

On the Rise of Modern Software Documentation

Marco Raglianti  

REVEAL @ Software Institute – USI, Lugano, Switzerland

Csaba Nagy  

REVEAL @ Software Institute – USI, Lugano, Switzerland

Roberto Minelli  

REVEAL @ Software Institute – USI, Lugano, Switzerland

Bin Lin  

Radboud University, Nijmegen, the Netherlands

Michele Lanza  

REVEAL @ Software Institute – USI, Lugano, Switzerland

Abstract

Classical software documentation, as it was conceived and intended decades ago, is not the only reality anymore. Official documentation from authoritative and official sources is being replaced by real-time collaborative platforms and ecosystems that have seen a surge, influenced by changes in society, technology, and best practices. These modern tools influence the way developers document the conception, design, and implementation of software. As a by-product of these shifts, developers are changing their way of communicating about software. Where once official documentation stood as the only truth about a project, we now find a multitude of volatile and heterogeneous documentation sources, forming a complex and ever-changing *documentation landscape*.

Software projects often include a top-level README file with important information, which we leverage to identify their documentation landscape. Starting from $\sim 12\text{K}$ GitHub repositories, we mine their README files to extract links to additional documentation sources. We present a qualitative analysis, revealing multiple dimensions of the documentation landscape (*e.g.*, content type, source type), highlighting important insights. By analyzing instant messaging application links (*e.g.*, Gitter, Slack, Discord) in the histories of README files, we show how this part of the landscape has grown and evolved in the last decade.

Our findings show that modern documentation encompasses communication platforms, which are exploding in popularity. This is not a passing phenomenon: On the contrary, it entails a number of unknowns and socio-technical problems the research community is currently ill-prepared to tackle.

2012 ACM Subject Classification Software and its engineering → Collaboration in software development; Human-centered computing → Collaborative and social computing

Keywords and phrases software documentation landscape, GitHub README, instant messaging

Digital Object Identifier 10.4230/LIPIcs.ECOOP.2023.43

1 Introduction

Times are changing. This is even more true for software engineering. Major shifts have occurred, induced by the emergence of platforms like GitHub and StackOverflow, fundamentally changing how developers (and users) communicate about software projects: Mailing lists and forums are declining in favor of multi-media instant messaging platforms, such as Gitter, Slack, Discord, and GitHub Discussions, *e.g.*, [9, 18, 27, 30, 33, 45, 46, 49, 50, 56, 58, 66, 67].

Software documentation, a critical asset for developers [3], has been studied extensively with respect to its quality and usefulness [4, 10, 14, 19, 21, 54, 61, 79]. Nevertheless, the impact of the subtle but constant drift induced by new platforms is still to be evaluated. What are the implications for program comprehension if a tweet can influence how developers treat a



© Marco Raglianti, Csaba Nagy, Roberto Minelli, Bin Lin, and Michele Lanza;
licensed under Creative Commons License CC-BY 4.0

37th European Conference on Object-Oriented Programming (ECOOP 2023).

Editors: Karim Ali and Guido Salvaneschi; Article No. 43; pp. 43:1–43:24

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

45 bug [38]? Can the tweets on the usage of an API also serve as documentation? Classical
46 software documentation, as we have known it, is being replaced by “communication”.

47 Documentation went from a clunky, and rather unloved, endeavor to becoming a fast-paced
48 and volatile side dish. The utopia of “*on-demand documentation*” by Robillard *et al.* [55], is
49 being replaced by a dystopia of an ever-changing landscape; documentation is waved away
50 with sentences like “*check Discord*” or “*it’s in the pull request comments.*”

51 This change is more than just cosmetic, it is considerably affected by the richness of
52 new media, influencing the cognitive processes that underlie communication [53]. Modern
53 media-rich platforms offer vastly different mechanisms which are simply not there in classical
54 electronic communication means. Moreover, developers do not only hold ephemeral discussions
55 that they must be able to access now. They share knowledge (*e.g.*, code examples, screenshots,
56 howtos) that is important for them in the future, and they do not have (or rather: take) the
57 time to persist it in a classical software documentation form (*e.g.*, Wiki). Instant messaging
58 is just too enticing for that. But, they will need long-term access to this knowledge and want
59 to keep it searchable¹ [20] and organizable² [44]. They choose their platforms accordingly,
60 for example, avoiding limitations in retrievable history [2], and are willing to pay significant
61 sums for such services [12].

62 As the cards on documentation are being reshuffled, things seem murky: What happens
63 to the body of knowledge contained in the repositories of classical communication platforms?
64 What is the impact on standard software documentation? How do developers use modern
65 platforms, and what does this imply?

66 **The Spectrum Example.** *Spectrum, a multi-forum community hosting platform, was hosting dozens of software related communities about frameworks (e.g., React, Laravel), UI design (e.g., Figma), front-end coding (e.g., CodePen), and developers’ networks in general (e.g., SpecFM). On Aug 24, 2021, to preserve history while pushing forward the adoption of new communication infrastructures, it was announced that “the time has come for the planned archival of Spectrum to focus our efforts on GitHub Discussions” [36]. Spectrum has become “read-only – no 404s or lost internet history.” The Spectrum team acknowledged the importance of conversations held on the platform and tried to avoid the limitations of relying on the Internet Archive for preservation [71]. Many Spectrum communities had already moved to GitHub Discussions, for reliability and flexibility reasons: Having code and the community in the same place outweighed other factors in the decision to change.*

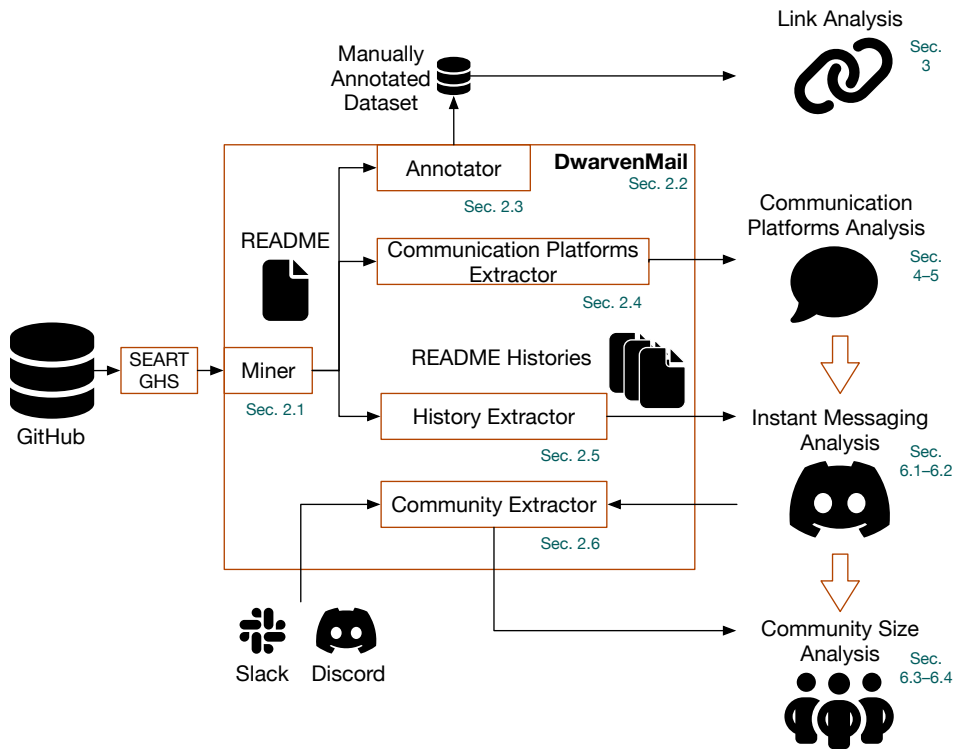
67 We present an overview of the *documentation landscape* (*i.e.*, a map of potential docu-
68 mentation sources) emerging from the analysis of ~12K GitHub projects. We explore
69 current trends in documentation platforms and the relationship between documentation and
70 communication platforms, exemplified by the tendency in a project’s README to include
71 the latter as an indirect source of the former.

72 We show the most representative values in different dimensions characterizing the land-
73 scape. We then proceed more in-depth with the history of modern communication platforms.
74 We show how some platforms have seen increasing adoption, reached a plateau, and finally
75 started their decline. Our analysis provides insights into the many implications of this
76 ongoing phenomenon for software documentation. Finally, we discuss possible features that
77 future platforms should have to mitigate some of the perils introduced by these continuous
78 shifts. We use the following icons to highlight salient points:

79  Insight  Idea/Future Work  Threat

¹ Especially for large communities, without limitations, as reported in this blog post.

² As demonstrated by the presence of an ecosystem built on top of instant messaging applications.



■ **Figure 1** DWARVENMAIL and Analyses Overview

2 Dataset Creation and DWARVENMAIL

This section details the procedure and tool support (DWARVENMAIL) we implemented to collect the data for our analyses (Figure 1). We present the initial dataset, the mining procedure, our manual annotation, and details about the individual analyses we performed.

2.1 Project Mining

Starting from all repositories currently hosted on GitHub we used SEART-GHS [13] to compose a relevant dataset, applying the following filtering criteria: at least 2,000 commits (*i.e.*, to eliminate toy projects), more than 10 contributors (*i.e.*, to ensure that a certain number of people need to interact with each other to tackle the development effort), and more than 100 stars (*i.e.*, to ensure that the projects are relevant to at least a handful of people). We only considered projects created before July 1, 2022 and excluded forks [22]. SEART-GHS currently monitors about 1.2M GitHub repositories. Projects excluded in SEART-GHS for having less than 10 stars [13] would have been excluded by the more restrictive criterion we applied, removing projects with less than 100 stars.



