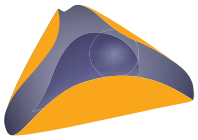


MASTER IN COMPUTER SCIENCE

Analysis of Developer Information Needs on Collaborative Platforms

Master Thesis

Mathias Birrer

from

Bern, Switzerland

Faculty of Science
at the University of Bern

June 2020

Prof. Dr. Oscar Nierstrasz

Pooja Rani

Software Composition Group
Institute for Computer Science

University of Bern, Switzerland

b
UNIVERSITÄT
BERN

UNIVERSITÉ DE
NEUCHÂTEL

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Acknowledgment

A very special gratitude goes to Pooja Rani who supported me throughout this thesis with great patience, precious advice and many interesting discussions! It was great to work with you and to benefit from your never-ending motivation and dedication to support the persons around you.

Many thanks also to Sebastiano Panichella for the valuable inputs, the proof-reading and the immense experience he shared with me. It was inspiring to work with such an accomplished researcher.

A big thank you goes also to Prof. Dr. Oscar Nierstrasz for the opportunity to complete this thesis at the Software Composition Group. It was great to experience the whole SCG team, get insights into many interesting projects and benefit from the shared knowledge.

Thanks for all the support and encouragement!

Abstract

Developer information needs are diverse, and so are the sources to satisfy those needs. Besides internal sources, developers frequently use external sources of information (*e.g.*, Q&A sites, mailing lists). Researchers use data from these sources to gain insights into developer information needs and build solutions to support the development process. However, along with the opportunities, the diversity of external sources poses challenges for research.

In this thesis, we identify and describe various external sources used by researchers to investigate developer information needs. We analyze the methodology of relevant literature to extract common practices, identify challenges, and assess data extraction and preprocessing reproducibility. To further increase knowledge about developer information needs and the process of analyzing these information needs, we conduct a case study about *Code Comment Convention* questions using data from multiple sources. The case study follows best practices identified in the relevant literature and is conducted with the help of a self-developed prototype tool. The prototype tool is designed to help researchers extract, manage, and preprocess a dataset from multiple sources in a documented and reproducible way.

Contents

1 Introduction **1**

1.1 Goal and Research Questions 4
1.2 Summary of Main Contributions 5
1.3 Outline 5

2 Related Work **6**

2.1 Surveys analyzing mining textual data 6
2.2 Diversity of sources 7
2.3 Existing tools and frameworks 8

3 Sources of Developer Information Needs **9**

3.1 Overview 9
3.2 Methodology 10
3.3 Results 10
3.3.1 Q&A Sites 11
3.3.2 Mailing lists 13
3.3.3 Bug Reports 14
3.3.4 Newsgroups 15
3.3.5 User Reviews 15
3.4 Discussion 16

4 Reproducibility of Studies of Developer Information Needs **17**

4.1 Overview 17
4.2 Methodology 18
4.3 Results 19
4.3.1 Extraction Methodology 19
4.3.2 Preprocessing Methodology 21
4.3.3 Data Management 22
4.3.4 Dataset Availability 23
4.3.5 Reproducibility 23

4.4 Discussion . . . 26

5 Case Study: Code Commenting Conventions 28
5.1 Overview . . . 28
5.2 Methodology . . . 30
5.2.1 Data Selection . . . 31
5.2.1.1 Stack Overflow . . . 31
5.2.1.2 Quora . . . 34
5.2.2 Data Preparation . . . 34
5.2.3 Analysis . . . 36
5.2.3.1 LDA . . . 36
5.2.3.2 Taxonomy Definition . . . 37
5.2.3.3 Mailing lists . . . 38
5.3 Results . . . 38
5.3.1 LDA . . . 38
5.3.2 Manual Analysis . . . 41
5.4 Discussion and Implications . . . 43

6 Makar: Data Management Tool 45
6.1 Requirements . . . 45
6.2 Features . . . 46
6.3 Architecture . . . 49
6.3.1 Overview . . . 49
6.3.2 Data model . . . 50
6.3.2.1 Data model for user-defined schema . . . 50
6.3.2.2 Data model for Makar functionality . . . 51
6.3.3 System design . . . 51
6.3.3.1 User-defined Schemas . . . 51
6.3.3.2 Data Transformation . . . 53
6.3.3.3 Import Adapters . . . 54
6.3.3.4 Export Adapters . . . 54
6.3.3.5 Searching and Filters . . . 55
6.3.3.6 Background Jobs . . . 55
6.3.4 Technology . . . 55
6.3.4.1 Core Application . . . 55
6.3.4.2 Containerization . . . 56
6.3.4.3 Technical Requirements . . . 56
6.4 Benefits and Future Improvements . . . 56
6.4.1 Benefits . . . 57
6.4.1.1 Diverse Use Cases . . . 57

6.4.1.2 Exploratory Studies 57
6.4.2 Future Improvements 58
6.4.2.1 Data Analysis and Visualizations 58
6.4.2.2 Performance 58
6.4.2.3 Data Volume 58

7 Threats to Validity **60**

8 Conclusion and Future Work **62**

A Appendices **64**
A.1 List of papers for RQ1 and RQ2 64
A.2 Tool comparison 67

1 Introduction

Software development is a complex process, which can be conceptualized as a software development life cycle, effectively partitioning a developer's work into multiple phases [75]. Agile practices [13, 28, 39] have been proposed and implemented a long time ago and have structured the work process for many developers. More recently, *DevOps* extended traditional agile practices with automated approaches to software testing and deployment[51]. Software development is a life cycle with multiple phases, all posing their unique challenges to developers. For simplicity, we distinguish the following high-level phases: **planning, implementation, testing, releasing**, and **maintenance**. In each of these phases, developers encounter complex scenarios, face various challenges, and need information to solve the task at hand. We can imagine a mobile app developer, who wants to implement a new app idea. This developer talks to *possible users* in the planning phase, then consults *mailing lists* and *Q&A sites* during implementation. In the testing phase, the developer uses a new testing framework; thus the *framework documentation* is an important information source. For the release, the developer asks a question on a *Q&A site* about the configuration of the continuous integration tool. Then, during maintenance, the developer reads *user reviews* to detect possible bugs and get feedback on his development.

Throughout the development phases, the developer information needs continuously change, and thus, a developer considers different sources of information depending on the phase. To effectively implement changes, a developer needs to consult a variety of information to understand the software's structure and its components [46]. Meyer *et al.* have shown that developers have highly fragmented work with

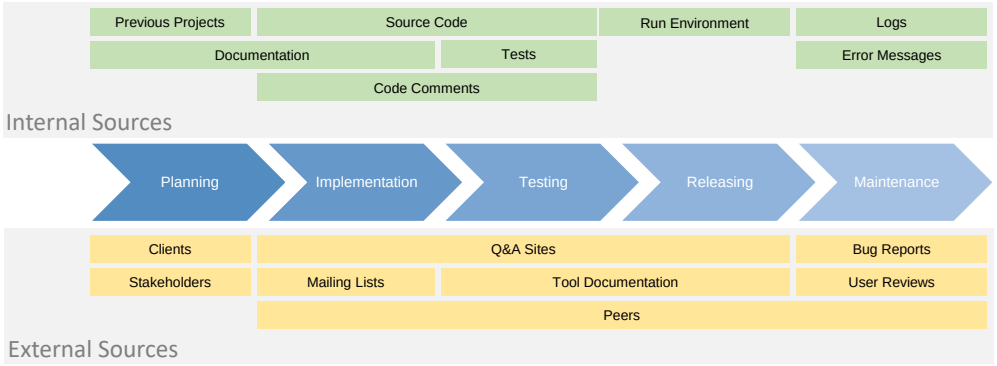


Figure 1.1: Software developers use different information sources in every development phase

diverse activities [58], which supports the assumption that developers regularly switch between different sources of information. The sources of information for developers can be divided into **internal sources** and **external sources**. Internal sources are artifacts of the software itself, like source code, code comments, existing documentation, or runtime logs. External sources are artifacts from other software components (*e.g.*, documentation), web resources, mailing lists, and other developers. Figure 1.1 shows possible internal and external sources in the software development process. The usage of internal sources is well researched [46, 59, 72], which led to various techniques to improve IDEs to increase developer productivity by including optimized code searches, debuggers, version control, and code analysis tools. However, with an increasing landscape of tools and web resources, supporting developers in specific phases of the development process, not all information needs can be satisfied in the IDE. As a result, developers often switch from the code editor to the web browser [10]. Developers spend up to 50% of their time on the web [10]. The reasons to switch from the code editor to the web browser are diverse: developers ask their peers questions, read documentation on project-specific platforms, or use various online sources to meet their information needs. These sources of information often differ vastly in terms of topics, level of technicality, community, rules of engagement, and discussion culture, and thus, are differently used by developers throughout the development phases. For example, community websites and mailing lists encourage asking general questions, whereas GitHub and JIRA, provide platforms to ask project-specific questions. The types of questions developers can ask vary within community websites depending on the website rules, technicality, and topics. For example, on Quora[1] , a rapidly-growing general-purpose Q&A platform, developers post questions about opinions on technologies, techniques, tools, and core concepts of programming. In contrast, Stack Overflow[2] is a more specialized platform where developers ask questions about specific programming errors, handling errors, and opinions on specialized concepts. The research about external sources is not as evolved as about internal sources. Nevertheless, various studies have been conducted [63, 65, 89] to bring information from external sources to the developer. The amount of data available to software developers and for researchers has grown exponentially in the past

[1] www.quora.com
[2] www.stackoverflow.com

years [12]. However, current tools supporting the developer often cannot analyze and generate information from multiple types of data and sources.

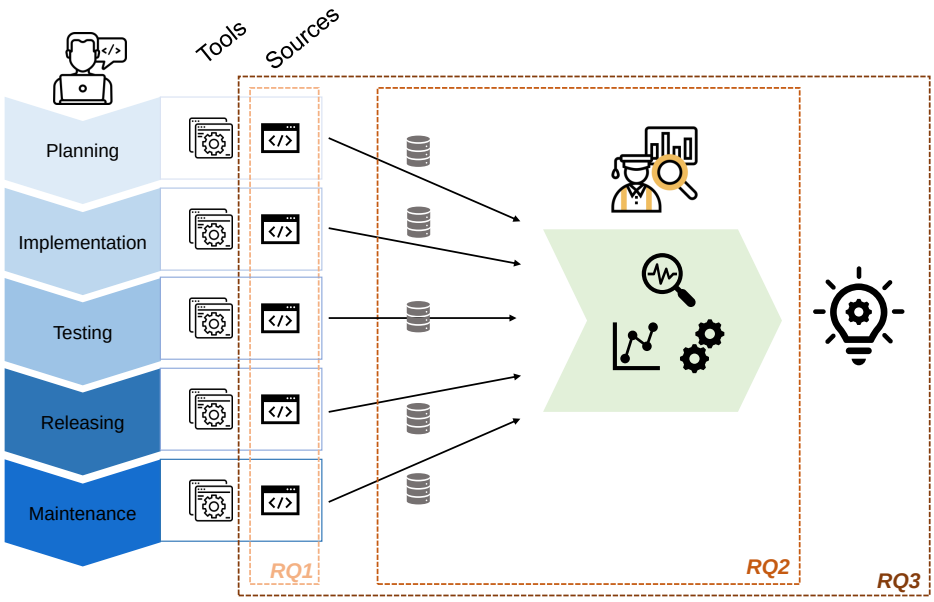


Figure 1.2: Software development pipeline with various information sources producing data for researchers

Studying developer activities and engagement related to various development tasks from various sources and understanding them in context can guide the development of tools and practices to help developers find the desired information more easily. Therefore, researchers have recently focused on leveraging the useful content of these sources, *i.e.*, Git, CVS [24], archived communications, *e.g.*, mailing lists [77, 79], execution logs [33], newsgroups [41], online forums and Q&A sites [6, 9, 11, 85, 96, 98] to comprehend the developer information needs and find the technology trends. Figure 1.2 illustrates how researchers analyze data from developer information sources. Researchers extract data from external sources of developers' information, preprocess them, and analyze the data to find new insights and solutions to improve software development. The extraction, preprocessing, and analysis of unstructured data for Software Engineering tasks has many challenges [12, 24], such as mapping data from one source to another and managing the gathered data. Automating the analysis process to better document the study is another challenge faced by research studies, resulting in unclear steps for preprocessing data. Prior studies have proved the necessity of data preprocessing [24, 64] and pointed out that 50% of articles did not report whether word stemming is used or not, which is a common preprocessing step in the analysis of unstructured data. Imprecise reporting of data extraction and preprocessing steps are a threat to reproducibility. However, research has not addressed to what extent reproducibility is achieved in such studies. Computational reproducibility in the scientific software context is currently a much discussed topic. Multiple approaches to increase reproducibility are proposed [7, 48, 73].

This thesis works towards a standardized approach to preprocess and analyze data from developers' external information sources. First, we investigated which external sources of information are used by researchers to conduct studies about software development (see *RQ1* in figure 1.2). For this, we analyzed the different types of data sources and their use in research to gain insight into development activities. Next, we investigated multiple aspects of data extraction, preprocessing, and tool usage reported in studies found in *RQ1*, with a particular focus on reproducibility. With insights from *RQ1* and *RQ2*, we conducted a case study about code commenting conventions (*RQ3*). In the case study, we applied good practices previously identified, addressed challenges found in other studies, and combined data from multiple sources to further extend knowledge about developer information needs. For this case study, we developed a prototype tool that offers a standardized approach for collection, management, and preprocessing of data from different sources and addresses reproducibility issues found in the analyzed studies. Using the proposed prototype in the case study, we achieve efficient data extraction, data management, reproducible preprocessing steps and dataset generation. We believe that this could help developers as well as researchers to efficiently work with different types of data and provide support for various development activities.

1.1 Goal and Research Questions

We address the following three research questions:

- **RQ1**: *Which data sources are typically analyzed by researchers to understand developer information needs?*
  We conducted a literature survey to gather popular sources of information for developers that are used by researchers to analyze developer information needs. We identify the challenges for researchers and the characteristics of each source.

- **RQ2**: *How do researchers analyze developer information needs on collaborative platforms?*
  We identify common practices and tools to conduct studies in the field of analyzing developer information needs empirically. We perform an empirical study on a corpus of relevant papers concerning specific aspects of their methodology. We analyze the aspects of data retrieval, data preprocessing, data management, dataset availability, and reproducibility for selected studies. Our results demonstrate that these previous works lack reproducibility practices. We identify that many of the analyzed papers lack complete documentation of their methodology, and reproducibility is impossible or hard to achieve.

- **RQ3**: *What are developers' questions about code commenting?*
  Code comments can be an essential part of the source code for developers to understand existing code artifacts or automatically generate documentation. Despite that, there is little previous work gaining insights on code comments from external sources. We conduct a case study, investigating the conventions and questions of developers about code comments. For the case study, we collect data from multiple sources and use a prototype tool to collect, manage, preprocess the data, and provide a reproducible dataset. The prototype tool called *Makar* is based on modern technology and

easy to set up and use. It is designed to collect and manage data from external sources as well as to overcome issues identified in *RQ2*.

1.2 Summary of Main Contributions

The main contributions of this thesis are the following:

Literature survey of data sources. We investigate which web resources are used by researchers to collect data for answering questions about developer information needs. We highlight multiple external sources, characteristics, and differences.

An empirical investigation of research methodology. We analyze research methodology of literature concerning developer information needs using data from collaborative platforms. We provide insights into different approaches and possibilities to gather and preprocess data. We show common practices in the research methodology and highlight shortcomings.

Multi-source case study. We conduct a multi-source study investigating developers' questions about *Code Commenting Conventions*. The study includes data from multiple web resources to build a taxonomy of developers' code comment questions. We use a prototype tool to gather, preprocess, and manage data, following the common practices identified in the empirical investigation.

Development of prototype tool We propose a prototype tool called Makar, which mitigates specific difficulties researchers face when conducting studies about developer information needs using data from collaborative platforms. Makar focuses on providing a convenient method to gather and preprocess data from multiple sources and provide a reproducible dataset.

1.3 Outline

This thesis is structured as follows. In chapter 2, the related work is presented, and a short overview of the state-of-the-art is given. In chapter 3, insights and results to *RQ1* are discussed, followed by the analysis of different methodologies and practices of relevant studies to answer *RQ2* in chapter 4. Chapter 5 presents the case study and its results about developers' questions on *code commenting conventions*. The next chapter introduces the prototype tool *Makar* and provides some insight into its design, functionality, and intended use. Followed by chapter 8, which concludes this thesis and addresses future work.

2

Related Work

This chapter provides background information and related work on the analysis of specific sources of information. The focus is not on mining studies but on various sources of information and how they are used in research. We present an overview of studies mining textual data, studies using multiple sources of data, and related work in tools that facilitate studies using various data sources.

2.1 Surveys analyzing mining textual data

Bavota gave an overview of current applications and future trends of mining unstructured data in the SE field[12] and hinted the opportunities in utilizing various software repositories in SE field. Chen *et al.* surveyed 167 research articles on the use of topic models applied to software engineering repositories mining [24]. They found that a significant number of research studies focused on a few software engineering tasks and used elementary topic modeling techniques to grasp the overview of the topics. They also pointed out the need to have consistent and better documentation of data preprocessing. Similarly, Farias *et al.* analyzed 107 research articles published in MSR conferences [25]. They evaluated the studies based on goals of software analysis performed, data sources used, and assessment performed. Their main finding is that structured repositories are mined more compared to unstructured repositories. However, there is a rise in utilizing unstructured data in the past three years. In contrast to others, Ahmad *et al.* covered a single source, Stack Overflow, and focused specifically on the software development aspect [3]. Nazar *et*

al. conducted a literature review on *Summarizing Software Artifacts* focusing on bug reports, source code, mailing lists and developer discussion artifacts [61]. They discuss different machine learning and data mining techniques that are used in analyzing structured and unstructured data.

All these survey papers have mainly focused on analyzing the quality of questions posted on the websites, using diverse topic models in mining software repositories, applications, and the purpose of mining unstructured data. However, our focus is specific to empirical studies conducted to analyze unstructured, external sources and explore their data extraction, management and preprocessing workflow.

2.2 Diversity of sources

Researchers have expanded from using more traditional sources, like source code artifacts or source code versioning, to also consider modern web resources, like user reviews or Q&A sites. App store data or user reviews are an important source of information for app developers. Harman *et al.* propose to use similar approaches as in traditional MSR studies, to analyze app store data [38]. Nagappan *et al.* [60] give an outlook of *Future Trends in Software Engineering Research for Mobile Apps*.

In recent times, studies have started focusing on aspects of multi-source analysis. Ting *et al.* [84] proposed a methodology to collect and analyze multi-source data for extracting and mining social network data. Similarly, Panichella *et al.* combined data from versioning systems, issue trackers, mailing lists and IRC chats to investigate communication links across multiple communication channels [66]. Hu *et al.* [42] focused on unifying data for the domain of travel route planning. Canfora *et al.* investigated cross-system bug fixing by combining data from two source code repositories (FreeBSD and OpenBSD), as well as mailing lists [22]. They combined information about bug fixes with social interactions from mailing lists. Pokharel *et al.* [69] proposed a framework to analyze trends and topics from multiple social media platforms.

One of the ways researchers perform such analysis is to download the required data from different sources either by using a service provided by the source or crawling the website. For example, in the case of Stack Overflow, researchers can interact with the public API provided, while for Quora, they have to write specific crawlers. Once the dataset is downloaded, they clean the dataset and store it in a database and then analyze it. The tools, services, and processes used for performing these activities vary across teams and it can be hard to describe in adequate detail the replication process for another researcher. In addition to the lack of a clear and detailed process, the quality of results varies depending on the choice of technology and approaches [16]. Amann *et al.* [8] pointed out the reproducibility issues in software mining studies. González characterized the reproducibility in the area of empirical software engineering studies and proposed various elements to characterize them, such as data source and retrieval methodology [34]. Based on the reproducibility elements proposed by González, we assessed the reproducibility of selected literature and reported their shortcomings. Given the benefits of analyzing data from different sources, and challenges in conducting such studies, we conducted a case study analyzing code commenting convention questions from Stack Overflow, Quora and mailing lists.

Bettenburg *et al.* suggested to the research community to develop standardized processes. In order to standardize the process for our case study, we developed a tool, *Makar* and used to handle data throughout the study.

