

Università  
della  
Svizzera  
italiana

Software  
Institute

# MAPPING THE DOCUMENTATION LANDSCAPE OF OPEN SOURCE PROJECTS

**Tommaso Rodolfo Masera**

September 2023

*Supervised by*  
**Prof. Dr. Michele Lanza**

*Co-Supervised by*  
**Dr. Csaba Nagy**  
**Marco Raglianti**



# Abstract

As software systems grow in complexity, they must be adapted to environmental changes to keep providing satisfactory results. The documentation of software systems is expected to receive the same treatment. As this documentation grows, it constitutes an ever-changing, and increasingly complex documentation landscape composed of diverse documentation sources.

We aim to investigate the evolution of documentation landscapes by examining the sources that compose them. We mine the README files present in GitHub repositories which contain links referencing the documentation sources employed by the respective projects.

We analyze 9,169 projects and present a taxonomy that describes the various types of documentation sources. We develop RagnaDok, a tool to extract data and analyze the evolution of the documentation landscape via the taxonomy that we define. We present an overview of the aggregate documentation landscape across all projects. We finally present three case studies that demonstrate how our approach can be effective when analyzing the documentation landscape of a project, but also its limitations.



To all who have supported me thus far



# Acknowledgements

First and foremost, I must thank my supervisor Prof. Dr. Michele Lanza. During the time spent working on my thesis, I got to see how much of an outstanding professor you are. Other than the veteran expertise in research and the insights that you always had to offer, what was truly inspiring to me was your passion towards teaching. I could feel that you really care to help others learn and I feel lucky that I got the opportunity to learn from you. If teaching was an art (and, in my opinion, it can be), I'd say that you have mastered it.

Secondly, I must extend my thanks to my co-supervisors, Dr. Csaba Nagy and Marco Raglianti. It's hard to overstate how grateful I am for your assistance through this work. It would not have been possible without you. Your dedication and passion towards research should be an inspiration to all.

I must of course thank my family for their unconditional love and support. I would not be who I am today if it wasn't for my mother and father, Manuela and Luca. Truly, I could not have wished for better parents. And to my brother, Pietro, I'm glad we spent our years doing a master and training together.

It's hard to describe how thankful I am towards my dearest friend, Teodora. You have made each and every step of the way feel much lighter, and every bad day much better. If I had the strength to get this far, I owe it to you.

I want to thank my friends that have been by my side ever since my bachelor years here at USI. Claudio, Joey, Andrea, Gianmarco, Federico, Ted, Jacob, Alessio, Joao, and many others. I would not be where I am today without your constant help throughout my years here, from when I was just getting acquainted with the field of informatics until today. I have learned so much from you, and I am glad you stuck around during these years, it was a wild and fun ride.





# Contents

<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Thesis Contributions	2
1.2 Document Structure	2
<b>2 State of the Art</b>	<b>5</b>
2.1 Documentation	5
2.1.1 Early Work	5
2.1.2 The 90s and Early 2000s	6
2.1.3 The Current Scenario	6
2.2 Mining Software Repositories	7
2.3 Conclusion	8
<b>3 Mapping the Documentation Landscape</b>	<b>9</b>
3.1 Documentation Landscape	9
3.2 Dataset Generation and Overview	10
3.2.1 Sourcing and Retrieving the Data	10
3.2.2 Evolutionary Modeling	11
Building README Histories	11
3.2.3 Piecing Histories Together	12
Mimir: The Historian	12
3.2.4 Documentation Source Identification	13
Parsing the README	13
History Chaining	14
Manual Inspection	18
3.3 Conclusion	19
<b>4 Taxonomy of Documentation Sources</b>	<b>21</b>
4.1 Conflicts Between Categories	21
4.2 Blog	23
4.2.1 Medium	23
4.2.2 Wordpress	23
4.2.3 Custom Blog	23
4.3 Forum	24
4.3.1 Custom Forum	24
4.3.2 GitHub Discussions	24
4.3.3 StackOverflow	24
4.4 Mailing List	25
4.4.1 Custom Mailing List	25

4.4.2	Google Groups	25
4.4.3	Mailman	25
4.5	Wiki	26
4.5.1	Custom Wiki	26
4.5.2	GitHub Wiki	26
4.5.3	Wikipedia	26
4.6	Document	27
4.6.1	Multimedia Document	27
	Audio	27
	Image	27
	Video	27
4.6.2	Textual Document	28
	Book	28
	Text File	28
4.7	Homepage	29
4.7.1	Homepage Detection	29
	Project Homepage	29
	Third-Party Project Homepage	30
	Specific Subsection	30
4.8	Repository-Related	31
4.8.1	Bug Tracker	31
	Bugzilla	31
	Jira	32
4.8.2	Issue Tracker	32
4.8.3	Pull Request	32
4.8.4	Relative File	32
4.8.5	Repository	32
	Bitbucket	32
	GitHub	33
	Launchpad	33
	SourceForge	33
	Weblate	33
4.8.6	Source File	33
4.9	Community Platform	35
4.9.1	Custom Community Platform	35
4.9.2	Instant Messaging	35
	Discord	36
	Gitter	36
	IRC	36
	Slack	36
	Telegram	36
4.9.3	Media Sharing	36
	Imgur	37
	Vimeo	37
	YouTube	37
4.9.4	Social Media	37
	Facebook	37
	Instagram	37
	TikTok	38

Twitter	38
4.10 Conclusion	38
<b>5 RagnaDok Implementation</b>	<b>41</b>
5.1 System Architecture	41
5.2 Data and History Mining	42
Using the Log	42
Filtering Commits	43
Building Histories	43
5.2.1 Post-Mining Processing	43
Finding Documentation Sources	44
Rejected Sources Recovery	44
5.2.2 History Chaining	44
5.3 Visualization	45
5.3.1 README History Visualization	45
5.3.2 Documentation Landscape Visualization	48
Documentation Landscape of a Project	48
Aggregate Documentation Landscape	49
Summary View	50
Mining View	50
5.4 Conclusions	51
<b>6 Analysis and Discussion</b>	<b>53</b>
6.1 Dataset	53
6.1.1 Quantitative Analysis	53
Outlier Projects	53
6.2 Documentation Landscape in the Wild	54
6.2.1 Early Years	55
6.2.2 Towards Modern Sources	57
6.2.3 The Current Landscape	57
6.2.4 Documentation Landscape of the Subcategories	58
6.3 Conclusions	59
<b>7 Case Studies</b>	<b>61</b>
7.1 An Explosion of Sources: scikit-learn	61
7.1.1 Overview	62
7.1.2 Beginnings	62
7.1.3 Growth Over the Years	65
7.1.4 Explosion of Sources	66
7.1.5 Conclusion	66
7.2 fish-shell/fish-shell: A Simple Landscape	68
7.2.1 Overview and Evolution	68
7.2.2 Conclusion	69
7.3 GCC: Hidden Landscape	71
7.3.1 Overview	71
7.3.2 Initial State	72
7.3.3 First Documentation Sources	74
7.3.4 Modern Landscape	76
7.4 Limitations	78
7.4.1 Hic Sunt Leones	78

7.4.2 Threats to Validity	79
7.5 Conclusions	79
<b>8 Conclusions</b>	<b>81</b>
8.1 Contributions	81
8.2 Future Work	82
8.3 Final Words	83
<b>A SCC and CLOC Comparison</b>	<b>85</b>
<b>B Inconsistency from GitHub Search</b>	<b>87</b>

# List of Figures

1.1	Evolution of the Documentation Landscape of the scikit-learn Project	2
3.1	Metaphorical Documentation Landscape of a Project	9
3.2	Evolutionary Model for README Files	11
3.3	Evolutionary Model for Chained Histories	13
3.4	Process to Detect Documentation Sources	14
3.5	Chaining Criteria in Practice	15
4.1	The Eight Top-Level Categories in the Taxonomy	21
4.2	Taxonomy of the Blog Category	23
4.3	Taxonomy of the Forum Category	24
4.4	Taxonomy of the Mailing List Category	25
4.5	Taxonomy of the Wiki Category	26
4.6	Taxonomy of the Document Category	27
4.7	Taxonomy of the Homepage Category	29
4.8	Taxonomy of the Repository-Related Category	31
4.9	Taxonomy of the Community Platform Category	35
4.10	Taxonomy of the Documentation Landscape	39
5.1	Architecture of RagnaDok	41
5.2	Example Output of Custom Git Log	42
5.3	Visualization for README histories	45
5.4	View for a Specific README Version	46
5.5	Documentation Landscape of the Elasticsearch Project	48
5.6	Aggregate Documentation Landscape	49
5.7	Summary View of the Mined Projects	50
5.8	Mining View	51
6.1	README History Outliers	54
6.2	Presence in Projects of Top Level Categories of the Taxonomy Over Time	55
6.3	Aggregate Documentation Landscape between 2000 and 2015	56
6.4	Aggregate Documentation Landscape between 2015 and 2023	58
6.5	Documentation Landscape of Community Platforms and its Subcategories	59
7.1	Empty Documentation Landscape for Reference	61
7.2	Documentation Landscape over Time of scikit-learn	62
7.3	Initial Documentation Landscape of scikit-learn	64
7.4	scikit-learn in 2011 After One Year	65
7.5	Commit Referencing the Blog Entry	65
7.6	scikit-learn in 2016	66
7.7	The Explosion of the Documentation Landscape	67
7.8	Documentation Landscape over Time of Fish Shell without Recovered Sources	68

7.9 Fish Shell in 2012 when the First Sources Appeared . . . . .	69
7.10 Evolution of the Documentation Landscape of Fish Shell . . . . .	70
7.11 Documentation Landscape over Time of GCC . . . . .	71
7.12 Documentation Landscape over Time of GCC without Recovered Sources . . . . .	72
7.13 SourceForge and Bugzilla Appearances in the Landscape of GCC . . . . .	73
7.14 GCC in 1993 . . . . .	73
7.15 GCC in 1997 . . . . .	73
7.16 GCC in 1995 without Recovered Sources . . . . .	74
7.17 First Occurrence of a Mailing List in the Landscape . . . . .	75
7.18 SourceForge and Bugzilla Appearances in the Landscape of GCC . . . . .	75
7.19 GCC in 1999 . . . . .	76
7.20 GCC in 2004 . . . . .	76
7.21 Wiki Appearance and Bugzilla Disappearance in the Landscape of GCC . . . . .	77
7.22 Forum and Github Appearances in the Landscape of GCC . . . . .	78

# List of Tables

3.1	Initial Dataset for Repository Mining . . . . .	10
6.1	Repository and History Data . . . . .	53
6.2	Outlier Projects . . . . .	54
6.3	Status of the Aggregate Landscape in its First Five Years (2000-2003-2005) . . . . .	56
6.4	Status of the Aggregate Landscape between 2009 and 2015 . . . . .	57
6.5	Status of the Aggregate Landscape between 2015 and 2023 . . . . .	57
7.1	Descriptive Statistics of scikit-learn . . . . .	62
7.2	Descriptive Statistics of fish-shell . . . . .	68
7.3	Descriptive Statistics of GCC . . . . .	71
7.4	Occurrences of README Extensions Across All README Histories . . . . .	78
A.1	CLOC and SCC Output Comparison . . . . .	85
B.1	List of Projects Excluded from the Analysis . . . . .	87





## Chapter 1

# Introduction

Documentation plays a crucial role in software development. It goes beyond information contained within the source code, encompassing design decisions and supplementary material that support the maintenance and evolution of a software system.

Unfortunately, documentation is oftentimes undervalued as it does not impact the functioning of a software system: Documentation is a non-executable artifact [47], meaning code will run regardless of how well-documented it is. Consequently, documenting software can be perceived as needlessly expensive, as the immediate benefits may seem low compared to the demanding and resource-consuming process of creating documentation.

Even when documentation is present and maintained, numerous issues can arise. As discussed by Aghajani et al., documentation can be incorrect and incomplete, meaning that developers may lose time by following wrong (potentially unsafe) instructions and code examples, or fail to find what they need because it was not documented altogether [1]. Furthermore, documentation can be poorly written and outdated, impacting its readability, usability, usefulness, and maintainability since a developer may struggle to understand it, use it effectively or apply meaningful changes to it [1].

Because of these problems [1] and the lack of useful complementary information [46], a software system that evolves needs to ensure that its documentation co-evolves with it.

When we refer to software documentation, we often imagine source code comments and technical manuals that describe how a system works, but, in reality, it comprises a much broader array of sources. These documentation sources originate from diverse platforms with varying structures (e.g., a blog, a mailing list, an instant messaging channel), collectively forming the *documentation landscape* of a software project [41]. Raglianti has identified four major archetypes of the sources that revolve around a project maintained over a versioning system: Code, Documents, Multimedia, and Community [41]. Within these archetypes, he identified thirteen source types to be relevant. Leveraging this classification, we aim to develop a fine-grained taxonomy to classify the documentation sources.

To better introduce the concept of documentation landscape, we present an interesting case that Raglianti et al. found (Figure 1.1). This view presents the documentation sources that appeared in the main README file of the *scikit-learn*<sup>1</sup> project over time. We can observe how the documentation landscape of the scikit-learn project is defined by its sources, and evolves over time. The project did not present many documentation sources for a decade until a sudden and rapid growth in recent years. This is a single case study that was examined in depth and is not representative of all projects. With our thesis, we want to delve deeper into the documentation landscape of many open-source projects, develop a taxonomy of the documentation sources that compose it, and map the landscape to observe its evolution across single and multiple projects. This research aims to identify documentation evolution patterns, potentially similar to the one exhibited by the scikit-learn project.

---

<sup>1</sup><https://github.com/scikit-learn/scikit-learn>

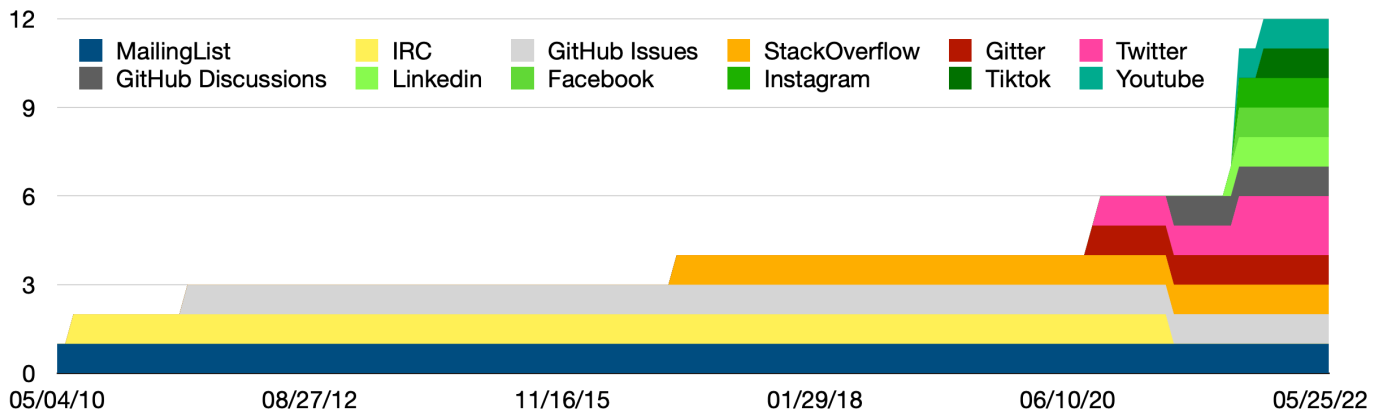


FIGURE 1.1: Evolution of the Documentation Landscape of the scikit-learn Project [43]

We have built RagnaDok, a tool that mine software repositories on GitHub data, reconstructs the history of their README files, and ultimately visualizes them to analyze the various landscapes that appear across projects via the documentation sources of README files as they evolve over time. By analyzing the evolution of README files and determining the documentation sources that they contain, we aim to explore the documentation landscape and shed light on how documentation and its sources have evolved in the last two decades.

## 1.1 Thesis Contributions

The main contributions of this thesis consist in:

1. A comprehensive taxonomy that describes the documentation sources that compose the documentation landscape.
2. An approach to mine, model and reconstruct the history of README files.
3. RagnaDok: a tool that implements our approach and taxonomy.
4. Visualizations of the evolution of README files and of the documentation landscape.
5. The resulting documentation landscape from 9,169 software systems.
6. Analysis on case studies that show the documentation landscape of open-source projects.

## 1.2 Document Structure

We organize the document as follows:

- In Chapter 2, we describe the state of the art with respect to the evolution of software and its documentation and repository mining.
- In Chapter 3, we describe our approach to create the documentation landscape via the taxonomy that we employ to categorize the documentation sources. Additionally, we present the evolutionary domain modeling to represent the evolution of README files in a repository.
- In Chapter 4, we present the taxonomy of the documentation landscape, to categorize the documentation sources.

- 
- In Chapter 5, we present RagnaDok, a supporting tool to implement our approach, extract data from README files, analyze the data to extract documentation sources, and visualize the documentation landscape.
  - In Chapter 6, we present an overview of the aggregate documentation landscape that we derive from all the projects in our dataset.
  - In Chapter 7, we provide an analysis and insights over three case studies to show examples of the documentation landscape of a project, the benefits, and the limitations of our approach.
  - In Chapter 8, we summarize our contributions and present potential directions for future work.



## Chapter 2

# State of the Art

### 2.1 Documentation

Over the past fifty years, the generation of software documentation and software have evolved in tandem. Numerous platforms and tools have emerged to aid developers in communication and, consequently, in producing software documentation. For instance, many modern tools and platforms allow developers to discuss code and provide code examples (*e.g.*, forums [36], instant messaging platforms [10], mailing lists [15]).

In the early days of software development, face-to-face interaction was predominant, as programs written between the 1960s and the 1970s were relatively small and teams often worked in the same place [57]. However, with the advent of modern platforms (*e.g.*, Discord) developers can now collaborate across the globe, without ever even meeting one another in person.

#### 2.1.1 Early Work

As early as 1976, Tausworthe considered documentation fundamental for the creation and maintenance of quality software [59]. He defined a taxonomy of documentation levels to standardize the documentation process, aiming to facilitate the generation of new documentation and reduce the time needed to review or redesign software.

Seminal empirical studies conducted by Shneiderman *et al.* and Ramsey *et al.* on the utility of early forms of documentation (*i.e.*, flowcharts and program design language) played a critical role in shaping documentation practices [54, 45]. Although no statistical difference was initially found between groups that used flowcharts compared to groups that did not use any, program design languages were deemed to offer a better aid to developers when compared to flowcharts [54, 45]. This research laid the foundation for discussions on the effectiveness of various forms of documentation.

In the early 1980s, Sheppard *et al.* conducted an empirical study on the impact that documentation has on software developers' performance, highlighting the ongoing relevance of documentation already in that era [52]. This research led to the development of innovative tools like Playback by Neal and Simons. Playback was a tool to record user interactions (*e.g.*, keystrokes). Developers would be told to solve specific tasks given a piece of software and its documentation [33].

The advent of new technologies, such emails, marked the beginning of remote collaboration, overcoming physical distance. These technologies paved the way to a new approach to software development. With these, developers were able to contact each other remotely, but this could disrupt the workflow. For instance, phone calls could be highly beneficial, as they allowed developers to discuss technical matters remotely but they inevitably caused an interruption [57]. Emails already mitigated this by offering an asynchronous way of communication that is less disruptive. The need for tools that would simplify remote communication was rising.

### 2.1.2 The 90s and Early 2000s

In 1995, [Parnas and Madey](#) advocated for a functional way to document software systems and design decisions by conforming it to a formal mathematical standard to resemble the way engineering documents were produced [35]. They stated that software documentation is not as clear or detailed as its engineering counterpart; while the latter allows to calculate and derive results, the former limits itself to only offer example usage scenarios and descriptions of what can be expected from the software or of appealing features. Furthermore, they highlighted the issue of software documentation being considered a chore rather than an essential task, inevitably resulting in outdated documents that do not reflect the current state of the software. Software ageing further exacerbates these issues and plays a part in the way documentation is produced too. [Parnas](#) stated that software can age poorly. For instance, a piece of software that is maintained by many developers via different approaches, slowly loses its original design. Because documentation is already neglected, these different changes become harder and more inconsistent as time goes on [34]. Documentation was also deemed to be an “unattractive” research subject. Quoting from Parnas:

“Last year, I suggested to the leader of an Esprit project who was looking for a topic for a conference, that he focus on documentation. His answer was that it would not be interesting. I objected, saying that there were many interesting aspects to this topic. His response was that the problem was not that the discussion wouldn’t be interesting, the topic wouldn’t sound interesting and would not attract an audience [34].”

During the same period, developers began to adopt new tools to generate and share documentation. Instant messaging platforms such as ICQ<sup>1</sup> and IRC started to be used by developers for technical discussions [17, 20, 57], although they were not adopted consistently across developer teams. Instant messaging channels were not the only documentation sources to rise in the early 2000s. With the advent of the internet, we can see websites such as blogs, forums and Q&A sites (such as StackOverflow<sup>2</sup>) slowly appear on the web. [Parnin et al.](#) described that StackOverflow, in particular, introduced the concept of crowd-sourced documentation. Documentation can emerge from both questions and answers that include discussions and code examples regarding Application Programming Interfaces (APIs) [36]. [Ponzanelli et al.](#) developed Prompter, an Eclipse IDE plugin, to aid developers by automatically retrieving relevant StackOverflow posts given the code context in the IDE [38]. [Treude and Robillard](#) used machine learning techniques to enhance API documentation via StackOverflow posts to offer further insights that were not contained in the official documentation [60].

This era of software documentation culminates with social media (*e.g.*, Facebook in 2004), giving people new ways to communicate and collaborate. Developers began making use of social media to enhance software development [4]. [Storey et al.](#) describe how social media is used by developers to stay informed, to coordinate their work and to create informal documentation, further reinforcing the concept of crowd-sourced documentation [56].

### 2.1.3 The Current Scenario

In the present day, we observe a large number of heterogeneous documentation sources that constitute the documentation landscape of a project. As a practical example, we can take the *scikit-learn*<sup>3</sup> repository on GitHub. Its README file, prominently displayed on the main page of the repository, reports a section for documentation and communication which includes various documentation sources.

Software developers now rely on a plethora of sources to maintain and produce documentation. These include (but are not limited to) instant messaging platforms (*e.g.*, Gitter [10], Discord [42], Slack [7]), mailing

---

<sup>1</sup><https://icq.com/>

<sup>2</sup><https://stackoverflow.com/>

<sup>3</sup><https://github.com/scikit-learn/scikit-learn>

lists [15], crowd-sourced documentation from forums, Q&A sites (e.g., StackOverflow), social media (e.g., [Twitter for commits](#)<sup>4</sup> linked on [scikit-learn](#)), multimedia (e.g., YouTube videos, podcasts), blogs and external websites that do not adhere to any of the previous categories such as personal/company websites that have a completely arbitrary structure (e.g., the [scikit-learn website](#)<sup>5</sup>). [Mezouar et al.](#) found that Twitter could have helped developers with finding bugs several days prior to their discovery [32].

From this current situation, we can observe that the documentation sources present many different attributes. One of these attributes is their persistency. For instance, instant messaging platforms tend to be fast-paced and volatile compared to forums and mailing lists [41]. The high throughput of messages can bury a topic of discussion and quickly shift the focus while a forum represents a more organized archive, with threads that prevent discussions to completely trail off.

Documentation, now more than ever, takes a multi-faceted shape characterized by the variety of the sources that are used to generate it. Crowd-sourced documentation has become commonplace, transforming the documentation landscape of software projects into a complex ecosystem with diverse and heterogeneous components.

## 2.2 Mining Software Repositories

Like documentation, software versioning and maintenance has been a relevant subject all the way back to the 1970s, with the first version control system (SCCS) in 1972 [48, 49]. Today, Git stands as the dominant version control system, with platforms like GitHub serving as popular hubs for open-source projects. In particular, GitHub's popularity and easy-to-access nature make it a great resource to obtain a relevant dataset for both our thesis and Mining Software Repositories (MSR) research.

While GitHub offers a wealth of data for research, it comes with its own challenges. [Bird et al.](#) highlighted the promises and perils that come with mining Git repositories [5]. [Kalliamvakou et al.](#), more specifically, analyzed 434 GitHub repositories and found that most of them are not projects, have a low number of commits, are personal (i.e., no collaboration), or are inactive [23].

Over the years, numerous approaches have been developed to aid MSR research in extracting useful data from software repositories. [Spadini et al.](#) developed PyDriller, a Python framework for mining Git repositories [55]. [Salis and Spinellis](#) created RepoFS, a tool to visualize a Git repository as a virtual user-level file system [51]. [Clem and Thomson](#) applied static analysis to GitHub repositories to allow better navigation and understandability of the code [8]. They allow users and developers alike to more easily explore the information contained in a project.

Our thesis focuses on README files within software repositories. The official GitHub documentation<sup>6</sup> specifies that a README file should typically include information about what the project does, why the project is useful, how users can get started with the project, where users can get help with the project, and who maintains and contributes to the project. GitHub README files have already been used to extract build commands [18] and software requirements [39]. In addition, READMEs allowed researchers to extract developer skills to recommend jobs to developers based on what projects they worked on [19] or, conversely, to recommend potential employees to a company [14].

Lastly, [Prana et al.](#) have conducted a study over the content of README files to take a first look at what the content of a README document is really like [40]. They provided a manual and extensive annotation and classification of the sections that a README file can include, and they also designed a classifier to automatically predict such categories of sections.

README files on software repositories can contain many useful insights, and that is why we choose them as our target for data mining.

---

<sup>4</sup>[https://twitter.com/sklearn\\_commits](https://twitter.com/sklearn_commits)

<sup>5</sup><https://scikit-learn.org/stable/>

<sup>6</sup><https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-readmes>

## 2.3 Conclusion

Software documentation has been an integral part of the software development landscape for decades. From its early beginning in source code comments and face-to-face interactions to the complex, crowd-sourced documentation landscape of today, it has continually evolved to meet the changing needs of developers.

Platforms like GitHub hold an abundance of data and present many challenges to be faced when trying to analyze these data. Nevertheless, these platforms are a crucial avenue for research in the MSR field, so crucial that much of the research has been dedicated to the development of specific tools to help explore software repositories.

In our research, we focus on the pivotal role of README files within software repositories, aiming to leverage their content for insights into documentation practices.



## Chapter 3

# Mapping the Documentation Landscape

In this chapter, we present our approach to gather and analyze data regarding the documentation sources that make up the documentation landscape. We begin by defining what a documentation landscape is, and we explain how we aim to extract the data necessary to represent the landscape in two main steps. Firstly, we analyze how README files evolve over time in a project. Secondly, we explore the data from the README to identify potential documentation sources in their content. These documentation sources are then used to map the documentation landscape of the project.

### 3.1 Documentation Landscape

We define the documentation landscape of a project as the resulting composition of all its documentation sources. To better understand the documentation landscape, let us use a metaphor. A documentation source is a URL that references a location that provides complementary documentation regarding the project (*e.g.*, the project's homepage, a wiki, a mailing list).

A documentation landscape of a project can be viewed as a house and its garden (Figure 3.1). The house represents the project that was taken into consideration, while the flowers in the garden represent each documentation source. Flower patches of similar flowers would then form a category of sources. The documentation sources may also be external to the project, and may point to sources larger than the project itself. These are not part of the garden *per se*, they are flowers that belong to other gardens that appear in this project.

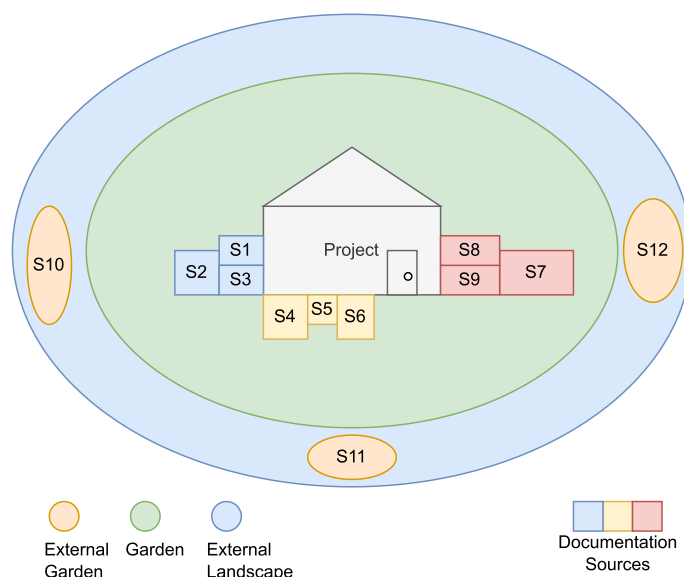


FIGURE 3.1: Metaphorical Documentation Landscape of a Project

The state of a project’s garden depends heavily on the documentation sources that are present. A project can go from a neatly ordered garden to a barren or overgrown garden of documentation sources.

## 3.2 Dataset Generation and Overview

Our initial entry point is a dataset acquired via GitHub Search [9]. We gather an initial total of 9,534 repositories with various filtering parameters (Table 3.1). We set a minimum threshold for commits, contributors and stars. This allows us to gather repositories with a good amount of contributions and popularity, indicating a reasonable size and relevance. In addition, we make sure to exclude forked projects, as these would present duplicate landscapes. We vet the results from GitHub Search by removing duplicates, forks, and projects that no longer exist. The final number of repositories that we obtain is 9,169. We get this lower number of projects due to inconsistencies between GitHub Search and the GitHub API. The inconsistencies can manifest when the database of GitHub Search is outdated or when an alias is used to call the GitHub API. In the first case, it is possible that an outdated project has been deleted and, therefore, it cannot be mined anymore. We encounter this case, for example, with the *chinnkarahoi/jd\_scripts* project,<sup>1</sup> which appears in the GitHub Search dataset, but is no longer accessible. In the second case, we find that multiple aliases lead us to the same project, resulting in duplicates. These aliases can also lead to forked projects. In our final dataset, we removed 23 projects that are no longer accessible and 8 projects which are forks (see Appendix B). Finally, 3 projects fail to load due to problems with memory, and we also excluded them.

Repository #	Commit #	Contributor #	Star #	Forks	Requested On
9,169	2,000+	20+	100+	Excluded	2022-11-10

TABLE 3.1: Initial Dataset for Repository Mining

We now retrieve the README files of these projects to extract the documentation sources that are present in their content.

### 3.2.1 Sourcing and Retrieving the Data

We extract data from software repositories hosted on GitHub. These repositories are versioned using Git,<sup>2</sup> a distributed version control system which we leverage to reconstruct the evolution of README files. Git tracks the development history of a system via commits. Specifically, each commit keeps track of relevant information about which files were affected by it, when it was made and who made the commit. Via commits, it is possible to reconstruct snapshots of the system. The main actions that can be performed on a file in a Git commit are:

- **Addition:** A new file is created and added to the Git repository for tracking.
- **Deletion:** A file is deleted and ceases to exist in the Git repository.
- **Modification:** An existing file is modified.
- **Rename:** An existing file is renamed without being moved to another path.
- **Move:** An existing file is moved to a different path. It has priority over a file rename in case the name is also changed.

<sup>1</sup>[https://github.com/chinnkarahoi/jd\\_scripts](https://github.com/chinnkarahoi/jd_scripts)

<sup>2</sup><https://git-scm.com/>

On top of commits, Git also relies on the concept of branches and tags. A tag is used to mark a specific commit within the project. Typically tags are employed to mark a release version of a system. A branch in a Git repository represents an alternate timeline of the project. Usually a project has a stable default branch called `master` or `main` where all the features and changes are merged when ready. This default branch is the “main timeline” of the project, the one that presents the fully functioning system. At any point in time, another branch can be made, splitting from the main timeline and adding separate changes to the project. This branch can then be merged back into the stable branch when it is ready. A typical use case for branches involves a developer who needs to create a new feature for a system. They create a new branch, commit their changes to introduce the new feature, and finally merge this branch with the stable branch. Our analysis focuses on the main stable branch of each project that we examine.

Commits are the main medium that we use to extract the data as we can easily check the version of a repository given a commit.

To be able to begin this extraction, we must model the evolution of a README file and build its history.

### 3.2.2 Evolutionary Modeling

We model the evolution of the README files within a repository by representing them via the following two entities (Figure 3.2):

- **Readme History:** It contains an ordered list of Readme Version to temporally represent the file and the path to the README file it refers to. It also contains data to know if this particular file was deleted on a certain date or whether it was renamed from or to another filename.
- **Readme Version:** It contains all the relevant data of the README file such as the date and commit hash of the version, the content of the file and its documentation sources contained within.

The notion of history is the main component of the model and, through each version that it contains, it allows us to track the evolution of the file through time.