Filtering based on the number of stars might not be sufficient to select relevant projects. Nevertheless, starring can be important for project developers and managers [8]. We used this criterion as a common method for filtering out toy projects in GitHub (e.g., see [80]).

We performed the filtering on July 14, 2022, resulting in 12,461 projects exported as JSON input for DWARVENMAIL. After removing 374 aliases and 6 renamed forks, the final scraped dataset consists of 12,081 projects.

43:4 On the Rise of Modern Software Documentation

98 Table 1 presents an overview of the projects according to their languages. The *All* column
 99 shows the total number of projects, the *CP* column the projects where we could identify
 100 communication platforms (see Section 2.4), and the *IM* column the projects with instant
 101 messaging platforms. Percentages are derived with respect to the *All* column, while Δ
 102 percentages are relative to the *Total* row percentages.

103 Overall, 57.3% of the projects we analyzed feature at least one communication platform.
 104 An interesting observation is that systems written in “lower level / traditional” languages (C,
 105 PHP, Shell) tend to be below the overall average, while systems written in more “modern”
 106 languages (C#, Go, Rust, TypeScript) are more inclined to feature communication platforms.
 107 The difference is even more evident for recently popularized languages if we consider projects
 108 with instant messaging platforms (*e.g.*, Go and Rust increase from +8.1% to +11.6% and
 109 from +9.5% to +16.7% respectively).

110 We performed multiple *One Proportion Z-Tests* (one for each language) and the difference
 111 in the proportion of projects using communication platforms for each language (r) and
 112 the overall dataset (r_0) is statistically significant for C, C#, Go, PHP, Rust, Shell, and
 113 TypeScript ($H_0 : r = r_0$, two-tailed Bonferroni corrected p-value < 0.0038). The same results
 114 hold for projects with instant messaging platforms.

■ **Table 1** Projects and Represented Languages

Language	Projects						
	All	CP	CP %	$\Delta_{CP}\%$	IM	IM %	$\Delta_{IM}\%$
C	1,240	548	44.2%	-13.1%	248	20.0%	-9.2%
C#	543	378	69.6%	+12.3%	214	39.4%	+10.2%
C++	1,707	926	54.2%	-3.1%	469	27.5%	-1.7%
Go	677	443	65.4%	+8.1%	276	40.8%	+11.6%
Java	1,510	860	57.0%	-0.4%	432	28.6%	-0.6%
JavaScript	1,528	899	58.8%	+1.5%	440	28.8%	-0.4%
PHP	733	379	51.7%	-5.6%	165	22.5%	-6.7%
Python	1,806	1,094	60.6%	+3.3%	557	30.8%	+1.7%
Ruby	406	238	58.6%	+1.3%	103	25.4%	-3.8%
Rust	244	163	66.8%	+9.5%	112	45.9%	+16.7%
Shell	205	93	45.4%	-11.9%	35	17.1%	-12.1%
TypeScript	895	575	64.2%	+6.9%	314	35.1%	+5.9%
Other / Unspecified	587	328	55.9%	-1.4%	160	27.3%	-1.9%
Total	12,081	6,924	57.3%		3,525	29.2%	

115 2.2 Tool Support: DWARVENMAIL

116 To support our analyses, we developed DWARVENMAIL, a Python application to scrape
 117 GitHub and extract information about projects’ README files and their history. It features
 118 an object-oriented domain model to facilitate the extraction of insights from exploration.

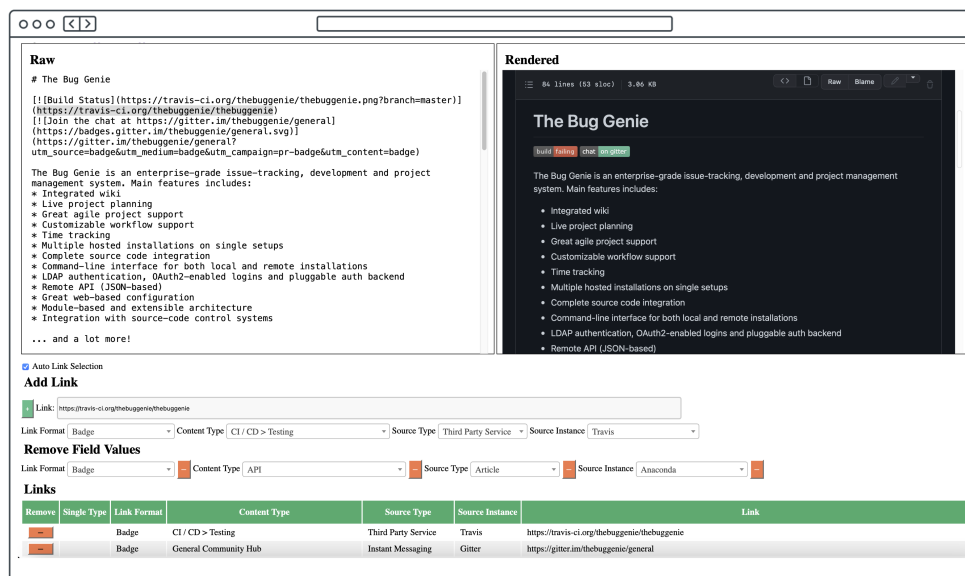
119 DWARVENMAIL also supports manual inspection, link extraction, and classification from
 120 README files (see Section 2.3). DWARVENMAIL takes the list of projects in the dataset
 121 and uses the REST API of GitHub and web scraping [81] to extract the information needed
 122 to build its internal domain model. It is implemented as a multiprocessing application to speed
 123 up the scraping. Each process uses a different API key to access GitHub in parallel through
 124 PyGitHub [23]. Parallelization is handled at project level: Each process gets a project from
 125 a queue and starts to fetch the data. Processes are also responsible for not exceeding GitHub
 126 rate limits associated with their API keys.

127 2.3 Manual Annotation

128 To examine the documentation sources and communication platforms of the projects, we
 129 performed a qualitative analysis of their README files. We relied on open card sorting,
 130 a well-established method for knowledge elicitation and classification [7, 41, 63, 76, 77], to
 131 incrementally refine the list of possible sources with flexible categories.

132 We manually reviewed the README files of the projects, extracted their links to
 133 documentation sources and organized them into categories. Given the considerable effort
 134 needed to annotate README files manually, we opted for a saturation approach [57]. We
 135 started with a sample set of 35 projects selected through stratified sampling, ensuring a
 136 balanced distribution among programming languages.

137 Two authors independently annotated each project README. Then we repeated the
 138 process in subsequent batches with 5 projects per batch until no new labels were added in
 139 two consecutive batches. We reached saturation after annotating 60 projects. In the end,
 140 we discussed conflicts and merged categories where needed. The process resulted in 2,349
 141 links with 282 link types, which we discuss in Section 3. The creation of manually annotated
 142 datasets was supported by the *Annotator* module of DWARVENMAIL (Figure 2).



■ Figure 2 DWARVENMAIL Annotator – Project Annotation Page

143 An annotator ran the Flask application locally, pulled from Git the latest updates by
 144 other annotators, started a batch of annotations, committed, and pushed the modified files.

145 The Annotator’s homepage shows a list of projects to annotate and the annotation status
 146 (*i.e.*, who annotated what). Selecting a project opens the project annotation page (Figure 2)
 147 where one can browse the README of the selected project.

148 The project annotation page uses two side-by-side panes to present the README. The
 149 left one represents the raw Markdown version of the README. The right pane shows a
 150 partially rendered version (*i.e.*, similar to what a user sees on GitHub).

151 2.4 Parsing Links: Strategy & Heuristics

152 We performed a quantitative analysis of communication platforms in README files (Sec-
153 tions 4 and 5). To support automatic platform extraction in such a large number of projects
154 we used an approach based on Regular Expressions (RE).

155 For each communication platform that we discovered, we devised REs that would match
156 the link as closely as possible, while retaining sufficient generality to abstract specific aspects
157 (*e.g.*, project name, internet domain, optional protocol). Possibly more than one RE has been
158 associated with each platform. To fine-tune the REs, DWARVENMAIL features a detailed log
159 generation for manual inspection of candidate and invalid links during the refinement.

160 DWARVENMAIL parses all the links in a README according to the set of identified REs.
161 When a link is found, specific exclusion criteria are applied. A set of rules removes links
162 to images, badge icons, platforms' generic homepages, and partial or invalid URLs (*e.g.*,
163 shorthands for Markdown sections captured by the REs).

164 The remaining links are normalized in a standard format and duplicates are removed.
165 Platforms not directly linked in the README (*e.g.*, collected in a list on the *Community*
166 page of the project website) are omitted by the employed scraping algorithm. To reduce the
167 false positive rate, DWARVENMAIL also verifies that links point to valid web pages (*i.e.*, the
168 server does not respond with an HTTP 404 Not Found). After this refinement, we obtain the
169 final set of communication platforms referenced by the project READMEs.

170 2.5 Parsing README Histories

171 For projects referencing Gitter, Slack, and Discord as communication platforms, we analyzed
172 the history of their READMEs to discover when those platforms appeared for the first time.
173 In this case, the approach outlined above to exclude invalid links (Section 2.4) would not
174 produce the desired results, because a link that is not valid today could have been valid in
175 the past. This cannot be checked without an archive, or historical information. Hence, in
176 our approach we assume that links with proper format were valid in the past. To reduce
177 false positives, we used the most specific format able to capture the link.

