Mineração de Soluções Relevantes para Tarefas de Programação a partir de Resultados de Mecanismos de Busca

Adriano Mendonça Rocha

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2022

Adriano Mendonça Rocha

Mineração de Soluções Relevantes para Tarefas de Programação a partir de Resultados de Mecanismos de Busca

Tese de doutorado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Marcelo de Almeida Maia

Uberlândia
2022

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

R672 2022

Rocha, Adriano Mendonça, 1986-

Mineração de soluções relevantes para tarefas de programação a partir de resultados de mecanismos de busca [recurso eletrônico] / Adriano Mendonça Rocha. - 2022.

Orientador: Marcelo de Almeida Maia.
Tese (Doutorado) - Universidade Federal de Uberlândia, Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.te.2022.565
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Maia, Marcelo de Almeida ,1969-, (Orient.). II. Universidade Federal de Uberlândia. Pós-graduação em Ciência da Computação. III. Título.

CDU: 681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:
Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Tese de Doutorado 17/2022, PPGCO				
Data:	29 de setembro de 2022	Hora de início:	14:00	Hora de encerramento:	17:30
Matrícula do Discente:	11623CCP003				
Nome do Discente:	Adriano Mendonça Rocha				
Título do Trabalho:	Mineração de Soluções Relevantes para Tarefas de Programação a partir de Resultados de Mecanismos de Busca				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Engenharia de Software				
Projeto de Pesquisa de vinculação:	-				

Reuniu-se, por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Prof. Dr. Uirá Kulesza - UFRN; Prof. Dr. Eduardo Magno Lages Figueiredo - UFMG; Prof. Dr. Fabiano Azevedo Dorça - FACOM/UFU; Prof. Dr. Flávio de Oliveira Silva - FACOM/UFU e Prof. Dr. Marcelo de Almeida Maia, orientador do candidato.

Os examinadores participaram desde as seguintes localidades: Uirá Kulesza - Natal/RN; Eduardo Magno Lages Figueiredo - Belo Horizonte/MG; Fabiano Azevedo Dorça, Flávio de Oliveira Silva e Marcelo de Almeida Maia - Uberlândia/MG. O discente participou da cidade de Uberlândia/MG.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Marcelo de Almeida Maia , apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

Aprovado

Esta defesa faz parte dos requisitos necessários à obtenção do título de Doutor.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Este trabalho é dedicado à todos os
que me ajudaram ao longo desta caminhada.

Agradecimentos

Primeiramente agradeço ao meu orientador, Professor Doutor Marcelo de Almeida Maia, por ter sempre me orientado de forma a me despertar interesse pela área pesquisada. Além de me transmitir o conhecimento necessário para a realização do presente estudo, sempre disposto a tirar as dúvidas relacionadas à pesquisa.

À minha família pela força, apoio e compreensão que depositou em mim durante esta jornada.

À todas pessoas que de alguma forma contribuíram com o presente estudo.

“Duas coisas instruem o homem, qualquer que seja a sua natureza:
o instinto e a experiência.”
(Blaise Pascal)

Resumo

O desenvolvimento de software é uma atividade intensiva em conhecimento. Documentações oficiais para desenvolvedores podem não ser suficientes para todas as suas necessidades. A busca de informações na *Web* é uma prática usual, mas encontrar informações realmente úteis pode ser um desafio, pois nem sempre as melhores soluções estão entre as primeiras páginas ranqueadas. Assim, os desenvolvedores têm que ler e descartar páginas irrelevantes, ou seja, páginas que não possuem exemplos de código ou que possuem conteúdo não focado na solução desejada. Este trabalho tem como objetivo entender como a qualidade do *ranking* retornado por mecanismos de busca pode influenciar o desempenho de desenvolvedores durante a resolução de tarefas de programação, e propor uma abordagem para minerar soluções relevantes para tarefas de programação a partir de resultados de buscas. Em uma análise preliminar, avaliamos as 20 principais páginas retornadas pelo mecanismo de pesquisa do Google, para 10 consultas diferentes, e observamos que apenas 31% das páginas avaliadas são relevantes para desenvolvedores. Diante disso, realizamos um primeiro estudo com desenvolvedores que mostrou que estes gastaram menos tempo durante a resolução de tarefas de programação, ao utilizarem um *ranking* de qualidade superior (em média, os desenvolvedores gastaram por volta de 4 minutos a mais, durante a resolução das tarefas, ao utilizarem um *ranking* de qualidade inferior). Em um segundo estudo, propusemos e avaliamos três abordagens diferentes para minerar páginas relevantes retornadas pelo mecanismo de busca. Os filtros propostos se mostraram eficazes na remoção de páginas irrelevantes, e desta forma, melhoraram a qualidade do *ranking*. Concluímos que desenvolvedores podem se beneficiar desses filtros, de maneira a aumentar sua produtividade durante realizações de tarefas de programação.

Palavras-chave: Tarefas de Programação. Mineração de Soluções Relevantes. Motores de Busca.

Mining Relevant Solutions for Programming Tasks from Search Engine Results

Adriano Mendonça Rocha

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2022

UNIVERSIDADE FEDERAL DE UBERLÂNDIA – UFU
FACULDADE DE COMPUTAÇÃO – FACOM
PROGRAMA DE PÃS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO – PPGCO

The undersigned hereby certify they have read and recommend to the PPGCO for acceptance the thesis entitled **"Mining Relevant Solutions for Programming Tasks from Search Engine Results"** submitted by **"Adriano Mendonça Rocha"** as part of the requirements for obtaining the **Doctoral degree in Computer Science**.

Uberlândia, ___ de ________ de _____

Supervisor: ______________________________
Prof. Dr. Marcelo de Almeida Maia
Universidade Federal de Uberlândia

Examining Committee Members:

____
Prof. Dr. Eduardo Figueiredo
Universidade Federal de Minas Gerais

____
Prof. Dr. Uirá Kulesza
Universidade Federal do Rio Grande do Norte

____
Prof. Dr. Fabiano Azevedo Dorça
Universidade Federal de Uberlândia

____
Prof. Dr. Flávio de Oliveira Silva
Universidade Federal de Uberlândia

Abstract

Software development is a knowledge-intensive activity. Official documentation for developers may not be sufficient for all developer needs. Searching for information on the web is a usual practice, but finding really useful information may be challenging, because the best solutions are not always among the first ranked pages. So, developers have to read and discard irrelevant pages, that is, pages that do not have code examples or that have content not focused on the desired solution. This work aims to understand how the ranking quality returned by search engines can influence the performance of developers when solving programming tasks, and to propose an approach to mine relevant solutions for programming tasks from search engines results. In a preliminary analysis, we evaluated the top 20 pages returned by Google's search engine, for 10 different queries, and observed that only 31% of the pages evaluated are relevant to developers. Therefore, we carried out a first study with developers that showed that they spent less time solving programming tasks, when using a higher quality ranking (on average, developers spent around 4 minutes more during solving tasks, when using a lower quality ranking). In a second study, we proposed and evaluated three different approaches to mine relevant pages returned by the search engine. The Google search engine that was used as a baseline returns a fair number of pages that are irrelevant to developers. The filters we propose are effective in removing irrelevant pages, in this way, a better quality ranking is generated. Developers can benefit from these filters in order to increase their productivity while performing programming tasks.

Keywords: Programming Tasks. Mining Relevant Solutions. Search Engines.

List of Figures

List of Tables

Acronyms list

Contents

Chapter 1

Introduction

Software development is a knowledge-intensive task (MURPHY et al., 2019), where the search for information is a constant challenge. A survey conducted by Stack Overflow in 2022, which asked how much time developers typically spend looking for answers or solutions to problems encountered at work, reveals that 62% of developers spend more than 30 minutes a day searching for answers or solutions to problems, and 25% spending more than an hour each day. For a team of 50 developers, the amount of time spent searching for answers/solutions adds up to between 333-651 hours of time lost per week across the entire team. A total of 36,198 developers participated in the survey[1]. Programmers search for code very frequently, conducting an average of five search sessions with 12 total queries each workday (SADOWSKI; STOLEE; ELBAUM, 2015). These data show the importance of efficient means for developers to obtain information to perform their functions.

The Web is a medium used by software developers to obtain relevant information related to software development, which offers a vast variety of information sources, such as, tutorials, blogs, question and answer (Q&A) services. A challenge in searching for information on the Web is the large amount of publicly available information (BAJRACHARYA; OSSHER; LOPES, 2014). Paradoxically, the overload of available content burdens developers, who must search, screen and analyze content from several web pages until they find those that are really relevant for the particular task (Sharma et al., 2010; Kataria; Sapra, 2016).

Among the various sources of technical content, Stack Overflow is a Q&A service specific for software developers, with a large volume of content, that also challenges the search for relevant content (SILVA et al., 2019; SILVA et al., 2020). Moreover, Stack Overflow does not cover all the API functions of a chosen programming technology, and the best solutions for a programming task may also not be always available on Stack Overflow (DELFIM et al., 2016).

Search engines are one the most-used services on the Web (AGRAHRI; MANICKAM;

[1] https://survey.stackoverflow.co/2022/

RIEDL, 2008). Developers also use general purpose search engines, such as Google or Bing, to find technical information for their task in hand (HORA, 2021b; NIU; KEIVANLOO; ZOU, 2017; SIM et al., 2011; RAHMAN et al., 2018). The use of search engines to find solutions related to software development has the inherent problem that the best solutions are not necessarily always among the best ranked pages (CHATTERJEE; JUVEKAR; SEN, 2009; HORA, 2021a). The top ranked results for such queries may not contain relevant documents to the user's search intent (ZHUANG; CUCERZAN, 2006). Problems that typically appear among the first returned pages from a search query are lack of code examples or little relevance for the task in hand, for example, pages with content that is not very focused on the query performed by the developer. Generally, people are biased in clicking on those items that are presented at the top of the result list, returned by the search engine (KEANE; O'BRIEN; SMYTH, 2008). When it comes to software development, when clicking on the first pages and these are not relevant to developers, they possibly can waste time looking for an inadequate solution for a task, hindering their performance.

Another problem is that search engines, in some cases, may have a negative behaviour of avoiding fresh web pages. This behaviour may be explained because one of the weighting factors used by these mechanisms is the popularity of the pages, that is, popular pages tend to stay at the top of the ranking. Completing the cycle, users click more frequently on these pages, further increasing their popularity. Fresh pages tends to be not popular because they are new, and thus may be returned in lower positions of the ranking. The negative cycle tends to persist because few users click on and/or create links to them, keeping them in lower ranking positions, even if such pages have high quality content (CHO; ROY, 2004; ZHUANG; CUCERZAN, 2006). In software development, developers would need to screen lower quality content in best ranked pages until finding relevant solutions, which is a task that requires time and effort (XIA et al., 2017). This phenomenon justifies the importance of approaches that search pages for the relevant content and not just for their popularity.

The aim to improve search engine results has been a very active research area along the past decades. Many approaches focus on specific solutions to mine specific content from the general query results. For instance, Saraswathi e Vijaya (2013), Kim, Collins-Thompson e Teevan (2016) propose to detect link spam in the results. Zahera, El-Hady e El-Wahed (2011) use crowd information to improve the search experience. Zhuang e Cucerzan (2006) propose clustering query results to facilitate classifying the pages. Amin e Emrouznejad (2011) propose a meta search engine approach combining the results of different search engines. Caramia, Felici e Pezzoli (2004) proposes a mining approach query results based on a thematic databases of web pages. Zhao et al. (2017) proposes the identification of major contents using topic modeling on query results to facilitate screening the returned pages. Hong, Vaidya e Lu (2011), Lau e Horvitz (1999), Rahman, Roy e Lo (2019) propose approaches to help query recommendation or reformulation to help users finding more easily the desired content. In this thesis, as in those previous work, we follow the approach of using search engine results as initial input, and mine the results to improve the developer experience when querying for their problem in hand. Below we show the main contributions of this thesis in relation to related work:

- Development of a filter that removes outliers pages returned by search engines. Outlier pages are those that have a very high or very low number of method calls, in relation to the average number of method calls in the first N pages returned by the search engine.
- Development of a filter that uses a clustering algorithm, having as attribute the occurrences of methods present in the pages returned by the search engine, in order to remove irrelevant pages for software developers.
- Development of an approach that combines the two filters mentioned above, in order to improve the ranking quality of pages returned by search engines.

The two filters and the approach that we are proposing are a novelty in the area of Software Engineering, since in related work no such filters and approach are available to filter relevant documentation for developers in generic web pages. As the results of this thesis show, the two proposed filters are effective in removing irrelevant pages for software developers and the approach improves the ranking quality of pages returned by search engines.

1.1 Motivation

We present a motivating example to illustrate the problem of how low quality content in search hinders developers when looking for development solutions.

Consider a scenario where a developer wants to learn how to save data from an Android application to a web server, using the JAVA programming language and the Android API. The developer would use, for instance, the following search query, which seems to be simple and with sufficient details: "*how to save data from application to web server Java Android*". Inspecting the results, we observed that the top-1 page has no source code, but only a question posted by a user on how to save data from an Android application to a remote server. So after reading this page, the developer would need to check the next ones. In the top-2, there is also no source code, but only a text with an overview of the data and file storage on Android. The top-3 presents an explanation and source code of how to save data locally in an Android application. However, the solution on the page addresses a different function than the one desired by the developer, and the developer would still need to continue reading the next pages. The top-4 page provides content related to how to store data in Google Drive. Note that the content on this page also addresses a different solution than the developer researched. The top-5 page provides explanation and source code on how to download files in Android. Again, the content in this page does not address the solution desired by the developer. In the top-6 page, there is no source code, just an overview of the FCM (Firebase Cloud Messaging) architecture, which is a technology that offers messaging capabilities for different platforms. The top-7 page is a website of a software development company, where there is no content related to the search query, only the company's portfolio. The top-8 page provides a solution on how to save data on the remote server, but the solution lacks focus because it also explores how to receive notifications when the stored data is changed, which was not the original goal in the query. The top-9 presents a solution on how to store data in a SD card, which is again a different solution than what the developer looks for through the searched query. The top-10 provides a solution explaining how to save data to a file on Android, therefore, the solution addresses a function different from the one desired by the user. Finally, only in the top-11, the developer finds a relevant solution with a high degree of focus. The adequate page found only in the 11th position illustrates a scenario where the developer may need to read and analyze the content of several pages that are not related to the performed query, thus wasting time and effort.

Regarding this motivating example, one would question if the ranked results are not satisfactory because of an ill-formulated query. Although, we agree that a different query would produce a different ranking, the query proposed in the example seems to be a possible one that developers would launch. Moreover, small changes in the query formulated by the developer do not affect much the results returned by the search engine (HORA, 2021b). Alternative approaches for automatic reformulating queries may produce a list of pages with more adequate semantic matching with the query, but still does not address some issues such as lack of focus in returned pages. So, we envision that approaches that mine for relevant content in whatever list of returned pages, independently of the query quality, may play an important role for improving the developer performance.

1.2 Goals

In order to better understand the aforementioned problems, and find a solution for them, we propose two studies in this thesis.

The first study aims to understand how the quality of the ranking returned by these engines can influence developers, positively or negatively, during the development of programming tasks. To achieve this goal, we recruited undergraduate and graduate students to participate in a qualitative study, where they developed programming tasks. The tasks were organized in pairs with similar degree of difficulty. We aim at evaluating the performance of the participants by keeping the level of difficulty constant and setting the ranking quality as the treatment variable.

The second study proposes an approach that is based on filtering pages considering the relevance of the content itself and not its popularity. Figure 1 presents an overview of the approach. First, a query related to the programming task is elaborated and given as input to a search engine (Google is used as choice service). The n returned pages returned are automatically filtered in the potential relevant content. We consider the relevant solutions those that have code examples and focus on the specific searched problem, because generally developers look for solutions that contain code examples for programming tasks consulted in search engines. Relevant solutions for developers should be focused on the queried programming task, since solutions with little focus may not satisfy the developer, due to the lack of content that solves the programming task or the excess of content that is not related to it, that is, content that implements other functionalities that do not belong to the programming task (CHATTERJEE; JUVEKAR; SEN, 2009; HORA, 2021a).

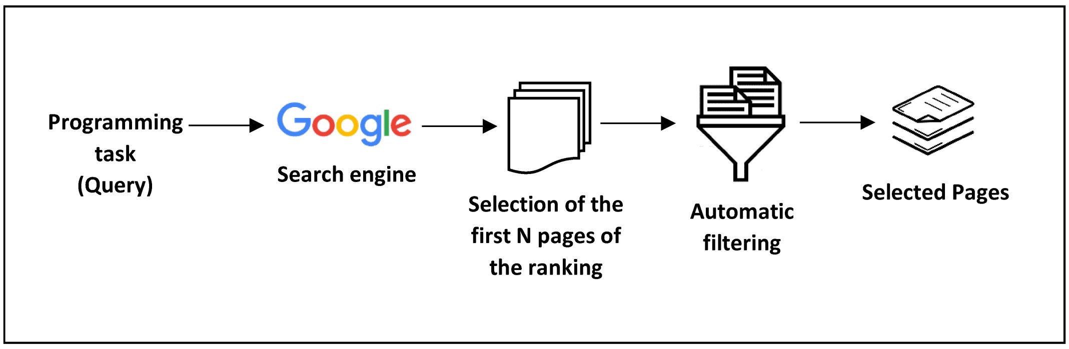

Figure 1 – Main steps of the proposed approach.

Summing up, this work has the **general goal** to understand how the ranking quality returned by the search engine can influence the performance of developers when solving programming tasks, and to develop and evaluate filters capable of removing irrelevant pages for software developers, contained in the results returned by search engines, in order to obtain a better ranking.

Below we show the **specific goals** of the two main studies carried out in this work. First, the specific goals of the study on the influence that the ranking quality exerts on the performance of developers when solving programming tasks are:

- ❏ Understand if the addition of a list of methods that most occur on pages in the pages returned by the search engine exerts any influence on developers when performing programming tasks. As search engines are of general use, when they are used to search for pages related to software development, these engines usually return pages without source code examples or pages with content that is not focused on the search performed by the developer. With that in mind, we propose to include a list of the methods that most occur on the page in the region below the descriptions of the pages returned by the search engine.

❏ Understand how irrelevant pages present in lower quality ranking can undermine the goal of developers to find solutions to problems related to software development. Irrelevant pages are those that do not contribute to the solution sought by the developer. Pages with no focus on the query searched by the user in search engines are examples of irrelevant pages. Irrelevant pages can harm developers while performing programming tasks, since these developers can spend considerable time analyzing and studying the content of these pages, or even coding, to end up not having the intended solution.

The specific objectives of the study on the relevance of pages returned by the Google's search engine and the proposed approaches to remove irrelevant pages returned by search engines are presented below:

❏ Propose a ground-truth and analyze the relevance of pages returned by the Google search engine to software developers, given as input queries related to programming tasks. As explained above, search engines are not designed exclusively to search for content related to software development, when these engines are used for this purpose, they usually return pages without source code examples or pages with content not focused on the search performed by the developer. Considering this issue, we propose to carry out a study on the pages returned by the Google search engine, in order to analyze the relevance of the pages returned to software developers.

❏ Develop and evaluate a filter that removes outliers pages returned by search engines. Outlier pages are those that have a very high or very low number of method calls, in relation to the average number of method calls in the first N pages returned by the search engine. Generally, pages with few method calls in their solutions tend to not have code examples that solve the problem queried by the developer, and pages with a large number of method calls tend have complex and extensive code examples.

❏ Develop and evaluate a filter that uses a clustering algorithm, having as an attribute the total occurrences of methods present in the pages returned by the search engine, in order to remove irrelevant pages for software developers. To obtain pages that possibly have solutions focused on the user query, we propose to apply a clustering algorithm using a proxy metric based on the sharing of common method calls between the pages.

❏ Develop and evaluate a filter that uses a clustering algorithm, having as an attribute the unique occurrences of methods present in the pages returned by the search engine, in order to remove irrelevant pages for software developers.

❑ Analyze whether the application of the proposed filters can improve the ranking quality returned by the search engines. We hope that once irrelevant pages are removed from the ranking returned by the Google search engine, a better quality ranking is generated. As filters can also remove relevant pages, this analysis is important to verify if, in the end, it is worth applying them to the results returned by the search engine.