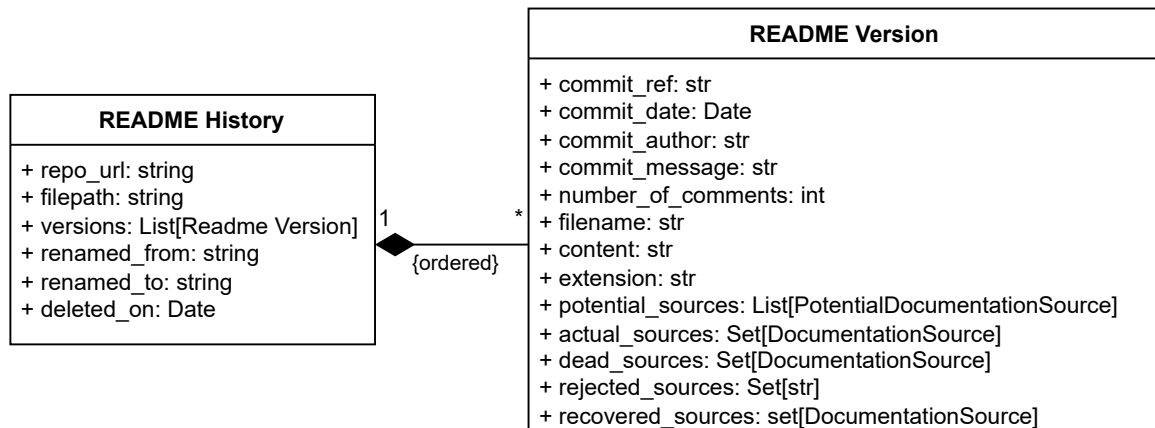


FIGURE 3.2: Evolutionary Model for README Files

#### Building README Histories

Using Git commits, we identify those that are relevant to us. A commit is relevant when it modifies any README file in the repository. With these commits, we are able to build the README histories of the project. In addition, we separately keep track of commits that involved the deletion or the rename of a README file. We use these to be able to determine when a history ends and to simplify history chaining (see Section 3.2.3).

### 3.2.3 Piecing Histories Together

The model we just presented is enough to represent histories, but it is not complete for our purpose. Mining data for README histories is not as straightforward as it may appear. For each unique README file, we can get every version of such file and build its history. The path to the file is unique, but this same file may be renamed, moved to a separate directory, or deleted altogether and recreated with the same name later down the line with a completely different structure and meaning. Because of this, the same file can be represented by multiple histories.

Git provides information about file renames. For example, if one file is moved or renamed without changing its content and then committed, Git will see that as a rename. If this file is renamed or moved and its content is changed significantly, in the *same* commit, Git will state that the old file was deleted and that a new file was added while, in fact, this is the same file that had its content modified. Git does not explicitly track commits, but tries to detect it. To ensure that the rename is detected, the file move must be added to the Git index. The Git index is the staging area between the workspace and the repository, where all changes can be set up and combined before committing them. If a developer fails to add this move to the index prior to committing, the rename will not be detected. Given this limitation in tracking file histories, we cannot solely rely on the renames provided by Git and we must implement our own way to piece histories together: Mimir.

#### Mimir: The Historian

Git's limitation in chaining histories and the need to manage the numerous histories that a repository can have motivate the need for an entity to take the role of our historian: Mimir, named after the figure in Norse mythology. The model in Figure 3.2 can thus be extended with the following new entities and relationships (see Figure 3.3):

- **Chained README History:** It is the concatenation of the sub-histories that compose it, sorted by version dates.
  - **subHistories:** A chained README history contains all the histories that compose it as separate entities, to keep them individually accessible.
  - **flattenedHistory:** For continuity between the README versions contained within the sub-histories of a chained README history, we flatten and sort the sub-histories into a single history.
- **Mimir:** It keeps track of all the README histories of a specific repository and processes the data to chain together the histories. We tie each project that we mine to a Mimir. Each Mimir oversees the histories of the project it is assigned to.
  - **histories:** Each Mimir has access to all the histories of the project that it oversees.
- See the definition above for README History and README Version.

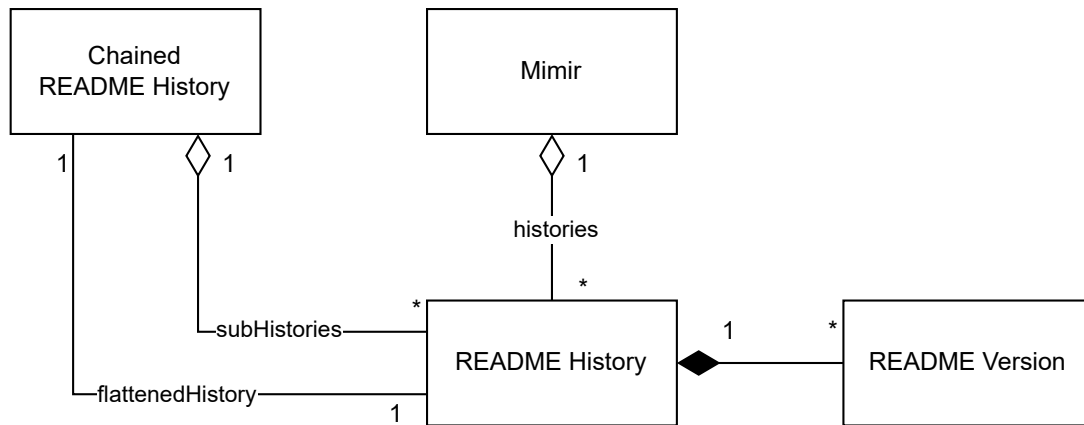


FIGURE 3.3: Evolutionary Model for Chained Histories

When we chain histories, we are checking whether two (or more) separate histories represent the same file. For instance, if `README.txt` is renamed to `README.md` at some point in time during the development, we have two histories for these two names that, in reality, represent the same file. A file rename guarantees that the same file is represented by multiple filenames and allows us to save expensive computations.

### 3.2.4 Documentation Source Identification

Our thesis is based on the assumption that, if a URL is important enough to exist in the README file of a project, then it could (and should) point to a relevant documentation source. Despite this, not all URLs are documentation sources (*e.g.*, link to a logo image, a badge).

In addition, we must consider the status of a documentation source, which can be either live or dead. We define a live source as a URL that points to a source that responds to a HEAD or GET request and that could potentially be visited via a browser (*e.g.*, a URL that points to `https://example.com`). A dead source is instead a URL that does not respond to requests, the link is effectively dead in this case. The source is indeed dead when its URL is broken, but there is a possibility that it used to work perfectly fine at the time it was referenced.

### Parsing the README

To identify the documentation sources contained within a README file, we rely on regular expressions that identify patterns for URLs and non-URLs. The non-URLs in our case are e-mail addresses, IRC nodes and filenames. These help us with finding what we consider a *potential* documentation source, which we must verify to be legitimate (*i.e.*, identify actual source, see Figure 3.4).

Regular expressions work well with our approach because we need to find these patterns in a large number of README files and remain efficient despite their limits (they can capture parts of the README content that are not documentation sources). Furthermore, the verification step allows us to find out whether a documentation source is live or dead.

We also maintain a record of what we reject during the identification process. These rejected sources are then re-examined to check if any source can be recovered. We apply this further step because some documentation sources may be truncated URLs, or some non-URLs that we cannot ping for a result. For instance, a README file may reference a source file without indicating its proper path. Such a source file is a documentation source that we would fail to capture. The recovery step allows more room for error, allowing us to find documentation sources that could be relevant and that remained undetected in the prior identification step. Sources can remain undetected when we reject them, so we make a second pass on the

rejected sources. Due to the potential noise that recovered sources can add to the analysis, we introduce the option to exclude them. We will exclude them in our analysis, unless we specify otherwise.

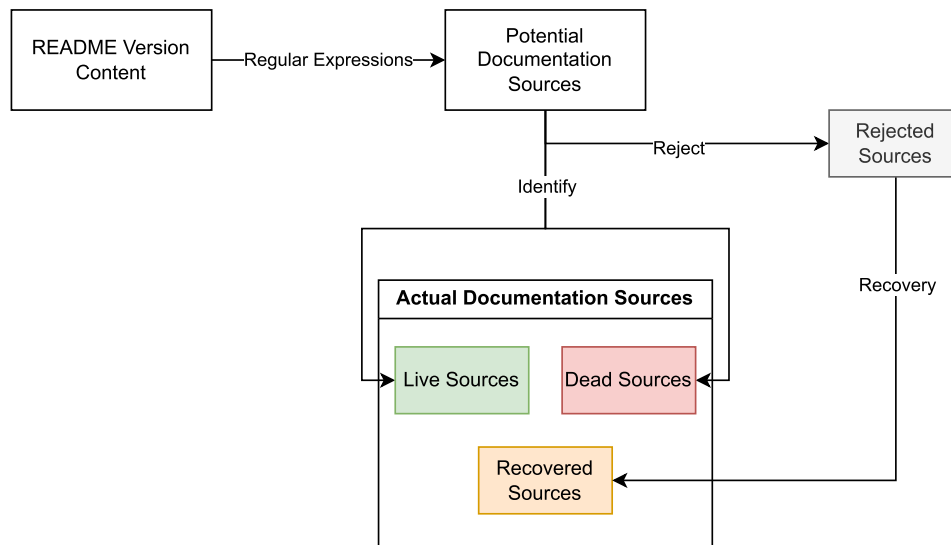


FIGURE 3.4: Process to Detect Documentation Sources

## History Chaining

With the data that we use to represent README files over time and their documentation sources, we move onto the final step for processing our data: history chaining.

As we have previously mentioned, multiple histories may pertain to the same file that was renamed over time and, in particular, Git itself is unable to keep track of some of these actions in specific situations (e.g., if a file is renamed *and* heavily modified in the *same* commit, then the rename will be lost, the old filename will be marked as deleted and the new filename will be marked as added). To address this issue we propose the chaining process described in Algorithms 1 and 2. The chaining algorithm belongs to Mimir.

The chaining algorithm is computationally expensive, as it iteratively chains pairs of histories, until no more histories can be chained. In particular, to be able to compare histories and find out whether they should be chained, we match every pair of histories that satisfies a specific criterion.

The first step to the algorithm involves using the renames that Git successfully identified. If a rename is confirmed to have happened between two histories, then we can be sure that they represent the same file, and we can chain these histories together.

We then identify four different criteria that represent the temporal relationship between two histories. We exemplify them in Figure 3.5. A Chain Criterion can take one of the following values:

1. **Perfect Match:** When, given a pair of histories, a history ends on the exact same commit and date where the other history begins. This is a specific case of overlap, and we choose to examine it separately as an optimization. This is because perfect match in both date and commit hash with a history's end and another's beginning may mean that the file was renamed and that Git did not detect it.
2. **Separate:** When there is no overlap between the two histories.
3. **Contained:** When one of two histories in a pair begins after and ends before the other respectively.
4. **Overlapping:** When one history ends after the other has begun, excluding the cases in which one of the two histories is contained in the other.

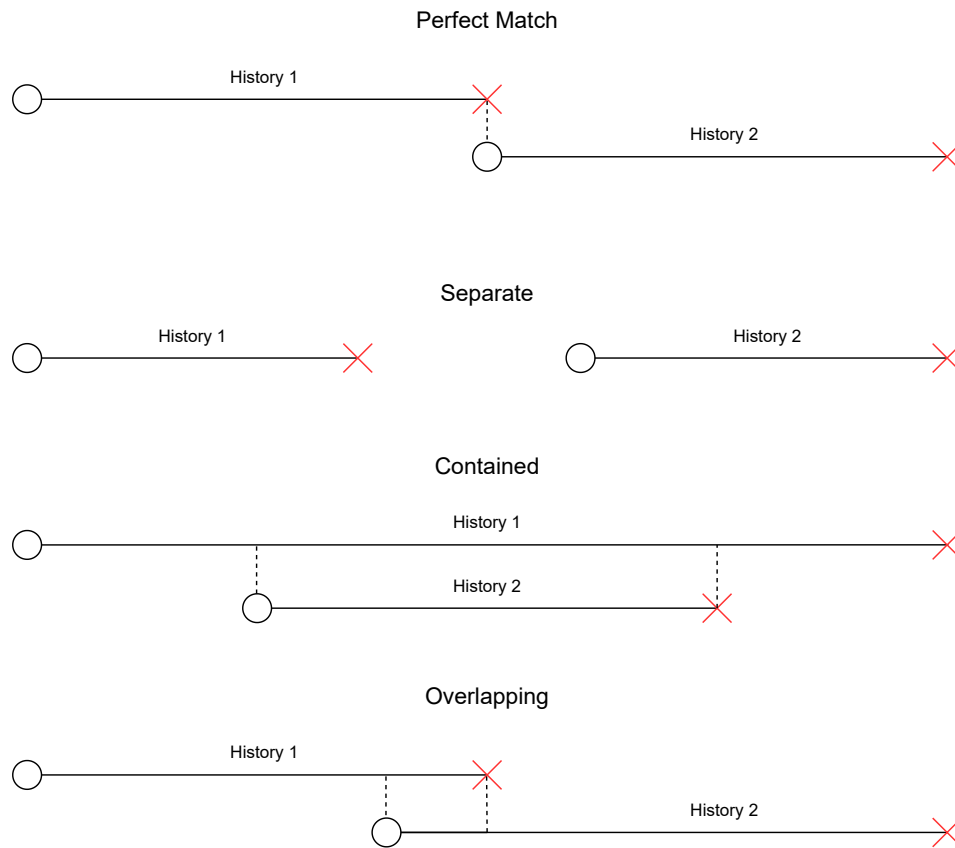


FIGURE 3.5: Chaining Criteria in Practice

**Algorithm 1** Algorithm to Chain Histories

```

1: procedure CHAINHISTORIES
2:   INPUT: histories
3:   OUTPUT: chainedHistories
4:   if repositoryPreviouslyMined then
5:     SimilarityPairs ← readSimilarityPairs()
6:   end if
7:
8:   chainedHistories ← chainByRename()
9:   criteria ← [Main Considered Criteria]
10:
11:  while hasPendingChaining do
12:    // ASSUME THIS IS THE LAST TIME WE CHAIN, BECOMES TRUE IF WE CHAIN
13:    hasPendingChaining ← False
14:    intermediateChainedHistories ← List()
15:    for all criterion in criteria do
16:      pairs ← historyPairs(chainedHistories, criterion)
17:      chainPairs(pairs, intermediateChainedHistories)
18:    end for
19:    if not hasPendingChaining then
20:      pairs ← historyPairs(chainedHistories, ChainCriterion.OVERLAPPING)
21:      chainPairs(pairs, intermediateChainedHistories)
22:    end if
23:    // ADD UNCHAINED HISTORIES TO INTERMEDIATE RESULT FOR NEXT ITERATION
24:    refillResults(chainedHistories, intermediateChainedHistories)
25:    chainedHistories = intermediateChainedHistories
26:  end while
27: end procedure

```

▷ Use previously computed pairs if they exist

▷ Use renames for first chains

▷ Calculate and chain pairs for each main criterion

▷ No chain so far, try with overlapping criterion

When we look for pairs that are eligible for chaining, we must note that we check overlapping pairs only if no other criterion chained successfully. This is because, in the case of many histories that co-exist, one history is likely to overlap with many other unrelated histories. We do this as an optimization as the chaining algorithm is very expensive, and reducing the number of pairs that we compare is paramount. We often encounter projects with a large number of separate README histories that have co-existed at some point in time (e.g., the Arduino project<sup>3</sup> by itself has 111 separate histories prior to chaining).

After obtaining the required pairs for each criterion, we compare the matching versions of the histories (e.g., last version of the first history in the pair with the first version of the second history in the pair) and we measure their similarity. using the common diff utility, as implemented in the `diff` Python standard library<sup>4</sup>. If the similarity ratio that we obtain is above a selected threshold, we chain the histories and we create a chained history.

Once a history is chained, we move onto the next step of the chaining. We must now check for transitive histories. Transitivity between a pair of chained histories manifests when each of the histories share a common sub-history. In this case, we can assume that both of these histories can be further chained together. With transitivity, we can save time and resources during chaining.

For further clarification, Algorithm 2 contains the more relevant sub-functions used during the chaining. The functions `makeChainedHistories` and `chainTransitiveHistories` are not included. They produce a chained README history in both cases, with the latter using the transitivity that we just defined to generate the chained history.

---

<sup>3</sup><https://github.com/arduino/Arduino>

<sup>4</sup><https://docs.python.org/3/library/difflib.html>



**Algorithm 2** Subprocedures for History Chaining (cont.)

---

```

1: procedure CHAINBYRENAME
2:   result = EmptyList
3:   for all history1, history2 in histories do
4:     if history1 renamed from history2 or history1 renamed to history2 then
5:       result.append(makeChainedHistory(history1, history2))
6:     end if
7:   end for
8:   refillResults(histories, result)
9: end procedure
10:
11: procedure REFILLRESULTS(chainedHistories, intermediateChainedHistories)
12:   INPUT: chainedHistories, intermediateChainedHistories
13:   OUTPUT: union of chainedHistories and intermediateChainedHistories
14:   result ← intermediateChainedHistories
15:   for all history in chainedHistories do
16:     found ← False
17:     for all intermediate in intermediateChainedHistories do
18:       if history = intermediate then
19:         found = True
20:       else if history is a chained history and intermediate is a normal history then
21:         if intermediate in history.subHistories then
22:           found ← True
23:           break
24:         end if
25:       else if intermediate is a chained history and history is a normal history then
26:         if history in intermediate.subHistories then
27:           found ← True
28:           break
29:         end if
30:       else if history and intermediate are both chained histories then
31:         for all sub1, sub2 in history.histories, intermediate.histories do
32:           if sub1 == sub2 then
33:             found ← True
34:             break
35:           end if
36:         end for
37:       end if
38:       if found then
39:         break
40:       end if
41:     end for
42:     if not found then
43:       result.append(history)
44:     end if
45:   end for
46:   return result
47: end procedure

```

---

---

Subprocedures for History Chaining

---

```

48: procedure HISTORYPAIRS(histories, criterion)
49:   INPUT: histories, criterion
50:   OUTPUT: all history pairs that satisfy criterion
51:   pairs ← List()
52:   for all history1 in histories do
53:     for all history2 in histories do
54:       // DEPENDING ON THE CRITERION, CALL A DIFFERENT PAIR FUNCTION
55:       if criterion = ChainCriterion.PERFECT_MATCH then
56:         // RETURN A PAIR IF HISTORIES HAVE MATCHING DATES AND COMMIT HASHES
57:         p ← perfectMatchPairs(history1, history2)
58:       else if criterion = ChainCriterion.SEPARATE then
59:         // RETURN A PAIR IF HISTORIES EXIST SEPARATELY NEVER OVERLAPPING
60:         p ← separatePairs(history1, history2)
61:       else if criterion = ChainCriterion.CONTAINED then
62:         // RETURN A PAIR IF ONE HISTORY EXISTS ENTIRELY DURING THE OTHER
63:         p ← containedPairs(history1, history2)
64:       else if criterion = ChainCriterion.OVERLAPPING then
65:         // RETURN A PAIR IF THE HISTORIES OVERLAP
66:         p ← overlappingPairs(history1, history2)
67:       end if
68:       if p is not Null then
69:         pairs.append(p)
70:       end if
71:     end for
72:   end for
73:   return pairs
74: end procedure
75:
76: procedure CHAINPAIRS(pairs, intermediateChainedHistories)
77:   INPUT: pairs, intermediateChainedHistories
78:   OUTPUT: nothing, updates intermediateChainedHistories
79:   for all p in pairs do
80:     if appendChainedHistory(p, intermediateChainedHistories) then
81:       hasPendingChaining ← True
82:       intermediateChainedHistories ← chainTransitiveHistories(intermediateChainedHistories)
83:     end if
84:   end for
85: end procedure
86:
87: procedure APPENDCHAINEDHISTORY(p, chainedHistories)
88:   pair = ReadmeVersionPair.from(p)
89:   if pair in similarityPairs then
90:     similarity ← similarityPairs[pair]
91:   else
92:     similarity ← diffSimilarity(pair.first.content, pair.second.content)
93:     similarityPairs[pair] ← similarity
94:   end if
95:   if similarity ≥ THRESHOLD then
96:     chainedHistories.append(makeChainedHistory(p.first, p.second))
97:     return True
98:   end if
99:   return False
100: end procedure

```

---

## Manual Inspection

Our ultimate goal is to capture the documentation landscape via the sources that compose it. We achieve this via the taxonomy that we develop (Chapter 4) iteratively via a manual inspection of the data.

This manual analysis is a fundamental step to define a comprehensive and complete taxonomy. We develop the taxonomy starting from the archetypes and source types defined by Raglianti [41]. We started with a sample set of 5 representative projects:

- scikit-learn/scikit-learn<sup>5</sup>

---

<sup>5</sup><https://github.com/scikit-learn/scikit-learn>

- [arduino/Arduino](https://github.com/arduino/Arduino)<sup>6</sup>
- [exoplatform/platform](https://github.com/exoplatform/platform)<sup>7</sup>
- [apache/tomcat](https://github.com/apache/tomcat)<sup>8</sup>
- [pmd/pmd](https://github.com/pmd/pmd)<sup>9</sup>

We discussed the proposed categories and specifically focused on uncategorized cases. By focusing on those cases that we failed to capture in the taxonomy, we are able to modify the taxonomy accordingly.

In each subsequent iteration of the taxonomy, we aim at fitting documentation sources in the current taxonomy, and we have two cases. We either accommodate for everything in a satisfactory way, or we identify new sources or better categories and must refine the taxonomy. After each step, we disambiguate potential conflicts (*e.g.*, a category is too broad, too specific, overlaps with another category). When the conflicts are solved, the categories change accordingly. They can be split into multiple categories, merged into broader categories, or new categories can appear or disappear altogether.

### 3.3 Conclusion

In this chapter we described our approach to model, extract, and examine evolutionary data for README files and their documentation sources. We introduced the documentation landscape and the documentation sources that compose it. With our approach defined, we can develop our taxonomy that we present in the next chapter.

---

<sup>6</sup><https://github.com/arduino/Arduino>

<sup>7</sup><https://github.com/exoplatform/platform>

<sup>8</sup><https://github.com/apache/tomcat>

<sup>9</sup><https://github.com/pmd/pmd>



## Chapter 4

# Taxonomy of Documentation Sources

In this chapter, we present our taxonomy to capture the documentation sources and define the documentation landscape. We design the taxonomy as a hierarchy where the root node is the abstract concept of Documentation Source.

To reach the version of the taxonomy that we present, we took existing documentation sources that we detected and then we manually annotated the sources with categories from the taxonomy. We repeat this process iteratively (see Section 3.2.4).

A common pattern in the taxonomy is that some subcategories tend to have specific names of platforms (e.g., Wordpress) and a generic subcategory at the same level called *Custom [Category Name]* (e.g., Custom Blog). This is because more widespread platforms are easily identifiable and can be more easily isolated as a phenomenon, indicating a category (e.g., Wordpress). A less used, or project-specific platform (e.g., a project's blog) cannot be isolated as its own widespread medium and becomes a custom blog.

Under Documentation Source, we define eight top-level categories (Figure 4.1). We present the entire taxonomy in Figure 4.10. Each leaf subcategory in the hierarchy has its own heuristic to determine whether it belongs to that category. When a category is detected, its own supercategories can be derived by following the hierarchy upwards. In the following sections we define each top-level category and their sub-categories. The detection methods can vary. Some are simpler, using only string matching while others required more complex approaches such as regular expressions or partial string matching on specific parts of the URL that points to a documentation source. Unless specified otherwise, we focus on the top level domain of the URL for the string matching (e.g., given `https://example.com/examples/myCoolExample.html`, the top level domain is `example.com`).

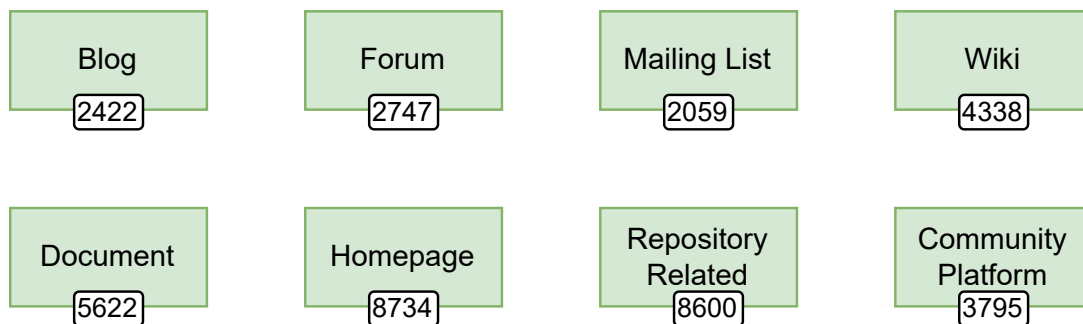


FIGURE 4.1: The Eight Top-Level Categories in the Taxonomy

In the following sections, we are going to define the categories and subcategories of the taxonomy.

### 4.1 Conflicts Between Categories

Due to the detection methods that we have chosen, the same documentation source can be detected by multiple categories. To prevent inconsistencies, we develop priority rules between categories to solve the

conflicts. If more than one category detects the source, then the one with higher priority will be assigned to the source. We prioritize categories in order as follows. For example, if a documentation source is detected by Source File, Relative File and Homepage, we prioritize the detection of Source File as it is the highest in the following priority list:

1. Source File
2. Document
3. Relative File
4. Blog
5. Forum
6. Mailing List
7. Community Platform
8. Wiki
9. Repository-Related
10. Homepage
11. Uncategorized

We keep track of some specific categories that cause conflicts while, if none of them successfully detected the source, we mark the source as uncategorized. We also include categories that are not top-level. We do this as they may conflict between each other due to similar nature (*e.g.*, a source file referenced on the project's repository is both a relative file and a source file, and we wish to keep it as a source file).

## 4.2 Blog

A Blog, short for weblog, is an informational website that, typically, contains a series of posts in reverse chronological order<sup>1</sup>. In particular, blogs tend to have a diary-like structure with their content.

The Blog category has three subcategories: Custom Blog, Wordpress and Medium (Figure 4.2).

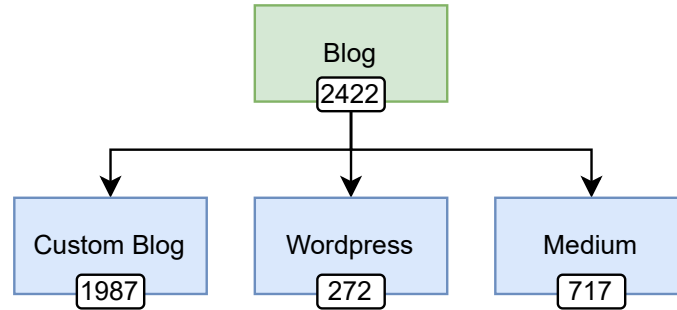


FIGURE 4.2: Taxonomy of the Blog Category

### 4.2.1 Medium

A blog that is hosted under the Medium domain<sup>2</sup>. Medium is an American online publishing platform developed by Evan Williams. It launched in August 2012, and is regularly regarded as a blog host<sup>3</sup>.

**Detection Method:** Contains the substring `medium`.

### 4.2.2 Wordpress

A blog that is hosted under the Wordpress domain<sup>4</sup>. WordPress is a web content management system. It was originally created as a tool to publish blogs<sup>5</sup>.

**Detection Method:** Contains any of the following substrings: `wordpress`, `wp.me`, `wp.com`.

### 4.2.3 Custom Blog

A blog that is either self-hosted or that exists under its own specific domain that does not fall under any other categories.

**Detection Method:** Contains the substring `blog`.

<sup>1</sup><https://en.wikipedia.org/wiki/Blog>

<sup>2</sup><https://medium.com/>

<sup>3</sup>[https://en.wikipedia.org/wiki/Medium\\_\(website\)](https://en.wikipedia.org/wiki/Medium_(website))

<sup>4</sup><https://wordpress.com/>

<sup>5</sup><https://en.wikipedia.org/wiki/WordPress>

## 4.3 Forum

A Forum is similar to community platforms in the way that most of the content is typically crowd-sourced, with users being able to create a thread of discussion around a given topic. It distinguishes itself as its own category due to the combination of crowd-sourced documentation and persistency of the documentation that is generated. The threaded structure of a forum along with the fact that threads persist across years are the key aspects of this category. We define three subcategories for Forum: Custom Forum, GitHub Discussions, StackOverflow (Figure 4.3).

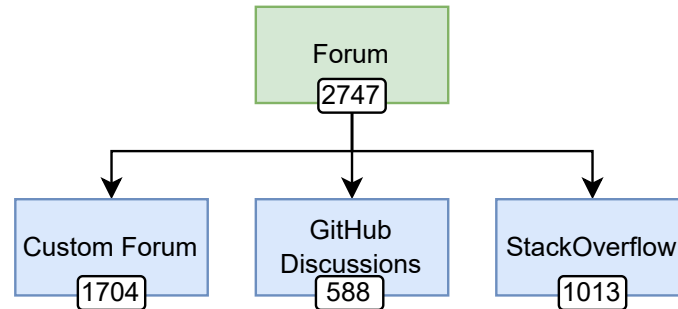


FIGURE 4.3: Taxonomy of the Forum Category

### 4.3.1 Custom Forum

A Custom Forum is a forum that has its own structure and that cannot be grouped under a more widespread forum platform.

**Detection Method:** Contains any of the following substrings: `forum`, `discuss` and does not satisfy GitHub Discussions.

### 4.3.2 GitHub Discussions

GitHub Discussions is a collaborative communication forum for the community around an open source or internal project<sup>6</sup>. We group here forums hosted on GitHub Discussions.

**Detection Method:** Contains `github.com` and `dicussions`.

### 4.3.3 StackOverflow

StackOverflow has a more Q&A approach to its threads, but it is still a forum. StackOverflow sources are grouped here.

**Detection Method:** Contains the substring `stackoverflow`.

<sup>6</sup><https://docs.github.com/en/discussions>



## 4.4 Mailing List

A mailing list is a computer-based communication system that distributes emails to a predefined group of recipients. It operates as a centralized platform, where messages submitted by authorized users are replicated and delivered to each subscriber on the list. Its main aspect is a collaborative interaction leading to persistent crowd-sourced documentation. Mailing List has three subcategories: Custom Mailing List, Google Groups and Mailman (Figure 4.4).

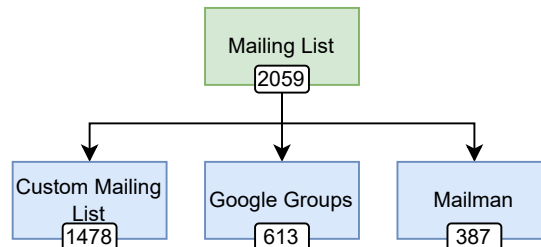


FIGURE 4.4: Taxonomy of the Mailing List Category

### 4.4.1 Custom Mailing List

A source that references a mailing list that is not on any widespread mailing list platform.

**Detection Method:** Contains the substring any of [list, mailing, mail and does not satisfy Google Groups or Mailman.

### 4.4.2 Google Groups

A source that references Google Groups<sup>7</sup>. Google Groups is a service from Google that provides discussion groups for people sharing common interests<sup>8</sup>.

**Detection Method:** Contains the substring `groups.google`.

### 4.4.3 Mailman

A source that references Mailman<sup>9</sup>. Mailman is free software for managing electronic mail discussion and e-newsletter lists<sup>10</sup>.

**Detection Method:** Contains the substring `mailman`.

<sup>7</sup><https://groups.google.com>

<sup>8</sup>[https://en.wikipedia.org/wiki/Google\\_Groups](https://en.wikipedia.org/wiki/Google_Groups)

<sup>9</sup><https://list.org/>

<sup>10</sup><https://list.org/>

## 4.5 Wiki

A Wiki is a website that allows collaborative modification of its content and structure directly from the web browser. The key feature of wikis that distinguishes it from other sources is the ability for any user to edit the content. Sometimes the user may be requested to identify themselves. This makes a wiki the product of a collaborative effort from the community rather than a more authoritative source curated by few. Under the Wiki category, we identify three subcategories: Custom Wiki, GitHub Wiki, Wikipedia (Figure 4.5).

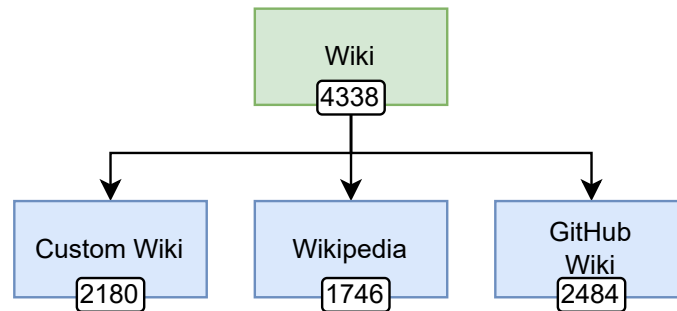


FIGURE 4.5: Taxonomy of the Wiki Category

### 4.5.1 Custom Wiki

A source that references a wiki that is not on any widespread wiki platform.

**Detection Method:** Contains the substring `wiki` and does not satisfy Wikipedia or GitHub Wiki.

### 4.5.2 GitHub Wiki

A source that references a wiki on GitHub.

**Detection Method:** Contains both `github.com` and `wiki`.

### 4.5.3 Wikipedia

A source that references Wikipedia<sup>11</sup>. Wikipedia is a free-content online encyclopedia written and maintained by a community of volunteers<sup>12</sup>.