178 2.6 Community Size

179 We include the Discord and Slack community size (*i.e.*, number of members) in our domain
180 model. The most popular way to add people to a Discord server is through an *invite link* [15].
181 Clicking on an invite link, brings the user to a page with metadata about the server (*e.g.*,
182 number of total members, number of online members). We gathered Discord community
183 sizes by scraping the data from these invite pages. In the case of Slack, only 15% of the
184 projects in our dataset have information about the community size on the invite page.



185 *More effort is needed to explore communities that do not conform to a standard
and/or customize their invite link to pursue specific goals (e.g., authorization workflow,
authentication, spam prevention, analytics).*

186 Extending the percentage of projects whose community size is correctly scraped could
187 improve the reliability of results discussed in Sections 6.3 and 6.4.

188 2.7 Data Availability and Replication Package

189 We provide a replication package, publicly available on Figshare [51], containing the source
190 code of DWARVENMAIL, the input dataset, the manually annotated projects, the serialized
191 domain model of the scraped dataset, charts and tables exported from DWARVENMAIL.

3 Documentation Landscape

We define the documentation landscape of a software system as all the possible sources of information able to support design, implementation, comprehension, maintenance, and evolution of the system. Software documentation is a fundamental asset for developers and practitioners [3], when it is correct and up-to-date [10, 14, 19, 21, 54, 61], with its costs and benefits [79]. Modern software documentation is an ever expanding field. New sources include blogs [43], Twitter [72], StackOverflow [47], instant messaging applications [9, 18, 30, 33, 45, 46, 49, 50, 56, 58, 66, 67], news aggregators [6], and forums [27].

GitHub README files in Markdown (.md) format are a good starting point for a project from where all relevant documentation should be reachable.

Documentation sources in README files can either be directly referred to or behind multiple steps of indirection. An example of the former case is an invitation link that can be copy/pasted directly in Discord to access the community server of the project. In the latter case, the README could point to a community web page which in turn contains links to the mailing list, a Slack channel for Q&A, and potentially other communication and documentation sources.

The manual annotation presented in Section 2.3 produced 282 *single type* link tags. The links can come in many flavors thanks to the markdown format, ranging from pure textual hyperlinks to badges and images that link to external resources. We inspected them and identified three key dimensions of the documentation landscape: *content type*, *source type*, and *source instance*. We split single type tags into these three dimensions. We analyzed examples of each link type to disambiguate or enrich the classification when the original annotation had missing information.

Table 2 shows the top-15 most representative values for each dimension. The complete list of tags is available in the replication package [51].



The three key dimensions we propose to describe the documentation landscape of a software system are content type, source type, and source instance, exemplified as links in GitHub READMEs.

Source type, source instance, and content type could describe a link like: “This link is in the form of a *Badge*, it points to a *Wiki* on *Travis.com*, and contains information related to *CI / CD*.” Each dimension is instantiated with one of the possible tags for that category, forming a signature of the documentation source pointed by the link.



Exploring these dimensions could improve the automatic extraction of links and their features, to characterize and understand the (evolution of the) documentation landscape.

Link format: Link formats come in many flavors, also due to the fact that markdown files, while being textual, are usually inspected using a (multimedia capable) web browser. Badges, for example, are very common in GitHub README files, used to convey imminent information through iconic representation of a summary of the pointed resource (*e.g.*, build status passing) where the link itself allows, if followed, to reach more extensive information (*e.g.*, build process report). Masked links are another common practice to add links (not exclusively) to markdown documents.



Not all links in a raw README file are human readable links in the rendered README.

Content type: This is the primary dimension of the documentation landscape, denoting **what** kind of information is present in the landscape. There is a smooth gradient in content types regarding the number of links, but it is worth noting that there is no “standard”, but

■ **Table 2** Top-15 Most Relevant Tags, Number of Projects, and Links for Each Dimension. The percentage indicates the ratio of projects containing at least one link with the specified tag.

(a) Content Type

Projects	Content Type	Links
36 (60%)	General Community Hub	141
29 (48%)	Official Documentation	97
28 (47%)	License	68
25 (42%)	Contributing	56
23 (38%)	Issues	52
23 (38%)	CI/CD	50
21 (35%)	Project Repository	76
20 (33%)	Relevant Projects	132
20 (33%)	Dependency/Environment	84
19 (32%)	Releases	60
16 (27%)	In-Repository Resource	67
16 (27%)	Package Repository	47
14 (23%)	CI/CD > Testing	24
13 (22%)	Installation Instructions	24
11 (18%)	Code Coverage	22

(b) Source Type

Projects	Source Type	Links
55 (92%)	Homepage/Website	436
41 (68%)	Collaborative Platform	188
36 (60%)	Third Party Service	169
34 (57%)	Wiki	125
32 (53%)	Repository	166
28 (47%)	Sourcefile/Sourcefolder	151
25 (42%)	Instant Messaging	84
11 (18%)	Auxiliary README	21
10 (17%)	Readme Section/Anchor	44
9 (15%)	Mailing List	27
9 (15%)	Forum	20
8 (13%)	Image/GIF	21
8 (13%)	Blog	20
7 (12%)	Email Address	12
6 (10%)	Video	13

(c) Source Instance

Projects	Source Instance	Links
33 (55%)	GitHub	179
16 (27%)	GitHub Workflows	50
13 (22%)	GitHub Releases	27
12 (20%)	Travis	22
11 (18%)	Gitter	34
9 (15%)	Google Groups	24
7 (12%)	Discord	18
7 (12%)	Codecov	14
7 (12%)	Python Package Index	13
6 (10%)	Twitter	12
6 (10%)	StackOverflow	9
5 (8%)	Slack	18
5 (8%)	Maven	11
5 (8%)	Read the Docs	5
4 (7%)	GitHub Profile	108

rather project-specific landscapes. Most relevant are *general community hubs*: Discord servers, Slack workspaces, Gitter rooms, IRC channels, mailing lists, and forums, with their internal structure for different topics, dedicated to a general community of users and practitioners.



Content type is relevant to interpret what a piece of documentation is about. There is no standard to the documentation landscape, each project develops its own. Even the top content types (community hubs, official documentation) are present in only half of the projects.

Source type: The *source type* dimension refers to the format of the content at the link's destination. This dimension is relevant for automatically extracting the documentation landscape since it determines how the content can be retrieved and parsed. *Homepage / websites*, the most relevant source type by a large margin, can be scraped with traditional web scraping techniques. *Collaborative platforms* like GitHub and Bugzilla could be addressed via their custom APIs. *Image / GIF > Screenshots*, further down in terms of relevance, would benefit from image segmentation and analysis approaches to extract, for example, documented user interface features. We also notice that links to mailing lists are fewer than those to IM applications, a trend we analyze in more detail in Section 4.



Source type captures *how* documentation is presented and how it can be accessed. Almost all projects feature a head quarters website, i.e., the go-to place to learn about a project. These starting points are then often complemented by a plethora of other sources, ranging from Wikis to forums and instant messaging platforms.

When analyzing the evolution through time of a README file we detect in many cases that the source types come and go, inducing “tectonic movements” in the landscape, as we can observe in the example depicted in Figure 3.

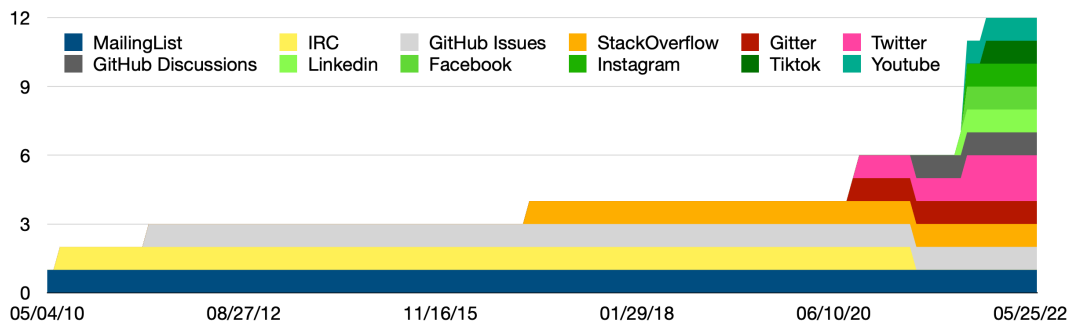


Figure 3 Evolution of Communication Platforms in the “Scikit-learn” Project

The Scikit-learn project, born in 2010, sees a mailing list as its initial documentation landscape, complemented shortly after by an IRC channel (which stopped existing a decade later). GitHub Issues is added within the project’s first year, while StackOverflow becomes part of the landscape in 2017. It is within the past 2 years that the landscape experiences an earthquake, with many new sources appearing, while the IRC channel is removed (it is worth noting that IRC and its successor, Gitter, co-exist for a year). At the time of writing the project in question features 11 different sources.



The documentation landscape of projects evolves together with the project. Especially in the past few years the source types have exploded in number, rendering the landscape highly dispersive.



The fact that there are more sources does not imply that the overall documentation of the system is better, on the contrary: We have observed an overall trend toward more volatile sources, mostly due to the rise of instant multimedia messaging platforms.

Source instance: The third dimension is a derivate of source type. For each type we can have multiple possible source instances, usually of a competing nature (see Section 5) with a similar purpose. Rather unsurprisingly for GitHub projects, GitHub itself with related instances of profiles, workflows, releases, and instant messaging (i.e., Gitter) takes top three, the 5th, and the 15th places. Services for package repositories (e.g., Python Package Index [48], Maven [62]) and CI/CD (e.g., Travis CI [73], Codecov [11]), messaging applications like Slack [60]/Discord [16], and also articles on the Medium platform [1] represent interesting research avenues.