2.3 Existing tools and frameworks

Given the increasing trend of analyzing data from unstructured sources, researchers perform common steps and adopt similar methodologies. Automating and standardizing the workflow of such studies can reduce the repetitive efforts, and improve the ease of reproducibility. Shang *et al.* acknowledge the importance of using automated software to perform MSR studies with large data sets [76]. However, they critically see the use of specially developed one-off solutions and instead propose using off-the-shelf large-scale data processing platforms like *MapReduce*. Their work allows us to efficiently answer concrete answers concerning a software project, based on the evolution of the source code. Voinea and Telea published a framework for visual data mining and analysis of software repositories [90]. Their framework consists of scriptable data connectors for multiple source code versioning systems, a set of data filters, components to compute various metrics, and multiple visualization possibilities.

Ghezzi *et al.* developed SOFAS, which is a platform to enable systematic and repeatable analysis for MSR studies [32]. To prove applicability of their platform, they try to replicate 88 previous MSR studies, achieving a 30% replicability and 32% partial replicability. More systematic and standardized replicability in MSR studies is their main motivation for the presented platform.

Dyer *et al.* presented *Boa*, a domain-specific programming language, and infrastructure to support MSR studies [29]. They motivate their work with the very high costs of building an infrastructure for downloading, storing, and analyzing data for MSR related tasks. Therefore, reproducibility is often hard to achieve. In contrast to our tool, the focus of *Boa* is solely on data extracted from software repositories.

Pokharel *et al.* propose a framework to automatically extract emerging topics from multiple social media sites [69]. Their framework covers data preprocessing, automated analysis, and visualization, but only for the limited use case of finding emerging topics. Besides scientific projects and ongoing research, there are customer-facing solutions that allow preprocessing and analysis of diverse data.

Dwivedi *et al.* have compared four suitable data analysis tools and provided recommendations based on the technical skill-level of the users [27].

As stated, previous work has proposed multiple solutions to automate MSR studies, improve reproducibility and support researchers. However, the usage of those tools in research remains almost non-existent. The goal of the proposed tool in this thesis is to offer a standardized platform for collecting, preprocessing and providing a reproducible dataset from multiple sources.

3

Sources of Developer Information Needs

3.1 Overview

Developers have traditionally relied on mailing lists, source code versioning and issue tracking as primary tools for collaboration. The growing demand for software, increasing complexity, and the trend towards global software development has introduced new collaboration and knowledge sharing tools [14]. Developers need to communicate over various channels as well as store and retrieve information from multiple sources. *Mailing lists* for technical discussions in an open-source project, a *team chat* for collaboration with co-workers, the company *wiki* for cross-project development resources and standards, *documentation* for the testing library, a *Q&A site* to solve problems with the CI tool, and *bug reports* from customers: these are some examples of commonly used sources. Those sources are diverse from each other and are used at different development phases. Many of the sources of information are located on the web. Thus it is not a surprise that developers regularly conduct web searches to find answers to their open questions [97]. Developers choose different sources of information depending on the phase of development, technicality of the question, or chosen technology. Many of the web resources that offer information on software components and answers to developer questions are collaborative platforms or social media sites. Storey et al. characterize social media sites with "*an underlying architecture of participation that supports crowdsourcing as well as many-to-many broadcast mechanism*" [82] and state that developers frequently visit such sites and resources. The sources significantly differ on the level of technicality, community

characteristics, and rules of engagement. Due to such differences, researchers analyze the content of various collaborative platforms and investigate these differences in detail.

Previous work has analyzed mailing lists to predict software defects [100], Q&A sites to facilitate program repair [53], or automated comment generation [30] and combined bug reports and mailing lists to generate source code description [65]. In this chapter, we want to focus on a few external sources of developer information needs. In particular, we investigate which sources are frequently used by researchers and present an overview of the characteristics, challenges using these sources, and what insights can be drawn from its data. With the insights from this analysis, we answer *RQ1* and provide background information for *RQ2*. Therefore, we formulate our first research question: *Which data sources are typically analyzed by researchers to understand developer information needs?*

3.2 Methodology

To address *RQ1*, we analyzed a corpus of relevant literature that focuses on the data extracted from collaborative platforms about developer information needs. We extracted, categorized and investigated the information sources found in the body of literature. An overview of the methodology for *RQ1* and *RQ2* is shown in Figure 3.1.

The corpus of relevant literature is based on previous work [71], which presents a systematic literature review on software developer information needs. From this corpus of literature, we selected the technology-centric sub-group of papers for our study. This yielded 29 papers for our corpus of literature. Additionally, we increase the corpus of relevant literature by a targeted search on *Google Scholar* for papers using data from related sources as already present in the initial set of literature.

We selected only papers that study developer information needs and use data from a collaborative platform. This yielded an additional 13 papers, resulting in a total of 42 papers. The full list of papers is in Table A.1.

We carefully analyzed all papers and extracted all sources of data used in the selected literature. Some papers reported multiple sources and some sources were reported by multiple papers. Overall we extracted a list of 18 different sources. In the next step, we grouped the sources into five categories according to the type of collaborative platform. For each of the categories, we gathered background information from the corresponding papers about the type of source, its characteristics, challenges and problems that are addressed.

3.3 Results

Based on the corpus of relevant literature, we found the sources of developer information needs listed in table 3.1. We categorized each source as *Q&A Site*, *Mailing list*, *Newsgroup*, *Bug Report* or *User Review*. Please note that one paper can have multiple social media sites as sources, for example, Sharif and Buckley report on six different mailing lists (Apache BSF, Eclipse JDT, Element Construction Set, Eboard,

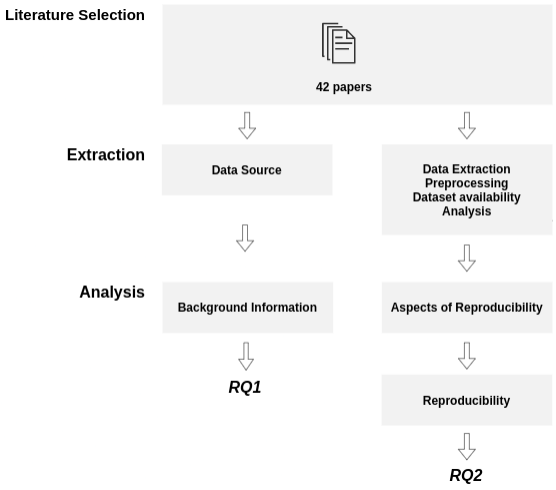


Figure 3.1: Methodology for *RQ1* and *RQ2*

SwingWT, Reciprocate)[77].

Table 3.2 aggregates the extracted sources per category. Some papers use sources from more than one category, *e.g.*, Zagalsky *et al.* used developer information needs data from Mailing lists and Stack Overflow and thus mentioned in categories *Q&A* and *Mailing lists*[99].

The collected social media platforms may not be a complete list but the assigned categories represent a diverse choice for developers to satisfy their needs for information and for researchers to find data to investigate those needs. In the following sections, we present the aggregated insights from the literature, grouped by category. The focus of the analysis was to uncover the characteristics, challenges and problems addressed in each category of sources in order to shed some light on the research of technology-centric sources.

3.3.1 Q&A Sites

Q&A sites are a valuable resource for developers. Even when documentation on a project or software component is available, it is hard to apply the found information to the actual software development tasks [50]. Therefore, developers often need to ask peers [46] and spend a significant amount of time finding answers [20]. Q&A sites support such a scenario and provide infrastructure to pose questions, answer questions or find answers to existing questions. Existing Q&A sites evolved to "*archives with millions of entries that contribute to the body of knowledge in software development*" and often replace official documentation [85].

It is evident that in the analyzed corpus of literature, the majority uses Q&A sites, especially Stack Overflow, as a source. Stack Overflow belongs to the most popular Q&A sites for developers [93] with millions of questions, answers, and comments on a wide range of topics. The data from Stack Overflow is made officially accessible through various channels (Data Dump, Data Explorer, API), which eases the process

category	platform / project	number of papers
Q&A Site	stackoverflow.com	25
Q&A Site	quora.com	2
Mailing list	Apache BSF Site	3
Mailing list	Eclipse JDT	3
Mailing list	Element Construction Set	1
Mailing list	Eboard	1
Mailing list	SwingWT	1
Mailing list	Reciprocate	1
Mailing list	R	1
Mailing list	Qt	1
Mailing list	Ubuntu	1
Bug Report	Eclipse	1
Bug Report	Mozilla	1
Newsgroup	Java Swing Framework	2
User Review	Google Play Store	6
User Review	Apple App Store	1
User Review	Windows Phone Store	1
User Review	Blackberry App Store	1

Table 3.1: Collaborative platforms reported as sources for data about developer information needs

of gathering massive amounts of data from Stack Overflow. Due to the ease of use of Stack Overflow combined with its popularity among developers, it is a target to conduct a diverse set of studies concerning software engineering and textual data mining [4, 11, 35, 74, 98, 99]. Despite the ease of extraction and the vast amount of data available on Stack Overflow, researchers face challenges in conducting their studies, namely data selection and preprocessing, to avoid deterioration of the results by noise. To address this challenge, multiple approaches of tag selection [4, 35, 74], keyword selection [67] or using the whole dataset [11] are proposed by various studies. The data from Stack Overflow contains HTML tags, source code and other metadata that need to be cleaned prior to any automated analysis. Among the studies using Stack Overflow data, a common practice of preprocessing [47, 52, 87, 88, 98] has emerged, which makes the extracted data suitable for analysis.

Quora is another Q&A platform used by developers, but not many research studies concerning software engineering have investigated this source. This is mainly since extracting data from Quora is somewhat tricky. The site does not provide any facility of a public API, an interface or Data Dump to get the data. Additionally, the web site itself is difficult to scrape because of limitations for not logged-in users and extensive use of Javascript in the frontend. Unfortunately, the two relevant studies [43, 96], which used data from Quora, did not explicitly report how the data was obtained, making it difficult for other researchers to apply a similar methodology and replicate the study.

category	number of papers
Q&A Site	27
Mailing list	5
Bug Report	1
Newsgroup	2
User Review	8

Table 3.2: The number of papers that per source platform category

In conclusion, research has repeatedly shown that Q&A sites, led by Stack Overflow, are an often used and valuable online resource in software development. Also the convenient access to data from Stack Overflow has made possible many studies and insight into developer information needs.

Observation 3.1 *Stack Overflow* is very popular data source to investigate developer information needs. It has enabled many insights into a diverse set of development-related topics.

3.3.2 Mailing lists

Mailing lists are a kind of public forums. Everyone can post a message to the list, which can be seen by all subscribers. Additionally, the messages are archived and serve as a log of communication between developers.

Mailing lists are often used in large, long-lived open source software projects [37]. For example, archetypal and well-known mailing lists are those of the Linux[1] and Apache[2] projects. With the rise of social Q&A sites, user support activities tend to shift away from mailing lists [86]. Nevertheless, mailing lists are a significant source of information on open source projects [37] and are still actively used in many projects.

The research on developer information needs using mailing list data is focused on analyzing developer communication [65, 81] and categorizing of information seeking methods [77–79]. Mailing lists have the advantage that the communication records often span multiple years and are related to one specific project. This allows researchers to uncover the evolution of aspects of developer information needs and how they differ in multiple stages of a project. Sharif *et al.* characterize mailing lists as "*the backbone of open source communications*" [78], pointing to the fact that many long-lived open source projects used and still use mailing lists as their primary communication tool between developers.

Despite the significant amount of communication logs, researchers have several challenges in using mailing list data. Four out of the six analyzed studies extract the data manually, which is a time-intensive task and yields little data. If available, the extraction of *MBOX* files, which contain large parts of a mailing list, can be an efficient method to collect a large amount of data with little effort. Two studies in our set of literature used this possibility [65, 81]. Unfortunately, the mail's content in mailing list is often bloated

[1] `https://lkml.org/`
[2] `https://mail-archives.apache.org/mod_mbox/`

with footers, automatically generated content or stack traces that need to be cleaned from the dataset. As mailing list data consists only of unstructured text (*i.e.*, no tags or assigned topics), the data is difficult to split semantically or to extract specific topics.

Observation 3.2 *Mailing lists* are a valuable resource to research developer communication and information seeking. Long-lived open source projects using mailing lists offer a long timespan of data. The unstructured nature of emails poses challenges for researchers.

3.3.3 Bug Reports

Similar to mailing lists, bug tracking systems, which handle bug reports, are essential for the communication between developers or developers and the community[21]. Bug reports offer two different ways to help developer information needs. First, a developer fixing a bug can find detailed information about the environment, configuration and circumstances that lead to the bug, providing crucial information to fix the bug. Second, a user posting a bug report can obtain valuable feedback from developers on how to integrate or use the product or software component.

As a user of a software component, it may not be a quick solution to file a bug report, in case the implementation is not working as expected. However, it is a direct way to get in contact with developers and experts and gain more in-depth insight into the functionality of the software. Bug reports (resolved and unresolved) are a valuable source of information for other users that happen to have similar difficulties in implementing or using certain software.

Breu *et al.* show that fixing a bug with bug reports is a collaborative process as the active participation of the developer and user is essential for successfully resolving the bug [21]. Bug reports are special sources of information because their quality and level of technicality can vary greatly. This is because bug reports are mostly formulated by users, which are often not on the same level of understanding the software as that of the developer who fixes the bug. Bettenburg *et al.* have studied the characteristics of a good bug report. They state that bug reports with (i) stack traces, (ii) that are easy to read, and (iii) include code samples have a lower lifetime, meaning they get fixed sooner [15].

Researchers investigating the developer information needs do not commonly use bug reports or bug tracking systems as data sources. We included one relevant study which investigates developer information needs in bug reports [21] and one study which used bug reports as a second source of developers' communication besides mailing lists [65]. In both studies, the bug reports have been extracted manually due to the unavailability of more efficient extraction methods, as well as human interpretation that was needed to extract only interesting parts of the bug report [21]. Nevertheless, bug reports could be a valuable data source for future research as they often are connected to source code artifacts and commits, which can be used to link the communication with specific parts of the source code.

Observation 3.3 *Bug reports* are not that popular to research developer information needs. This is mainly due to the difficult extraction of the data and the needed human interpretation.

3.3.4 Newsgroups

Newsgroups or developer forums are similar to mailing lists in terms of purpose, community, structure, and level of technicality. Developers (or other interested persons) can discuss or ask questions specific to a particular software project or component. Hou *et al.* state that newsgroups can be valuable to debug problems, discussing issues or learning about existing bugs [41]. Developers often receive quick answers from the community for their information need.

We included two studies in our relevant literature that used data from developer newsgroups [40, 41]. Both studies extracted the data from the newsgroup sites manually and performed a manual analysis to investigate developers' challenges with a specific software component. The research using newsgroup data is rather old. Whether researchers shifted away from using newsgroups as a data source or started to analyze different aspects is unclear.

3.3.5 User Reviews

Compared to mailing lists or newsgroups, user reviews are a relatively new source of information. It is a new source of information for researchers since app stores became only available with mobile devices' upcoming usage. In terms of structure, community, and level of technicality, user reviews are entirely different from the other sources. A user review consists only of text, potentially combined with a rating. No discussion, no code snippets, no stack traces, just a statement from mostly non-technical users regarding a specific software product.

App developers compete intensively for users, continually improving the product to avoid losing market share among similar apps. Therefore, user reviews are an essential tool to get feedback from users, receive feature requests or detect possible bugs. Ultimately, developers consider user reviews to drive the development and evolution of the app. Researchers started supporting developers on this issue and have proposed multiple solutions to guide release planning and evolution [63, 89] or provide a meaningful grouping of reviews [23, 31].

The simple structure of user review data is an advantage of this source of data. Additionally, the lack of source code artifacts makes the preprocessing straightforward and allows the handling of all user reviews as natural language. Analysis techniques like sentiment analysis are ideally suited for user reviews in order to provide developers with more in-depth insight into their reviews [31]. The lack of an efficient extraction method is a significant drawback for researchers. None of the big app marketplaces provides an API or publishes a dataset with user review data. Thus, five of the eight selected studies in the relevant literature use web scraping as the data extraction method. This extraction method is not only time-intensive to perform and hard to reproduce but Martin *et al.* state that many of those studies suffer from a sampling bias which can affect the results [54].

Observation 3.4 *User reviews* or app reviews are a more recent source of developer information needs data for researchers. The simple structure of user reviews is convenient for researchers, but the lack of efficient extraction methods poses challenges.

3.4 Discussion

In this chapter, we have identified and characterized five sources of data for studies about developer information needs. The sources differ significantly in structure, community, level of technicality, possible methods of extraction, and use in research. The main finding is that Stack Overflow dominates the other sources in terms of studies conducted using its data. The main reasons for the popularity of Stack Overflow as data source are (i) the ease of extraction, (ii) the amount of data available and (iii) the diversity of the data available. Researchers can extract a large dataset with little effort about various aspects of developer information needs from Stack Overflow.

Other sources of information (*e.g.*, user reviews, bug reports) do potentially offer a large amount of data but are difficult to extract. This leads to incomplete and hard to reproduce datasets, possibly posing a threat to the validity of such studies. Sources that are often extracted manually (*e.g.*, mailing lists and newsgroups) yield only small datasets, limiting the possible methods of analysis of the data and insights that can be generated.

The described sources do also offer different views on developers' work. For example, Stack Overflow offers a window to developers' internal perspective, while user reviews provide information on the communication between users and developers.

Making more of the described sources as available as Stack Overflow could greatly benefit researchers as well as developers. We have also seen that the choice of data source heavily influences the further methodology of the studies because the structure, amount of data and content are vastly different. In the next chapter, we want to investigate the selected studies' methodologies, focusing particularly on the reproducibility aspect.

4

Reproducibility of Studies of Developer Information Needs

4.1 Overview

With the increasing usage of collaborative platforms by developers, researchers started conducting experiments on these platforms (identified in *RQ1*) to understand different software development aspects. However, which common steps and patterns such studies have, and which challenges researchers face in conducting these studies is explored only in a few studies.

Bettenburg *et al.* previously identified several challenges when processing unstructured data from mailing lists and highlights the need for preprocessing for such studies [17]. Similarly, Chen *et al.* [24] pointed out the inconsistency in performing and documenting the data preprocessing steps across current research. Amann *et al.* shed some light on the issue of reproducibility in software mining studies [8]. González *et al.* highlighted various essential factors, including data preprocessing step for reproducing a study [34]. Our analysis focuses on the methodology prior to the analysis part of each project, meaning *the extraction, management and preprocessing* of the data. The *goal* is to find similarities and shared workflows that can be described and reused for later work. Additionally, we investigate different aspects of reproducibility in the analyzed body of literature. As the data on these collaborative platforms is growing, also the insights researchers can gain from it can evolve. As Menzies *et al.* mentioned in their *Seven Principles for Software*

Analytics that the system (made for data analysis) should be able to repeat the data analysis and the growing size of the dataset should be considered [57]. Therefore it is of interest to have an efficient and comparable research methodology which can be applied to (i) the same dataset on different points in time or (ii) to similar datasets from different platforms.

4.2 Methodology

For this chapter, we analyzed the same literature used in chapter 3 (see Table A.1 for the full list). González characterized the reproducibility of a study in empirical software engineering studies [34]. We used parts of their proposed elements such as *Data source*, *Retrieval methodology*, and *Extraction methodology*. Additionally, we are interested in gathering knowledge about how the dataset is managed throughout the study and which tools are used at various steps. Unlike their study, we dissected the extraction methodology element into individual elements such as data extraction, data preprocessing, and data management. We carefully analyzed how the data in each paper was extracted, managed and preprocessed. We extracted the following methodological steps (elements) from each paper:

- **Data Extraction**: Depending on the data source, ways of extracting data and the tools used to extract it can vary. We collect the data source, the extraction methodology, and the tools used to extract data from the data source. We assess how well the extraction can be reproduced.
- **Data Management**: After retrieving the raw dataset, it can be stored in a database or as files to facilitate various preprocessing and analysis steps. We gathered this information by asking the question: How was the data stored and managed? We extract how the data was stored and managed throughout the project and which tools were used.
- **Data Preprocessing**: How was the data preprocessed? Which preprocessing steps have been performed, and how are they documented? We extracted all preprocessing steps and assessed how they are documented. We distinguished between simple text-cleaning steps and more sophisticated natural language cleaning steps and extracted which tools were used to support the process. Gupta *et al.* identified various language cleaning steps and techniques used in the area of mining repositories [36]. We identified these techniques in our study.
- **Dataset Availability**: Is a dataset made public by the authors? We extracted whether a dataset with the data from the presented study is officially made public.
- **Analysis**: What kind of analysis is being done with the data? We identified and differentiated various kinds of analyses presented in the papers. We differentiated between categorization, topic modeling, empirical analysis and other methods. We extracted tools that supported the analysis.