1.3 Research Questions

We formulated the following research questions related to the study on the influence that the ranking quality exerts on the performance of developers when solving programming tasks:

RQ1: *Does the ranking quality of query results have any influence on the performance of developers in the development of programming tasks?* This research question aims to show the influence that the ranking quality exerts on the performance of developers during the development of programming tasks. Our hypothesis is that lower quality rankings make developers spending more time trying to solve the proposed programming task, since there are more pages (irrelevant pages) in top positions of the rank that do not contribute to solving the proposed programming task.

RQ2: *Is there any influence by adding a list of frequent methods in the description of the pages in the result returned by the search engine for developers?* This research question aims to show the influence that the addition of information about the most frequent methods occurred in the returned pages exerts on the developers, during the analysis of the page descriptions. Our hypothesis is that the list helps developers, since developers can use the names of the methods present in the list to implement the desired solution or use the information from the list of methods to decide whether or not to enter the page. For example, in cases where the method list is empty, there is no source code related to methods, so the developer looking such code may decide not to enter the page.

RQ3: *How do irrelevant pages present in lower quality rankings hinder developers' goal of finding relevant solutions?* This research question aims to show how irrelevant pages present in lower quality rankings hinder developers to find solutions to problems related to proposed programming tasks. Our hypothesis is that when developers find bad code on a irrelevant page, they spend a lot of time trying to reuse such code. They may try to fix the bad code by searching for content on other pages, spending even more time. In the worst case, when the code cannot be fixed, all the effort is wasted, and developers must start searching for the solution on another page.

We formulated the following research questions related to the study on the relevance of pages returned by the Google's search engine and the proposed approaches to remove irrelevant pages returned by search engines:

RQ4: *To what extent does Google's search engine return relevant pages to software developers?* This research question is intended to show the relevance of the pages returned by the search engine to software developers. Since this mechanism is general-purpose, it is not focused on software development. Our hypothesis is that many pages that are irrelevant to software developers are returned by the Google search engine.

RQ5: *To what extent applying outlier pages removal filter w.r.t. the page size help to remove irrelevant pages returned by search engines?* This research question aims to investigate if the proposed filter removes irrelevant outlier pages returned by search engines. Pages with few method calls compared to the average of method calls of the first N pages returned by the search engine, generally, seems not to implement something functional for developers. Pages with many method calls in relation to the average usually seems to implement functionality beyond those related to the query performed by the developer in the search engine, losing focus on the main solution. Our hypothesis is that the proposed filter would help to remove these pages, improving the ranking quality.

RQ6: *To what extent applying clustering algorithms w.r.t. the total method calls in returned pages help to remove irrelevant pages returned by search engines?* This research question aims to show that the proposed filter that uses the clustering algorithm, having as attribute the total occurrences of each method call, removes irrelevant pages returned by search engines. The idea of such clustering is to aggregate pages that share the same method calls in the same cluster. Our hypothesis is that pages that share common method calls and that have a small variety of other different method calls (which do not appear on other pages) would have solutions that are more focused on the developer's query.

RQ7: *To what extent applying clustering algorithms w.r.t. unique occurrences of method calls help to remove irrelevant pages returned by search engines?* This research question aims to show that the proposed filter that uses the clustering algorithm, having as attributes unique occurrences of method calls, removes irrelevant pages returned by search engines. The idea for applying clustering is similar to the previous one, however, now the hypothesis is that the number of occurrences of the method calls would make no difference, but only if the method occurs or not.

RQ8: *To what extent applying the filters proposed in this work improve the ranking quality of pages returned by search engines?* This research question aims to show that the application of the filters proposed in this work improves the ranking quality returned by search engines. Our hypothesis is that the pages removed by the proposed filters improve the ranking quality.

1.4 Thesis Organization

This thesis is organized as follows. This chapter has presented the introduction, showing a motivating example to illustrate how the problem of low quality of web pages returned by search engines harms the performance of developers. This chapter also introduced the two main studies developed in this thesis. The first study aims to investigate the influence that ranking of query results has on the performance of developers when performing programming tasks, to demonstrate the importance of the proposed filters for the generation of a better quality ranking for software developers. The second study is about the mining of pages in the results returned by search engines, which aims to evaluate the proposed approaches to filter pages relevant to software developers. The goals and research questions of this thesis have also been shown.

Chapter 2 presents the main concepts related to this thesis, such as search engines, grounded theory and clustering algorithm. Search engines are important in this work, as the proposed approach aims to improve the results returned by these mechanisms. We used grounded theory as inspiration to extract the findings from the study in Chapter 3, where the main goal is to understand the influence that a higher quality ranking and a lower quality ranking have on the performance of developers when performing programming tasks. Clustering algorithm is used by the proposed approach to group web pages that have a set of method calls in common in their solutions, in clusters.

Chapter 3 presents the study on the influence that the ranking quality of query results exerts on the performance of software developers during the resolution of programming tasks. The main objective of this study is to understand how the ranking quality influences the performance of developers when solving programming tasks. We also verified whether the addition of information about the methods that occur most frequently on the pages present in the ranking can influence the developers, when they are performing programming tasks. We also aim to understand how irrelevant pages present in lower quality ranking can hinder developers' goal of finding solutions to problems related to software development.

Chapter 4 presents the study on mining the results returned by the search engine. The main objective of this study is to develop a filtering approach capable of removing irrelevant pages for software developers present in the results returned by search engines. We evaluate the proposed variations of the filtering approach to assess their effectiveness for improving the ranking quality returned by these search engines.

Chapter 5 presents previous works related to the two main studies developed in this thesis. Finally, Chapter 6 presents the conclusion and future work.

Chapter 2

Background

This chapter aims to present the main concepts related to the thesis, such as search engines, grounded theory and clustering algorithm. Search engines are central components of this work because we claim that their results have irrelevant pages that possibly hinder developer performance, and thus a solution to improve the results returned by these mechanisms are needed. We used grounded theory as inspiration to extract the findings from the study in Chapter 3, where the main goal is to understand the influence that a higher quality ranking and a lower quality ranking have on the performance of developers when performing programming tasks. A clustering algorithm is used by the proposed approach to group web pages that have in common a set of method calls in their solutions.

2.1 Search Engines

A search engine is software created to make it easier to find pages on the web. Users enter textual information in search engines, and these search for Web pages, in a systematic way, and return ranked pages, as results. These results are known as: search engine results pages (SERPs). Various types of information can be returned by search engines, such as: links to web pages, videos, images, articles, among others.

Web Queries are formed by words or a set of words that users enter in the search bar of search engines. The search bar is well located on all major search engines such as: Google, Yahoo, Bing, etc. Users indicate the pages they want to obtain based on the keywords entered in the search box of search engines.

Search Engine Results Pages (SERP) are the pages returned and displayed by search engines in response to a query from a user. We have two types of results: results retrieved by the search engine algorithm (organic search) and sponsored results (advertisements).

Results are ranked by relevance. Typically, each result consists of a title, a link to the actual web page, and a brief description of the page. For sponsored results, advertisers choose what they want to show as a result.

Ranking is a document classification problem. Where given a query Q and a collection D of documents related to that query, after sorting through criteria, the best results should appear at the top of the results list (top of the rank).

Search engines are used by software developers for different purposes, such as: searching for reusable code examples, learning new concepts related to programming, searching for information about APIs, searching for information related to bugs in the source code, among others (SADOWSKI; STOLEE; ELBAUM, 2015; XIA et al., 2017). There are several general purpose search engines, the most used by software developers is Google (SIM et al., 2011).

These search engines are optimized for textual content search and treat source codes as plain text, that is, these engines tend to ignore the semantics of source codes (RAHMAN et al., 2018). Because the criteria used by these mechanisms were not implemented exclusively for content related to software development. For example, the Google search engine, when it was created, prioritized the pages that were popular, that is, pages that were highly referenced by other pages, were considered as relevant, so they were placed at the top of the ranking. A problem with this approach is that it does not take into account the content of the page, that is, pages with relevant solutions can be poorly ranked, since the approach prioritizes the frequency of references to the page. Another issue is that pages related to official API documentation are heavily referenced by other pages, so these pages are prioritized by Google's search engine, even if they only contain a brief description of technical elements (classes, methods, etc.) API, without examples of their use. While relevant pages with an example of the use of technical elements are not prioritized, staying in lower positions in the ranking. Another problem is that popular pages tend to stay at the top of the ranking and users click more frequently on these pages, further increasing their popularity. Fresh pages tend to be not popular because they are new, and thus may be returned in lower positions of the ranking. The negative cycle tends to persist because few users click on and/or create links to them, keeping them in lower ranking positions, even if such pages have high quality content (CHO; ROY, 2004; ZHUANG; CUCERZAN, 2006).

Google's algorithm is always undergoing updates to improve the quality of page ranking, but even so, this engine is still not optimized for content related to software development. For example, the search algorithm currently examines several factors, such as the query words, the relevance and usability of the pages, the specialty of the fonts, and the user's location and settings[1]. We can observe that these criteria are not strongly related to content related to software development. In this way, when a developer looks for solutions in these mechanisms, irrelevant pages are usually returned, such as: pages without code examples and pages with content not focused on the query.

In view of the limitation of search engines mentioned above, in this work we propose

[1] https://www.google.com/search/howsearchworks/how-search-works/ranking-results/

filters to improve the ranking quality returned by the Google search engine for content related to software development.

2.2 Grounded Theory

Grounded theory is a systematic methodology that has been used by many social scientists in qualitative research, by collecting, analyzing data, and applying inductive reasoning to create hypotheses and theories.

Studies that apply grounded theory usually begin with the collection of qualitative data. Through the collected data, scientists create concepts that can be revised as new data are collected. The created concepts are named through codes. As data are collected and revised, new code can be created and grouped into higher-level concepts. Subsequently, these concepts are grouped into categories. From these categories, a hypothesis or a new theory is created.

The main stages of the grounded theory are presented below:

1. **Open Code** are sentence (phrase) encodings from collected data.

2. **Concepts** are groupings of codes generated in the previous step, with the objective of producing a higher level of abstraction compared to the codes.

3. **Categories** are groupings of concepts generated in the previous step, in order to create a higher level of abstraction compared to the concepts.

4. **Core category** is considered the main theme or problem for the participants. It should be central and related to several other categories.

5. **Selective Coding.** Once the core category is established, the researcher stops open coding and moves on to selective coding, which is a process that involves coding focused on that core category, delimiting the coding only to those codes that relate to the core category, in order to produce a hypothesis or a theory.

6. **Memoing** is the ongoing process of writing theoretical memos throughout the Grounded Theory process. The memos are theoretical notes on the data and the conceptual connections between the categories. They must be written as new ideas about codes and their relationships emerge.

7. **Sorting** is the process of classifying the memos to form a theoretical outline. At this stage, the researcher must classify the ideas and not the data. A recommendation is to avoid sorting memos in chronological order, but to sort memos by topic so that related topics can be sorted one after the other. A theory outline can be generated using the topic names in the same order. This sketch can form the theory sketch.

8. **Theoretical Coding** is the process that involves conceptualizing how categories relate to one another.

9. **Write-up** is the final stage of Grounded Theory, which consists of writing the theory, which follows the theoretical schema generated as a result of sorting and theoretical coding.

One such problem is the question of how deeply and broadly the researcher should familiarize himself with the research topic before empirical study. Problems also include the need to focus on the research problem and choose the sampling method. Data analysis is a multi-step process that requires the researcher's sensitivity and time to elaborate the findings that emerge from the data (BACKMAN; KYNGAS, 1999). Another problem is the large amount of data generated by the Grounded Theory methodology, which makes it difficult for the researcher to manage.

An advantage of the Grounded Theory methodology is that the researcher does not need to predetermine *a priori* what would be found, and how the phenomena should be viewed. So, the value of Grounded Theory is that it avoids making assumptions, and instead takes a more neutral view of observed phenomena (SIMMONS, 2006). Another advantage of Grounded Theory is that as the method is exploratory, this makes this methodology suitable for investigating social processes that were not very attractive in previous research. A disadvantage of Grounded Theory is the large amount of data produced during the application of the approach, generally managing this large amount of data is a difficult task. Another disadvantage of Grounded Theory is the difficulty in presenting research findings.

In this work, we partially use Grounded Theory as inspiration to extract and categorize the findings from recorded videos of developers, during the resolution of proposed programming tasks, as shown in Chapter 3. We encoded the trajectory that the subjects followed to solve the tasks, through analysis of the recorded videos. To overcome the disadvantage related to the large amount of data generated by the Grounded Theory methodology, we will organize the generated data into files. Each participant will have a folder and the data obtained through the analysis of the videos of the tasks will be stored in files. In this way, information management will be made more effective. Regarding the disadvantage related to the difficulty of presenting the findings in the research, we will organize the findings and present them using tables.

2.3 Clustering Algorithms

Clustering algorithms are applied to sets of objects, in order to separate them into clusters. Where objects with similar characteristics are allocated in the same *cluster*. Clustering techniques can be applied in different areas, as shown below:

- ❏ In Biology, there is a need to classify different species of plants and animals, so clustering can be applied to facilitate this task.
- ❏ In Document Analysis, a major difficulty is labeling a large amount of data. One solution is to apply the clustering technique with the objective of grouping similar documents into clusters, in order to later perform analyzes more easily on these documents.
- ❏ In Libraries, the clustering technique can be used to group different books based on topics and information.
- ❏ In Marketing, the clustering technique can be used to segment consumers, in order to group people with similar characteristics, so that companies can better target messages and advertisements.

Clustering is generally used when no classes have been defined in advance for the data set. In this work, as we do not know the number of different solutions contained in the N pages returned by the search engine, clustering is suitable for separating and grouping pages that have similar solutions in the same cluster. The k-means algorithm is an example of a clustering algorithm and works as follows:

1. Set a cluster number.
2. Randomize the centroids of clusters.
3. For each object, calculate its distance between the centroids of clusters.
4. Allocate the object in the *cluster* with the smallest distance.
5. Update the centroids of clusters. For each cluster, calculate the average of all vectors of attributes corresponding to the objects present in the cluster.
6. Repeat steps 3, 4 and 5 for a set number of times or until the centroids of the cluster do not change much during the iterations.

Let us illustrate an example of application of the K-means algorithm. In the first step we will set the cluster number equal to two. In the second step, we will randomize the centroids of the two clusters. Figure 2 shows an illustration after the application of the first and second steps of the algorithm, where the data are illustrated by means of blue points and the positioned centroids are represented by the character C in green and purple colors.

In the third step, the distance between an object (point) and the centroids of the two clusters is calculated for each object. In the fourth step, the objects are allocated to the closest one, that is, the object is allocated to the cluster that has the smallest distance

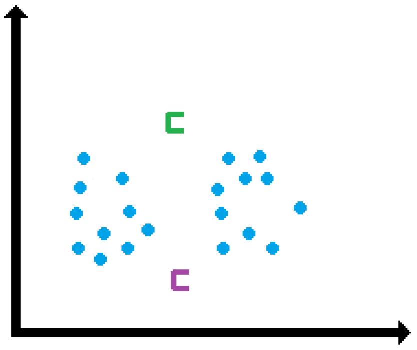

Figure 2 – Example of the application of the first and second steps of the k-means algorithm.

calculated in the previous step. Figure 3 shows an illustration after applying the third and fourth steps of the algorithm, where the blue dots are the objects allocated to cluster A and the red dots are the objects allocated to cluster B.

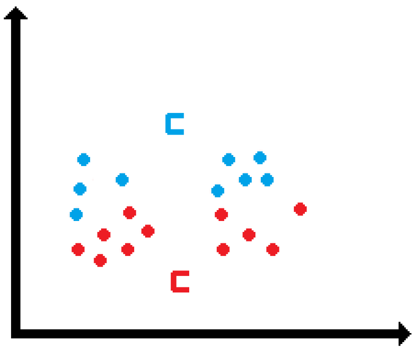

Figure 3 – Example of the application of the third and fourth steps of the k-means algorithm.

In the fifth step, the centroids of the two clusters are updated. For each cluster, the average of all attribute vectors corresponding to the objects present in the cluster is calculated. Figure 4 shows an illustration after updating the centroids of the two clusters.

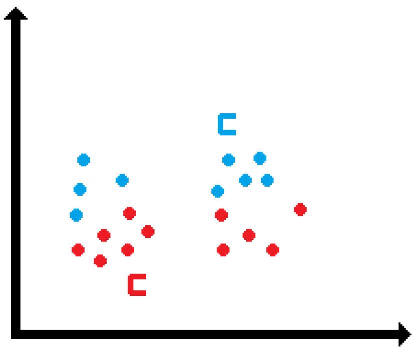

Figure 4 – Example of the end of the application of the k-means algorithm.

In the sixth step, the third, fourth and fifth steps are repeated, up to a defined number of times or until the centroids of the clusters do not change during the iterations. Figure 5 shows an illustration after the algorithm is finished.

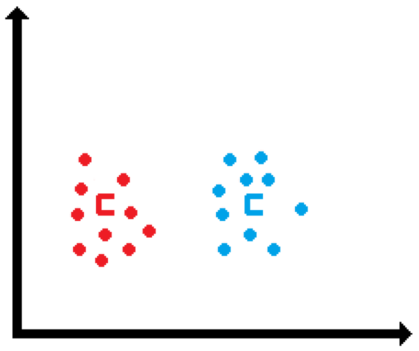

Figure 5 – Example of application of the k-means algorithm.

In this work, the clustering algorithm k-means is used in order to group web pages that have a set of method calls that are similar in their solutions, in the same cluster.

We have chosen k-means as the clustering algorithm because it is more adaptable to different contexts, in addition to guaranteeing convergence. Some limitations of k-means are choosing K manually and clustering outliers[2]. We mitigated the choice of K by performing preliminary tests, where we found a sub-optimal value for K. Regarding outlier clustering, the problem occurs because centroids can be dragged by outliers, or outliers might get their own cluster instead of being ignored. To mitigate this limitation, we propose a filter to remove outliers pages, and we apply k-means only after this filtering.

[2] https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages

Chapter **3**

The Influence of Search Engine Ranking Quality on the Performance of Developers

Software development is an activity that requires a constant search for information by software developers. One problem faced by developers is the lack of code examples in the official documentation of software development technologies (KIM et al., 2010). Code examples help developers better understand the technology (NYKAZA et al., 2002; ROBILLARD, 2009), as well as providing software reuse (BUSE; WEIMER, 2012; JOHNSON, 1992). An alternative widely used by developers is to resort to the Web for solutions that have code examples to solve problems related to their daily software activities (STOLEE; ELBAUM; DOBOS, 2014; SIM et al., 2011; GALLARDO-VALENCIA; SIM, 2009; KIM et al., 2010). Search engines, such as Google, Bing and Yahoo, help developers to find pages with solutions to their problems (HORA, 2021b; NIU; KEIVANLOO; ZOU, 2017; FISCHER; STACHELSCHEID; GROSSKLAGS, 2021). However, the best solutions are not always among the first ones returned by these search engines (CHATTERJEE; JUVEKAR; SEN, 2009; HORA, 2021a). Due to the lack of quality of top-ranked pages, users may spend a considerable time to find the desired solution (XIA et al., 2017).

In view of the problems previously reported, this study aims to understand the influence that the order of the pages returned by the ranking of search services exerts on the performance of developers when performing programming tasks.

3.1 Study Setting

The aim of the current study is to answer the following research questions:

RQ1: *Does the ranking quality of query results have any influence on the performance of developers in the development of programming tasks?*

RQ2: *Is there any influence by adding a list of frequent methods in the description of the pages in the result returned by the search engine for developers?*

RQ3: *How do irrelevant pages present in lower quality rankings hinder developers' goal of finding relevant solutions?*

To answer the research questions, we designed an qualitative study with software developers to compare their performance when solving programming tasks, when using ranking of pages returned by the search engine with two different quality levels (Higher Quality Ranking and Lower Quality Ranking). The general idea of the qualitative study is to get developers to execute pairs of programming tasks, varying the ranking quality between the pairs of tasks and trying to keep the other variables constant. Therefore, the first and second tasks proposed in the qualitative study have similar levels of difficulty, the same applies to the third and fourth tasks. In this way we can compare the performance of the developers when changing the quality level of the ranking.

Videos from participants' computers were recorded using software. Subsequently, we performed an analysis on the recorded videos in order to verify the performance of developers when using rankings with different levels of quality. To define the time limit that the developers could perform each task, we solved the tasks and count the time, we found an average solution time around 30 minutes, so the time limit was defined as twice the average time taken to solve the programming tasks, that is, the time limit for participants to solve each task was one hour. For each programming task, the developers answered a form containing questions related to the pages available for solving the programming task.