Source instance can be seen as *where* (or by whom) documentation is “hosted.”



Source instances vary wildly, and new players constantly enter the stage. For example, recent changes in the pricing model of Slack might have influenced the ongoing mass migration toward other instant messaging platforms, of which there are dozens, with Discord quickly becoming the preferred alternative.



Tags in the three dimensions appear in different combinations, not all equally likely. Further research on the most common patterns could shed light on form and content interplay in software documentation.

4 Modern Communication Platforms

One of the recent major shifts in software development has been the emergence of various multimedia instant messaging platforms, such as Slack [60], Discord [16], and Gitter [39].

They not only experienced an increase in popularity but also seem to be a major suspect for the decline of other classical communication means, such as mailing lists and forums. We start by analyzing the platforms actually used by projects in our dataset. The scraping, based on regular expressions (see Section 2.4), took place between Aug 28 2022 at 21:01 and Aug 30 2022 at 02:12, leading to 12,081 scraped projects. Of those, 6,924 (57.3%) mention at least one such modern communication platform in their README files: 2,897 had 1 platform link, while 4,027 had 2 or more platform links. The remaining 5,157 projects had no platform links. The percentage is higher than the one reported by Käfer *et al.* [31] (57.3% vs. 46.7%), which can be explained by the fact that their analysis dates back 4 years.

We grouped communication platforms into three main categories: asynchronous, instant messaging, and social media.

Figure 4 summarizes number of links in READMEs (*Links*) and number of projects with at least one link (*Projects*) for each type of platform.

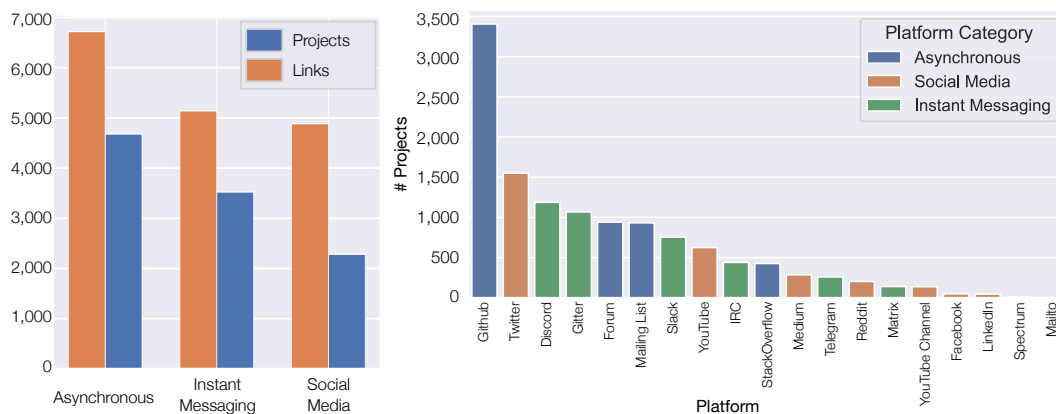


Figure 4 Platform types.

Figure 5 Number of projects linking to at least one platform.

There are Instant Messaging platforms (*e.g.*, IRC, Slack, Discord), where communication can happen in real-time. In Asynchronous platforms (*e.g.*, forum, mailing list, GitHub issues or discussions), communication usually takes some time to be processed and made available to other community members. The boundary between the two types has been blurred in the recent years by the technological improvements to the supporting infrastructure. We also considered Social Media platforms (*e.g.*, Facebook, Twitter, Youtube). While their technical features can overlap with the other types, their huge user-base and ease of forming social connections make them stand out. Table 3 shows a complete list of the platforms we considered with a short description.

A project README can have multiple links to a single platform. This is particularly true for Social Media links where Twitter accounts of the main contributors or maintainers are all referenced.

■ **Table 3** Communication Platforms

Platform	Description
Discord [16]	Voice, video, text messaging multimedia platform
Facebook [37]	Social media and social networking service
Forum	General category for web based discussion sites
GitHub [24]	GitHub infrastructure for project development
Gitter [39]	Voice, video, text messaging multimedia platform
IRC	Text-based instant messaging chat system
Linkedin [35]	Business social media & professional networking
Mailing List	E-mail based communication among recipients
Matrix [70]	Communication protocol implemented by clients
Medium [1]	Online publishing platform and social journalism
Reddit [52]	Social news aggregation, rating, discussion, and multimedia sharing
Slack [60]	Voice, video, text messaging multimedia platform
Spectrum [36]	Text-based web instant messaging chat system
StackOverflow [64]	Question and Answer website
Telegram [69]	Voice, video, text messaging multimedia platform
Twitter [75]	Social media and social networking service
Youtube [25]	Video hosting and sharing platform

299 In Figure 4, we see that the number of projects that use a specific platform is significantly
 300 lower than the number of links. For example, project *OpenAPITools/openapi-generator* [42]
 301 mentions 20 different Twitter accounts and 17 YouTube resources.

302 The identified categories are only a rough means to group similar platforms. In Figure 5
 303 we show the number of projects having at least one reference to a specific platform.


304 Given our initial input set, it is not surprising to find GitHub to be the most referenced:
 305 The infrastructure is integrated enough to warrant support for the community with its own
 306 Issues and Discussions systems. This uniform consensus is followed by a more fragmented
 307 mix of Twitter, Discord, Gitter, Forums, Mailing Lists, and others in decreasing order of
 308 “popularity.” Far from being irrelevant, these platforms are used by hundreds of projects
 309 exclusively or in synergy. The next section sheds light on these synergies, complementarities,
 310 and on the competition between similar platforms.

311 5 Coexistence and Competition

312 Communication platforms can have different features and cater to different audiences. To
 313 cover development or users’ needs, projects can opt for using multiple media at the same
 314 time. What choices are made by core developers in terms of number and variety of platforms
 315 to include in a README?

316 Figure 6 depicts a non-exhaustive list of examples of overlaps between communication
 317 platforms (extracted with the REs presented in Section 2.4) used exclusively and side-by-side.

318 Around 38% of projects that use either Discord or Slack also include GitHub Issues in
 319 their READMEs (Figure 6a).

320  *GitHub Issues can also be used without an explicit link in the README, as just a tab
 of the project, if enabled. Some platforms may be implicitly assumed to be available
 even if not present in the README.*

321 Overall, 2,105 out of 3,208 projects (66%) using GitHub Issues, also have other commu-
 322 nication platforms referenced in the README. Similar ratios are found, for example, for
 323 Discord with 801 out of 1,187 projects (67%).

324  *Multiple communication platforms of different types can and do coexist.*


325 GitHub has significant overlaps with the whole category of instant messaging, and with



■ **Figure 6** Communication Platforms Overlaps.

326 specific asynchronous platforms (*e.g.*, StackOverflow, see Figure 6b). However, 1,270 projects
 327 rely only on the integrated support provided by GitHub.

328 In general, if we consider the three main categories, we find that asynchronous platforms
 329 are used exclusively in 48% of projects, instant messaging follows with 35%, and social
 330 platforms seem the most frequently used as a complementary option (76%, see Figure 6c).

331  *It is not clear if different categories are mutually exclusive and why in a considerable
 amount of projects they tend to be used in conjunction.*

332  *This analysis should be complemented by how the user-base is distributed over these
 platforms.*

333 **6** Instant Messaging: A Deep Dive

334 What makes instant messaging platforms appealing to developers? The steady growth in
 335 the number of projects including at least one platform of this kind is a piece of evidence
 336 supporting the need for fast and rich communication.

337 Instant messaging platforms fulfill a very specific role: Providing communication in
 338 real-time, possibly with rich media sharing capabilities (*e.g.*, links, videos, files), and Voice
 339 over IP conferencing (*i.e.*, VoIP). Two instances of these platforms are seldom found together.
 340 Similar characteristics, audiences, and usages make competition the prevailing paradigm.
 341 Figure 6d shows that 97% of projects opting for these platforms choose between one of the

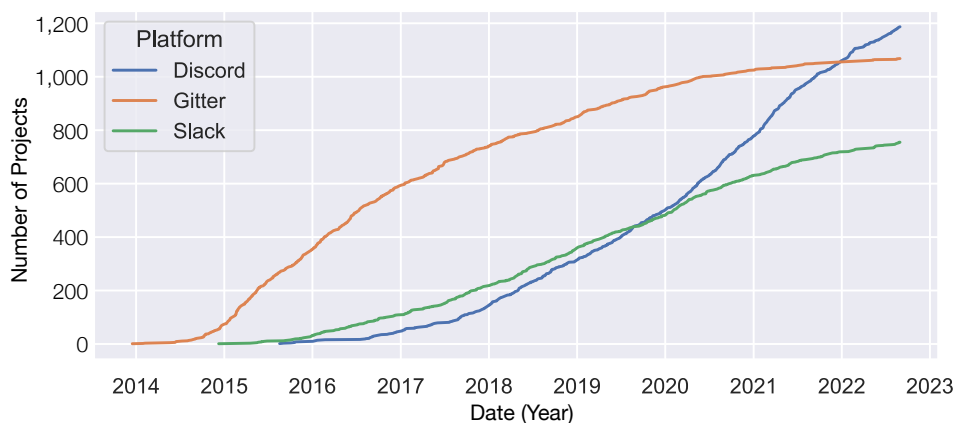
342 three alternatives. Three projects include links (see Section 2.4) to all the platforms and also
 343 other instant messaging (e.g., Spectrum), but only *PowerShell/PowerShell* has a significant
 344 Discord community (more than 10k members).



