.0	0.0	0.0	0.0	0.0	0.0088	0.0	0.0095	0.0	0.0	0.0471	0.0	0.0825	0.0	0.0759	0.0
23	0.0180	0.0176	0.0	0.0	0.0	0.0	0.0	0.0	0.0174	0.0	0.0187	0.0	0.0	0.0557	0.0	0.0271	0.0	0.0299	0.0
24	0.0114	0.0111	0.0	0.0	0.0	0.0	0.0	0.0	0.0110	0.0	0.0119	0.0	0.0	0.0353	0.0	0.0	0.0	0.0379	0.0
25	0.0276	0.0135	0.0120	0.0	0.0	0.0	0.0	0.0	0.0044	0.0	0.0096	0.0	0.0	0.0665	0.0	0.0069	0.0	0.0688	0.0
26	0.0075	0.0073	0.0098	0.0	0.0	0.0	0.0	0.0	0.0036	0.0	0.0039	0.0	0.0	0.0194	0.0	0.0057	0.0	0.0499	0.0
27	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0255	0.0	0.0275	0.0	0.0	0.0817	0.0	0.0	0.0	0.0392	0.0
28	0.0120	0.0117	0.0	0.0	0.0	0.0	0.0	0.0	0.0115	0.0	0.0125	0.0	0.0	0.0986	0.0	0.0	0.0	0.0794	0.0
29	0.0136	0.0066	0.0353	0.0	0.0	0.0	0.0	0.0	0.0066	0.0	0.0071	0.0	0.0	0.0140	0.0	0.0	0.0	0.0790	0.0

Figure 31: Snippet of Datsset1 of TF-IDF calculation

In the Figure 32 below is presented a snippet from the vector dataset of the Dataset2, when the **x-axis** represents the words, while the **y-axis** represents the id of the document in Dataset2.

case	dq	parameters	convert	frombase64string	node	innertext	file	handling	probably	should be in	separate	class	files	write
15776	0.3361	0.6345	0.3039	0.2716	0.3039	0.3361	0.3361	0.0	0.0	0.0	0.0	0.0	0.0	0.0
821	0.0	0.0	0.0	0.0	0.0	0.4014	0.2917	0.2917	0.2753	0.2267	0.1930	0.2917	0.2357	0.2547
6022	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15029	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13052	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3848	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7390	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
235	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13028	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
19921	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5725	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4782	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14830	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15266	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
19868	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
17748	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12868	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13721	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12141	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14368	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
447	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2138	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6778	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 32: Snippet of Datsset2 of TF-IDF calculation

By comparing two figures above (Figure 31 and Figure 32), it can be seen that in the Figure 32 appears much more values with zero, than in the Figure 31. This mean that in Dataset1 respectively in Figure 31 words have a higher frequency of occurrence than in Dataset2.

## 4.6 Cosine Similarity Index Calculation

A commonly used approach to matching similar documents is based on counting the maximum number of common words between the documents. However, this approach has a natural defect. That is, as the size of the document increases, the number of common words tends to increase even if the documents talk about different topics. The cosine similarity helps overcome this fundamental natural defect [103].

By definition, Cosine Similarity measures the cosine of the angle between two non-zero vectors of an inner product space. Two cosine vectors aligned in the same orientation will have a similarity measurement of 1, whereas two vectors aligned perpendicularly will have a similarity of 0. If two vectors are diametrically opposed, meaning they are oriented in precisely opposite directions (i.e. back-to-back), then the similarity measurement is -1.

However, Cosine Similarity is often used in positive space, between the bounds 0 and 1.

Cosine Similarity measurement starts by finding the cosine of the two non-zero vectors. This can be derived using the Euclidean dot product formula:

$$A \cdot B = \|A\| \|B\| \cos(\theta) \quad (5)$$

For given the two vectors the cosine similarity is defined as:

$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}} \quad (6)$$

Where  $A_i$  and  $B_i$  are components of vector A and B

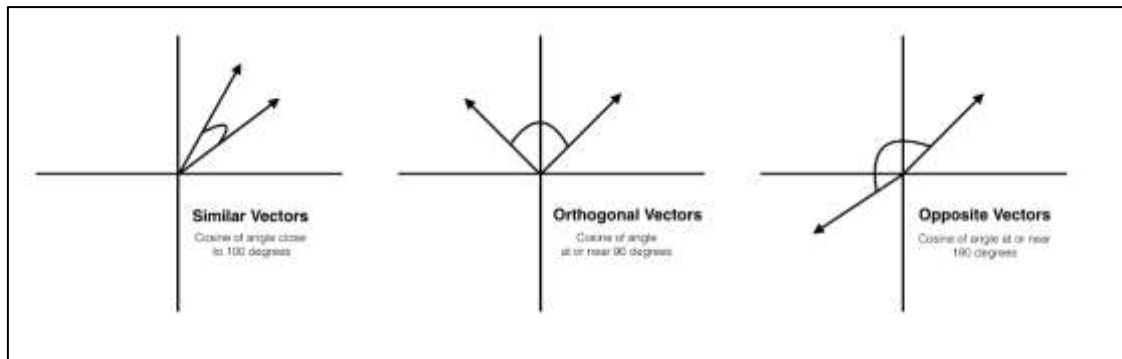


Figure 33: Vectors meaning

Source: <https://www.oreilly.com/library/view/mastering-machine-learning/9781785283451/ba8bef27-953e-42a4-8180-cea152af8118.xhtml/>

Cosine Similarity is not concerned, and does not measure, differences in magnitude (length), and is only a representation of similarities in orientation

#### 4.6.1 Applications of Cosine Similarity

Cosine Similarity has applications that extend beyond abstract mathematics. The measurement is used in processes of data mining, information retrieval, and text matching. Once the vectors are assigned to properties of a variable, the measurement becomes a valuable tool for understanding similarities between objects.

Machine learning uses Cosine Similarity in applications such as data mining and information retrieval. For example, a database of documents can be processed such that each term is assigned a dimension and associated vector corresponding to the frequency of that term in the document. This allows for a Cosine Similarity measurement to distinguish and compare documents to each other based upon their similarities and overlap of subject matter.

In this work of thesis, cosine similarity is used to find the similarity of words of documents in dataset presented in above sections.

Calculation of cosine similarity for both datasets are done by a solution implemented in Python and using *sci-kit-learn* and *pyodbc* packages.

Part of the implemented algorithm for calculating cosine similarity index is presented in Figure 34 below.

```

for i in range(X.shape[0]):
    a = X[i]
    a_ID = doc_ids[i]
    for j in range(i + 1, X.shape[0]):
        b = X[j]
        cosine = cosine_similarity(a, b)[0][0]
        if cosine <= 0.00001:
            continue
        b_ID = doc_ids[j]
        out_str += '{ }\t{ }\t{:.4f}\n'.format(a_ID, b_ID, cosine)

        if len(out_str) > 100000:
            fout.write(out_str)
            out_str = ''
fout.write(out_str)

```

Figure 34: Main part of algorithm for calculating CSI

The algorithm presented in the above figure, calculate the cosine similarity index between documents in the dataset. We eliminate rows with CSI=0, because for our purpose of work do not represent relevant values, by applying *cosine* <= 0.00001.

### Statistics

Some statistics from calculation of Cosine Similarity Index (CSI) in Dataset1 and Dataset2 are shown in the Table 2 respectively Table 3 below.

Indicator	No. for Dataset1	No. for Dataset2
Total No records	179,126,648	39,808
CSI =1	2,865	1,438
CSI <=0.5	178,213,280	37,564
CSI between 0.50 and 1.00	910,731	806

Table 3: CSI for Dataset1 and Dataset2

Based on data in the table above, results that most of the words have index lower than 0.5, which means that in selected samples in the both datasets, most of the words are unique, or has rare similarity.

## 4.7 Chapter Summary

Whereas dataset is one of the most important in RSSE, in this paper are presents two datasets, created from 557 repositories from GitHub.

The first dataset or Dataset1 as we refer to in this paper contains 19,590 lines of code. The structure of this dataset contains the codes for each class within a repository.

The second dataset or Dataset2 contains 1,191,340 lines of code. The structure of this dataset contains statements of the classes from Dataset1, which are generated by own algorithm implemented in .NET solution considering control flow graph rules.

For both datasets, TF-IDF and cosine similarity are calculated. From the calculated results, it can be seen that most of the words have an index lower than 0.5, which means that in selected samples in both datasets, most of the words are unique or has rare similarity.

These datasets can be used in future work as input in different ways in recommender systems field, like finding recommendations, intelligent code completion, code search, code generation, or snippet mining.

For this thesis work, these datasets will be used to generate recommendation for users.

## Part IV

# **Application and Evaluation**

# 5

## Recommender Engine



## 5.1 Introduction

Nowadays, everyone who is using any online platform experiences the increasing impact of products and applications that utilise machine learning. Out of all the applications of machine learning, recommendation engine techniques underwrite some of the most prolific utilities in everyday experience [53].

In real life, the recommendation engines have an important impact on connection between the customer and the platform being used. The influence of automated recommendation engines is inescapable in today's digital economy, whether it is a snippet of news feed on a social media homepage, what to watch next in a streaming service, a suggested article in an online news portal, a new friend recommendation, or a suggested product to buy from an e-commerce site. [104] [53].

### 5.1.1 Types of Recommendation Engines

The type of algorithms for building a recommendation engine is shown in the figure below; each of them has its own strengths and limitations [105].

Association analysis is included as a type of recommendation engine method in this taxonomy. Still, it is distinct because of the absence of user personalisation and the window of data it considers for a recommendation [106].

Recommendation engine methods are broadly classified into Collaborative Filtering and Content-based Filtering.

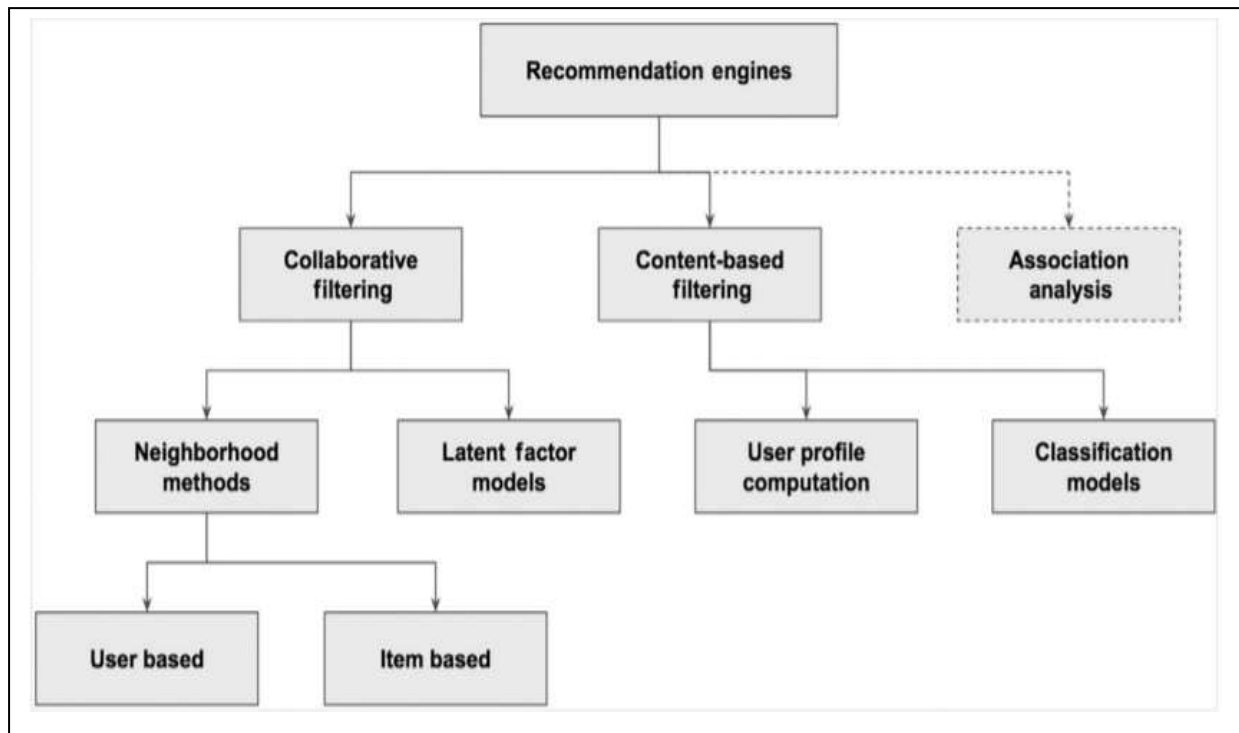


Figure 35: Taxonomy of recommendation engines [53]

### Collaborative Filtering

Collaborative filtering methods based on collecting and analysing a large amount of information on users' behaviours, activities or preferences. Using the user information to predict what users will like. The predictions are based on their similarity to other users.