We analyzed reproducibility with respect to data extraction, preprocessing and dataset availability. We combined findings concerning reproducibility from those three sections into a section only concerning reproducibility.

4.3 Results

In this section, we want to present insights gathered from the body of literature. To identify the common practices and assess the reproducibility of the studies reported in the area of empirical studies performed on collaborative platforms, we extracted various elements from the selected literature.

4.3.1 Extraction Methodology

Gathering data for analysis can be a tedious task. Depending on the source, some extraction methods are not possible; for example, when there is no API or existing data dump provided, researchers are forced to extract the data manually or build a web scraper. The extraction method is essential for general reproducibility and credibility of a project. If the data is gathered manually (*i.e.*, manually copying text from the web), then it is hard to redo the extraction later in time or to replicate the methodology.

Analyzing the data extraction element, we found the following categories to classify extraction methods:

- **web-scraping**: Building or using a tool that is automatically accessing resources on the web and extracting data.
- **API**: Using an official or third-party API to collect data from a source. Popular developer services offer APIs to access their data or even interact with their service.[1,2,3]
- **data dump**: Using an existing dataset or database which is made available by the source itself or other researchers. This extraction method also includes MBOX datasets, which are often used when analyzing data from mailing lists. Most known and often used is the Stack Exchange Data Dump[4], which is officially provided by Stack Exchange and updated regularly.
- **query-tool**: Using a specialized tool to access the data source directly or directly query a database. An example is the Stack Exchange Data Explorer.[5]
- **manual**: Using no tool, API or pre-existing dataset and manually collecting data from the web.
- **not specified**: The literature gives no insight into how the dataset was acquired.

Figure 4.1 shows the distribution of the extraction methods used in the analyzed body of papers, on which we can see that 45% of the analyzed studies use *data dump* as extraction method. This high proportion of *data dump* is mostly due to the highly popular Stack Exchange Data Dump. The popularity of the Stack Exchange Data Dump is achieved through (i) ease of access and (ii) large amount of data. Only the Stack Overflow Posts dataset is currently more than 80 GB in size and offers valuable data for a diverse set of research questions.

[1] `https://developer.github.com/v3/`
[2] `https://developer.atlassian.com/server/jira/platform/rest-apis/`
[3] `https://api.stackexchange.com/`
[4] `https://archive.org/details/stackexchange`
[5] `https://data.stackexchange.com/`

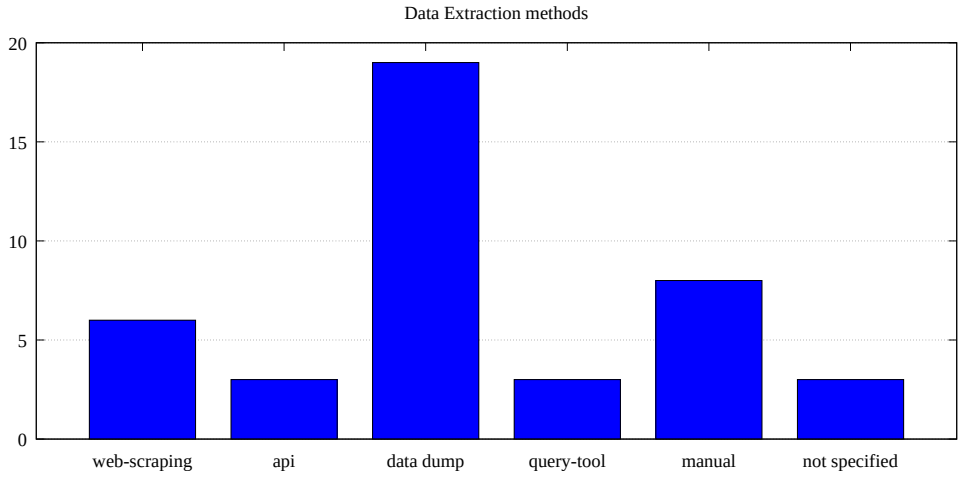


Figure 4.1: Number of papers with the corresponding method of data extraction used

Observation 4.1 *Data Dump* extraction method is the most used because the Stack Overflow Data Dump yields much data with little effort.

A more surprising result in the extraction methodology is that 19% of the papers report manual extraction of data, meaning that no tool, API or preexisting dataset was used to collect the data. The data sources of those papers are bug reports [21], mailing lists [77–79, 81], newsgroups [40, 41] and app reviews [23]. The main reasons stated for using a manual data extraction process are twofold. Either there is no API or dataset available [21][41][40][23] or the extraction process is too complex to automate [77][79]. A good example of necessary manual extraction is given by Sharif and Buckley, which analyze mailing list data, they state that "*initial investigations showed that most of the questions in programmers' emails were asked without explicit indicators like question marks or signaling words such as 'what, where…'. As a result, the questions in the mailing list had to be extracted manually*" [77]. This shows that although there are APIs and tools available to gather unstructured data, it is not always possible to automate the process entirely.

Observation 4.2 *Manual Extraction* is the only option in certain use cases because human interpretation of the data is needed.

Where no existing dataset or API is available, but manually collecting data is too time-consuming to do and yields too little data, researchers build web-scrapers or automated crawlers. Despite the presence of convenient methods for web scraping (*e.g.*, Beautiful Soup for Python[6]), the process of building a working and efficient web scraper is not easy. Modern web sites have a complex and often dynamic structure, which makes them hard to be scraped to extract data. These web sites are built to make human interaction

[6] `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`

as easy as possible, which does not facilitate programmatic access. Additionally, there are some legal questions when it comes to web scraping. The *Terms of Use* of a web site often prohibit access for web scrapers and other means of programmatic access to the resource. However, the enforceability of the *Terms of Use* can be unclear [49]. Also, the web resource can contain copyrighted material, which is not allowed to be scraped or republished. On community platforms, the data generated by its users does not necessarily belong to the platform itself. The web scraper also must not degrade the usage of the web site for other users. An unrestricted web scraper can effectively capture most resources of a web site and make it unresponsive for other users or disrupt the service entirely. The legal questions around web scraping often leave the usage of such scrapers in a gray area; the legal landscape around this topic is still evolving [80]. Overall, there are various challenges to build or use a web scraper, but it often remains a valid option to gather a large amount of data from a web resource. In the analyzed literature, we saw that web scraping is commonly used for gathering data in the area of developer information needs. Especially for app review sources, the use of web scraping is high. Overall, 14% of the analyzed papers used or built a web scraper to collect data.

Observation 4.3 Despite technical and legal challenges, *Web Scraping* is regularly used by researchers, especially for mining app reviews.

4.3.2 Preprocessing Methodology

In terms of data preprocessing, we analyzed the aspects of (i) *text cleaning*, (ii) *language cleaning* and (iii) *tool usage*. We distinguish *text cleaning* from *language cleaning* as follows: *Text cleaning* is only concerned with removing language-independent artifacts from the unstructured text. Examples for *text cleaning* are removing HTML tags, removing code snippets or punctuation and special characters. *Language cleaning* is concerned with normalizing human language data to facilitate analysis and minimize noise in the data. Examples of *language cleaning* are stop word removal or word stemming. In *tool usage*, we identified the tools used for *text cleaning* or *language* cleaning.

We found that 31% of the analyzed papers performed at least one *text cleaning* step. The most performed *text cleaning* steps were *HTML tag removal* (24% of studies) and code removal (17% of studies). Those steps are dependent on the data source. All papers that reported to have performed *HTML tag removal* have Stack Overflow as a primary data source. Not surprisingly, the Stack Overflow data coming from the API, the query tool and the official data dump contains HTML in its post body, which should be removed before doing any analysis that works with natural language. On the other hand, not all papers with Stack Overflow as data source report to have done any *text cleaning*. Looking further into those papers, we found that all papers that performed manual classification of data (*e.g.*, open coding) do not perform any *text cleaning* as it is generally not needed. Papers that reported a manual extraction of data do not report using any *text cleaning* except for one paper, which applies transformations to lowercase words and removal of non-alpha-numeric characters to get more uniform data [23].

Observation 4.4 *Text Cleaning* steps depend on the data source and the extraction methodology. In case of manual analysis of the data, preprocessing is often completely omitted.

We found that 60% of the papers report on having done at least one *language cleaning* step. The most performed *language cleaning* steps are removing stop words (48% of studies) and applying stemming (33% of studies). We have found that *language cleaning* depends on the kind of data analysis, for example, whether automated analysis (*e.g.*, topic modeling) is done with the data. This is comprehensible because certain analysis techniques require data in a specific form; otherwise, the analysis will yield unclear or noisy results. 76% of the papers that do at least one *language cleaning* step perform a topic modeling analysis with the data. None of the papers which conducted manual analysis performed any *language cleaning* steps.

Observation 4.5 *Language Cleaning* is mostly applied when automated analysis is performed with the data.

In terms of tool usage, we found that only 24% reported using any tools or libraries for the preprocessing. Moreover, we found that the reported tools are mostly used for *language cleaning* steps and not for *text cleaning* steps. Tools that were used to conduct the analysis, after the preprocessing, were reported significantly more often (40%). The tools that were reported most often are (i) The Stanford parser[7] for natural language parsing, (ii) the Python NLTK[8] and (iii) Porter Stemmer[9]. Some papers mention having developed and used their own tools but do not elaborate on details.

Observation 4.6 *Tool usage* is reported for specific *language cleaning* steps, but not for *text cleaning* steps or managing the full preprocessing workflow. Tools for analysis are reported significantly more often.

4.3.3 Data Management

Data management has no direct impact on reproducibility. However, it has effects on how well researchers can interact with their data, how it can be shared among the team and how the study is organized. Unfortunately, we have found only few insights into data management in the analyzed literature. We have found that 79% of the studies do not report how the data is managed. In 14% of the studies, the authors explicitly mention that a database was used to store the data, and 7% reported using spreadsheets.

Observation 4.7 Aspects of *Data management* are mostly not reported in the analyzed literature.

[7] https://nlp.stanford.edu/software/lex-parser.shtml
[8] https://www.nltk.org/
[9] https://tartarus.org/martin/PorterStemmer/index.html

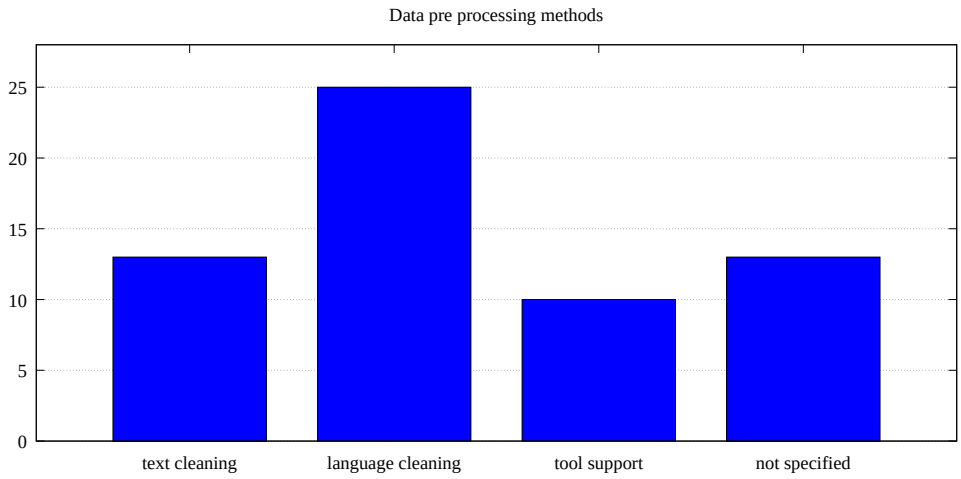


Figure 4.2: Multiple aspects of the preprocessing quantitatively compared

4.3.4 Dataset Availability

Providing a dataset to support reproducibility is considered good practice. A publicly available dataset allows other researchers to validate results and to reproduce the study. Also, we have already seen that data extraction from some sources is difficult to do and even more difficult to reproduce. Therefore, it is of great importance to publish datasets for any source used in a study. *Dataset availability* is an important aspect of reproducibility. Of course, it is not always possible to publish the full dataset of a study due to privacy or confidentiality concerns. However, this does not apply in the analyzed literature, since all datasets are extracted from publicly available data. In our analysis, we did not distinguish between partial and full datasets. Also, we did not distinguish whether a published dataset is the resulting, the extracted or some intermediate dataset. We assumed that any dataset can support reproducibility and hence, is a valuable part of the study.

Despite our low requirement to *Dataset availability*, we have found that only 38.1% of the analyzed studies do publish a dataset. This number is unexpectedly low and identifies an issue that should be addressed in the research community.

Observation 4.8 Only 38.1% of the analyzed studies publish a dataset.

4.3.5 Reproducibility

We analyzed how well the methodology in terms of data treatment is reproducible based on the gathered data extraction, data preprocessing, and dataset availability information from the literature. Reproducibility is essential to confirm the results, to apply the same methodology to different data or also to perform different analyses, but with the same data. Especially for studies on data from collaborative platforms,

it can be of high interest to perform similar studies later in time as the data generated by the users is constantly growing and ever-evolving. Menzies *et al.* also highlighted similar concerns in their study [57]. Comparing results or showing how results evolve over time can be of great interest for researchers, but this is only possible if the studies can be reproduced in a similar fashion, including relatively exact reproducibility of the dataset used for analysis and answering research questions.

Based on three aspects, we extracted a general reproducibility score from each paper. The aspects we took into account were the following:

- **Dataset availability** This aspect shows whether the paper makes any dataset available. Many authors decide to make a replication package available for other researchers. The dataset's availability supports reproducibility as other researchers can confirm the results or use the dataset for other research questions.
- **Extraction reproducibility** This aspect shows whether data extraction can be considered reproducible. Reproducible data extraction clearly states the data source, extraction method and parameter. It is not needed, but the availability of a data extraction script or similar is helpful for data extraction reproducibility. Manual extraction is not considered reproducible.
- **Processing reproducibility** This aspect shows whether the performed preprocessing steps, if performed, can be considered reproducible. A reproducible preprocessing clearly states all processing steps and the order in which they are performed. All processing steps need to be either unambiguous, clearly defined how it was performed, or tool usage is specified. An unclear or ambiguous processing step description results in non-reproducible processing.

We analyzed the above aspects for all papers in the body of literature and rated them with a binary score. The results from the analysis of the three reproducibility aspects are shown in figure 4.3.

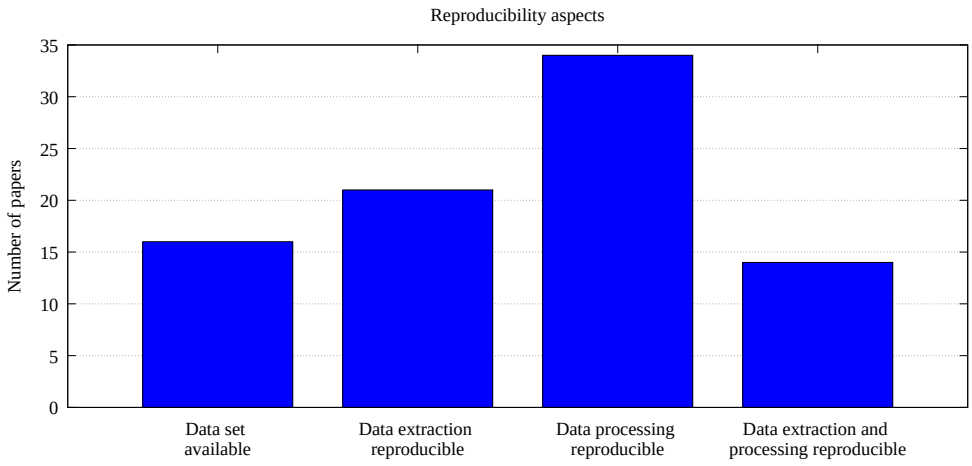


Figure 4.3: Reproducibility analysis

We have found that only 38% of papers publish the used or generated dataset officially in their paper. In our opinion, this is a rather low number, as it is normally not difficult to publish a replication package and to allow other researchers to confirm or reuse the data and analysis.

Observation 4.8 Only a few studies make a dataset publicly available.

For data extraction, we have found that 50% papers report a reproducible data extraction. When looking further into this aspect, we see that 76% of them use a data dump as data source, which can be easily reported in a reproducible way. It is clear that manual extraction is very hard to do in a reproducible way, but in that case, it would be advisable to make the dataset available. Looking only at papers that use a hard to reproduce extraction method (web scraping and manual), we found that 7 of 14 papers make the dataset available. This is a better ratio than overall, but still half of the papers miss the chance to make their work reproducible by providing a dataset when their extraction method is hard to reproduce.

Observation 4.9 *Dataset availability* is higher among studies that used a hard to reproduce data extraction method. Nevertheless, those extraction methods are mostly reported insufficiently to be adequately reproduced.

For data processing, we found that 81% of papers report reproducible data processing. A significant proportion of those papers (38%) that report reproducible data processing are papers which did not report any reproducible preprocessing. From the papers which actually take preprocessing steps, we found 21 from 29 relevant papers describe the process in a reproducible way.

Additionally, we can look at the combination of extraction and processing reproducibility to show how many of the papers report a complete workflow in a reproducible way. One could argue that it is not of much use if the data extraction is reproducible, but the preprocessing not, as well as the other way around. We have found that only 33% of the papers have both reproducible data extraction and data processing.

The sum of the three scores serves as a simple reproducibility score that summarizes the overall reproducibility of the analyzed body of literature. The results are shown in figure 4.4.

We found only one paper [96] with a score of 0, meaning that this paper does not report anything that supports reproducibility.

A total of 43% of papers have a score of 1, which means they fulfill only one of the three aspects analyzed. Generally, this is not enough to allow other researchers to reproduce the dataset and results reliably.

38% of the papers have a score of 2, which indicates already a moderate reproducibility. In fact, all of the papers with a score of 2, report a reproducible data processing method. This means that either the dataset is provided or the extraction is reproducible, in both cases, the resulting dataset should be reproducible fairly well.

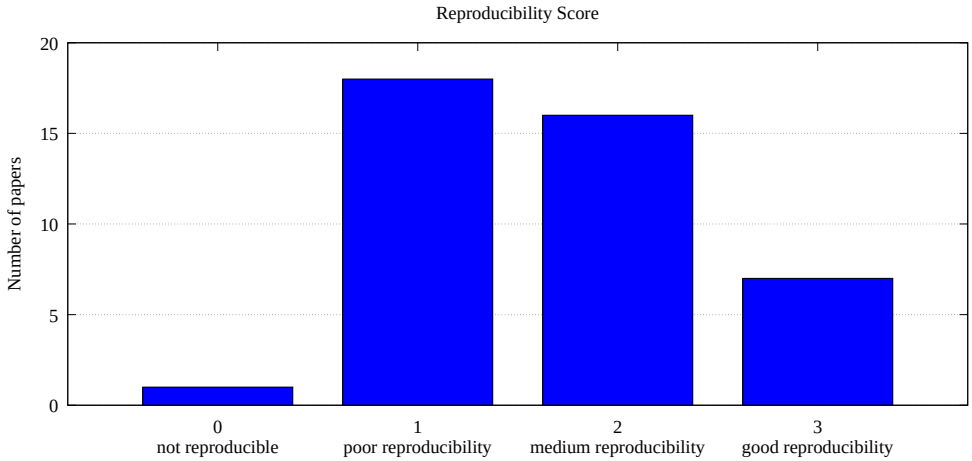


Figure 4.4: Reproducibility Score

17% of the papers have a score of 3, which indicates good reproducibility. With a given dataset, as well as reproducible extraction and processing, it is possible for other researchers to validate the results, to apply different research questions on the same dataset or to perform the same analysis with updated data.

Observation 4.10 Most of the analyzed studies report their methodology insufficiently to be reproduced properly. This makes it difficult for researchers to validate the results, reuse the dataset or apply a similar methodology to a different dataset.