345 *Gitter, Discord, and Slack are selected by projects as alternatives, very seldom coexisting. This can be a possible strategy for successful projects not to spread their community too thin over multiple platforms with similar capabilities.*

346 6.1 Gitter, Discord, and Slack: A Timeline

347 Based on README history and mining links for each version of the README as detailed
 348 in Section 2.5, for each project, we look for the first appearance date of Gitter, Discord, and
 349 Slack (Figure 7).



■ **Figure 7** Timeline of cumulative adoption date of Slack, Discord, and Gitter.

350 Gitter appeared for the first time at the end of 2013, followed one year later by Slack,
 351 and then Discord after 8 months. All three platforms show a “ramp-up” period of slightly
 352 more than one year after their first appearance, followed by a steady growth at different
 353 rates. Both Gitter (in mid-2020) and Slack (in 2022) reached a *plateau* where just a handful
 354 of projects added them to their communication platforms in the last year. Discord, on the
 355 other hand, is still growing significantly.

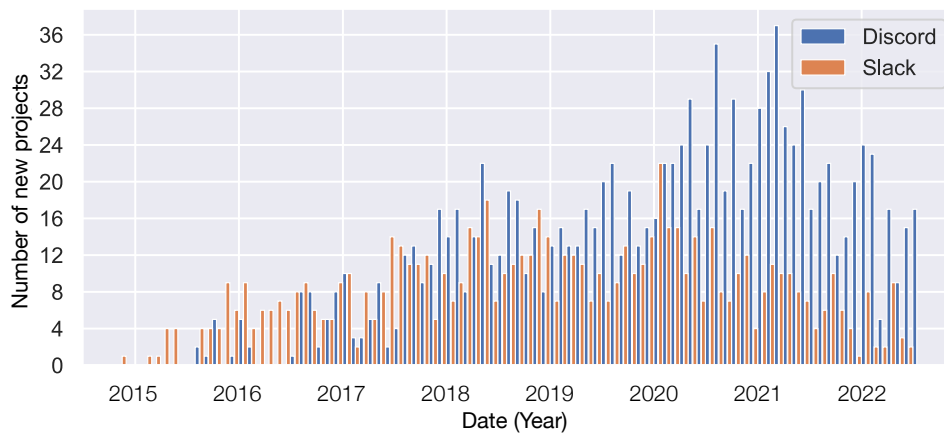
356 Since the beginning of 2020, Discord consistently outperformed Slack in terms of number
 357 of new projects adopting the platform for their community (Figure 8). The monthly growth
 358 rate has been higher than the highest for Slack in the previous years. It has also been at
 359 higher levels more consistently and for a longer period.

360 The comparison between additions of Gitter and Discord (Figure 9) shows a similar or
 361 even more evident tendency. The decline of the former and the growth of the latter are
 362 almost perfectly mirroring each other.

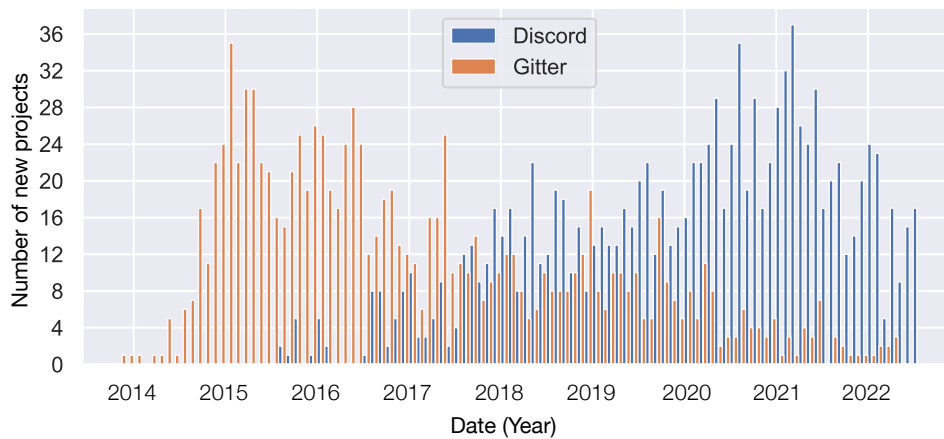


363 *While one platform stops being added to projects, another is on the rise. This happened in the past and is bound to happen again in the future.*

364 There is no guarantee that the example of the Spectrum platform we highlighted in
 365 Section 1 will be followed when Gitter goes out of fashion. It is also possible that the entire
 366 history of discussions, bug fixing sessions, and design decisions will just disappear.



■ **Figure 8** Monthly new projects adopting Discord and Slack.



■ **Figure 9** Monthly new projects adopting Gitter and Discord.

367 6.2 Throughput and Volatility

368 We investigated four Discord communities. Reactflux, Vue Land, and Angular.js are
 369 respectively tied to React, Vue, and Angular (web development frameworks). We compared
 370 them with each other and with the Discord.js community (Discord bot development).

371 In Figure 10, we show how the average number of messages per member in a sample
 372 period of three months (*i.e.*, May–July, 2022) has high variability. While this might be due
 373 to a number of factors, we are interested in the sheer scale of the messages exchanged on
 374 those platforms every day.

375 Around 550 messages are exchanged per day in Vue Land and Angular.js. In Discord.js,
 376 instead, users exchange 305 messages *each hour*, totaling more than 7k messages a day.

377 The throughput of these servers means information is lost if one does not pay attention to
 378 notifications. Only a few messages are visible at a time and they scroll up quickly, putting full
 379 conversations behind the event horizon in a matter of minutes. Alert filters and community
 380 policies (*e.g.*, forbidden mentioning of server wide tags) can only partially mitigate this
 381 problem. The trade-off between losing potentially interesting discussions and being constantly
 382 interrupted by notifications is the choice many modern developers face when dealing with
 383 these kinds of communities.

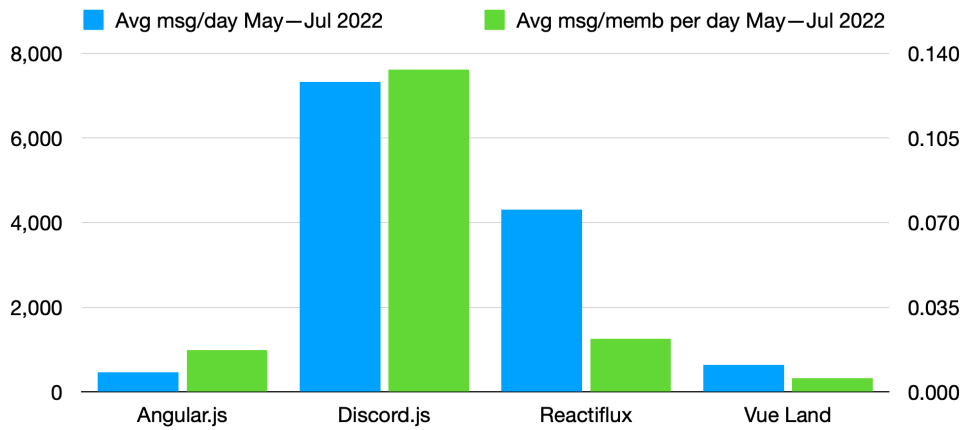



Figure 10 Messages per day and average messages per day per member from May to July 2022 for four example Discord servers.

384  Application of summarization, visualization, and information retrieval techniques is fundamental to deal with scalability problems of these platforms.

385 6.3 Community Sizes

386 Discord communities in our dataset vary in size between 2 and 500,000 members. Figure 11
 387 depicts Discord community size with respect to project age (*i.e.*, days since creation).

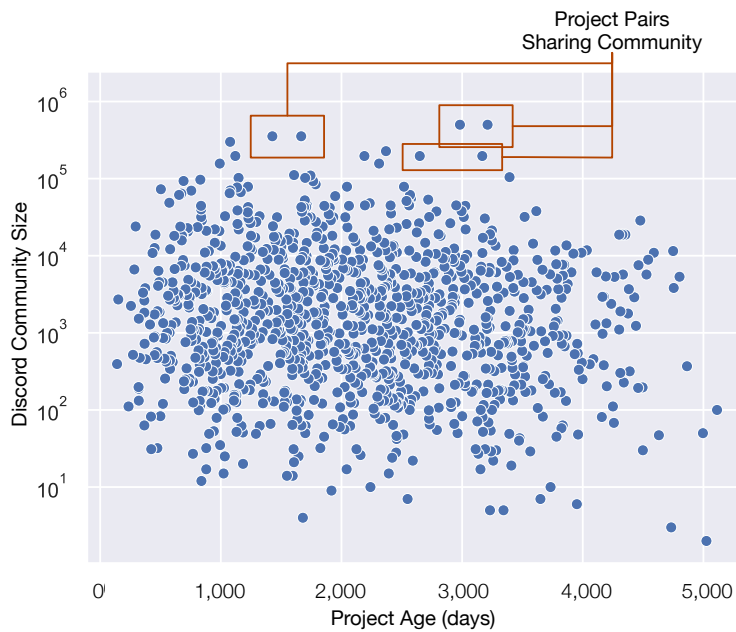



Figure 11 Discord community size with respect to project age (days from creation).

388  This should be investigated more in-depth to see if it is a breakpoint at which particular actions should be taken to keep the community growing.



Extraction of Slack community sizes has proven more difficult due to the high variance in invite page formats. Gitter does not even have an invite page to scrape, and, to the best of our knowledge, the community size cannot be automatically retrieved.