A key advantage of the collaborative filtering approach is that it does not rely on machine analysable content. Therefore, it is capable of accurately recommending complex items such as movies without requiring an "understanding" of the item itself.

Some of the real-world applications of collaborative filtering include Google News recommendations [107], music recommendations by Last.fm, and recommended Twitter follows [108].

### Content-based Filtering

The content-based filtering method is based on a description of the item and a profile of the user's preference. In a content-based recommendation system, keywords are used to describe the items. A user profile is built to indicate the type of item this user likes.

In other words, these algorithms try to recommend items similar to those a user liked in the past (or is examining in the present). In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended. This approach has its roots in information retrieval and information filtering research.

The next section presents an approach for recommendation using cosine similarity calculated in the above sections.

## 5.2 Recommendation using Cosine Similarity

The cosine similarity measure is one of the most used similarity measures, but the determination of the optimal measure comes down to the data structures. The choice of distance or similarity measure can also be parameterized, where multiple models are created with each different measure. The model with a distance measure that best fits the data with the most minor generalization error can be the appropriate proximity measure for the data [109].

Document datasets are usually long vectors with thousands of variables or attributes.

For simplicity, consider the example of the vectors with X (1,2,0,0,3,4,0) and Y (5,0,0,6,7,0,0).

Calculation of the cosine similarity index for the given example is shown in Figure 36 below.

$$\begin{aligned}x \cdot y &= \sqrt{1 \times 5 + 2 \times 0 + 0 \times 0 + 0 \times 6 + 3 \times 7 + 4 \times 0 + 0 \times 0} = 5.1 \\||x|| &= \sqrt{1 \times 1 + 2 \times 2 + 0 \times 0 + 0 \times 0 + 3 \times 3 + 4 \times 4 + 0 \times 0} = 5.5 \\||y|| &= \sqrt{5 \times 5 + 0 \times 0 + 0 \times 0 + 6 \times 6 + 7 \times 7 + 0 \times 0 + 0 \times 0} = 10.5 \\ \text{Cosine similarity}(|x \cdot y|) &= \frac{x \cdot y}{||x|| \ ||y||} = \frac{5.1}{5.5 \times 10.5} = 0.08\end{aligned}$$

Figure 36: Cosine similarity index calculation [109]

The following sections presents a model of recommendation using cosine similarity. The proposed model will use datasets presented in the sections above.

### 5.2.1 Implementation

The model use as input parameter a snippet code from developer users and return as output a list of class methods as a recommendation.

Developer user will be able to choose which of proposed recommendation to use in their solution, in case of when recommendation engine propose more than one recommendation.

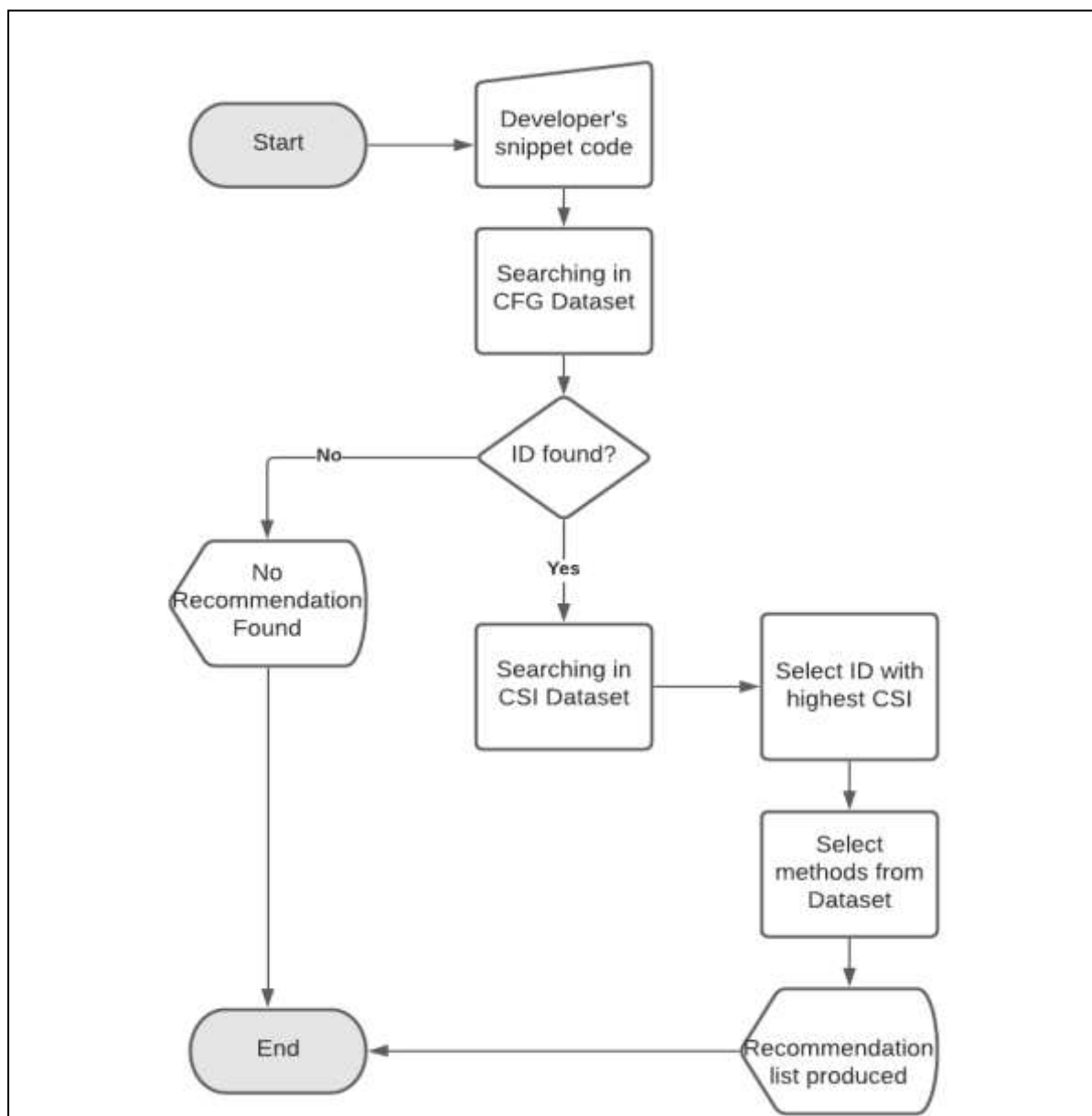


Figure 37: Recommendation model using cosine similarity

The Figure 37 above shows a model of generating recommendation based on calculation of cosine similarity index.

The model uses a dataset with three variants of them.

- CFG Dataset - contains data based on control flow graph structure
- CSI Dataset - contains calculated cosine similarity index for CFG Dataset
- Class Method Dataset – contains methods of classes separated from original dataset.

The Figure 38 below presents relationship between datasets mention above, which we are using in the model shown in the Figure 37 above.

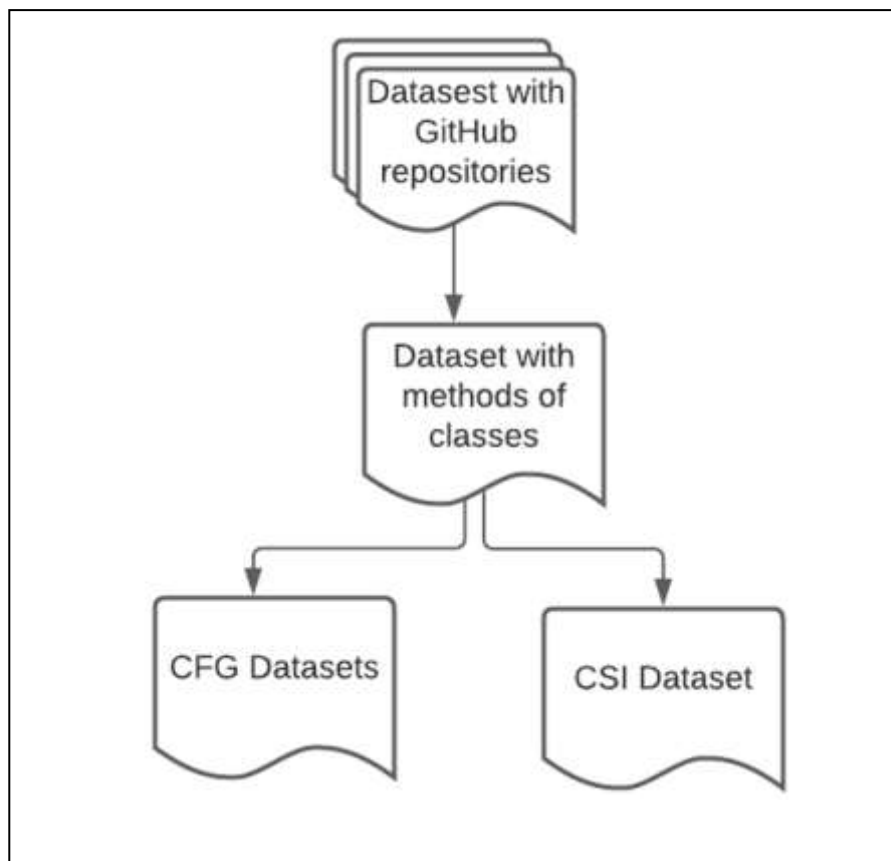


Figure 38: Relationship between datasets

### 5.2.2 How algorithm works

To explain how the algorithm of the model generate recommendation works let's use as input parameter this value *'aesDecrypt'*.

At the same time, this value can be considered as developer user snippet code.