4.4 Discussion

In this chapter, we have investigated different aspects of the methodology of papers that address questions about developer information needs. By analyzing a relevant body of literature, we have shown that researchers use several different data extraction methods. The ease of extraction from data dumps as the Stack Overflow data dump leads to many research projects using this data to answer various research questions. Especially if the study includes an analysis part like topic modeling, where a large dataset is mandatory to yield consistent results, it is crucial to have an accessible and automated data extraction method in place. Projects with manual analysis methods (like Open Coding for taxonomy creation) tend to also use manual data extraction as the size of the dataset is not as important.

The analysis of the data processing has shown that depending on the data source and later analysis, the preprocessing steps are very similar in many of the papers. While the *text cleaning* steps are highly dependent on the data sources, the *language cleaning* steps depend on the analysis to be done with the data. We have identified common preprocessing steps like removal of source code, HTML, and punctuation, stop word removal and word stemming, which are present in a large part of the analyzed papers. These

preprocessing steps can be seen as a common workflow for research projects that investigate developer information needs from collaborative platform data. The common workflow is illustrated in Figure 4.5.

Figure 4.5: The study workflow found in the studies investigating developer information needs

Although the processing done in the various projects is often similar, the papers often do not report tool usage. While tools and libraries for specific preprocessing steps (mostly for *language cleaning* like stemming or stop word removal) are mentioned, a tool that supports the whole process cannot be identified.

The analysis of reproducibility aspects has shown that there are big differences when it comes to reproducibility of the datasets. While reproducibility is generally desirable for all projects, it does not seem to be an important aspect of all papers we have analyzed. Some researchers provide a replication package that is often extremely helpful in understanding and reproducing a project's methodology. However, more often, the reported methodology is not as exact as it should be, leaving open some ambiguities or questions which lead to uncertainties when comparing or reproducing a project. Concerning reproducibility, we have not found common practices, as in the data processing, which lead to comparable methodology.

5

Case Study: Code Commenting Conventions

5.1 Overview

As discussed in the previous chapters, developers use multiple sources to understand code artifacts and implement changes. Usually, before consulting any external sources, developers look at documentation and the source code, including code comments. Code comments are considered to be the most important type of documentation for understanding code [26], and studies in the past have shown that commented code is easier to comprehend than uncommented code [83, 95]. Code comments are written in natural language and easily understood by other developers (or the author at a later point in time). Due to the nature of code comments, the syntax and style used by developers to write code comments cannot be checked or enforced automatically. Thus, it is a challenge to achieve consistent code comment practices and conventions across a project [5, 45].

The community of developers, organizations and language designers has identified the need for consistent code comments. The official documentation of programming languages like Java[1] or Python[2] includes sections about code commenting conventions. Also, large organizations publish their own style guidelines

[1] `https://www.oracle.com/technetwork/java/javase/documentation`
[2] `https://www.python.org/doc/`

(*e.g.*, Google[3], Eclipse[4] and Apache (Project Spark)[5]), addressing code comments among other code conventions. These published guidelines and common practices are picked up by developers, who try to follow best practices in the programming language or technology they use. The guidelines do not present strict rules, and tool support to enforce them is limited. Thus developers get confused about which convention to apply in a specific scenario. Allamanis *et al.* defined this as the convention interference problem [5].

This uncertainty leads developers to ask coding style related questions to their peers, on project-specific platforms and Q&A sites. On Q&A sites like Stack Overflow and Quora, we can find many questions related to code commenting conventions. Table 5.1 contains examples of questions from developers that seek information about code commenting practices. From the examples, we can also already see that the questions on Quora and Stack Overflow are different. However, what kinds of questions about commenting conventions are asked and to which extent the questions differ on these Q&A sites is not yet explored. In this case study, we explore unstructured data from Stack Overflow and Quora to find questions about code commenting conventions.

Source	**Question**
Stack Overflow	*How to "comment-out" (add comment) in a batch/cmd?* [105]
Stack Overflow	*How to properly document keys of array return type?* [106]
Quora	*What should I comment on code? What, how, or why?* [103]
Quora	*What's a good comment/code ratio?* [104]

Table 5.1: Examples of *code commenting conventions* questions found on Q&A sites

Identifying and investigating these questions can support:

- **communities and organizations** to uncover frequently asked questions about code commenting conventions and thus improve the style guidelines by addressing such questions,
- **developers** by presenting the most significant issues concerning code commenting conventions and facilitating them to find good practices,
- **tool developers** by providing a direction to improve the existing documentation tools and to design the new tools for verification of code comments style, and
- **researchers** by uncovering insights that enable future research in the direction of multi-source analysis and development activities.

The research questions we want to address in this case study are the following:

1. *What are the topics covered by developers about commenting conventions?*
   To answer this question, we extract all code comment related questions from Stack Overflow

[3] `https://google.github.io/styleguide/`
[4] `https://wiki.eclipse.org/DocumentationGuidelines/StyleGuidelines`
[5] `https://spark.apache.org/contributing.html`

and apply a topic modeling techniques called *Latent Dirichlet Allocation* (LDA) to find different emerging topics of questions.

2. *What are the types of questions and problems developers discuss on various platforms?*
   To answer this question, we extract code comment related questions from Stack Overflow and Quora. We manually analyzed a statistically significant sample set of questions and categorized them in the dimensions of question type and information type.

By investigating these research questions, we can identify common issues about *code commenting conventions* discussed by developers on Q&A sites. We can identify the shortcomings of currently available tools and practices on this topic and propose solutions to address them. Also, we can get insight into how Stack Overflow and Quora can serve as complementary sources of developer information.

5.2 Methodology

To investigate the high-level topics in code comment convention questions, we extract the relevant tags and their questions from Stack Overflow and use a topic modeling technique called LDA. The results from LDA are used to answer *RQ1*. To apply LDA, we did not include questions from Quora due to the low number of relevant tags and questions available in Quora. To answer *RQ2*, we extend the dataset from Stack Overflow with relevant posts from Quora to achieve a more significant diversity of questions. We manually analyzed a statistically significant sample and extracted the question type and information type of the questions in the sample. The methodology to answer both *RQ1* and *RQ2* is illustrated in figure 5.1. More details about the methodology are discussed in the following sections.

We perform automated analysis (LDA) and manual analysis to get a more complete picture on *code commenting conventions*. LDA can grasp high-level and non-hierarchical topics. The subtle differences of meaning in questions about *code commenting conventions* and the diversity of the terms in *comment* and *convention* tags make it hard to extract a more fine-grained and meaningful list of topics with LDA. Therefore, we use manual analysis to to understand the topics in details, to uncover connections between topics, and to get a more differentiated view on the data.

The data handling in this case study, including extraction and preprocessing, has been done with a prototype tool called *Makar*, which we implemented using Docker, to facilitate its usage and the replicability of the analysis performed in this study. Thus, the tool helped to process our data in a consistent and reproducible way. It allowed us to manage data from different sources in the same tool, apply common preprocessing steps, and to publish a consistent and reusable dataset. In the detailed description of the methodology, we will make references to functionalities used in the tool, in order to achieve complete documentation of the process. We present a complete description of the tool in chapter 6.

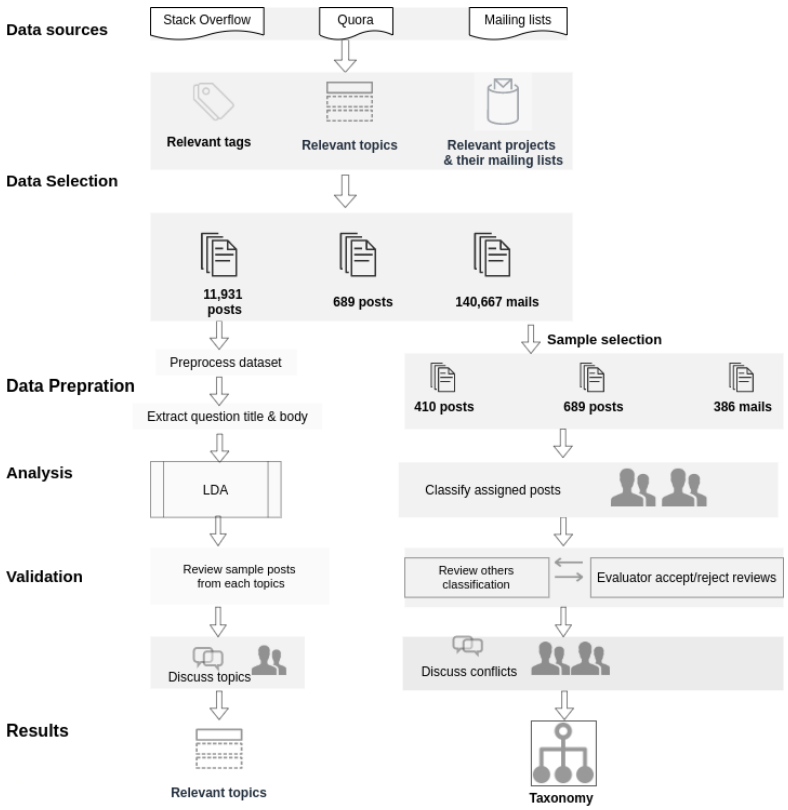


Figure 5.1: Case Study Pipeline

5.2.1 Data Selection

5.2.1.1 Stack Overflow

The data from Stack Overflow was collected using *Stack Exchange Data Explorer*.[6] This official tool from Stack Overflow allows one to query all Stack Exchange sites in an SQL-like Query language. We extracted all questions and answers with at least one relevant tag with the queries shown in figure 5.2 and the queries are accessible by permalink: *questions*[7] and *answers*[8]. For each question and answer, we extracted body, title and additional metadata. We extracted all tags separately because there is a limit on how many records can be extracted at once. All extracted attributes can be found in Table 5.2.

To define the set of relevant tags for commenting conventions, we used a hybrid approach inspired by the previous studies by Yang *et al.* [98] and Emad *et al.* [1]. The hybrid approach consists of extracting the tags based on the defined keywords and a heuristic to find additional relevant tags. We defined the

[6] https://data.stackexchange.com/

[7] https://data.stackexchange.com/stackoverflow/query/1236330/stack-overflow-questions-with-a-specific-tag

[8] https://data.stackexchange.com/stackoverflow/query/1236333/stack-overflow-answers-with-a-specific-tag

```sql
SELECT
  P.Id,
  P.ParentId,
  P.CreationDate,
  P.ViewCount,
  P.Title,
  P.Body,
  P.Tags,
  P.PostTypeId
FROM
  Posts as P
WHERE
  -- Set your specific tag here
  P.PostTypeId = 1 AND P.tags LIKE '%<comments>%'
ORDER BY
  P.ViewCount DESC;
```

(a) Extract all question for a specific tag

```sql
SELECT
  PP.Id,
  PP.ParentId,
  PP.CreationDate,
  P.ViewCount,
  P.Title,
  PP.Body,
  P.Tags,
  PP.PostTypeId
FROM
  Posts as P
JOIN
  Posts PP ON PP.ParentId = P.Id
WHERE
  -- Set your specific tag here
  PP.PostTypeId = 2 AND P.tags LIKE '%<comments>%'
ORDER BY
  P.ViewCount DESC;
```

(b) Extract all answers for a specific tag

Figure 5.2: The Stack Exchange Data Explorer allows SQL-like data extraction.

Name	Description
Id	ID of the post
Title	Title of the post
Body	Body of the post
Tags	Tags of the post
CreationDate	Creation date of the post
ViewCount	Number of views for that post

Table 5.2: Extracted attributes from Stack overflow

keywords as *comment* and *convention*. We searched the keywords on the Stack Overflow tag page[9] to get the tags that contain the keywords. We collected 26 tags from this search. We assessed the relevancy of the tags by looking at their description and the top ten questions belonging to each tag and thus filtered the set of tags by removing the tags we considered irrelevant. As both keywords *comment* and *convention* are widespread in various contexts, we had to discard many tags that were not relevant for our study. For example, *facebook-comments*, a popular tag that is only concerned with the comment feature on Facebook, but utterly irrelevant to our study. This process resulted in nine relevant tags: *comments, commenting, code-comments, block-comments, autocommenting, comment-conventions, convention, conventions, coding-style*. We name this set of tags as *initial tags*. We extracted all questions that have at least one of the nine *initial tags*, resulting in 6,087 questions.

However, not all *code comment convention* related tags contain the keywords *comment* and *convention*. For example, the question "*How to write documentation comments in ANSI C?*"[10] is highly relevant but does not contain any tags including one of the keywords. We applied the heuristics used in previous work [98] to find other relevant tags. We extract all tags from the questions that contain at least one of the *initial tags* and we name this set of tags *candidate tags*. The *candidate tags* can be described as all tags co-occurring with the *initial tags*. For each *candidate tags t* we compute three values:

- ***a*** is the number of questions from the initial question set whose tags contain *t*

[9] `https://stackoverflow.com/tags` (Verified on 2020-02-20)
[10] `https://stackoverflow.com/questions/8889942`

- ***b*** is the number of questions from all questions on Stack Overflow whose tags contain *t*
- ***c*** is the total number of questions extracted from the *initial tags*. This number is the same for all *t*

Based on these values, we calculate the heuristics *H1* and *H2*:

$$H1 = a/b$$

$$H2 = a/c$$

H1 indicates the exclusivity of a tag *t* to *code comment conventions* topic. The higher the value of *H1*, the more exclusive *t* is to the *initial tags* and therefore to *code comment conventions* topic. A value of 1 means that the candidate tag *t* only occurs in combination with at least one of the initial tags. We select exclusive tags by setting a threshold *Thre1* for *H1* heuristic. For example, by setting *Thre1* to 0.2, a tag *t* that appears together with at least one of the *initial tags* in less than 20% of all questions will be removed.

However, exclusivity alone cannot serve as a good heuristic to find related tags; we also need a measure of how representative a candidate tag *t* is for all questions extracted from the initial tags. Thus, we use *H2* as the second heuristic and define its threshold *Thre2* to filter less relevant tags.

We manually configure both thresholds, *Thre1* and *Thre2*, by inspecting the filtered *candidate tags* for each combination. Table 5.3 shows the resulting tags from different combinations of *Thre1* and *Thre2*. Configuring reasonable thresholds was not an easy task due to the ambiguity present in the tags. For example, *comments* is being used in many different contexts; some of the contexts are not relevant to our study. However, since there is a high number of relevant questions with this tag, we cannot discard it without losing a substantial amount of relevant data. Therefore, we needed to manually exclude certain candidate tags, which appeared as representative and exclusive but were not related to our study. We decided to exclude all tags which are only specific to (i) a tool/service or (ii) a programming language. After manual exclusion of certain candidate tags, we set the thresholds to (0.13, 0.009). These thresholds yielded five additional tags: *documentation, code-documentation, todo, readability, naming*. In the next iteration, we extracted all questions with at least one of the five additional tags, resulting in a total of 11,931 questions.

(*Thre1*, *Thre2*)	Tags
(0.13,0.009)	documentation, todo, code-documentation, readability, naming
(0.13,0.005)	documentation, todo, code-documentation, readability, naming, docblocks, reply
(0.15,0.01)	todo, naming
(0.15,0.009)	todo, code-documentation, naming
(0.1,0.01)	documentation, todo, readability, naming
(0.1,0.02)	documentation, naming

Table 5.3: Set of additional tags for different values of *Thres1* and *Thres2*

5.2.1.2 Quora

Extracting data from Quora is much more complicated than from Stack Overflow. There is no API, no existing data dump or other sources to collect the data easily. Thus we built a web-scraper to extract questions and answers related to specific topics. The scraper is built in Python using *selenium*[11] to enable automated browsing, and *BeautifulSoup*[12] to parse HTML.

Quora's organization of questions is similar to Stack Overflow, but instead of tags, Quora has topics. A question can have several topics. Unfortunately, Quora has no index for all its topics. We mapped the selected Stack Overflow tags to Quora topics by searching the tags in the Quora search interface using the mapping strategy for multiple sources[1]. We determined the relevance of topics by inspecting the first page of questions. We found the following relevant topics with this approach: *Code Comments, Source Code, Coding Style, Programming Languages, Comment (computer programming).*

Firstly, we scraped the complete list of questions from each topic index page. Then we scraped every question found on the topic index. For each question, we extracted the attributes shown in Table 5.4. This resulted in 689 collected questions. We manually analyzed relevancy to our study for all questions and ended up with only 68 questions relevant to code comments. We looked through all topics from the relevant questions but found no other relevant topics in the context of our study.

Name	Description
url	URL of the question
title	Title of the question
body	Body of the question
topics	Topics of the question
answers	All answers to the question

Table 5.4: Extracted attributes from Quora

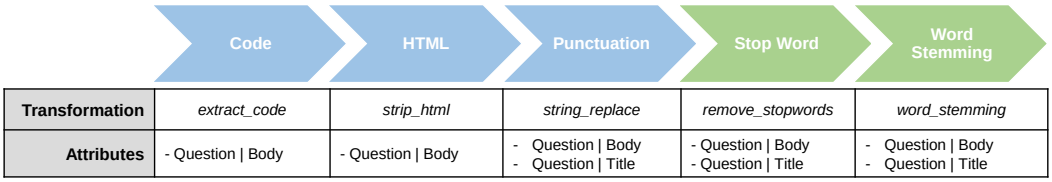
5.2.2 Data Preparation

Data preprocessing is only needed to prepare the data for LDA analysis and answer case study *RQ1*. For manual analysis and building of the taxonomy, the data does not need to be preprocessed.

The preprocessing follows closely the common practices discussed in chapter 4. The data from Stack Overflow contains HTML, code snippets, links and natural language. To produce meaningful results with LDA analysis, the data need to be cleaned and should contain normalized natural language. Figure 5.3 shows an illustration of the preprocessing. All preprocessing steps are performed with *Makar* and its built-in transformations.

- **Code** is removed with the *extract_code* transformation. This transformation extracts `<code>` tags

[11] `https://www.selenium.dev/`
[12] `https://pypi.org/project/beautifulsoup4/`



Transformation	*extract_code*	*strip_html*	*string_replace*	*remove_stopwords*	*word_stemming*
Attributes	- Question | Body	- Question | Body	- Question | Body - Question | Title	- Question | Body - Question | Title	- Question | Body - Question | Title

Figure 5.3: All preprocessing steps with the used transformation in *Makar*

from the given attributes and saves it to a new attribute. For the case study, we did not need the extracted code snippets for further analysis, but this step would allow for easy access to the extracted snippets. This transformation was applied only to the question body as the question title does not contain any code.

- **HTML** is removed with the *strip_html* transformation. This transformation removes all HTML tags from the given attribute. Only the tags are removed, not the content of tags. This transformation was applied only to the question body as the question titles do not contain any HTML tags.
- **Punctuation Removal** is done with the *string_replace* transformation. This transformation accepts a string or regular expression as search parameter and a string as replacement parameter. For this preprocessing step, we replaced `/[.!?\\*\-\_&#$+%()\[\]{}]/` with the empty string, effectively removing punctuation and special characters. This transformation was applied to each question body and title.
- **Stop Word Removal** is done with the dedicated *remove_stopwords* transformation. This transformation removes all stop words from the *Snowball English Stop Word List*[13], effectively removing very common words that would not contribute to usable results for topic modeling. This transformation was applied to question body and title.
- **Word Stemming** is done with the *word_stemming* transformation. This transformation applies the *Snowball Stemming Algorithm*[14], which is an implementation of the *Porter Stemming Algorithm*[15], to the given attribute. This transformation was applied to question body and title.

As described, all preprocessing steps have been done with transformations in *Makar*. The transformations have been set up so that each step produced a new attribute on the data record, allowing us not to lose any information and retrace every change we did on the data. At the beginning of the case study, it was unclear which combination of question and answer attributes leads to the best result. In this phase, the flexible approach to preprocessing found in *Makar* helped us to work with our data and try different options efficiently.