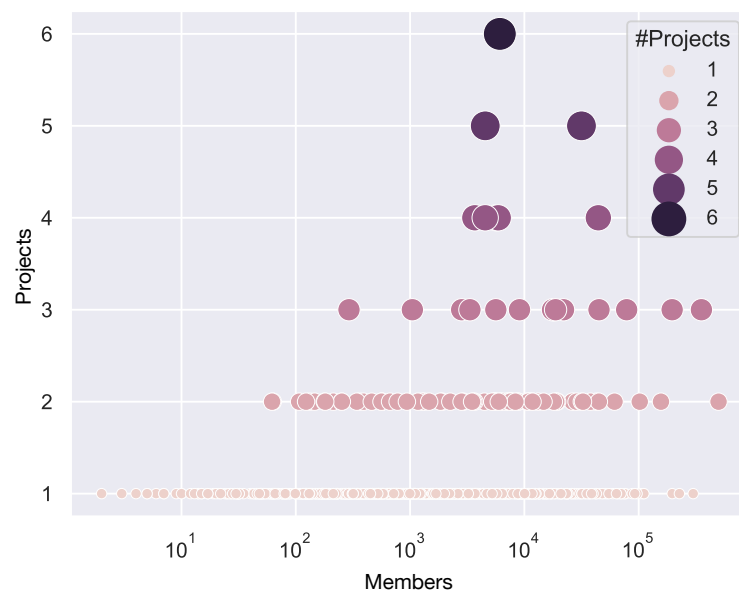
389

390 6.4 Different Projects, Same Community

391 Being in the same Discord community means sharing the same server (i.e., each project has
392 a link in the README, possibly with different formats, but pointing to the same Discord
393 server). We consider this a case of “different projects, same community”.

394 Figure 11 shows horizontal pairs in the top part of the scatterplot, suggesting that different
395 projects might share the same community: Our initial hypothesis that “same size of the
396 community means same community” might not apply, especially for smaller communities.
397 Nevertheless, it is unlikely for two different large communities to have the same number of
398 members at the same time. We manually inspected the projects in those pairs and they
399 are indeed different projects referring to the same wider community. For Discord we could
400 reliably use the community name to confirm our hypothesis, as parsed from the invitation
401 metadata (Section 2.4).

402 Figure 12 shows how many projects share a community with respect to community size.



■ **Figure 12** Projects referencing the same community.



Further analysis can show if projects are tightly coupled (e.g., different projects from the same organization, new major versions of the same project) or if different projects have an underlying reason to cater to the same audience.

403

404 6.5 Technical, Social, and Ethical Challenges

405 Some community platforms are public, some allow anonymous access, some require a form
406 of registration or access permission. We found communities with (automatic) procedures
407 to accept new members but, in general, it is hard to devise a general automatic “agent” to
408 explore all of them.

409 The sheer amount of customization that is possible, even in a simple invite landing page
410 of Slack, has been an obstacle to getting reliable data about communities lying behind those
411 pages. Exploring a larger and more varied sample could shed light on platform dependent
412 similarities and differences.



413 *Machine readable APIs for communities can greatly benefit not only research in this field, but also open new possibilities, e.g., the automatic migration of community-generated content to preserve the history of projects when the underlying technologies evolve.*

414 **Socio-ethical challenges:** Communities usually are digital aggregations of people's
415 thoughts, ideas, rants, strengths, and weaknesses. Collecting such information can be seen as
416 poking inside someone's house. One can do it if having legitimate reasons to do so. One can
417 be welcome if providing benefits for the community. But one can also be faced with concerns
418 about privacy and legitimate use of collected information.

419 Companies owning the platform sometimes are more keen to share their data than
420 administrators of communities they host. While Slack has a monetization policy tied to its
421 history retrievability, Discord allows unlimited access to a wealth of historical information
422 via its API.

423 A big role in the extensibility of our study is played by the attitude of administrators of
424 interesting communities. While information might be public (*i.e.*, anyone with a Discord
425 account can automatically join a server and browse its entire content), to comply with
426 Discord's Terms of Service we need to ask for permission to add a bot to extract useful
427 information for DWARVENMAIL. In this crucial step a fundamental role is played by personal
428 beliefs and perceptions of the benefits of such a bot by the administrators and the community
429 itself.



430 *Accessibility of information is ever more beyond the fence of what is technically possible, towards the barrier of what is ethically and socially accepted.*

431 **7 Threats to Validity**

432 Our analyses are based on a dataset of public GitHub projects as the only source. This poses
433 a threat to the generalizability of our results with respect to the type of projects hosted on
434 GitHub. Open source projects developed in this social coding style might differ significantly
435 from closed source projects developed by a small team of hired developers. There are also no
436 guarantees that the results presented can be generalized to projects hosted on other similar
437 repositories (*e.g.*, SourceForge).

438 The current study presents a limited generalizability with respect to the format of
439 README files. Although our sampling procedure (Section 2.3) ensures generalizability with
440 respect to the project's main programming language, different README file formats could
441 provide different link types and formats not fully captured by our analysis.



442 *We found evidence of more than 15 different README formats. While most share a similar structure for external links, systematic analysis of these formats could improve the generalizability of the results.*

443 Limiting the extraction of the documentation landscape to what is reachable from the
444 main README file (*i.e.*, ignoring links to other READMEs in submodules of a project)
445 poses a threat to construct validity. This threat is partially mitigated by the magnitude of
446 the phenomenon we highlighted, emerging despite the limited scope, and calling for discussion
447 and further investigation (*i.e.*, also considering auxiliary documentation sources as a starting

43:18 On the Rise of Modern Software Documentation

448 point to map the landscape).

449 Our analysis benefits from verifying the validity of links whenever possible (*i.e.*, if the
450 resource referred by the link is still available we expect an HTTP 200 OK response). When
451 mining GitHub we verified the links we found in a two step process. The time interval
452 between the first pass for scraping and the second pass for verification was short enough to
453 guarantee that most links were in their intended state. Obsolete links may be possible and
454 are part of the present study.



455 *The analysis lacks accuracy when links are redirected or reused. Moreover, in the effort to reconstruct link patterns for previous standard link formats of some platforms (e.g., Slack) we adopt a conservative approach where if the format follows reasonable patterns it is accepted as a valid link in the history of a README file. We have no guarantee nor a way to discover if the link was valid in the past.*

456 The only possibility to study the evolution and validity of such links is to constantly
457 monitor README files and their evolution over a period of time. Link validity can be
458 checked as soon as the change in the README is triggered. This kind of study is outside of
459 the scope of the presented work.



460 *Semantic analysis of the pointed links could improve relatedness, reducing false positives in link validity. Automatic link validity and relatedness to the source topic should be investigated.*

461 Links that are not visually represented in the rendered README are currently part of
462 the analyses. This threat to the validity of our conclusions is partially mitigated by the low
463 frequency of such occurrences. We found only 3 non-rendered links in 2 manually annotated
464 projects (0.1% of links, 3.3% of projects).



465 *We did not perform an analysis based on project types. Relationships between project type, intended audience, and the resulting documentation landscape could provide insights on how to leverage the landscape for projects of different natures and at different maturity stages.*

466 8 Related Work

467 Communication channels, especially those tightly coupled with collaborative development
468 platforms (*e.g.*, GitHub), are fundamental for successful software development.

469 Hoegl *et al.* [29] and Lindsjrn *et al.* [34] found communication to be an essential
470 subcontract of teamwork quality. Tantisuwankul *et al.* analyzed the communication channels
471 of GitHub projects [68]. Studying 70k library projects in 7 ecosystems, they identified
472 13 communication channels as “a form of knowledge transfer or sharing” (*e.g.*, licenses,
473 change logs). They found that GitHub projects adopt multiple channels, which change
474 over time, to capture new and update existing knowledge. Storey *et al.* [65] conducted
475 a large-scale survey with 1,449 GitHub users to understand the communication channels
476 developers find essential to their work. On average, developers indicated they use 11.7
477 channels across all their activities (*e.g.*, email, chat, microblogging, Q&A websites). They
478 concluded that “communication channels shape and challenge the participatory culture in
479 software development.” Hata *et al.* [27] studied early adopters of GitHub Discussions, finding
480 that developers considered them useful and important. Lima *et al.* [32] used NLP to detect
481 related discussions of OSS communities in GitHub Discussions.

482 Treude and Storey [74] interviewed users of a community portal, finding that clients,
483 developers, and end-users are involved in the process of externalizing developer knowledge.

484 Nugroho *et al.* [40] studied how Eclipse developers utilize project forums, concluding that
485 forums are essential platforms for linking various resources in the Eclipse ecosystem besides
486 representing an important source of expert knowledge.

487 Modern social media are becoming another information source for development activities.
488 Mezouar *et al.* [38] studied how tweets can improve the bug fixing process. They observed how
489 issues for Firefox and Chrome are usually reported earlier through Twitter than on tracking
490 systems. This can potentially decrease the lifespan of a bug. Guzman *et al.* [26] analyzed
491 the usage characteristics, content, and automatic classification of tweets about software
492 applications. They found that tweets contain useful information for software companies but
493 stressed the need for automatic filtering of irrelevant information.

494 Instant messaging platforms, from Internet Relay Chat (IRC) to Discord, went from simple
495 text messages to rich multimedia support with integrated DevOps workflows (*i.e.*, Slack
496 integrations). Yu *et al.* [78] learned how real-time (*i.e.*, IRC) and asynchronous (*i.e.*, mailing
497 lists) communications were used and balanced across the GNOME GTK+ project. Shihab
498 *et al.* [59] analyzed IRC meeting logs and found that developers actively contributed through
499 meeting channels.