- For this value algorithm first search in *'CFG Dataset'* to find the most similar value.
- The next step is searching in *'CSI Dataset'* to find data with the highest cosine similarity index.
- Because the algorithm will return many values, we decided to get the top five selected values with the highest CSI. For these data, the algorithm will search in *'Dataset with methods class'* to give user complete methods of classes as a recommendation. In this case, recommendation engine will propose recommendation, as it is shown in Figure 39 below.

MethodBody	CosineSimIndex	NoCommits
this.aesKey = new MetroFramework.Controls.Metro	67%	5
this.components = new System.ComponentModel.C	65%	15
this.components = new System.ComponentModel.C	65%	15
this.components = new System.ComponentModel.C	65%	15
this.metroLabel1 = new MetroFramework.Controls.	65%	15

Figure 39: Results after applying cosine similarity

The first column *"MatchedTile"* presents content similarity from queried data, *"MethodBody"* presents recommendations found in the dataset, *"Score"* presents index score of the similarity and *"NoCommits"* presents number of the commits in the GitHub for that repository.

## 5.3 Recommendation using Semantic Similarity

Based on literature review, semantic similarity is defined over a set of documents or terms, whereas, the idea of the distance between items is based on the likeness of their meaning or semantic content as opposed to lexicographical similarity [110].

The semantic similarity is often confused with semantic relatedness. Semantic relatedness includes any relation between two terms, while semantic similarity only includes "is a" relations. For example, "car" is similar to "bus" but is also related to "road" and "driving".

In [111] semantic similarity is defined as the similarity between two classes of objects in a taxonomy. A class  $C_1$  in the taxonomy is considered a subclass of  $C_2$  if all the members of  $C_1$  are also members of  $C_2$ . Therefore, the similarity between two classes is based on how closely they are related in the taxonomy.

There are different techniques for measure similarity as are: Dice coefficient, distance-based similarity, Resnik (IJCAI 1995), but much more popular is Wu & Palmer (ACL 1994) [111].

Wu and Palmer in [112] proposed the following similarity measure based on use of subclass links between classes:

$$sim(C_1, C_2) = \frac{2N_3}{N_1 + N_2 + 2N_3} \quad (7)$$

- Where  $N_1$  and  $N_2$  are the number of subclass edges from  $C_1$  and  $C_2$  to their closest common superclass;  $N_3$  is the number of subclass edges from the closest common superclass of  $C_1$  and  $C_2$  to the root class in the taxonomy



### 5.3.1 Implementation

In our model we use SQL Semantic Search solution to find recommendations for users. Semantic search tries to improve search by understanding the contextual meaning of the terms and tries to provide the most accurate answer from a given document repository. The technologies behind semantic search are mostly used to access unstructured data.

There are several techniques to implement semantic search [113].

One type of structured data is ontologies. An ontology formally describes available concepts in a specific domain. One possibility to query structured data is conceptual graph matching. In this method, each query and the data are represented as trees of concepts called ontologies. The search engine compares the query with each tree in the database and finds the best matching tree [114].

Based on Microsoft documentation, the SQL Semantic Search supports only unigrams. A unigram is a single word, whereas n-grams are word combinations, also known as term and phrase extraction respectively [115]. Semantic Search provides deep insight into unstructured documents stored in SQL Server databases by extracting and indexing statistically relevant key phrases. Then it uses these key phrases to identify and index documents that are similar or related.

Semantic search builds upon the existing full-text search feature in SQL Server. While full-text search lets query the words in a document, semantic search lets query the meaning of the document. Solutions that are now possible include automatic tag extraction, related content discovery, and hierarchical navigation across similar content.

To use semantic search in SQL Server, the first step is to install Semantic Language Database, which is a database with statistical base data to classify the keywords found in the indexed documents. To use semantic search on a text column of a regular table it is needed to create a semantic search index for that text column. The importance of the keywords it is determined by using a variation of the TF-IDF.

To find similar or related documents in a specific column, we will use a function called *SEMANTICSIMILARITYTABLE*.

*SEMANTICSIMILARITYTABLE* returns a table of zero, one, or more rows whose content in the specified column is semantically similar to the specified document [115].

After querying a value '*aesDecrypt*' in our database the result will be as it is in the figure below.

The first column "*MatchedTile*" presents content similarity from queried data, "*MethodBody*" presents recommendations found in the dataset, "*Socre*" presents index score of the similarity and "*NoCommits*" presents number of the commits in the GitHub for that repository.

MatchedTitle	MethodBody	Score	NoCommits
this.aesKey = new MetroFramework.Controls.MetroTextBox()	this.aesKey = new MetroFramework.Controls.MetroTextBox(); this.aesMess	100%	15
this.aesMessage = new MetroFramework.Controls.MetroTextBox()	this.aesKey = new MetroFramework.Controls.MetroTextBox(); this.aesMess	100%	15
this.aesResult = new MetroFramework.Controls.MetroTextBox()	this.aesKey = new MetroFramework.Controls.MetroTextBox(); this.aesMes	100%	15
this.aesEncrypt = new MetroFramework.Controls.MetroButton()	this.aesKey = new MetroFramework.Controls.MetroTextBox(); this.aesMess	100%	15
v.AddOptional(controls)	Asn1EncodableVector v = new Asn1EncodableVector(certReqId, certTemplate);	87%	5
return controls	List<Control> controls = new List<Control>(); foreach (Control child in pare	67%	256
controls.AddRange(GetSelfAndChildrenRecursive(child))	List<Control> controls = new List<Control>(); foreach (Control child in pare	65%	256
this.groupBoxAlg.Controls.Add(this.radioButtonDES)	this.components = new System.ComponentModel.Container(); System.Com	63%	144
this.Controls.Add(this.progressBar1)	this.components = new System.ComponentModel.Container(); System.Com	63%	144

Figure 40: Results after applying semantic search

Explanation:

- MethodBody:
- Score:
- NoCommits:

## 5.4 Presenting Recommendations

The user interface is a part of a model by which the user interacts with the application. In the RSSE, the first phase in creating a recommendation system is to generate a useful recommendation. But these recommendations must be provided with a user interface that allows the user to become aware that recommendations are available, to decide if any of the recommendations have value for them, and react to suggestions in order for that recommendation to be useful for them [116]. Deliver recommendation is the front end of the recommendation systems in software engineering and is responsible for how recommendations will appear to the user. Otherwise, the back end of the system is responsible for deciding what to recommend to the user. Usually, in the RSSE, more attention is paid to back end part than to the front end part because the good recommendation is more useful for the user than the way how the user can use it.

Some factors need to be considering when designing the user interface for a recommender such are understandability, transparency, accessibility, trust, and distraction. In the next sections will be descried each of them.

### 5.4.1 Understandability

In the literature for RSSE the *understandability* refers to whether a user is able to determine what a recommender is suggesting. There are two primary dimensions to understandability: obviousness and cognitive effort. These two dimensions are independent. A user interface can be non-obvious but once learned, may require significantly less cognitive effort.

The obviousness dimension describes how easy or hard it is for a user to recognise the kind of recommendation being provided. What a recommendation is may be readily apparent to a user [116] [117].

The effort dimension describes how much cognitive effort is required to understand the recommendation's meaning when presented. A recommendation for which the cognitive effort is at-a-glance will be easy for a user, once trained, to recognise the meaning [116] [117].

### 5.4.2 Transparency

A *transparency* refers to understanding a recommendation from a user. To understand what a recommendation is, a user must be able to determine why the recommendation is being provided [116]. In [118] it was described that transparency is related to rationale. If it is clear why a recommendation is being given, the transparency is high. In general, it is better when the more information that the system can provide about the rationale for a recommendation.

### 5.4.3 Accessibility

*Accessibility* refers the way how the user can access the recommendation, which he first understands what recommendation is and why it is provided [119]. Usually, a recommender system delivers to the users more than one recommender (as in our case), so the user must have the possibilities to choose which recommendation is more useful for their needs. Accessibility is related to but different from, understandability and transparency. Easy to use and transparent recommendations may be more likely to be easy to assess. In general, in RSSE, the system can make it easy for the user to assess the value of a recommendation by comparing the recommendation against the alternatives with respect to the user's values [116].

### 5.4.4 Trust

The *Trust* is related to how the user has faith if the recommendation provided from RSSEs is one that is need for him, so trust is especially important in RSSEs. An RSSE that makes a recommendation to change a user's code and that will make the change automatically requires a significant amount of trust [120].

### 5.4.5 Distraction

The *timing* refers the moment when the users ask from the system for the recommendation or the time when should an RSSE make a recommendation. For some RSSEs, delivering a recommendation in the middle of a user's work is critical. For example, as it is described in

[121] the *BeneFactor* can suggest that a user complete a manual refactoring by using a tool. In this case, the longer the RSSE waits to make the recommendation, the less time the user will save in taking the recommendation. If the user completes the refactoring manually, the recommendation has lost its value [116].

In our case the model will make recommendations only when the user explicitly asks for it.

### 5.4.6 Implementation

In our implementation, we used reactive initiation of the reaction of the user with the recommendations. This means that the user needs to ask for a recommendation by clicking a button as it showed in Figure 41 in area 2.

The interface for presenting recommendations to the users is divided into four areas or zones.

**The first area** – this part was reserved for the user. In this area, the user should write the code for which he can ask the model for recommendations.

**The second area** – in this part, the user can choose which implementation wants to ask for a recommendation. He can choose Cosine similarity implementation or SQL Semantic implementation. Both of these implementations are explained in detail in sections 5.2 and 5.3. By clicking in the button “Search” the model will start working to generate recommendations.

**The third area** – in this part the application will show recommendations produced by the model based on the implementation type chosen by the user. The results will appear in three columns. First column (*Recommendations*) presents the recommendations recommended by the model. The second columns (*Index*) present the index of the similarity of the codes asked from the user for the recommendation with the recommendation founded in the database by the model. The third column (*Commits*) presents the number of commits in the GitHub for specific recommendations, which is taken from GitHub when creating datasets as explained in detail in section 4.3

**Fourth area** – this part is reserved for the user to appear the selected recommendation from the third area. Double-clicking in the selected row from the area 3, in the part 4 will show the

whole code of the recommendation recommended by the model. The code appear in this part is the final result of the model, which can be used form the user in their projects.

Write Code

1

aesDecrypt

search by

☒ Cosine Similarity
 ☐ SQL Semantic

Search...