3.1.1 Definition of Programming Tasks for the Qualitative Study

For the qualitative study, we selected four programming tasks in the JAVA language that use native APIs. We chose four programming tasks in order to make pairwise comparisons between pairs of tasks with similar difficulty levels. In this way, for each pair of tasks, we try to change only the ranking quality, in order to measure the performance of the developers and, later, verify if the ranking quality influences the developer's performance during the resolution of programming tasks. We avoided choosing non-native APIs due to the difficulty that participants would have when configuring such APIs during the training phase for participating in the study. Therefore, for the first and second tasks, we chose the Swing API. For the third and fourth tasks, we chose the JDBC API.

To choose pairs of similar tasks, that is, similar first and second tasks, and similar third and fourth tasks, we searched the web for tutorials containing programming tasks for the Swing and JDBC APIs. After we found two pairs of programming tasks, we made adaptations to these programming tasks in order to make the task pairs as similar as possible. In order to carry out these adaptations, we tried to make the number of modifications performed by the developer while solving the pairs of programming tasks as similar as possible. For that, we built tables to compare the technical elements (classes, methods and control structures) manipulated by the developers during the resolution of the proposed programming tasks. Table 1 shows the comparison of the technical elements that will have to be handled by developers during the resolutions of the first and second tasks, after the adaptations we made in these tasks to make them as similar as possible.

Table 1 – Comparison of the technical elements handled during the resolutions of the first and second tasks.

Technical Elements	**1st Task**	**2nd Task**
# Methods Inserted	10	14
# Methods Changed	4	3
# Methods Removed	3	3
# Classes Inserted	3	3
# Changed Classes	3	3
# Classes Removed	3	3
# IFs	3	3
# Variables Inserted	4	0
Totals of Changes in Source Code	**33**	**32**

As we can observe in Table 1, the Technical Elements column shows the technical elements that must be handled by developers to perform the proposed programming task. The 1st Task and 2nd Task columns show the amount of inclusion, changes and removal of methods and classes, and the amount of inclusion of IF structures and variables that developers must perform to solve the respective tasks of schedules. As we can observe, the amount of changes that the developer must make is similar, that is, 33 changes in the source code of the first task and 32 changes in the source code in the second task. We consider that the difficulty of including methods and variables are similar.

Table 2 shows the comparison of the technical elements that will have to be handled by developers during the resolutions of the third and fourth tasks. As we can see, the amount of changes that the developer must make is similar, that is, 10 changes in the source code for both programming tasks. Next, we describe the programming tasks proposed in the qualitative study.

The **first task** is to make changes to a partially implemented JAVA project. The project consists of a simple user registration system, containing fields *Name*, *Cell Phone Number* and three radio-type options for the user to choose the type of preferred game. There are two buttons below the fields, the *Submit* button and *Reset* button. When the user presses the *Submit* button, the information in the *Name* and *Cell Phone* field is printed in a print area present next to the fields. When the user presses the *Reset* button, the information entered in the fields is erased. The proposed task is to change the options from the radio type to the checkbox type and print the options chosen in the print area.

The **second task** is to make changes to a partially implemented JAVA project. The project consists of a simple chat application configuration system, containing three con-

Table 2 – Comparison of the technical elements handled during the resolutions of the third and fourth tasks.

Technical Elements	**3rd Task**	**4th Task**
# Methods Inserted	7	9
# Methods Changed	0	0
# Methods Removed	0	0
# Classes Inserted	2	1
# Changed Classes	0	0
# Classes Removed	0	0
# IFs	1	0
# Variables Inserted	0	0
Totals of Changes in Source Code	**10**	**10**

figuration options: audio, automatic download and storing conversation history. In front of each option is a button with the text “Enable”. When the user presses the button, nothing happens. The proposed task is to change the JButton type buttons to the toggle button type and change the text in the toggleButton to “Disable” when pressed by the user.

The **third task** is to make changes to a partially implemented JAVA project. The project consists of a login system, containing *Username* and *Password* fields. Other than a login button, when the user presses this button, nothing happens. The proposed task consists of implementing codes responsible for verifying in the database if the information entered in the *Username* and *Password* fields is present in the database. In addition to implementing two messages. If the information entered in the login form is correct in the database, the message “Correct login.” should appear on the screen, otherwise the message “Incorrect username or password.” should appear on the screen.

The **fourth task** is to make changes to a partially implemented JAVA project. The project consists of a contact registration system, containing the fields: *Name*, *Telephone Number*, *E-mail*, *Address*, *City* and *State*. The system also has a button called “Register”, when the user presses this button, nothing happens. The proposed task consists of implementing codes responsible for inserting the information typed by the user in the form fields into the database, and if the insertion of data in the database is performed successfully, the message “Contact registered successfully” should appear on the screen.

3.1.2 Building Higher Quality Ranking and Lower Quality Ranking

We use the following structure in the construction of queries: “how to” + programming task + programming language + API(s).

We use the sentence “how to” in order to filter only the pages related to how to implement some functionality in the program. This filter prevents pages with content related to the debug-corrective type from being returned by the search engine. Since the objective of this work is to obtain content that helps the developer to implement functionalities to solve programming tasks, content related to error/bug correction is not applied in this work. To obtain the content needed to solve the programming task, we include the sentence related to the proposed task in the construction of the query. This causes content related to the problem to be solved by the developer to be returned by the search engine. To obtain the content related to the programming language and API used in the proposed task, we include the name of both the programming language and the API name in the query construction. Regarding the order of sentences in the query and the addition or removal of some terms, these issues do not largely affect the results returned by the search engine, as shown in the work of Hora (2021b).

Following the aforementioned method, we build the following queries and obtained the list of the top-20 pages returned from Google, for each of the four proposed programming tasks.

1. *how to insert multiple checkboxes in form java swing.*
2. *how to create switch button java swing.*
3. *how to create login with database java swing jdbc.*
4. *how to insert data in database java swing jdbc.*

To build the Higher Quality Ranking, for each of the proposed programming tasks, we analyzed the first 20 pages returned by Google, and evaluated the solutions present on the pages, considering the criteria Degree of Focus, Solution Size and Code Examples proposed in the study of Chapter 4. Pages evaluated using the Degree of Focus criterion can contain one of the following values. *(1) Very low, solutions that are not related to the search performed by the user. (2) Low, solutions that have some connection with the search performed by the user. (3) Neutral, when it is not possible to evaluate the page for this criterion. (4) High, solutions that are related to the search, however, it has minor focus deviation. (5) Very high, solutions completely related to the search performed by the user.* Pages evaluated considering the Solution Size criterion can contain one of the following values, considering the number of lines of source code observed in the solutions. *(1) Very small, far below the observed average size. (2) Small, marginally below observed the average size. (3) Neutral, when it is not possible to evaluate the page for this criterion. (4) Large, marginally above the observed average size. (5) Very large, far above the observed average size.* Pages evaluated against the Code Example criteria can have a binary value of *"yes" or "no"*, depending on containing or not code examples in their solutions. After evaluating the solutions on the pages, we sort (rank) the pages, where the pages with a very high degree of focus, with code examples and solution size approaching the average size of the solutions found in the 20 pages (evaluated with small or large values for the Solution Size criterion), are placed at the top of the ranking.

To build the Lower Quality Ranking, we first carried out a study with 14 different queries, given as input in the Google search engine, in order to know in which position of ranking the relevant page occurs. That is, a relevant page with a very high degree of focus, with code examples and size of the solution that approximates the average size of the solutions found in the 20 pages. Among the 14 queries analyzed, we considered the 10 queries selected in the study of the Chapter 4 and the four queries selected in this study, as discussed earlier. Table 3 shows the 14 queries with their respective positions where the relevant pages occur. To define where the relevant page will occur in Lower Quality Ranking, we calculate the median of the positions that the relevant pages occur, excluding queries where the relevant page occurs in the first position, as such queries are considered as Higher Quality Ranking, we find the median equals to **five**. So, we build the Lower Quality Ranking as follows. In the first four positions, we have pages with low-focus solutions (solutions that are not focused on the query performed by the user). In the fifth position, we place the relevant page. In the sixth and seventh position we place pages with low focus. In the eighth position we place a page with high focus. In the ninth and tenth positions we place pages with low focus and so on.

Table 3 – Positions where the relevant pages occur.

Queries	**Position**
Query 1	10th
Query 2	4th
Query 3	4th
Query 4	4th
Query 5	15th
Query 6	1st
Query 7	4th
Query 8	1st
Query 9	5th
Query 10	5th
Query 11	15th
Query 12	7th
Query 13	3rd
Query 14	1st
Median	5th

3.1.3 Recruitment for the Qualitative Study

The target audience to participate in the present study were undergraduate students who are currently enrolled in the Object-Oriented Programming course and graduate students in Software Engineering, both at the Federal University of Uberlândia. We contacted these students as they were the closest contacts we had. We sent an email with an invitation to participate in the study to these students. The group of graduate students in Software Engineering has 19 members and the group of undergraduate students who take the Object-Oriented Programming course has 80 members. In total, four participants agreed to participate in the study, three of them belonging to the group of graduate students and one participant belonging to the group of undergraduate students. Considering that the number of participants was limited, we limited the scope of the research, and we chose to conduct a qualitative study, since we would not able to obtain statistically significant results. In this way, the number of participants is sufficient to answer the formulated research questions.

3.1.4 Ranking Evaluation

In order to evaluate the rankings related to the programming tasks proposed in the qualitative study, we created a form containing the following multiple-choice and open-ended questions. In multiple choice questions, the participant chooses one of five Likert-scale options. In open questions, participants are free to write their answer. For each proposed programming task, a form was answered by the participants after the task was completed or the time limit established. The proposed form has the following questions:

1. Regarding navigability by ranking, how agile/productive was it to find the solution for the programming task?

   a) **Very agile/productive.** The solution was easily found among the first pages of the ranking. That is, you browsed one to two pages of the ranking to find the solution.

   b) **Agile/productive.** The solution was found among the first pages of the ranking, but it was necessary to visit some pages to find it. That is, you browsed three to five pages to find the solution.

   c) **Neutral.** When it was not possible to evaluate this item.

   d) **Not agile/productive.** The solution was found among the ranking pages, but it was necessary to visit several ranking pages to find it. That is, you browsed from six to ten pages to find the solution.

   e) **Very little agile/productive.** The solution was found after a lot of effort, browsing through most of the ranking pages or the solution was not found among the ranking pages. That is, you browsed eleven or more pages to find the solution, or the solution was not found after browsing through the pages.

2. Was the summary below the link, which shows the five methods that occurred most on the pages, relevant to your decision to click on the page?

   a) **Very relevant.** Only the method list itself was used in the decision to click on the page or not. That is, you only used the contents of the list of methods in the decision to click or not to click on the page. Another possibility is that other content in the link and page description was irrelevant to the decision of whether or not to click the page.

b) **Relevant.** The content of the method list contributed to the decision whether or not to click on the page link. That is, you used both the content of the method list and the content of the link and page description in the decision to click on the page or not.

c) **Neutral.** When it was not possible to evaluate this item.

d) **Irrelevant.** The contents of the method list did not contribute to the decision whether or not to click on the page link. That is, you read the contents of the list of methods, but this was not used in the decision to click or not to click on the page.

e) **Very irrelevant.** The contents of the method list were not used in the decision to click or not on the page. That is, you did not read the list of methods, so it was not used in the decision to click on the page or not.

3. Regarding the pages you used to solve the programming task, what were your positive points? What did they have that helped you solve the task?

   The purpose of this question is to collect data on the characteristics of relevant pages (pages that were used to solve the proposed task). Considering our objectives, this information is important for the improvement of the filters proposed and evaluated in this work. Once the relevant pages are filtered out of search engine results, a higher quality ranking can be generated.

4. Regarding the pages that you did NOT use to solve the programming task, what were your negative points? What did they NOT have that did NOT help you solve the task?

   The purpose of this question is to collect data on the characteristics of irrelevant pages (pages that did not contribute to solve the proposed task). This information is also important for the improvement of proposed filters, once the bad characteristics of the pages are identified, more efficient filters can be developed.

3.1.5 Qualitative Study Steps

As we mentioned earlier, we sent an e-mail with an invitation to participate in the study to students. In this email, we put information about the study and ask those interested in contributing to the study to respond to the email.

We send a second email to people who replied to the first email agreeing to contribute to the study. This second email, we put a link to an explanatory video, to explain how the study would be conducted, along with training videos on how to install and use a program to record the participant's computer screen, and how to share the recorded video using a cloud storage service. At the end of the e-mail, there was a link to an online form, where the participant answers some basic questions, such as: degree of comfort in reading web pages with content in English, years of experience with the JAVA programming language and a question asking if the participant has watched all the training videos sent in the email.

We sent a third e-mail to the participants who answered the questionnaire present in the second e-mail. This third e-mail contained the instructions of what should be done by the participants. We recorded explanatory videos on how to import the projects of the proposed programming tasks into the IDE and explanations of the statements of the proposed programming tasks. For each programming task, the participant watches the video about the statement, then they start recording computer screen, after that they solve the tasks using the pages of the ranking available, and then they answer the form containing questions about the pages available for solving the proposed task. After carrying out all the steps, the participant sends the videos to the researchers.

3.1.6 Methodology for Evaluating the Results

In order to analyze and evaluate the results obtained through the participants' videos, we used Grounded Theory as inspiration to extract and categorize the findings, presented in Chapter 2.

The first step was to analyze the resolution of task 01 of participant 01, in order to encode their actions. We did the same procedure for participants 02, 03 and 04. From the analysis of participant 03, no new code was created. The following codes were created after analyzing the resolution of task 01 of the four participants, in front of each code there is a description of how we identified it in the video:

1. **Analysis of the solution source code in the IDE.** The participant stopped at the source code in the IDE.

2. **Change the solution source code in the IDE.** The participant typed or removed characters in the source code in the IDE.

3. **Quick analysis of the textual content on the page.** The participant scrolled slowly through the textual content of the page, with few short stops.

4. **Quick analysis of the source code on the page.** The participant scrolled slowly through the source code present on the page, with short stops.

5. **Analysis of textual content on the page.** The participant stopped at the textual content of the page (long stops).

6. **Analysis of the source code present on the page.** The participant stopped at the source code present on the page (long stops).

7. **Skipped textual content on the page.** The participant quickly scrolled through the textual content of the page (non-stop), or the participant used the browser's search bar with a technical element name as input (class, method, etc.).

8. **Skipped the source code present on the page.** The participant quickly scrolled through the source code present on the page (without stops).

9. **Copy of source of source code present on the page.** The participant copied a snippet of source code present on the page and pasted it into the source code in the IDE.

10. **Copy of some part of the ranked content.** The participant copied page link content (description or list of most frequent methods on the page) of ranking and pasted it into the source code in the IDE.

11. **Analysis of ranked list of pages.** The participant rolled slowly through the list of ranked pages with long stops.

After we defined the above codes, we analyzed the videos with the resolutions of the four proposed tasks, for the four participants, totaling 16 analyses. In each analysis, we measured the time the participant spent to perform each action coded above, on each page of the ranking used in solving the given task.

We grouped the previous code into the following categories:

1. **Ranked list of pages.** This category groups the following codes related to ranking: 10. Copy of some part of the ranked content and 11. Analysis of ranked list of pages.

2. **Page searching.** This category groups the following codes related to pages: 3. Quick analysis of the textual content on the page, 4. Quick analysis of the source code on the page, 5. Analysis of the textual content on the page, 6. Analysis of the source code present on the page, 7. Skipped textual content on the page, 8. Skipped the source code present on the page and 9. Copy of source code present on the page.

3. **Editing in the IDE.** This category groups the following codes related to the IDE: 1. Analysis of the solution source code in the IDE and 2. Change the solution source code in the IDE.

We analyzed the time duration that participants spent in each of the categories. We also calculated the total time spent solving each task by adding the total time spent searching the ranked list of pages, plus the total time spent searching the pages, plus the total time spent editing in the IDE.

We also analyzed the use of each page visited by the participants. Here are the types of pages we define:

1. **Useful.** Page content was used to completely solve the desired solution.
2. **Partially useful.** The page content partially contributed to the desired solution.
3. **Useless.** The page content did not contribute to the desired solution.

For each task, we calculated the time participants spent on useful, partially useful, and useless pages.

3.2 Results

This section presents the results obtained through the analysis of the participants' videos. Four participants participated in the qualitative study, as each participant solved four tasks, we obtained a total of 16 tasks (samples). Eight tasks were performed using the Higher Quality Ranking and the other eight tasks were performed using the Lower Quality Ranking. In the Higher Quality Ranking, in one of the tasks, one of the participants did not record the computer screen, so in the paired analysis of tasks, we do not consider the pair of tasks corresponding to this task. Another participant solved the task using only the content of the descriptions of the pages in the rank, in this case the participant did not enter the pages returned by the search engine.

3.2.1 On the Influence of Ranking Quality

Table 4 shows the results of the questionnaire for the question *"Regarding navigability by ranking, how agile/productive was it to find the solution for the programming task?"*. Where ++, +, − and −−, represent the options of the form very agile/productive, agile/productive, not agile/productive and very little agile/productive, respectively.

Table 4 – Results of the navigability by ranking, for the four proposed programming tasks.

Navigability	++	+	−	−−
Task 1 (higher quality ranking)	●	●	●	
Task 2 (lower quality ranking)			● ●	
Task 3 (higher quality ranking)	● ● ●			
Task 4 (lower quality ranking)		● ● ●	●	

As we can observe in the Table 4, for the tasks related to the higher quality ranking (Tasks 1 and 3), the participants found it more agile and productive, in relation to the tasks related to the lower quality ranking (Tasks 2 and 4). When we make a pairwise comparison between Tasks 1 and 2, we observe in Table 4 that we have more positive evaluations for Task 1, which uses the higher quality ranking, than Task 2, which uses the lower quality ranking. This observation is also valid for the pairwise comparison between Tasks 3 and 4, where the evaluation of Task 3, which uses higher quality ranking, is more positive than Task 4.

Tables 5, 7 and 8 show the findings we found in the results of the analysis of the participants' videos. The Table 5 shows the findings related to RQ1 (*Does the ranking quality of query results have any influence on the performance of developers in the development of programming tasks?*), where the first one shows that the resolution time of tasks that use the Higher Quality Ranking is smaller than the resolution time of the tasks that use the Lower Quality Ranking. This finding was obtained through an analysis of the six pairs of task resolutions, where five samples had the resolution time of tasks that use the Higher Quality Ranking smaller than the resolution time of the tasks that use the Lower Quality Ranking.

The second finding shows that in tasks related to Higher Quality Ranking, participants only use the pages at the top of the rankings. This finding was obtained through an analysis of the eight tasks related to the Higher Quality Ranking, in three tasks the participants used only the first page of the ranking to solve the task, in another three tasks the participants used the first and second pages of the ranking. In another task, one of the participants did not use any pages, only the content of the description of the pages in the ranking. One of the participants did not record the resolution of the task.

The third finding shows that participants visited more pages related to Lower Quality Ranking than those related to Higher Quality Ranking. On average, participants visited 4.6 pages related to Lower Quality Ranking and 1.1 pages related to Higher Quality Ranking.

The fourth finding shows that participants visited more useless pages related to Lower Quality Ranking than those related to Higher Quality Ranking. A total of 23 useless pages, related to Lower Quality Ranking, were visited by the participants. Only one useless page, related to Higher Quality Ranking, was visited by one of the participants.

The fifth finding shows that two tasks related to Lower Quality Ranking were not completely solved by two participants, out of a total of 16 tasks solved by the participants. On the other hand, in the Higher Quality Ranking, all tasks were completely resolved by the participants.

The sixth finding shows that in partially completed tasks, participants visited pages in the lowest positions of Lower Quality Ranking. Participant 02 visited the 14th page and participant 03 visited the 11th page.

Table 5 – Findings related to **RQ1**, found in the results of the analysis of the participants' videos.