500 Lin *et al.* [33] argued Slack played an increasingly significant role, sometimes replacing
501 emails. They found various benefits of Slack over mailing lists. Developers use it for team-wide
502 purposes (*e.g.*, communicating with teammates, file and code sharing), community support
503 (*e.g.*, special interest groups), and personal benefits (*e.g.*, networking, social activities). They
504 also observed that developers commonly used bots to support their work. Chatterjee *et al.* [9]
505 analyzed the conversations of developers from five Slack programming communities and
506 developers' StackOverflow posts. They found prevalent useful information, including API
507 mentions and code snippets with descriptions in both sources.

508 Alkadhi *et al.* [5] examined “rationale” elements (*i.e.*, discussed issues, alternatives,
509 pro-/con-arguments, decisions) in Atlassian HipChat messages of three software development
510 teams. They found frequent, valuable discussions with elements of rationale. However, they
511 also emphasized the need for automated tools due to the high volume of chat messages.

512 Shi *et al.* [58] conducted an empirical study on developers' Gitter chats. They manually
513 analyzed 749 dialogs and performed an automated analysis of over 173K dialogs of OSS
514 communities. Interestingly, developers tend to converse more on Wednesdays and Thursdays.
515 They also found interaction patterns among conversations and noticed that developers tend
516 to discuss topics such as API usage and errors. They argue the need for better utilization
517 and mining of knowledge embedded in the massive chat history of OSS communities.

518 Hata *et al.* analyzed links in source code comments [28] in a large-scale study (~10
519 million links) extracted from files of the main language of the project. We focus on README
520 file links, which are independent of the project language.

521 Ebert *et al.* [17] conducted an empirical study to understand which communication
522 channels are used in GitHub projects and how they are presented to the audience, finding
523 that the most common were chats, mail-related, social media, and GitHub channels. Käfer
524 *et al.* [31] analyzed GitHub communication channels, finding that “Mailing lists are being
525 replaced by modern enterprise chat systems in OSS development.” Our work broadens the
526 scope beyond communication channels and adds details needed to identify the current status,
527 understand how it has evolved, and obtain meaningful insights on why this is happening.

528 Each of the previously discussed studies focuses on a specific part of the documentation
529 landscape, recognizing the importance of the sources for knowledge management and doc-
530 umentation. What is still missing is a higher level understanding of the phenomenon that
531 shifts the relative importance of these sources over time, intra- and inter-project.

532 **9** Conclusions and Future Work

533 Classical software documentation is being replaced by “communication”. At least in open
 534 source software on GitHub, it is supported by a plethora of platforms characterized by
 535 high throughput, volatility, and heterogeneity. The original vision of on-demand developer
 536 documentation [55] advocated for a paradigm shift. A shift did happen, but it was not in the
 537 direction foreseen by Robillard *et al.* five years ago. The new communication platforms bring
 538 new challenges and opportunities for modern software documentation. It is time to shed
 539 light on new forms of documentation. A comparison with classical documentation and where
 540 it survives, unscathed by the new media and the needs of modern software development,
 541 might help rethink the role of documentation itself. Research efforts in this direction can
 542 help maintain documentation useful for software comprehension, maintenance, and evolution,
 543 independently of the form it will take.

544 To achieve this we need a better understanding of the phenomenon occurring to software
 545 documentation sources. We regard the present work as scratching the surface of what
 546 has turned into an emerging heterogeneous, complex, and ever-changing documentation
 547 landscape, a *terra incognita* full of possibilities and threats.

548 **Acknowledgements:** This work is supported by the Swiss National Science Foundation
 549 (SNSF) through the project “INSTINCT” (SNF Project No. 190113). Marco Raglianti would
 550 also like to thank the Swiss Group for Original and Outside-the-box Software Engineering
 551 (CHOOSE) for sponsoring the trip to the conference.

552 — **References** —

- 553 1 A Medium Corporation. Medium. URL: <https://medium.com/>.
- 554 2 Tim Abbott. Why Slack’s free plan change is causing an exodus. URL: <https://blog.zulip.com/2022/08/26/why-slacks-free-plan-change-is-causing-an-exodus/>.
- 555 3 Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele
 556 Lanza, and David C. Shepherd. Software documentation: The practitioners’ perspective. In
 557 *Proceedings of ICSE 2020 (International Conference on Software Engineering)*, pages 590–601.
 558 ACM, 2020.
- 559 4 Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura
 560 Moreno, Gabriele Bavota, and Michele Lanza. Software documentation issues unveiled. In
 561 *Proceedings of ICSE 2019 (International Conference on Software Engineering)*, pages 1199–
 562 1210. IEEE/ACM, 2019.
- 563 5 Rana Alkadhi, Teodora Lata, Emitza Guzman, and Bernd Bruegge. Rationale in development
 564 chat messages: An exploratory study. In *Proceedings of MSR 2017 (International Conference
 565 on Mining Software Repositories)*, pages 436–446. IEEE/ACM, 2017.
- 566 6 Maurício Aniche, Christoph Treude, Igor Steinmacher, Igor Wiese, Gustavo Pinto, Margaret-
 567 Anne Storey, and Marco Aurélio Gerosa. How modern news aggregators help development
 568 communities shape and share knowledge. In *Proceedings of ICSE 2018 (International Confer-
 569 ence on Software Engineering)*, pages 499–510. ACM, 2018.
- 570 7 Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code
 571 review. In *Proceedings of ICSE 2013 (International Conference on Software Engineering)*,
 572 pages 712–721. IEEE, 2013.
- 573 8 Hudson Borges and Marco Tulio Valente. What’s in a GitHub star? Understanding repository
 574 starrng practices in a social coding platform. *Journal of Systems and Software*, 146:112–129,
 575 2018.
- 576 9 Preetha Chatterjee, Kostadin Damevski, Lori Pollock, Vinay Augustine, and Nicholas A Kraft.
 577 Exploratory study of Slack Q&A chats as a mining source for software engineering tools. In
 578

- 579 *Proceedings of MSR 2019 (International Conference on Mining Software Repositories)*, pages
580 490–501. IEEE/ACM, 2019.
- 581 10 Jie-Cherng Chen and Sun-Jen Huang. An empirical analysis of the impact of software
582 development problem factors on software maintainability. *Journal of Systems and Software*,
583 82(6):981–992, 2009.
- 584 11 Codecov. Codecov. URL: <https://about.codecov.io/>.
- 585 12 David Curry. Slack revenue and usage statistics (2022). URL: <https://www.businessofapps.com/data/slack-statistics/>.
- 586 13 Ozren Dabic, Emad Aghajani, and Gabriele Bavota. Sampling projects in GitHub for MSR stud-
587 ies. In *Proceedings of MSR 2021 (International Conference on Mining Software Repositories)*,
588 pages 560–564. IEEE/ACM, 2021.
- 589 14 Barthél my Dagenais and Martin P Robillard. Creating and evolving developer documenta-
590 tion: Understanding the decisions of open source contributors. In *Proceedings of FSE 2010*
591 *(International Symposium on Foundations of Software Engineering)*, pages 127–136. ACM,
592 2010.
- 593 15 Discord. Invites 101. URL: <https://support.discord.com/hc/en-us/articles/208866998-Invites-101>.
- 594 16 Discord, Inc. Discord. URL: <https://discord.com/>.
- 595 17 Verena Ebert, Daniel Graziotin, and Stefan Wagner. How are communication channels on
596 GitHub presented to their intended audience? – A thematic analysis. In *Proceedings of EASE*
597 *2022 (International Conference on Evaluation and Assessment in Software Engineering)*, pages
598 40–49. ACM, 2022.
- 599 18 Osama Ehsan, Safwat Hassan, Mariam El Mezouar, and Ying Zou. An empirical study of
600 developer discussions in the Gitter platform. *Transactions on Software Engineering and*
601 *Methodology*, 30(1):1–39, 2020.
- 602 19 Andrew Forward and Timothy C Lethbridge. The relevance of software documentation,
603 tools and technologies: A survey. In *Proceedings of DocEng 2002 (Symposium on Document*
604 *Engineering)*, pages 26–33. ACM, 2002.
- 605 20 freeCodeCamp. Our experience with Slack. URL: <https://www.freecodecamp.org/news/so-yeah-we-tried-slack-and-we-deeply-regretted-it-391bcc714c81>.
- 606 21 Golara Garousi, Vahid Garousi-Yusifoglu, Guenther Ruhe, Junji Zhi, Mahmoud Moussavi,
607 and Brian Smith. Usage and usefulness of technical software documentation: An industrial
608 case study. *Information and Software Technology*, 57:664–682, 2015.
- 609 22 GitHub. Fork a repo. URL: <https://docs.github.com/en/get-started/quickstart/fork-a-repo>.
- 610 23 GitHub. PyGithub. URL: <https://github.com/PyGithub/PyGithub>.
- 611 24 GitHub, Inc. GitHub. URL: <https://github.com/>.
- 612 25 Google, LLC. YouTube. URL: <https://www.youtube.com/>.
- 613 26 Emitza Guzman, Rana Alkadhi, and Norbert Seyff. A needle in a haystack: What do Twitter
614 users say about software? In *Proceedings of RE 2016 (International Requirements Engineering*
615 *Conference)*, pages 96–105. IEEE, 2016.
- 616 27 Hideaki Hata, Nicole Novielli, Sebastian Baltes, Raula Gaikovina Kula, and Christoph Treude.
617 GitHub Discussions: An exploratory study of early adoption. *Empirical Software Engineering*,
618 27(1):1–32, 2022.
- 619 28 Hideaki Hata, Christoph Treude, Raula Gaikovina Kula, and Takashi Ishio. 9.6 million
620 links in source code comments: Purpose, evolution, and decay. In *Proceedings of ICSE 2019*
621 *(International Conference on Software Engineering)*, pages 1211–1221. IEEE, 2019.
- 622 29 Martin Hoegl and Hans Gemuenden. Teamwork quality and the success of innovative projects:
623 A theoretical concept and empirical evidence. *Organization Science*, 12(4):435–449, 2001.
- 624 30 Jialun Aaron Jiang, Charles Kiene, Skyler Middler, Jed R. Brubaker, and Casey Fiesler.
625 Moderation challenges in voice-based online communities on Discord. *Proceedings of HCI 2019*
626 *(Human-Computer Interaction)*, 3(CSCW):1–23, 2019.
- 627
628
629
630