**Detection Method:** Contains the substring `wikipedia`.

<sup>11</sup><https://en.wikipedia.org/>

<sup>12</sup><https://en.wikipedia.org/wiki/Wikipedia>

## 4.6 Document

With Document, we encapsulate files that are meant to be human-readable or human-understandable. This ranges from simple text files to more complex formats such as specific markup files, PDFs, or multimedia documents.

Document has two main subcategories: Textual Document and Multimedia Document (Figure 4.6).

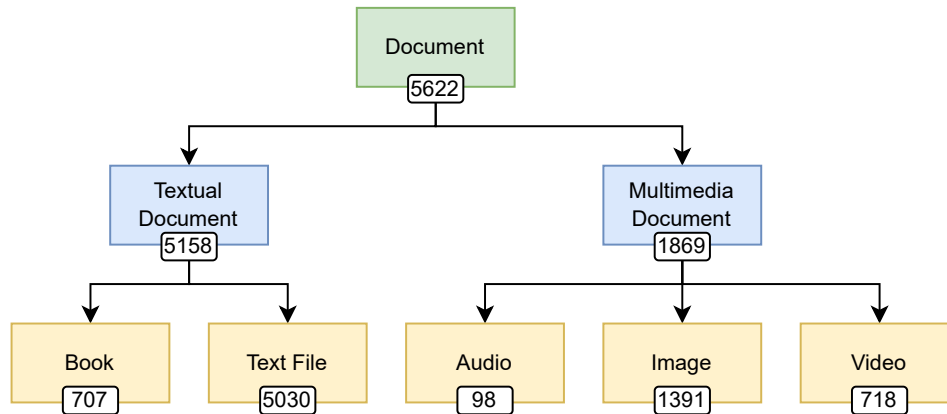


FIGURE 4.6: Taxonomy of the Document Category

### 4.6.1 Multimedia Document

A Multimedia Document involves typical multimedia such as audio files, images or videos. The category has three children.

#### Audio

An audio file.

**Detection Method:** Ends with any of the following file extensions: mp3, wav, wma, aac, flac, alac, ogg, aiff, dsd, pcm, aif, mp2, m4a, m4b, m4p, m4r, mid, midi, mka, mpa, ra, ram, tta, wv, wvp, 3gp, aa, aax, act, aifc, amr, ape, awb, dct, dss, dvf, gsm, iklax, ivs, m3u, m3u8, m4r, mmf, mpc, msv, oga, opus, ra, rm, sln, vox, w64, wma, wv, xspf, 8svx

#### Image

An image file.

**Detection Method:** Ends with any of the following file extensions: png, jpg, jpeg, gif, bmp, svg, tiff, tif, eps, raw, cr2, nef, orf, sr2, webp, heif, heic

#### Video

A video file or a link to a specific video on a platform.

**Detection Method:** The source must match detection for YouTube and must contain watch?v= as the video format for YouTube videos, or it must match Vimeo and must contain a video ID according to the regular expression d+, or it must end with any of the following file extensions: mp4, mov, avi, wmv, flv, mkv, webm, m4v, mpeg, mpg.

## 4.6.2 Textual Document

As previously described, textual documents include text files with any degree of complexity to them, so long as they are human-readable. We split a Textual Document in two main categories.

### Book

A source that references a physical book. While a physical book cannot be directly accessed via a URL, a website where a book can be purchased allows access to the book.

**Detection Method:** Ends with any of the following: [.epub, .mobi] or contains any of the following substrings: amazon, barnesandnoble, kobo, goodreads, abebooks, bookdepository, booktopia, indiebound, alibris, thriftbooks, booksamillion, bookshop, bookfinder, bookish, bookbub, bookriot, bookpage, bookforum, bookreporter, bookbrowse.

### Text File

A source that references a text file.

**Detection Method:** Does not contain any of the following words: index, home, main, default, welcome, landing, start, base, front and ends with any of the following extensions: txt, md, rst, xml, json, yaml, yml, csv, tsv, tex, rtf, doc, docx, odt, pdf, djvu, fb2, xps, cbz, cbr, cb7, cbt, cba, chm, pdb, prc, azw, azw3, lit, ps, pml, htmlz, txtz, rtfz, pdfz. We exclude names that can reference the homepage of a website to avoid homepages that are not actually text files (*e.g.*, index.html).

## 4.7 Homepage

With Homepage we describe a documentation source that points to the home page of a project-related website. In particular, we split Homepage between Project Homepage and Third-Party Project Homepage (Figure 4.7). A Project Homepage and its third-party counterpart represent the same type of source in terms of content and structure. The difference between one or the other is their relevance to the project they are referenced by. As a practical example, the `scikit-learn` project referencing their own personal website <https://scikit-learn.org/> would fall into the Project Homepage category. If this same website were referenced by another project that makes use of `scikit-learn`, this would fall under the Third-Party Project Homepage category.

This category of the documentation landscape is characterized by the lack of crowd-sourced content. A homepage is typically maintained and curated by its owners, while a user can only consult the source without being able to participate in its generation.

Both subcategories of Homepage have a shared common child: the Specific Subsection category. We specify this as the homepage of a website is not always directly referenced and, instead, the URL of the project website points to a location at a deeper path. These are specific subsections under the homepage domain.

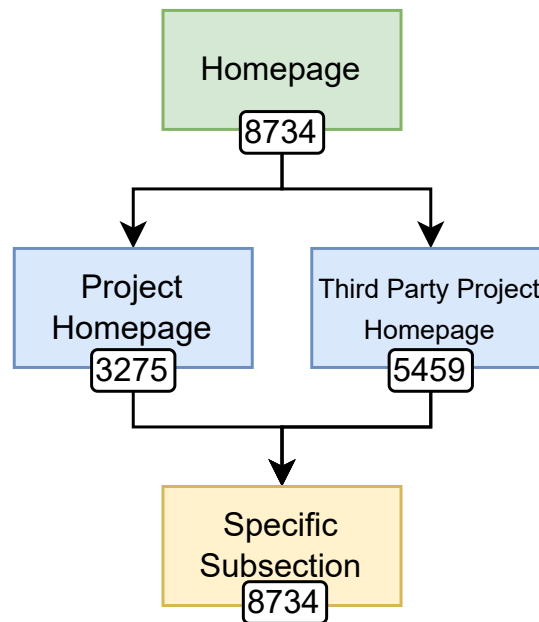


FIGURE 4.7: Taxonomy of the Homepage Category

### 4.7.1 Homepage Detection

Detecting homepages requires more complex steps compared to other sources.

#### Project Homepage

Given the name of the GitHub project it belongs to in the form of `<owner>/<name>`, contains either `<owner>` or `<name>`.

### Third-Party Project Homepage

Given the name of the GitHub project it belongs to in the form of <owner>/<name>, contains neither of them and contains any top-level domain from Internet Assigned Numbers Authority (IANA)<sup>13</sup>.

### Specific Subsection

Specific Subsection is trickier to detect. First, we remove `https://` or `http://` from the URL if it exists. Then, we identify three cases:

1. In this case, we have one or less occurrences of the `/` character. If there are no `/` characters in the URL, then there is no deeper path that is referenced. A specific subsection must at least have one occurrence of `/`, and the part after the first `/` must be non-empty. For instance:
  - `example.com`: Not a specific subsection
  - `example.com/`: Not a specific subsection
  - `example.com/<some content here>`: Potential to be a specific subsection
2. In this case, we can have between one and two `/` characters in the URL, and the part following the second occurrence of `/` must be empty. Here, we also take into consideration some common words that can appear in the URL of a homepage: `index`, `home`, `main`, `default`, `welcome`, `landing`, `start`, `root`, `base`, `front`. We do this because, at times, a homepage may take the following shape: `example.com/index.html/`. While this has a deeper section, it indicates a homepage. Therefore, a URL that has between one and two occurrences of the `/` character is a specific subsection if all of the following conditions are satisfied:
  - when there is only one occurrence of `/`, the part of the string following the `/` is non-empty (e.g., `example.com/<some content here>`)
  - when there are two `/` characters, it ends with the second occurrence of `/`
  - the content following the first occurrence of `/` does not contain any of the aforementioned “homepage words”
3. In this case, we have more than two occurrences of `/` or the URL contains a `#` character. A large number of slashes indicates a deeper path and the `#` character indicates a specific section referenced typically in HTML or Markdown documents.

---

<sup>13</sup><https://data.iana.org/TLD/tlds-alpha-by-domain.txt>

## 4.8 Repository-Related

Repository-Related is a category for documentation sources that document internal project aspects. It has six subcategories: Bug Tracker, Issue Tracker, Pull Request, Relative File, Repository, Source File (Figure 4.8).

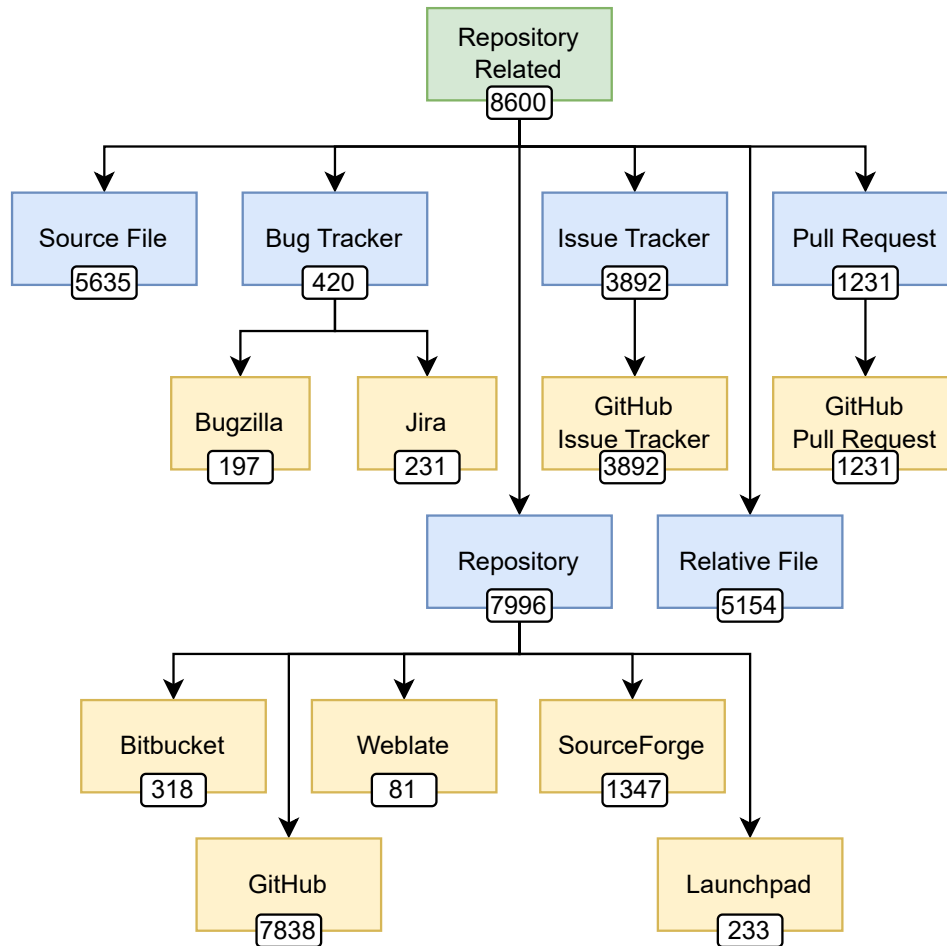


FIGURE 4.8: Taxonomy of the Repository-Related Category

### 4.8.1 Bug Tracker

A Bug Tracker is a tool that documents bugs by allowing developers to report them during the development of a project. Because of this, it may be regarded as a less generic type of Issue Tracker, as it pre-dates issue tracking systems. We identify two subcategories for it.

#### Bugzilla

A source that references Bugzilla<sup>14</sup>. Bugzilla is a web-based general-purpose bug tracking system and testing tool<sup>15</sup>.

**Detection Method:** Contains the substring bugzilla.

<sup>14</sup><https://www.bugzilla.org/>

<sup>15</sup><https://en.wikipedia.org/wiki/Bugzilla>

## Jira

A source that references Jira<sup>16</sup>. Jira is a proprietary issue tracking product developed by Atlassian that allows bug tracking<sup>17</sup>.

**Detection Method:** Contains the substring `jira`.

### 4.8.2 Issue Tracker

An Issue Tracker is a tool that documents “issues”. Issues are typically opened by developers and users to notify project maintainers of problems or bugs, to request features, and more useful project-specific aspects. Under Issue Tracker, we specify the GitHub Issue Tracker as its only subcategory. We detect a GitHub Issue Tracker documentation source if the URL contains both `github.com` and `issues` substrings.

### 4.8.3 Pull Request

From the GitHub documentation<sup>18</sup>:

*“Pull requests let you tell others about changes you’ve pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.”*

Under Pull Request, we include only the subcategory of GitHub Pull Request, which documents the contributions that are made by developers.

**Detection Method:** Contains both `github.com` and `pull` substrings.

### 4.8.4 Relative File

A Relative File is a type of file that is referenced via an existing relative path on the project repository that we cannot identify to be a Source File or a Textual Document.

**Detection Method:** Contains substrings `github` and `/blob/` and does not satisfy Source File or Textual Document. We check for those substrings because a relative path in a README file is not a full URL (e.g., `../../file.file`). We must reconstruct this URL as GitHub represents specific files at specific commits, and we check if this file exists in the reconstructed URL. We see this part in depth in Section 5.2.1.

### 4.8.5 Repository

A Repository is a documentation source that holds the source code of software projects. We identify five subcategories of Repository with the following sources.

#### Bitbucket

A source that references Bitbucket<sup>19</sup>. Bitbucket is a Git-based source code repository hosting service owned by Atlassian<sup>20</sup>.

<sup>16</sup><https://www.atlassian.com/software/jira>

<sup>17</sup>[https://en.wikipedia.org/wiki/Jira\\_\(software\)](https://en.wikipedia.org/wiki/Jira_(software))

<sup>18</sup><https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests>

<sup>19</sup><https://bitbucket.org/product>

<sup>20</sup><https://en.wikipedia.org/wiki/Bitbucket>



**Detection Method:** Contains the substring `bitbucket`.

### GitHub

A source that references the GitHub platform<sup>21</sup>. GitHub is a platform and cloud-based service for software development and version control using Git<sup>22</sup>.

**Detection Method:** Contains the substring `github.com` and does not detect any of GitHub Discussions, GitHub Wiki, Relative File.

### Launchpad

A source that references Launchpad<sup>23</sup>. Launchpad is a web application and website that allows users to develop and maintain software<sup>24</sup>.

**Detection Method:** Contains the substring `launchpad.net`.

### SourceForge

A source that references SourceForge<sup>25</sup>. SourceForge is a web service that offers software consumers a centralized online location to control and manage open-source software projects<sup>26</sup>.

**Detection Method:** Contains the substring `sourceforge.net`.

### Weblate

A source that references Weblate<sup>27</sup>. Weblate is an open source web-based translation tool with version control<sup>28</sup>.

**Detection Method:** Contains the substring `weblate.org`.

## 4.8.6 Source File

A Source File is a file that contains source code and it separates itself from a simple textual document. Sometimes, when a project is in development, official documentation may not have been generated yet. In this case, a source file may be referenced instead for further information regarding the project and some of its features. An example of this is the `javalang` framework<sup>29</sup> on GitHub. Its README states the following:

*For more detail on what node types are available, see the `javalang/tree.py` source file until the documentation is complete.*

A source file that is referenced in a README file represents a documentation source that is strictly related to the repository<sup>30</sup>.

---

<sup>21</sup><https://github.com/>

<sup>22</sup><https://en.wikipedia.org/wiki/GitHub>

<sup>23</sup><https://launchpad.net/>

<sup>24</sup>[https://en.wikipedia.org/wiki/Launchpad\\_\(website\)](https://en.wikipedia.org/wiki/Launchpad_(website))

<sup>25</sup><https://sourceforge.net/>

<sup>26</sup><https://en.wikipedia.org/wiki/SourceForge>

<sup>27</sup><https://weblate.org/en-gb/>

<sup>28</sup><https://en.wikipedia.org/wiki/Weblate>

<sup>30</sup><https://github.com/c2nes/javalang>

**Detection Method:** Ends with any of [py, java, c, cpp, cs, js, ts, php, css, scss, less, xml, json, yaml, cc, h, hpp, hxx, hh, yml, sh, bat, cmd, ps1, psm1, psd1, ps1xml, pssc, vbs, vba, vb, bas, frm, cls, ctl, xsl, xslt, xsd, wsf, wsc, wsh, ini, inf, reg, cfg, config, conf, properties, prop, props, sln, csproj, vbproj, vcxproj, vcproj, vcproj, xcodeproj, dproj, cbproj, pbxproj, pbproj, xib, storyboard, plist, nib].

## 4.9 Community Platform

A Community Platform encompasses multiple platform types that share the way that documentation is generated and consumed on them. Documentation generation and consumption on a community platform tends to be strongly crowd-sourced and very volatile. The key point is that the user is the one who provides documentation. We split Community Platform into four subcategories: Custom Community Platform, Instant Messaging, Media Sharing, and Social Media (Figure 4.9).

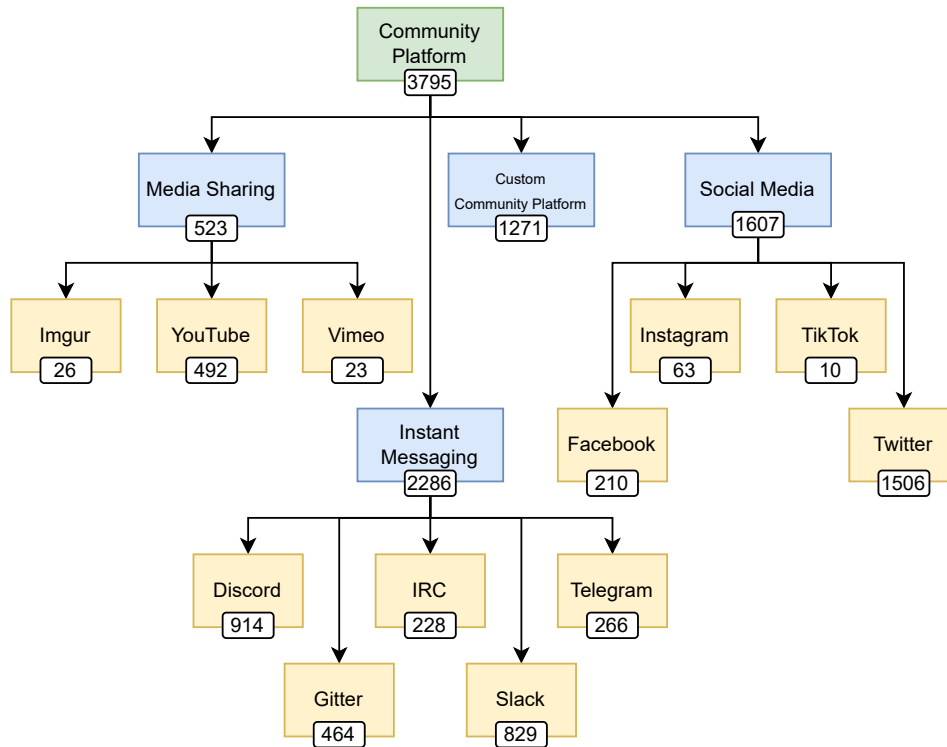


FIGURE 4.9: Taxonomy of the Community Platform Category

### 4.9.1 Custom Community Platform

A Custom Community Platform is a platform that does not fall under any of the other subcategories that still satisfies the requirements of a community platform. For instance, a project may have its own custom community portal with its own login page that does not adhere to any other more widespread platform.

**Detection Method:** Contains the word `community`.

### 4.9.2 Instant Messaging

Instant Messaging is a form of online communication that offers real-time text transmission over the internet. The user of an instant messaging platform plays a crucial role in generating documentation that is highly volatile.

Instant messaging provides a way to quickly ask and answer questions without any structure, and it can switch from one topic to another in a matter of seconds. Furthermore, due to the structure of instant messaging platforms (*i.e.*, messages being sorted from most recent to oldest, only recent messages are easily shown), newer discussions can bury old conversations.

An Instant Messaging platform has five subcategories: Discord, Gitter, IRC, Slack, and Telegram.

## Discord

A source that references the Discord application<sup>31</sup>. Discord is an instant messaging and VoIP social platform<sup>32</sup>.

**Detection Method:** Begins with any of [<https://discord.gg>, <https://discord.com/invite>] or contains the word discord.

## Gitter

A source that references the Gitter application<sup>33</sup>. Gitter is an open-source instant messaging and chat room system for developers and users of GitLab and GitHub repositories<sup>34</sup>.

**Detection Method:** Begins with <https://gitter.im> or contains the word gitter.

## IRC

A reference to an Internet Relay Chat (IRC) server or channel. IRC is a text-based chat system for instant messaging<sup>35</sup>.

**Detection Method:** Contains any of the following whole words: irc, oftc, freenode.

## Slack

A source that references the Slack application<sup>36</sup>. Slack is a cloud-based freemium cross-platform instant messaging service created by Slack Technologies and currently owned by Salesforce<sup>37</sup>.

**Detection Method:** Contains the substring slack.

## Telegram

A source that references the Telegram application<sup>38</sup>. Telegram Messenger is a globally accessible freemium, cloud-based and centralized instant messaging (IM) service<sup>39</sup>.

**Detection Method:** Contains any of the following substrings: telegram, t.me.

### 4.9.3 Media Sharing

Media-sharing platforms are a place where a user mainly shares media such as videos or images. Crowd-sourced documentation is prevalent in this category. We identify three subcategories: Imgur, Vimeo, and YouTube.

---

<sup>31</sup><https://discord.com/>

<sup>32</sup><https://en.wikipedia.org/wiki/Discord>

<sup>33</sup><https://gitter.im/>

<sup>34</sup><https://en.wikipedia.org/wiki/Gitter>

<sup>35</sup>[https://en.wikipedia.org/wiki/Internet\\_Relay\\_Chat](https://en.wikipedia.org/wiki/Internet_Relay_Chat)

<sup>36</sup><https://slack.com/>

<sup>37</sup>[https://en.wikipedia.org/wiki/Slack\\_\(software\)](https://en.wikipedia.org/wiki/Slack_(software))

<sup>38</sup><https://telegram.org/>

<sup>39</sup>[https://en.wikipedia.org/wiki/Telegram\\_\(software\)](https://en.wikipedia.org/wiki/Telegram_(software))

## Imgur

A source that references Imgur<sup>40</sup>. Imgur is an American online image sharing and image hosting service<sup>41</sup>.

**Detection Method:** Contains the substring `imgur`.

## Vimeo

A source that references Vimeo<sup>42</sup>. Vimeo, Inc. is an American video hosting, sharing, and services platform provider<sup>43</sup>.

**Detection Method:** Contains the substring `vimeo`.

## YouTube

A source that references YouTube<sup>44</sup>. YouTube is an American online video sharing and social media platform<sup>45</sup>.

**Detection Method:** Contains any of the following substrings: `youtube`, `youtu.be`.

### 4.9.4 Social Media

Social media are websites and applications that enable users to create and share content of different types (e.g., written posts, images, videos), or to participate in social networking with a community. The key feature of social media is the ability to share content of various types with a large audience. This is in contrast with other sources that are more focused either on the discussion aspect (e.g., Forum) or on specific types of content (e.g., YouTube). Under Social Media, we define the four subcategories.

#### Facebook

A source that references Facebook<sup>46</sup>. Facebook is an online social media and social networking service<sup>47</sup>.

**Detection Method:** Remove `https://` or `http://` from the URL, take the top level domain of the URL. Top level domain must contain the substring `facebook`.

#### Instagram

A source that references Instagram<sup>48</sup>. Instagram is a photo and video sharing social networking service<sup>49</sup>.

**Detection Method:** Contains the substring `instagram`.

---

<sup>40</sup><https://imgur.com/>

<sup>41</sup><https://en.wikipedia.org/wiki/Imgur>

<sup>42</sup><https://vimeo.com/>

<sup>43</sup><https://en.wikipedia.org/wiki/Vimeo>

<sup>44</sup><https://www.youtube.com/>

<sup>45</sup><https://en.wikipedia.org/wiki/YouTube>

<sup>46</sup><https://www.facebook.com/>

<sup>47</sup><https://en.wikipedia.org/wiki/Facebook>

<sup>48</sup><https://www.instagram.com/>

<sup>49</sup><https://en.wikipedia.org/wiki/Instagram>

## TikTok

A source that references TikTok<sup>50</sup>. TikTok is a short-form video hosting service<sup>51</sup>.

**Detection Method:** Contains the substring `tiktok`.

## Twitter

A source that references Twitter<sup>52</sup>. Twitter, currently rebranding to X, is an online social media and social networking service<sup>53</sup>.

**Detection Method:** Contains the substring `twitter`.

## 4.10 Conclusion

In this chapter, we presented all the categories of our taxonomy (Figure 4.10). We aimed to keep the categories specific enough to be able to capture platforms that distinguish themselves in their attributes while also keeping them as broad as possible. We tried to strike a balance between specific instances of platforms (*e.g.*, Discord) and general categories (*e.g.*, Blog).

With the taxonomy defined, we see how it is implemented in RagnaDok to view the documentation landscape of open-source projects and its evolution.

---

<sup>50</sup><https://www.tiktok.com/en/>

<sup>51</sup><https://en.wikipedia.org/wiki/TikTok>

<sup>52</sup><https://twitter.com/>

<sup>53</sup><https://en.wikipedia.org/wiki/Twitter>

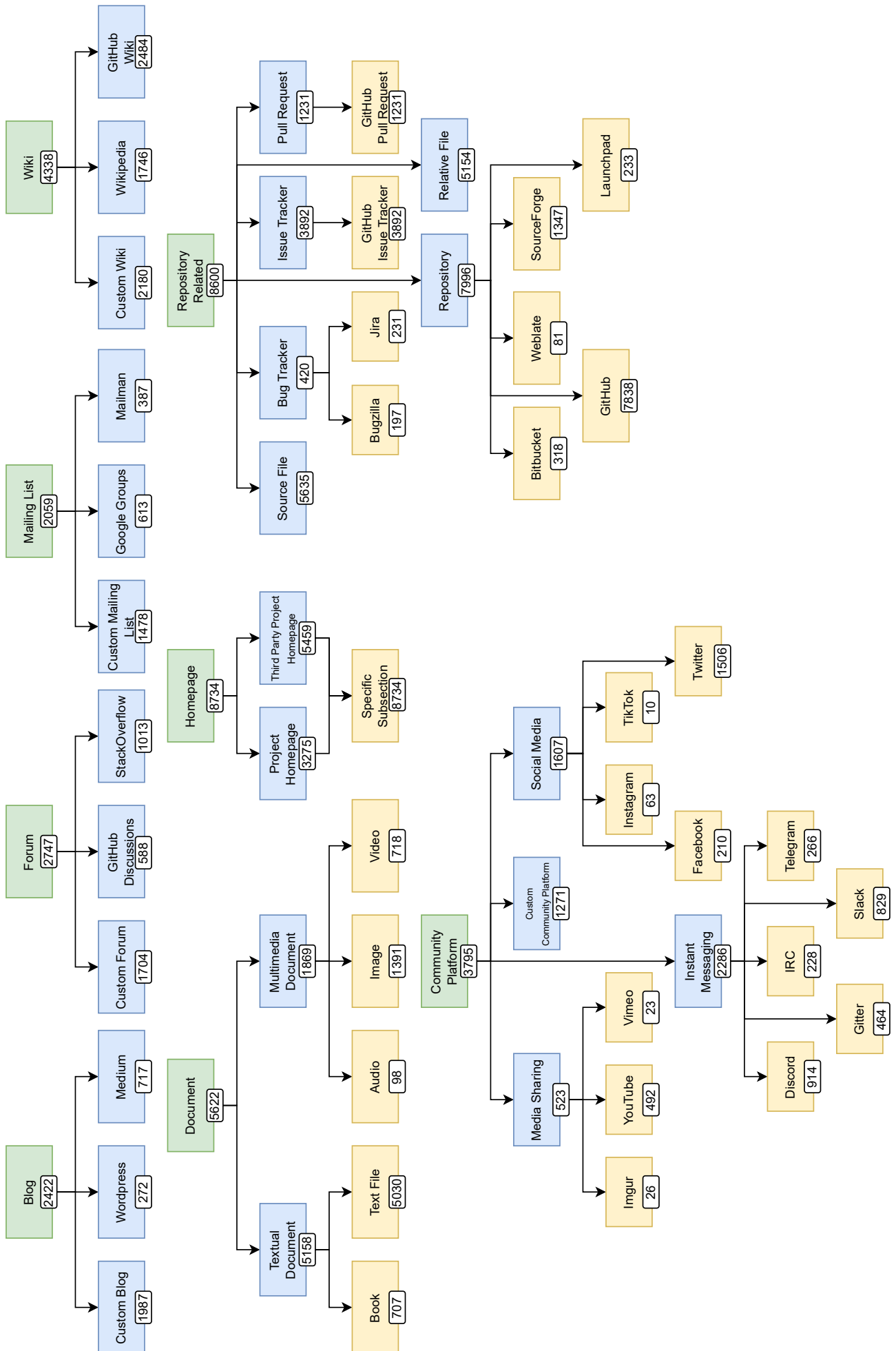


FIGURE 4.10: Taxonomy of the Documentation Landscape





## Chapter 5

# RagnaDok Implementation

In this chapter, we present the architecture and implementation of RagnaDok, a tool to visualize the documentation landscape based on the taxonomy presented in the previous chapter.

### 5.1 System Architecture

RagnaDok is a web-based application built using the React.js library for its frontend and Flask for its Python backend. It is currently deployed at <https://ragnadok.si.usi.ch/>.

The system is split into two main parts (Figure 5.1), the backend and the frontend. It depends on GitHub externally to mine data. The backend takes care of mining data, processing it and sending the results to the frontend. The frontend requests the data from the backend and displays it in various visualizations.

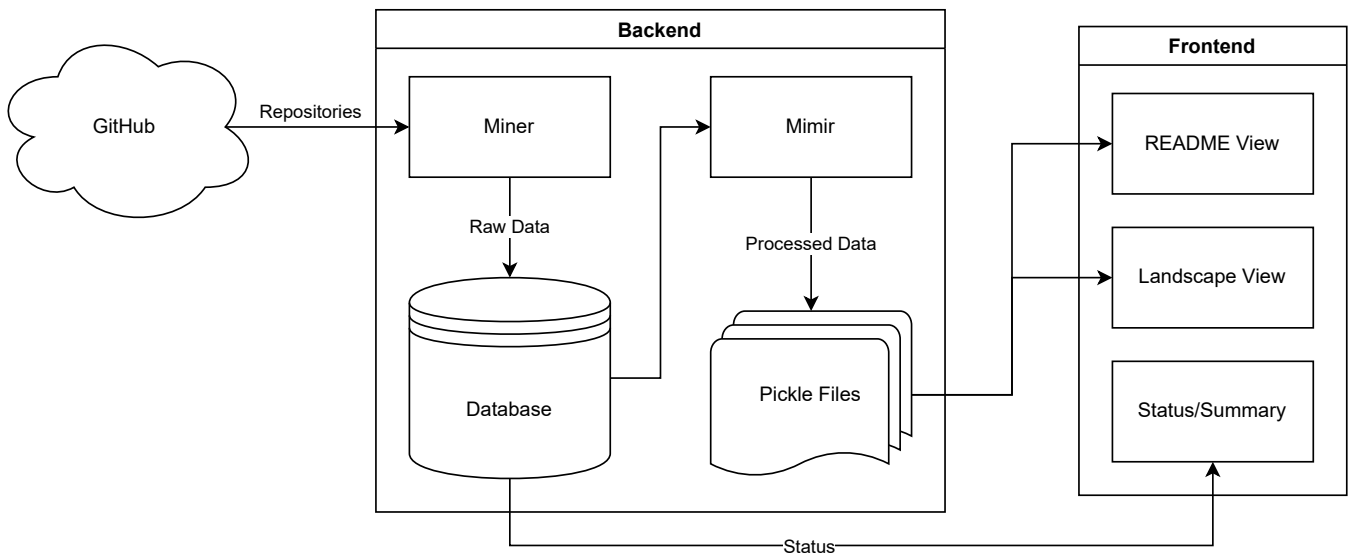


FIGURE 5.1: Architecture of RagnaDok

The mining process is split into two parts. Raw, unprocessed data is written to a SQLite database, which is also used as a synchronizing point for the status of the mining. Then Mimir, the historian, receives the raw data and processes it, writing it to a pickle file to store itself and all of its features in a binary file. The pickle files are then unpickled when their specific repository is requested by the frontend and they supply the requested data to its view.

## 5.2 Data and History Mining

The main actions that Git tracks in a commit are: Add, Modify, Delete, Rename, Move (see Section 3.2.1).

The miner begins by acquiring the software repository that we wish to examine. To get access to the repository we can choose to use the GitHub API or clone the repository on our machine. Between these two approaches, we choose cloning the repository locally. This is because the GitHub API has a limit of 5000 requests per hour and our objective is to mine multiple repositories concurrently. In an exploratory study of the thesis, we found that large projects could require a large number of requests by themselves and this would not scale. With a fresh clone, we can extract information regarding all the actions that were performed on all files without any requests at all.