#	**Findings**	**Sources for the Findings**
1	The resolution time of tasks that use the Higher Quality Ranking is smaller than the resolution time of tasks that use the Lower Quality Ranking, for most samples.	Of the eight pairs of tasks resolutions, we removed two pairs of tasks, as the participants did not complete the tasks completely. Of the six task resolution pairs, five had a shorter resolution time for tasks related to the higher quality ranking than the resolution time for tasks related to the lower quality ranking. Having an average of the time difference between the task pairs of 242 seconds (4 minutes and 2 seconds).
2	In tasks related to Higher Quality Ranking, participants only use the pages at the top of the rankings.	In three tasks the participants used only the first page of the ranking to solve the task, in another three tasks the participants used the first and second pages of the ranking. In another task, one of the participants did not use any pages, only the content of the description of the pages in the ranking. One of the participants did not record the resolution of the task.
3	Participants visited more pages related to Lower Quality Ranking than those related to Higher Quality Ranking.	On average, participants visited 4.6 pages related to Lower Quality Ranking and 1.1 pages related to Higher Quality Ranking.
4	Participants visited more useless pages related to Lower Quality Ranking than those related to Higher Quality Ranking.	A total of 23 useless pages, related to Lower Quality Ranking, were visited by the participants. Only one useless page, related to Higher Quality Ranking, was visited by one of the participants.
5	Two tasks related to Lower Quality Ranking were not completely resolved by the participants.	Of the total of 16 programming tasks, two tasks related to Lower Quality Ranking were not completely solved by two participants. On the other hand, in the Higher Quality Ranking, all tasks were completely resolved by the participants.
6	In the partially completed tasks, the participants visited pages in the lowest positions of the Lower Quality Ranking.	Participant 02 visited the 14^{th} page and participant 03 visited the 11^{th} page.

Answer to RQ1: *Does the ranking quality of query results have any influence on the performance of developers in the development of programming tasks?* As shown in the results of Table 5, in the Higher Quality Ranking, developers spend less time solving the programming task, as the best pages are at the top of the ranking. This way, developers don't spend time on irrelevant pages. Considering the participants of this study, on average, around 4 minutes more were spent during the resolution of tasks related to the Lower Quality Ranking. Regarding Higher Quality Ranking, other influencing factors were: participants did not use irrelevant pages and visit fewer pages. Regarding the Lower Quality Ranking, the influencing factors were: participants visit more irrelevant pages, did not complete two tasks. In these two tasks, participants visited the lowest ranking positions.

3.2.2 On the Influence of the List of Methods

Table 6 shows the results of the questionnaire for the question *"Did the summary below the link, where it shows the five methods that occurred most on the pages, have any relevance in your decision to click on the page?"*. Where ++, +, − and −−, represent the options of the form very relevant, relevant, irrelevant and very irrelevant, respectively.

Table 6 – Results of the addition of the list of methods in the summary below the links of the pages, for the four proposed programming tasks.

Added Methods List	++	+	−	−−
Task 1 (higher quality ranking)		● ●	● ●	
Task 2 (lower quality ranking)			● ●	
Task 3 (higher quality ranking)	●		●	
Task 4 (lower quality ranking)		● ●	●	

As we can observe in Table 6, for the pair of Tasks 1 and 2, the list of methods added in the summary below the links of the pages, was irrelevant for four task resolutions, for two task resolutions, the list of methods was relevant. As for the pair of Tasks 3 and 4, the list of methods added was relevant, for most participants, only one participant said that the list is irrelevant for these tasks. In order to verify the consistency between the responses of this form, we analyzed the participants' video. There was a case of inconsistency, where one of the participants used the methods present in the list of methods (copied it and pasted it in the source code present in the IDE), but in the form where an answer was requested about the relevance of the list of methods, the participant marked it as irrelevant. In this case, we changed that participant's answer to relevant, because the participant may have marked the wrong option on the form, since their actions observed in the video recording do not match the option marked on the form.

The Table 7 shows the findings related to RQ2 (*Is there any influence by adding a list of frequent methods in the description of the pages in the result returned by the search engine for developers?*), where the seventh finding shows that one of the participants used the methods in the list of methods to solve two programming tasks. we observed that this participant analyzed the list of methods from one of the pages, copied the name of the method, pasted it into the source code in the IDE and performed the task, without having to enter any ranking page. The same participant copied a method name from the method list to another proposed task.

Table 7 – Findings related to **RQ2**, found in the results of the analysis of the participants' videos.

#	Findings	Sources for the Findings
7	One of the participants used the methods in the list of methods to solve two programming tasks.	One of the participants analyzed the list of methods from one of the pages, copied the name of the method, pasted it into the source code in the IDE and performed the task, without having to enter any ranking page. The same participant copied a method name from the method list to another proposed task.
8	All but one participant skipped irrelevant pages.	We analyzed the tasks where the participants skipped irrelevant pages in the ranking, we observed that two participants pass the mouse cursor over the names of the methods present in the list of methods, which indicates that the participants analyzed the list, which may have an influence on the decision of the develop on whether or not to enter the page. The rest of the participants were stuck at the description of the pages where the list of methods is located, but it is not clear whether or not they were analyzing the list of methods.

Answer to RQ2: *Is there any influence by adding a list of frequent methods in the description of the pages in the result returned by the search engine for developers?* As shown in the results of Table 7, there seems to be an influence, since three participants skipped irrelevant pages present in the ranking, however one participant did not skip irrelevant pages. In addition to one of the participants having used the content of the lists of methods in two proposed tasks, in one of these tasks, the participant only used the content of the list, without having entered the ranking pages.

3.2.3 On the Influence of Irrelevant Pages on Developers

The Table 8 shows the findings related to RQ3 (*How do irrelevant pages present in lower quality rankings hinder developers' goal of finding relevant solutions?*), where the ninth finding shows that the time spent analyzing the Lower Quality Ranking pages is greater than the time spent analyzing the Higher Quality Ranking pages. Of the six pairs of task solving, we observed that four samples have more time spent on the analysis of the Lower Quality Ranking pages, compared to the analysis of the Higher Quality Ranking pages. Having an average of the time difference between the task pairs of 139 seconds (2 minutes and 19 seconds). On the other hand, the other two samples where the time spent analyzing the Higher Quality Ranking pages was longer than the Lower Quality Ranking pages had an average time difference between the tasks of 24 seconds.

The tenth finding shows that in solving the tasks related to Lower Quality Ranking, the participants partially use the solutions present in the pages to solve the tasks. In the Lower Quality Ranking, of the eight tasks, five tasks the participants combined the

Table 8 – Findings related to **RQ3**, found in the results of the analysis of the participants' videos.

#	**Findings**	**Sources for the Findings**
9	The time spent analyzing the Lower Quality Ranking pages is longer than the time spent analyzing the Higher Quality Ranking pages, for most samples.	Of the six pairs of task solving, we observed that four samples have more time spent on the analysis of the Lower Quality Ranking pages, compared to the analysis of the Higher Quality Ranking pages. Having an average of the time difference between the task pairs of 139 seconds (2 minutes and 19 seconds). On the other hand, the other two samples where the time spent analyzing the Higher Quality Ranking pages was longer than the Lower Quality Ranking pages had an average time difference between the tasks of 24 seconds.
10	In solving the tasks related to Lower Quality Ranking, most of the participants partially use the solutions present in the pages to solve the tasks.	In the Lower Quality Ranking, of the eight tasks, five tasks the participants combined the solutions of two or more pages to solve the proposed task. Already in the Higher Quality Ranking, only in one task the participant 04 combine the two-page solutions to solve the proposed task.
11	For all pairs of fully resolved tasks, the time spent on useless pages was higher for the Lower Quality Ranking.	Of the six task resolution pairs, we observed that all of them have more time spent on useless pages in the Lower Quality Ranking, compared to the Higher Quality Ranking. Having an average of the time difference between the task pairs of 105 seconds (1 minutes and 45 seconds).

solutions of two or more pages to solve the proposed task. Already in the Higher Quality Ranking, only in one task did participant 04 combine the two-page solutions to solve the proposed task.

The eleventh finding shows that for all pairs of tasks analyzed, the time spent on useless pages was higher for the Lower Quality Ranking. Of the six task resolution pairs, we observed that all of them have more time spent on useless pages in the Lower Quality Ranking, compared to the Higher Quality Ranking. Having an average of the time difference between the task pairs of 105 seconds (1 minutes and 45 seconds).

Answer to RQ3: *How do irrelevant pages present in lower quality rankings hinder developers' goal of finding relevant solutions?* As shown in the results of Table 8, on average, the time spent on irrelevant pages is greater than the time spent on relevant pages (developers spend 1 minute and 45 seconds more on irrelevant pages), because when developers encounter a bad code on a irrelevant page, they spend a lot of time trying to reuse the bad code. We observed that developers try to fix the bad code by searching for content on other pages, spending even more time. In case the developer cannot fix the bad code, all the effort in finding the solution is wasted, so the developer must start the search for the solution on another page.

3.3 Discussion

This section aims to discuss the findings shown in Tables 5, 7 and 8. First we will show the findings related to RQ1. The first finding was that the resolution time of the tasks that used the Higher Quality Ranking is less than the resolution time of the tasks that use the Lower Quality Ranking. We attribute this to the time spent on useless pages at Lower Quality Ranking, both reading the content of the pages and analyzing and coding codes do not contribute to the resolution of the task. When developers come across bad code, not knowing if it is bad, they try to use that code. The moment they realize that the code does not contribute to the solution, either the developers delete the bad code, or they try to fix the code by looking for information on other pages. The worst case is when developers cannot fix the code, wasting all the time invested in irrelevant pages.

The second finding was that in tasks related to Higher Quality Ranking, participants do not use irrelevant pages. This is because relevant pages are at the top of the Higher Quality Ranking. When the ranking has high quality, the participants already find the solution for the task on the first pages. In the analysis of the results of the qualitative study, we found that out of the six tasks related to the Higher Quality Ranking, in three tasks the participants used only the first page of the ranking to solve the task, in another three tasks the participants used the first and second pages of the ranking.

The third finding was that participants visited more pages related to Lower Quality Ranking than those related to Higher Quality Ranking. We observed that, on average, participants visited 4.6 pages related to Lower Quality Ranking and 1.1 pages related to Higher Quality Ranking. This is due to relevant pages are spread out in the rankings, so participants must visit more pages in the Lower Quality Ranking in search of those relevant pages. This is consistent with what we expected, putting a relevant page on the 5$^{\text{th}}$ position in a Lower Quality Ranking.

The fourth finding was that participants visited more useless pages related to Lower Quality Ranking than those related to Higher Quality Ranking. A total of 23 useless pages, related to Lower Quality Ranking, were visited by the participants. Only one useless page, related to Higher Quality Ranking, was visited by one of the participants. This is due to the best pages are at the top of the Higher Quality Ranking, so participants do not need to visit the useless pages that are at the bottom of the rank. This is consistent with our hypothesis that developers generally need to browse and read the pages until they find out that they are irrelevant for them.

The fifth finding was that two tasks related to Lower Quality Ranking were not completely resolved. Of the 16 tasks solved by the participants, two tasks related to Lower Quality Ranking were not completely resolved. This is due to bad pages are in the top positions of the rank, they hinder the participants to find the solution. Participants tried to solve the task with bad code from these pages. One of the participants did not complete the task, as this participant wasted a lot of time on irrelevant pages (visited six irrelevant pages), using around 53 minutes on the task. The other participant partially solved the task in less than 11 minutes, but this participant also visited irrelevant pages, which may have discouraged him from seeking the complete solution. This finding reinforces the hypothesis that inadequate pages may cause developers wasting a lot of time trying a solution that does not work properly.

The sixth found was that in the partially completed tasks, the participants visited pages in the lowest positions of the Lower Quality Ranking. Participant 02 visited the 14th page and participant 03 visited the 11th page. This is due to the good solutions are spread out in the rank. Bad solutions in the top positions make it difficult for the participants to find the good solution, so the participants try to look for good solutions on the pages in the lower positions of the rank.

As we can observe through the findings related to RQ1, the quality of the ranking has a strong influence on the performance of developers, since the resolution time of tasks related to the Higher Quality Ranking is shorter than the resolution time of tasks related to the Lower Quality Ranking. In the case of the Higher Quality Ranking, the relevant pages are at the top of the ranking, so the developer does not waste time analyzing irrelevant pages. In the Lower Quality Ranking, developers have to visit several useless pages that hindering their performance, until finding a relevant page that solves the desired problem.

Now we discuss the findings related to RQ2. The seventh finding was that one of the participants used the methods in the list of methods to solve two programming tasks This is due to the list of methods that we included in the description of the ranked pages. We observed that participant 02 performed the task only with the ranking content. This participant verified and copied the contents of the list of method calls placed in the description of the pages in the ranked list. This behaviour may be explained in cases where developers already expect some functions and theses list of methods work as a hint for them.

The eighth finding was that all participants skipped the irrelevant pages except one participant. Of the four participants, three skipped the irrelevant pages. This may be due to the list of frequent methods in the page descriptions in the rank. This list probably helps in the decision to enter the page or not, when faced with a list of methods that are not related to the desired solution, the participant decides to skip the page. For pages that do not have methods, the list of methods is empty, so the participant may infer that there is no source code on the page, consequently the participant would skip the page.

Finally, we will discuss the findings related to RQ3. The ninth finding was that the time spent analyzing the Lower Quality Ranking pages is longer than the time spent analyzing the Higher Quality Ranking pages. This is due to the fact that adequate solutions are spread out in the Lower Quality Ranking, so participants need to browse the rank to access other pages.

The tenth finding was that when solving the tasks using the Lower Quality Ranking, most of the participants partially use the solutions present in the pages to solve the tasks. In the Lower Quality Ranking, out of the eight tasks, in five tasks the participants combined the solutions of two or more pages to solve the proposed task. This is due to the bad solutions are among the first in the rank, the participants use the codes of these solutions. As the solution is not good, participants try to fix the bad solution with content from other pages.

The eleventh finding was that for all pairs of analyzed tasks, the time spent on useless pages was higher for the Lower Quality Ranking. Of the six task resolution pairs, we observed that all of them have more time spent on useless pages in the Lower Quality Ranking, compared to the Higher Quality Ranking. This is due to the fact that relevant solutions are spread out in the rank, so the participants need to analyze the useless pages, until they find a relevant solution.

As we can observe in the findings related to RQ3, irrelevant pages hinder the performance of developers. We show some characteristics of these pages reported by the participants and comment on them. First comment: *"The irrelevant pages had only formal definitions, with little applicability. One page did not even show the solution in Java."* As we can observe in this comment, irrelevant pages do not have examples with applicability. In this case, the developer wastes time looking for a practical example that the page does not offer. In the approach proposed in this thesis, we developed a filter to remove pages that do not have practical examples. Second comment: *"The irrelevant pages did not have examples that matched or related to what the task asked for."* As we can observe in this comment, irrelevant pages do not focus on the solution desired by the developer. In the approach proposed in this thesis, we developed a filter to remove pages with little focus. Third comment: *"The irrelevant pages had extensive content and sometimes did not have the necessary content to perform the task."* As we can observe in this comment, irrelevant pages that have extensive content and no focus on the proposed task, hinder the developer, because they need to analyze all the content, even if it does not contribute to the resolution of the task. Fourth comment: *"They were not specifically for the task I* *was looking to solve. I wanted snippet to save in the database, not to connect or consult in the database."* As we can observe in this comment, once again, irrelevant pages do not focus on the solution desired by the developer. In this way, the developer wastes time analyzing content that does not solve the problem. Fifth comment: *"They did not have the insertion, only codes for other operations, even when dealing with the same API."* In this comment, we once again realize that irrelevant pages do not focus on the solution desired by the developer. All previous comments reinforce the need for custom filters to filter search engine results for content related to software development.

3.4 Threats to Validity

In this section, we show some threats to the validity of this study.

Internal validity. This kind of threat is about how sure we can be that the treatment actually caused the outcome. In this study, a threat to internal validity was the degree of difficulty of the programming tasks considered in the study. Tasks with different degrees of difficulty between the lower quality ranking and the higher quality ranking can influence the results. To minimize this threat, we performed a paired analysis between pairs of tasks, with a similar degree of difficulty. We made studies and adaptations in these pairs of tasks to make them as similar in difficulty as possible.

External validity. This kind of threat is related to whether we can generalize the results outside the scope of our study. In this study, a threat to external validity was the number of study participants. A larger number of participants in the study could provide strong results. To overcome this limitation, we opted for an exploratory qualitative study, where we used Grounded Theory as inspiration to extract and categorize the findings. As this methodology is very rich in data analysis, the data obtained was detailed enough to answer the proposed research questions. Based on the results of this study, we can design quantitative studies in the future to reinforce these results. Another threat to external validity was the results were limited to the JAVA programming language, which is a popular language and has a lot of content on the Web. For other programming languages, the results may be different, especially for less popular languages, where there may be lack of content, even considering the whole Web.

Construct validity. This kind of threat is about the relation between the theory behind the study and the observation. In this study, a threat to construct validity was the interpretation of the videos of the resolutions of the tasks, carried out by the participants, since an action performed by the participant may not be the same as interpreted by the video analysis. For example, regarding the list of methods inserted in the description of the pages, when the participant stops at the description of the page, it is not possible to know exactly if he/she is reading or not the text of the description or the list of methods inserted, which could impact the results. To mitigate this threat, we focused on collecting more direct relationships, for example, in the case of the list of methods, we observed actions performed by the participants, such as: copying methods from the list of methods and pasting them into the source code present in the IDE and hovering the mouse cursor above the names of the methods present in the list of methods, followed by clicking on the link on the page. Another threat to the validity of the construction was the questionnaires designed to obtain information for the research. They may not capture the intention of the questions, as participants may answer the questions wrongly, because of subjective interpretations. To mitigate this threat, we have aimed to frame the questions as clearly as possible. We also analyzed the participants' video to verify that the answers to the questionnaires are consistent. For example, in the questionnaire where the participant is asked to rate the degree of relevance of the list of methods that we have inserted in the description of the page, we analyze the participant's video to verify if he/she made use of the methods present in the list or not, then we check the option that the participant marked in the questionnaire.

Conclusion validity. This kind of threat is about how sure we can be that the treatment is really (statistically) related to the actual outcome. This threat is not discussed in this work, since the objectives of the work are not related to statistics.

3.5 Conclusion

As we can observe in the results of this study, the ranking quality influences the performance of developers during the development of programming tasks. The time spent solving tasks that use a Higher Quality Ranking is less than the time spent solving tasks that use a Lower Quality Ranking. This is because relevant pages are at the top of the Higher Quality Ranking, so the developer does not waste time on irrelevant pages.

We can also observe that the list of frequent methods added below the descriptions of the ranking pages had an influence on the participants' decisions. One of the participants solved the task with only the content of the descriptions of the pages, including the methods indicated by the list. Other participants skipped irrelevant pages, possibly influenced by that information.

Another observation was that irrelevant pages negatively influence the development of programming tasks. In the Lower Quality Ranking, irrelevant pages are in the top positions of the rank and they hinder the participants to find the solution. When developers come across bad code on irrelevant pages, not knowing if the code is bad, they try to use it. The moment they realize that the code does not contribute to the solution, either the developers delete the bad code, or they try to fix the code by looking for information on other pages. The worst case is when developers cannot fix the code, wasting all the time invested in irrelevant pages.

The results of this study show the importance of developing approaches capable of improving the quality of the ranking returned by the search engine. As we noted, the Lower Quality Ranking hinder the performance of the developers who participated in the study. In the next chapter, we propose approaches to improve the quality of ranking returned by search engines.

Chapter 4

Mining Relevant Solutions from Search Engine Results

In this chapter, we propose an approach to mining relevant pages from search engine results for programming tasks by identifying and filtering out irrelevant pages from the ranked list. To evaluate the proposed approach, we performed comparisons with the results obtained through variations of the approach. All approaches evaluated using query results from the Google search engine.

4.1 The Mining Approach

We propose an approach to tackle the challenges related to filtering relevant content obtained by querying search engines, more specifically, removing top-ranked pages with undesirable features for developers, such as, lack of code examples and low focus.

The input for the mining approach is a query denoting a programming-related task. This query is executed on a search engine (in our case, Google), and the top-n pages are selected for an automatic filtering process to obtain pages with relevant solutions.

In this section, we present the steps of the proposed approach and discuss the components implemented in each one.

4.1.1 Mining Steps

The main steps of the approach are:

1. Preparation of a query related to the programming task, for example, "how to make login with php mysql". The queries are systematically created as: "*how to*" + *programming task* + *programming language* + *API.*

2. Querying the Google search engine taking the above query as input.

3. Selection of top-n pages returned by the search engine that have the following characteristics: textual content must be in English and must not contain only PDF, PowerPoint, Excel and Word files. To filter the content in English, the approach uses the search engine in the English version and the proposed query was formulated in English. The links returned by the search engine ending with the above file extensions are removed.