2

Recommendations

3

Recommendations	Index (%)	Commits
this.aesKey = new MetroFramework.Controls.MetroTextBox(); t...	67.0000	5
this.components = new System.ComponentModel.Container(); ...	65.0000	15
this.components = new System.ComponentModel.Container(); ...	65.0000	15
this.components = new System.ComponentModel.Container(); ...	65.0000	15
this.metroLabel1 = new MetroFramework.Controls.MetroLabel(); ...	65.0000	15

4

```

this.aesKey = new MetroFramework.Controls.MetroTextBox();
MetroFramework.Controls.Metro TextBox();
this.aesEncrypt = new MetroFramework.Controls.MetroButton();
MetroFramework.Controls.MetroButton();
this.aesKey.Lines = new string[0];
this.aesKey.MaxLength = 32;
this.aesKey.PromptText = "AES Key";
this.aesKey.SelectedText = "";
this.aesKey.TabIndex = 0;
this.aesMessage = new
MetroFramework.Controls.Metro TextBox();
this.aesResult = new MetroFramework.Controls.Metro TextBox();
this.aesDecrypt = new
MetroFramework.Controls.MetroButton();
this.SuspendLayout();
// aesKey
//
this.aesKey.Location = new System.Drawing.Point(24, 64);
this.aesKey.Name = "aesKey";
this.aesKey.PasswordChar = '\0';
this.aesKey.ScrollBars = System.Windows.Forms.ScrollBars.None;
this.aesKey.Size = new System.Drawing.Size(237, 23);
this.aesKey.UseSelectable = true;
this.aesKey.KeyDown += new
System.Windows.Forms.KeyEventHandler(this.aesKey_KeyDown);
// aesMessage
//
this.aesMessage.Lines = new string[0];
this.aesMessage.Location = new System.Drawing.Point(24, 94);
this.aesMessage.MaxLength = 32;
this.aesMessage.Name = "aesMessage";
this.aesMessage.PasswordChar = '\0';
this.aesMessage.PromptText = "AES Message";
this.aesMessage.ScrollBars =
System.Windows.Forms.ScrollBars.None;
this.aesMessage.SelectedText = "";
this.aesMessage.Size = new System.Drawing.Size(237, 23);
this.aesMessage.TabIndex = 1;
this.aesMessage.UseSelectable = true;
this.aesMessage.KeyDown += new
System.Windows.Forms.KeyEventHandler(this.aesMessage_KeyDown);
// aesResult
//

```

Figure 41: User interface for presenting recommendations

## 5.5 Chapter Summary

This chapter involved the modelling solution and presenting recommendations to the users.

The first part of the chapter presented implementations of the two types of approaches for producing a recommendation for the users.

Implementation of the generating recommendations using the Cosine Similarity approach was presented in section 5.2, and implementation of the generating recommendations using SQL Semantic Search approach, was presented in 5.3. Both of these implementation using the same datasets.

In the second part of this chapter, the user interface of the model is described, by which the user interact with the model. In our case users are software developers.

In the section 5.4 it was described in the detail how the users can ask for recommendations and how it can use the chosen recommendation in their project.

# 6

## Evaluation



## 6.1 Introduction

One of the most challenge in the area of Recommendation Systems for Software Engineering is how relevant are the recommendations produced by the tool and delivered to user developers. By which premises should base user developer to accept the relevant recommendation in cases when system tool recommend more than one recommendation.

The evaluation of previous recommender systems is usually focused on comparing the prediction quality of different algorithms, performance or mode sizes [122], [74]. However, RSSE are designed for the use by humans on typical developer machines with limited resources and evaluations of recommender systems should take this into account.

In our model, as it is mentioned above, we have implemented two approaches of recommenders; both of them use the same input data that is generated by the same static analysis, have the same interface to the outside, and create the same kind of proposals.

For both of these recommenders approaches, we also have to use two approaches to measure the recommendation's evaluation. First approach is an automated evaluation process that measures the effectiveness of a recommender system in terms of precision, recall and F-measure and the second approach is a subjective measurement by using experienced developers' thoughts.

In a nutshell, the evaluation process simulates the usage of the tool by a human user. When using a recommender system, a user typically enters a query into the system, in response to which the system returns a set of recommendations. The user then follows those relevant recommendations for the task at hand. After changing the code, the recommendation system may be queried again to see whether new recommendations are made [122].

**Definitions:**

- Precision quantifies the number of positive class predictions that actually belong to the positive class.

In RSS precision is defined as the ratio between relevant recommendations made and the total number of recommendations made by the system for a particular query [74]

$$\text{Precision} = \frac{\text{Recommendations}_{\text{made}\cap\text{relevant}}}{\text{Recommendations}_{\text{made}}} \quad (8)$$

- Recall quantifies the number of positive class predictions made out of all positive examples in the dataset.

In RSS recall is defined as the ratio between the relevant (correct) recommendations made by the system for the given query and the total number of recommendations that it should have made [74]

$$\text{Recall} = \frac{\text{Recommendations}_{\text{made}\cap\text{relevant}}}{\text{Recommendations}_{\text{relevant}}} \quad (9)$$

- F-Measure provides a single score that balances both the concerns of precision and recall in one number.

The F-Measure has been widely accepted by the research community as a means to correlate precision and recall by computing their harmonic means:

$$F = \frac{(1 + \beta^2) * \text{precision} * \text{recall}}{\beta^2 * \text{precision} + \text{recall}} \quad (10)$$

The F-Measure allows to emphasize either recall or precision by accordingly assigning the  $\beta$  parameter. For our evaluation, we equally weight precision and recall ( $\beta = 1$ ). The resulting formula is called the F1 measure [123] [74].

In the next section will be presented evaluation of the both approaches of recommenders organized in the below steps.

1. Performance measure, by calculating Precision, Recall and F1 - Measure
2. Subjective measure, by including 10 experience developers in this area

Calculating T-Test for steps 1 and 2 for both recommenders approach respectively, in order to find significance between them

## 6.2 Performance measures

To assess the effectiveness of a recommender system for a given query, we use recall, precision and F1 measures that express the ability of the system to

- Find all relevant recommendations, and
- Filter recommendations not relevant for the given query.

Recall is defined as the ratio between the relevant recommendations made by the system for the given query and the total number of expected recommendations that should have been made. Precision is defined as the ratio between relevant recommendations made and the total number of recommendations made by the system for a particular query.

However, with two numbers characterizing the "goodness" of systems, the question arises when a system is "better" than another. It might be the case that one system has a very high recall but a very low precision. Another system might have only a medium recall but also a medium precision. Therefore, F-Measure is used to correlate precision and recall by computing their harmonic means.

In our case the performance measure it was calculated for 10 random querying cases in the model.

For calculating performance measure for recommendation using Cosine Similarity approach, we followed the rules below:

- As a *relevant recommendations* are classified recommenders which has number of GitHub commits more than 5 and cosine similarity index more than 0.6
- As an *expected recommendations* are classified recommenders which has number of GitHub commits more than 5
- A total a *total number of recommendations* are selected top 5 recommenders with highest score of cosine similarity index.

The result for this approach are presented in the following Figure 42.

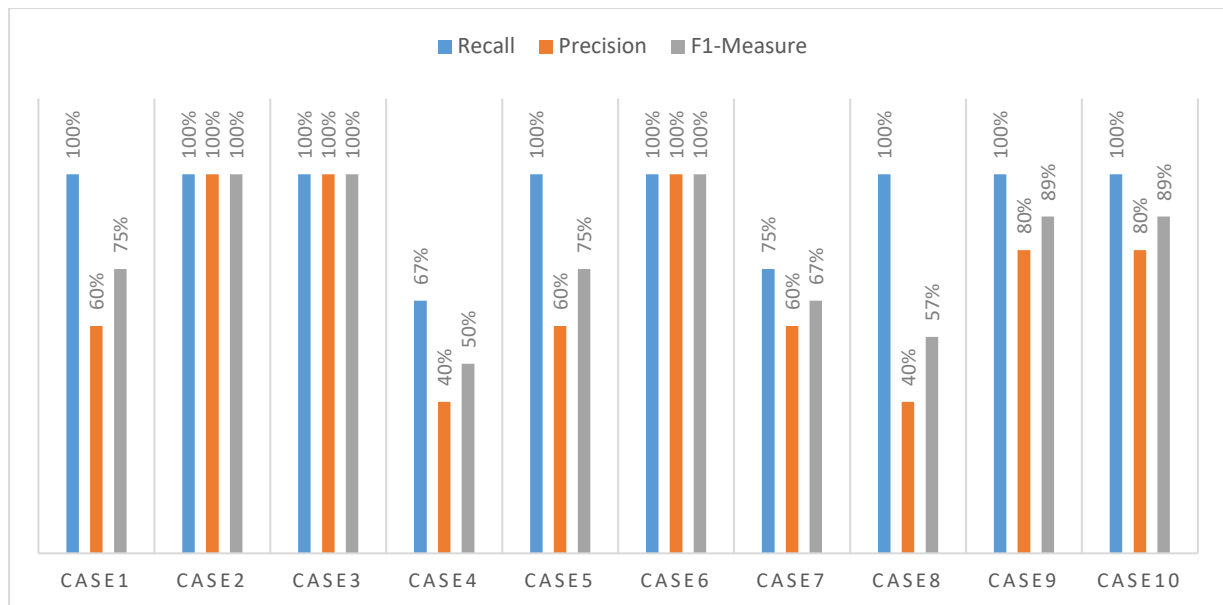


Figure 42: Performance measure for recommendation using Cosine Similarity

Average values of the performance measurements for data in the Figure 42 above:

- Recall = 0.94,
- Precision = 0.72 and
- F1-measure= 0.80

For calculating performance measure for recommendation using SQL semantic search approach, we followed the below rules:

- As a *relevant recommendations* are classified recommenders which has number of GitHub commits more than 5 and semantic search score more than 0.6
- As an *expected recommendations* are classified recommenders which has number of GitHub commits more than 5
- A total a *total number of recommendations* are selected all recommenders provided from the model

The results for this approach are presented in the following Figure 43.

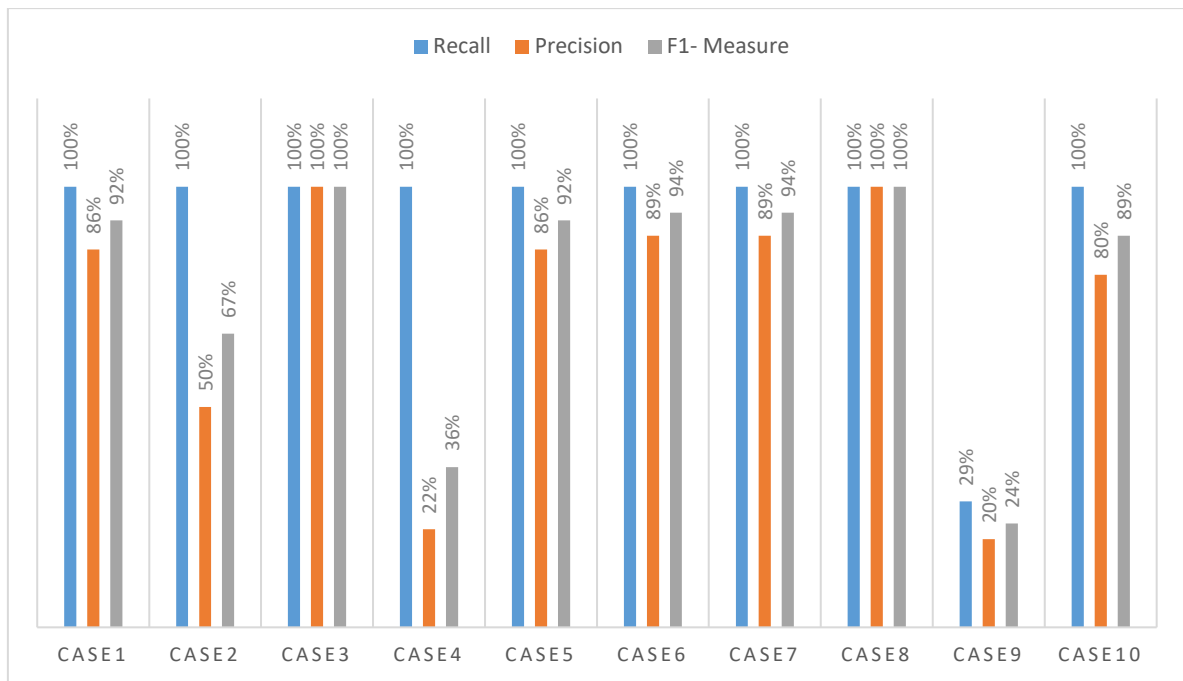


Figure 43: Performance measure for recommendation using Semantic Search

Average values of the performance measurements for data in the Figure 43 above:

- Recall = 0.93,
- Precision = 0.72 and
- F1-measure= 0.79

Based on this value recommendation produced by using Cosine Similarity approach are little more valid than recommendation produced by using SQL Semantic Search

## 6.3 Subjective Evaluation

As it was mentioned above, the evaluation process of the RSSE simulates the usage of the tool by a human user. Based on this reason for evaluation of the model we thought to hear the voice of the user developer.