### Using the Log

Git maintains a log of the commits performed on a repository. We run the `git log` command with some custom flags in order to have a formatted output of the information about commits that we can parse. The flags that we employ are the following:

- `-C [path]`: takes a path argument and tells Git to use that path to run its commands. We pass the absolute path to the cloned repository as the argument to ensure that the Git commands are executed in the correct directory. This flag is actually a flag for the `git` command and not for the `git log` command, so it must come before `log`.
- `-name-status`: displays the action performed on the files that were modified in a commit in the form of a letter at the start of the line. For instance, we get `A` for additions and `M` for modifications. With this option, renames are shown in the log with `R[N]` where `[N]` is a number between 0 and 100 representing the confidence that Git has that the file has been renamed rather than being a new file altogether.
- `-pretty=tformat:[format]`: allows us to format other information in the commit as we prefer, such as the date, the author, or the commit hash. The custom format that we pass in for our log is the following: `"CommitHash:%H%nDate:%ad%nCommitAuthor:%an%nCommitMessage:%B"` to display the hash of the commit, its date and its message.
- `-date=unix`: With the `-date=unix` option, we format the date as a Unix timestamp. This is to avoid inconsistencies that can happen due to time zones.

The output of this command may look like the following example:

```
CommitHash: 6eabfa27e7c380ed762f088092ee89d4bcc0b91a
Date: 1688314939
CommitAuthor: John Git
CommitMessage: add super cool feature

A       NewFile.java
R100    old_file.txt      new_file.txt
D       deleted_file.py
```

FIGURE 5.2: Example Output of Custom Git Log

## Filtering Commits

The custom formatted output of the `git log` command contains all files that have been affected by each commit. We can now filter all the existing commits based on whether they modified a README file. To do so, we need to define what classifies as a README file. We normalize the filename to lower case and use a heuristic-based approach to filtering. A file is a README if:

- its filename contains one of the following substrings as whole words in its filename: `readme`, `infotext`, `read me`, `read.me`, `read_me`, `read-me`. A whole word in our case is a substring that matches one of the given substrings that is not bounded by any other alphabetical character. For example, `this_is_a_readme.txt` is considered as a README file because the substring `readme` is a whole word bounded by underscore characters. Instead, `ReadMessage` will not be included as, while the substring `readme` is present, it is not a whole word.
- its filename does not have any of the following extensions: `.py`, `.java`, `.c`, `.cpp`, `.cc`, `.js`, `.jsx`, `.png`, `.jpg`, `.jpeg`, `.gif`, `.svg`, `.ico`, `.ino`. This is to avoid files that contain the substring, but that are not text files, at least according to their extensions.

Heuristic based approaches to README identification have clear limitations. Although the case of `ThisIsAReadme.txt` will not be captured by our heuristics, this should have a limited impact on our results. From manual inspection, a typical README file does not have a very complicated or long name. In such a case, we prefer to ignore complex filenames in favour of simpler and more accurate ones. When the heuristic is satisfied for any single affected file in a commit, we mark this commit as a README commit to check out during the next step of the mining. We always include the first and last commits made on a repository regardless of their modified files as they represent the start and end of the repository.

## Building Histories

With information on all the commits that are relevant to us, we can now begin to build the README histories of the project. We perform a `git checkout` at each commit that we identify and we check all the modified files. For each modified file, we check whether the heuristics for README files are satisfied. If they are, we create a new README history (or we update a previously existing one), and we append to it a new README version which contains all the relevant information for that README file and its commit.

For each commit, we use the `scc`<sup>1</sup> command line tool to count the lines of comments in the repository as source code comments could represent a type of documentation source. We compared `scc` to the more known `cloc`<sup>2</sup> with some performance tests (see Appendix A) and `scc` provided faster outputs with a negligible impact in accuracy.

In addition, we separately keep track of commits that involved the deletion or the rename of a README file. We use these to be able to determine when a history ends and to simplify history chaining (see Section 3.2.3).

Once the raw unprocessed history data is mined, we store it into an SQLite database and prepare to process it.

### 5.2.1 Post-Mining Processing

At this stage, the raw data has been saved to the database. We consider a project fully mined when its data has also been processed. So, to follow up on and complete the mining, we call upon our historian, Mimir. As described in Section 3.2.3, this class offers a number of features useful to process the data after it is mined. In particular, it chains the histories together and, having control over all histories and versions of a repository, it can use them to identify potential documentation sources and verify their status.

<sup>1</sup><https://github.com/boyter/scc>

<sup>2</sup><https://github.com/AlDanial/cloc>

## Finding Documentation Sources

We now want to find the documentation sources contained within the README versions that we have mined. After identifying them, we want to confirm their status. The status of a documentation source can be either live, dead, rejected or recovered. Where a live source represents an accessible URL, a dead source is a URL that does not respond, a rejected source is something that we identified but that failed to respond appropriately, and a recovered source is a rejected element that satisfied a broader recovery step.

With the access that Mimir has, it can process all versions across all histories. We now begin with processing the content of each version to identify potential documentation sources contained within. We achieve this by using different regular expressions to capture them. The focus is on capturing regular URLs, URLs described with an HTML format (*i.e.*, `<a href="URL">`), IRC nodes or channels, and email addresses.

We now further process these potential sources by going through each of them to see whether they are live sources, dead sources, or even not sources at all. In this step, we reject a source typically because one of the regular expressions could have over-captured parts of the content that do not provide any real source.

To control the status of the potential sources, we ping them. We do this first via a HEAD request as it is lighter and faster. If this fails for any reason, we retry with a GET request. Pinging these documentation sources is often not as straightforward. Sometimes URLs are left incomplete (*e.g.*, `example.com` rather than `https://example.com`) or they instead represent semantic relative paths within the README that become URLs when displayed in a specific mark-up language such as Markdown or reStructuredText (*e.g.*, `./CONTRIBUTING.MD`). Because of this, a GET request that fails does not mean that we should mark the source as dead or rejected right away.

We retry making a request by appending `http://` or `https://` to the URL, or by reconstructing the relative path via the commit hash. We can do this because GitHub has a clear URL pattern that we can use. GitHub shows a file from a specific commit with the following pattern:

```
https://github.com/<PROJECT OWNER>/<PROJECT NAME>/blob/<COMMIT HASH>/<PATH TO FILE>
```

We have access to all the required information to reconstruct the relative path URL. When the response status code is not a 404 Not Found, we can then say that the referenced file was a relative path in this README at this point in time.

Should every attempt that we made fail, we mark the source as rejected.

## Rejected Sources Recovery

After the pinging step, we make a second pass on the rejected source. Sometimes a source file is referenced without the proper relative path, but it could still be a relevant documentation source. The same concept is valid for other potential sources. We now check if we reject any source that is shaped like a source file or a URL that has one of the top level domain endings taken from IANA.<sup>3</sup>

With this final step, a documentation can be either live, dead, rejected or recovered.

### 5.2.2 History Chaining

We now have all the data we need to represent README files over time along with the documentation sources that they contain. The final step before the data is fully processed is history chaining. The chaining follows the steps that we have taken in Section 3.2.3. When it is complete, the Mimir that oversees the project is saved to a pickle file for re-use. In the case of projects with many histories, we postpone the chaining as it is one of the most expensive steps of the mining.

---

<sup>3</sup><https://data.iana.org/TLD/tlds-alpha-by-domain.txt>

## 5.3 Visualization

To explore the data that we mined and processed, we offer two different visualizations for README histories and for the documentation landscape. The former being a polymetric view of the content and documentation sources present in README histories over time, while the latter offers project-specific and aggregate views of the documentation landscape and the sources that compose it. A polymetric view is a lightweight software visualization technique enriched with software metrics information [25]. In our case, we enrich the view by adding metrics to the width and the height of rectangles.

### 5.3.1 README History Visualization

Given a GitHub project, we can visualize its histories, versions and documentation sources over time. It allows us to observe the evolution of the README files of a project. We present our README history visualization and highlight the main parts that compose it in Figure 5.3.

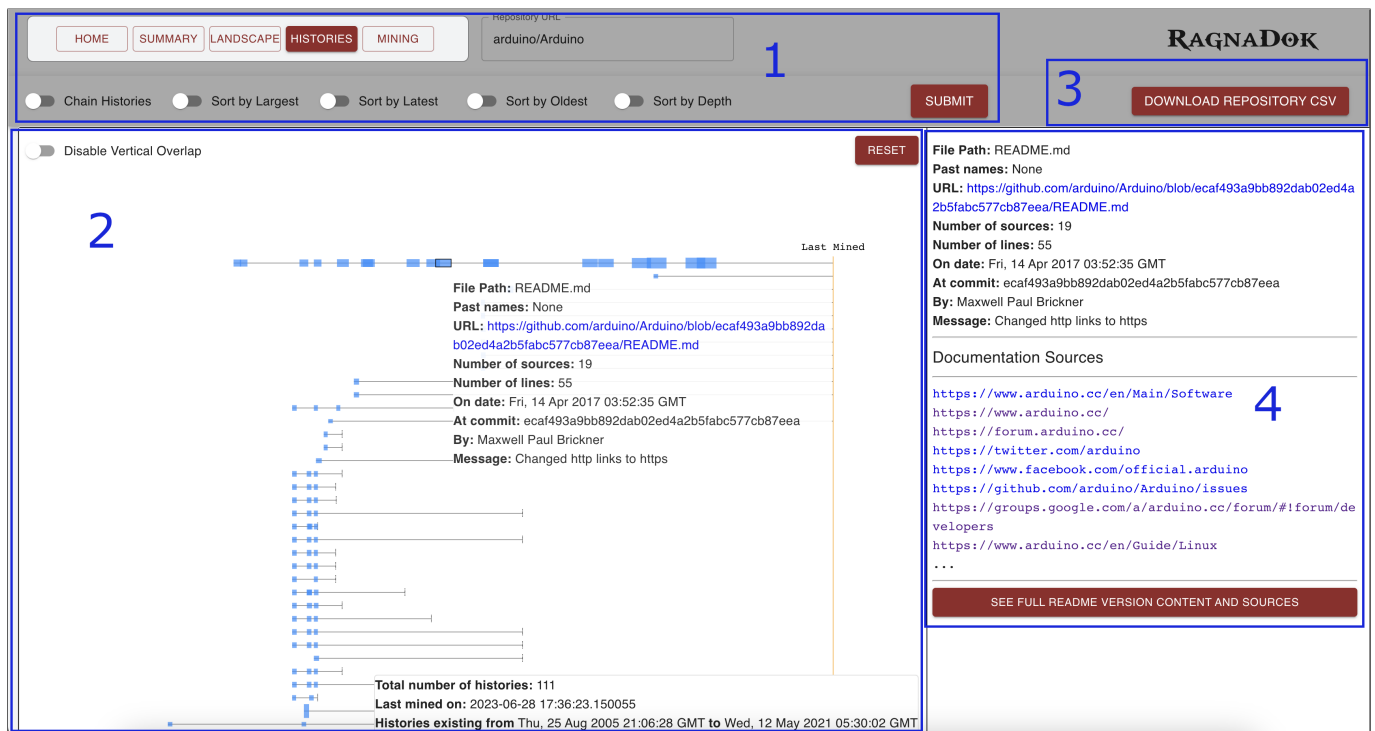


FIGURE 5.3: Visualization for README histories

These main components are:

1. **Option Bar:** It allows selection of repositories that have been analyzed and offers various options to the visualization such as sorting the histories or visualizing chained histories instead.
2. **View:** The view produces an SVG view of the histories. The view encapsulates the entire lifespan of a project by displaying each individual README version as a rectangle. The width and height of each rectangle indicate its number of documentation sources and its number of lines respectively. To be able to view smaller nodes with dimensions close to 0, we trade a more faithful view of the node size by giving them a minimum and maximum dimension that they can reach. Each rectangle can be hovered and selected to view insights about the version that it represents. Once selected, the view

that we see in point 4 can be seen for that specific version. Finally, on the bottom right corner, we display useful data to know how long a project has existed and when it was last updated.

- 3. **Repository CSV Button:** This button allows to download the documentation sources that belong to the latest version of the project. It calls a backend endpoint that we used to generate CSV files to analyze for a manual annotation of the documentation sources, as mentioned in Section 3.2.4. The manual annotation was an iterative and useful process to tweak the taxonomy of the documentation landscape.
- 4. **README Version Information:** This side-tab displays information regarding the version that has been selected and the first documentation sources that are present in the file. All the documentation sources that we find in this version, along with the file content, can be seen by pressing the button at the bottom.

Moving onto the version information tab, we can choose to view the content of the version and its sources. This leads to the version view (Figure 5.4).

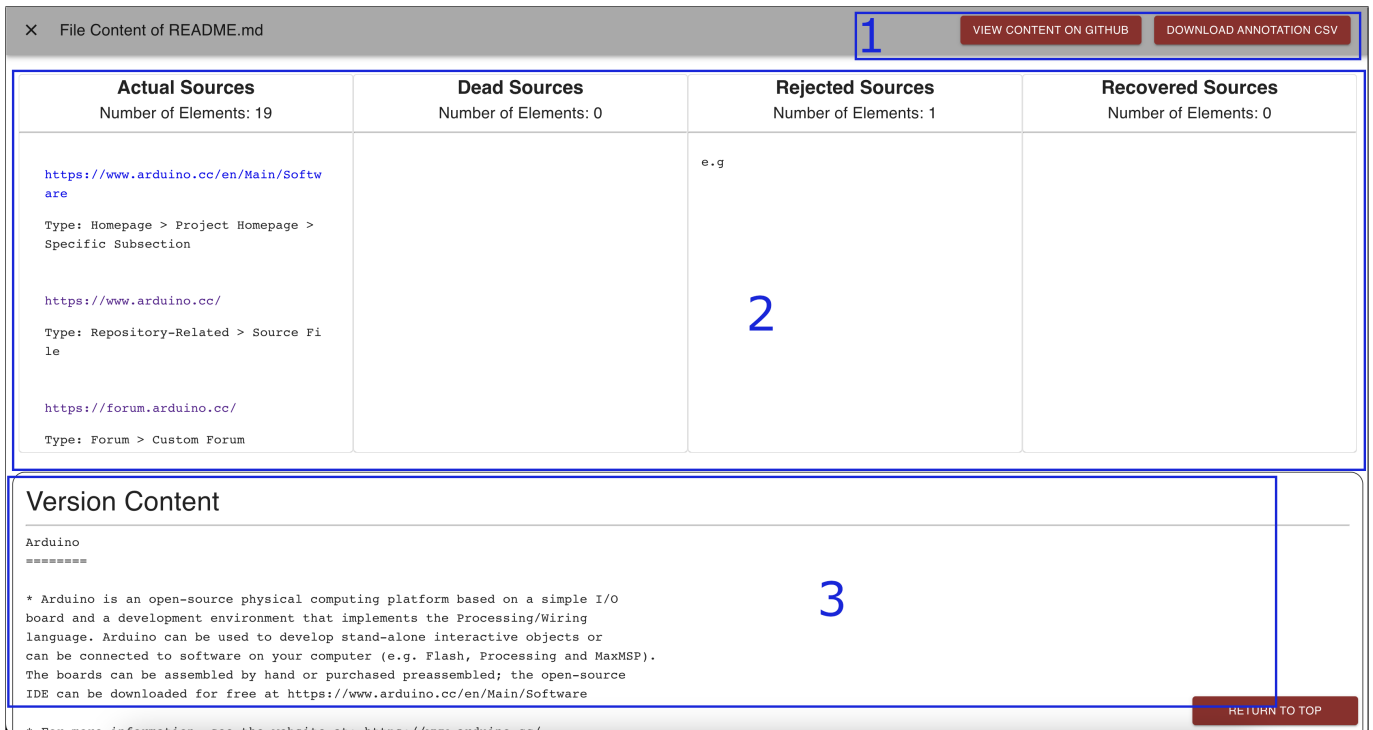


FIGURE 5.4: View for a Specific README Version

Here we can see its main components:

- 1. **Utility Buttons:** These buttons allow to view the README directly on GitHub and download another annotation CSV respectively. Much like the previous CSV file, this too contains documentation sources ready for manual annotation, with the only difference being that this isolates the version rather than the entire project.
- 2. **Documentation Source Tab:** Here we display all the documentation sources that we identify within this README version. They are separated by status as we described in Section 5.2.1, where "actual" means live sources.

- 
3. **Version Content:** While we can view the content on GitHub via one of the utility buttons, we also display the raw content of the README version directly on this page.

### 5.3.2 Documentation Landscape Visualization

To visualize the documentation landscape, we employ the taxonomy that we have defined in Chapter 4. Each category of the taxonomy represents a documentation source type that we map onto the visualization.

We achieve this visualization via a customized treemap with a fixed layout. We visualize the documentation landscape of a single project and the aggregate landscape across all the projects that have been mined and analyzed. They share the same layout with a few differences in the way their data is grouped and displayed. These differences come mainly from the coloring and bucketing of the data. Data bucketing, also known as data binning, is a data pre-processing technique used to reduce the effects of minor observation errors<sup>4</sup>. Data values that fall into a specific interval are replaced by a unique value that represents that interval. In our case, we keep the explicit interval and we bucket by color. We do this to capture the frequency of a documentation source category more easily given a range and because relying solely on opacity will yield possibly undistinguishable shades of colors for very similar values.

#### Documentation Landscape of a Project

For a single project, we visualize the landscape by counting the amount of documentation sources per category. We change color and opacity based on how large this amount is and we bucket it into separate ranges. Relying solely on color opacity to detect the presence of a documentation source can make it harder, especially for values that are close to each other. Because of this, we use buckets with ranges that clearly indicate how present a source is. The ranges are: 0, 1, 2 to 5, 6 to 10, 11 or more. We choose these ranges as, first and foremost, they clearly outline whether a source is present or absent in a project. Secondly, we choose 11 as the top bucket, as, beyond the intermediate ranges, the category is saturated. We present an example visualization in Figure 5.5.

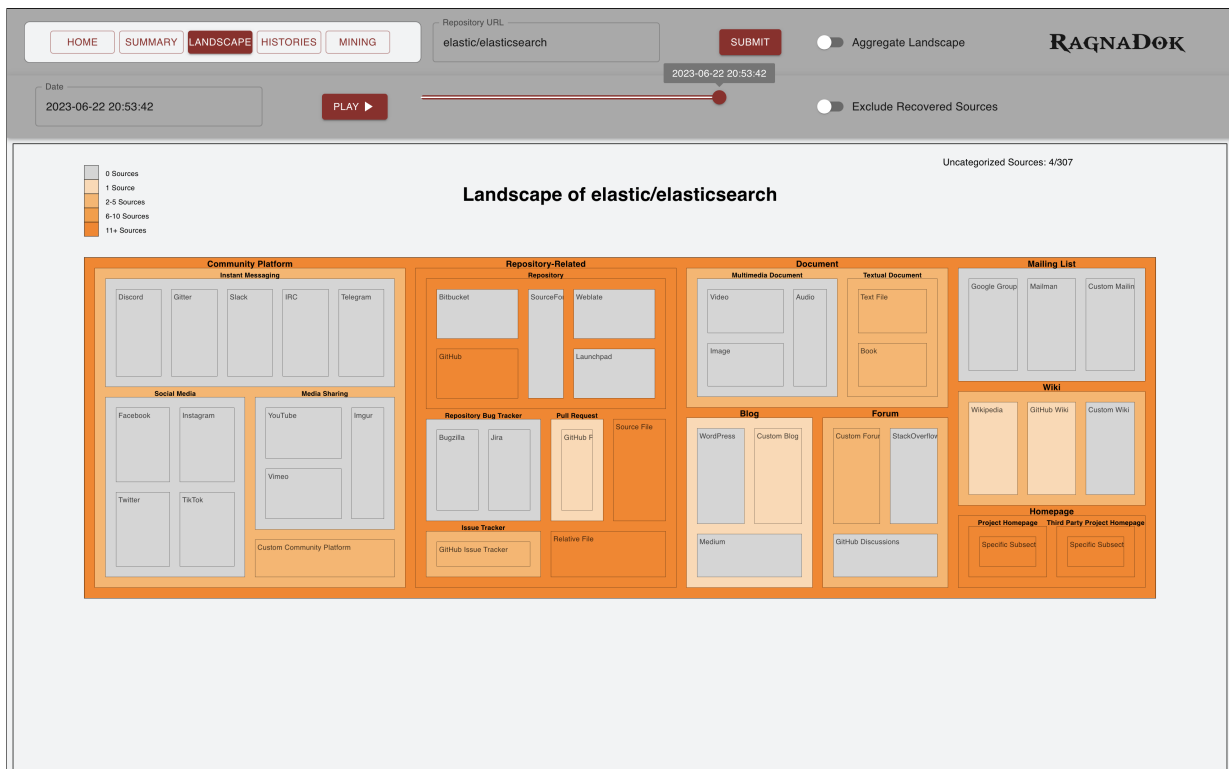


FIGURE 5.5: Documentation Landscape of the Elasticsearch Project

<sup>4</sup>[https://en.wikipedia.org/wiki/Data\\_binning](https://en.wikipedia.org/wiki/Data_binning)



The view is split into two main parts:

- **Option Bar:** The option bar offers various settings for the visualization. It allows to choose the project to visualize, or to visualize the aggregate landscape instead. It also allows to exclude recovered documentation sources, which can be source of noise under some circumstances. On top of it, the landscape can be viewed at any commit date that modified a README file in the repository. The date can be picked from a list in the top-left corner, or it can be selected with a slider. The Play button plays the evolution from start to finish with a short interval between each version.
- **Landscape View:** The landscape view displays the documentation landscape of the project that was selected. It displays the custom treemap with a legend that explains the colors and the number of documentation sources that we did not categorize within our taxonomy. A treemap is a space-filling method of visualizing large hierarchical data sets [22]. In the example visualization (Figure 5.5), we can see how the latest version of Elasticsearch<sup>5</sup> that we have mined contains many documentation sources in the GitHub, Source File, Relative File and Homepage categories. Other categories instead show with a lower number of occurrences.

### Aggregate Documentation Landscape

We visualize the aggregate documentation landscape by focusing on the overall presence of a documentation source among all projects (Figure 5.6). We bucket the data differently compared to the previous view and we skew this bucketing towards smaller percentages. We focus on capturing every 10-percentile until 50%, after which the documentation is so popular we focus on capturing whether it surpasses larger thresholds of 75% and 90%. Instead of using opacity, we use a white-to-black color scale to indicate absence and presence of a category respectively.

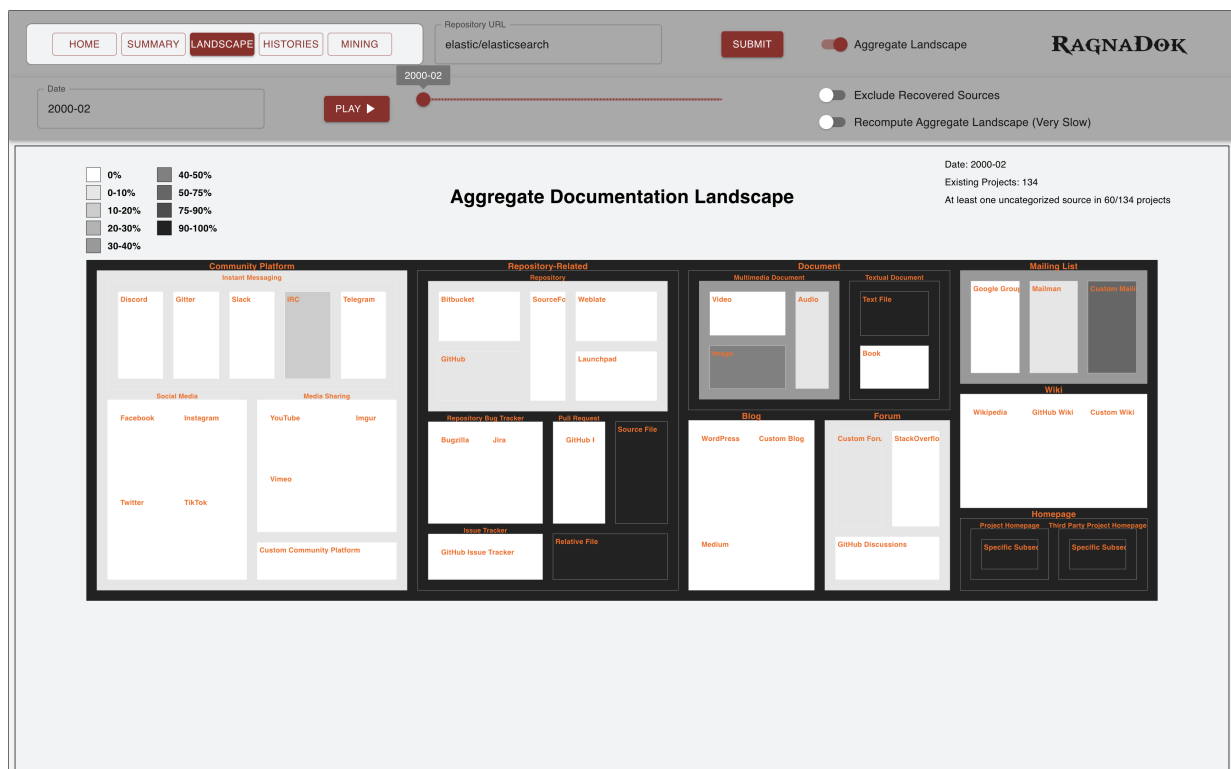


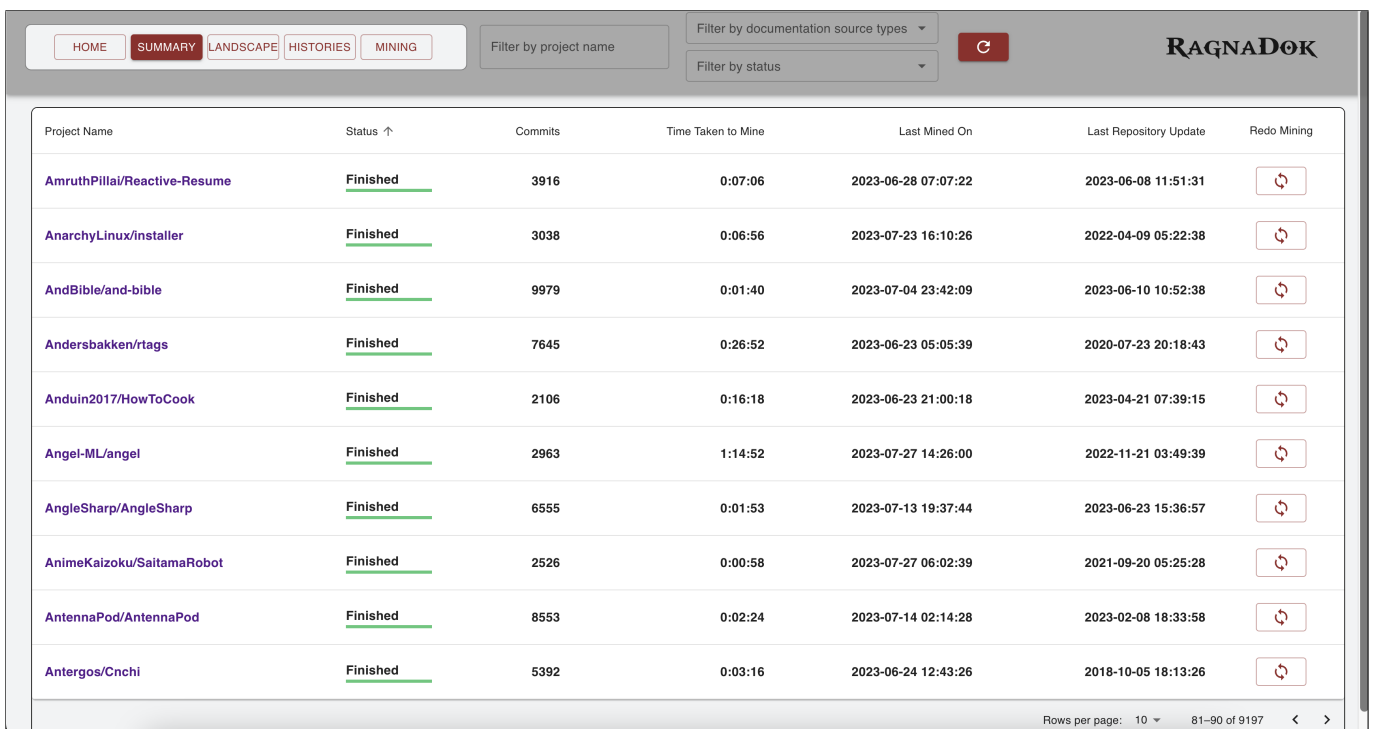
FIGURE 5.6: Aggregate Documentation Landscape

<sup>5</sup><https://github.com/elastic/elasticsearch>

The view maintains the same aspect as the single documentation landscape view, it has a larger legend for each bucket range along with the number of projects that existed at the time and the number of projects in which our taxonomy fails to capture at least one documentation source. In addition, the option bar allows to recompute an updated aggregate documentation landscape from the latest version of the data in the system. Computing the entire aggregate landscape is expensive and, because it depends on many projects, it tends to vary less over short time intervals. Because of these reasons, we choose to display the aggregate landscape after the year 2000 and we compute it every trimester.

### Summary View

We present a page to view the summary of the projects that we have mined and their current status, along with other useful data (Figure 5.7). In addition, the toolbar at the top of the page allows to filter by project name, source categories and by mining status. The display table can be sorted by any of the displayed parameters.



Project Name	Status ↑	Commits	Time Taken to Mine	Last Mined On	Last Repository Update	Redo Mining
<a href="#">AmruthPillai/Reactive-Resume</a>	Finished	3916	0:07:06	2023-06-28 07:07:22	2023-06-08 11:51:31	
<a href="#">AnarchyLinux/Installer</a>	Finished	3038	0:06:56	2023-07-23 16:10:26	2022-04-09 05:22:38	
<a href="#">AndBible/and-bible</a>	Finished	9979	0:01:40	2023-07-04 23:42:09	2023-06-10 10:52:38	
<a href="#">Andersbakken/rtags</a>	Finished	7645	0:26:52	2023-06-23 05:05:39	2020-07-23 20:18:43	
<a href="#">Anduin2017/HowToCook</a>	Finished	2106	0:16:18	2023-06-23 21:00:18	2023-04-21 07:39:15	
<a href="#">Angel-ML/angel</a>	Finished	2963	1:14:52	2023-07-27 14:26:00	2022-11-21 03:49:39	
<a href="#">AngleSharp/AngleSharp</a>	Finished	6555	0:01:53	2023-07-13 19:37:44	2023-06-23 15:36:57	
<a href="#">AnimeKaizoku/SaitamaRobot</a>	Finished	2526	0:00:58	2023-07-27 06:02:39	2021-09-20 05:25:28	
<a href="#">AntennaPod/AntennaPod</a>	Finished	8553	0:02:24	2023-07-14 02:14:28	2023-02-08 18:33:58	
<a href="#">Antergos/Cnchi</a>	Finished	5392	0:03:16	2023-06-24 12:43:26	2018-10-05 18:13:26	

FIGURE 5.7: Summary View of the Mined Projects

### Mining View

Lastly, we present a page to mine projects directly from the UI (Figure 5.8). We used this to mine the repositories in the thesis. The interface presents an option to mine a single project or mine multiple projects. The option to mine multiple projects will be disabled as it is very resource consuming and we do not wish for it to be abused. We also allow to chain the projects which histories we postponed during mining.

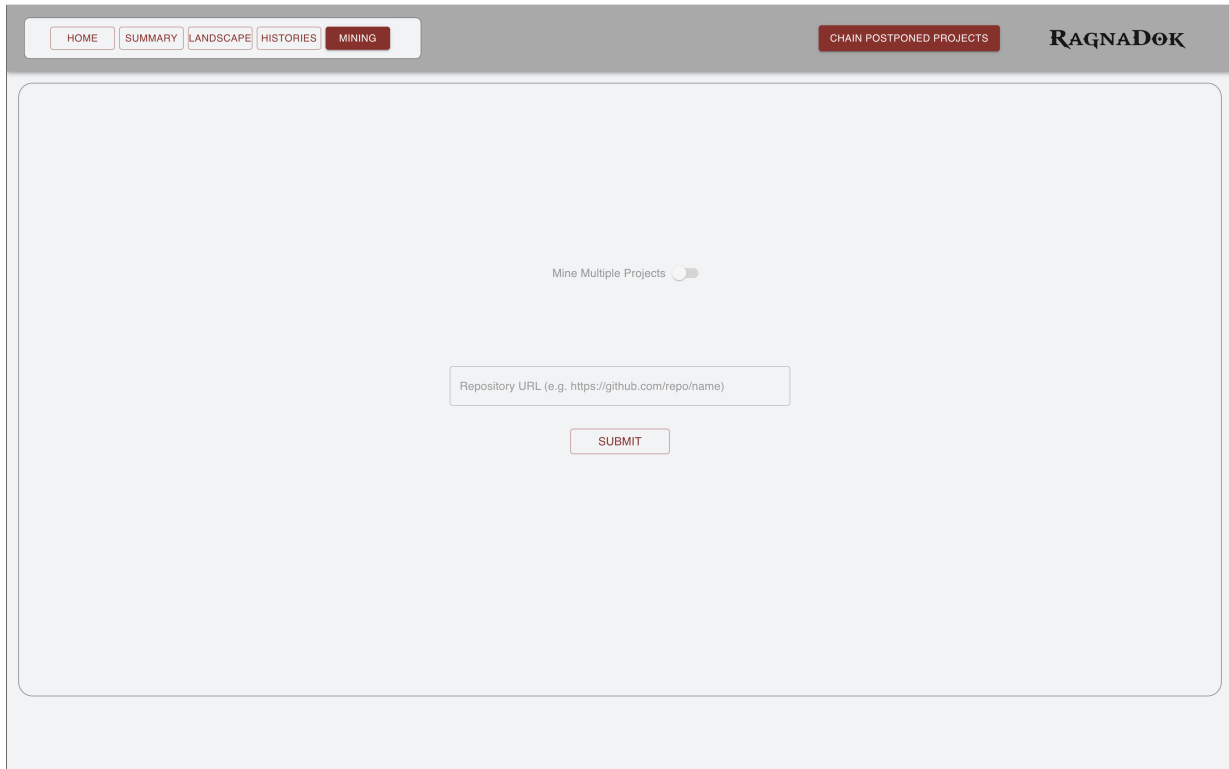


FIGURE 5.8: Mining View

## 5.4 Conclusions

In this chapter, we went over RagnaDok, the tool that implements our approach and taxonomy. We described the way it mines and extracts data and the visualizations that it provides to map the documentation landscape of a project. We can now discuss and analyze the data that we extracted via practical examples of the aggregate documentation landscape and specific case studies.