4. Two-phase automatic filtering of the n pages selected in the previous step. The first phase aims to remove outlier pages, that is, those that have few occurrences of method calls or a large number of occurrences of them in relation to the upper and lower limits based on the average occurrence of method calls in all the n pages. The second phase uses a clustering algorithm is also applied in order to obtain pages with solutions more focused on the user's query. The filtering mechanism is detailed in the next subsection.

4.1.2 Parameter calibration

In order to define the filtering mechanisms for the mining approach, we need also to define some parameters related to criteria to evaluate the pages returned by the search engine. We will rely on a bootstrap approach to define such parameters, i.e., we test the approach with an arbitrary parameter, and then evaluate the pages to define if those pages adhere to the criteria, and then if the pages should or not be filtered out from the search engine results. We run this process iteratively changing the respective parameter until we find a sub-optimal value. For this calibration, we collect the search results of three queries using the chosen parameters (*01 - how to create table java swing, 02 - how to create simple calculator in android and 03 - how to implement login php*). The top-20 ranked pages were selected for each query. Then, we manually evaluate such pages and calculate the *F-measure* to find which parameter should be chosen according the respective best *F-measure*. The manual evaluation criteria are defined in Section 4.2.3, which are the same used to define the ground-truth for evaluating the proposed approach.

4.1.3 Lower and Upper Limits for Outlier Pages

We hypothesize that the number of occurrences of method calls in a page should not be too small because of the possible lack of examples to explain a programming solution. Moreover that number of occurrences of method calls should also not be too large because of the possible complexity and lack of focus of the solution that would hinder the developer comprehension. In order to define such lower and upper limits for this number in a page, we have used the calibration bootstrap method defined in the previous subsection. These limits will be used in the outlier filter to be presented in Subsection 4.1.4.

The lower limit is calculated using the expression:

$$lowerLim = \frac{averageOccurrenceMethodCalls}{DIVISOR} \quad (1)$$

The upper limit is calculated using the expression:

$$upperLim = averageOccurrenceMethodCalls \times FACTOR \quad (2)$$

Following, we show how the constants DIVISOR and FACTOR were defined. For each query, we analyzed the 20 pages using the Degree of Focus criterion defined in Subsection 4.2.3.2. The pages evaluated with the values “High” or “Very High” were classified as “*Selected*” and the pages evaluated with the values “Low”, “Very Low” and “Neutral” were classified as “*Not Selected*”. For each query, considering the average total occurrence of the method calls present in the 20 pages, the DIVISOR values of the expression lower limit varied from 2 to 10, by 0.5, and the multiplication FACTOR of the upper limit also was varied from 1.5 to 4, by 0.5 in 0.5. Then, for each query, the respective results were manually assessed, and the metrics *precision*, *recall* and *F-Measure* were calculated. We selected the respective DIVISOR equals 8 and the multiplication FACTOR equals 2 that resulted in the highest F-Measure.

4.1.4 Page filtering mechanism

Given a set of n web pages as input, filtering is performed through the pre-processing steps, removing outlier pages and selecting pages with common method calls, which are explained below.

Pre-processing. The first pass in this step is to obtain the method calls present in the source code of each of the n web pages given as input, using a simple Java regular expression: `.*\(.*\)`. For example, the following Java Swing API methods are recognized by the regular expression: *getFirstRow()*, *getColumnName(column)*, *getValueAt(row, column)* and *setPreferredWidth(100)*. A filter was implemented to remove the method calls found within page scripts and style tags. We implemented 15 rules, considering the opening and closing of script and style tags. For example, when processing the page’s source code, if a script opening tag is found, then the code following that tag is disregarded until the corresponding closing tag is found. For the sake of conciseness, we omit the complete rules in this thesis. Nonetheless, the details of the created rules are publicly available[1]. After preliminary tests, some extracted method calls still had attached unwanted characters or words. To address this problem a filter was created to remove method calls that contain any of the following words or characters: function, -webkit, jQuery, gt;alert, {, }, (,), ‘, ˆ, ’, / and \. Method calls containing only one character from the alphabet have also been removed, because generally such method calls are user-defined. For example, the $x()$ method, when obtaining the method name, results in x, then this method is removed.

[1] https://doi.org/10.5281/zenodo.6467629

We performed tests to check the effectiveness of regular expression to find method calls. We took the first 5 programming tasks shown in Table 12 and then used them as input to the Google search engine. For each programming task, we obtained the method calls from the first 3 pages returned from the search engine, manually. Then we calculate the metrics Precision, Recall and F-measure of the algorithm. The results show that the 15 pages analyzed have a median of method calls of 45, an average of 52.9, a maximum of 104, a minimum of 0 and a standard deviation value of 29.8. Of the 15 pages, 12 had precision and recall equals 1, the other pages had precision and recall, varying from 0.933 to 0.981, and from 0.959 to 0.971, respectively, demonstrating the effectiveness of the algorithm to find method calls.

Removing outlier pages regarding number of method calls. This step has as input the number of method calls returned in the previous pre-processing step. Given the number of occurrences of method calls for each page, the average number of occurrences for n pages is calculated. Pages that have few method calls or a large number of method calls in relation to the average number of occurrences of method calls are removed. The lower and upper limits were defined in Subsection 4.1.3. Pages that have the number of occurrence of method calls greater than the lower limit and less than the upper limit are selected in this step. Generally, pages with few method calls in their solutions tend to not have code examples that solve the problem queried by the developer, and pages with a large number of method calls tend have complex and large code examples.

Selection of pages sharing common method calls. To obtain pages that possibly have solutions focused on the user query, we propose to apply a clustering algorithm using a proxy metric based on the sharing of common method calls between the pages. The purpose of this algorithm is to aggregate pages that share the same method calls in the same cluster. Our hypothesis is that clusters that have pages with similar solutions are more focused on user query. The instances given as input for this algorithm are the pages, the attributes are the method calls, and the attribute values are the number of occurrences of the respective method calls in the instance (page). The number of clusters k generated by the algorithm needs to be defined as input. We define as $k = numberOfInstances\ /\ d$. We calibrate the approach calculating a sub-optimal value for d, varying its value from 1.5 to 4 (0.5 in 0.5). We use the same queries defined in Section 4.1.2 and collected the respective result pages, and apply the clustering algorithm with the above different number of clusters k. For each of them, we bootstrap the filtering of clusters with solutions more focused on user's query, and calculate the metrics Precision, Recall and F-Measure. The best F-Measure was obtained for $d = 2$.

During the calibration tests of the approach parameters shown in Section 4.1.2, we observed that pages containing a wide variety of method calls, result in a large vector of attributes, which can impair the quality of the generated clusters. To work around this problem, before executing the clustering algorithm, an attribute selection algorithm is first applied to obtain only relevant attributes. After the attribute selection, the vector of attributes is reduced and clusters of better quality can be generated.

To select the cluster that contains more pages with common method calls, we calculated for each cluster the number of unique occurrences of method calls existent in the different pages of the cluster. The cluster with the highest value is selected.

Next, we show an example of the the clustering selection procedure. Assume the user enters the query "*how to create table java swing*" as input to the approach. To make the example simpler let us consider only the first 7 pages returned by the search engine. Table 9 shows the pages and method calls within them. Method calls that are present on at least two pages in the same cluster are highlighted in bold. We can observe that Cluster 2 has pages with solutions more focused on the user's query, since the pages have basic method calls for creating a table using JAVA and the Swing API. Cluster 1, on the other hand, has pages with solutions that have lost their focus a little, as there are methods that could be omitted when creating a table. For example, the BorderLayout() method is related to table layout. Cluster 1 also has other API (AWT) method calls that could be replaced by Swing API methods, for example, the GridLayout(), WindowAdapter() and FlowLayout() methods. The pages present in Cluster 3 also have low focus. For example, Page 2 has methods related to database manipulation, which indicates that the solution, in addition to creating a table, it also addresses database-related issues that are not related to the user query. Page 6, on the other hand, is related to the creation of a table that can be edited by the user, in addition to having other components such as JComboBox that is not related to the user query. Table 10 shows the clusters with their respective method calls that occur on at least two pages and the sum of those occurrences, showing that Cluster 2 is selected as it has the largest sum of occurrences. Because pages with solutions possibly more focused on the user's query are obtained by selecting the cluster containing more pages with common method calls in their solutions.

4.2 Study Setting

This subsection aims to show: 1) the definition of the baselines used to compare the results obtained; 2) the definition of queries used in the evaluation of approaches will also be shown; 3) the process used to build the ground-truth to evaluate the pages returned by the Google search engine.

4.2.1 Definition of baselines

The evaluation of the previous mining mechanisms will be conducted through a comparison with the baseline (Google) and variations of the approach. The purpose of comparing different variations of the approach is to show the effectiveness of the different

Table 9 – Example of application of the clustering algorithm on the pages returned by the search engine.

Pages	Method Calls on the Page	Cluster
Page 1	**JFrame()**, setSize(), **setLayout()**, GridLayout(), addWindowListener(), WindowAdapter(), windowClosing(), JLabel(), **JPanel()**, FlowLayout(), **add()**, **setVisible()**, setText(), **JTable()**, **JScrollPane()**	Cluster 1
Page 2	super(), forName(), getConnection(), createStatement(), executeQuery(), next(), getString(), println(), **JTable()**, **setSize()**, **setVisible()**	Cluster 3
Page 3	**JFrame()**, **JTable()**, setBounds(), **JScrollPane()**, **add()**, **setSize()**, **setVisible()**	Cluster 2
Page 4	**JTable()**, setFillsViewportHeight(), **JScrollPane()**, setPreferredSize(), Dimension(), **setLayout()**, BorderLayout(), **add()**, **JPanel()**, setOpaque(), **JFrame()**, setDefaultCloseOperation(), setContentPane(), **setVisible()**	Cluster 1
Page 5	**JFrame()**, **setTitle()**, **JTable()**, **JScrollPane()**, **add()**, **setSize()**, **setVisible()**	Cluster 2
Page 6	JFrame(), Integer(), Boolean(), EditableTableModel(), **JTable()**, createDefaultColumnsFromModel(), JComboBox(), getColumnModel(), getColumn(), setCellEditor(), DefaultCellEditor(), add(), JScrollPane(), **setSize()**, **setVisible()**, getRowCount(), getColumnCount(), getValueAt(), getColumnName(), getColumnClass(), getClass(), isCellEditable(), setValueAt()	Cluster 3
Page 7	**JTable()**, **add()**, **JScrollPane()**, **setTitle()**, setDefaultCloseOperation(), pack(), **setVisible()**, invokeLater(), Runnable(), run()	Cluster 2

filters added in each of the approaches. The main characteristics of the baseline and the proposed approaches are shown in Table 11.

The JG (Just Google) baseline is just a raw Google search, getting the first returned pages. The GOR (Google + Outlier Removal) approach consists of applying the JG approach and then executing the filter for removing outliers pages explained in Subsection 4.1.4. The GORCTO (Google + Outlier Removal + Clustering + Total Occurrence of Method Calls) approach consists of applying the GOR approach and then executing the clustering algorithm, where the attribute values are formed by the total sum of occurrences of each method call on the processed page (instance), for example, if the PHP method *isset* occurs 4 times on page X, then the value of the attribute *isset* for page X will be 4. The GORCUO (Google + Outlier Removal + Clustering + Unique Occurrence of Method Calls) approach consists of applying the GOR approach and then executing the clustering algorithm, where the values of the attributes are formed by the presence of occurrence of method calls on the page, that is, if a given method call occurs at least once on the page, the value of its attribute will be 1, otherwise 0.

To evaluate the baseline and the three variations of the approach, we collect their results on the n pages returned from the queries defined in the next subsection. For each approach, precision, recall and F-Measure are calculated and compared.

Table 10 – Clusters with their respective method calls that occur on at least two pages and the sum of those occurrences.

Cluster	Method Calls	Occ. > 1	Sum of Occ.
Cluster 1	JFrame()	2	14
	setLayout()	2	
	JPanel()	2	
	add()	2	
	setVisible()	2	
	JTable()	2	
	JScrollPane()	2	
Cluster 2	JTable()	3	**19**
	JScrollPane()	3	
	add()	3	
	setVisible()	3	
	setSize()	3	
	JFrame()	2	
	setTitle()	2	
Cluster 3	JTable()	2	6
	setSize()	2	
	setVisible()	2	

Table 11 – Main filtering features of baseline and approaches

Acronym	Features
JG	**J**ust **G**oogle (Baseline)
GOR	**G**oogle + **O**utliers **R**emoval
GORCTO	**G**oogle + **O**utliers **R**emoval + **C**lustering + **T**otal **O**ccurrences of Method Calls
GORCUO	**G**oogle + **O**utliers **R**emoval + **C**lustering + **U**nique **O**ccurrences of Method Calls

4.2.2 Definition of queries for assessment

To compare the baseline and the approaches, the first step is to define queries related to the programming task as follows:

1. We analyzed a Stack Overflow survey[2] to choose a popular programming language, and JAVA was the choice.

2. In order to choose the APIs that are used in the programming tasks, we adopted the criterion of choosing the most popular APIs (number of questions) of the JAVA language in Stack Overflow[3]. We checked the JAVA language API tags on Stack

[2] https://insights.stackoverflow.com/survey/2020#most-popular-technologies
[3] https://stackoverflow.com/tags

Overflow and ranked them by popularity. The APIs were categorized by application domain and one API was selected from each domain. When there were several APIs in each domain, we chose the most modern ones.

3. To select the programming tasks, we adopt the following criteria. For each selected API selected in the previous step, in the Coursera[4], a query was made with the name of the language (JAVA) + the name of the API. For each query, we analyzed the course summaries and selected the first programming task found that manages a product or a semi-product. Coursera was chosen because it is the largest online course platform on the Web, with respect to the number of users [17].

4. The queries to be executed in the search engine were built as follows: "how to" + programming task + programming language + API. We add the prefix "how to" before the programming task in order to find the pages that are related to how to implement the programming task consulted, unlike pages of the type of debug-corrective. Regarding the number of tokens present in the programming task that we defined, there is an average of 3.3 tokens per query. We tried to choose a small number of tokens per task, but expressing the intention of the task and with a number of words commonly used by developers (three words on the median), as shown in the work of Hora (2021b). The Table 12 shows the defined queries.

Table 12 – Selected APIs, application domains and queries.

APIs	**Domains**	**Queries**
Spring	Web Development	how to upload image java spring
JPA	Persistence	how to implement crud operations java jpa
Selenium	Web Testing	how to search a product item on an e-commerce web application java selenium
JavaFX	Desktop UI	how to implement menu java javafx
Tensorflow	Machine Learning	how to build a neural network java tensorflow
OpenCV	Computer Vision	how to classify image java opencv
JUnit	Unit Testing	how to implement matchers test java junit
Jackson	Structured Data Proc.	how to process JSON data java jackson
libGDX	Game Engine	how to implement animation java libgdx
OpenGL	Computer Graphics	how to draw 2D objects java opengl

The queries are executed on the Google search engine and the first 20 pages are obtained for each query. We process the content of these 20 pages (for each query) according each baseline procedure, and then evaluate them using a ground-truth defined below. We carried out a preliminary study to choose the number of pages that would be given as input for the approaches. We varied the number of pages (from 3 to 20 pages) and found that from 15 pages the results begin to converge. So, we decided to add a safety margin and evaluate the first 20 pages. As Google considers the history of previous searches performed by the user as a criterion for selecting the pages returned[5], we executed the queries using the browser in private mode without a logged account.

[4] https://www.coursera.org
[5] https://www.google.com/search/howsearchworks/how-search-works/ranking-results/

4.2.3 Ground-truth

This subsection shows the steps to build the ground-truth. To evaluate the pages returned by search engines, we defined the criteria presented in Subsection 4.2.3.2, which were based on the quality indicators and dimensions discussed in the next subsection.

4.2.3.1 Evaluation of the Pages Returned by the Search Engine

Arthur and Stevens [18] defined four quality indicators (accuracy, completeness, usability and expandability) for evaluating software documentation. Accuracy is defined as the consistency between the code and all code documentation, for all requirements. A documentation is complete if all the required information is present. Usability is defined as the suitability of the documentation regarding to the ease with which one can extract needed information. Expandability is the capability of the documentation to be modified in reaction to changes in the system. Although, these quality indicators are not completely suitable for evaluating the pages related to software development returned by search engines, as they were created for evaluating software documentation, they may serve as starting point to compose evaluation criteria related to the quality of the pages returned by search engines.

Smart [19] proposed three dimensions (easy to use, easy to understand and easy to find) to assess the quality of the documentation. Easy-to-use dimension is related to the ease of users to complete tasks related to their work through documentation, which must be accurate and include all and only the essential parts. The easy-to-understand dimension is related to the documentation's ability to be unambiguous, but to have appropriate writing styles, using examples and metaphors. The easy-to-find dimension is related to the ability to organize the documentation, where it must be coherent and make sense to the user, the information must be retrieved quickly and must be visually effective.

In order to evaluate the quality of the pages returned by search engines, in the next subsection, we defined three criteria (degree of focus, code examples and solution size) based on the quality indicators of the work of Arthur and Stevens [18] and the dimensions for the evaluation of documentations of the work of Smart [19], discussed previously. We consider relevant pages those with solutions that have a high or very high degree of focus, code examples and solution size not far from the average size of the solutions present in the first 20 pages returned by the search engine. In the next subsections, we present how we defined such thresholds.

4.2.3.2 Criteria for Ground-truth Construction

To assess the pages returned by the search engine, we use the following criteria.

Degree of focus. It consists of assessing how the solutions are related to the programming task searched by the user:

1. Very low. The solution is not related to the search, i.e., it contains implementations of other features that are not semantically related to the search.

2. Low. The solution has some connection with the search performed by the user, i.e., it contains implementations of several features that are not related to the search.

3. Neutral. When it is not possible to evaluate the page for this criterion.

4. High. The solution is related to the search, however, it has minor focus deviation, presenting implementations of few other features that are weakly related to the search.

5. Very high. The solution is completely related to the search carried out by the user, i.e., the present features in the solution teach what the user wants to learn.

This criterion is based on the *completeness* quality indicator of the work of Arthur and Stevens [18], since for having a high degree of focus, a page must have all the information required for the solution of the programming task expressed by the user query. This criterion is also based on the easy-to-use dimension of the work of Smart [19], since for having a high degree of focus, a page must have a precise solution, that is, *all* and *only* the essential parts.

Code examples. The pages have a binary value "*yes*" or "*no*" for this criterion, depending on containing or not code examples in their solutions. In this work we are focusing on solutions where the developer can reuse source code, since most developers look for code reuse when searching in search engines. So for the purpose of this work, pages that do not have code examples are not interesting for developers who are looking for a solution to a programming task searched on search engines [9].

Solution size. To evaluate this criterion, we visually analyzed the size of the solutions of the 20 pages returned by the search engine, considering the number of source code lines. After that, according to that general visualization, we evaluated the solution of each page using the possible values for the criterion:

1. Very small. Far below the observed average size.

2. Small. Marginally below the observed average size.

3. Neutral. When it is not possible to evaluate the page for this criterion.

4. Large. Marginally above the observed average size.

5. Very large. Far above the observed average size.

4.2.3.3 Ground-truth construction

The pages in the ground-truth are classified as "Selected" or "Not-selected", considering the criteria previously defined and the Equation 3, where *ce*, *ss* and *dof* represent the code examples, solution size and degree of focus criteria, respectively.

$$selectedPages \equiv (ce == \text{“}yes\text{”} \wedge (ss == 2 \vee ss == 4) \wedge dof \geq 4) \tag{3}$$

Pages belonging to the "Selected" class are considered adequate solutions for the programming task. The pages classified as "Selected" must have code examples of how to implement the programming task surveyed in the search engine, so they must have the value "yes" for the Code Examples criterion. For the Degree of Focus criterion, the pages must have a value greater than or equal to 4, indicating that features in the solution provide what the user wants to learn. Pages with values less than or equal to 2 for this criterion should not be selected, because they contain code or content that is not related to the query. Regarding to the Solution Size criterion, pages with a value of 2 or 4 are selected, as they present solutions that are close to the average number of method calls found on the n pages obtained by the search engine. The pages that have the values 1 or 5 for this criterion are far from the average, so they are not selected. Pages rated with value three should not be selected because they could not be evaluated. For example, for the Solution Size criterion, pages that do not have source code are rated with three.