[13] `http://snowball.tartarus.org/algorithms/english/stop.txt`
[14] `https://snowballstem.org/algorithms/porter/stemmer.html`
[15] `https://tartarus.org/martin/PorterStemmer/`

5.2.3 Analysis

5.2.3.1 LDA

To answer the case study *RQ1*, we extracted the high-level topics from Stack Overflow data using LDA (Latent Dirichlet Allocation). LDA is well-suited for discovering topics from text written in natural languages [19] and has also been used in multiple studies [4, 11, 74, 98, 102] analyzed in chapters 3 and 4. LDA operates on a set of documents, which are texts in natural language. We concatenated the *title* and *body* of the 11 931 extracted and preprocessed questions and used that data as documents for LDA. We used *Topic Modeling Tool*[16], which is a GUI for MALLET [56]. *Topic Modeling Tool* allows one to easily extend the results of LDA with metadata, which facilitates the later analysis of results. MALLET is a popular topic modeling tool and has been used in various similar studies [4, 11, 74].

LDA requires a set of hyperparameters to be defined:

- k is the number of topics to be generated
- α represents the document-topic density
- β represents the topic-word density

Based on the nature of the data and the given hyperparameters, the results can differ significantly. For k we tried values between five and 25 to find the suitable number of topics, where the topics are distinguishable by looking at the most important keywords in the topic. We manually analyzed the results for multiple values of k and selected $k = [14, 15, 16]$ as best values for our study context. In the next iteration, we optimized α and β by selecting hyperparameters with the highest average topic probability of the dominant topic among all questions. We found that $k = 14, \alpha = 5, \beta = 0.01$ yields the best results for our defined criteria. This set of hyperparameters resulted in an average topic probability for the dominant topic of 72%. We ran LDA with the selected hyperparameters and for 1,200 sampling iterations to stabilize the results.

LDA produces a set of topics and calculates how likely a specific document (question) belongs to those topics. Each topic has a list of most important topic words and most important documents, ranked by probability of the document belonging to this topic. To give the topics meaningful names, we inspected the most important topic words and the top 15 ranked documents of each topic.

We compared the distribution of the relevant tags from the Stack Overflow questions over the generated topic models. This analysis showed us two things: (i) We see whether a relevant tag is more or less evenly distributed across topics or prevalent in one specific topic. (ii) Additionally, we see which tags are concentrated on a specific topic. This gives us a useful comparison between the semantic topic modeling done with LDA and the user-dependent tag assignment from Stack Overflow.

[16] `https://github.com/senderle/topic-modeling-tool`

5.2.3.2 Taxonomy Definition

To build the taxonomy about *code commenting conventions*, we used the questions from Stack Overflow and Quora. We selected a statistically significant sample set of 410 questions from the 11,931 Stack Overflow questions, reaching a 95% confidence level and 5% error margin. To select 410 representative questions, we considered the distribution of questions based on the number of questions present in a tag. We used a stratified sampling approach with each tag as a strata. From each strata, we selected the proportional number of questions by random sampling without replacement. The Quora dataset contains 689 questions and is much smaller than the Stack Overflow dataset. Thus we were able to inspect all questions manually and classify relevant and irrelevant questions, effectively discarding all questions not related to code comments. This yielded a total of 68 questions relevant to code comments, which were used in addition to the sample of Stack Overflow question to build the taxonomy.

We used *Card Sorting* to classify the questions on two dimensions: question type and information type. The dimension about question type is based on a previous study from Beyer *et al.* about categorization of Android App Development Issues on Stack Overflow [18]. We adapted the categories to our needs and classified the questions according to the following question type categories:

- *Implementation strategies:* The questioner does not know how to implement something. The questions are about how to use features of specific tools or how to integrate solutions into their own code. The questions can be identified by *"How to ... ?"* sentences.
- *Implementation Problems:* The questioner knows how to implement a solution but has troubles in making it work. The questions can be identified by *"What is the problem ... ?"* sentences.
- *Error*: The question contains exceptions, stack traces or errors from the code. The questioner wants to know about a specific error, why it happens and how to fix it.
- *Limitation and Possibilities*: The question is about the possibilities and limitations of a programming language, tool or framework. The questions can be identified by *"Is it possible ... ?"* sentences.
- *Background Information*: The question is about general or detailed background information about a programming language, software component or tool. The question is not related to a specific implementation, problem or error. The questions can be identified by *"Why ... ?"* sentences.
- *Best practice*: The question is about best practice and advice concerning an implementation or solution to a problem. Often, the question presents multiple approaches and intends to identify the most recommended one.

The information type is more detailed, and is determined by inspecting the title and body of the question, extracting the type of information the questioner is seeking. We constructed the categories using *Open Card Sorting*. Three evaluators, all having more than three years of programming experience, participated in the study. We followed an iterative approach to converge on the classification. In the first iteration, we divided all questions into random, equally sized sets. Each evaluator categorized one set of questions in both dimensions. In the second iteration, each evaluator reviewed the classified questions of the other

evaluators. In the third iteration, the initial evaluator accepted or rejected the disagreements and proposed changes from the reviewer. In case the evaluator and reviewer did not agree, the third evaluator reviewed the classification and proposed a final decision, which then was decided by majority voting mechanism.

5.2.3.3 Mailing lists

Despite previous work from Aghajani *et al.* about *Documentation Issues* [1], which we considered closely related to our topic and uses mailing list data, we did not find comment convention related discussions in mailing lists. We mined *@dev* and *@users* mailing lists from multiple ASF Projects. We selected the projects with the highest number of LOC among the projects with the same primary programming language. We selected *Lucene* (Java), *Ambari* (JS), *OpenOffice* (C++), *Cloudstack* (Python) and *Subversion* (C), resulting in a total of ten mailing lists. However, we did not find enough relevant conversations to be viable for this study.

In order to show that the topic of *Code Commenting Conventions* is not discussed in mailing lists, we extracted all conversation threads from five different ASF mailing lists, resulting in 140,667 mail threads. We selected a statistically significant sample of 386 mails, randomly and proportional to the total number of mails per mailing list. We analyzed the sample for any conversation relevant to *Code Commenting Conventions*. We concluded that this topic is not discussed in mailing lists because we did not find any relevant conversation. Thus, we considered questions posted in mailing lists to be out of scope for our analysis, as no relevant information was found.

5.3 Results

In this section we present the results of the automated analysis (LDA) and the manual analysis (taxonomy). The topics from LDA and the categories of the taxonomy evolved mainly from the same dataset (*i.e.*, Stack Overflow dataset), but are not related to each other as the analyses have been done independently. The results from LDA give a high-level overview of topics of questions about *Code Commenting Conventions* and the manual analysis provides more detailed insights into the topic. Nevertheless, it is of interest to compare the results from both analyses.

For both analyses, the proportion of the data that was not relevant to *Code Commenting Conventions* had to be excluded from the results. In the LDA analysis, the resulting topics and its questions were manually analyzed to assess the relevancy of topics. This resulted in *relevant topics* and *irrelevant topics* present in the LDA results. In the manual analysis, the evaluators discarded *irrelevant questions* from the sample set and considered only *relevant questions* to form the taxonomy.

5.3.1 LDA

The LDA analysis yielded the topics shown in Table 5.6. We manually inspected the 15 top-ranked questions of each topic (*i.e.*, the questions with the highest probability of belonging to the topic) to assign

meaningful topic names. During this task, we saw that not all topics can be considered relevant to code commenting conventions. This was expected and understandable since some of the selected tags are very general and ambiguous in nature. Also, we have found similar results in the manual analysis, in which we had to discard a large proportion of questions because they were not relevant to code commenting conventions. Based on the inspection of the 15 top-ranked question, we classified each topic as relevant or irrelevant. This yielded eight relevant and six irrelevant topics.

Finding 5.1 LDA was able to capture the most expected topics like *Documentation Generation*, *Comments Syntax* and *IDE & Editors* and was also able to separate common irrelevant topics like *Comment Functionality on Websites* or *Exceptions*.

Figure 5.4 shows the distribution of questions over the LDA topics. The three topics with the most questions assigned (we consider only the top topic per document) are *Comments Syntax*, *Comment Functionality on Websites* and *Documentation Generation*. We considered two of theses three topics to be relevant. We can see that, except for *Comment Functionality on Websites*, all irrelevant topics have lower numbers of documents assigned. With this insight, we can conclude that our selection of relevant tags to extract data for the LDA analysis yielded a large amount of relevant data.

Finding 5.2 The relevant topics with the highest number of questions are *Comments Syntax* and *Documentation Generation*.

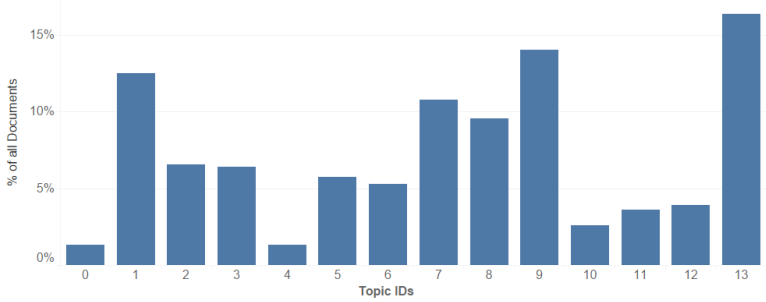


Figure 5.4: Distribution of documents over the generated LDA topics

To investigate the selection of relevant tags further, we analyzed the distribution of those tags over the generated LDA topics. Figure 5.5 shows how each of the selected relevant tags is distributed over the topics. Based on the chart, we observed that (i) we can get an indication about the generality of the selected relevant tags, (ii) we can compare how well the LDA topics match the tag assignments and (iii) we can compare distribution of irrelevant and relevant topics across tags. Concerning the generality of the selected tags, we can observe that the tag *todo* is highly concentrated in topic 2 compared to other topics, indicating that this tag is not used in a general way. On the other hand, we have tags like *commenting* or *comments* which are distributed over almost all topics, indicating that those tags are assigned to a diverse set of

questions and thus used in a more general sense. Overall, we can observe that some of the tags match quite well with the generated LDA topics, meaning that they are present in only two to three of the topics with a high proportion of 30% and more. Examples of such tags are *autocommenting*, *block-comments*, *todo* or *readability*.

Finding 5.3 Tags like *commenting* and *comments* are present in all LDA topics, indicating their generality and ambiguity. These tags do not help to differentiate the topics. Tags that are more congruent with LDA topics are *todo*, *readability* and *block-comments*.

If we compare irrelevant and relevant topics, it is evident that the irrelevant topics generally do not contain significant proportions of one or multiple specific tags (*e.g.*, topics 0, 4, 10 and 11). This can be explained partly because of the lower number of documents in the irrelevant topics, but also indicates that the irrelevant topics mostly contain questions with tags we can consider more general or ambiguous.

Topic #	Relevancy	Topic name	Topic words
0	IR	Exceptions	error naming file fortran color doxygen input command files target stop message words folder make src load open define data
1	R	Documentation generation	documentation sphinx generate api doxygen document web files python html docs file create project code django documenting swagger package output
2	R	IDE & editors	studio comments visual documentation comment todo eclipse xcode xml code android list add intellij 2010 auto tags generate doxygen project
3	R	Processing code comments	comments code comment source file documentation files php add excel write vba python git header xml word adding cell read
4	IR	testing+naming conventions of variables/files	unit tests service called documented number check create list testing data entities stream position byte undocumented hash rspec
5	R	Project documentation	api find software reference user good app offline docs library project version code ruby developer android access application manual
6	R	Project naming conventions	naming convention conventions java class names variable method classes coding whats standard fluent net case nhibernate file methods package structure
7	R	Code entities naming conventions	function variable object naming convention names class variables document javascript method functions practice array good return type data parameter java
8	R	Comments writing strategies	documentation class document method methods comments documenting doxygen xml functions python code function type classes javadoc interface reference properties java
9	IR	Comment Functionality on Websites	comments comment facebook wordpress post page plugin php box form add api system display show jquery working posts count user
10	IR	Frameworks for thread commenting	rails comments ruby controller convention model error commenting mvc action id aspnet create method polymorphic system custom view
11	IR	Database	sql table database mysql comments column comment query server naming tables columns data conventions php convention oracle multiple db cakephp
12	IR	Readability	code readability readable python statements make write statement performance java loop list programming large condition improve strings difference function
13	R	Comments syntax	comments comment line code html file python php block lines javascript regex remove text multi single commenting multiline style vim

Table 5.6: Topics generated by LDA with assigned topic name and most important topic keywords

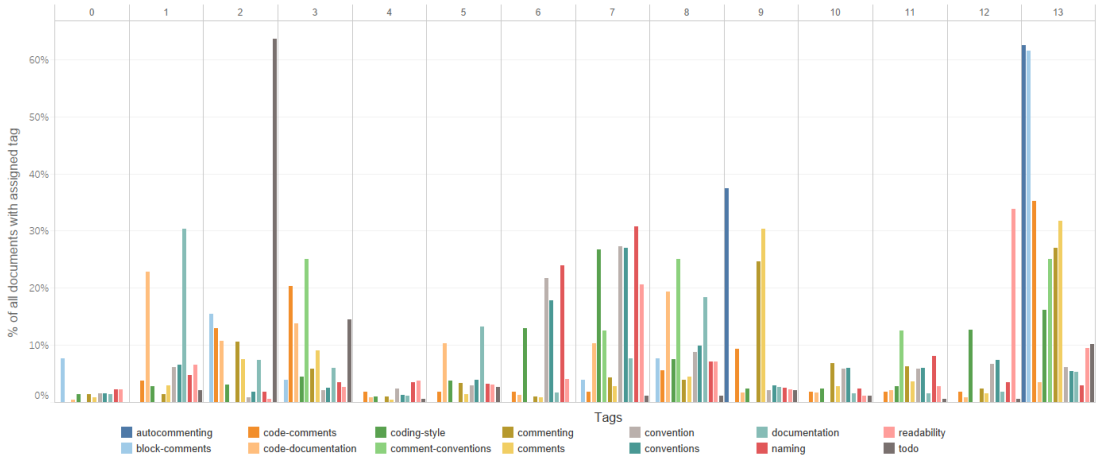


Figure 5.5: Distribution of documents over the generated LDA topics

5.3.2 Manual Analysis

The first result of the manual analysis is the classification of question type. With this data, we can compare questions on Quora and Stack Overflow. Figure 5.6 shows the percentage of questions for each question type on Quora compared to Stack Overflow. We know that Quora is an opinionated Q&A site, but we observe that other types of questions such as *Background Information* and *Best Practice* can be found on Quora. As expected, the more hands-on question types like *Error* and *Implementation Problems* remain exclusive to Stack Overflow.

Finding 5.4 Quora and Stack Overflow clearly differ in the question type. Quora's questions are mostly of type *Opinion* and *Background Information*, while Stack Overflow has *Implementation Problems* and *Error* type questions.

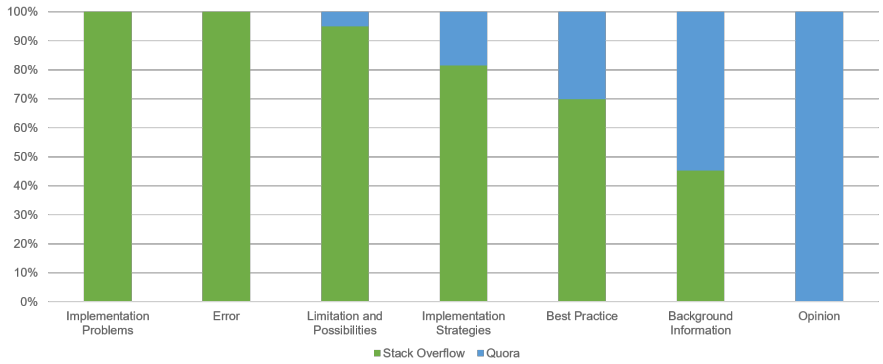


Figure 5.6: Comparison of question type in Quora and Stack Overflow

The second result we obtained from the manual analysis is the taxonomy of developers' *Code Commenting Conventions* questions. Figure 5.7 shows the first two levels of the taxonomy. For simplicity reasons, the third level is omitted in the illustration. A distinctive trait of the emerged taxonomy is that three of the five top-level categories share common subcategories. The subcategories *Syntax & Format*, *Using Feature*, *Asking for Feature*, *Process Comments* and *Change comment template* are present in *Language*, *Tool* and *IDE & Editor*. This shows that developers often have similar questions (subcategories), but within different contexts (top-level categories).

Finding 5.5 Developers often have similar questions but in different contexts. The contexts are categorized by *Language*, *Tool* and *IDE & Editor*.

To find out the most common developer information needs concerning *Code Commenting Conventions*, we break down the total numbers of questions per first-level and second-level categories. Figure 5.8 shows the first-level categories colored by the second-level category as a treemap. We can see that questions regarding *Syntax & Format* are most frequently found in our data. The largest part of those questions is related to the category *Language*. This indicates that when it comes to questions about code comments, developers most often ask about the *Syntax & Format* of a specific programming language. The second most prominent subcategory is *Asking for Feature*, from which most of the questions are in context of the category *Tool*. This shows that developers often do not know about the availability of features for documentation tools that work with code comments (*e.g.*, documentation generation tools).

Finding 5.6 *Syntax & Format* and *Asking for Feature* are the most asked for information types.

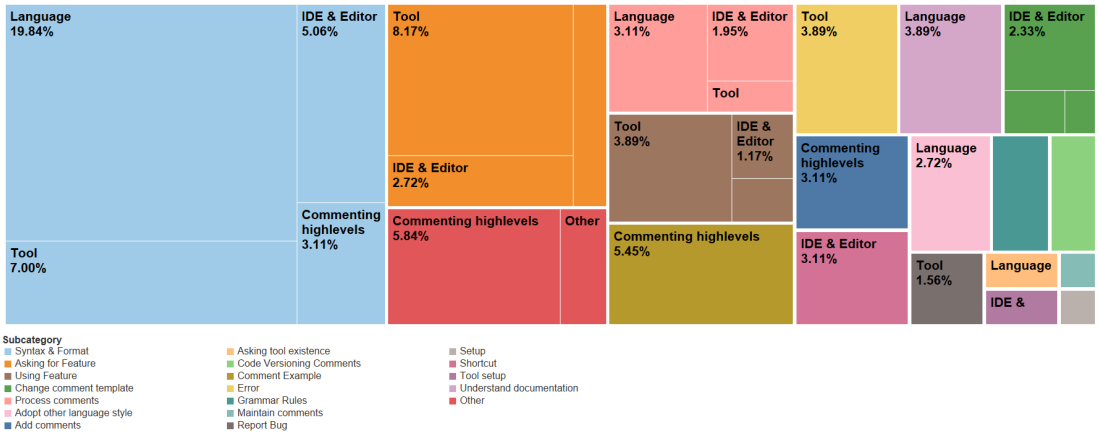


Figure 5.8: Percentage of first-level category (*Label*) questions grouped by second-level category (*Color*) from the *Code Commenting Conventions* taxonomy

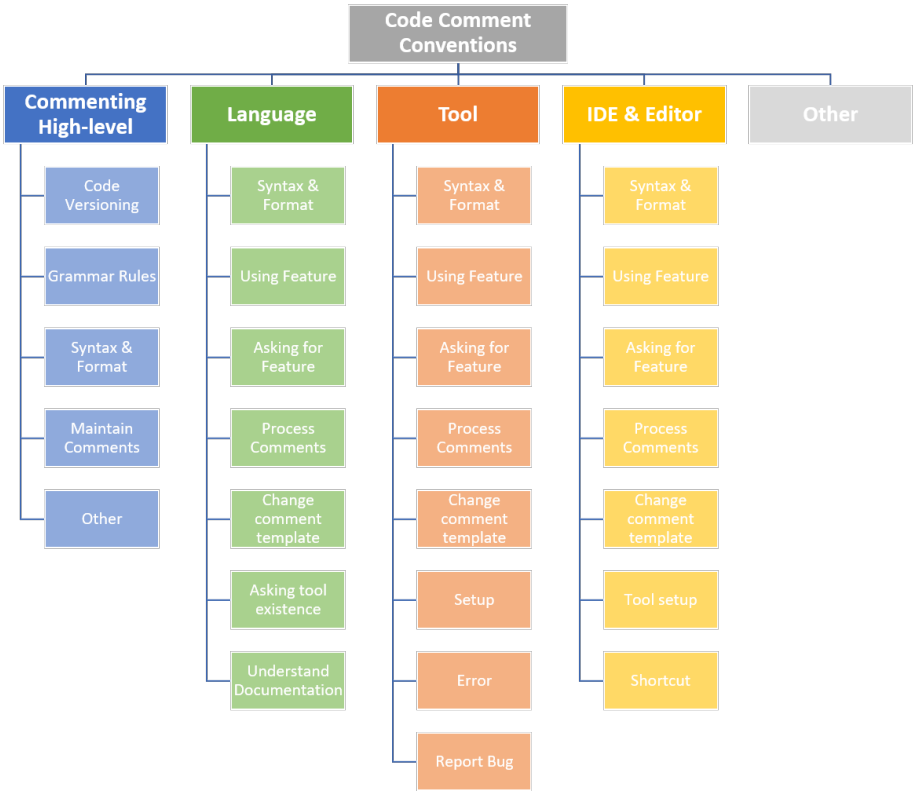


Figure 5.7: Taxonomy of questions about *Code Commenting Conventions* resulting from manual analysis

5.4 Discussion and Implications

This case study provides high-level topics of developers' questions about *Code Commenting Conventions* with the automated analysis, as well as more detailed insights into the topic with the manual analysis. The automated analysis yielded eight relevant topics and successfully divided them from the irrelevant topics present in our dataset. This analysis showed the challenges in working with data about code commenting conventions: the generality and ambiguity of the most important keywords and tags make it difficult to capture a clean dataset. To fully automate the extraction of such datasets is very hard to achieve, as it often needs manual intervention and human interpretation. Also, during manual analysis, the ambiguity and broad usage of the term *comment* pose some difficulties. Often it was hard to decide whether a question is relevant and which categorization should be applied.