## Chapter 6

# Analysis and Discussion

In this chapter, we provide an analysis over the data that we have mined and processed with RagnaDok, our tool. We also present a number of case studies to look in-depth at the documentation landscapes of specific projects to give examples of what the landscape can look like. In addition, we also analyse the comprehensive documentation landscape that we extract from all the projects that we mined. Finally, we describe the limitations and threats to the validity of our thesis.

### 6.1 Dataset

We now provide a quantitative and qualitative analysis of the data that we gathered through RagnaDok, the tool that implements our approach.

#### 6.1.1 Quantitative Analysis

We summarize general data regarding the projects that we mined and their histories (Table 6.1) across a total number of 9,169 analyzed repositories.

Metric	Total Sum	Average	Median	Minimum	Maximum
# of Histories	808,291	88.10	16	1	52,889
# of Sources	4,000,553	436.07	121	0	103,626
# of Source Categories	178,017	19.40	18	0	53

TABLE 6.1: Repository and History Data

We can see how, despite the high average of 88.1 histories per repository, the median lies around 16. This high discrepancy is due to some outliers that present an extremely high number of histories (Figure 6.1).

#### Outlier Projects

We find some outliers with a number of histories that range in the thousands, and, among these, we identify three repositories that present an anomalous extreme number of histories (Table 6.2). These repositories do not contain a single software systems that evolved over time. Instead, they contain a large amount of aggregate information regarding a certain subjects.

Project Name	Number of Histories
demisto/content	13,986
doocs/leetcode	52,889
qmk/qmk_firmware	14,168

TABLE 6.2: Outlier Projects

The *doocs/leetcode*<sup>1</sup> project is a repository that holds solutions in many programming languages to programming problems from LeetCode<sup>2</sup>. Each of the solutions contains its own README file.

The *demisto/content*<sup>3</sup> repository displays the following in its README:

“This repo contains content provided by Demisto to automate and orchestrate your Security Operations. Here we will share our ever-growing list of playbooks, automation scripts, report templates and other useful content.”

This project uses GitHub as storage for much more than a singular software system.

The *qmk/qmk\_firmware*<sup>4</sup> project might be the one that distinguishes itself among these three. The repository revolves around a singular system: firmware for keyboards. The large number of histories is given by the keyboards directory on the repository. It holds information on all the supported keyboards, each with its own dedicated README file. Despite this difference among the projects, it becomes an outlier because of the sheer number of README histories that it contains.

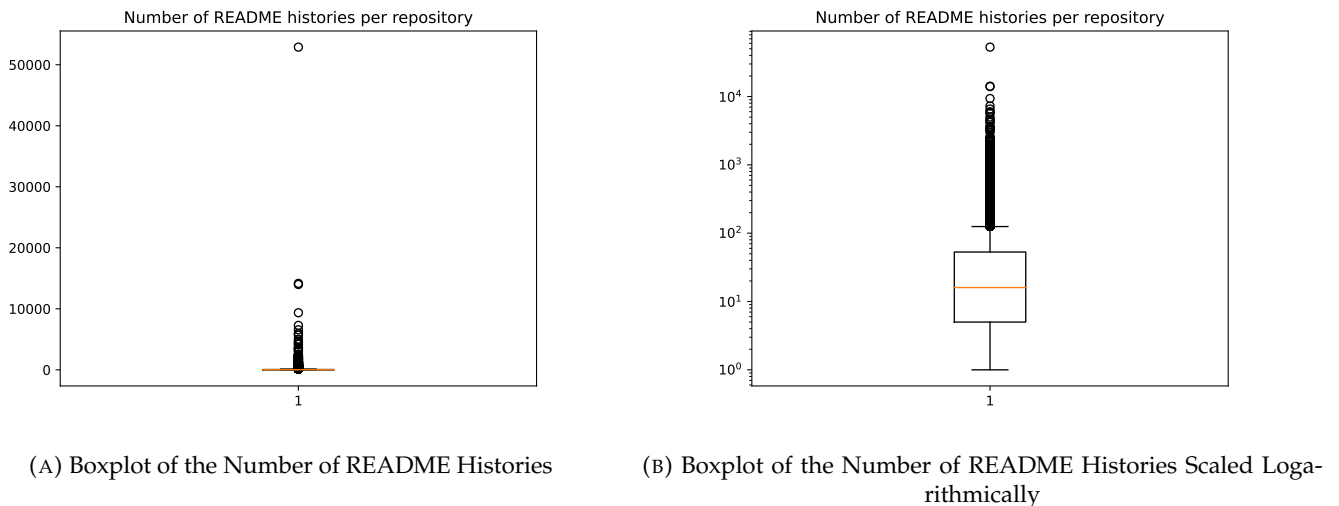


FIGURE 6.1: README History Outliers

## 6.2 Documentation Landscape in the Wild

In this section, we go over the evolution of the general documentation landscape that we get by aggregating the data from the 9,169 repositories that we examined. We present the aggregate documentation landscape starting from the year 2000. We do this because the aggregate documentation landscape is computationally expensive to produce over the number of projects that we mined and because the number of existing

<sup>1</sup><https://github.com/doocs/leetcode>

<sup>2</sup><https://leetcode.com/>

<sup>3</sup><https://github.com/demisto/content>

<sup>4</sup>[https://github.com/qmk/qmk\\_firmware](https://github.com/qmk/qmk_firmware)

projects before the year 2000 is low, representing around 1% of the total projects. Because the landscape is affected by multiple projects changing over time, we opt to take six-monthly snapshots as individual changes to a single project are less likely to impact the landscape, while we can observe differences on a longer period of time.

Our objective is to observe and understand what happened to the documentation sources over time, and notice which sources are used across projects, and how their popularity and usage has changed across two decades. We can already observe how the eight top level categories evolved over time (Figure 6.2).

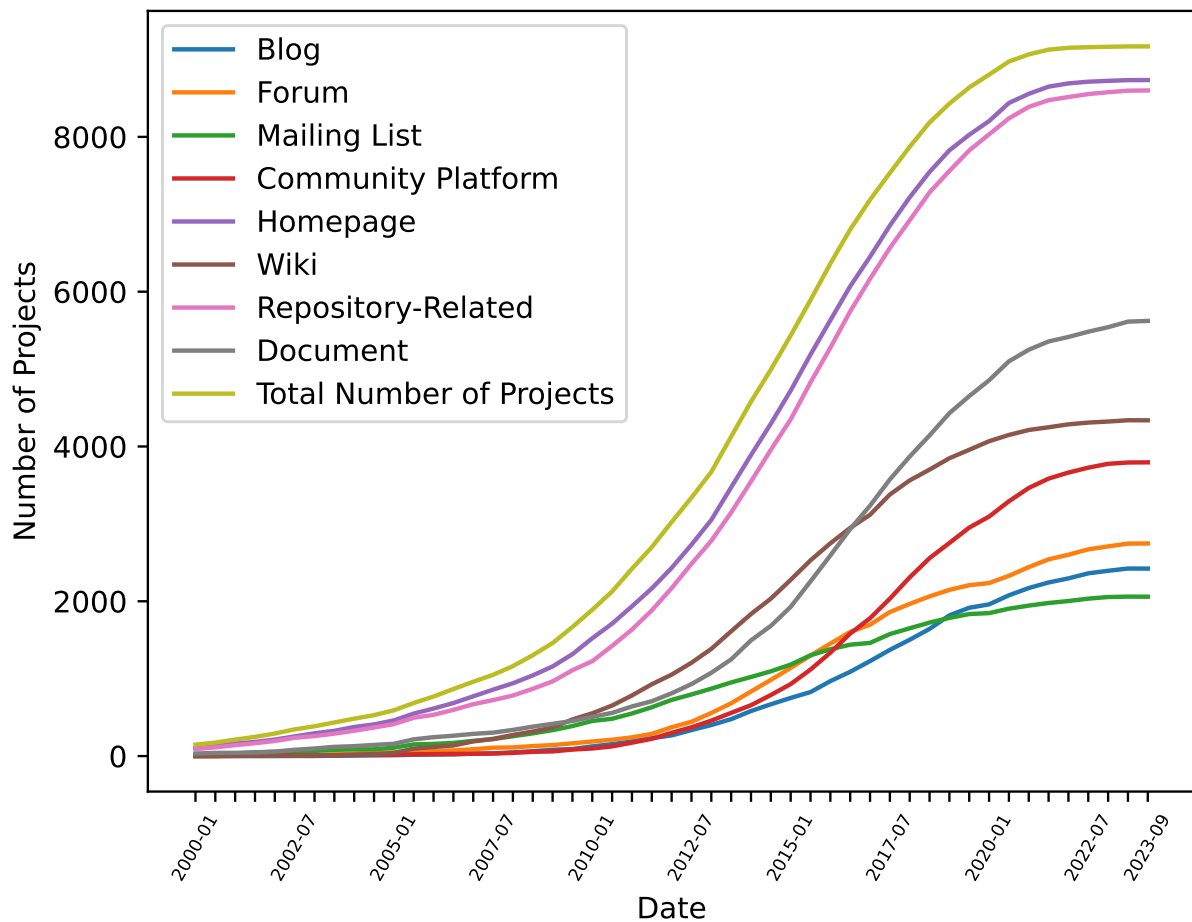


FIGURE 6.2: Presence in Projects of Top Level Categories of the Taxonomy Over Time

We can see that homepages and repository-related documentation sources follow the pattern of the number of projects. Most of the projects reference a homepage, whether it is external or internal, as well as some documentation source that is tightly related to the repository itself. We also see how many of the categories that hold newer documentation sources (*e.g.*, Community Platform) rise sharply in recent years, as other categories are used in less and less projects (*e.g.*, Mailing List). The full picture lets us see a documentation landscape that has expanded for more than two decades and that might continue doing so.

### 6.2.1 Early Years

In this section, we see how the documentation landscape begins growing during its first five years into the 2000s. With an increasing number of projects being created, the documentation landscape begins to expand.

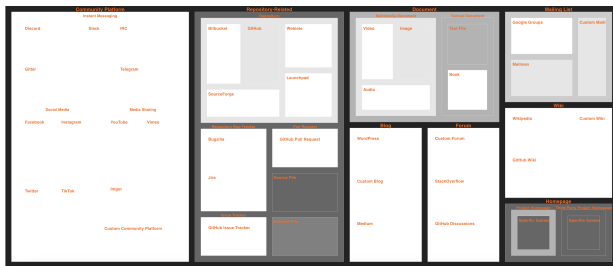
Number of projects	147	Number of projects	385	Number of projects	591
Category	Percentage	Category	Percentage	Category	Percentage
Blog	0.00%	Blog	0.26%	Blog	1.86%
Community Platform	0.00%	Community Platform	2.34%	Community Platform	2.88%
Document	22.45%	Document	25.45%	Document	26.90%
Forum	0.00%	Forum	3.38%	Forum	8.12%
Homepage	72.11%	Homepage	76.36%	Homepage	78.00%
Mailing List	10.20%	Mailing List	17.14%	Mailing List	18.27%
Repository-Related	62.59%	Repository-Related	67.01%	Repository-Related	69.88%
Wiki	0.00%	Wiki	1.30%	Wiki	6.94%
Uncategorized	27.89%	Uncategorized	22.34%	Uncategorized	22.00%

TABLE 6.3: Status of the Aggregate Landscape in its First Five Years (2000-2003-2005)

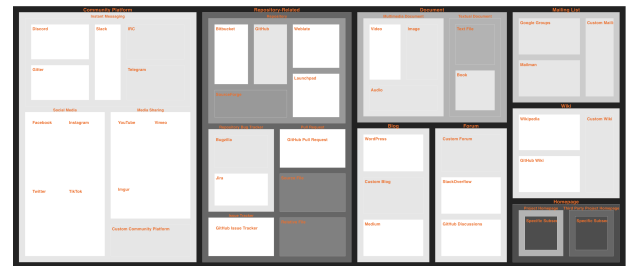
The starting point is at the beginning of 2000. Here, we can find that the most common categories are files (textual and source) within the repository and homepages. The main platforms for communication are mailing lists. (Figure 6.3a). GitHub seems to be present in the landscape at this point in time, but this is due to a misclassification in a project, as GitHub did not exist yet at the time.

In 2003, we see that SourceForge joins homepages, documents and files as a widespread documentation source, appearing in most of the projects. Mailing lists become more popular as Google Groups and Mailman are used. The first community platforms seem to appear along with the first blogs, forums, and wikis. Bugzilla starts being referenced for bug tracking and reporting. We find Telegram in the documentation landscape, but this is due to another misclassification.

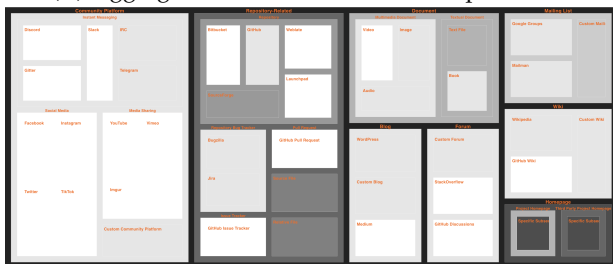
In 2005, we find the first Wordpress and Wikipedia documentation sources in the landscape together with Jira while most of the other sources remain stable or grow in usage.



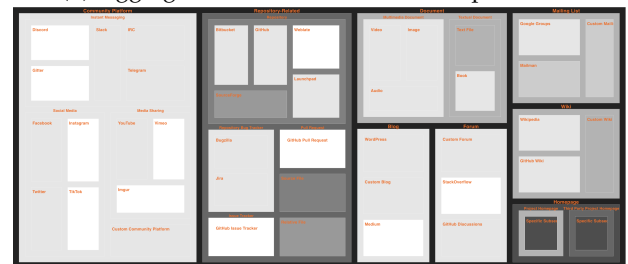
(A) Aggregate Documentation Landscape in 2000



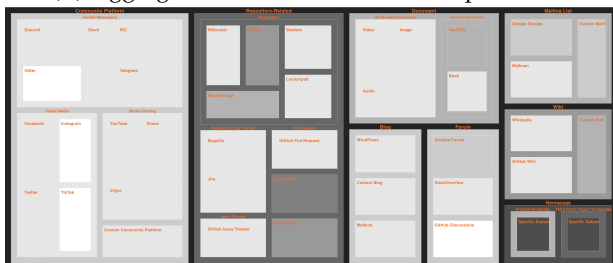
(B) Aggregate Documentation Landscape in 2003



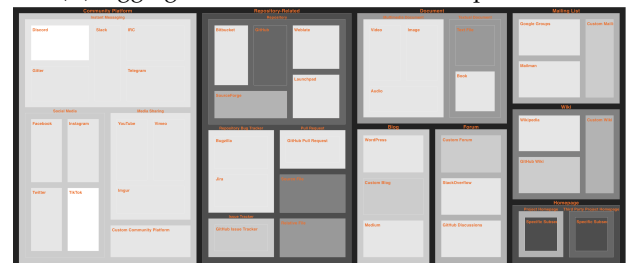
(C) Aggregate Documentation Landscape in 2005



(D) Aggregate Documentation Landscape in 2009



(E) Aggregate Documentation Landscape in 2012



(F) Aggregate Documentation Landscape in 2015

FIGURE 6.3: Aggregate Documentation Landscape between 2000 and 2015



### 6.2.2 Towards Modern Sources

The documentation landscape continues growing steadily between 2005 and 2015. As the number of projects increases, so does the number of sources and categories (Table 6.4). Some even start decreasing in their usage, possibly being replaced by newer ones.

Number of projects	1460	Number of projects	3339	Number of projects	5434
Category	Percentage	Category	Percentage	Category	Percentage
Blog	5.55%	Blog	10.06%	Blog	13.82%
Community Platform	4.11%	Community Platform	11.14%	Community Platform	17.11%
Document	28.56%	Document	27.85%	Document	35.48%
Forum	9.73%	Forum	13.27%	Forum	20.96%
Homepage	79.32%	Homepage	81.79%	Homepage	86.90%
Mailing List	22.95%	Mailing List	23.84%	Mailing List	21.75%
Repository-Related	66.10%	Repository-Related	74.39%	Repository-Related	80.05%
Wiki	25.34%	Wiki	36.12%	Wiki	41.90%
Uncategorized	21.64%	Uncategorized	22.34%	Uncategorized	29.67%

TABLE 6.4: Status of the Aggregate Landscape between 2009 and 2015

In 2009, we begin seeing social media and media-sharing websites on the rise, with YouTube, Twitter and Facebook showing up. Repositories become more popular with Launchpad, Bitbucket, and GitHub wikis starting to appear (Figure 6.3d). We also find another false positive for Slack, as it did not exist yet at this time.

Moving forward to the latter half of 2012, we see how community platforms and media-sharing websites start occupying more of the landscape. Google Groups follows along, and so does GitHub. Vimeo and Imgur make an appearance together with Medium and StackOverflow. GitHub now appears in a large percentage of all projects at this point in time, and some of them have a GitHub wiki and GitHub pull requests. Discord is another false positive, as it did not exist in 2012.

Finally, in 2015, we see how social media becomes more relevant, with the addition of Instagram to the landscape. While instant messaging remains stable, we see how now Slack, Gitter and Telegram do exist in the landscape (on top of the false positives in past years). As GitHub takes over, we notice how SourceForge has slightly decreased in usage. Blogs also become more popular. At this point in time we also see how Bugzilla and Jira are steadily replaced by the GitHub issue tracking system which, among other things, tracks bugs (Figure 6.3f).

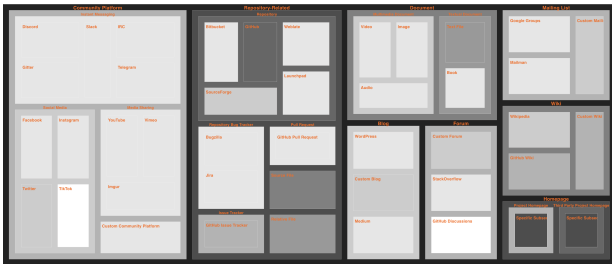
### 6.2.3 The Current Landscape

We now move onto viewing the documentation landscape from 2015 to 2023, the time of writing of this thesis. In this section, we see how some older documentation sources have declined and have been overtaken by newer ones. The landscape is more stable at this point, only a few sources grow more popular during the years, and the sample size increases to our total number (Table 6.5).

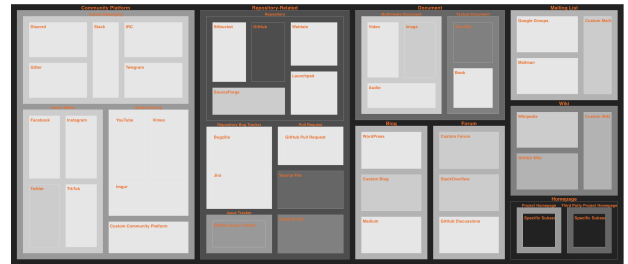
Number of projects	6363	Number of projects	8430	Number of projects	9169
Category	Percentage	Category	Percentage	Category	Percentage
Blog	15.21%	Blog	21.57%	Blog	26.42%
Community Platform	21.00%	Community Platform	32.66%	Community Platform	41.39%
Document	40.74%	Document	52.55%	Document	61.32%
Forum	22.84%	Forum	25.47%	Forum	29.96%
Homepage	88.50%	Homepage	92.84%	Homepage	95.26%
Mailing List	21.69%	Mailing List	21.17%	Mailing List	22.46%
Repository-Related	82.92%	Repository-Related	89.68%	Repository-Related	93.79%
Wiki	43.20%	Wiki	45.65%	Wiki	47.31%
Uncategorized	34.10%	Uncategorized	45.73%	Uncategorized	51.83%

TABLE 6.5: Status of the Aggregate Landscape between 2015 and 2023

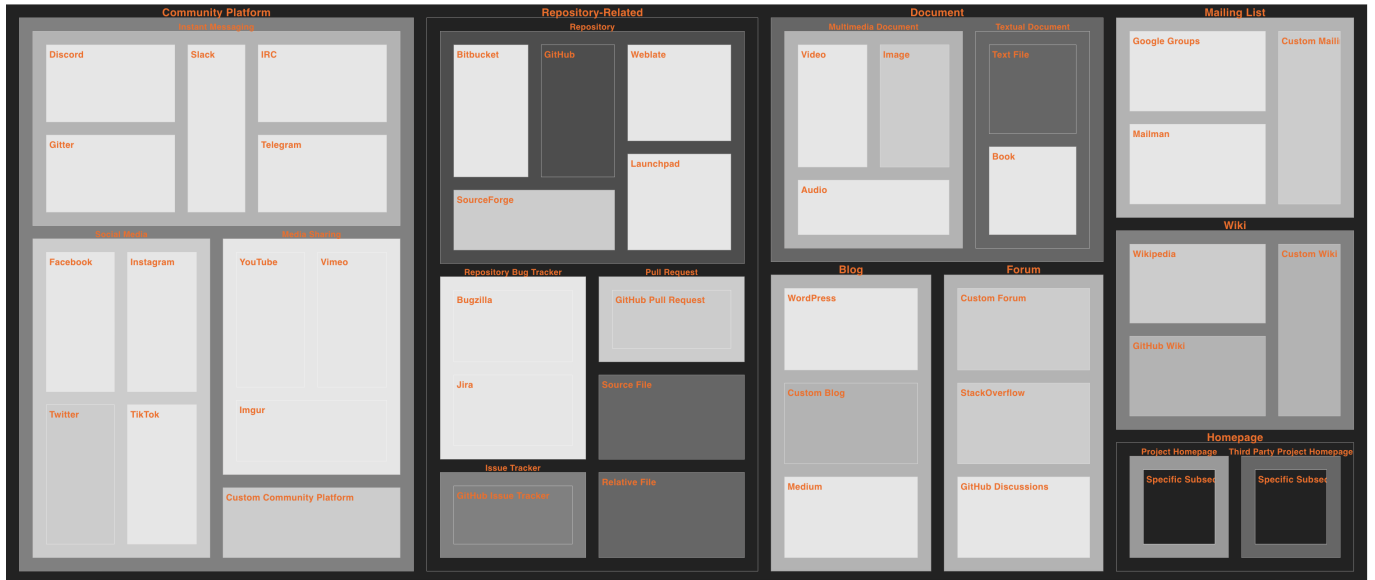
We can see a sort of stability in the landscape now, most of the sources are well established and the top level categories do not change too much, but we can see how many documentation sources are into the process of being (or have been) replaced. For instance, we see how more than a third of the landscape uses instant messaging platforms as tools. IRC makes up a very small percentage of this nowadays and it is no longer the leading channel with Discord and Slack picking up the slack. We also see Bugzilla and Jira disappear from the landscape. They are not fully gone, but they now exist in such a small fraction that they end up represented as non-existent. Issue tracking from GitHub has replaced older bug trackers for the most part, GitHub itself reigns supreme in the Repository category. Forums and wikis are the only categories where GitHub coexists peacefully.



(A) Aggregate Documentation Landscape in 2016



(B) Aggregate Documentation Landscape in 2019



(C) Aggregate Documentation Landscape in 2023

FIGURE 6.4: Aggregate Documentation Landscape between 2015 and 2023

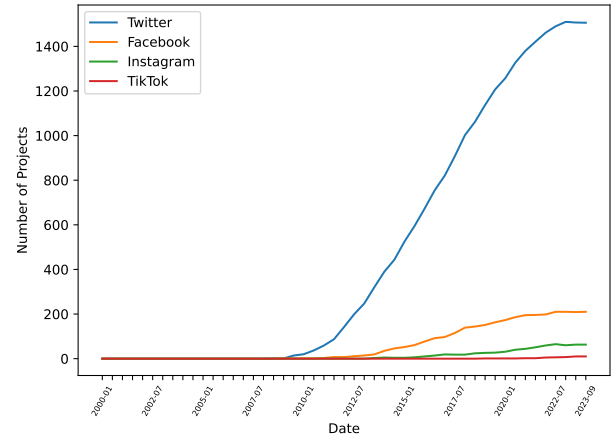
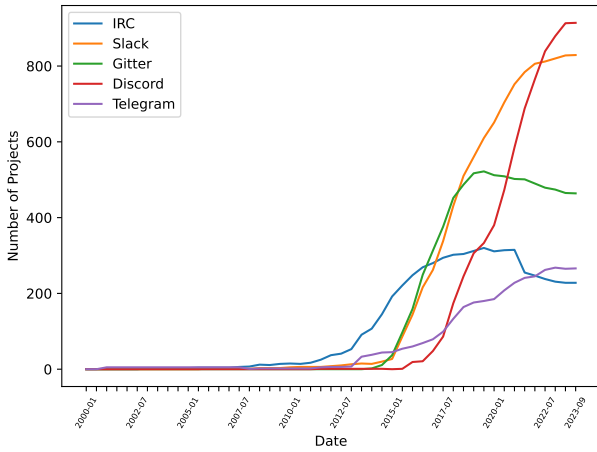
### 6.2.4 Documentation Landscape of the Subcategories

We also take a closer look at the landscape of some specific subcategories of the documentation landscape. In this section, we present how social media, instant messaging, and community platforms evolve over time (Figure 6.5). We can clearly see that Twitter is the favoured social media platform in open-source projects, and the number of Twitter documentation sources that we see is also under-represented in our dataset. We do not capture Twitter handles (e.g., @username), we only focus on the name of the platform, so we can expect the number of Twitter documentation sources to be even greater (Figure 6.5b).

We also see that instant messaging platforms show a very interesting scenario. Older platforms like IRC are in decline, while newer and more popular platforms like Discord and Slack have been on the rise in recent years, quickly overtaking their competitors (Figure 6.5a).

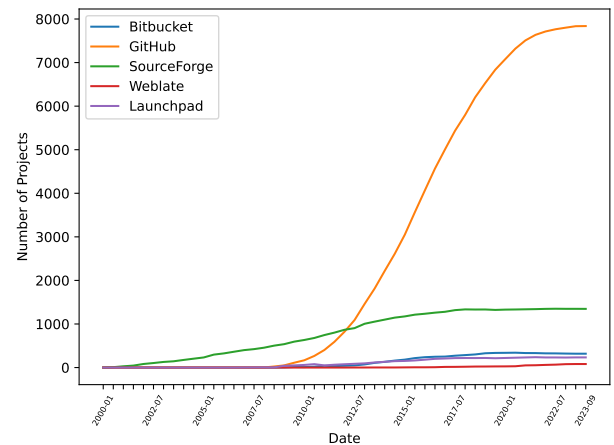
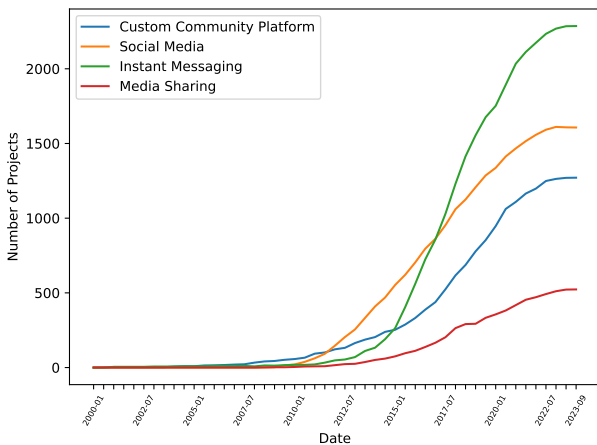
Overall, we notice that instant messaging is the favoured community platform by developers, with social media trailing behind (Figure 6.5c). Custom platforms and media sharing platforms have been on the rise, but they are not as popular as the former two. From this, we gather that software development has shown a greater interest towards developing a community around the project to collaborate.

In addition, we can see how GitHub towers over other repository category. Since its appearance, it has become more and more popular, leaving the other categories behind (Figure 6.5d).



(A) Documentation Landscape of Instant Messaging Platforms

(B) Documentation Landscape of Social Media



(C) Documentation Landscape of Community Platforms

(D) Documentation Landscape of Repository

FIGURE 6.5: Documentation Landscape of Community Platforms and its Subcategories

### 6.3 Conclusions

In this chapter, we have seen how the aggregate documentation landscape has evolved over time. We have been able to see that the documentation landscape has been in constant growth during the past twenty years. While it more stable now, some older sources are far less popular than they used to be and, instead, newer and faster platforms are taking over the landscape. We cannot know what kinds of software will be developed, but it is not unreasonable to imagine that these newer documentation sources will one day be replaced by even better ones. Let us now take a closer look at specific case studies.



## Chapter 7

# Case Studies

In this chapter, we present 3 case studies to show interesting and various shapes that the documentation landscape of a project can take over time. Some of the following projects present an empty landscape when they were created. Please, refer to Figure 7.1 for an example. In fact, it is not uncommon for the first commit on a repository to be a project setup file (e.g., `.gitignore` file), a small source file, or a short README file that has not yet evolved to reference sources.

In addition, in Section 5.2.1, we mentioned how we perform a recovery step for sources. We will see how this step can affect the documentation landscape either by completing it or by damaging it.

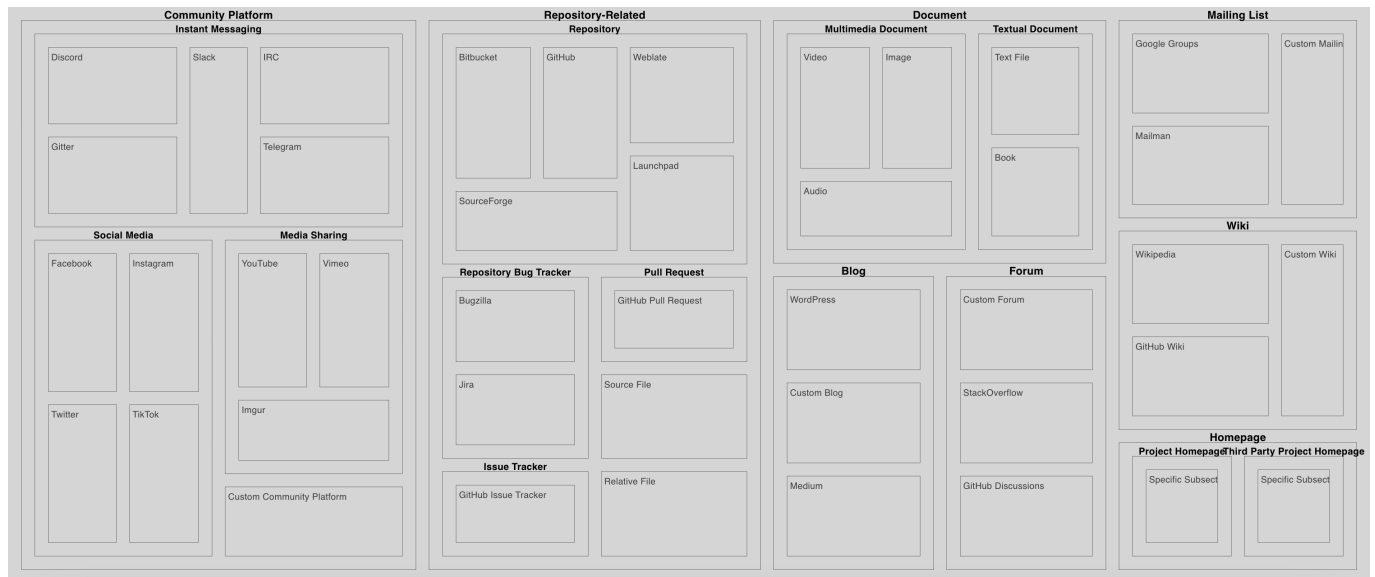


FIGURE 7.1: Empty Documentation Landscape for Reference

## 7.1 An Explosion of Sources: scikit-learn

The scikit-learn project<sup>1</sup> on GitHub is a popular open-source Python module that provides a comprehensive collection of tools for various machine learning tasks. With this case study, we aim to show how the documentation landscape of the project evolves and presents a rapid growth in documentation sources in recent years.