In the the construction of the ground truth, we evaluated the first 20 pages returned by Google, for the 10 queries shown in Table 12, totaling 200 evaluated pages. Regarding the chosen number of evaluated pages, we proceeded with a sequential evaluation of the best ranked pages returned by Google, and observed that typically adequate solutions were found up to the 15th position, so we decided to add a safety margin and evaluate up to the 20th position. In addition to our evaluation, another researcher evaluated 40 of the 200 pages. The evaluation of 16 pages was identical. The evaluation of 18 pages had small divergences, but they did not change the actual binary classification of the page. The evaluation of 6 pages had divergences in the values of the evaluated criteria, which were discussed to reach consensus and helped to improve and assess the confidence on the final rating of the 200 pages, which mostly mitigates a threat to construction validity.

About the number of pages evaluated, we must argue that although we provide quantitative data analysis, the ground-truth construction is a qualitative work, so a larger scale manual construction is typically prohibitive. In total, we evaluated 200 pages (20 pages per query), as this evaluation was thorough and careful, it took months to perform and curate. The 200 assessed pages number can be considered sufficient to support our conclusions.

The data obtained in this study is available at a Zenodo repository[6].

The following research questions are answered at the end of this study:

[6] https://doi.org/10.5281/zenodo.6467629

RQ4: *To what extent does Google's search engine return relevant pages to software developers?*

RQ5: *To what extent applying outlier pages removal filter w.r.t. the page size help to remove irrelevant pages returned by search engines?*

RQ6: *To what extent applying clustering algorithms w.r.t. the total method calls in returned pages help to remove irrelevant pages returned by search engines?*

RQ7: *To what extent applying clustering algorithms w.r.t. unique occurrences of method calls help to remove irrelevant pages returned by search engines?*

RQ8: *To what extent applying the filters proposed in this work improve the ranking quality of pages returned by search engines?*

4.3 Results

This section shows the results for the approach assessment using the methodology shown in Section 4.2. The baseline and the three approaches are compared for each query to the ground-truth.

Figures 6, 7 and 8 show the evaluation of the 20 pages returned by the Google search engine, for each of the ten queries. The x-axis represents the possible values for the given criterion. The y-axis represents the number of pages. The red, orange and yellow bars of the boxplots represent the undesirable values for the evaluated pages, since the pages evaluated with these values have undesirable characteristics for software developers. The green and blue bars of the boxplots represent the desirable values for the pages, that is, those that have relevant characteristics for the developers.

4.3.1 Study on the pages returned by the Google search engines

In this section, we will answer the fourth research question (RQ4). We observe in Figure 6 that several pages returned by Google's search engine have no focus, that is, the solutions in theses pages are not related to the query (as shown in the red bar) or have low focus, that is, those solutions are little related to the query carried out by the user (as shown in the orange bar). Moreover, these pages contain implementations of features that are not related to the programming task. The yellow bar shows the pages (25 % of the total pages evaluated) that could not be evaluated for the Degree of Focus criterion due to the absence of source code.

Figure 7 shows that for all evaluated queries, some pages have solutions that are much larger than the average in relation to the other pages (red bar). These pages add up to 12% (24 pages) of the total evaluated pages. Pages with large solutions usually have no focus. Moreover, 91.7% of the pages with very large solutions have low or very low degree of focus. The approach proposed in this work aims at removing pages with solutions far

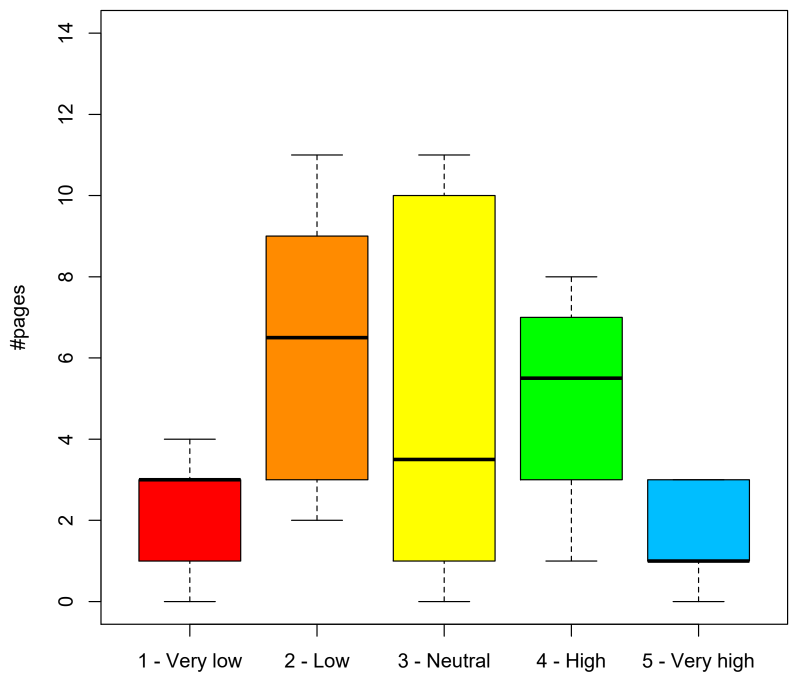

Figure 6 – Results of the evaluation of the pages returned by the search engine for the degree of focus criterion.

above the average, which would improve the result. The orange bar shows that a small portion of the evaluated pages (4% of the total pages evaluated), have the value "Very small". Pages that have very small solutions in relation to the average also tend to lose focus. We observed that 87.5% of the pages that have very small solutions have a low and very low degree of focus. The proposed approach also filter these pages. The yellow bar shows the pages (25 % of the total pages evaluated) without source code.

Figure 8 shows that for all evaluated queries, there are pages returned by the search engine that do not have code examples (25% of the total pages evaluated). This result also shows a variation from 0 to 11 pages without source code depending on the evaluated query, revealing that depending on the query, around half of the pages do not have source code among the first 20 pages returned.

When applying the Equation 3 of Subsection 4.2.3 to the values of the criteria shown in Figures 6, 7 and 8, only 31% of the pages returned by the search engine were selected, that is, pages that tend to be considered good solutions for developers. The rest of the solutions (69%) are not interesting for the developer who is learning a programming task, as they have undesirable characteristics, such as: no code examples, low degree of focus or/and pages with solution sizes that are far from average. This shows the importance of developing filters to remove pages with such characteristics.

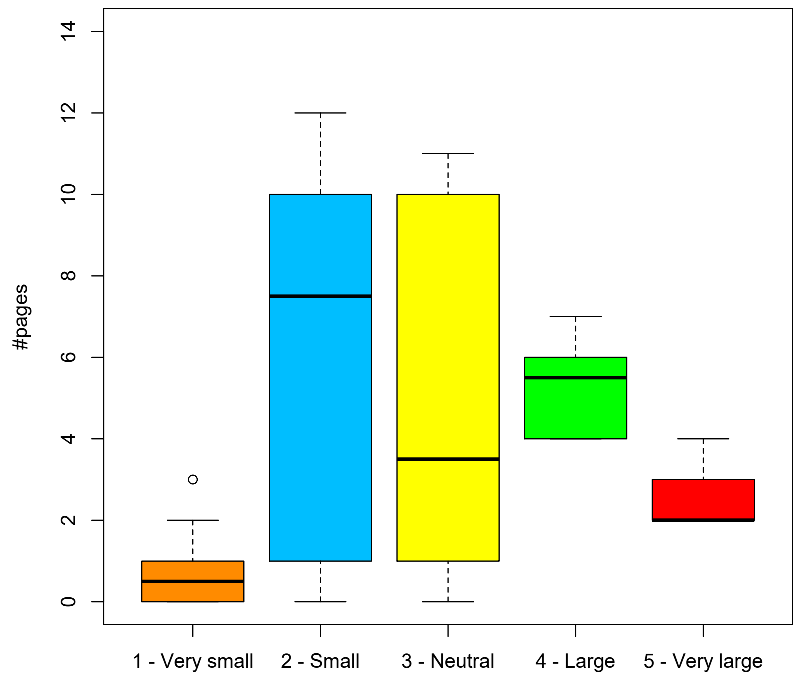

Figure 7 – Results of the evaluation of the pages returned by the search engine for the solution size criterion.

Answer to RQ4: *To what extent does Google's search engine return relevant pages to software developers?* Only 31% of the total pages returned by Google's search engine have relevant solutions for software developers. The rest of the solutions have undesirable characteristics, such as: no code examples, low degree of focus or/and pages with solution size that are far from average (outlier pages). 25% of the total evaluated pages, do not have code examples. Only 33% of the analyzed pages have high or very high degree of focus on the solutions sought by the user. 16% of the total evaluated pages, are outlier pages.

4.3.2 Effectiveness of Page Filtering

In this section, we answer the fifth, sixth and seventh research questions (RQ5, RQ6 and RQ7). This subsection shows the results of the comparison of the baseline and the three approaches (JG, GOR, GORCTO and GORCUO) defined in Section 4.2. Having the result of the selection of pages obtained in the previous subsection, for each of the approaches, the metrics were calculated incrementally in the number of pages: Precision, Recall, F-Measure. The incremental evaluation of the metrics was carried out as follows: starting with 3 pages, the metrics were calculated, then it was increased to 4 pages and the metrics were calculated again, and so on, up to 20 pages.

Figure 9 shows the graph comparing the precision of the baseline and the three evaluated approaches. The yellow, orange, green and blue bars represent the JG, GOR, GORCTO and GORCUO approaches, respectively. On the X axis of the graph are the

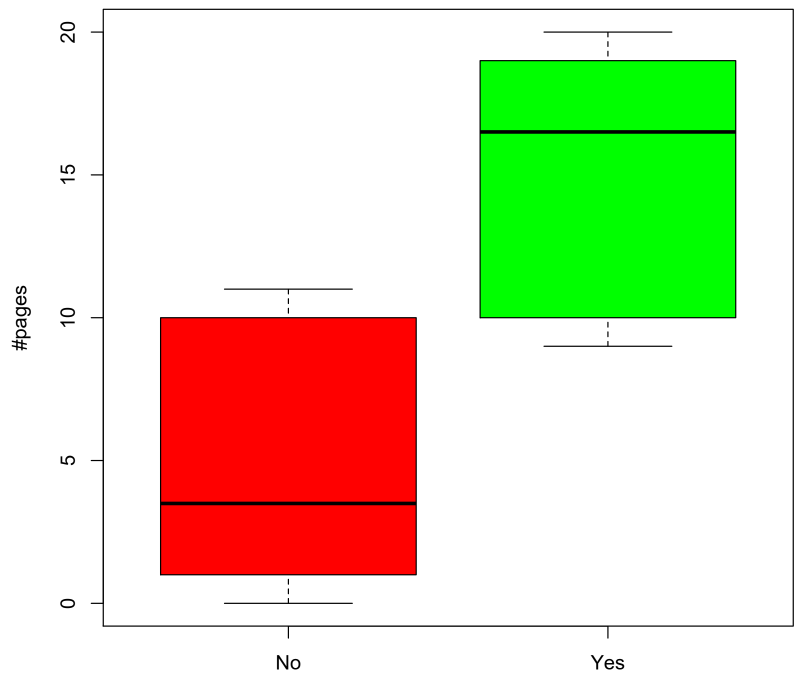

Figure 8 – Results of the evaluation of the pages returned by the search engine for the code example criterion.

number of pages given as inputs for the approaches, which varies from 3 to 20 pages, increasing by one. On the Y axis are the values of the metric precision, obtained by the given approach when being executed having as input the ten queries evaluated in this work. The graph shows that the JG (Just Google) baseline has low precision, where the medians of the yellow bars are below 40%, for most pages given as input. This reinforces the need for filters to remove irrelevant pages returned by the search engine. The GOR approach, which implements the filter for removing outliers pages, which manages to remove irrelevant pages, increases the precision for all page numbers given as inputs, compared to JG. The graph shows that the GORCTO approach is very unstable, with a very large variation in precision, for all page numbers given as inputs. The GORCUO approach, on the other hand, has bars with higher medians compared to the other three approaches, with the best results for the precision metric, being very effective for removing irrelevant pages. The GORCUO approach shows best results in the intervals 15 to 19 of page numbers given as input, with the best result with 16 pages given as input.

Figure 10 shows the results of the recall metric for the four approaches. For the JG approach, the recall will always be 100%, as this approach is used to obtain the 20 pages used in the evaluation. The graph reveals that even after applying the filter for removing outlier pages, the GOR approach has 100% recall for most of the page numbers given as input (except for numbers 4, 5, 19 and 20). This indicates that the outlier page removal filter is effective in removing only irrelevant pages. The graph shows that the GORCTO approach has the worst results for this metric, with great variability in its values and low median for most of the page numbers given as input. The GORCUO approach has

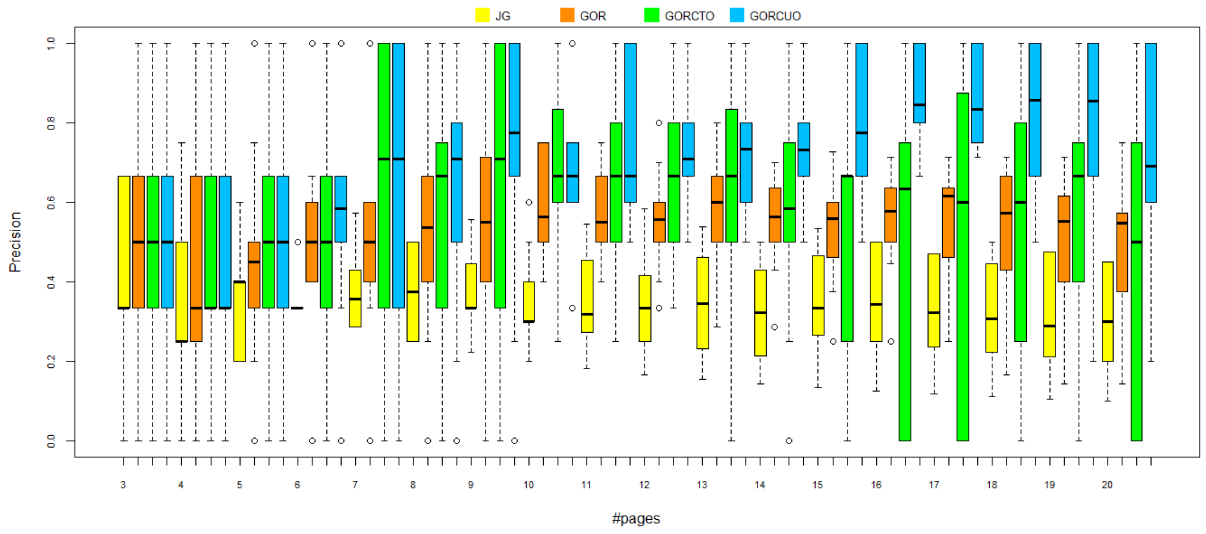

Figure 9 – Precision of the baseline and the three approaches increasing the number of pages given as input.

a median of around 70% for this metric. Even removing a portion (around 30%) of the relevant pages, the rest of the relevant pages are already sufficient for developers to obtain the desired solutions. Since the GORCUO approach is very efficient at removing irrelevant pages, as shown in the precision graph in Figure 9, this approach has been proven to be the best option for developers to obtain relevant pages.

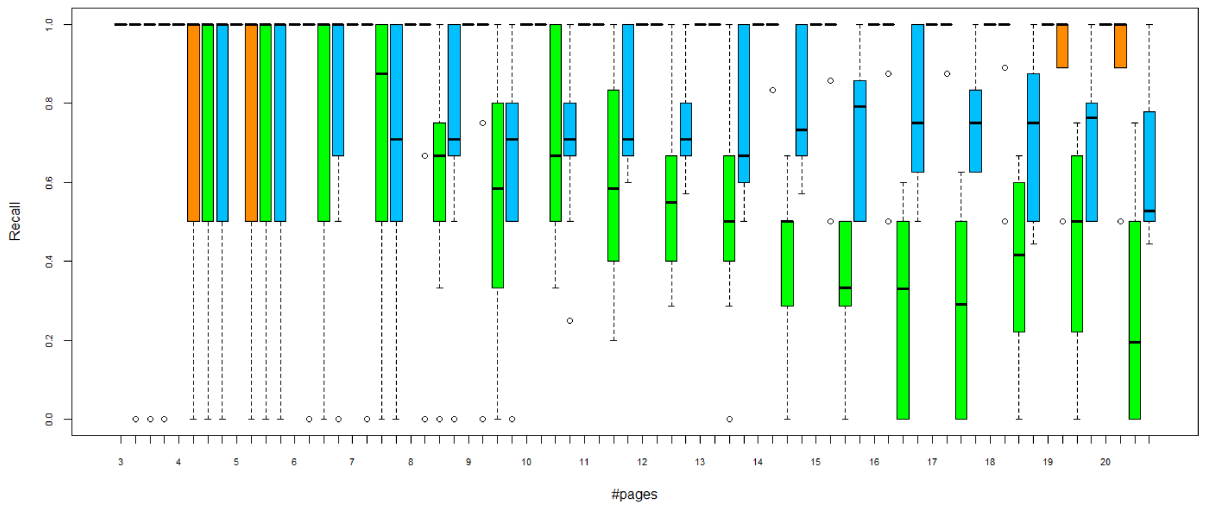

Figure 10 – Recall of the baseline and the three approaches increasing the number of pages given as input.

For the F-Measure metric, Figure 11 shows that among the four approaches, GORCUO (blue bar) performed better. The GORCTO approach (green bar) varies widely for the vast majority of the landing pages, in addition to having a low overall median. The GOR approach (orange bar) shows better results than the GORCTO approach. The JG approach (yellow bar), which uses only the Google search engine, has a low overall average.

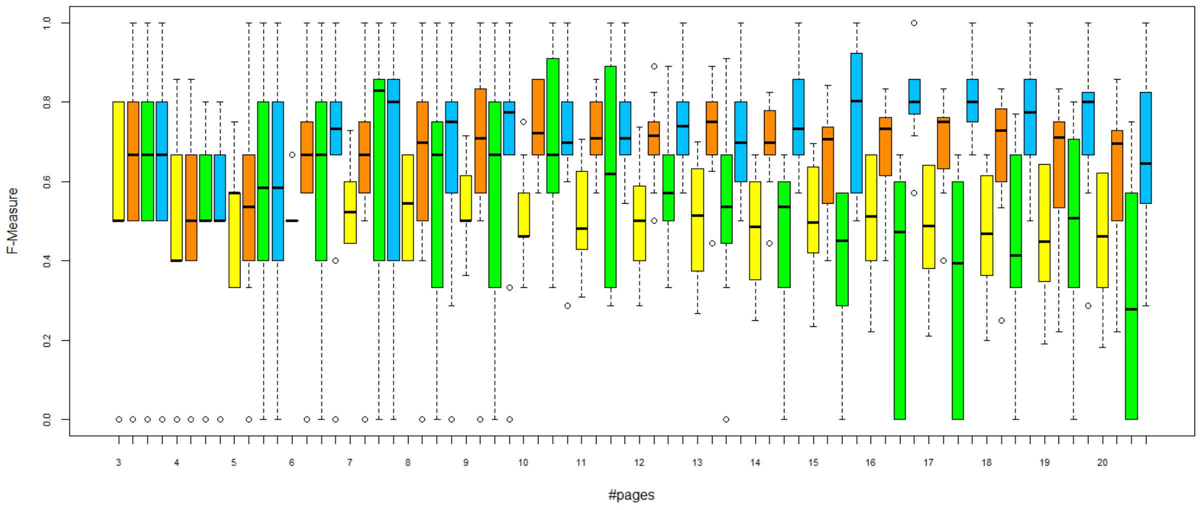

Figure 11 – F-Measure of the baseline and the three approaches increasing the number of pages given as input.

Answer to RQ5: *To what extent applying outlier pages removal filter w.r.t. the page size help to remove irrelevant pages returned by search engines?* The results shown in the F-Measure reveal that the GOR filter, which removes outlier pages, helps remove irrelevant pages for software developers. For all page numbers given as input, the filter has F-Measure higher than Google Search Engines (JG), except for page number equal to 5, where the median of the JG is slightly above the median of the GOR filter.

Answer to RQ6: *To what extent applying clustering algorithms w.r.t. the total method calls in returned pages help to remove irrelevant pages returned by search engines?* The results of the Precision, Recall and F-measure reveal that the approach (GORCTO) that uses the clustering algorithm having as input the total occurrence of method calls on the pages returned by the Google search engine, has very large variability in results, for most page numbers given as input to the approach. In addition, the GORCTO approach removes many pages relevant to software developers. So, this approach is not recommended for removing irrelevant pages.