The results from the manual analysis show the most important aspects of developer information needs about *Code Commenting Conventions*. Developers are often unsure about *Syntax & Format* of code comments, as well as *Available Feature* in code comment related tools. Code Comments are used and relevant in different contexts: *Language*, *Tools* and *IDE & Editors*. In each of those contexts, developers have different requirements or intended usages of code comments, but due to their nature, the structure of

code comments remains hard to enforce. In this area of conflict, developers seek information, conventions, and guidance on Q&A sites and other external resources. Language designers and tool developers should be more concise in documenting the *Syntax & Format*, *available Features* and *usage instructions* of their product with respect to code comments.

A comparison between the LDA and manual analysis shows similarities. With both methods, we have seen that *Syntax & Format* (*Comments Syntax* in LDA) are the most asked questions. The LDA topics *Documentation generation* and *IDE & Editors* can be mapped to *Tool*, *IDE & Editor* respectively from manual analysis. LDA was able to extract the topics *Syntax & Format* and *Processing code comments*, which we also identified in the manual analysis. However, only during manual analysis, we were able to distinguish the multiple contexts (*Language*, *Tool*, *IDE & Editor*) of those information types.

6

Makar: Data Management Tool

Chapters 3 and 4 have shown that the gathering, preprocessing, and storing data by mining collaborative platforms is often poorly documented or completely left out in the research studies. For transparency, reproducibility, and collaboration, it is essential that the data from empirical studies is handled and preprocessed transparently and precisely documented. This thesis proposes *Makar*, a data management tool, which addresses the described difficulties and requirements for data management and data preprocessing.

The case study presented in Chapter 5 serves as running example in this chapter. To illustrate the features and functionalities we show how *Makar* supported the case study.

6.1 Requirements

The motivation for *Makar* is to support researchers in conducting studies with multiple sources and provide a tool that helps to build reproducible datasets. The literature analysis presented in Chapter 3 and Chapter 4 was used to collect requirements for *Makar*. From the literature survey, we identified that mining studies lack supports in managing the diverse data collected from various sources. Specifically, importing the data from various sources, preprocessing it in a consistent way, querying it to prepare subset of datasets, and exporting it to perform offline analysis and to share it to the research community. The tool intends to cover the common use cases found in the literature, while being extensible to support additional or more specific scenarios gathered from the case study. *Makar* is built primarily for researchers conducting studies with

multiple data sources, but can also be of use to developers managing various datasets.

The functional requirements for *Makar* can be divided into four sections: *data import*, *data management*, *data processing* and *data export*.

Data import requirements are that it should be possible to import data into *Makar* from a diverse set of sources. Standard and multi-purpose sources, like CSV and JSON files, are a core requirement. Additionally, direct import interfaces to data sources like *Stack Overflow* and *Github* should be supported.

Data management requirements include functionality around how to define dataset, how to search it, and how to build collections (*i.e.*, building subsets of the data). The tool should support managing data in various forms, and the user should be able to define what columns and column types the data has. Users can group records into collections. Every record can be part of multiple collections.

Data processing requirements include functionality to transform data into a form to be used for later analysis steps (*e.g.*, cleaning the data) or performing aggregation and filtering steps. Data processing includes common operations like stop word removal, HTML removal, word stemming, string replacement or record-join operations. These operations should apply to the specified collection (*i.e.*, all records of the collection). Each operation or a set of operations when applied to a collection is launched as a task in the tool. Operations applied to a large set of records can lead to long-running tasks. Thus, operations automatically run as background-tasks, which are independent of the tool interface.

Data Querying requirements include functionality to search records in the data. Search functionality should allow the user to search for keywords in specific fields of the data efficiently. The search queries can be saved for later use. Collections can be based on saved search queries, making them always contain all records matching the query.

Data export requirements include functionality to export data in various formats from the tool. Exporting the data allows to use it in various analysis tools, perform offline work or share it with peers. This requirement assures the export functionality for custom-built collections. Users should be able to customize the export functionality by selecting the data fields to be exported.

Non-functional requirements are also defined for the tool. It should be easily extensible in areas where different data projects could have other requirements. Those are namely the import adapters, the data processing steps and the export adapters. These components are designed in an extensible way, which allows future users to adapt the tool to new requirements. The tool should be able to handle large amounts of data (scale of 100k records) and still have acceptable usage performance (*e.g.*, for search queries).

6.2 Features

Based on the collected requirements, the tool supports several features to fulfill the requirements. This section gives a short overview of currently available features in *Makar*.

Importing data from multiple sources is available. Direct data import from *CSV* and *JSON* files allows

importing from arbitrary sources that provide the data in the mentioned file formats. Additionally, *Makar* has direct import adapters for the following sources:

- **Apache Mailing List Archive**[1] All archived Apache Mailing lists can be imported directly into *Makar*. The import adapter allows the mails to be imported by date (send date) or by keyword (subject and body). For example, it is possible to import all mails from the *dev@lucene.apache.org* mailing list sent in 2019 and containing the keywords *error* and *code comment*.
- **Github Pull Requests** (via Github Archive)[2] Pull Requests can be imported into *Makar* directly. The import adapter allows date range restrictions, keyword search and filtering by the number of stars or forks of a repo.
- **Github Issues** Via the Github API[3] Issues can be directly imported into *Makar*. The import adapter accepts a search query that can be parsed by the API.
- **Stack Overflow** Search Excerpts[4] can be imported directly into *Makar*. The import adapter accepts a search query that can be parsed by the API. A more sophisticated import adapter for Stack Overflow is not implemented as Stack Overflow offers the Data Explorer[5] to easily extract large amounts of data and export it to *CSV*.

The import adapters can be extended easily for future projects and other data sources. By extending the `ImportAdapter` class, developers can easily add new custom import adapters.

Case Study Example 6.1. To import the Stack Overflow data we used the *CSV* import adapter, for the Quora data we used the *JSON* import adapter and to import the mailing list data we used the *Apache Mailing List Archive* import adapter.

Managing data in *Makar* is made possible with *Schemas* and *Collections*. A *Schema* is used to group records that belong to the same dataset and have the same attributes and types. All records belong to a *Schema*. *Schemas* can be updated at any time, which allows adding or removing attributes from datasets. *Collections* are more flexible and allow grouping of heterogeneous records. A record can belong to many *Collections*. A *Collection* can be combined with a *Filter* (saved search query), which allows the collection to be updated dynamically and always include all records which can be found with the used filter.

Case Study Example 6.2. To manage diverse datasets, we built *Schemas* for the Stack Overflow, Quora and Mailing list datasets. *Collections* facilitated building multiple datasets, such as the sample set used in the manual analysis (see 5.2.3.2), as well as the sample set of the mailing list data (see 5.2.3.3).

[1] `https://mail-archives.apache.org/mod_mbox/`
[2] `http://www.gharchive.org/`
[3] `https://developer.github.com/v3/search/#search-issues`
[4] `https://api.stackexchange.com/docs/excerpt-search`
[5] `https://data.stackexchange.com/stackoverflow`

Processing data in *Makar* is achieved through *Transformation Steps*. A *Transformation Step* is a single operation that is applied to all records in a collection. Figure 6.1 shows the *strip_html* transformation available in the tool. It is recommended to define *Transformation Steps* to produce new attributes as opposed to overwriting existing attributes with new values. This way, all data from intermediate steps in the preprocessing are preserved and can be inspected at any time. As for import adapters, the *Transformation Steps* are designed to be extended for more use cases. The *Transformation* class offers a simple interface to design custom processing steps.

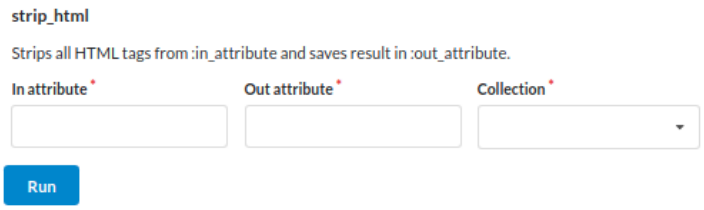


Figure 6.1: Example for a *Transformation Step*: Remove all HTML-Tags

Currently implemented *Transformation Steps* grouped by topic are:

- **Text cleaning:** *string_replace* (replace search term with custom value), *strip_html* (remove all HTML tags), *sanitize_html* (remove all HTML tags, except <a>, <p>, <b>and <i>), *extract_code* (extract HTML <code>tags)
- **Natural language processing:** *word_stemming* (apply Snowball stemming algorithm), *remove_stopwords* (remove all stopwords from Snowball stopword list), *word_list* (remove all punctuation and save words as a list)
- **Data restructuring:** *join* (merge records that have the same value on defined attributes), *extract_value* (create new records from elements in a list), *explode_attribute* (split text on defined substring), *remove_duplicates* (remove duplicated records), *sample_selection* (create a new collection with a randomized sample), *static_value* (add a static value to all records)
- **Arithmetic and counting:** *count_occurrence* (count how often a value occurs in a collection), *arithmetic* (apply the result of an expression to all records in a collection), *word_frequency* (count word occurrence and save the list of word with their frequency in a list)

The data preprocessing workflow for a typical unstructured text analysis project (based on data sources identified in *RQ1*) can be implemented with the above-defined *Transformation steps*. All preprocessing steps done for the case study (*i.e.*, *extract_code*, *strip_html*, *string_replace*, *remove_stopwords*, *word_stemming*) discussed in Chapter 5 have been performed with the currently available *Transformation Steps*.

Case Study Example 6.3. A majority of the preprocessing steps available in the tool have been used in the case study to clean and preprocess the datasets. Figure 5.3 in Section 5.2.2 gives detailed information about the used preprocessing steps.

Exporting data from *Makar* is also available for different formats. Exporting data is achieved through *Collections*. The user can select which attributes (from all available attributes in the *Collection*) are to be selected for the export and then export the data in the required format. Currently, the tool supports *CSV*, *JSON*, and text format. Additionally, the tool supports the facility of adding more complicated export formats via export adapters. A custom export adapter to support LDA analysis with Mallet[6], by providing the data in the required format, is also available.

Case Study Example 6.4. All datasets from the case study have been fully exported as *CSV* to be made available in the replication package. We built and used custom export adapter to export the Stack Overflow dataset in the exact format needed by the LDA tool *Mallet*. This allowed us to efficiently perform LDA analysis with multiple variations of the Stack Overflow dataset and find the best results.

6.3 Architecture

6.3.1 Overview

Makar is based on a relational database and a web application. The database holds the data to be managed in the tool, as well as the data that is required for the functionality in the web application. The web application provides the graphical user interface to access all functionality of the tool. The database and the web application can be easily deployed as docker containers [7], which allows the tool to be used on different operating systems reducing complicated installation procedures. *Makar* is developed as a web application so that it can be hosted on an accessible, and possibly powerful server to allow multiple users to work with the same dataset.

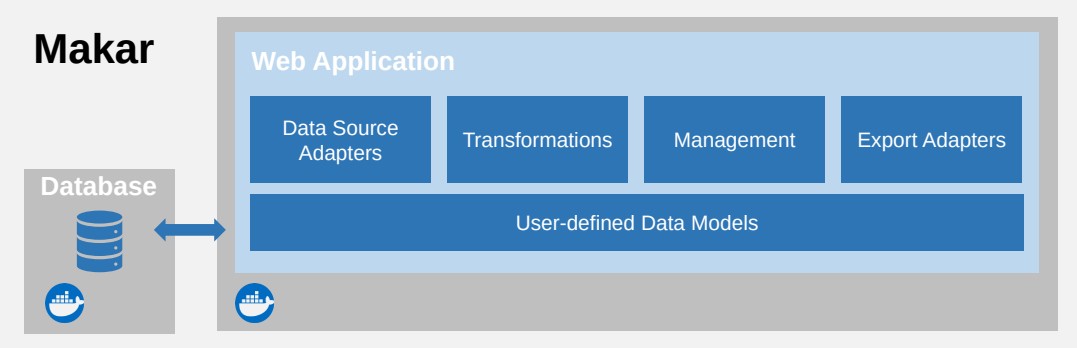


Figure 6.2: Architecture overview of *Makar*

The choice of a relational database for tasks involving large amounts of data can be seen as controversial, but has multiple advantages over a Big Data oriented solution:

[6] http://mallet.cs.umass.edu/
[7] https://www.docker.com

- Sophisticated query possibilities
- Coherent and defined structure of data
- Adapters for web application frameworks
- Ability to define flexible data models
- Mature technology

While Big Data oriented solutions have advantages in:

- Efficiently handling large sizes of data
- Better performance for data transformation
- Support for parallel operations

If *Makar* would have its core functionality only in the field of data processing and transforming, then solutions like Spark [8] or Hadoop [9] could have been better options to integrate. However, because an essential part of *Makar* is *data management*, the functionality and possibilities offered by traditional relational databases outweigh the disadvantages of transformation performance.

6.3.2 Data model

The data model describes the elements of data in *Makar* and how they relate to each other. *Makars'* data model needs to support the user-defined schemas (*i.e.*, the data users want to manage) and the functionality (*e.g.*, *Collections*, *Transformations*, *Jobs*). Therefore, the data model for *Makar* has two scopes:

Dynamic data schemas. It must allow users to manage user-defined data records, meaning that the user can define the attributes and types of the data to be handled with *Makar*.

Makar functionality. It must allow the managed data to be organized and searched efficiently, supporting collections, fine-grained search capabilities, and user-friendly navigation of the data.

6.3.2.1 Data model for user-defined schema

The data handled in *Makar* consists of *records*, which comply with a user-defined *schema*. All *records* have an arbitrary number of *record values*, whose name and type is defined in the *schema*. As shown in Figure 6.3, *records* belong to a *schema*, and *record values* belong to a *record*.

The *schemas* table could be omitted entirely, as it is not necessary to restrict all records in a dataset to a shared *schema*. The actual present attributes and types of a record could easily be gathered through a single database query. However, doing this for every record in a large dataset introduces much additional overhead. The user-defined *schema* is not enforced on the database level; thus the *schemas* table is just a table to group a set of *records* belonging to the same dataset, the schema itself being enforced

[8] `https://spark.apache.org/`
[9] `https://hadoop.apache.org/`

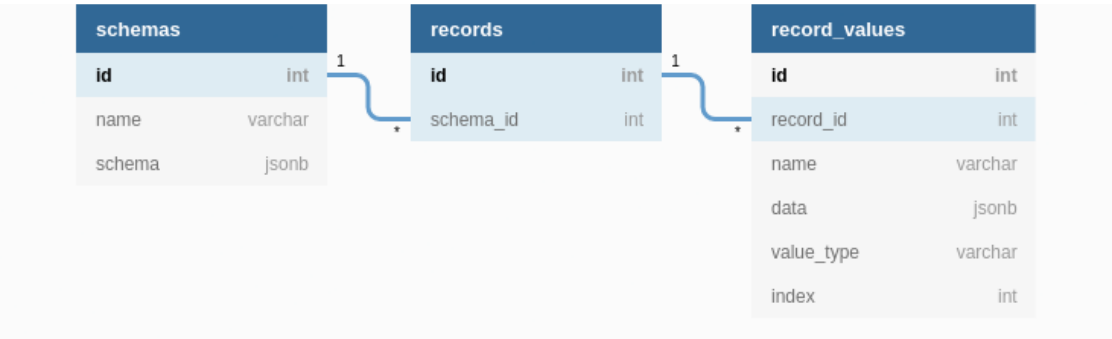


Figure 6.3: Database schema for tables concerned with user-defined data handling

programmatically on every record. Nevertheless, it is convenient for usability and performance reasons to have certain guarantees about the attributes and types of data. This makes the querying, handling and processing of the data less error-prone, and more efficient.

6.3.2.2 Data model for Makar functionality

The additional entities in the data model are required to support the functionality of *Makar* and are designed around the *schemas*, *records* and *record_values* tables explained above.

The database table designed for the functionality concerning data management and data processing are *collections*, *collections_records*, *jobs* and *filters*. The *collections* and *collections_records* tables provide an arbitrary number of *records* to be grouped into collections. The *filters* table holds user-defined and reusable search queries. A *collection* can reference a *filter* that supports the functionality of dynamic collections, meaning that the collection is automatically built based on a user-defined search query (*e.g.*, a *filter*).

The *jobs* table holds all information about performed processing steps.

An overview of all database tables needed for the *Makar* functionality is shown in Figure 6.4.

6.3.3 System design

This subsection gives a short overview of the core components in *Makar* and how they are designed.

6.3.3.1 User-defined Schemas

The ability to define the data attributes and the types of data to be handled by *Makar* is the central functionality. This allows to handle data from various sources and shape the dataset exactly as needed. This functionality is enabled through the data model presented in 6.3.2.1, a JSON-based DSL, and programmatic checks and logic.

The user defines the schema with a DSL, as shown in Listing 1. The shown example is an excerpt of the schema used in the case study for the Stack Overflow dataset. The DSL to define the *schema* is

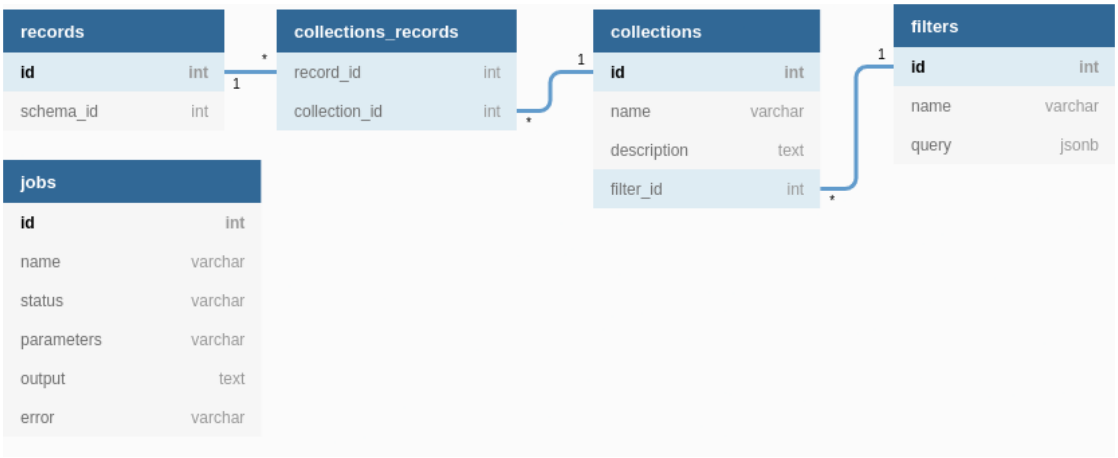


Figure 6.4: Database schema for tables concerned with tool functionality

very simple: On the top-level, it accepts entities *name* and *attributes*. The elements of *attributes* accept *name*, *type* and *multiple*. The following types are currently available: *string*, *int*, *date* and *bool*. With `"multiple": "true"` an attribute can be defined to be a list of elements.

```
{
  "name": "Stack Overflow Posts",
  "attributes": [
    { "name": "id", "type": "int" },
    { "name": "parent_id", "type": "int" },
    { "name": "creation_date", "type": "date" },
    { "name": "view_count", "type": "int" },
    { "name": "body", "type": "string" },
    { "name": "tags", "type": "string", "multiple": "true" }
  ]
}
```

Listing 1: Schema definition for Stack Overflow dataset in *Code Commenting Conventions* case study

The schema definition is persisted in the database in table *schemas* (see Figure 6.3) and is loaded and parsed for every *record*. The parsed schema is cached to increase performance.