<sup>1</sup><https://github.com/scikit-learn/scikit-learn>

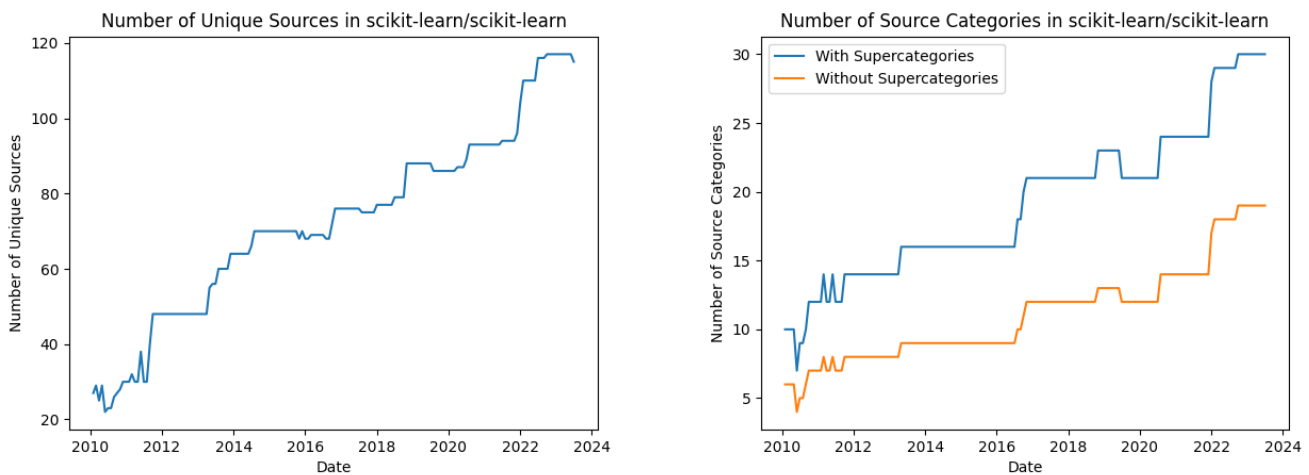
### 7.1.1 Overview

At the time of mining, the project has 70 README histories and 30,155 commits. We also present the number of unique documentation sources and categories on the latest version that we have mined with and without recovered documentation sources (Table 7.1). The amount of sources and categories are very close to each other in each case, this means that we capture most of the landscape via clear identification (*i.e.*, live or dead sources) and that the small number of recovered sources may potentially expose additional insight.

	With Recovered	Without Recovered
Total Number of Unique Sources	74	60
Number of Source Categories	28	27

TABLE 7.1: Descriptive Statistics of scikit-learn

The scikit-learn library has existed for little over a decade at the time of writing and, throughout its life, it presents a gradual growth in the documentation sources that it contains (Figure 7.2a). Furthermore, we observe that the landscape already presents documentation sources at the beginning of the project, along with a rather steep growth in the categories that it covers during its most recent years (7.2b).



(A) Unique Documentation Sources over Time for scikit-learn

(B) Unique Source Categories over Time of scikit-learn

FIGURE 7.2: Documentation Landscape over Time of scikit-learn

### 7.1.2 Beginnings

The landscape of scikit-learn starts empty, with only a text file being referenced on the first commit (Figure 7.3a). Minutes after the first commit (Figure 7.3b), the documentation landscape grows quickly referencing files across the repository, SourceForge and some external websites. The SciPy website<sup>2</sup> is mentioned as scikit-learn is built upon it, SourceForge mentions the Windows installation for the ctypes library and, on the following day, the main README file of the project is created for the first time. The README mentions SourceForge, but in this case it was not a project, but a mailing list hosted on SourceForge.<sup>3</sup>

<sup>2</sup><https://scipy.org/>

<sup>3</sup><https://sourceforge.net/projects/scikit-learn/lists/scikit-learn-general>

Only three months later, a Git mirror of the scikit-learn project<sup>4</sup> is referenced in the README file along with a custom wiki entry<sup>5</sup> about the Git mirrors for NumPy and SciPy. Unfortunately, this wiki entry is now unavailable. Both these entries will disappear in an update to the README two months later. In June of 2010, the #scikit-learn IRC channel on `irc.freenode.net` is first mentioned in the README. Within this short timespan, the README has also now become a `README.rst` in the reStructuredText mark-up syntax. GitHub also reappears in the landscape as it is now mentioned for cloning instead of the SourceForge repository of the project. In November, a reference to the homepage of scikit-learn appears for the first time.

In March of 2011, the project's landscape includes many documentation sources, as well as the GitHub issue tracking system that is mentioned in the README for bug reporting (Figure 7.4).

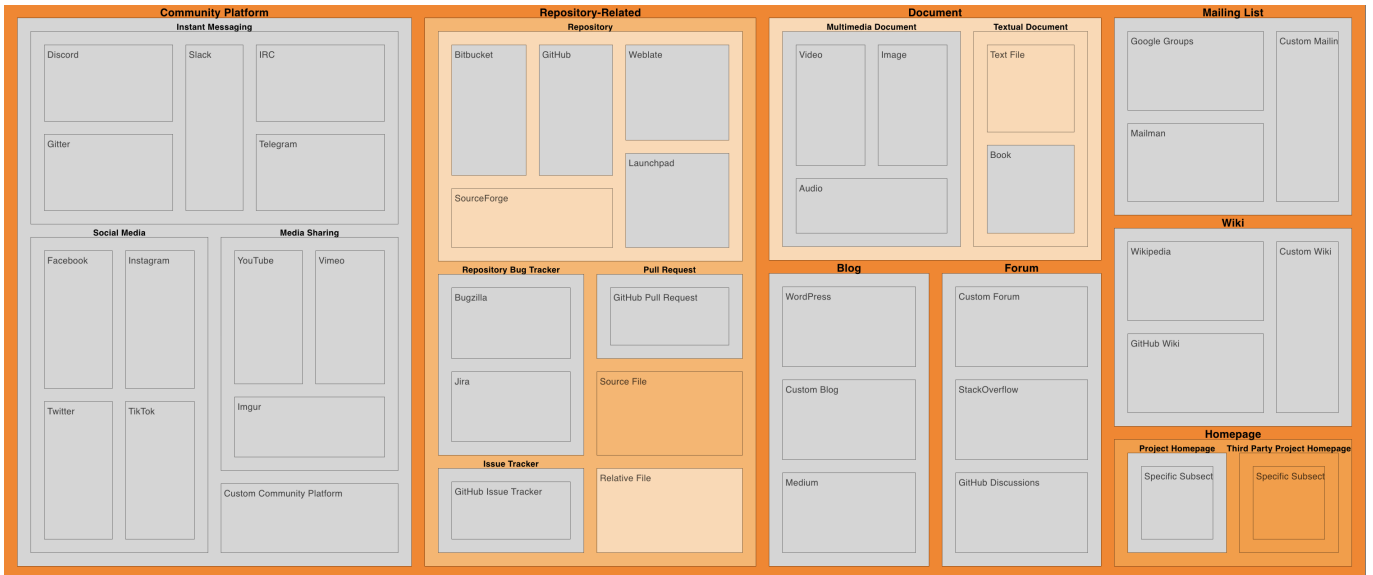
---

<sup>4</sup><https://github.com/yarikoptic/scikit-learn>

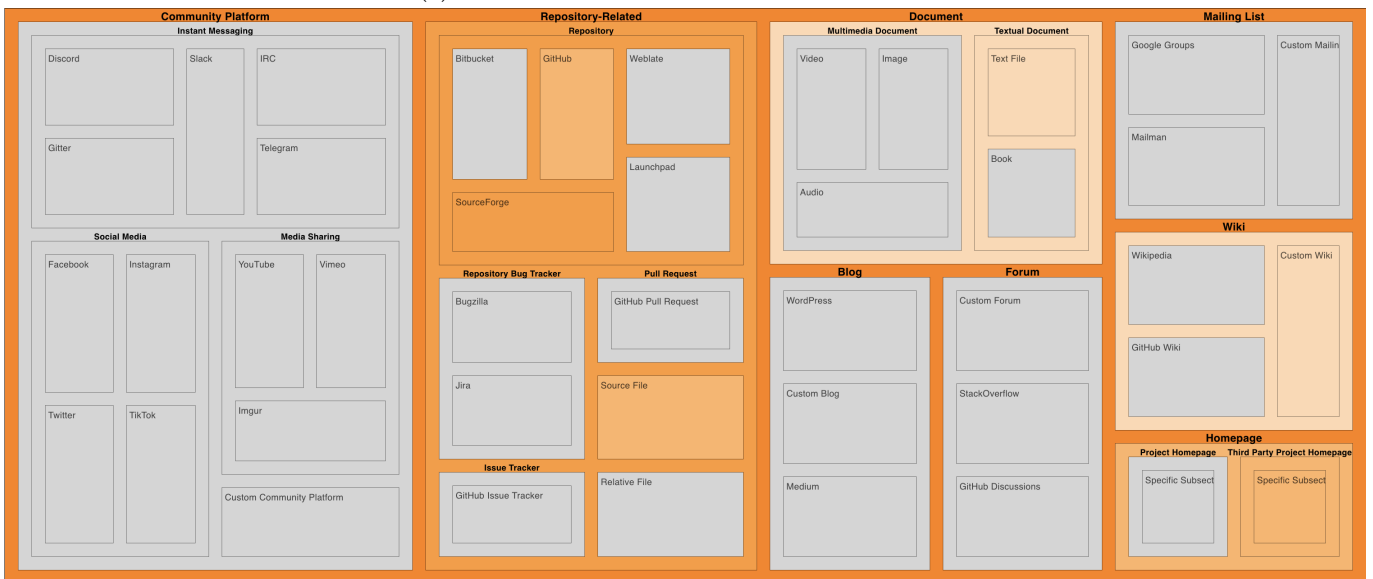
<sup>5</sup><http://projects.scipy.org/numpy/wiki/GitMirror>



(A) scikit-learn on its First Commit in 2010



(B) scikit-learn Three Hours After its First Commit



(C) scikit-learn Three Months After its First Commit

FIGURE 7.3: Initial Documentation Landscape of scikit-learn



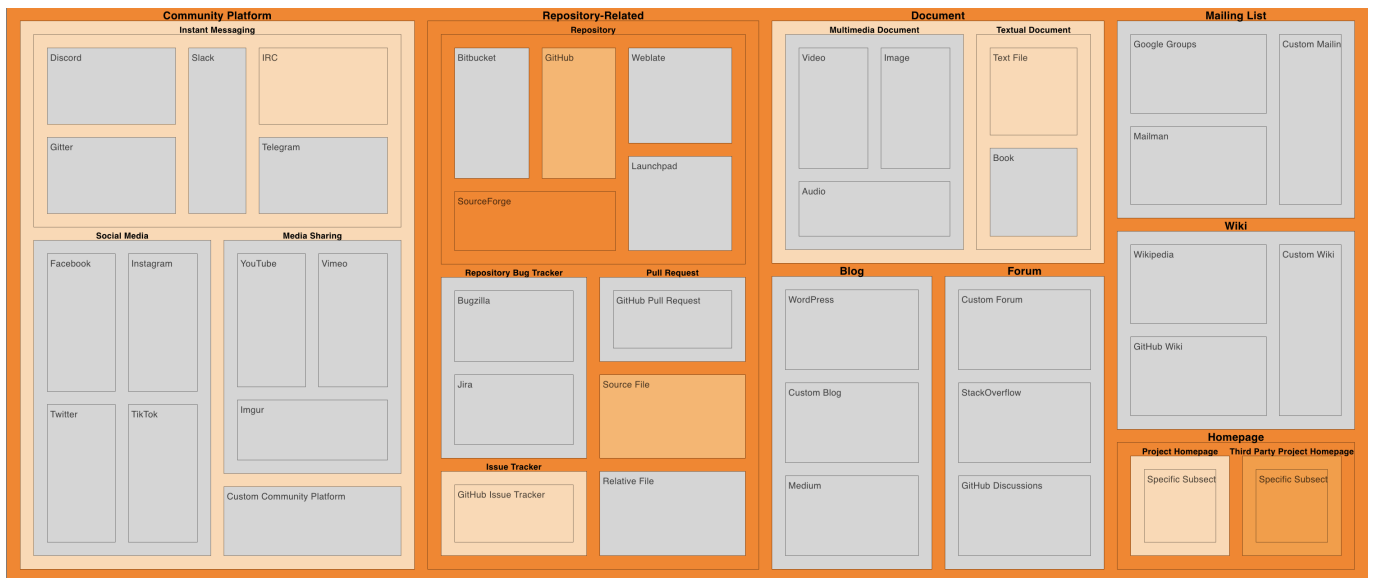


FIGURE 7.4: scikit-learn in 2011 After One Year

### 7.1.3 Growth Over the Years

In August of 2012, some images are added to a secondary README file as examples of the dataset and the landscape stabilizes until 2013, when a custom blog appears. Despite the path looking seemingly secondary: `doc/themes/scikit-learn/static/ML_MAPS_README.txt`, this blog entry references instructions to edit the README according to the commit message (Figure 7.5). This post gives instructions on how to edit a machine learning cheat sheet of the scikit-learn algorithms, making it relevant to the landscape. The landscape now remains dormant until 2016, when the old mailing list from SourceForge that we were not able to identify via our tool is swapped for a Mailman mailing list. The project had been using a mailing list since it started, and now switched its source. In addition the README also mentions questions tagged with `scikit-learn` on Stack Overflow.<sup>6</sup> The current landscape will not change until 2020 (Figure 7.6), when the number of source types increases dramatically.

```

File Path: doc/themes/scikit-learn/static/ML_MAPS_README.txt
Past names: None
URL: https://github.com/scikit-learn/scikit-learn/blob/707887d85841f98711ce82028906a3490456d365/doc/themes/scikit-learn/static/ML\_MAPS\_README.txt
Number of sources: 5
Number of lines: 94
On date: Tue, 23 Apr 2013 15:16:09 GMT
At commit: 707887d85841f98711ce82028906a3490456d365
By: Jaques Grobler
Message: add instructions for editing Readme, and script needed for that

```

FIGURE 7.5: Commit Referencing the Blog Entry

<sup>6</sup><http://stackoverflow.com/questions/tagged/scikit-learn>

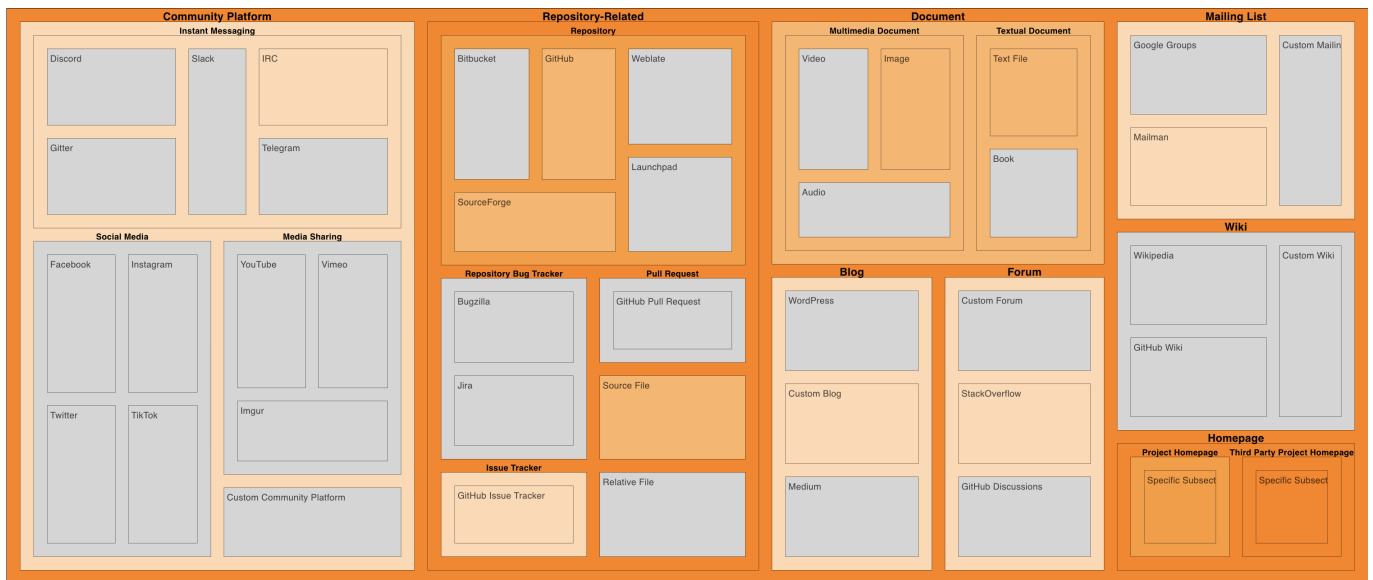


FIGURE 7.6: scikit-learn in 2016

### 7.1.4 Explosion of Sources

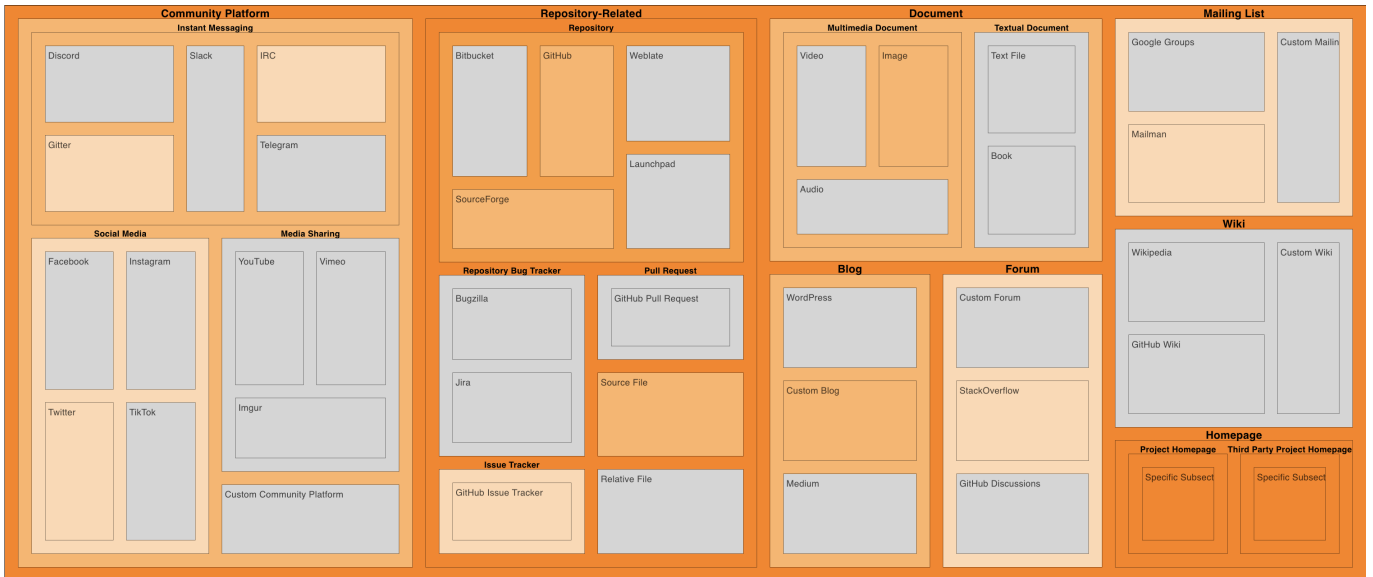
In July of 2020, both Gitter and Twitter appear in the landscape of the project (Figure 7.7a). One year later IRC is removed from the main README, and GitHub Discussions makes its appearance as a forum along with it. At the end of 2021, five more sources make an appearance all at once. These sources are: the official YouTube channel, a Twitter account for the commits made on the repository, an Instagram account, a Facebook account, and a LinkedIn account which remains uncategorized in the taxonomy. In 2022, a TikTok account is referenced and, in 2023, the Twitter account dedicated to commits is removed. The documentation landscape of scikit-learn changed significantly in the span of only three years and maintained a rapid evolution throughout this period.

What should be noticed about some of these sources is that, while they are present and referenced, we do not know to what degree they are used. A good example for this is the Instagram page of scikit-learn.<sup>7</sup> In this case, the page exists, but it can hardly be considered active. At the time of writing, the most recent post was made two months prior, in June, with the second most recent post being a year ago. The page is inactive and with a small following. While it exists in the landscape, it does not contribute to it as much.

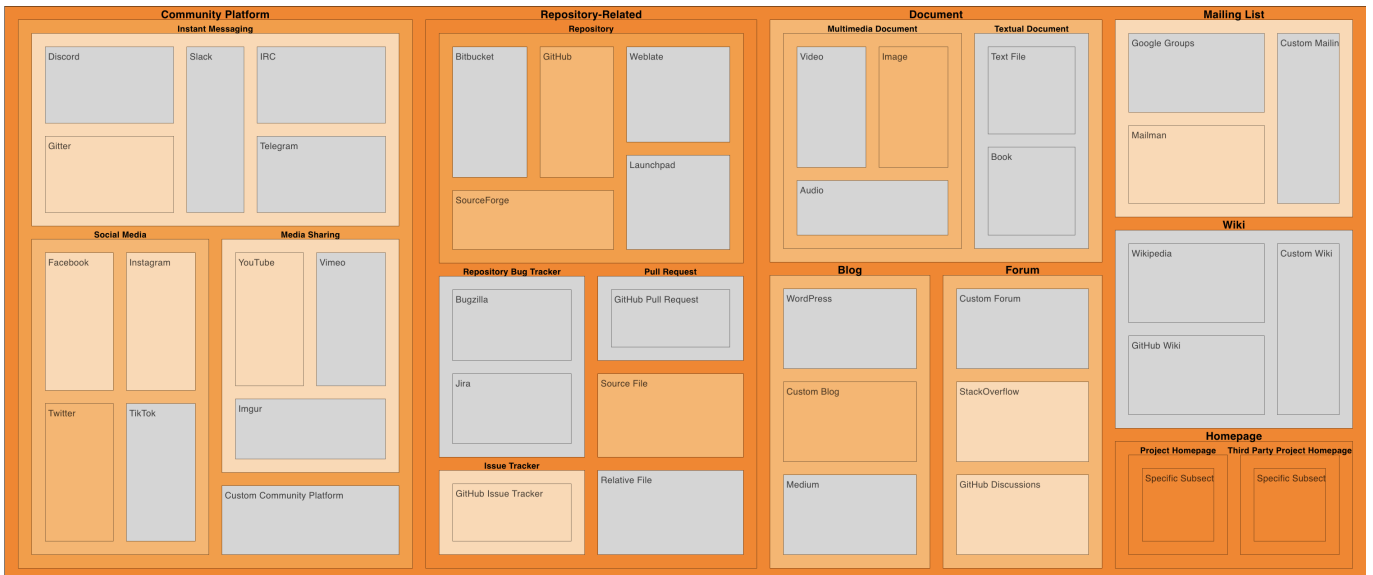
### 7.1.5 Conclusion

Within the span of 13 years, the documentation landscape of scikit-learn has grown steadily and has maintained a stable structure for the first half of its lifespan. From 2020 onwards, the landscape evolves rapidly, covering almost twice as many categories in the taxonomy. Thanks to the landscape view on RagnaDok, we can clearly see the landscape being taken over throughout the years and, in particular, we can see how many of these sources are in a light shade of orange, indicating the presence of a *single* source per category. When a source appears only once, the view lets us know that it is likely the project's official source. Sources that appear in a larger number, are likely to point to a larger set of documentation sources, such as source files or other repositories. The one exception is the homepage category. We can see it is packed with documentation sources. It is not uncommon to reference external websites, and we can clearly see how specific subsections of the project's homepage are referenced over time, saturating the category.

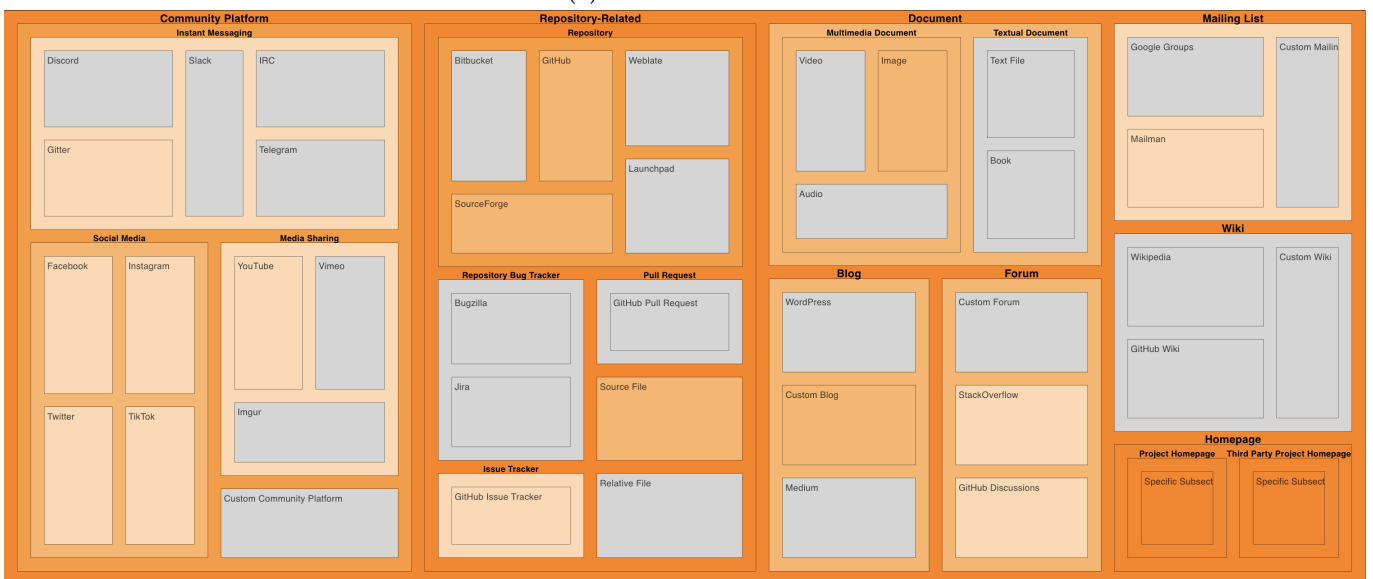
<sup>7</sup><https://www.instagram.com/scikitlearnofficial/>



(A) scikit-learn in 2020



(B) scikit-learn in 2021



(C) scikit-learn in May 2023

FIGURE 7.7: The Explosion of the Documentation Landscape

## 7.2 fish-shell/fish-shell: A Simple Landscape

Fish shell<sup>8</sup> is a modern, user-friendly command-line shell for Unix-like operating systems. It stands out for its interactive features, syntax highlighting, and extensive auto-suggestions that enhance the command-line experience. Fish aims to make the terminal more accessible for both beginners and experienced users. Its powerful scripting capabilities, combined with a vast collection of plugins and extensions, make Fish shell a popular choice for those seeking a more productive and user-friendly command-line environment.

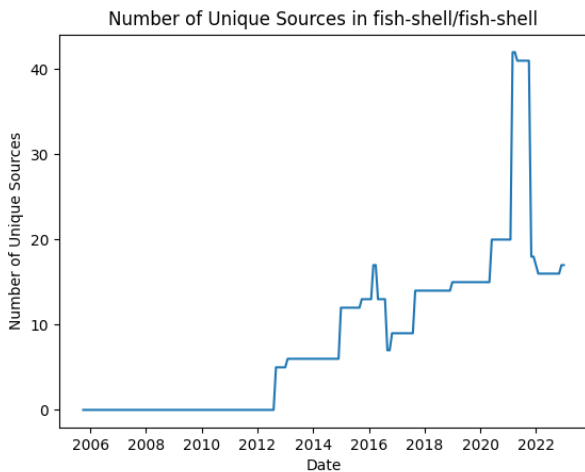
### 7.2.1 Overview and Evolution

Fish shell has existed for 16 years at the time of writing. It presents 17,412 commits and 10 total README histories. This is a simple case study, where we see how RagnaDok keeps track of the landscape as it evolves. We can observe that, without recovered sources, the number of unique documentation sources drops dramatically severely affecting the number of categories in the landscape. Because of this, we will view this landscape without the recovered sources.

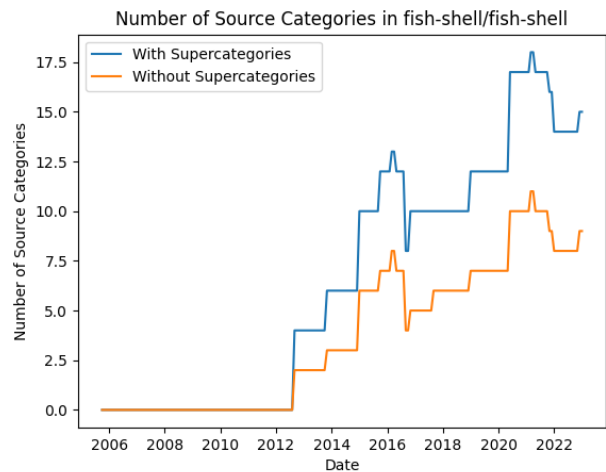
	With Recovered	Without Recovered
Total Number of Unique Sources	98	17
Number of Source Categories	18	17

TABLE 7.2: Descriptive Statistics of fish-shell

In the Fish shell project, we can see a gradual growth both in documentation sources and categories within its documentation landscape, with two larger spikes in 2016 and 2021 respectively (Figure 7.8). These two spikes match two separate updates to the README of a library used by the project. Because of this, we can ignore the spikes as they do not belong to the real landscape, but to their own landscape for the library that its README represents.



(A) Unique Documentation Sources over Time for Fish Shell



(B) Unique Source Categories over Time of Fish Shell

FIGURE 7.8: Documentation Landscape over Time of Fish Shell without Recovered Sources

One peculiar thing that is noticed right away when looking at the documentation landscape of Fish shell, is that the project is created in 2005 and its README is not updated until 2012 (Figure 7.9). This is

<sup>8</sup><https://github.com/fish-shell/fish-shell/>

because the README<sup>9</sup>, through this seven-year gap, has been referencing the official documentation via a command to be executed in the terminal. By running this command, the user can view the relevant documentation. In 2012, we now see SourceForge and a third-party homepage. In this case, the SourceForge documentation sources really points to a mailing list hosted on SourceForge. The third-party homepage in question is actually the project's own homepage. We do not find this because the project's name does not match the project's homepage domain. Fish shell used to reference its website under a different domain that is now unused.<sup>10</sup>

Soon after, in 2015, we see GitHub being mentioned, both with its wiki and via actual repositories. We also see Mailman appearing in another sub-README for a library.