We chose 10 experienced developers and ask from them to evaluate the same cases in which we calculated performance measurements. Developers had evaluated cases by answering a questionnaire with five question about the relevance of the recommendations for each of the cases:

- Not relevant
- Poor relevant
- Moderate
- Satisfactory
- Very Satisfactory

Then we calculated performance measurement (precision, recall and F1-Measure) for each evaluated cases by developers.

In the following Figure 44 and Figure 45 respectively, are presented the cumulative average of these measurements.

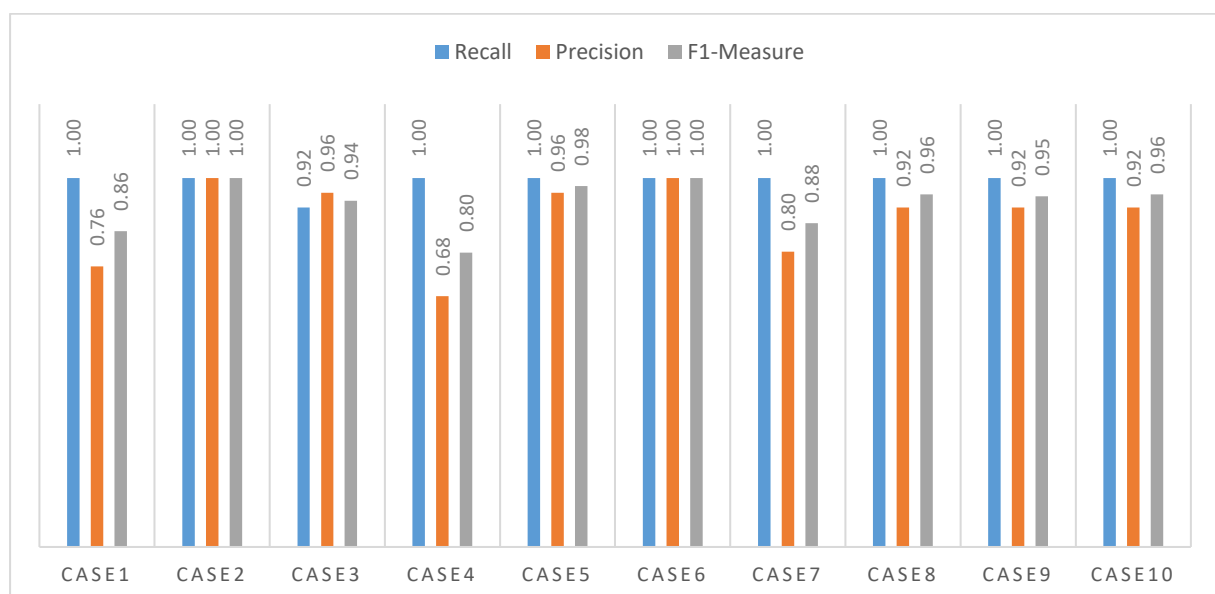


Figure 44: Performance measure for subjective evaluation of recommendation using Cosine Similarity

Average values of the performance measurements for data in the Figure 44 above:

- Recall = 0.99,
- Precision = 0.89 and
- F1-measure= 0.93

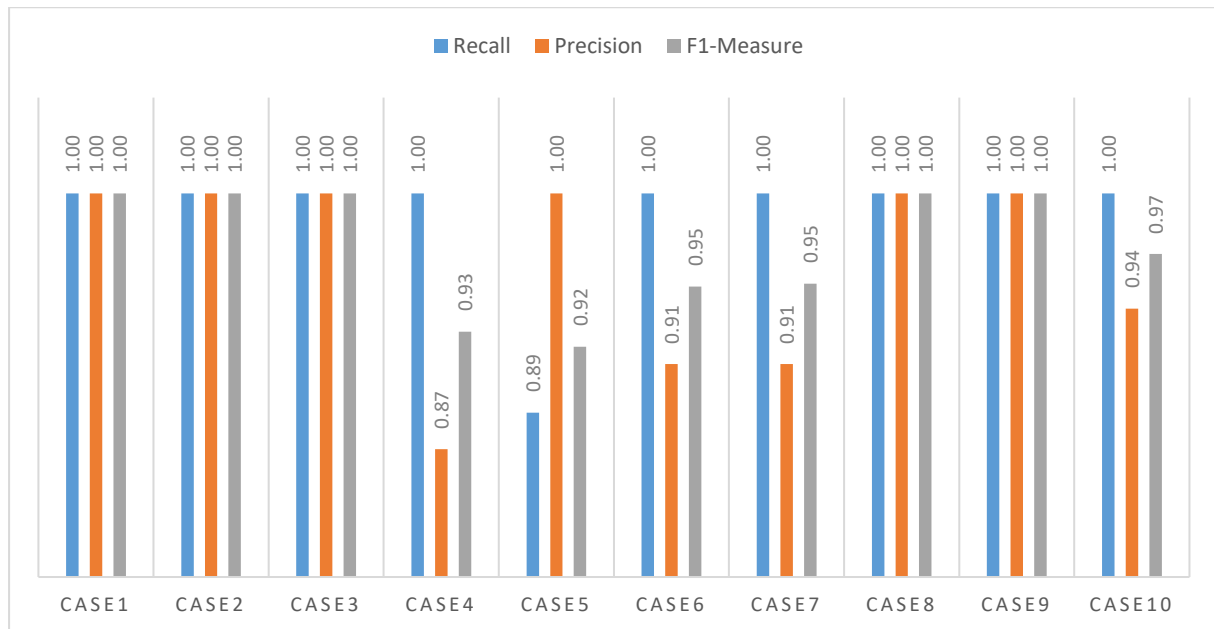


Figure 45: Performance measure for subjective evaluation of recommendation using Semantic Search

Average values of the performance measurements for data in the Figure 45 above:

- Recall = 0.99,
- Precision = 0.96 and
- F1-measure= 0.97

Based on this value recommendation produced by using SQL Semantic Search approach are little more valid than recommendation produced by using Cosine Similarity



## 6.4 t-Test

A t test is a type of statistical test that is used to compare the means of two groups. It is one of the most widely used statistical hypothesis tests in pain studies [124].

t-Tests can be divided into two types. There is the independent t-test, which can be used when the two groups under comparison are independent of each other, and the paired t-test, which can be used when the two groups under comparison are dependent on each other. t-Tests are usually used in cases where the experimental subjects are divided into two independent groups [124].

The formula for the two-sample t-test is shown below.

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{s^2 \left( \frac{1}{n_1} + \frac{1}{n_2} \right)}} \quad (11)$$

In this formula,  $t$  is the t-value,  $\bar{x}_1$  and  $\bar{x}_2$  are the means of the two groups being compared,  $s^2$  is the pooled standard error of the two groups, and  $n_1$  and  $n_2$  are the number of observations in each of the groups.

A larger  $t$ -value shows that the difference between group means is greater than the pooled standard error, indicating a more significant difference between the groups [124].

In our case we will calculate t-Test using type of t-Test independent groups. More of, we will calculate t-Test between three groups:

1. F1-Measure values of automatic performance measures of cosine similarity recommendations and SQL semantic search recommendations
2. F1-Measure values of automatic performance measures of cosine similarity recommendations and subjective evaluation for cosine similarity recommendations
3. F1-Measure values of automatic performance measures of SQL semantic search recommendations and subjective evaluation of SQL semantic search recommendations

Most statistical software includes a t-test function. This built-in function will take raw data and calculate the t-value. It will then compare it to the critical value, and calculate a p-value.

In our case calculation of t-Test for three groups listed above will be done using Excel data analysis tool.

Results of t-Test calculation for the three groups are showed in the figures below

	<i>Variable 1</i>	<i>Variable 2</i>
Mean	80.16	78.83
Variance	334.22	759.75
Observations	10.00	10.00
Hypothesized Mean Difference	-	
df	16.00	
t Stat	0.13	
P(T<=t) one-tail	0.45	
t Critical one-tail	1.75	
P(T<=t) two-tail	0.90	
t Critical two-tail	2.12	

Figure 46: t-Test calculation of first group

In the Figure 46 above are presented t-test calculation of the F1- Measure between two groups: a) automatic performance measures of cosine similarity recommendations and b) automatic performance measures of SQL semantic search recommendations.

As it showed, the t value is 0.13, which means there are no significant differences between these groups.

But based on the mean values we can conclude that the first group with mean value 80.16 has performed significantly better than second group with mean value 78.83

Consequently, recommendation using cosine similarity approach, it can be said that are more relevant than recommendation using SQL semantic search.

Results in the Figure 46 below present t- test calculation of F1 Measure between two another groups: a) automatic performance measures of cosine similarity recommendations and b) subjective evaluation for cosine similarity recommendations

Based on these results t value is -2.14 and p value is 0.05.

Because the p-value is 0.05, it can reject the null and assume that a significant difference exists between these groups. Based on the Mean values, the second group performed significantly better with a value of 93.30 than the first group with a mean value of 80.16.

	<i>Variable 1</i>	<i>Variable 2</i>
Mean	80.16	93.30
Variance	334.22	43.12
Observations	10.00	10
Hypothesized Mean Difference	-	
df	11.00	
t Stat	-2.14	
P(T<=t) one-tail	0.03	
t Critical one-tail	1.80	
P(T<=t) two-tail	0.05	
t Critical two-tail	2.20	

Figure 47: t-Test calculation of the second group

Results in Figure 48 below present t-test calculation of the F1 Measure between two other groups: a) automatic performance measures of SQL semantic search recommendations and b) subjective evaluation for SQL semantic search recommendations

Based on these results t value is -2.10 and p value is 0.06.

Because of p-value is near 0.05, the result is inconclusive in rejecting the null and an inconclusive assumption that a significant difference exists between these groups and based on the Mean values, the second group performed significantly better with a value 97.21 than the first group with mean value 78.88.

	<i>Variable 1</i>	<i>Variable 2</i>
Mean	78.88	97.21
Variance	753.99	10.40
Observations	10	10
Hypothesized Mean Difference	0	
df	9	
t Stat	-2.10	
P(T<=t) one-tail	0.03	
t Critical one-tail	1.83	
P(T<=t) two-tail	0.06	
t Critical two-tail	2.26	

Figure 48: t-Test calculation of the third group

## 6.5 Chapter Summary

This chapter deals with the model evaluation presented in the previous chapter.

There are two types of evaluations presented in this chapter: a) automatic performance measure and b) subjective evaluation performed by 10 experienced developers.

Based on the data gained by calculation for the first type of evaluations a), recommendation produced using the Cosine Similarity approach is more valid than the recommendation produced by using SQL Semantic Search.