The user-defined schema is not a strict guarantee that the records belonging to a specific schema comply with the definition at any time. The schema is only enforced while reading and writing the values of a record. The schema can be changed at any time, for example, to extend the schema with additional attributes.

The record class `app/models/records.rb` is responsible for enforcing the user-defined schema when writing or reading values programmatically. The following guards are applied to enforce the schema

when writing a value: (i) check the attribute name is available on the schema (ii) check the value has the correct type or can be cast and (iii) handle list elements correctly.

Choosing Ruby as the language for *Makar* has some advantages in terms of meta-programming to support user-defined schemas within the application. Considering the schema defined in listing 1, we can define setters and getters for all user-defined attributes via the *method_missing* method in Ruby. Listing 2 demonstrates how the getters and setters are dynamically defined in *Makar*.

```ruby
# Redirect methods that correspond to getter and setter of valid attributes
def method_missing(method_name, *arguments, &block)
  method_name_s = method_name.to_s
  if schema.attributes.include?(method_name_s.gsub('=', ''))
    if method_name_s =~ /(.*)=$/
      store_value(method_name_s.gsub('=', ''), *arguments)
    else
      fetch_value(method_name_s)
    end
  else
    super
  end
end

# Example code
so_record.respond_to?(:title) # => false
so_record.title = "I need help!"
so_record.title # => "I need help!"
```

Listing 2: Dynamically defined setters and getters in *Makar*

6.3.3.2 Data Transformation

The design of the extensible *Transformation Steps* relies mostly on a DSL to easily define inputs to the *Transformation* and background job processing.

Listing 3 shows an exemplary *Transformation Step*. The `INPUTS` constant is a hash with defined input names and types. The view accessed by the users renders automatically all input fields defined by `INPUTS`, and during object creation, the inputs are verified and made accessible as instance variables. The `transform` method is given to a background job worker, which then executes the job.

```ruby
class Transformation::StripHtmlTransformation < Transformation

  INPUTS      = {
    in_attribute: :string,
    out_attribute: :string,
    collection: :collection
  }.freeze

  def init
    @sanitizer = Rails::Html::FullSanitizer.new
  end

  def transform!
    @collection.records.each do |r|
      r.store_value(
         @out_attribute.to_s,
         @sanitizer.sanitize(
           r.fetch_value(@in_attribute.to_s)
        )
      )
    end
  end
end
```

Listing 3: *StripHtmlTransformation* as an example for a Transformation Step

What is to be done inside the `transform` method is entirely up to the developer. This allows for maximal freedom in defining custom *Transformation Steps*.

Case Study Example 6.5. The *StripHtmlTransformation* as shown in Listing 3 was used to remove all HTML tags from the data in the Stack Overflow dataset.

6.3.3.3 Import Adapters

The design of *Import Adapters* is very similar to *Transformation Steps*. The *Import* class provides an interface to *Makar*, which handles inputs and provides helper methods. Only the `inputs` and the `run!` methods need to be implemented to define a custom *Import Adapter*. All inputs get validated and made available as instance variables in the `run!` method. Examples on how *Import Adapters* work are `CsvImport` or `GithubApiImport`, that can be found in the code.

6.3.3.4 Export Adapters

Export Adapters have a similar interface like *ImportAdapters*. Only the `inputs` and `export!` need to be implemented to define a new *ExportAdapter*. The `export!` method needs to return a file, that is delivered to the user. The `MalletExport` is available in the code to serve as an example how to add

more *Export Adapters*.

6.3.3.5 Searching and Filters

The search functionality is implemented with the help of *ransack*[10]. *ransack* is a RoR plugin that enables advanced search capabilities by handling complex queries with nested `AND` and `OR` statements and various predicates (*e.g.*, less than, greater than, equal, matches, starts with, ...). Those search capabilities are made accessible to the users to allow fine-grained and specific exploration of the dataset.

As already stated in Section 6.2, *Filters* are saved search queries. The search query is persisted as a hash in the database and can be reused. *Filters* are an own model within *Makar*, allowing them to be linked to *Collections*. A *Collection* that is connected with a *Filter* contains exactly the records of the dataset that match the *Filters'* search query.

6.3.3.6 Background Jobs

Transformations are executed in background jobs. The background jobs are implemented with the RoR plugin *delayed_job*[11]. The background jobs allow asynchronous execution of potentially long-running *Transformations* in the background. The job to be executed is marshalled to an object and saved in the database. The job handler picks the job to be executed, runs it and stores the result and potential errors in the database. It is possible to deploy multiple workers and allow parallel execution of multiple jobs.

6.3.4 Technology

6.3.4.1 Core Application

Makar is a Ruby on Rails (RoR)[12] web application backed by a Postgresql[13] database. RoR is a mature web application framework. The large number of available plugins (called Gems) and the convention-over-configuration approach of the framework enable developers to build high-quality web applications efficiently. For *Makar*, the primary motivations for using RoR were (i) the meta-programming possibilities of Ruby (see section 6.3.3.1 and listing 2), (ii) the availability of plugins to directly access APIs of popular data sources like Stack Overflow, Github and Mailing lists and (iii) the ease of building feature-rich web applications.

The choice of Postgresql as a database is mainly motivated by the additional functionality Postgresql offers compared to other relational database management systems like MySQL[14] or MariaDB[15]. It is possible to store JSON objects directly in the database, which is a significant advantage for *Makar* to store arbitrary data and data types within the same database column. Additionally, Postgresql is fully supported by RoR.

[10] `https://github.com/activerecord-hackery/ransack`
[11] `https://github.com/collectiveidea/delayed_job`
[12] `https://rubyonrails.org/`
[13] `https://www.postgresql.org/`
[14] `https://www.mysql.com/`
[15] `https://mariadb.org/`

The use of a BigData processing engine is already shortly discussed in section 6.3.1. Spark[16] (including Hadoop) for speeding up data processing and running *Transformation Steps* has been tested during the development of *Makar*. While processing speed for pre-defined *Transformations* and large enough datasets was better with Spark, the handling of user-defined schemas is very hard to achieve. The combination of data management functionalities like collections, user-defined schemas, and search capabilities, together with the technology difference between RoR and Spark, turned out to be impractical and not promising for *Makar*.

6.3.4.2 Containerization

Researchers use various environments to build study pipelines. Using a different environment or operating system can make it hard to use the tools in the study pipeline. To ensure maximal compatibility and ease of installation on different operating systems, *Makar* is built to be run in a Docker[17] container. The database and job worker processes also run in their own containers. All containers are combined and orchestrated with Docker Compose[18], making the installation and usage of the application as easy as possible. More information and instructions can be found in the `README` of the project.

6.3.4.3 Technical Requirements

Since the tool is dockerized, the technical requirements to run the tool are minimal. The following is needed:

- Docker (>= 19.0)
- Docker Compose (version 3)

6.4 Benefits and Future Improvements

Makar has its intended use as a versatile tool for small- to mid-scale projects which need to extract, manage and preprocess data from diverse sources. The tool target researchers who perform empirical studies on online sources frequently with similar workflow pipelines. The tool can help them in repeating the study workflow efficiently, producing a reproducible dataset, and reporting a transparent process that can eventually help other researchers to extend the study. Additionally, it can assist developers to collect, aggregate and manage information to support their work.

The tool focuses on user-defined schemas and extensibility to support a wide range of projects with different use-cases. The orientation of *Makar* brings along some limitations and use-cases for which *Makar* is not well suited. This section addresses benefits of the tool, but also known limitations and how they could mitigated in future work.

[16] `https://spark.apache.org/`
[17] `https://www.docker.com/`
[18] `https://docs.docker.com/compose/`

6.4.1 Benefits

6.4.1.1 Diverse Use Cases

With its user-defined schemas, the various transformation steps, and the import adapters, *Makar* is suited for diverse use cases. In contrast to previous work [90], it can cope with a diverse set of data types, not only software repository data. Also, its use-case is not limited to find emerging topics (*e.g.*, top keywords on Twitter) from social media sites [69], nor a special DSL needs to be learned [29]. Additionally, users are not restricted to a single dataset. Managing multiple datasets from various sources and with different data types is a default functionality in *Makar*. This allows researchers to combine, compare and aggregate a diverse set of sources in order to extract insights that cannot be found in single-source studies. With its focus on extensibility, *Makar* can be tailored efficiently for specific projects.

Case Study Example 6.6. The case study included three datasets, which were managed within *Makar*. During the case study, we built another dataset to extract relevant tags from Stack Overflow using the approach mentioned in Section 5.2.1.1. This extraction process required numerous arithmetic operations and computations on the dataset. This is a great example to show how a specific use case can be customized and supported in *Makar*.

6.4.1.2 Exploratory Studies

Existing tools focus on highly efficient processing and analysis workflows (*e.g.*, Knime[19], Rapidminer[20]) to analyze large datasets. A list of compared tools is shown in Table A.2. We tested and compared the tools concerning available functionality of data mining, preprocessing, data management, natural language processing (NLP), machine learning (ML) and visualization. Most of the tools incorporate the ability to ease the process of building machine learning or NLP projects. However, what they lack is the ability to manually explore the data, investigate small samples, or do ad-hoc searches. Researchers using new sources of data or combining multiple sources cannot be sure about the quality and characteristics of the data, right from the beginning. Human interpretation, ad-hoc testing of simple hypotheses or assessment of data quality is often required to determine the plausible approach for the study. *Makar* is well suited to cover such aspects of researchers' work. The data imported in the tool can be accessed and inspected at any time during the process. Advanced search features facilitate a user to filter and search the dataset according to various data attributes. Dynamic collections make it simple to group or separate parts of the dataset. These features make the tool convenient for users to explore the data and to interact with the data manually. We have seen in chapter 5, that not all steps of such a study can be automated, making this aspect of *Makar* valuable for its users.

[19] `https://www.knime.com`
[20] `https://rapidminer.com/`

Case Study Example 6.7. *Makar* allowed us to quickly explore how often well-known documentation and code commenting related tools were discussed in the datasets. This information could lead to additional results being reported in the case study. Being able to quickly check certain hypotheses or collect additional information can support researcher in formulating results and gain deeper insights.

6.4.2 Future Improvements

6.4.2.1 Data Analysis and Visualizations

Makar does not support any visualization or analysis features. The focus is intentionally on data management features such as *import*, *managing*, *processing*, and *export* of data as this process is often very similar in various empirical studies. Analysis and visualization are more specific to a particular project and its goals, and thus it makes more sense to rely on specific tools for those tasks. The effort to build multi-purpose visualization or analysis functionality (*e.g.*, graphs, topic modeling, classifiers) would be high. However, the possibility to match requirements for different projects and use-cases would be rather low, compared to the often similar approaches in preprocessing the data. We have seen in Chapter 4 that analysis methodologies are more diverse and also more tools are used to conduct this part of the studies.

To assure the usefulness of *Makar* in analysis procedures (generally performed after preprocessing steps) in various research studies, the tool offers extensible *Export Adapters* (see section 6.2), to export the preprocessed data in required format for further analysis. As described in chapter 5, the case study required different and specific preprocessing steps for LDA topic modeling using Mallet and manual analysis. With the help of *Export Adapter*, the tool produces the data in the same format as required by Mallet.

However, it is possible to extend *Makar* with visualization or analysis components. Determining which components could be relevant for a wide range of use-cases in this area would be an interesting direction. Then implement the required features according to the collected requirements can be an interesting task for future work.

6.4.2.2 Performance

As already discussed in sections 6.3.4.1 and 6.3.1, the tool is not specialized for high-performance processing of ultra-large datasets. Depending on the transformation and size of the dataset, some processing steps will need a longer period of time to finish. To mitigate the problem of long-running transformations, *Makar* runs the transformation in background jobs, which can be queued or run in parallel if required. It is possible to offload performance-intensive processing steps to specialized environments. How this can be achieved could be addressed in future work.

6.4.2.3 Data Volume

Although Postgresql is better suited to handle massive data volumes compared to other relational database management systems, *Makar* is not built to handle datasets that scale up to millions of records. Theoret-

ically, it is possible to hold an arbitrary amount of records in *Makar*, but the performance of searching, running transformations and accessing records would degrade to a non-acceptable level.

A BigData project cannot profit from the functionalities of *Makar*, and we advise to build a specific processing and analysis pipeline with better-suited tools and environments. With improvements to performance-intensive processing steps, as mentioned in 6.4.2.2, this limitation can be partially mitigated. Nevertheless, the projects assessed in chapter 4 have datasets that can be handled by *Makar*.

7

Threats to Validity

Literature Selection. The most significant threat to validity is the selection of literature used in *RQ1* and *RQ2*. We used only *Google Scholar* to extend the systematic literature survey from previous work [71] with more studies. The search of literature was performed by two people. The selected studies may not represent all work around analyzing developer information needs using external sources. However, our focus was on finding popular and frequently used sources of information. More papers included in the literature selection could have yielded more identified categories in *RQ1* but would not change the main findings in *RQ1* and *RQ2*. Our goal was the identify popular sources of information used by researchers, describe challenges and opportunities, uncover common methodologies, and address issue found in the studies. We believe we achieved these goals with the presented selection of literature.

Case Study. Threats to construct validity in the case study are potential imprecision in the measurements. We filtered Stack Overflow tags to mitigate bias in the selection of developers' questions about *Code Commenting Conventions*. It is possible that the tag-based approach missed relevant questions because the tags did not properly reflect the questions' content. For selection of posts in Quora, we mapped Stack Overflow tags to Quora topics. This processed is potentially biased and it is possible that we missed topics containing relevant Quora posts. To mitigate bias in the manual analysis, we performed also LDA, an automated topic modeling analysis. The combination of manual and automated analysis was performed to diminish possible bias in our investigation.

Threats to internal validity can be identified in our process to build the taxonomy. To mitigate this threat,

all questions were validated by at least two reviewers. In case of disagreement between the two reviewers, a third reviewer was involved to reach agreement. Multiple iterations, involving three different authors, were applied to reach the resulting taxonomy.

Threats to external validity concern the choice of Stack Overflow and Quora as data source. Although both sources are popular among developers and often used for development related discussions, other sources of information like *Github*, *Reddit* or *Jira* could be considered for future work.

Prototype Tool. We developed *Makar* with the essential features needed from a reproducibility perspective, identified in Chapter 4. The tools' focus is on including those important features and provide good usability to its' users rather than performance in preprocessing large datasets. Nevertheless, we believe that *Makar* can support diverse projects and be of value to researchers and developers. Improving on performance aspects is possible and left for future work.

The tool has only been evaluated in the context of the case study presented in this thesis. An evaluation of it usability and performance should be considered for future work. The performed comparison against similar tools is possibly biased as the selection of tools was done by one person. We focused on comparing *Makar* to other tools that serve similar use-cases and can handle data from external sources. Since we did not find any comparable tool in the analyzed literature, we believe that *Makar* is a novel contribution.

8

Conclusion and Future Work

We have identified and characterized five categories (*Q&A Site*, *Mailing list*, *Bug Report*, *User Review* and *Newsgroup*) of data sources for researchers conducting studies about developer information needs. We have shown that Q&A sites, respectively Stack Overflow, currently enable many studies. Ease of extraction is a significant factor when choosing sources, and thus Stack Overflow is hugely popular among researchers. The extraction method determines the amount of data that can be extracted and limits the possible analysis methods available to researchers. Therefore, providing access to more data sources and sharing extracted datasets is crucial to enable advanced research in developer information needs. We have analyzed relevant literature on various aspects of the methodology, focusing on assessing the reproducibility of the studies. We considered data extraction, preprocessing, and dataset availability as important aspects of reproducibility and have shown that many of the analyzed studies do not report those aspects in adequate detail. This makes it hard for other researchers to reproduce such studies, rebuild the dataset, or apply similar methodologies to the same dataset.

To deepen our insight into the various challenges identified in the relevant literature and to extend knowledge about developer information needs, we conducted a case study about *Code Commenting Conventions*. We used data from multiple sources to perform an automated and manual analysis. We uncovered high-level topics of developers' questions about *Code Commenting Conventions*, proposed a taxonomy of such questions, and did shed some light on issues developers face with *Code Commenting Conventions*. Performing the case study, we faced challenges regarding the generality and ambiguity of the

involved keywords *comment* and *convention*.

Despite LDA being an automated analysis, we faced various challenges to achieve meaningful results. The case study has shown that it is hard to achieve full automation as manual intervention and human interpretation is needed. For example, the tag selection had to be precisely calibrated, and the interpretation of the topics, yielded by LDA, needed manual analysis of the results. However, we were able to automate various intermediate steps, especially in preprocessing the data.

We developed a prototype tool called *Makar*, which was used for extraction, preprocessing and management of data throughout the case study. The tool was developed to address issues of reproducibility identified in the relevant literature. It allows users to extract, preprocess, and manage data from multiple sources in a documented and reproducible way. The prototype tool is easy to use, can be extended with custom import adapters, preprocessing steps, and export adapters. It can be used for a wide range of studies, being of help to researchers as well as developers.

Future work is possible in three areas:

Developer information needs The research on developer information needs can be continued with a focus on multi-source studies. Much of the previous work considers only a single source of data. This can be improved by combining data from multiple sources and generating more profound and more connected insights. An analysis of methodology, common practices and challenges of existing multi-source studies could prove valuable to researchers. Further research in this direction could lead to better solutions for supporting and standardizing the research process, ultimately improving the reproducibility of the studies.

Code Commenting Conventions The study on *Code Commenting Conventions* can be extended by considering more data sources. Many software projects have dedicated *Contribution Guidelines*, addressing also code comments. One could extract such guidelines and analyze how they are discussed in issues and implemented in the source code.

Prototype Tool *Makar* has multiple options for future work. It can be extended with more data source adapters, allowing for easy data extraction from a wide range of developer information needs data sources. Also, extending the tool with components for data analysis and visualization is possible. Although we did not find similar tools used in the analyzed relevant literature, there are other tools similar to that of *Makar* (see Table A.2). Future work should identify similar tools and evaluate *Makar* against those tools.