Answer to RQ7: *To what extent does applying clustering algorithms w.r.t. unique occurrences of method calls help to remove irrelevant pages returned by search engines?* The F-Measure shown in Figure 11 reveal that the GORCUO filter is effective to remove irrelevant pages for software developers. For all page numbers given as input, the filter has higher F-Measure than Google search engines (JG). For example, for the number of pages equal to 17, the median of the GORCUO filter is around 0.8, while the Google search engine has a median around 0.5, for the same number of pages given as input.

4.3.3 Ranking Quality

In this section, we will answer the eighth research question (RQ8). Table 13 shows the results of metrics Hit@K, Recall@K, MRR@K, MAP@K and NDCG@K, for K=5, considering the 20 pages returned by the search engine as input. As the GOR, GORCOT and GORCOU approaches select pages applying their filters, the final number of pages is less or equal 20. If the respective number of pages selected by these approaches is less than 5, the number of pages is completed up to five with the first non-selected pages of the JG approach, so we can calculate the metrics for K=5.

Regarding the results of the Hit metric, shown in Table 13, we can observe that, for the 10 evaluated queries, the approaches JG, GOR and GORCOU manage to find at least one relevant page among the first 5 pages returned. The GORCOT approach, for two queries, could not find any relevant page among the first five returned pages.

The Recall@K metric shown in Table 13 was calculated considering the total number of relevant pages found in the first 20 pages returned by the JG approach. The table shows the results of this metric for K=5, that is, for the first 5 pages returned by the approaches. Of the baseline and the three evaluated approaches, GORCUO obtained the best results. So, after applying the filters, more relevant pages are brought to the top-5 ranking using this approach. The GOR approach had the second highest value for this metric, which indicates that after applying the filter that removes outlier pages, the top-5 ranking has more relevant pages than the JG approach, which uses only the ranking returned by Google. The GORCOT approach, which uses outlier removal as the GOR approach, had results very close to the JG approach, this indicates that after the application of the clustering from the GORCOT approach, the top-5 of the ranking worsens in relation to the top-5 of the GOR approach. This is due to filter instability of the GORCOT clustering.

Table 13 shows the results of the MRR, MAP and NDCG metrics, for K=5. As we can observe, the GORCUO approach presents better results for all metrics (MRR, MAP and NDCG). These results show that, after applying the filters proposed in this approach, we obtain a top-5 ranking of better quality.

Table 13 – Comparison of Hit, Recall@K, MRR, MAP and NDCG metrics results, for K=5, considering the 20 pages returned by the search engine as input, for the 10 queries evaluated.

	Hit@K	**Recall@K**	**MRR@K**	**MAP@K**	**NDCG@K**
JG	**1**	0.347	0.595	0.558	0.663
GOR	**1**	0.472	0.658	0.624	0.739
GORCOT	0.8	0.378	0.72	0.657	0.655
GORCUO	**1**	**0.624**	**0.867**	**0.839**	**0.897**

Answer to RQ8: *To what extent does applying the filters proposed in this work improve the ranking quality of pages returned by search engines?*
As we can observe in Table 13, the filters of the GOR and GORCUO approaches improve the top-5 ranking quality, for the 3 analyzed metrics (MRR, MAP and NDCG). The GORCUO approach performed better than GOR. The GORCOT approach, on the other hand, improves the ranking quality for the MRR and MAP metrics, compared to the JG approach. For the NDCG metric, the GORCOT approach has a slightly lower value than the JG approach. So, the final recommendation is that the GORCUO approach is the best one to improve ranking of programming solutions from search engines.

4.4 Discussion

The results of the evaluation carried out on the first 20 pages returned by the Google search engine, for the 10 analyzed queries, reveal that only 31% of the evaluated pages have relevant solutions for software developers. The rest of the solutions have undesirable characteristics, such as: no code examples, low degree of focus or/and pages with solution size that is far from average. These results show the importance of developing filters to remove irrelevant pages for developers.

Getting relevant pages in search engine results is a difficult task for developers, as they may have to inspect those pages manually, which obviously takes more effort to get the desired result. We observed that pages with high or very high degree of focus are scattered in the ranking returned by search engines. This may be due to the fact that search engines consider generic features that are not customized for solutions related to software development [20]. For example, pages may contain high quality text content, but may lack source code examples. Moreover, pages may have source code, may contain high quality textual content, but the solution may not be focused on the query searched by the developer, that is, the solution may not solve directly the user's problem. Some implications and possible consequences of using the ranked list of pages as-it-is are: *1) the user stops browsing at a solution ranked in the first positions, but with low degree of focus.* A consequence is that the user may spend more time abstracting the desired solution from low-focus content, rather than looking for more focused solutions in the lower ranking positions. *2) the user continues to look for solutions with better focus on lower ranking positions.* A consequence is that the developer has to read several pages until finding one with content focused on the desired solution, demanding more effort. As we can observe in this study, there are pages with content with a high degree of focus only in the 15^{th} position of the ranking, many developers may not examine the content of the pages in deep positions of the ranking. Therefore, these developers may not find a good solution that is at the bottom of the ranking.

Another feature that makes pages returned by search engines irrelevant to software developers is the absence of source code. The results show that some of the pages returned by search engines do not have source code. Probably, source code is not a major priority for search engines when ranking pages. A developer looking for code examples would spend considerable time visiting pages without source code.

The results showed that the GOR filter, which removes outlier pages, is effective in removing irrelevant pages returned by search engines. This is due to the filter removing pages that do not have source code or with small source code in relation to the average size of the solutions present in the pages returned by the search engine. The filter also removes those pages with large solution size relative to an average size. As shown in the results of this work, pages with solutions that are too small or too large compared to the average size of the solutions returned by the search engine have low or very low degree of focus, which makes the solution irrelevant for developers. For example, for the query *"how to implement menu java javafx"*, evaluated in this study, one of the pages returned by the engine just defines the SeparatorMenuItem class, without a practical example of use. Since the page has only three method calls that are related to the class's constructor method, that page is removed by the GOR filter, as the number of occurrences of methods is lower than the average occurrence of method calls for this programming task, having an average of 66.8 calls. For the same query, another page returned by the search engine has an extensive solution, with 147 method calls, which addresses features such as radio button and checkbox in the solution. These features make the solution on that page lose focus. The GOR filter also removed this page.

The results showed that the GORCTO filter has a large variability in the results for the different queries given as input. This filter uses a clustering algorithm where the attribute is the total method calls found in the page solutions, which makes it very unstable. So, pages that have been clustered in the same cluster may have solutions that lose focus. For example, for the query *"how to implement crud operations java jpa"* evaluated in this study, two pages (A and B) were clustered in the same cluster, even though one of the pages contained a solution that lost focus. These pages were inadequately clustered in the same cluster due to the characteristic of the GORCTO approach of using the total hits of method calls as the attribute value for the clustering algorithm. For this case, pages A and B have some methods in common with occurrence value equal or very close, these methods help these pages to be in the same cluster. For example, the *begin* method occurs 4 times on page A and 4 times on page B. The *getTransaction* method occurs 10 times on page A and 9 times on page B. The *find* method occurs 4 times on page A and 4 times on page B. The *commit* method occurs 4 times on page A and 5 times on page B. However, on page B there are several methods that do not occur on page A. For example, the *getText*, *parseInt*, *removeById* and *showInputDialog* methods. These methods are related to the graphical user interface part of the solution. In other words, page B also addresses issues related to the graphical interface, losing the main focus of the user query, which is to implement CRUD operations. As cluster selection considers the sum total of occurrences of method calls that occur on at least two pages in the clusters, pages with low focus are placed in the same cluster as pages with high focus, as a portion of the method calls on those pages occur multiple times.

The results showed that the GORCUO filter is effective in removing irrelevant pages, since the filter uses a clustering algorithm where the attribute is a single occurrence of each method call. This feature prevents interference from method calls that often occur in solutions from different pages, which reduces the quality of the generated clusters. For example, if the GORCOU approach was applied in the example of the previous paragraph, probably pages A and B would not be clustered in the same cluster, since the occurrences of the method calls would be counted only once. For example, the *getTransaction* method would have an attribute value equal to 1 for pages A and B, instead of 10 for page A and 9 for page B. In this way, this method would have no weight at the time of the agglomeration of the pages, that is, pages A and B would not be so close together because of this method call and other method calls with high occurrence numbers as attribute value.

The results showed that the GORCOU filter improves the quality of the top-5 ranking, as shown in the results of the MRR, MAP and NDCG metrics (Table 13). Even removing some relevant pages as shown in the Recall metric results (Figure 10), this filter is effective in removing irrelevant pages as shown in the Precision (Figure 9) and MAP metric results. This improvement in ranking quality is due to the use of a filter that uses a clustering algorithm, where the attribute is a unique occurrence of each method call. This algorithm manages to mine the relevant pages spread out in the ranking, bringing them to the top of the ranking, consequently, improving the ranking quality.

4.5 Threats to Validity

In this section, we show some threats to the validity of this study.

Internal validity. This kind of threat is about how sure we can be that the treatment actually caused the outcome. In this study, a threat to internal validity was the number of pages returned by the search engine considered in the study of mining approaches. A possible larger number of pages could influence the results. To minimize this threat, we conducted a preliminary study varying the number of pages (from 3 to 20 pages) and found that from 15 pages the results begin to converge.

External validity. This kind of threat is related to whether we can generalize the results outside the scope of our study. In this study, to threaten external validity, the results of the proposed approaches are limited to the JAVA programming language. The approaches may have different results with other programming languages.

Conclusion validity. This kind of threat is about how sure we can be that the treatment is really (statistically) related to the actual outcome. In this study, we constructed a ground-truth with 10 queries, analyzing 20 pages per query (200 pages in total). So, enlarging the number of queries would possibly impact the results. To mitigate this threat, we have carefully chosen queries in distinct API domains to cover a broad range of content.

4.6 Conclusion

In this chapter, we have shown that the Google search engine may return a non-negligible number of pages with low focus on the solution sought by the developer. The results also show that there are several pages returned with no code samples. Pages with these characteristics are not interesting for developers looking for a solution for their problem in hand, motivating our proposed approach to filter only relevant solutions for developers.

Our study compared the Google baseline and three different approaches for filtering relevant solutions. The proposed approach that removes outlier pages and applies a clustering algorithm with unique occurrences of method calls as attribute value obtained higher precision and F-Measure than the other three evaluated approaches. These results demonstrated the effectiveness of creating filters to improve the results returned by search engines.

Chapter 5

Related Work

This chapter aims to present the research related to the two main studies carried out in this thesis: 1) the study on the influence that the ranking quality exerts on the performance of developers when performing programming tasks, and 2) the study on the mining of relevant solutions for developers from search engine results.

5.1 On the Influence of the Ranking Quality for Developers

Rahman et al. (2018) investigated if general-purpose search engines like Google are an optimal choice for source code-related searches. They aimed at understanding whether the performance of searching with Google varies for code vs. non-code related searches. To carry out these investigations, the authors collected search logs from 310 developers that contain nearly 150,000 search queries from Google and the associated result clicks. To differentiate between code-related searches and non-code related searches, they built a model which identifies the code intent of queries. Leveraging this model, they built an automatic classifier for code and non-code related query. The authors noted that code related searching often requires more effort (e.g., time, result clicks, and query modifications) than general non-code search, which indicates code search performance with a general search engine is less effective. The results of this work show the importance of general purpose search engines for source code search, which motivated us to carry out a study on the influence of the ranked list of result pages on the performance of developers during programming tasks and the development of filters to improve the ranking quality returned by search engines.

Sadowski, Stolee e Elbaum (2015) conducted a case study related to how developers search for code using the Google search engine. The results of the study show that programmers search for code very frequently, conducting an average of five search sessions with 12 total queries each workday. The authors also show that programmers are generally seeking answers to questions about how to use an API, what code does, why something is failing, or where code is located. Based on the free text answers, the most common questions were about finding code samples, meaning the developers who participated in this study look for examples more than anything else. The results of this study also motivated us to carry out a study on the influence of the ranking quality on the performance of developers, since developers perform several queries on search engines during the day in search of code examples for solving day-to-day problems and these search engines can return irrelevant pages to developers, hindering their performance during the resolution of programming.

Keane, O'Brien e Smyth (2008) evaluated whether people are biased in their use of a search-engine, specifically, whether they are biased in clicking on those items that are presented at the top of the result list, returned by the search engine. To test this bias hypothesis, they simulated the Google environment systematically reversing Google's normal relevance-ordering of the items presented to users. The results showed that people do manifest some bias, favoring items at the top of result lists, although they also sometimes seek out high-relevance items listed further down the list. These results motivated us to better understand how developers behave when faced with a lower quality ranking returned in top-ranked positions of a search engine result.

Several other works have studied the impacts that search engines exert during activities related to software development. In the work of Fischer, Stachelscheid e Grossklags (2021), the authors showed the effects that Google search engines have on software security. They showed that the probability of an insecure result appearing in the top three is 22.78%, while only 9.19% secure ones. In the work of Hora (2021b), the authors conducted an empirical study to understand what developers search in the web and what they find. They found that queries performed by developers typically start with keywords, e.g. Python, Android, etc., they have three words on the median, tend to omit function words, and are similar among each other. Another observation was that minor changes to queries do not broadly affect Google search results, however some cosmetic changes can have a non-negligible impact. This study motivated us to investigate the impact that the ranking quality of the pages returned by the search engine has on the developer. Since the results show how developers make use of the Google search engine to search for content related to software development on the Web.

Cho e Roy (2004) studied the impact that search engines have on the evolution of the popularity of web pages. They analytically estimated how long it takes for a new page to attract a large number of web users, while searches return only popular pages at the top of search results. The results of this work show that search engines can have an immensely worrying impact on the discovery of new web pages. These results show that there may be new pages at the bottom of the rankings that have content that is relevant to the user. In the context of our work, there may be relevant solutions in lower positions on new pages. These results also motivated us to carry out a study on the influence that the ranking exerts on the performance of developers while performing programming tasks.

5.2 On Mining Relevant Solutions from Search Engine Results

Research in the field of mining search engine results is a practice that has given good results in different contexts. For example, in the work of Saraswathi e Vijaya (2013), they improve the quality of search engine results by identifying and removing spam links. In the work of Suneetha, Sameen Fatima e Shaik Mohd. Zaheer Pervez (2011), the authors aim at organizing web search results into clusters facilitating quick browsing options to the browser providing interface to results precisely. On the other hand, in the work of Suo, Zhang e Zhang (2009), the authors analyze the results of the search engines using a new summary approach which calculates the sentence weight utilizing the information of the distance between words. In this thesis, we propose an approach to improve the ranking quality of pages returned by the Google search engine through filters and clustering algorithm.

Several other researchers (LAU; HORVITZ, 1999; KIM; COLLINS-THOMPSON; TEEVAN, 2016; ZAHERA; EL-HADY; EL-WAHED, 2011; HONG; VAIDYA; LU, 2011; ZHUANG; CUCERZAN, 2006; AMIN; EMROUZNEJAD, 2011; CARAMIA; FELICI; PEZZOLI, 2004) have been working to improve search engine ranking results, as these search engines return many pages irrelevant to users, whatever area of expertise. This shows the importance of research related to this topic. Regarding the area of software development, appropriate solutions for developers are not always among the first pages ranked by search engines. They can be mixed with other pages with inadequate characteristics, such as: without code examples and little focus on the desired solution. It is a challenge to remove pages with these inappropriate characteristics. Xia et al. (2017) evaluate the quality of software development-related pages returned by search engines. They argue that several reviewers questioned that the quality of the content of the pages returned by search engines is low, which requires more time to find the desired content. The approach that we are proposing removes pages with inappropriate content for developers, selecting only the relevant pages.

Zhuang e Cucerzan (2006) address two common problems in search, frequently occurring with underspecified user queries: the top ranked results for such queries may not contain relevant documents to the user's search intent, and fresh pages may not get high rank scores for an underspecified query due to their freshness and to the large number of pages that match the query, despite the fact that a large number of users have searched for parts of their content recently. The authors argue that such problems can be solved if users' search goals are well understood, and propose a method called Q-Rank to effectively refine the ranking of search results for any given query by constructing the query context from the query logs. The results of the work evaluation show that Q-Rank gains a significant advantage over the current ranking system of a large-scale commercial Web search engine, being able to improve the search results' relevance for 82% of the queries, as reflected in the discounted cumulative gain numbers obtained. Furthermore, because Q-Rank is independent of the underlying ranking scheme, it is highly portable and can be easily integrated with any existing search engine system. Our work also addresses the problems mentioned above, but in the context of software development: results returned by search engines do not contain documents relevant to the user's intention and fresh pages may not be at the top of the ranking. Our approach differs from the approach of this work, because we apply filters to the pages returned by the search engine, while in this work, the authors propose a method for building a query from a query log.

Zahera, El-Hady e El-Wahed (2011) propose a method for query recommendation, which given a query submitted to a search engine, suggests a list of queries that are related to the user input query. The purpose of the approach is to improve the results returned by search engines. The method proposed by the authors is based on clustering processes in which groups of semantically similar queries are detected. The clustering process uses the content of historical preferences of users registered in the query log of the search engine. This facility provides queries that are related to the ones submitted by users in order to direct them toward their required information. The filters that we propose in this thesis also aim to improve the results returned by search engines, but our approach works with the contents present in the returned pages, unlike the above work, which decides for an approach of query recommendation that could be seen as complementary to ours.

Kim, Collins-Thompson e Teevan (2016) explore how crowd sourcing can be used at query time to augment key stages of the search pipeline. First, they investigate the use of crowdsourcing to improve search result ranking. When the crowd is used to replace or increase traditional retrieval components, such as, query expansion and relevance scoring, they observed that they could increase robustness against failure for query expansion and improve overall precision for results filtering. However, there is a limitation on these gains, due to the extra cost and time that the crowd requires. They find that using crowd workers to support rich query understanding and result processing appears to be a more worthwhile way to make use of the crowd during search. The current work proposed in this thesis also seeks to improve the results returned by search engines through the proposed filter.

Hong, Vaidya e Lu (2011) present a clustering approach based on a key insight, where search engine results can be used to identify query similarity. They propose a similarity metric for diverse queries based on the ranked URL results returned by a search engine. This is used to develop an algorithm for clustering queries. The approaches proposed in this thesis also use the clustering approach, but with different objectives. The clustering algorithm is applied to the pages returned by the search engine to group pages that have method calls in common in the same clusters, in order to obtain pages with solutions more focused on the user's query. Whereas in the above work, the clustering approach is applied to queries with the aim of grouping similar queries into clusters.

Zhao et al. (2017) address the problem of identifying irrelevant items from a small set of similar documents using Web search engine suggestions. They collected Web pages through Web search engines and inspected the page contents using topic models. Among each cluster of pages sharing the same topic the proposed technique discovers potential content organization in the current page cluster and identifies pages that are out of focus from that topic. The metrics of the approach mainly consist of search engine suggest frequency (search suggestions provided by search engines) and inter-document similarity measures. The intuition is that Web pages collected via the same search queries are more likely to share similar contents. The authors verify this intuition by implementing a subtopic based document selection framework and making quantitative evaluation against human made labeled data sets. The results show that suggest frequency analysis along with inter-document similarity measure is effective at filtering off-topic documents in small data sets with satisfactory performance. In this thesis, we use a clustering algorithm to group pages that have method calls in common, in order to obtain relevant pages with content focused on the user's query. In the work cited above, the authors use topic models to identify irrelevant pages based on the textual content of the pages that does not take into account the source code of the solutions present on the pages.

Agichtein, Brill e Dumais (2006) show that incorporating user behavior data can significantly improve ordering of top results in real web search setting. They examine alternatives for incorporating feedback into the ranking process and explore the contributions of user feedback compared to other common web search features. The authors report results of a large scale evaluation over 3,000 queries and 12 million user interactions with a popular web search engine. They show that improving implicit feedback can increase other features, the accuracy of a competitive web search ranking algorithms by as much as 31% relative to the original performance. The approach proposed in this thesis also aims to improve the ranking quality of pages returned by search engines, but for content related to software development.

The identification of method calls in the source code of web pages returned by search engines is challenging because in the source code of the page, they are mixed with method calls that are part of the construction of the web page. For example, method calls within the *script* and *style* tags present on the web page are not part of the solution sought by the developer, as they are related to the construction of the web page itself. Stylos e Myers (2006) obtain the method calls present in the source code of the web pages by means of a list of method calls extracted from the official documentation of the JAVA language, with the limitation of collecting method calls only from this programming language. The approach proposed in this thesis effectively recognizes method calls of different computer programming technologies, through regular expressions, obtaining these method calls automatically.