Otherwise, based on the data gained by calculation for the second type of the evaluations b), recommendations produced by using the SQL Semantic Search approach are more valid than recommendations produced by using Cosine Similarity

And at the end of the chapter is presented t-Test calculation between groups presented on section 6.4.

Based on the values of  $t$  and  $p$ , there is significant difference between some of the groups.

Based on the values of *Mean*, when the performance of the measurements is done automatically from the model, recommendations recommended by using the cosine similarity approach performed better than those recommended by using SQL semantic search.

Otherwise, when t-Test calculations are done between performance measurements calculated automatically by the solution and performance by subjective calculation for two types of approaches for producing recommendations, the subjective evaluation performs better.

Part V

# Conclusion

**7**

# **Conclusion**

In this thesis there are produced several contributions. The previous parts are presented motivated the problem, it was presented a unique technique for producing a dataset in accordance with control flow graph structure, it was introduced models for generating recommendations and the infrastructure around them, and have presented in-depth, how we have applied techniques to evaluate the results from the model produced.

This final part will first discuss our insights in how the individual components of this work have facilitated our own applications. We will then present an outlook to future work that is enabled by the results of this work, and will end this thesis with a summary.

## 7.1 Discussion

This thesis has introduced a complete model first for creating dataset, producing recommendations and in the last using produced recommendations from the user. This work we think will be helpful to facilitate studies on the development process in software engineering in general. The main contribution we believe is in helping users to use secure codes in their project. After reporting the results in the previous chapters, we will now discuss our insights, the experienced advantages and disadvantages, and opportunities for improvement that we have identified. In the next section, we will go through the main components of this thesis.

### 7.1.1 Related Work

In this part of thesis are analysed a lot of research papers in order to find gaps between research papers and our proposal for work. Based on literature reviewed described in more detailed in chapter III we found some gaps and we grouped papers in three main parts: Mechanism to collect data, Recommendation Engine to Analyse Data and Generate Recommendations and User Interface to Deliver Recommendations. For each of these groups gaps identified are shown in the next section.

#### **Mechanism to Collect Data**

In the papers reviewed in this part we have found that some approaches, are mainly focused on collecting metadata about codes from various repositories. In this aspect as a gap that can be identified is lack of automatic suggestion from the code structure itself. The method proposed in our approach delve deeper in the code structure by analysing and representing the code type through a control-flow graph which latter can be used for generating a dataset for recommendation purposes.

#### **Recommendation Engine to Analyse Data and Generate Recommendations**

Many of the above analysed papers of this group focus their attempts on building recommendation from simplistic code analysis. The focus is given either to the perspective of



method invocations used in the codes subject of the analysis, either they imply a direct natural language processing techniques in order to gain insights from the code and latter perform some machine learning technique for generating recommendations or they extract limited features from the code, such as functions to generate the control-flow-graph (CFG) from the code for recommendation purposes. The goal of our approach is to use a more holistic methods towards code analysis by using Control Flow Graph techniques for code pattern analysis. In this way, a system will recommend a more effective and appropriate action for software developers to automatically ensure that their software is more secure.

### **User Interface to Deliver Recommendations**

In almost all of the papers on this group, there is a lack of attention given to Control Flow Graphs (CFG) for code pattern extraction. CFGs are a well-founded and described in software testing. However, as the literature review conducted by the authors suggest that a very little attention has been given to CFGs for code pattern analysis that can be used for generating datasets which furthermore can be utilized in Machine Learning for code recommendation. As a result, this research path is worth exploring.

#### **7.1.2 Dataset Creation**

In RSSE the data are one of the more crucial parts. Today there are in existence different platforms where developers posts theirs code or full project with source codes. Our intention in this work of thesis, is to help developers during developing processes by offering them the codes developed before from other developers, which thought are experienced developers.

In this thesis we have produced a dataset based on Control Flow Graph structure, based on data gathered from GitHub platform. The process of gathering data and producing dataset in structure of CFG it was presented in detail in Chapter IV.

So, we have gathered codes from 557 repositories from GitHub, and as a result we have produced a dataset with more than a million rows of code in CFG structure.

To be sure that repositories in GitHub are quite trusted or enough mature, we skipped repositories with less than five commits.

Also, on the dataset the vectorising was performed using TF-IDF and then cosine similarity calculation. From the calculated results, it was seen that most of the words have index lower than 0.5, which means that in selected samples in datasets most of the words are unique, or has rare similarity.

The main contribution in this part of thesis is a unique dataset with CFG structure, vectorising using TF-IDF and calculation using cosine similarity.

### **7.1.3 Application and Evaluation**

In this part of the thesis, a model is presented of the application implemented and the model is evaluated. The application includes generating recommendation by the model and presenting it to the user. In our case, the user is the software developer.

For generating recommendations, we have implemented two types of approaches. The first one is by implementing cosine similarity, and the second one is by implementing SQL Semantic Search technique.

Both of these implementations use the same dataset presented in Chapter IV.

By using the user interface of the application, the user will have possibility to choose which type of implementation want to use for generating recommendation. He can also choose the possibility to compare recommendations recommended by the application from the two options. It will depend from the user which recommendation he will want to use in their project.

One of the crucial things in RSSE is transparency to the user, which is closely related to the trustiness of the user in the application.

To achieve this in the user interface together with recommendations we deliver two other important columns: index and number of the commits in GitHub for each of the recommendations delivered.

The index is expressed in percentages. For the cosine similarity approach, the index presents the percentage of similarity of the user's code with recommendations delivered from the

model, otherwise, for the SQL Semantic Search approach the index presents the matching percentage of the user's code with recommendations delivered from the model.

Another challenge in the area of RSSEs is how relevant are the recommendations produced by tool and delivered to user developer.

To address this issues we have performed two approaches to measure the evaluation of the recommenders in terms of precision, recall and F1- measure.

First approach is an automated evaluation process based on the rules described in detail in section 6.2 and the second approach is a subjective measurement by using ten experienced developer's thoughts.

We performed also t-Test techniques to compare the means of two groups in statistical terms. We found that, based on the values of the *mean*, when the performance of the measurements is done automatically from the model, recommendations recommended by using cosine similarity approach performed better than recommendations recommended by using SQL semantic search. Otherwise, when t-Test calculations are done between performance measurements calculated automatically by the model and performance by subjective calculation for two types of approach for producing recommendation, the subjective evaluation performed better.

## 7.2 Future work

This thesis has presented technical research and implementation results in the previous part. Instead of closing a line of research, many opportunities for future work have popped up, some easier to achieve than others. In this chapter, we will present an outlook on future work that we anticipate.

**Improving the Datasets** - We have spent major effort for this thesis to compile a dataset with control flow graph structure by source codes gathering from GitHub platform. For our purpose the source codes gathered from GitHub are only from C# language and concentrated in cryptography codes. But, in our work, we have implemented a model which is not dependent only on the codes of C# and not only on cryptography codes. So, it is an open door for the future work to compile a dataset with source code from another programming language and also from different field of interest.

Another improvement in the dataset will be gathering data from another platform except for GitHub, and enabling the model to compare results between platforms. It will be very useful, for example, to compare data between GitHub and StackOverflow, and to analyse the impact of copy and paste behaviours of the developer.

In the StackOverFlow platform the developers post questions / problems and ask for solution. Most of the solutions in that platform are snippet codes with some explanation. Otherwise, in GitHub developers post their projects. So, considering this chance, in GitHub will be found the snippet codes from StackOverFlow without any modification. Analysing the impact of the codes from StackOverFlow in the real projects from the perspective of security will be another open opportunity for the future work.

**Recommendation Engines** – In this thesis we have implemented two approaches for recommendation engine: recommendation using Cosine Similarity and recommendation using SQL Semantic Search.

Both of these approaches are techniques of Machine Learning. It will be a good opportunity for future work, implementing another data mining algorithms using the same dataset, with the focus in Artificial Intelligence, or Neural Network etc.

**Upgrading User Interface** – In this thesis work we have implemented a user interface by which users have possibilities to ask from the application for recommendations, based on codes that they writes.

Developers use APIs of frameworks and libraries on a daily basis. As urgent future work it will be integrating our solution in the programming tools, in our case Visual Studio.

This will be very useful for the developers to ask from the model for recommendations in real time.

In our model, we store data in the local database. This approach can be very limited to use from worldwide users and probably users will have difficulties to configure the database for using model. Centralization of the database will be a good solution for this issue. As a future work in this aspect will be upgrading the model for using a centralization database in cloud.

## SUMMARY

Recommender Systems for Software Engineering (RSSEs) are software tools that can assist developers with a wide range of activities, from reusing codes to suggest developers what to do during development [5]. These tools can be used also to guide and recommend a developer for next activities, based on codes writes from other developers.

The purpose of this thesis is to pave the way for a generic platform for generating recommendations for developers during their daily programming activities. The model presented in this thesis is concentrated more on recommendations about codes for developers that are non-security experts. This was determined by datasets that we have created from source codes gathered from GitHub, which are from the area of cryptography and only from C# programming language. But, in general, we think that the model will be able to generate recommendations from any area and any programming language, based on the data that are in the dataset. So, in the future, if we want to use our model for generating recommendations for other area of software engineering or other programming language except C#, it will need to compile a dataset with data from that area or that programming language.

In part III of the work of this thesis, it was presented the way how we produced the dataset, which then is used for the modelling recommendation engine. GitHub is a very popular platform, which is used by developers for posting and sharing projects. We used this platform for gathering data to create our dataset for our work. We have created a dataset based on the control flow graph structure.

But, first, we have implemented an algorithm that will be able to generate a control flow graph structure from usually source codes. This implementation is another valuable contribution to this thesis. The dataset created is available and can be used by other researchers of different area of RSSEs.

In part IV it was presented the part of recommender engine, evaluation and user interface for delivering recommender to user.

The recommender engines in RSSEs have an important impact in the connection between the user and platform that he is using. In this thesis we build to way of recommender for generating recommendation to the user. The first one is based on Cosine Similarity technique, otherwise the second one is based on SQL Semantic Search. By using the user interface provided from our model, the user has possibilities to choose which one he want to use for generating recommendation. To address relevance of the recommendation produced from our recommender model, we performed evaluation of the recommendations produced by using values of precision, recall and F1-measure.

Overall, we think that the work presented in this thesis opens the road for novel studies. By using our dataset, we see a unique opportunity to study the recommender systems in securing software engineering processes, especially using the control flow graph technique. Our model has the potential to be valuable to the whole community during the developing processes.

We expect that future work presented in section 7.2 above will open an opportunity for other researches, and will have significant impact in the field of recommender systems in software engineering.