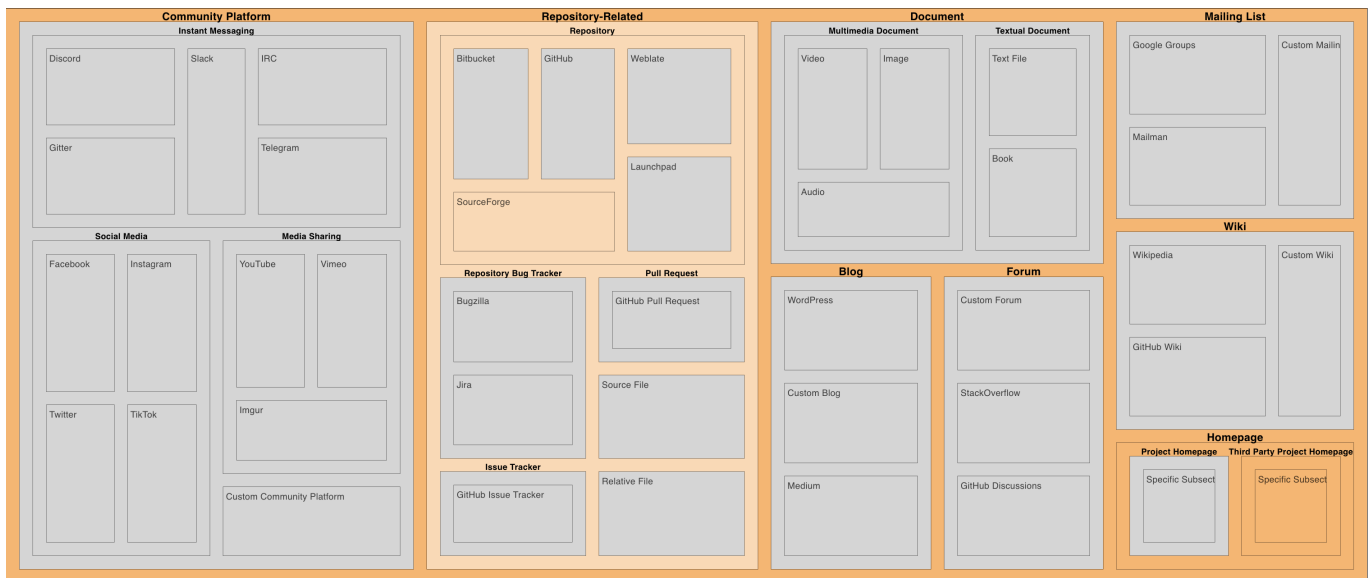


FIGURE 7.9: Fish Shell in 2012 when the First Sources Appeared

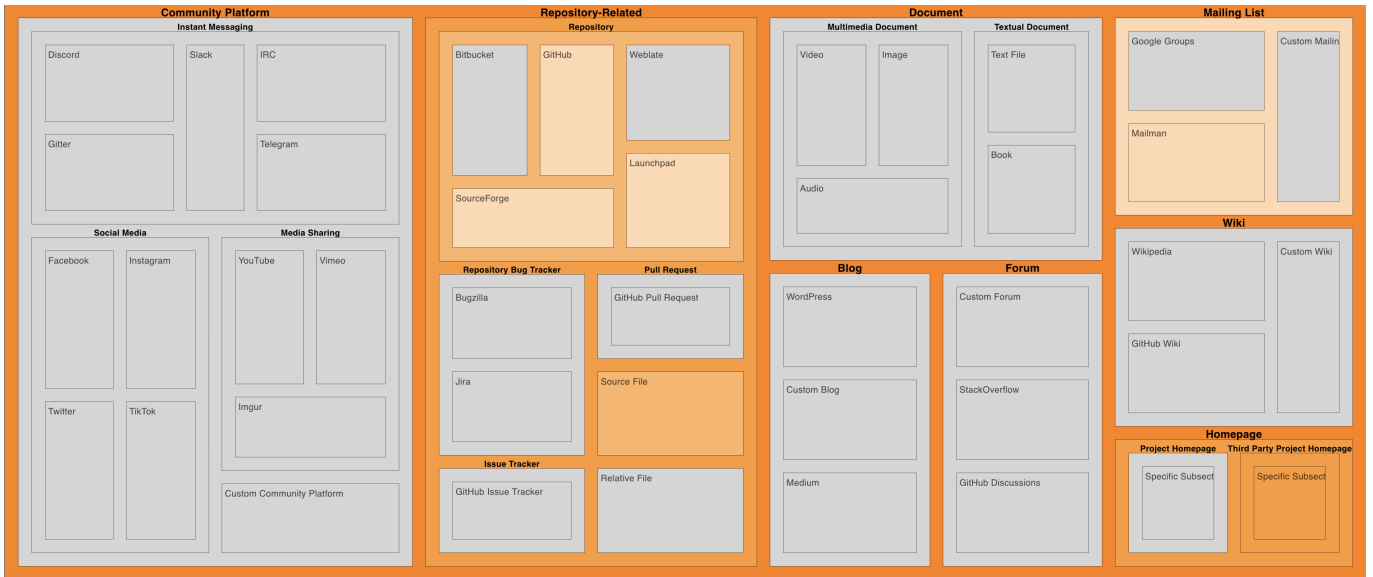
In 2017, we find a Launchpad archive in the main README of the repository and the landscape stabilizes for the next three years. Through the years, the project began referencing GitHub issues and, in 2020, the official Gitter channel for Fish shell and Stack Overflow posts tagged `fish` make an appearance in the README. In the final stage of the landscape, we see Stack Overflow disappearing. If we look deeper into this, we see that it was replaced in favour of `fish` tagged posts on Stack Exchange.

## 7.2.2 Conclusion

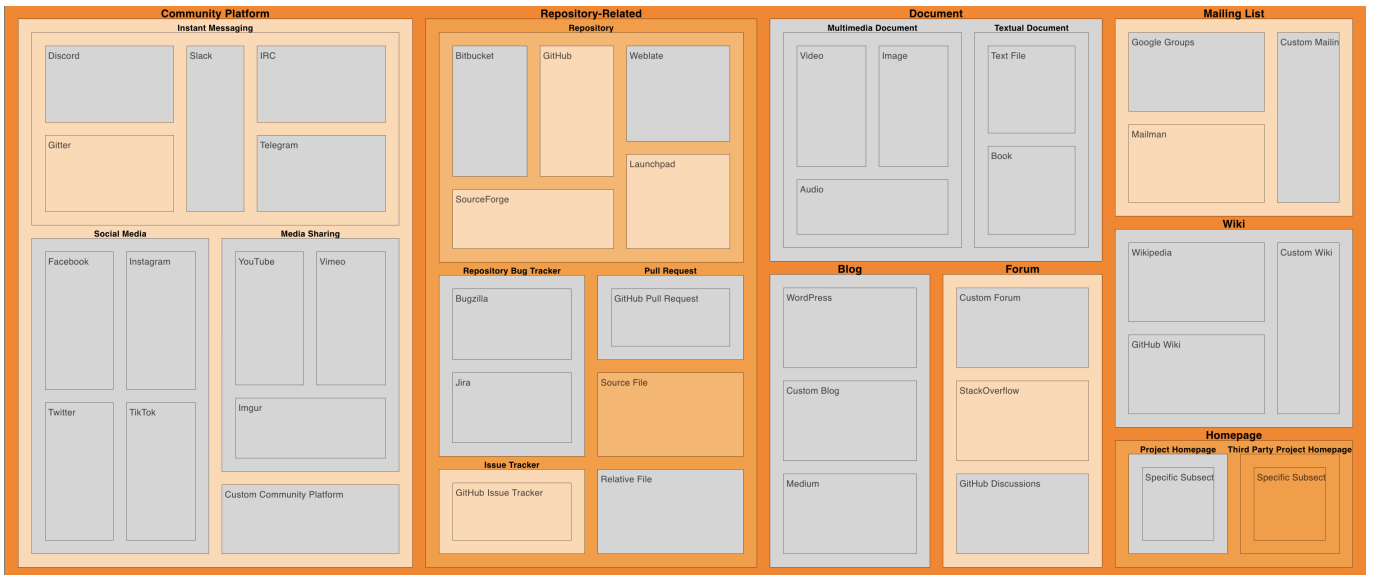
The Fish shell project presents a very simple landscape that is slightly affected by README files of external libraries. Nonetheless, we are able to observe the evolution of its documentation landscape over time with ease. Most of its sources appear in the top-level README file, giving us a genuine landscape that is not too noisy.

<sup>9</sup><https://github.com/fish-shell/fish-shell/blob/149594f974350bb364a76c73b91b1d5ffddaa1fa/README>

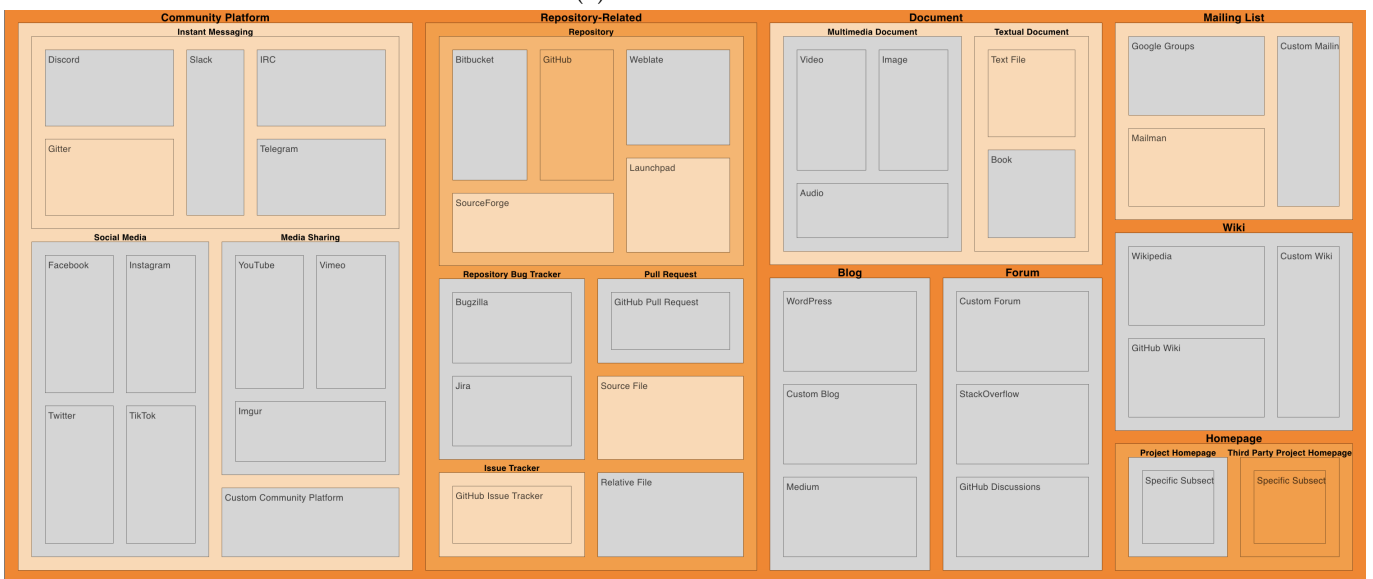
<sup>10</sup><https://ridiculousfish.com/shell/>



(A) Fish Shell in 2017



(B) Fish Shell in 2020



(C) Fish Shell in 2023

FIGURE 7.10: Evolution of the Documentation Landscape of Fish Shell

## 7.3 GCC: Hidden Landscape

The GNU Compiler Collection (GCC) stands as a cornerstone within the realm of software development. It serves as an open-source suite of compilers that facilitate the translation of high-level programming languages into machine code, enabling software to run on various hardware architectures. Initiated by Richard Stallman and developed by the Free Software Foundation (FSF), GCC exemplifies the principles of free and open-source software. With its extensive history and widespread adoption, GCC remains a pivotal tool for developers across the globe, supporting the foundation of many software projects and technological advancements.

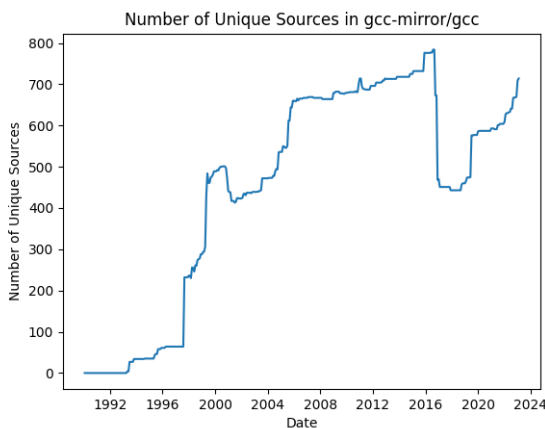
### 7.3.1 Overview

Through its numerous years of existence, the project has over 200,000 (202,236 at the time of mining) commits on GitHub at the time of writing and 320 total separate README histories. We present the total number of README histories and commits, along with the amount of unique documentation sources and categories that they cover on the most recent version of the project that we have analyzed both with and without recovered documentation sources (Table 7.3).

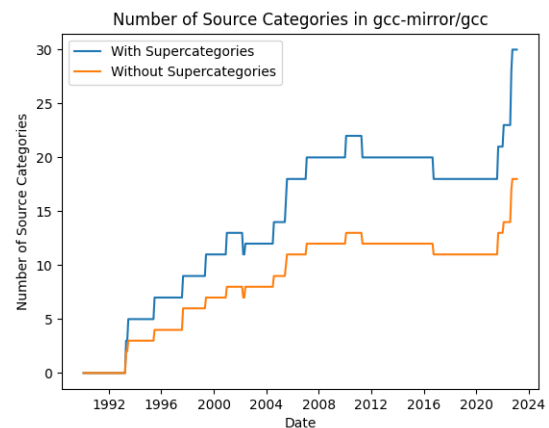
	With Recovered	Without Recovered
Total Number of Unique Sources	915	488
Number of Source Categories	32	26

TABLE 7.3: Descriptive Statistics of GCC

We observe that the GCC project presented an empty documentation landscape during its first years of development until 1993 (Figure 7.11). The number of categories that the documentation landscape covers has increased gradually through the years with a spike in 2022-2023 (Figure 7.12b), while the number of documentation sources peaks early on and then oscillates between 600 and 800 unique sources (Figure 7.12a).



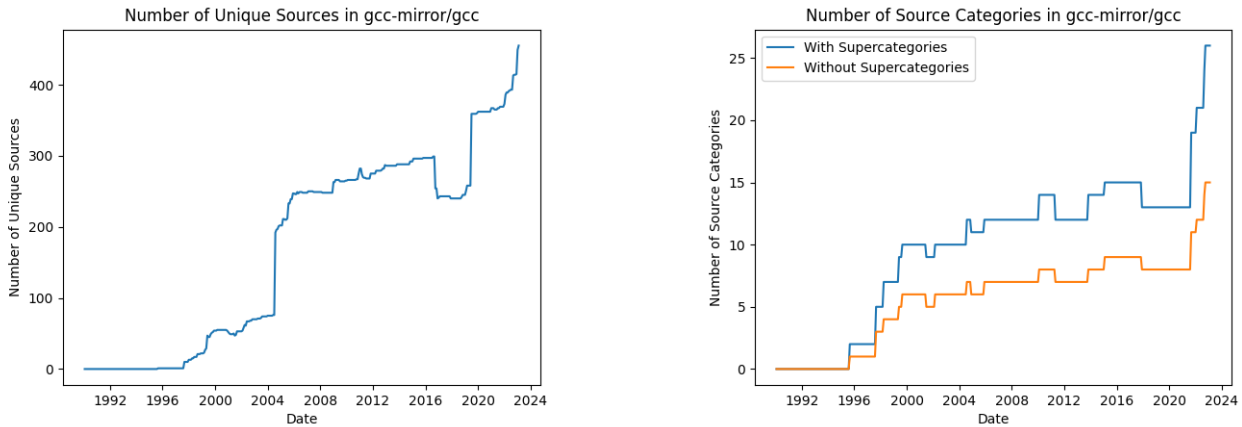
(A) Unique Documentation Sources over Time for GCC



(B) Unique Source Categories over Time of GCC

FIGURE 7.11: Documentation Landscape over Time of GCC

We also present that, when we remove sources that we recover, the documentation landscape shrinks dramatically in size, while it still contains a similar number of categories over time. From this, given that the number of categories is not too different, we can begin to assume that the recovered sources are bloating the documentation landscape of the project.



(A) Unique Documentation Sources over Time for GCC without Recovered Sources (B) Unique Source Categories over Time of GCC without Recovered Sources

FIGURE 7.12: Documentation Landscape over Time of GCC without Recovered Sources

### 7.3.2 Initial State

The GCC project presented README files, but it did not have any documentation source during its first three years of maintenance.

In 1993 we see the first documentation sources of GCC in the file `gcc/README.ALTOS` (Figure 7.13a). It contains only four sources, which we were not able to verify via pinging, but that we find through our recovery step. At this point in time, we find that the initial sources were one source file and three occurrences of specific subsection. The documentation sources contained in `gcc/README.ALTOS` are:

- `jkp@sauna.hut.fi`
- `info-gcc@prep.ai.mit.edu`
- `jkp@cs.hut.fi`
- `os/exec.c`

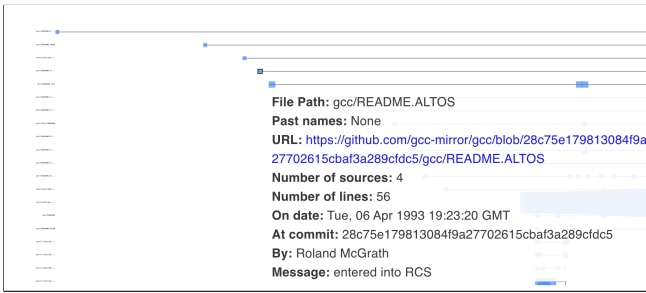
In this case, upon manual inspection, we can see that the file content contains e-mail addresses that are not detectable via pinging. One of these is a mailing list, but we are not able to find it with the address alone. We do not consider e-mail addresses as documentation sources in our taxonomy. Therefore, in reality, there is only one source file as a documentation source at this point in time. In addition, because these are sources that we recover, and that we do not identify directly, we also look at the landscape without recovered sources, giving us an empty landscape (Figure 7.1).

In 1997, we notice a large spike in the number of documentation sources, with a total of 386. Despite this rapid growth (that is only at its beginning), the documentation landscape presents hardly any new categories (Figure 7.15). We find various file types, and we see a drastic increase in homepages and source files.

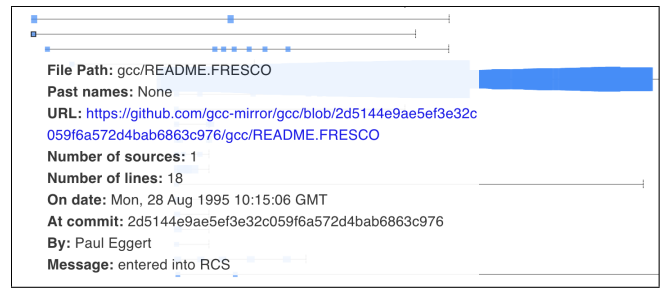
In fact, when we exclude the documentation sources that we recover, we can observe in 1995 the very first actual documentation source of the documentation landscape of GCC (Figure 7.16). With this, we find the very first source in the `gcc/README.FRESCO` file pointing to `http://www.faslab.com/fresco/HomePage.html`, a dead documentation source on a currently for-sale domain (Figure 7.13b).

We now established that the documentation sources that we recover bloat the documentation landscape of the project. For this reason, we are going to strictly focus on the landscape that includes sources that we could clearly identify as either live or dead during the mining process.





(A) First Appearance of Documentation Sources for GCC



(B) First Real Documentation Source in the GCC Project

FIGURE 7.13: SourceForge and Bugzilla Appearances in the Landscape of GCC

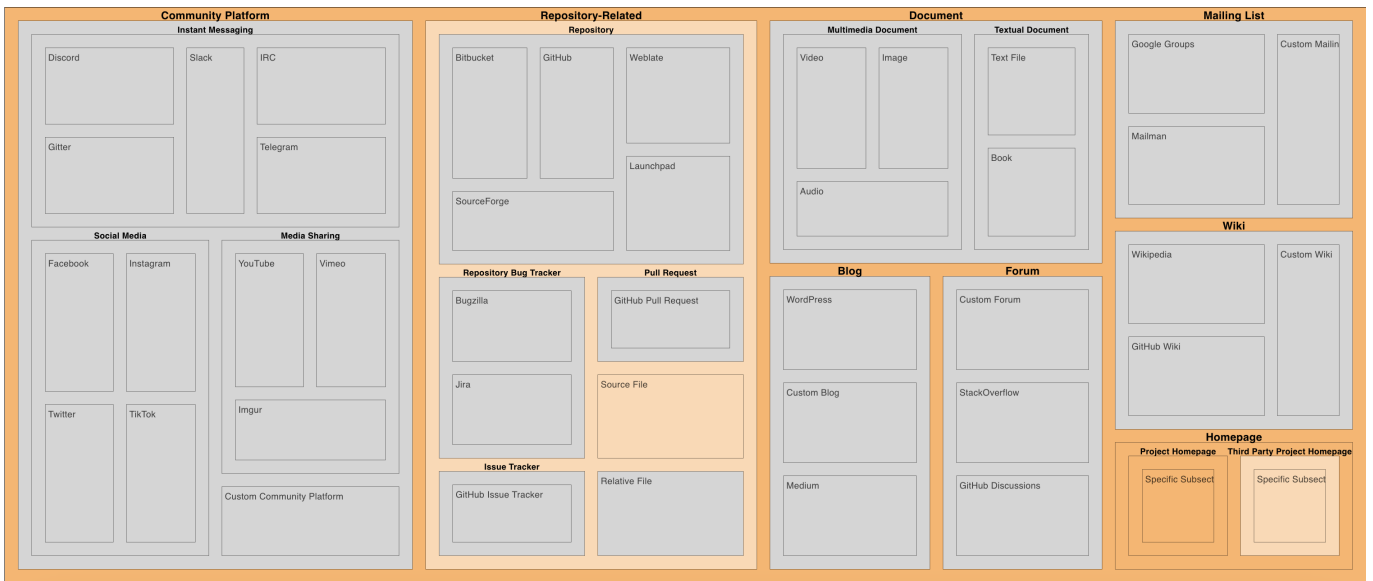


FIGURE 7.14: GCC in 1993

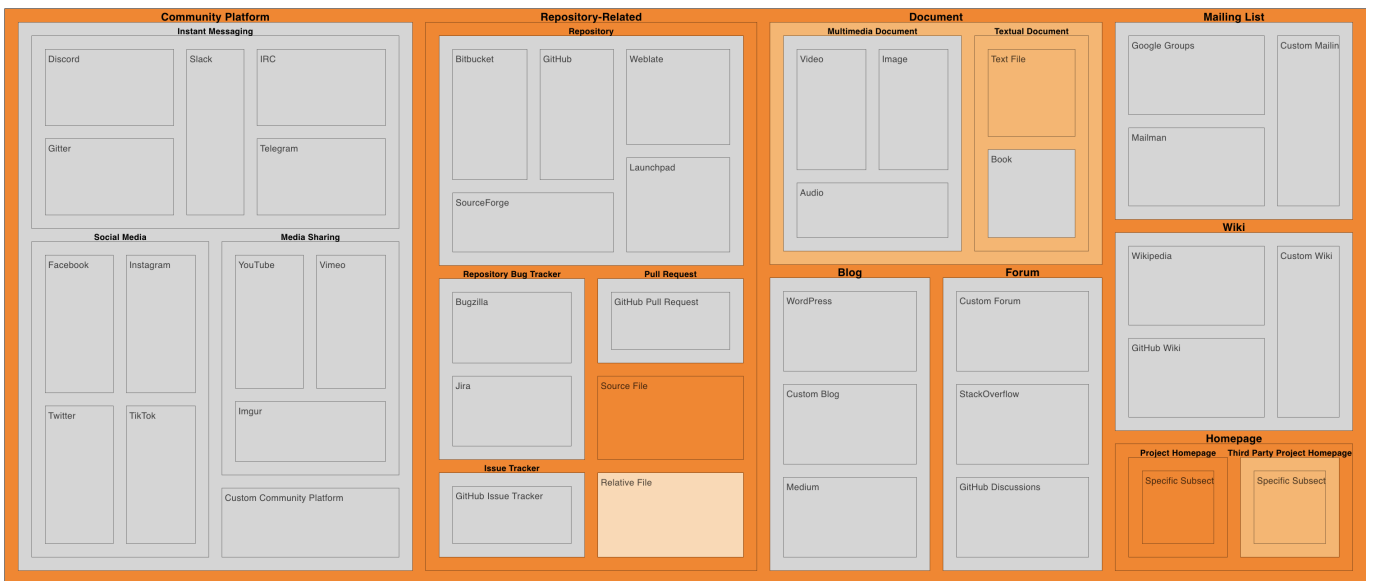


FIGURE 7.15: GCC in 1997

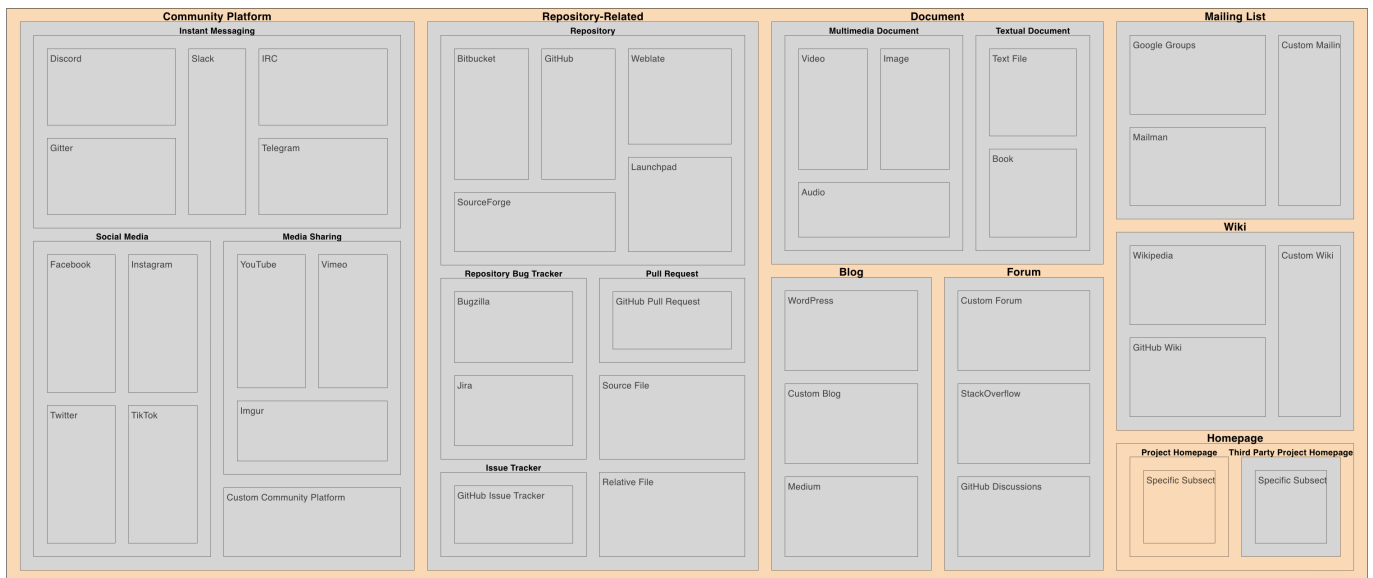


FIGURE 7.16: GCC in 1995 without Recovered Sources

### 7.3.3 First Documentation Sources

Our next main points of interest are present between the years 2000 and 2004, where we can observe a gradual and steep growth in documentation sources, respectively (Figure 7.12).

In fact, with a total of 54 documentation sources in the landscape, we can find a new README: `zlib/README` in 1999 (Figure 7.17). The file contains 10 sources (31 if we count recovered sources), one of which looks to be the first mailing list that is explicitly mentioned in the landscape (Figure 7.19). Text files are now mentioned as well.

Upon closer investigation in the `zlib/README` file, we find that the “mailing list” that we were referring to points to a now dead URL: <http://web2.airmail.net/markn/articles/zlibtool/zlibtool.htm>. Satisfying the criteria for a custom mailing list, it is categorized as one due to our priority rules for conflict solving (Section 4.1). From a direct excerpt of the README file:

*Mark Nelson <markn@tiny.com> wrote an article about zlib for the Jan. 1997 issue of Dr. Dobb's Journal; a copy of the article is available in <http://web2.airmail.net/markn/articles/zlibtool/zlibtool.htm>*

We find that, in reality, this URL used to point to an article. Only one year later, we identify the first occurrence of SourceForge being mentioned in the landscape. The file `fastjar/README` appears (Figure 7.18a) and quotes its own project on SourceForge, that is still alive to this day<sup>11</sup>.

In 2004, we find the first occurrence of Bugzilla being mentioned in the README file: `libjava/README` (Figure 7.18b). The commit message reveals that obsolete information was removed from the README file. In a prior version, the content of the README used to read:

*Please submit bug reports via this URL: <http://gcc.gnu.org/cgi-bin/gnatsweb.pl?database=gcc>*

And it was updated to reference Bugzilla for bug reports instead:

*Please submit bug reports via this URL: <http://gcc.gnu.org/bugzilla>*

<sup>11</sup><https://fastjar.sourceforge.net/>

We can also notice how the false positive mailing list disappears from the landscape. In addition to Bugzilla, despite it not appearing in the landscape, we have reason to believe that IRC was already present in the landscape of GCC in 2002 at its earliest. Our heuristics may fail to capture this, or it may not be referenced in README files. But thanks to the internet archive, we are able to find the earliest version of the IRC node for GCC.<sup>12</sup> Together with IRC existing in 2002, the wayback machine also allows us to see that mailing lists were indeed used already in 2000.<sup>13</sup> Some of these relevant documentation sources are not quoted in README files, they are instead present on the main website of the project and, therefore, not easily identifiable.

In its first decade of life, GCC presents a rather stable landscape, with very few new documentation source types appearing.

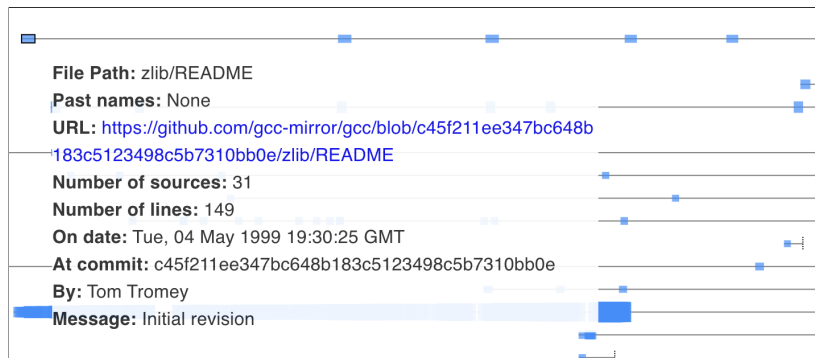
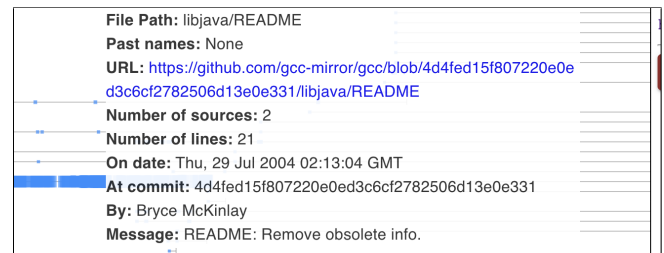


FIGURE 7.17: First Occurrence of a Mailing List in the Landscape



(A) First Occurrence of SourceForge in the Landscape



(B) First Occurrence of Bugzilla in the Landscape

FIGURE 7.18: SourceForge and Bugzilla Appearances in the Landscape of GCC

<sup>12</sup><http://web.archive.org/web/20020720151422/irc://irc.oftc.net/>

<sup>13</sup><http://web.archive.org/web/20000817013350/https://gcc.gnu.org/lists.html>

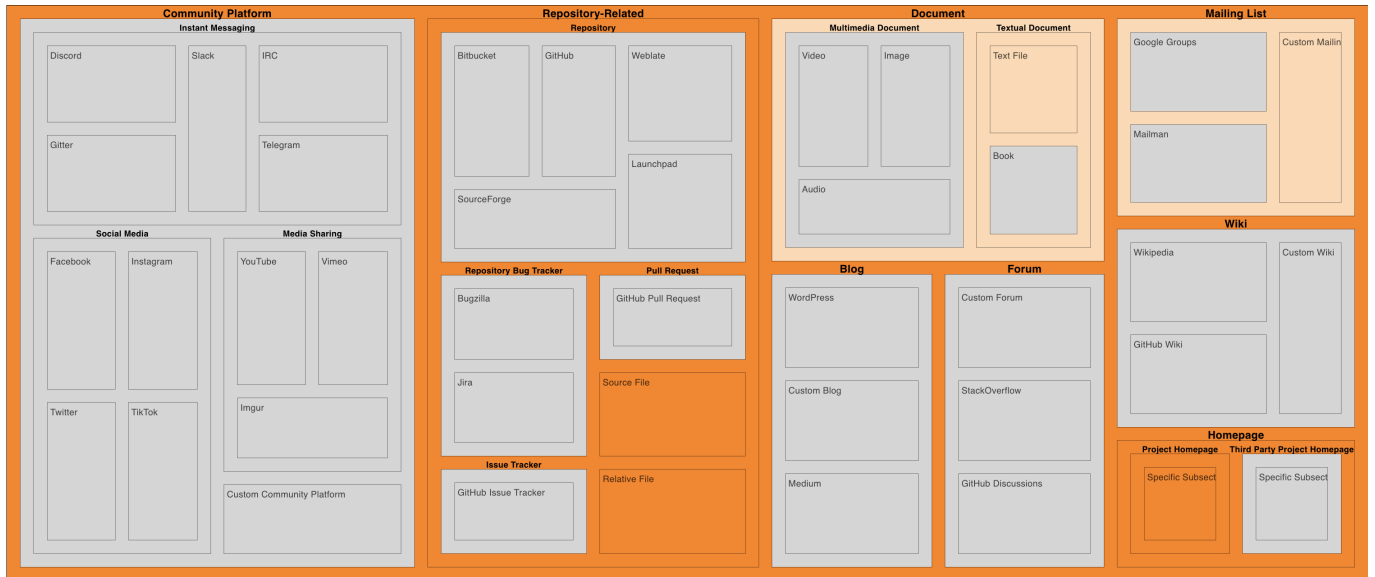


FIGURE 7.19: GCC in 1999

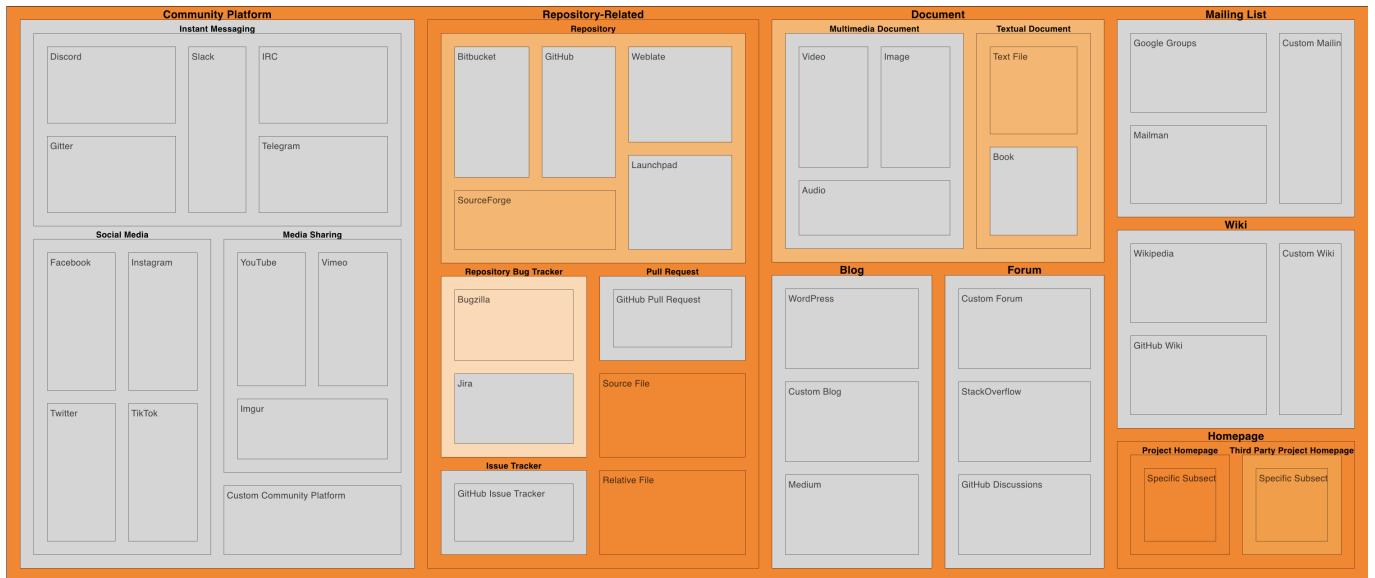


FIGURE 7.20: GCC in 2004

### 7.3.4 Modern Landscape

The documentation landscape remains stable without any radical changes in the categories that it covers until 2010, where a custom wiki is referenced in the file `libffi/README` (Figure 7.21a).