A

Appendices

A.1 List of papers for RQ1 and RQ2

Title	Year	Reference
Information needs in bug reports: improving cooperation between developers and users	2010	[21]
An empirically-based characterization and quantification of information seeking through mailing lists during Open Source developers' software evolution	2015	[78]
Developing Schema for Open Source Programmers' Information-Seeking	2008	[77]
Open Source Programmers' Information Seeking During Software Maintenance	2011	[79]
How the R Community Creates and Curates Knowledge: A Comparative Study of Stack Overflow and Mailing Lists	2016	[99]
What Can Programmer Questions Tell Us About Frameworks?	2005	[41]
Empirical Analysis of the Logging Questions on the StackOverflow Website	2018	[35]
What are mobile developers asking about? A large scale study using stack overflow	2016	[74]
What are developers talking about? An analysis of topics and trends in Stack Overflow	2014	[11]

What Do Concurrency Developers Ask About? A Large-scale Study Using Stack Overflow	2018	[4]
How do programmers ask and answer questions on the web?: Nier track	2011	[85]
A Manual Categorization of Android App Development Issues on Stack Overflow	2014	[18]
What Concerns Do Client Developers Have When Using Web APIs? An Empirical Study of Developer Forums and Stack Overflow	2016	[87]
Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis	2017	[102]
What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts	2016	[98]
Mining Questions about Software Energy Consumption	2014	[67]
Mining Questions Asked by Web Developers	2014	[9]
An Exploratory Analysis of Mobile Development Issues using Stack Overflow	2013	[52]
A Study on the Most Popular Questions About Concurrent Programming	2015	[68]
What Questions Do Programmers Ask About Configuration as Code?	2018	[70]
An empirical study on developer interactions in StackOverflow	2013	[92]
What are Software Engineers asking about Android Testing on Stack Overflow?	2017	[88]
Mining Testing Questions on Stack Overflow	2016	[47]
Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions	2011	[40]
Development Emails Content Analyzer: Intention Mining in Developer Discussions	2015	[81]
Release Planning of Mobile Apps Based on User Reviews	2016	[89]
Crowdsourcing user reviews to support the evolution of mobile apps	2018	[63]
Why people hate your app: making sense of user feedback in a mobile app store	2013	[31]
Causal Impact Analysis Applied to App Releases in Google Play and Windows Phone Store	2015	[55]
The app sampling problem for app store mining	2015	[54]
AR-miner: mining informative reviews for developers from mobile app marketplace	2014	[23]
Mining User Opinions in Mobile App Reviews:A Keyword-based Approach	2015	[91]
CODES: Mining source code descriptions from developer communications	2012	[65]
User feedback in the appstore: An empirical study	2013	[62]

Mining Query Subtopics from Questions in Community Question Answering	2015	[96]
Attentive Interactive Convolutional Matching for Community Question Answering in Social Multimedia	2018	[43]
StackOverflow and GitHub: Associations Between Software Development and Crowd-sourced Knowledge	2013	[86]
Using and Asking: APIs Used in the Android Market and Asked About in Stack Overflow	2013	[44]
Which Non-functional Requirements do Developers Focus on?	2015	[101]
Detecting API Usage Obstacles: A Study of iOS and Android Developer Questions	2013	[94]
Classifying Stack Overflow Posts on API Issues	2018	[2]

Table A.1: Literature selection for RQ1 and RQ2

A.2 Tool comparison

Tool	URL	Costs	Data Mining	Preprocessing	Data Mgmt.	NLP	ML	Visualization
Octoparse	[a]	Free (Limited)	Yes	No	No	No	No	No
Knime	[b]	Free	Extension	Yes	No	Yes	Yes	Yes
Pentaho Data Integration	[c]	Paid	No	Yes	?	No	Yes	?
Apache Mahout	[d]	Free	No	No	No	No	Yes	No
Birt	[e]	Free	No	No	No	No	No	Yes
Rapidminer	[f]	Commercial, Educational License	No	Yes	No	No	Yes	Yes
ELKI	[g]	Free	No	No	No	No	Yes	Yes
Keel	[h]	Free	No	No	No	No	Yes	Yes
WEKA	[i]	Free	No	No	No	No	Yes	Yes

Table A.2: Various tools with partially similar functionality as *Makar*

[a] https://www.octoparse.com/
[b] https://www.knime.com
[c] https://www.hitachivantara.com/en-us/products/big-data-integration-analytics/pentaho-data-integration.html
[d] https://mahout.apache.org
[e] https://www.eclipse.org/birt/
[f] https://my.rapidminer.com
[g] https://elki-project.github.io/
[h] http://www.keel.es/
[i] https://www.cs.waikato.ac.nz/ml/weka/index.html

Bibliography

[1] E. Aghajani, C. Nagy, O. L. Vega-Márquez, M. Linares-Vásquez, L. Moreno, G. Bavota, and M. Lanza. Software documentation issues unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1199–1210. IEEE, 2019.

[2] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider. Classifying Stack Overflow posts on API issues. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 244–254, March 2018.

[3] A. Ahmad, C. Feng, S. Ge, and A. Yousif. A survey on mining Stack Overflow: question and answering (Q&A) community. *Data Technologies and Applications*, 52(2):190–247, 2018.

[4] S. Ahmed and M. Bagherzadeh. What do concurrency developers ask about?: A large-scale study using Stack Overflow. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '18, pages 30:1–30:10, New York, NY, USA, 2018. ACM.

[5] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton. Learning natural coding conventions. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 281–293, New York, NY, USA, 2014. ACM.

[6] M. Allamanis and C. Sutton. Why, when, and what: Analyzing Stack Overflow questions by topic, type, and code. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 53–56, Piscataway, NJ, USA, 2013. IEEE Press.

[7] Y. AlNoamany and J. A. Borghi. Towards computational reproducibility: researcher perspectives on the use and sharing of software. *PeerJ Computer Science*, 4:e163, Sept. 2018.

[8] S. Amann, S. Beyer, K. Kevic, and H. Gall. *Software Mining Studies: Goals, Approaches, Artifacts, and Replicability*, pages 121–158. Springer International Publishing, Cham, 2015.

[9] K. Bajaj, K. Pattabiraman, and A. Mesbah. Mining questions asked by web developers. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 112–121. ACM, 2014.

[10] L. Bao, Z. Xing, X. Wang, and B. Zhou. Tracking and analyzing cross-cutting activities in developers' daily work (n). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 277–282. IEEE, 2015.

[11] A. Barua, S. W. Thomas, and A. E. Hassan. What are developers talking about? an analysis of topics and trends in Stack Overflow. *Empirical Software Engineering*, 19(3):619–654, 2014.

[12] G. Bavota. Mining unstructured data in software repositories: Current and future trends. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 5, pages 1–12. IEEE, 2016.

[13] K. Beck. Manifesto for agile software development, 2001.

[14] A. Begel and N. Nagappan. Global software development: Who does it? In *2008 IEEE International Conference on Global Software Engineering*, pages 195–199, 2008.

[15] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, page 308–318, New York, NY, USA, 2008. Association for Computing Machinery.

[16] N. Bettenburg, E. Shihab, and A. E. Hassan. An empirical study on the risks of using off-the-shelf techniques for processing mailing list data. In *2009 IEEE International Conference on Software Maintenance*, pages 539–542. IEEE, 2009.

[17] N. Bettenburg, E. Shihab, and A. E. Hassan. An empirical study on the risks of using off-the-shelf techniques for processing mailing list data. In *2009 IEEE International Conference on Software Maintenance*, pages 539–542, 2009.

[18] S. Beyer and M. Pinzger. A manual categorization of Android app development issues on Stack Overflow. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution*, ICSME '14, pages 531–535, Washington, DC, USA, 2014. IEEE Computer Society.

[19] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[20] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer. Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, page 1589–1598, New York, NY, USA, 2009. Association for Computing Machinery.

[21] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Information needs in bug reports: improving cooperation between developers and users. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 301–310. ACM, 2010.

[22] G. Canfora, L. Cerulo, M. Cimitile, and M. Di Penta. Social interactions around cross-system bug fixings: The case of FreeBSD and OpenBSD. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, page 143–152, New York, NY, USA, 2011. Association for Computing Machinery.

[23] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang. Ar-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th international conference on software engineering*, pages 767–778, 2014.

[24] T.-H. Chen, S. W. Thomas, and A. E. Hassan. A survey on the use of topic models when mining software repositories. *Empirical Software Engineering*, 21(5):1843–1919, 2016.

[25] M. A. de F. Farias, R. Novais, M. C. Júnior, L. P. da Silva Carvalho, M. Mendonça, and R. O. Spínola. A systematic mapping study on mining software repositories. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, pages 1472–1479, New York, NY, USA, 2016. ACM.

[26] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*, SIGDOC '05, pages 68–75, New York, NY, USA, 2005. ACM.

[27] S. Dwivedi, P. Kasliwal, and S. Soni. Comprehensive study of data analytics tools (RapidMiner, Weka, R tool, Knime). In *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, pages 1–8. IEEE, 2016.

[28] T. Dybå and T. Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9):833 – 859, 2008.

[29] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 422–431, Piscataway, NJ, USA, 2013. IEEE Press.

[30] Edmund, Wong. Mining question and answer sites for automatic comment generation, 2014.

[31] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 1276–1284, New York, NY, USA, 2013. Association for Computing Machinery.

[32] G. Ghezzi and H. C. Gall. Replicating mining studies with SOFAS. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 363–372. IEEE, 2013.

[33] M. W. Godfrey, A. E. Hassan, J. Herbsleb, G. C. Murphy, M. Robillard, P. Devanbu, A. Mockus, D. E. Perry, and D. Notkin. Future of mining software archives: A roundtable. *IEEE Software*, 26(1):67–70, 2008.

[34] J. M. González-Barahona and G. Robles. On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. *Empirical Software Engineering*, 17(1):75–89, 2012.

[35] H. Gujral, A. Sharma, S. Lal, A. Kaur, A. Kumar, and A. Sureka. Empirical analysis of the logging questions on the Stack Overflow website. In *2018 Conference On Software Engineering & Data Sciences (CoSEDS)(in-press)*, 2018.

[36] S. Gupta and S. Gupta. Natural language processing in mining unstructured data from software repositories: a review. *Sādhanā*, 44(12):244, 2019.

[37] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. v. Deursen. Communication in open source software development mailing lists. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 277–286. IEEE Press, 2013.

[38] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, 45(1):11:1–11:61, Dec. 2012.

[39] J. Highsmith and A. Cockburn. Agile software development: the business of innovation. *Computer*, 34(9):120–127, 2001.

[40] D. Hou and L. Li. Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions. In *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, ICPC '11, pages 91–100, Washington, DC, USA, 2011. IEEE Computer Society.

[41] D. Hou, K. Wong, and H. J. Hoover. What can programmer questions tell us about frameworks? In *Proceedings of the 13th International Workshop on Program Comprehension*, IWPC '05, pages 87–96, Washington, DC, USA, 2005. IEEE Computer Society.

[42] G. Hu, J. Shao, F. Shen, Z. Huang, and H. T. Shen. Unifying multi-source social media data for personalized travel route planning. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 893–896, New York, NY, USA, 2017. ACM.

[43] J. Hu, S. Qian, Q. Fang, and C. Xu. Attentive interactive convolutional matching for community question answering in social multimedia. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, page 456–464, New York, NY, USA, 2018. Association for Computing Machinery.

[44] D. Kavaler, D. Posnett, C. Gibler, H. Chen, P. Devanbu, and V. Filkov. Using and Asking: APIs Used in the Android Market and Asked about in Stack Overflow. In A. Jatowt, E.-P. Lim, Y. Ding, A. Miura, T. Tezuka, G. Dias, K. Tanaka, A. Flanagin, and B. T. Dai, editors, *Social Informatics*, pages 405–418, Cham, 2013. Springer International Publishing.

[45] B. W. Kernighan and R. Pike. *The Practice of Programming (Addison-Wesley Professional Computing Series)*. Addison-Wesley, 1 edition, Feb. 1999.

[46] A. Ko, B. Myers, M. Coblenz, and H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *Software Engineering, IEEE Transactions on*, 32(12):971 –987, Dec. 2006.

[47] P. S. Kochhar. Mining testing questions on Stack Overflow. In *Proceedings of the 5th International Workshop on Software Mining*, SoftwareMining 2016, pages 32–38, New York, NY, USA, 2016. ACM.

[48] M. Krafczyk, A. Shi, A. Bhaskar, D. Marinov, and V. Stodden. Scientific tests and continuous integration strategies to enhance reproducibility in the scientific software context. In *Proceedings of the 2nd International Workshop on Practical Reproducible Evaluation of Computer Systems*, P-RECS '19, page 23–28, New York, NY, USA, 2019. Association for Computing Machinery.

[49] V. Krotov and L. Silva. Legality and ethics of web scraping. 2018.

[50] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 492–501, New York, NY, USA, 2006. ACM.

[51] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles. A survey of devops concepts and challenges. *ACM Comput. Surv.*, 52(6), Nov. 2019.

[52] M. Linares-Vásquez, B. Dit, and D. Poshyvanyk. An exploratory analysis of mobile development issues using Stack Overflow. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 93–96, Piscataway, NJ, USA, 2013. IEEE Press.

[53] Z. Liu, H. Chen, X. Chen, X. Luo, and F. Zhou. Automatic detection of outdated comments during code changes. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 01, pages 154–163, 2018.

[54] W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang. The app sampling problem for app store mining. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 123–133, 2015.

[55] W. Martin, F. Sarro, and M. Harman. Causal impact analysis applied to app releases in Google Play and Windows Phone store. *RN*, 15(07), 2015.

[56] A. K. McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[57] T. Menzies and T. Zimmermann. Software analytics: so what? *IEEE Software*, 30(4):31–37, 2013.

[58] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz. The work life of developers: Activities, switches and perceived productivity. *IEEE Transactions on Software Engineering*, 43(12):1178–1193, 2017.

[59] R. Minelli, A. Mocci, and M. Lanza. I know what you did last summer: An investigation of how developers spend their time. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*, ICPC '15, pages 25–35, Piscataway, NJ, USA, 2015. IEEE Press.

[60] M. Nagappan and E. Shihab. Future trends in software engineering research for mobile apps. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 5, pages 21–32, 2016.

[61] N. Nazar, Y. Hu, and H. Jiang. Summarizing software artifacts: A literature review. *Journal of Computer Science and Technology*, 31(5):883–909, 2016.

[62] D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, pages 125–134, July 2013.

[63] F. Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia. Crowdsourcing user reviews to support the evolution of mobile apps. *Journal of Systems and Software*, 137:143 – 162, 2018.

[64] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 522–531. IEEE Press, 2013.

[65] S. Panichella, J. Aponte, M. D. Penta, A. Marcus, and G. Canfora. Mining source code descriptions from developer communications. In D. Beyer, A. van Deursen, and M. W. Godfrey, editors, *IEEE 20th International Conference on Program Comprehension, ICPC 2012, Passau, Germany, June 11-13, 2012*, pages 63–72. IEEE Computer Society, 2012.

[66] S. Panichella, G. Bavota, M. Di Penta, G. Canfora, and G. Antoniol. How developers' collaborations identified from different sources tell us about code changes. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 251–260. IEEE, 2014.

[67] G. Pinto, F. Castor, and Y. D. Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 22–31, New York, NY, USA, 2014. ACM.

[68] G. Pinto, W. Torres, and F. Castor. A study on the most popular questions about concurrent programming. In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU 2015, pages 39–46, New York, NY, USA, 2015. ACM.

[69] R. Pokharel, P. D. Haghighi, P. P. Jayaraman, and D. Georgakopoulos. Analysing emerging topics across multiple social media platforms. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW 2019, pages 16:1–16:9, New York, NY, USA, 2019. ACM.

[70] A. Rahman, A. Partho, P. Morrison, and L. Williams. What questions do programmers ask about configuration as code? In *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*, RCoSE '18, pages 16–22, New York, NY, USA, 2018. ACM.

[71] J. Richner. Software developers' information needs, Feb. 2019.

[72] M. P. Robillard, W. Coelho, and G. C. Murphy. How effective developers investigate source code: An exploratory study. *IEEE Trans. Softw. Eng.*, 30(12):889–903, Dec. 2004.

[73] G. Rodríguez-Pérez, G. Robles, and J. M. González-Barahona. Reproducibility and credibility in empirical software engineering: A case study based on a systematic literature review of the use of the szz algorithm. *Information and Software Technology*, 99:164 – 176, 2018.

[74] C. Rosen and E. Shihab. What are mobile developers asking about? A large scale study using stack overflow. *Empirical Software Engineering*, 21(3):1192–1223, 2016.

[75] N. B. Ruparelia. Software development lifecycle models. *SIGSOFT Softw. Eng. Notes*, 35(3):8–13, May 2010.

[76] W. Shang, B. Adams, and A. E. Hassan. An experience report on scaling tools for mining software repositories using mapreduce. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, page 275–284, New York, NY, USA, 2010. Association for Computing Machinery.

[77] K. Y. Sharif and J. Buckley. Developing schema for open source programmers' information-seeking. In *Information Technology, 2008. ITSim 2008. International Symposium on*, volume 1, pages 1–9. IEEE, 2008.

[78] K. Y. Sharif, M. English, N. Ali, C. Exton, J. Collins, and J. Buckley. An empirically-based characterization and quantification of information seeking through mailing lists during open source developers' software evolution. *Information and Software Technology*, 57:77–94, 2015.

[79] K. Y. Sharif, M. R. Mokhtar, and J. Buckley. Open source programmers' information seeking during software maintenance. *Journal of Computer Science*, 7(7):1060–1071, 2011.

[80] J. Snell and N. Menaldo. Web scraping in an era of big data 2.0. *Bloomberg Law News*, 2016.

[81] A. D. Sorbo, S. Panichella, C. A. Visaggio, M. D. Penta, G. Canfora, and H. C. Gall. Development emails content analyzer: Intention mining in developer discussions (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 12–23, 2015.

[82] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng. The impact of social media on software engineering practices and tools. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, page 359–364, New York, NY, USA, 2010. Association for Computing Machinery.

[83] T. Tenny. Procedures And Comments vs. The Banker's Algorithm. *ACM SIGCSE Bulletin*, 17(3):44–53, 1985.

[84] I.-H. Ting, H.-J. Wu, and P.-S. Chang. Analyzing multi-source social data for extracting and mining social networks. In *2009 International Conference on Computational Science and Engineering*, volume 4, pages 815–820. IEEE, 2009.

[85] C. Treude, O. Barzilay, and M.-A. Storey. How do programmers ask and answer questions on the web? (NIER Track). In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 804–807, New York, NY, USA, 2011. ACM.

[86] B. Vasilescu, V. Filkov, and A. Serebrenik. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *2013 International Conference on Social Computing*, pages 188–195. IEEE, 2013.

[87] P. K. Venkatesh, S. Wang, F. Zhang, Y. Zou, and A. Hassan. What concerns do client developers have when using web APIs? an empirical study of developer forums and stack overflow. In *IEEE international conference on web services (ICWS)*, pages 131–138, 2016.

[88] I. K. Villanes, S. M. Ascate, J. Gomes, and A. C. Dias-Neto. What are software engineers asking about Android testing on Stack Overflow? In *Proceedings of the 31st Brazilian Symposium on Software Engineering*, SBES'17, pages 104–113, New York, NY, USA, 2017. ACM.

[89] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Release planning of mobile apps based on user reviews. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 14–24, 2016.

[90] L. Voinea and A. Telea. Visual querying and analysis of large software repositories. *Empirical Software Engineering*, 14(3):316–340, 2009.

[91] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen. Mining user opinions in mobile app reviews: A keyword-based approach (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 749–759. IEEE, 2015.

[92] S. Wang, D. Lo, and L. Jiang. An empirical study on developer interactions in Stack Overflow. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1019–1024, New York, NY, USA, 2013. ACM.

[93] S. Wang and X. Yao. Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability*, 62(2):434–443, 2013.

[94] W. Wang and M. W. Godfrey. Detecting API usage obstacles: A study of iOS and Android developer questions. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 61–64, May 2013.

[95] S. N. Woodfield, H. E. Dunsmore, and V. Y. Shen. The Effect of Modularization and Comments on Program Comprehension. In *Proceedings of the 5th international conference on Software engineering*, pages 215–223. IEEE Press, 1981.

[96] Y. Wu, W. Wu, Z. Li, and M. Zhou. Mining query subtopics from questions in community question answering. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 339–345. AAAI Press, 2015.

[97] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing. What do developers search for on the web? *Empirical Software Engineering*, 22(6):3149–3185, 2017.

[98] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun. What security questions do developers ask? a large-scale study of stack overflow posts. *Journal of Computer Science and Technology*, 31(5):910–924, 2016.

[99] A. Zagalsky, C. G. Teshima, D. M. German, M.-A. Storey, and G. Poo-Caamaño. How the R community creates and curates knowledge: A comparative study of Stack Overflow and mailing lists. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 441–451, New York, NY, USA, 2016. ACM.

[100] Y. Zhang, B. Shen, and Y. Chen. Mining developer mailing list to predict software defects. In *2014 21st Asia-Pacific Software Engineering Conference*, volume 1, pages 383–390, 2014.

[101] J. Zou, L. Xu, W. Guo, M. Yan, D. Yang, and X. Zhang. Which non-functional requirements do developers focus on? an empirical study on Stack Overflow using topic analysis. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 446–449, May 2015.

[102] J. Zou, L. Xu, M. Yang, X. Zhang, and D. Yang. Towards comprehending the non-functional requirements through developers' eyes. *Inf. Softw. Technol.*, 84(C):19–32, Apr. 2017.

[103] *Quora discussion about What should I comment on code? What, how, or why?* https://www.quora.com/What-should-I-comment-on-code-What-how-or-why (Accessed on 2020-06-17).

[104] *Quora discussion about What's a good comment/code ratio?* https://www.quora.com/Whats-a-good-comment-code-ratio (Accessed on 2020-06-17).

[105] *Stack Overflow discussion 11269338.* https://stackoverflow.com/questions/11269338 (Accessed on 2020-06-17).

[106] *Stack Overflow discussion 54382326.* https://stackoverflow.com/questions/54382326 (Accessed on 2020-06-17).