# References

- [1] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backe and S. Fahl, "Stack Overflow Considered Harmful? The Impact of Copy&Paste on Android Application Security," *IEEE Symposium on Security and Privacy*, 2017.
- [2] S. Proksch, L. Johanes and M. Mezini, "Intelligent Code Completion with Bayesian Networks," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, p. Article No. 3 , 2015.
- [3] M. P. Robillard, E. Bodden, D. Kawrykow, M. Mezini and T. Ratchford, "Automated API Property Inference Techniques," *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 613-637, 2013.
- [4] M. Robillard and R. DeLine, "A field study of API learning obstacles," *Empirical Software Engineering*, pp. 703-732, 2011.
- [5] M. Robillard, R. Walker and T. Zimmermann, "Recommendation Systems for Software Engineering," *IEEE Software*, pp. 80-86, 2010.
- [6] S. Arzt, K. Ali, S. Nadi, E. Bodden, S. Erdweg and M. Mezini, "Towards Secure Integration of Cryptographic Software," *ACM International Symposium on New Ideas* , pp. 1-13, 2015.
- [7] S. Clark, T. Goodspeed, P. Metzger, Z. Wasserman, K. Xu and M. Blaze , "Why (special agent) Johnny (still) can't encrypt: a security analysis of the APCO project 25 two-way radio system," in *Proceedings of the 20th USENIX conference on Security* , Berkeley, 2011.
- [8] A. Whitten and J. D. Tygar, "Why Johnny can't encrypt: a usability evaluation of PGP 5.0," in *Proceedings of the 8th conference on USENIX Security Symposium - Volume 8* , Washington, 1999.
- [9] D. Lazar, H. Chen, X. Wang and N. Zeldovich, "Why does cryptographic software fail?: a case study and open problems," in *Proceedings of 5th Asia-Pacific Workshop on Systems*, New York, 2014.
- [10] A. Eckhardt, "Various aspects of user preference learning and recommender systems," 2013.
- [11] M. Robillard, R. Walker and T. Zimmermann, "'Foreword,'" *Proc. Int'l Workshop on Recommendation Systems for Software Engineering*, in *ACM Press*, 2008.



- [12] H.-J. Happel and W. Maalej, "Potentials and Challenges of Recommendation Systems for Software Development," in *ACM*, 2008.
- [13] M. P. Robillard, W. Maale, R. J. Walker and T. Zimmermann, *Recommendation Systems in Software Engineering*, Heidelberg New York Dordrecht London: Springer-Verlag, 2014.
- [14] R. Arnold and S. Bohner, "Impact analysis: Towards a framework for comparison," in *Proceedings of the Conference on Software Maintenance*, ICSM.
- [15] U. Pakdeetrakulwon, P. Wongthongtham and W. V. Siricharoen, "Recommendation Systems for Software Engineering: A Survey from Software Development Life Cycle Phase Perspective," in *ResearchGate*, 2014.
- [16] L. Ponzanelli, "Holistic recommender systems for software," in *Companion Proceedings of the 36th International*, Hyderabad, India, 2014.
- [17] A. Spillner, T. Linz and H. Schaefer, *Software Testing Foundations: A Study Guide for the Certified Tester Exam*, Rocky Nook, 2007.
- [18] B. Hambling, P. Morgan, A. Samaroo, G. Thompson and P. Williams, *SOFTWARE TESTING*, British Informatics Society Limited, 2010.
- [19] K. NAIK and P. TRIPATHY, "CONTROL FLOW GRAPH," in *SOFTWARE TESTING AND QUALITY ASSURANCE Theory and Practice*, Hoboken, New Jersey, JOHNWILEY & SONS, 2008.
- [20] J. A. Harer, L. Kim, R. L. Russell, O. Ozdemir, O. Ozdemir, E. Antelman and S. Chin, "Automated software vulnerability detection with machine learning," in *ResearchGate*, 2018.
- [21] F. E. Allen, "Control flow analysis," in *ACM* , 1970.
- [22] A. V. Phan, M. L. Nguyen and L. T. B, "Convolutional Neural Networks over Control Flow Graphs for Software Defect Prediction," 2018.
- [23] B. Anderson, D. Quist, J. Neil, C. Storlie and T. Lane, "Graph-based malware detection using dynamic analysis," *Journal in computer virology*, p. 247–258, 2011.
- [24] D. Bruschi, L. Martignoni and M. Monga, "Detecting selfmutating malware using control-flow graph matching," *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, no. Springer, p. 129–143, 2006.

- [25] D.-K. Chae, J. Ha, S.-W. Kim, B. Kang and E. G. Im, "Software plagiarism detection: a graph-based approach," in *22nd ACM international conference on Conference on information & knowledge management*, 2013.
- [26] X. Sun, Y. Zhongyang, Z. Xin, B. Mao and L. Xie, "Detecting code reuse in android applications using component-based control flow graph," in *IFIP International Information Security Conference*, 2014.
- [27] J. Han, M. Kamber and J. Pei, *Data Mining Concepts and Techniques*, Elsevier, 2012.
- [28] J. B. Schafer, "The Application of Data-Mining to Recommender Systems," in *University of Northern Iowa*.
- [29] K. Tsipis and A. Chorianopoulos, *Data Mining Techniques in CRM*, United Kingdom: John Wiley and Sons, 2009.
- [30] X. Amatriain, A. Jaimes, N. Oliver and J. M. Pujol, "Data Mining Methods for Recommender Systems," *Recommender Systems Handbook*, no. Springer, Boston, MA, pp. 39-71, 2011.
- [31] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17 , no. 6, pp. 734 - 749, June 2005.
- [32] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *20th International Conference on Very Large Data Bases*, 1994.
- [33] T. Cover and P. Hart, "Nearest neighbor pattern classification," *Information Theory, IEEE Transactions*, vol. 13, no. 1, p. 21–27, 1967.
- [34] J. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, p. 81–106, March 1986.
- [35] L. Rokach and O. Maimon, "Data Mining with Decision Trees: Theory and Applications," in *World Scientific Publishing*, 2008.
- [36] J. Zurada, " Introduction to artificial neural systems," in *West Publishing Co.*, St. Paul, MN, USA, 1992.
- [37] H. Kaur, G. Singh and J. Minhas, "A Review of Machine Learning based Anomaly Detection Techniques," *International Journal of Computer Applications Technology and Research*, vol. 2, no. 2, pp. 185-187, 2013.

- [38] S. J. H. Ch and S. Bae, "Evolutionary Neural Networks for Anomaly," *IEEE Transaction on Systems, Man, and Cybernetic*, vol. 36, no. 3, 2005.
- [39] N. Friedman, D. Geiger and M. Goldszmidt, "Bayesian network classifiers," in *Machine Learning*, 1997.
- [40] P. G. Teodora, J. D. Verdejo, G. M. Farnandez and a. E. Vazquez, "Anomaly-based network intrusion detection: Techniques, Systems and Challenges," *Journal of Computers & Security*, vol. 28, no. 1, pp. 18-28, 2009.
- [41] N. Cristianini and a. J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press, March 2000.
- [42] J. Hartigan, *Clustering Algorithms (Probability & Mathematical Statistics)*, John Wiley & Sons Inc.
- [43] G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu and Z. Chen, "Scalable collaborative filtering using cluster-based smoothing," in *SIGIR*, 2005.
- [44] B. Frey and D. Dueck, "Clustering by passing messages between data points," in *Science*, 2007.
- [45] B. Sarwar, G. Karypis, J. Konstan and J. Reidl, "Item-based Collaborative Filtering Recommendation Algorithms," *Tenth International Conference on World Wide Web*, pp. 285 - 295, 2001.
- [46] W. Lin, S. A. Alvarez and C. Ruiz, "Efficient adaptive-support association rule mining for recommender systems," in *Data Mining and Knowledge Discovery*, 2002.
- [47] M. Géry and H. Haddad, "Evaluation of Web Usage Mining approaches for user's next request prediction," in *Fifth International Workshop on Web Information and Data Management*, 2003.
- [48] A. Geyer-Schulz and M. Hahsler, "Evaluation of Recommender Algorithms for an Internet Information Broker based on Simple Association Rules and on the Repeat-Buying Theory," in *Fourth WEBKDD Workshop: Web Mining for Usage Patterns & User Profiles*, 2002.
- [49] A. Michail, "Data mining library reuse patterns using generalized association rules," in *22Nd International Conference on Software Engineering*, New York, 2000.
- [50] Z. Li and Y. Zhou, "Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code," in *Proceedings of the 10th European Software*

*Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, 2005.

- [51] M. Bruch, T. Schafer and M. Mezini, "FrUiT : IDE support for framework understanding," in *Proceedings of the 2006 OOPSLA Workshop on Eclipse Technology eXchange*, New York, 2006.
- [52] K. Vijay and D. Bala, "Text Mining," in *Data Science- Concepts and Practice*, Elsevier In, 2019, pp. 281-305.
- [53] B. D. Vijay Kotu, "Recommendation Engines," in *Data Science - Concepts and Practice (Second Edition)*, Morgan Kaufmann, 2019, pp. 343-394.
- [54] A. Rajaraman and J. D. Ullman, "Data Mining," in *Mining of Massive Datasets*, New York, NY, United States, Cambridge University Press, 2011, p. 1–17.
- [55] R. Holmes and G. C. Murphy, "Using Structural Context to Recommend Source Code Examples," in *International Conference on Software Engineering (ICSE)*, 2005.
- [56] S. Proksch, S. Amann, S. Nadi and M. Mezini, "Identifying Requirements of Static Analyses for Recommendation Systems in Software Engineering".
- [57] M. K. N. Mahrin, A. H. Mohamed and M. Naz'ri, "A survey on data mining techniques in recommender systems," in *Springer-Verlag part of Springer Nature*, Germany, 2017.
- [58] F. D. Nembhard, M. M. Carvalho and T. C. Eskridge, "Towards the application of recommender systems to secure coding," in *EURASIP Journal on Information Security*, 2019.
- [59] L. Pickard, B. Kitchenham and a. S. Linkman, "An investigation of analysis techniques for software datasets," in *nternational Software Metrics Symposium*, 1999.
- [60] S. Shirabad and T. Menzies, "The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering," <http://promise.site.uottawa.ca/SERepository>, University of Ottawa, Canada, 2005.
- [61] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton and J. Noble, "Qualitas corpus: A curated collection of java code for empirical studies," in *Asia Pacific Software Engineering Conference*, 2010.
- [62] A. A. Sawant and A. Bacchelli, "A dataset for API usage," in *International Conference on Mining Software Repositories*, 2015.

- [63] R. Dyer, H. A. Nguyen, H. Rajan and T. N. Nguyen, "BOA: Ultra-Large-Scale Software Repository and Source-Code Mining.," *Transactions on Software Engineering and Methodology*, 2015.
- [64] S. Negara, M. Vakilian, N. Chen, R. E. Johnson and D. Dig, "Is it dangerous to use version control histories to study source code evolution?," in *European Conference on Object-Oriented Programming*, Springer, 2012.
- [65] R. Robbes and M. Lanza, "SpyWare: A Change-aware Development Toolset," in *International Conference on Software Engineering, ICSE*, NY, 2008.
- [66] J. Spacco, J. Strecker, D. Hovemeyer and W. Pugh, "Software Repository Mining with Marmoset: An Automated Programming Project Snapshot and Testing System," in *International Workshop on Mining Software Repositories*, 2005.
- [67] L. Ponzanelli, A. Mocci and M. Lanza, "StORMeD: Stack Overflow Ready Made Data," in *International Conference on Mining Software Repositories*, 2015.
- [68] M. Conklin, J. Howison and K. Crowston, "Collaboration Using OSSmole: A Repository of FLOSS Data and Analyses," in *International Workshop on Mining Software Repositories*, AMC, 2005.
- [69] G. Gousios, "The ghtorrent dataset and tool suite," in *Working Conference on Mining Software Repositories*, NY, 2013.
- [70] F. Mccarey, M. Ó. Cinnéide and a. N. K. Rascal, "A Recommender Agent for Agile Reuse," *Artificial Intelligence Review*, p. 253–276, 2005.
- [71] W. Snipes, A. R. Nair and E. Murphy-Hill, "Experiences Gamifying Developer Adoption of Practices and Tools," in *International Conference on Software Engineering*, 2014.
- [72] V. Singh, L. L. Pollock, W. Snipes and N. A. Kraft, "A case study of program comprehension effort and technical debt estimations," in *International Conference on Program Comprehension*, 2016.
- [73] R. Minelli, A. Mocci, R. Robbes and M. Lanza, "Taming the ide with fine-grained interaction data," in *International Conference on Program Comprehension*, 2016.
- [74] M. Bruch, M. Monperrus and M. Mezini, "Learning from Examples to Improve Code Completion Systems," in *International Symposium on the Foundations of Software*, 2009.
- [75] C. Zhang, J. Yang, Y. Zhang, J. Fan, X. Zhang, J. Zhao and P. Ou, "Automatic Parameter Recommendation for Practical API Usage," in *International Conference on Software Engineering, IEEE*, 2012.