In 2011, the only occurrence of Bugzilla that existed in the landscape since it appeared in 2004 is gone. In fact, we can see it in the commit message itself (Figure 7.21b). The README now points to the generic bug reporting page. In a version of this page from February 21st 2011<sup>14</sup>, we notice how the bug reporting page points to the GCC bug database. Bugzilla is still in use at the time, it disappears from the documentation landscape of the project because it is no longer referenced on the repository. Instead, it is indirectly referenced by the official project website.

<sup>14</sup><http://web.archive.org/web/20110221073604/http://gcc.gnu.org:80/bugs/>

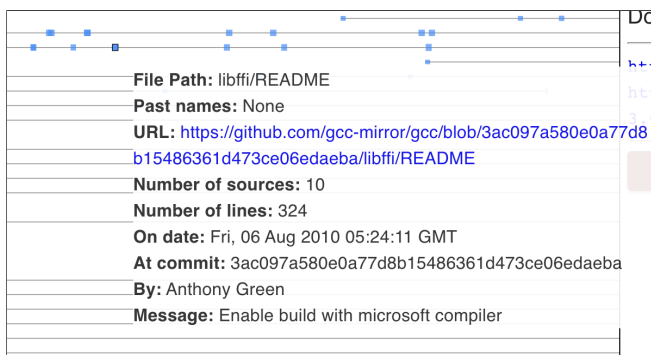
In 2013, a custom forum appears in the README: `libcilkrts/README` (Figure 7.22a). The source in question points to the forums for a new library that was added to GCC at <http://software.intel.com/en-us/forums/intel-cilk-plus/>. GitHub is referenced for the first time in 2015, in the `libffi/README` (Figure 7.22b), referencing the `libffi` project.<sup>15</sup> The README mentions that details on a merge can be seen at the git log of this project. In 2018, the `libcilkrts/README` file is deleted and, with it, the only occurrence of a forum in the documentation landscape so far.

In 2021, we see three new source categories rise all in the same commit: GitHub Issue Tracker, Custom Forum and Mailman. The old `libffi/README` file has been updated to a new markdown file format `libffi/README.md` with new documentation sources that are explicitly mentioned. We see the issue tracker of the `libffi` project<sup>16</sup>, along with two separate mailing lists:

- <https://sourceware.org/mailman/listinfo/libffi-announce>
- <https://sourceware.org/mailman/listinfo/libffi-discuss>

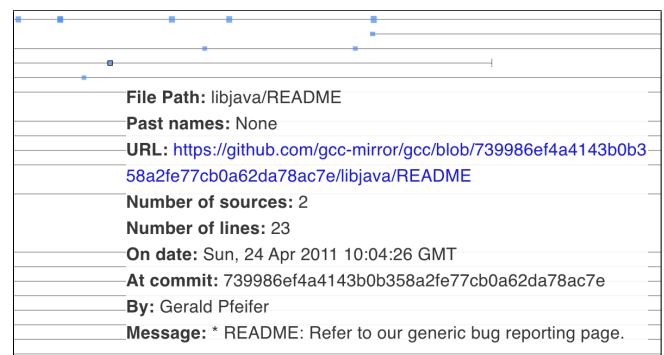
The second of these was erroneously categorized as a custom forum due to the word “discuss” being present in the name of the mailing list. On top of the GCC mailing list, which we do not directly find in README files, we now have clear lists referenced within the landscape.

GCC has existed over the course of three decades. Throughout its lifespan, it covers a rather small slice of the taxonomy, and the size of the landscape begins growing only past the 2000s. Especially in recent times, we observe a behaviour similar to `scikit-learn`, where the documentation landscape “explodes” with new documentation sources, albeit in a much more contained matter. Older projects, such as GCC and the Apache projects, tend to have their own website where sources are organized and mentioned there, rather than in a README file. Because of this, the landscape can be harder to determine. Many of the sources that we mention during the evolution of GCC come from README files of libraries, not belonging to the project itself. In fact, the GCC project originally had already 60 pages of written documentation when it was first created. This documentation is contained in many `.texi` files<sup>17</sup> that must be processed in order to be readable. Because of this, we could say that the documentation landscape of GCC has always existed, but it was not easily traceable via README files.



File Path:	libffi/README
Past names:	None
URL:	<a href="https://github.com/gcc-mirror/gcc/blob/3ac097a580e0a77d8b15486361d473ce06edaeba/libffi/README">https://github.com/gcc-mirror/gcc/blob/3ac097a580e0a77d8b15486361d473ce06edaeba/libffi/README</a>
Number of sources:	10
Number of lines:	324
On date:	Fri, 06 Aug 2010 05:24:11 GMT
At commit:	3ac097a580e0a77d8b15486361d473ce06edaeba
By:	Anthony Green
Message:	Enable build with microsoft compiler

(A) First Occurrence of a Wiki in the Landscape



File Path:	libjava/README
Past names:	None
URL:	<a href="https://github.com/gcc-mirror/gcc/blob/739986ef4a4143b0b358a2fe77cb0a62da78ac7e/libjava/README">https://github.com/gcc-mirror/gcc/blob/739986ef4a4143b0b358a2fe77cb0a62da78ac7e/libjava/README</a>
Number of sources:	2
Number of lines:	23
On date:	Sun, 24 Apr 2011 10:04:26 GMT
At commit:	739986ef4a4143b0b358a2fe77cb0a62da78ac7e
By:	Gerald Pfeifer
Message:	* README: Refer to our generic bug reporting page.

(B) Disappearance of Bugzilla from the Landscape

FIGURE 7.21: Wiki Appearance and Bugzilla Disappearance in the Landscape of GCC

<sup>15</sup><http://github.com/atgreen/libffi>

<sup>16</sup><https://github.com/libffi/libffi/issues>

<sup>17</sup><https://github.com/gcc-mirror/gcc/tree/master/gcc/doc>

<b>File Path:</b> libcilkrts/README	<b>File Path:</b> libffi/README
<b>Past names:</b> None	<b>Past names:</b> None
<b>URL:</b> <a href="https://github.com/gcc-mirror/gcc/blob/3038054c687e9400976012eea70333db70ad6a7b/libcilkrts/README">https://github.com/gcc-mirror/gcc/blob/3038054c687e9400976012eea70333db70ad6a7b/libcilkrts/README</a>	<b>URL:</b> <a href="https://github.com/gcc-mirror/gcc/blob/b1760f7f915a36ee9b4636fb54719c9b3ae59356/libffi/README">https://github.com/gcc-mirror/gcc/blob/b1760f7f915a36ee9b4636fb54719c9b3ae59356/libffi/README</a>
<b>Number of sources:</b> 5	<b>Number of sources:</b> 12
<b>Number of lines:</b> 85	<b>Number of lines:</b> 451
<b>On date:</b> Tue, 29 Oct 2013 18:37:47 GMT	<b>On date:</b> Mon, 12 Jan 2015 16:19:59 GMT
<b>At commit:</b> 3038054c687e9400976012eea70333db70ad6a7b	<b>At commit:</b> b1760f7f915a36ee9b4636fb54719c9b3ae59356
<b>By:</b> Balaji V. Iyer	<b>By:</b> Richard Henderson
<b>Message:</b> Added Cilk runtime library (libcilkrts) into GCC.	<b>Message:</b> Merge libffi to upstream commit
	c82cc159426d8d4402375fa1ae3f045b9cf82e16

(A) First Occurrence of a Forum in the Landscape

(B) First Occurrence of Github in the Landscape

FIGURE 7.22: Forum and Github Appearances in the Landscape of GCC

Especially with the GCC project, we have been able to see how our tool is limited and can completely fail at capturing the documentation landscape of certain projects. GCC is an extremely large project that had already established its documentation ahead of its versioning in Git. Because of this, we were not able to uncover what lay underneath. Instead, we have seen the evolution of the libraries that GCC employed via their README files. The project’s landscape, due to this hidden nature, is undetected in our tool and, instead, we mostly see the evolution via the README of the libraries it uses.

## 7.4 Limitations

Our thesis presents many limitations. These limitations can be external (Section 7.4.1), as they come from the source of our data and we cannot control it, or internal (Section 7.4.2), as they depend on our implementation and approach.

### 7.4.1 Hic Sunt Leones

The data that we extracted from GitHub presents a large amount of arbitrary content. The content of README files is human-generated, making it unpredictable and non-trivial to parse accurately. What we present is the result of curated data that is still far from perfect.

We previously specified (Section 5.2) that our heuristics for README files were limited and would not be able to capture all possible README names. Despite that, we are able to identify 4,117 unique file extensions for README files, only four of which actually represent a meaningful percentage in the dataset (Table 7.4). Because of this, we focus on markdown and reStructuredText files, while treating any other extension as a plain text file.

Extension	Occurrence	Percentage
.md	601,217	74.38%
No Extension	88,894	11.00%
.txt	55,524	6.87%
Others	52,565	6.50%
.rst	10,091	1.25%
Total	808,291	100.00%

TABLE 7.4: Occurrences of README Extensions Across All README Histories

### 7.4.2 Threats to Validity

While other extensions in total represent roughly 6% of the dataset, there is no extension in that category that occurs more than 1% of the times. Many times, these extensions appear in the order of magnitude of single digits, making them more unique than rare. Such as the `.luigi` extension with three occurrences.

This only affects file extensions, but it is already a good example that really represents how arbitrary human-generated content can be. If the extensions already show anomalies at this level, the content of a textual file has the potential to be far more complex.

We use the content of README files to find potential documentation sources that we then ping to determine their status. Due to the arbitrary nature of the content, this can be insufficient. There are cases in which URLs are not typed in their entirety (e.g., `example.com` instead of `https://example.com`). This simple case we can deal with by prepending the necessary keywords.

But this is not all, we can find source files that are referenced that do not exist on the project repository and that require context to know what they point to. Human error or README files that were not updated can cause relative URLs to point somewhere where the file does not exist anymore. Even a fully functioning URL cannot be guaranteed to be a documentation source; the URL could point to a domain that has been hijacked or taken, or even the URL could be alive and well, but it could point to social media page that has been inactive and, effectively, dead for a long time.

To curate this, we introduced the concept of “recovered” sources. This, too, is a limited approach. It scans the sources that we rejected during the pinging step and, if it satisfies some more lax requirements, we recover them. Recovered sources, other than opening the data to more false positives, also have no way to confirm whether they are live, dead, or non-documentation sources without manual intervention. We can see how this can put a bias on the landscape in the GCC case study (Section 7.3). We add an option to view the landscape without the recovered sources for this reason. On one hand, we can observe a landscape that is saturated by potential rejects, while on the other we observe a more curated but possibly incomplete landscape.

The manual inspection that we used to generate the taxonomy might not be sufficient to develop a true complete taxonomy. This is because we put emphasis on those documentation sources that the taxonomy would fail to capture during the iterative process. If we found uncategorized sources, we would modify the taxonomy to include them. Because of this, did not consider false positives for category detection as much. In particular, the Homepage category has a very broad detection method, and some of the uncategorized sources may wrongfully fall under this category.

## 7.5 Conclusions

This chapter outlines how our approach and tool allow us to map and view the documentation landscape of software repositories. We are able to see what documentation sources appear and disappear over time. The *scikit-learn* project shows us clearly how our mapping works when the landscape of a project contains a small number of documentation sources, and when the most important ones appear only once in the top-level README. We are also able to see how our approach can fail when it comes to particularly tricky cases. The GCC project does not present many documentation sources that we can find via our taxonomy. The ones we do identify do not even belong to the landscape of the project, but are instead found in the README files of libraries that the project uses. Having seen how our approach can succeed and fail, we can now draw our conclusions on our thesis.





## Chapter 8

# Conclusions

In this chapter, we summarize our contributions and propose possible avenues for future work.

This thesis presents an evolutionary analysis of the content of README files in open-source projects, and a taxonomy that maps documentation sources to the landscape that they constitute. From our analysis, we can observe that the documentation landscape has been expanding during the past decades. Despite some sources being stable, we have been able to capture the rise of many new types of platforms. Following this pattern, we can hypothesize that the documentation landscape will keep expanding as it has been so far. But this is not the only outlook on the future of the documentation landscape. Robillard et al. have introduced the concept of on-demand developer documentation [47]. With the advent of advanced large language models such as ChatGPT <sup>1</sup>, documentation can be generated on the fly, with little to no effort. Something of the sort may also push the documentation landscape backwards, causing it to shrink and implode as documentation sources become less useful when actually usable on-demand documentation will rise.

### 8.1 Contributions

In this thesis, we provide an approach and tool to mine, process, and visualize the evolution of README files and the documentation landscape of software repositories. We map the documentation landscape of projects and offer a comprehensive taxonomy of its documentation sources that compose it. We use this mapping to reconstruct the documentation landscape of a project from its creation and view the evolution of its documentation sources. We summarize our contributions as follows:

- **Mining of Evolutionary Data of the README Files of Git Repositories and Their Documentation Sources**

We reconstruct the history of README files stored in a Git repository by traversing the project's commits. With the commits, we focus on those that specifically modified a README file to extract information about the commit and its sources. We use these data model and map the documentation landscape through our taxonomy.

- **Approach to Trace and Chain the Histories of README Files via Mimir**

We can build the history of each unique README file within a Git repository. We create Mimir to oversee the repository's README histories, and we improve this history reconstruction step via history chaining. Files on a Git repository can be renamed, moved, or duplicated. In these cases, history chaining allows us to link different README histories that represent the same file even when their filenames differ.

- **Comprehensive Taxonomy of the Documentation Landscape**

---

<sup>1</sup><https://chat.openai.com/>

We present a taxonomy of the documentation landscape that categorizes the documentation sources across eight main categories that distinguish themselves in their attributes. With this taxonomy we are able to outline the documentation landscape of software systems and view its evolution over time.

## 8.2 Future Work

### Introduce Aging to READMEs and Sources

Through an improved evolutionary modeling of the system, we could better represent the aging of a README file and of the documentation sources. With aging, we would be able to determine how old a documentation source is. This has the potential to lead to active monitoring of live sources, to determine if they present any changes (*e.g.*, redirecting to a new domain, dying). These changes could tell us more about what happens to the documentation landscape of a system (*e.g.*, the system is renovating, the system is abandoned).

### Further Visualizations

Additional visualizations could help understand the documentation landscape of a software system with further ease. Such visualizations may include spider charts that summarize to which extent a documentation source appears in the landscape, or a multi-level doughnut chart that subdivides the landscape into sectors for each category, adding a layer for each level of the hierarchy that we reach. With these visualizations, we can compare the documentation landscape of projects in a much more compact way, leading into further investigation on the health of its documentation sources. For instance, we could see right away if a project contains a specific type of documentation source, and how much of the landscape that source occupies.

### Improve Storage

The processed data is stored in `pickle`<sup>2</sup> files. On top of not being a secure file, these files can be problematic when they are used to store objects that evolve and change over time, failing to open when a feature they used has been removed, for instance. A more backward-compatible approach would greatly improve the speed at which the model can be developed.

In addition, SQLite does not scale well when it comes to many concurrent accesses. The SQLite database has been a synchronization point during the mining. Switching to a database server that is built to deal with high levels of concurrency could help prevent the bottle neck that the SQLite database has been during the mining process.

### Improve the Chaining Algorithm

The chaining algorithm for history chaining is extremely computationally expensive. A large number of histories with particularly large files can slow it down significantly. Optimizing this algorithm could speed up mining drastically. The chaining algorithm uses text similarity to chain README histories. By setting a threshold, the algorithm decides if a pair of histories must be chained. Evaluating the algorithm by manually establishing a ground truth would determine its accuracy and would allow for fine-tuning of the threshold for best results. Furthermore, experimenting with other techniques to determine text similarity (*e.g.*, cosine similarity, language models), could further improve the algorithm in accurately chaining the histories.

---

<sup>2</sup><https://docs.python.org/3/library/pickle.html>

## Introduce Formal Concept Analysis

“Formal concept analysis has been developed as a field of applied mathematics based on the mathematization of concept and concept hierarchy. It thereby allows us to mathematically represent, analyze, and construct conceptual structures.” [12]

By introducing formal concept analysis to the taxonomy and its categories, we can produce concepts that describe the categories of the taxonomy. Through this, we can derive attributes of the documentation source categories to better describe the documentation landscape. These attributes can allow us to explore more deeply what characteristics of the sources are best-suited for developers. We have already observed a shift from more static, slow sources to faster, more volatile ones. If we could investigate the more popular attributes, we could find a medium to better support developers and the documentation that they produce.

## 8.3 Final Words

In this thesis, we presented an approach to explore the documentation landscape of software systems. Through the evolution of their README files, we are able to better understand what kinds of sources developers use to generate documentation. In these sources, we find a lack of standard in the way that they are presented, and this made this thesis a perilous voyage through caveats, exceptions and trade-offs. By producing a taxonomy of the documentation landscape, we believe that it will benefit the understanding of how documentation is used and evolves over time. We ultimately aim at shaping the future of a more structured and automatically explorable documentation landscape.



## Appendix A

# SCC and CLOC Comparison

CLOC is consistently outperformed by SCC, with the number of comments not differing too much between in most of the cases (Table A.1).

Repository Name	CLOC Output	SCC Output	CLOC Time	SCC Time
scikit-learn/scikit-learn	124,729	40,654	2.12s	0.07s
tensorflow/tensorflow	851,187	679,415	27.14s	0.7s
facebook/react	43,336	43,372	1.4s	0.09s
angular/angular.js	38,066	37,624	2.32s	0.04
google/guava	129,398	186,743	2.99s	0.08s
apache/spark	464,610	357,699	13.81s	0.26s
arduino/Arduino	128,803	73,804	0.78	0.03s
apache/tomcat	194,728	188,052	3.15s	0.06s
jquery/jquery	4,764	4,899	1.30s	0.01s
spring-projects/spring-boot	192,772	186,394	8.57s	0.13s
netty/netty	113,370	113,237	2.42s	0.04s
elastic/elasticsearch	364,733	346,224	20.91s	0.29s
square/okhttp	20,663	20,616	0.46s	0.021s
vuejs/vue	5,265	5,178	0.35s	0.01s
freeCodeCamp/freeCodeCamp	4,991	3,546	24.00s	0.57s
flutter/flutter	236,231	238,214	4.53s	0.11s
torvalds/linux	4,228,780	4,137,871	1m 16.88s	2.27s
TheAlgorithms/Python	37,523	25,139	1.01s	0.02s
microsoft/vscode	126,482	127,893	14.82s	0.10s
ohmyzsh/ohmyzsh	5,628	5,276	0.35s	0.02s

TABLE A.1: CLOC and SCC Output Comparison



## Appendix B

# Inconsistency from GitHub Search

We summarize the projects that cannot be found on GitHub or that are forks in our dataset (Table B.1). We have 31 total projects, 23 of which do not exist and 8 of which are forks. The inconsistencies can manifest when the database of GitHub Search is outdated or when an alias is used to call the GitHub API. Find more information at <https://github.com/seart-group/ghs/issues/166>.

Project Name	Exclusion Reason
ADBSQL/ AntDB	Does not Exist
XLabsProject/iw4x-client	Does not Exist
bitnami/bitnami-docker-kafka	Does not Exist
bitnami/bitnami-docker-laravel	Does not Exist
bitnami/bitnami-docker-mariadb	Does not Exist
bitnami/bitnami-docker-mongodb	Does not Exist
bitnami/bitnami-docker-moodle	Does not Exist
bitnami/bitnami-docker-nginx	Does not Exist
bitnami/bitnami-docker-postgresql	Does not Exist
bitnami/bitnami-docker-redis	Does not Exist
bitnami/bitnami-docker-redmine	Does not Exist
bitnami/bitnami-docker-wordpress	Does not Exist
burtonator/polar-bookshelf	Does not Exist
chinnkarahoi/jd_scripts	Does not Exist
freedesktop/poppler	Does not Exist
getferdi/ferdi	Does not Exist
glimpse-editor/Glimpse	Does not Exist
hove-io/mimirsbrunn	Does not Exist
taigaio/taiga-back	Does not Exist
wappalyzer/wappalyzer	Does not Exist
wayland-project/wayland	Does not Exist
wayland-project/weston	Does not Exist
wix/ricos	Does not Exist
WorldDBs/lotus	Fork
anurodhp/VaxProj	Fork
github/super-linter	Fork
iDevision/enhanced-discord.py	Fork
krassowski/jupyterlab-lsp	Fork
pliablepixels/zmNinja	Fork
qaprosoft/carina	Fork
terraform-providers/terraform-provider-oci	Fork

TABLE B.1: List of Projects Excluded from the Analysis





# Bibliography

- [1] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. Software Documentation Issues Unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1199–1210. IEEE, 2019.
- [2] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C. Shepherd. Software Documentation: The Practitioners’ Perspective. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 590–601. IEEE, 2020.
- [3] Emad Aghajani, Gabriele Bavota, Mario Linares-Vásquez, and Michele Lanza. Automated Documentation of Android Apps. *IEEE Transactions on Software Engineering*, 47(1):204–220, 2021.
- [4] Navid Ahmadi, Mehdi Jazayeri, Francesco Lelli, and Sasa Nesic. A Survey of Social Software Engineering. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering - Workshops*, pages 1–12. IEEE, 2008.
- [5] Christian Bird, Peter C. Rigby, Earl T. Barr, David J. Hamilton, Daniel M. German, and Prem Devanbu. The Promises and Perils of Mining Git. In *2009 6th IEEE International Working Conference on Mining Software Repositories*, pages 1–10. IEEE, 2009.
- [6] Ned Chapin, Joanne E. Hale, Khaled Md. Khan, Juan F. Ramil, and Wui-Gee Tan. Types of Software Evolution and Software Maintenance. *Journal of Software Maintenance and Evolution: Research and Practice*, 13(1):3–30, 2001.
- [7] Preetha Chatterjee, Kostadin Damevski, Lori Pollock, Vinay Augustine, and Nicholas A. Kraft. Exploratory Study of Slack Q&A Chats as a Mining Source for Software Engineering Tools. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 490–501. IEEE, 2019.
- [8] Timothy Clem and Patrick Thomson. Static Analysis at GitHub: An Experience Report. *Queue*, 19(4): 42–67, sep 2021.
- [9] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. Sampling Projects in GitHub for MSR Studies. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories*, pages 560–564. IEEE, 2021.
- [10] Osama Ehsan, Safwat Hassan, Mariam El Mezouar, and Ying Zou. An Empirical Study of Developer Discussions in the Gitter Platform. *ACM Transactions on Software Engineering and Methodology*, 30(1), 2021.
- [11] Andrew Forward and Timothy C. Lethbridge. The Relevance of Software Documentation, Tools and Technologies: A Survey. In *Proceedings of the 2002 ACM Symposium on Document Engineering*, page 26–33. ACM, 2002.
- [12] Bernhard Ganter, Gerd Stumme, and Rudolf Wille. *Formal Concept Analysis: Foundations and Applications*, volume 3626. springer, 2005.

- [13] Tudor Adrian Girba. *Modeling history to understand software evolution*. PhD thesis, University of Bern, 2005.
- [14] Gillian J. Greene and Bernd Fischer. CVExplorer: Identifying Candidate Developers by Mining and Exploring Their Open Source Contributions. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, page 804–809. ACM, 2016.
- [15] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, and Arie van Deursen. Communication in Open Source Software Development Mailing Lists. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 277–286. IEEE, 2013.
- [16] Tudor Gîrba, Jean-Marie Favre, and Stéphane Ducasse. Using Meta-Model Transformation to Model Software Evolution. *Electronic Notes in Theoretical Computer Science*, 137(3):57–64, 2005.
- [17] Mark Handel and James D. Herbsleb. What is Chat Doing in the Workplace? In *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, page 1–10. ACM, 2002.
- [18] Foyzul Hassan and Xiaoyin Wang. Mining Readme Files to Support Automatic Building of Java Projects in Software Repositories. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 277–279. IEEE, 2017.
- [19] Claudia Hauff and Georgios Gousios. Matching GitHub Developer Profiles to Job Advertisements. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 362–366. IEEE, 2015.
- [20] James D. Herbsleb, David L. Atkins, David G. Boyer, Mark Handel, and Thomas A. Finholt. Introducing Instant Messaging and Chat in the Workplace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, page 171–178. ACM, 2002.
- [21] Jialun Aaron Jiang, Charles Kiene, Skyler Middler, Jed R. Brubaker, and Casey Fiesler. Moderation Challenges in Voice-Based Online Communities on Discord. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW), 2019.
- [22] Brian Johnson and Ben Shneiderman. Tree-maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures. In *Proceeding Visualization '91*, pages 284–291, 1991.
- [23] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The Promises and Perils of Mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, page 92–101. ACM, 2014.
- [24] Andreas M. Kaplan and Michael Haenlein. Users of the World, Unite! The Challenges and Opportunities of Social Media. *Business Horizons*, 53(1):59–68, 2010.
- [25] Michele Lanza and Stephane Ducasse. Polymetric Views - A Lightweight Visual Approach to Reverse Engineering. *IEEE Transactions on Software Engineering*, 29(9):782–795, 2003.
- [26] Meir M. Lehman. On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle. *Journal of Systems and Software*, 1:213–221, 1979.
- [27] Meir M. Lehman. Programs, Life Cycles, and Laws of Software Evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980.
- [28] Meir M. Lehman and Laszlo A. Belady. *Program Evolution: Processes of Software Change*. Academic Press Professional, Inc., 1985.
- [29] Meir M. Lehman and Juan F. Ramil. An Approach to a Theory of Software Evolution. In *Proceedings of the 4th International Workshop on Principles of Software Evolution*, page 70–74. ACM, 2001.

- [30] Timothy C. Lethbridge, Janice Singer, and Andrew Forward. How Software Engineers Use Documentation: The State of the Practice. *IEEE Software*, 20(6):35–39, 2003.
- [31] Bin Lin, Alexey Zagalsky, Margaret-Anne Storey, and Alexander Serebrenik. Why Developers Are Slacking Off: Understanding How Software Teams Use Slack. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*, page 333–336. ACM, 2016.
- [32] Mariam El Mezouar, Feng Zhang, and Ying Zou. Are Tweets Useful in the Bug Fixing Process? An Empirical Study on Firefox and Chrome. *Empirical Software Engineering*, 23(3):1704–1742, 2018.
- [33] Alan S. Neal and Roger M. Simons. Playback: A Method for Evaluating the Usability of Software and Its Documentation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, page 78–82. ACM, 1983.
- [34] David Lorge Parnas. Software Aging. In *Proceedings of 16th International Conference on Software Engineering*, pages 279–287. IEEE, 1994.
- [35] David Lorge Parnas and Jan Madey. Functional Documents for Computer Systems. *Science of Computer Programming*, 25(1):41–61, 1995.
- [36] Chris Parnin, Christoph Treude, Lars Grammel, and Margaret-Anne Storey. Crowd Documentation: Exploring the Coverage and the Dynamics of API Discussions on Stack Overflow. Technical report, Georgia Institute of Technology, 2012.
- [37] Esteban Parra, Ashley Ellis, and Sonia Haiduc. GitterCom: A Dataset of Open Source Developer Communications in Gitter. In *Proceedings of the 17th International Conference on Mining Software Repositories*, page 563–567. ACM, 2020.
- [38] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. Prompter: A self-confident recommender system. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 577–580. IEEE, 2014.
- [39] Roxana Lisette Quintanilla Portugal and Julio Cesar Sampaio do Prado Leite. Extracting Requirements Patterns from Software Repositories. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 304–307. IEEE, 2016.
- [40] Gede Artha Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. Categorizing the Content of GitHub README Files. *Empirical Software Engineering*, 24(3):1296–1327, 2019.
- [41] Marco Raglianti. Topology of the Documentation Landscape. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 297–299. ACM, 2022.
- [42] Marco Raglianti, Roberto Minelli, Csaba Nagy, and Michele Lanza. Visualizing Discord Servers. In *2021 Working Conference on Software Visualization (VISSOFT)*, pages 150–154. IEEE, 2021.
- [43] Marco Raglianti, Csaba Nagy, Roberto Minelli, Bin Lin, and Michele Lanza. On the Rise of Modern Software Documentation. In *37th European Conference on Object-Oriented Programming (ECOOP 2023)*. Dagstuhl: Schloss Dagstuhl, 2023.
- [44] Juan F. Ramil and Meir M. Lehman. Metrics of Software Evolution as Effort Predictors - A Case Study. In *Proceedings 2000 International Conference on Software Maintenance*, pages 163–172. IEEE, 2000.
- [45] H. Rudy Ramsey, Michael E. Atwood, and James R. Van Doren. A Comparative Study of Flowcharts and Program Design languages for the detailed procedural specification of Computer Programs. Technical report, Science APPLICATIONS INC ENGLEWOOD CO, 1978.

- [46] Martin P. Robillard. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software*, 26(6):27–34, 2009.
- [47] Martin P. Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro, Neil Ernst, Marco Aurélio Gerosa, Michael Godfrey, Michele Lanza, Mario Linares-Vásquez, Gail C. Murphy, Laura Moreno, David Shepherd, and Edmund Wong. On-demand Developer Documentation. In *2017 IEEE International Conference on Software Maintenance and Evolution*, pages 479–483, 2017.
- [48] Marc J. Rochkind. The Source Code Control System. *IEEE Transactions on Software Engineering*, SE-1(4):364–370, 1975.
- [49] Nayan B. Ruparelia. The History of Version Control. *SIGSOFT Software Engineering Notes*, 35(1):5–9, jan 2010.
- [50] Hareem Sahar, Abram Hindle, and Cor-Paul Bezemer. How are Issue Reports Discussed in Gitter Chat Rooms? *Journal of Systems and Software*, 172:110852, 2021.
- [51] Vitalis Salis and Diomidis Spinellis. RepoFS: File System View of Git Repositories. *SoftwareX*, 9:288–292, 2019.
- [52] Sylvia B. Sheppard, Elizabeth Kruesi, and John W. Bailey. An Empirical Evaluation of Software Documentation Formats. In *Proceedings of the 1982 Conference on Human Factors in Computing Systems*, page 121–124. ACM, 1982.
- [53] Lin Shi, Xiao Chen, Ye Yang, Hanzhi Jiang, Ziyou Jiang, Nan Niu, and Qing Wang. A First Look at Developers’ Live Chat on Gitter. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, page 391–403. ACM, 2021.
- [54] Ben Shneiderman, Richard Mayer, Don McKay, and Peter Heller. Experimental Investigations of the Utility of Detailed Flowcharts in Programming. *Communications of the ACM*, 20(6):373–381, 1977.
- [55] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. PyDriller: Python Framework for Mining Software Repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 908–911. ACM, 2018.
- [56] Margaret-Anne Storey, Christoph Treude, Arie van Deursen, and Li-Te 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*, page 359–364. ACM, 2010.
- [57] Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. The (R)Evolution of Social Media in Software Engineering. In *Future of Software Engineering Proceedings*, page 100–116. ACM, 2014.
- [58] Viktoria Stray and Nils Moe. Understanding Coordination in Global Software Engineering: A Mixed-methods Study on the Use of Meetings and Slack. *Journal of Systems and Software*, 170:110717, 2020.
- [59] Robert C. Tausworthe. Standard Classification of Software Documentation. Technical report, NASA, 1976.
- [60] Christoph Treude and Martin P. Robillard. Augmenting API Documentation with Insights from Stack Overflow. In *2016 IEEE/ACM 38th International Conference on Software Engineering*, pages 392–403. IEEE, 2016.