Stylos e Myers (2006) and Diamantopoulos, Karagiannopoulos e Symeonidis (2018) propose approaches for obtaining pages and method calls, but those works are limited only to the JAVA programming language. The approach that we are proposing can obtain pages with method calls from several different computer programming technologies.

Mandelin et al. (2005) created a tool called Prospector, that given a class or an object previously known by the programmer, the tool helps to search for code examples. A disadvantage of that approach is the requirement of prior knowledge of the class or the object, greatly limiting its use by programmers who do not know the underlying programming technology. Our approach does not have this limitation, that is, the developer does not need to have prior knowledge of the method calls to use the proposed approach.

Chatterjee, Juvekar e Sen (2009) have proposed a new code search technique called Sniff, which retains the flexibility to perform a search in English, while obtaining small pieces of relevant code needed to perform the desired task. In Sniff, a programmer defines a query expressing the programming task in English and the tool returns a small set of relevant code snippets. Regarding the above work of Mandelin et al. (2005), Sniff has the advantage of eliminating the need for prior knowledge about the API to be reused.

Zheng, Zhang e Lyu (2011) mine the results of search engines to collect relevant information from a given API from an old library and then recommend candidates for the API from the new library that appear frequently in web search results returned by search engine. Our work also uses the results returned by search engines, but our goal is to improve the quality of ranking pages through filters and clustering algorithms, while the work cited above aims to recommend new libraries.

Chapter **6**

Conclusion

In this thesis, we have shown that the Google search engine may return a non-negligible number of pages with low focus on the solution sought by the developer. The results also show that there are several pages returned with no code samples. Pages with these characteristics are not interesting for developers looking for a solution for their problem in hand, motivating our proposed approach to filter only relevant solutions for developers.

In order to reinforce the importance of the proposed filters, we carried out a study with human subjects. The results showed that ranking quality influences the performance of developers when performing programming tasks. The findings of this study are summarized below.

The time spent solving programming tasks using a Lower Quality Ranking is generally greater than the time spent on a Higher Quality Ranking. In tasks related to Higher Quality Ranking, participants do not use irrelevant pages, because relevant pages are at the top of the ranking and the participants already find the solution for the task on the first pages. Participants visited more pages related to Lower Quality Ranking than those related to Higher Quality Ranking, because relevant pages are spread out in Lower Quality Ranking, so participants must visit more pages to find them. Our results showed that two tasks related to Lower Quality Ranking were not completely resolved. In those tasks, the participants visited pages in the lowest positions of the ranking. This is due to irrelevant pages in the top positions of the ranking hinder the participants to find the solution. One of the participants used the methods in the list of methods that we include in the description of the ranked pages to solve two programming tasks. All participants skipped the irrelevant pages except one participant, possibly be due to the list of frequent methods in the page descriptions in the rank. The time spent analyzing the Lower Quality Ranking pages is longer than the time spent analyzing the Higher Quality Ranking pages, because adequate solutions are spread out in the ranking, so participants need to browse the ranking to access theses solutions. For all pairs of analyzed tasks, the time spent on useless pages was higher for the Lower Quality Ranking, because when developers encounter a bad code on an useless page, they spend a lot of time trying to reuse the bad code. We observed that developers try to fix the bad code by searching for content on other pages, spending even more time. In cases where the developer cannot fix the bad code, all the effort is wasted, so the developer moves on to another page.

The above results motivated the proposal of an approach to filter relevant pages in the ranked list returned by search engines. Our study to evaluate the effectiveness of the proposal compared Google as baseline and three different approaches. The results showed that the outlier page removal filter used by the proposed approaches helps to remove irrelevant pages for software developers. For all page numbers given as input, the filter has F-Measure higher than Google Search Engines, except for page number equals to 5, where the median of the Google Search Engines is slightly above the median of the filter. Regarding the approach that uses the clustering algorithm having as input the total occurrence of method calls on the pages returned by the Google search engine, has very large variability in results, for most page numbers given as input to the approach. In addition, this approach removes many relevant pages for software developers. Therefore, we do not recommend it for removing irrelevant pages. The approach that applies a clustering algorithm with unique occurrences of method calls as attribute value obtained higher precision and F-Measure compared to baseline and the other evaluated approaches. In addition to improving the top-5 ranking quality, according to the results of the MRR, MAP and NDCG metrics. These results demonstrated the effectiveness of creating filters to improve the results returned by search engines.

As future work, the achieved approach can be used to create documentation focused on the programming task queried by a developer on a search engine. That documentation could contain containing descriptive explanations, code examples and frequently asked questions about method calls found in the solutions of the pages would be returned on-the-fly. Other future work would be empirically evaluating the actual benefits of filtering web search results during a development activity with a larger and representative sample in order to provide a more general and robust result.

The thesis statement is shown below:

Thesis Statement

Poor quality ranking returned by search engines may hinder developers' performance when solving programming tasks. Filters capable of eliminating irrelevant pages for software developers, present in the results returned by search engines could be developed and proved to be effective.

6.1 Bibliographic Production

The following article related to this thesis was submitted, and it is currently under review:

1. Mining Relevant Solutions for Programming Tasks from Search Engine Results. It was submitted to the journal IET Software. This article comprises the Chapter 4 of this thesis, where we carried out a study of the pages returned by Google and found that it returns many pages that are irrelevant to developers. So we propose an approach to mining relevant pages from search engine results for programming tasks by identifying and filtering out irrelevant pages from the ranked list.

The following articles related to the general theme of the thesis were already published during this research period:

1. Bootstrapping cookbooks for APIs from crowd knowledge on Stack Overflow. This paper was published in the journal Information and Software Technology, and it carried out with the collaboration of the research laboratory group. This article addresses the problem of the poor quality documentation for APIs by providing an alternative artifact to document them based on the crowd knowledge available on Stack Overflow, called crowd cookbook. A cookbook is a recipe-oriented book, and we refer to our cookbook as crowd cookbook since it contains content generated by a crowd (SOUZA et al., 2019).

2. Improving the Classification of Q&A Content for Android Fragmentation using Named Entity Recognition. It was accepted at the EPIA conference (Conference on Artificial Intelligence) and published as a book chapter by Springer. This article addresses the problem of low performance of classifiers when target classes have similar content. The objective of this study was to develop a Named Entity Recognizer (NER) model to recognize entities related to technical elements, and to improve textual classifiers for Android fragmentation posts from Stack Overflow using the obtained NER Model (ROCHA; MAIA, 2019).

Bibliography

AGICHTEIN, E.; BRILL, E.; DUMAIS, S. Improving web search ranking by incorporating user behavior information. In: **Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval**. New York, NY, USA: Association for Computing Machinery, 2006. (SIGIR '06), p. 19–26. ISBN 1595933697. Disponível em: <https://doi.org/10.1145/1148170.1148177>.

AGRAHRI, A. K.; MANICKAM, D. A. T.; RIEDL, J. Can people collaborate to improve the relevance of search results? In: **Proceedings of the 2008 ACM Conference on Recommender Systems**. New York, NY, USA: Association for Computing Machinery, 2008. (RecSys '08), p. 283–286. ISBN 9781605580937. Disponível em: <https://doi.org/10.1145/1454008.1454052>.

AMIN, G. R.; EMROUZNEJAD, A. Optimizing search engines results using linear programming. **Expert Syst. Appl.**, Pergamon Press, Inc., USA, v. 38, n. 9, p. 11534–11537, set. 2011. ISSN 0957-4174. Disponível em: <https://doi.org/10.1016/j.eswa.2011.03.030>.

BACKMAN, K.; KYNGAS, H. A. Challenges of the grounded theory approach to a novice researcher. **Nursing & Health Sciences**, v. 1, n. 3, p. 147–153, 1999. Disponível em: <https://doi.org/10.1046/j.1442-2018.1999.00019.x>.

BAJRACHARYA, S.; OSSHER, J.; LOPES, C. Sourcerer: An infrastructure for large-scale collection and analysis of open-source code. **Sci. Comput. Program.**, Elsevier North-Holland, Inc., USA, v. 79, p. 241–259, jan. 2014. ISSN 0167-6423. Disponível em: <https://doi.org/10.1016/j.scico.2012.04.008>.

BUSE, R. P. L.; WEIMER, W. Synthesizing API usage examples. In: **Proceedings of the 34th International Conference on Software Engineering**. IEEE Press, 2012. (ICSE '12), p. 782–792. ISBN 9781467310673. Disponível em: <https://doi.org/10.1109/ICSE.2012.6227140>.

CARAMIA, M.; FELICI, G.; PEZZOLI, A. Improving search results with data mining in a thematic search engine. **Computers & Operations Research**, v. 31, n. 14, p. 2387–2404, 2004. ISSN 0305-0548. Disponível em: <https://doi.org/10.1016/S0305-0548(03)00194-1>.

CHATTERJEE, S.; JUVEKAR, S.; SEN, K. Sniff: A search engine for java using free-form queries. In: CHECHIK, M.; WIRSING, M. (Ed.). **Fundamental Approaches to Software Engineering**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 385–400. ISBN 978-3-642-00593-0. Disponível em: <https://doi.org/10.1007/978-3-642-00593-0_26>.

CHO, J.; ROY, S. Impact of search engines on page popularity. In: **Proceedings of the 13th International Conference on World Wide Web**. New York, NY, USA: Association for Computing Machinery, 2004. (WWW '04), p. 20–29. ISBN 158113844X. Disponível em: <https://doi.org/10.1145/988672.988676>.

DELFIM, F. M. et al. Redocumenting APIs with crowd knowledge: a coverage analysis based on question types. **Journal of the Brazilian Computer Society**, v. 22, n. 1, p. 9, dez. 2016. Disponível em: <https://doi.org/10.1186/s13173-016-0049-0>.

Diamantopoulos, T.; Karagiannopoulos, G.; Symeonidis, A. Codecatch: Extracting source code snippets from online sources. In: **2018 IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)**. [S.l.: s.n.], 2018. p. 21–27.

FISCHER, F.; STACHELSCHEID, Y.; GROSSKLAGS, J. The effect of Google search on software security: Unobtrusive security interventions via content re-ranking. In: **Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security**. New York, NY, USA: Association for Computing Machinery, 2021. (CCS '21), p. 3070–3084. ISBN 9781450384544. Disponível em: <https://doi.org/10.1145/3460120.3484763>.

GALLARDO-VALENCIA, R. E.; SIM, S. E. Internet-scale code search. In: **Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation**. USA: IEEE Computer Society, 2009. (SUITE '09), p. 49–52. ISBN 9781424437405. Disponível em: <https://doi.org/10.1109/SUITE.2009.5070022>.

HONG, Y.; VAIDYA, J.; LU, H. Search engine query clustering using top-k search results. In: **Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Volume 01**. USA: IEEE Computer Society, 2011. (WI-IAT '11), p. 112–119. ISBN 9780769545134. Disponível em: <https://doi.org/10.1109/WI-IAT.2011.224>.

HORA, A. Characterizing top ranked code examples in Google). **Journal of Systems and Software**, 2021. Disponível em: <https://doi.org/10.1016/j.jss.2021.110971>.

_____. Googling for software development: What developers search for and what they find. In: **International Conference on Mining Software Repositories**. [s.n.], 2021. p. 1–12. Disponível em: <https://doi.org/10.1109/MSR52588.2021.00044>.

JOHNSON, R. E. Documenting frameworks using patterns. In: **Conference Proceedings on Object-Oriented Programming Systems, Languages, and Applications**. New York, NY, USA: Association for Computing Machinery, 1992. (OOPSLA '92), p. 63–76. ISBN 0201533723. Disponível em: <https://doi.org/10.1145/141936.141943>.

Kataria, S.; Sapra, P. A novel approach for rank optimization using search engine transaction logs. In: **2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)**. [S.l.: s.n.], 2016. p. 3387–3393.

KEANE, M. T.; O'BRIEN, M.; SMYTH, B. Are people biased in their use of search engines? **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 51, n. 2, p. 49–52, feb 2008. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/1314215.1314224>.

KIM, J. et al. Towards an intelligent code search engine. In: **Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence**. AAAI Press, 2010. (AAAI'10), p. 1358–1363. Disponível em: <https://doi.org/10.1609/aaai.v24i1.7503>.

KIM, Y.; COLLINS-THOMPSON, K.; TEEVAN, J. Using the crowd to improve search result ranking and the search experience. **ACM Trans. Intell. Syst. Technol.**, Association for Computing Machinery, New York, NY, USA, v. 7, n. 4, jul. 2016. ISSN 2157-6904. Disponível em: <https://doi.org/10.1145/2897368>.

LAU, T.; HORVITZ, E. Patterns of search: Analyzing and modeling web query refinement. In: **Proceedings of the Seventh International Conference on User Modeling**. Berlin, Heidelberg: Springer-Verlag, 1999. (UM '99), p. 119–128. ISBN 3211831517. Disponível em: <https://doi.org/10.1007/978-3-7091-2490-1_12>.

MANDELIN, D. et al. Jungloid mining: Helping to navigate the API jungle. In: **Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation**. New York, NY, USA: Association for Computing Machinery, 2005. (PLDI '05), p. 48–61. ISBN 1595930566. Disponível em: <https://doi.org/10.1145/1065010.1065018>.

MURPHY, G. et al. Enabling Productive Software Development by Improving Information Flow. In: _____. **Rethinking Productivity in Software Engineering**. Berkeley, CA: Apress, 2019. Disponível em: <https://doi.org/10.1007/978-1-4842-4221-6_24>.

NIU, H.; KEIVANLOO, I.; ZOU, Y. Learning to rank code examples for code search engines. **Empirical Softw. Engg.**, Kluwer Academic Publishers, USA, v. 22, n. 1, p. 259–291, fev. 2017. ISSN 1382-3256. Disponível em: <https://doi.org/10.1007/s10664-015-9421-5>.

NYKAZA, J. et al. What programmers really want: Results of a needs assessment for SDK documentation. In: **Proceedings of the 20th Annual International Conference on Computer Documentation**. New York, NY, USA: Association for Computing Machinery, 2002. (SIGDOC '02), p. 133–141. ISBN 1581135432. Disponível em: <https://doi.org/10.1145/584955.584976>.

RAHMAN, M. M. et al. Evaluating how developers use general-purpose web-search for code retrieval. In: **Proceedings of the 15th International Conference on Mining Software Repositories**. New York, NY, USA: Association for Computing Machinery, 2018. (MSR '18), p. 465–475. ISBN 9781450357166. Disponível em: <https://doi.org/10.1145/3196398.3196425>.

RAHMAN, M. M.; ROY, C. K.; LO, D. Automatic query reformulation for code search using crowdsourced knowledge. **Empir. Softw. Eng.**, v. 24, n. 4, p. 1869–1924, 2019. Disponível em: <https://doi.org/10.1007/s10664-018-9671-0>.

ROBILLARD, M. P. What makes APIs hard to learn? answers from developers. **IEEE Softw.**, IEEE Computer Society Press, Washington, DC, USA, v. 26, n. 6, p. 27–34, nov 2009. ISSN 0740-7459. Disponível em: <https://doi.org/10.1109/MS.2009.193>.

ROCHA, A. M.; MAIA, M. A. Improving the Classification of Q&A Content for Android Fragmentation Using Named Entity Recognition. In: ______. **Progress in Artificial Intelligence**. Switzerland: Springer, Cham, 2019. p. 731–743. Disponível em: <https://doi.org/10.1007/978-3-030-30244-3_60>.

SADOWSKI, C.; STOLEE, K. T.; ELBAUM, S. How developers search for code: A case study. In: **Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2015. (ESEC/FSE 2015), p. 191–201. ISBN 9781450336758. Disponível em: <https://doi.org/10.1145/2786805.2786855>.

Saraswathi, D.; Vijaya, A. A generic tool for link spam detection in search engine results using graph mining. In: **2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering**. [S.l.: s.n.], 2013. p. 282–286.

Sharma, A. K. et al. Web search result optimization by mining the search engine query logs. In: **2010 International Conference on Methods and Models in Computer Science (ICM2CS-2010)**. [s.n.], 2010. p. 39–45. Disponível em: <https://doi.org/10.1109/ICM2CS.2010.5706716>.

SILVA, R. F. G. et al. Recommending comprehensive solutions for programming tasks by mining crowd knowledge. In: **Proc. of the 27th Intl.Conf. on Program Comprehension (ICPC'2019)**. IEEE Press, 2019. p. 358–368. Disponível em: <https://doi.org/10.1109/ICPC.2019.00054>.

______. CROKAGE: Effective solution recommendations for programming tasks by leveraging crowd knowledge. **Empirical Software Engineering**, 2020. Disponível em: <https://doi.org/10.5753/cbsoft_estendido.2021.17295>.

SIM, S. E. et al. How well do search engines support code retrieval on the web? **ACM Trans. Softw. Eng. Methodol.**, Association for Computing Machinery, New York, NY, USA, v. 21, n. 1, dez. 2011. ISSN 1049-331X. Disponível em: <https://doi.org/10.1145/2063239.2063243>.

SIMMONS, O. E. Some professional and personal notes on research methods, systems theory, and grounded action. **World Futures**, Routledge, v. 62, n. 7, p. 481–490, 2006. Disponível em: <https://doi.org/10.1080/02604020600912772>.

SOUZA, L. B. et al. Bootstrapping cookbooks for APIs from crowd knowledge on stack overflow. **Inf. Softw. Technol.**, Butterworth-Heinemann, USA, v. 111, n. C, p. 37–49, jul 2019. ISSN 0950-5849. Disponível em: <https://doi.org/10.1016/j.infsof.2019.03.009>.

STOLEE, K. T.; ELBAUM, S.; DOBOS, D. Solving the search for source code. **ACM Trans. Softw. Eng. Methodol.**, Association for Computing Machinery, New York, NY, USA, v. 23, n. 3, jun 2014. ISSN 1049-331X. Disponível em: <https://doi.org/10.1145/2581377>.

Stylos, J.; Myers, B. A. Mica: A web-search tool for finding API components and examples. In: **Visual Languages and Human-Centric Computing (VL/HCC'06)**. [s.n.], 2006. p. 195–202. Disponível em: <https://doi.org/10.1109/VLHCC.2006.32>.

Suneetha, M.; Sameen Fatima, S.; Shaik Mohd. Zaheer Pervez. Clustering of web search results using suffix tree algorithm and avoidance of repetition of same images in search results using l-point comparison algorithm. In: **2011 International Conference on Emerging Trends in Electrical and Computer Technology**. [S.l.: s.n.], 2011. p. 1041–1046.

Suo, H.; Zhang, W.; Zhang, Y. Research on automatic summarization based on search engine result. In: **2009 International Conference on Web Information Systems and Mining**. [s.n.], 2009. p. 74–77. Disponível em: <https://doi.org/10.1109/WISM.2009.23>.

XIA, X. et al. What do developers search for on the web? **Empirical Softw. Engg.**, Kluwer Academic Publishers, USA, v. 22, n. 6, p. 3149–3185, dez. 2017. ISSN 1382-3256. Disponível em: <https://doi.org/10.1007/s10664-017-9514-4>.

ZAHERA, H. M.; EL-HADY, G. F.; EL-WAHED, W. F. A. Query recommendation for improving search engine results. **Int. J. Inf. Retr. Res.**, IGI Global, USA, v. 1, n. 1, p. 45–52, jan. 2011. ISSN 2155-6377. Disponível em: <https://doi.org/10.4018/ijirr.2011010104>.

ZHAO, C. et al. Identifying major contents among web pages with search engine suggests by modeling topics. In: **Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication**. New York, NY, USA: Association for Computing Machinery, 2017. (IMCOM '17). ISBN 9781450348881. Disponível em: <https://doi.org/10.1145/3022227.3022251>.

ZHENG, W.; ZHANG, Q.; LYU, M. Cross-library API recommendation using web search engines. In: **Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2011. (ESEC/FSE '11), p. 480–483. ISBN 9781450304436. Disponível em: <https://doi.org/10.1145/2025113.2025197>.

ZHUANG, Z.; CUCERZAN, S. Re-ranking search results using query logs. In: **Proceedings of the 15th ACM International Conference on Information and Knowledge Management**. New York, NY, USA: Association for Computing Machinery, 2006. (CIKM '06), p. 860–861. ISBN 1595934332. Disponível em: <https://doi.org/10.1145/1183614.1183767>.