- [76] L. Heinemann, V. Bauer, M. Herrmannsdoerfer and B. Hummel, "Identifier-based Context-dependent API Method Recommendation," in *European Conference on Software Maintenance and Reengineering, IEEE*, 2012.
- [77] S. Amann, S. Proksch and M. Mezini, "Method-call Recommendations from Implicit Developer Feedback," in *International Workshop on CrowdSourcing in Software Engineering*, 2014.
- [78] V. Raychev, M. Vechev and E. Yahav, "Code Completion with Statistical Language Models," in *Conference on Programming Language Design and Implementation*, 2014.
- [79] M. Asaduzzaman, C. K. Roy, K. A. Schneider and D. Hou, "CSCC: Simple, Efficient, Context Sensitive Code Completion," in *ICSME*, 2014.
- [80] A. Michail, "Data Mining Library Reuse Patterns in User-selected Applications," *International Conference on Automated Software Engineering. IEEE*, 1999.
- [81] H. Zhong, L. Zhang and H. Mei, "Inferring Specifications of Object-oriented APIs from API Source Code," *Asia-Pacific Software Engineering Conference*, 2008.
- [82] P. W. McBurney and C. McMillan, "Automatic Documentation Generation via Source Code Summarization of Method Context," *International Conference on Program Comprehension.*, 2014.
- [83] L. Ponzanelli, G. Bavota, M. D. Penta, R. Oliveto and M. Lanza, "Mining StackOverflow to Turn the IDE Into a Self-Confident Programming Prompter," *International Conference on Mining Software Repositories*, 2014.
- [84] M. Monperrus, M. Bruch and M. Mezini, "Detecting Missing Method Calls in Object Oriented Software," in *European Conference on Object-oriented Programming*, 2010.
- [85] Z. Li and Y. Zhou, "PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code," in *International Symposium on Foundations of Software Engineering*, 2005.
- [86] M. Pradel, C. Jaspan, J. Aldrich and T. R. Gross, "Statically Checking API Protocol Conformance with Mined Multi-object Specifications," in *International Conference on Software Engineering*, 2012.
- [87] M. Pradel, P. Bichsel and a. T. Gross, "A Framework for the Evaluation of Specification Miners Based on Finite State Machines," in *International Conference on Software Maintenance*, 2010.

- [88] T. Zimmermann, A. Zeller, P. Weissgerber and S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429 - 445, 2005.
- [89] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi and T. N. Nguyen, "Graph-based Mining of Multiple Object Usage Patterns," in *International Symposium on Foundations of Software Engineering*, 2009.
- [90] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, A. Tamrawi, H. V. Nguyen, J. A. Kofahi and T. N. Nguyen, "Graph-based Pattern-oriented, Context-sensitive Source Code Completion," in *International Conference on Software Engineering*, 2012.
- [91] D. Mandelin, L. Xu, R. Bodík and D. Kimelman, "Jungloid Mining: Helping to Navigate the API Jungle," in *Programming Language Design and Implementation*, 2005.
- [92] S. Thummalapenta and T. Xie, "Parseweb: A Programmer Assistant for Reusing Open Source Code on the Web," in *International Conference on Automated Software Engineering*, 2007.
- [93] H. Zhong, T. Xie, L. Zhang, J. Pei and H. Mei, "MAPO: Mining and Recommending API Usage Patterns," in *European Conference on Object-oriented Programming*, 2009.
- [94] R. Holmes, R. J. Walker and G. C. Murphy, "Approximate Structural Context Matching: An Approach to Recommend Relevant Examples.," in *Transactions on Software Engineering*, 2006.
- [95] M. P. Robillard, R. J. Walker and T. Zimmermann, "Development tools: Recommendation Systems for Software Engineering," *IEEE SOFTWARE* [www.computer.org/software](http://www.computer.org/software).
- [96] L. Moreno, G. Bavota, M. D. Penta, R. Oliveto and A. Marcus, "How Can I Use This Method?," in *International Conference on Software Engineering*, 2015.
- [97] O. Hummel, W. Janjic and C. Atkinson, "Code Conjurer: Pulling Reusable Software out of Thin Air," *IEEE*, 2008.
- [98] T. T. N. H. A. N. A. T. H. V. N. J. A. K. a. T. N. N. A. T. Nguyen, "Graph-based Pattern-oriented, Context-sensitive Source Code Completion," *International Conference on Software Engineering*, 2012.
- [99] TIOBE, "TIOBE Index for August 2020," 2020. [Online]. Available: <https://www.tiobe.com/tiobe-index/>.

- [100] S. García, J. Luengo and F. Herrera, *Data Preprocessing in Data Mining*, Springer International Publishing Switzerland, 2015.
- [101] K. Ottenstein and L. Ottenstein, "The program dependence graph in a software development environment," *ACM Sigplan Notices*, 1984.
- [102] J. Beel, B. Gipp, S. Langer and C. Breiting, "Research-paper recommender systems: a literature survey," *International Journal on Digital Libraries*, pp. 305-338, 2016.
- [103] B. Li and L. Han, "Distance Weighted Cosine Similarity Measure for Text Classification," in *International Conference on Intelligent Data Engineering and Automated Learning*, Berlin, Heidelberg, 2013.
- [104] B. Schafer, J. A. Konstan and J. Riedl, "E-Commerce Recommendation Applications," *Data Mining and Knowledge Discovery*, vol. 5, no. <https://doi.org/10.1023/A:1009804230409>, p. 115–153, 2001.
- [105] F. O. A. H. A. G. J. Bobadilla, "Recommender systems survey," *Knowledge-Based Systems*, vol. 46, pp. 109-132, 2013.
- [106] M. E. A. Ozgur Cakir, "A Recommendation Engine by Using Association Rules," *Procedia - Social and Behavioral Sciences*, vol. 62, pp. 452-456, 2012.
- [107] A. . S. Das, M. Datar, A. Garg and S. Rajaram, "Google news personalization: scalable online collaborative filtering," in *Proceedings of the 16th international conference on World Wide Web*, 2007.
- [108] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang and R. Zadeh, "WTF: the who to follow service at Twitter," in *Proceedings of the 22nd international conference on World Wide Web*, 2013.
- [109] K. Vijay and D. Bala, "Classification," in *Data Science (Second Edition)*, organ Kaufmann, 2019, pp. 65-163.
- [110] S. Harispe, S. Ranwez, S. Janaqi and J. Montmain, "Semantic similarity from natural language and ontology analysis," *Synthesis Lectures on Human Language Technologies*, vol. 8, no. 1, pp. 1-254, 2015.
- [111] D. Lin, "An Information-Theoretic Definition of Similarity," *Icml*, vol. 98, pp. 296-304, 1998.
- [112] Z. Wu and M. Palmer, "Verb semantics and lexical selection," *arXiv preprint cmp-lg/940603*, 1994.



- [113] Y. Chen, E. K. Garcia, M. R. Gupta, A. Rahimi and L. Cazzanti, "Similarity-based Classification: Concepts and Algorithms," *Journal of Machine Learning Research*, vol. 10, pp. 747-776, 2009.
- [114] R. Mistry and S. Misner, *Introducing Microsoft® SQL Server 2012*, Microsoft Press, 2012.
- [115] Microsoft, "Semantic Search," Microsoft, 17 08 2020. [Online]. Available: <https://docs.microsoft.com/en-us/sql/relational-databases/search/semantic-search-sql-server?view=sql-server-ver15#:~:text=Semantic%20search%20builds%20upon%20the,the%20meaning%20of%20the%20document..> [Accessed 24 01 2022].
- [116] E. M.-H. a. G. C. Murphy, "Recommendation Delivery, Getting the User Interface Just Right," in *Recommendation Systems in Software Engineering*, Springer, 2014, pp. 223-239.
- [117] E. Murphy-Hill and A. P. Black, "An interactive ambient visualization for code smells," in *Proceedings of the 5th international symposium on Software visualization*, 2010.
- [118] R. Sinha and K. Swearingen, "The role of transparency in recommender systems," in *CHI EA '02*, 2002.
- [119] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon and J. Riedl, "Applying Collaborative Filtering to Usenet News," *Communications of the ACM*, vol. 40, no. 3, pp. 77-87, 1997.
- [120] P. Viriyakattiyaporn and G. C. Murphy, "Improving program navigation with an active help system," in *Conference of the Center for Advanced Studies on Collaborative Research*, 2010, 2010.
- [121] X. Ge, Q. L. DuBose and E. Murphy-Hill, "Reconciling manual and automatic refactoring," in *34th International Conference on Software Engineering (ICSE)*, Zurich, 2012.
- [122] M. Bruch, T. Schäfer and M. Mezini, "On Evaluating Recommender Systems for API Usages," in *Proceedings of the 2008 international workshop on Recommendation systems for software engineering*, 2008.
- [123] C. J. V. Rijsbergen, "A non-classical logic for information retrieval," *The computer journal*, vol. 29, no. 6, pp. 481-485, 1986.
- [124] K. H. Yim, F. S. Nahm, K. A. Han and S. Y. Park, "Analysis of Statistical Methods and Errors in the Articles Published in the Korean Journal of Pain," *Korean Journal of Pain*, vol. 23, no. 1, p. 35-41, 2010.

- [125] T. Gvero, V. Kuncak, I. Kuraj and R. Piskac, "Complete Completion Using Types and Weights," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, New York, 2013.
- [126] A. Hindle , E. T. Barr, Z. Su, M. Gabe and P. Devanbu, "On the Naturalness of Software," in *Proceedings of the 2012 International Conference on Software Engineering*, NJ, 2012.
- [127] H. Kaur, G. Singh and J. Minhas, "A Review of Machine Learning based Anomaly," *International Journal of Computer Applications Technology and Research*, pp. 185 - 187, 2013.
- [128] V. Jyothsna, V. V. R. Prasad and K. M. Prasad, "A review of anomaly based intrusion detection systems," *International Journal of Computer Applications*, pp. 26-35, 2011.
- [129] K. Lewin, "Action research and minority problems," *Journal of social issues* , pp. 34-46, 1946.
- [130] S. Rose, N. Spinks and A. I. Canhoto, *Management Research*, Abingdon, New York: Routledge, 2015.