- 631 31 Verena Käfer, Daniel Graziotin, Ivan Bogicevic, Stefan Wagner, and Jasmin Ramadani.
632 Communication in Open-Source projects – End of the e-mail era? In *Proceedings of ICSE*
633 *2018 (International Conference on Software Engineering)*, pages 242–243. ACM, 2018.
- 634 32 Marcia Lima, Igor Steinmacher, Denae Ford, Evangeline Liu, Grace Vorreuter, Tayana Conte,
635 and Bruno Gadelha. Looking for related discussions on GitHub Discussions. In *arXiv*, 2022.
- 636 33 Bin Lin, Alexey Zagalsky, Margaret-Anne Storey, and Alexander Serebrenik. Why developers
637 are slacking off: Understanding how software teams use Slack. In *Proceedings of CSCW/SCC*
638 *2016*, pages 333–336. ACM, 2016.
- 639 34 Yngve Lindsjörn, Dag I.K. Sjøberg, Torgeir Dingsøy, Gunnar R. Bergersen, and Tore Dybå.
640 Teamwork quality and project success in software development: A survey of agile development
641 teams. *Journal of Systems and Software*, 122:274–286, 2016.
- 642 35 LinkedIn Corporation. LinkedIn. URL: <https://www.linkedin.com>.
- 643 36 Brian Lovin. Join us on our new journey. URL: [https://web.
644 archive.org/web/20220927203327/https://spectrum.chat/spectrum/general/
645 join-us-on-our-new-journey~e4ca0386-f15c-4ba8-8184-21cf5fa39cf5](https://web.archive.org/web/20220927203327/https://spectrum.chat/spectrum/general/join-us-on-our-new-journey~e4ca0386-f15c-4ba8-8184-21cf5fa39cf5).
- 646 37 Meta. Facebook. URL: <https://www.facebook.com/>.
- 647 38 Mariam El Mezouar, Feng Zhang, and Ying Zou. Are tweets useful in the bug fixing process?
648 An empirical study on Firefox and Chrome. *Empirical Software Engineering*, 23(3):1704–1742,
649 2018.
- 650 39 New Vector, Ltd. Gitter. URL: <https://gitter.im/>.
- 651 40 Yusuf Sulisty Nugroho, Syful Islam, Keitaro Nakasai, Ifraz Rehman, Hideaki Hata,
652 Raula Gaikovina Kula, Meiyappan Nagappan, and Kenichi Matsumoto. How are project-
653 specific forums utilized? A study of participation, content, and sentiment in the Eclipse
654 ecosystem. *Empirical Software Engineering*, 26(6):132, 2021.
- 655 41 N. Nurmuliani, D. Zowghi, and S. P. Williams. Using card sorting technique to classify
656 requirements change. In *Proceedings of IREC 2004 (International Requirements Engineering*
657 *Conference)*, pages 240–248. IEEE, 2004.
- 658 42 OpenAPI Tools. OpenAPI Generator. URL: [https://github.com/OpenAPITools/
659 openapi-generator](https://github.com/OpenAPITools/openapi-generator).
- 660 43 Dennis Pagano and Walid Maalej. How do developers blog? An exploratory study. In
661 *Proceedings of MSR 2011 (Working Conference on Mining Software Repositories)*, pages
662 123–132. ACM, 2011.
- 663 44 Papyrus. Easy company intranet & internal team wiki for Slack. URL: [https://papyrus.com/
664 slack-wiki-intranet/](https://papyrus.com/slack-wiki-intranet/).
- 665 45 Esteban Parra, Mohammad Alahmadi, Ashley Ellis, and Sonia Haiduc. A comparative study
666 and analysis of developer communications on Slack and Gitter. *Empirical Software Engineering*,
667 27(2):1–33, 2022.
- 668 46 Esteban Parra, Ashley Ellis, and Sonia Haiduc. GitterCom: A dataset of Open Source
669 developer communications in Gitter. In *Proceedings of MSR 2020 (International Conference*
670 *on Mining Software Repositories)*, pages 563–567. ACM, 2020.
- 671 47 Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza.
672 Mining StackOverflow to turn the IDE into a self-confident programming prompter. In
673 *Proceedings of MSR 2014 (Working Conference on Mining Software Repositories)*, pages
674 102–111. IEEE/ACM, 2014.
- 675 48 Python Software Foundation. Python Package Index. URL: <https://pypi.org/>.
- 676 49 Marco Raglianti, Roberto Minelli, Csaba Nagy, and Michele Lanza. Visualizing Discord servers.
677 In *Proceedings of VISSOFT 2021 (Working Conference on Software Visualization)*, pages
678 150–154. IEEE, 2021.
- 679 50 Marco Raglianti, Csaba Nagy, Roberto Minelli, and Michele Lanza. Using Discord conversations
680 as program comprehension aid. In *Proceedings of ICPC 2022 (International Conference on*
681 *Program Comprehension)*, pages 597–601. ACM, 2022.

- 682 51 Marco Raglianti, Csaba Nagy, Roberto Minelli, Bin Lin, and Michele Lanza. Replication
683 package. URL: <https://figshare.com/s/33c8af534dba61d72c41>.
- 684 52 Reddit. Reddit. URL: <https://www.reddit.com/>.
- 685 53 Lionel P Robert and Alan R Dennis. Paradox of richness: A cognitive model of media choice.
686 *IEEE Transactions on Professional Communication*, 48(1):10–21, 2005.
- 687 54 Martin P Robillard and Robert DeLine. A field study of API learning obstacles. *Empirical
688 Software Engineering*, 16(6):703–732, 2011.
- 689 55 Martin P. Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro,
690 Neil Ernst, Marco Aurélio Gerosa, Michael Godfrey, Michele Lanza, Mario Linares-Vásquez,
691 Gail C. Murphy, Laura Moreno, David Shepherd, and Edmund Wong. On-demand developer
692 documentation. In *Proceedings of ICSME 2017 (International Conference on Software Main-
693 tenance and Evolution)*, pages 479–483. IEEE, 2017.
- 694 56 Hareem Sahar, Abram Hindle, and Cor-Paul Bezemer. How are issue reports discussed in
695 Gitter chat rooms? *Journal of Systems and Software*, 172:110852, 2021.
- 696 57 Benjamin Saunders, Julius Sim, Tom Kingstone, Shula Baker, Jackie Waterfield, Bernadette
697 Bartlam, Heather Burroughs, and Clare Jinks. Saturation in qualitative research: Exploring
698 its conceptualization and operationalization. *Quality & Quantity*, 52(4):1893–1907, 2018.
- 699 58 Lin Shi, Xiao Chen, Ye Yang, Hanzhi Jiang, Ziyou Jiang, Nan Niu, and Qing Wang. A first
700 look at developers’ live chat on Gitter. In *Proceedings of ESEC/FSE 2021 (European Software
701 Engineering Conference and Symposium on the Foundations of Software Engineering)*, pages
702 391–403. ACM, 2021.
- 703 59 Emad Shihab, Zhen Ming Jiang, and Ahmed E Hassan. On the use of internet relay chat
704 (IRC) meetings by developers of the GNOME GTK+ project. In *Proceedings of MSR 2009
705 (Working Conference on Mining Software Repositories)*, pages 107–110. IEEE, 2009.
- 706 60 Slack Technologies. Slack. URL: <https://slack.com/>.
- 707 61 Ian Sommerville. *Software Engineering*. Pearson, 10th edition, 2015.
- 708 62 Sonatype. Maven Central Repository. URL: <https://central.sonatype.dev/>.
- 709 63 Donna Spencer. *Card Sorting: Designing Usable Categories*. Rosenfeld Media, 2009.
- 710 64 Stack Exchange, Inc. Stack Overflow. URL: <https://stackoverflow.com/>.
- 711 65 Margaret-Anne Storey, Alexey Zagalsky, Fernando Figueira Filho, Leif Singer, and Daniel M.
712 German. How social and communication channels shape and challenge a participatory culture
713 in software development. *IEEE Transactions on Software Engineering*, 43(2):185–204, 2017.
- 714 66 Viktoria Stray and Nils Brede Moe. Understanding coordination in global software engineering:
715 A mixed-methods study on the use of meetings and Slack. *Journal of Systems and Software*,
716 170:110717, 2020.
- 717 67 Keerthana Muthu Subash, Lakshmi Prasanna Kumar, Sri Lakshmi Vadlamani, Preetha
718 Chatterjee, and Olga Baysal. DISCO: A dataset of Discord chat conversations for software
719 engineering research. In *Proceedings of MSR 2022 (International Conference on Mining
720 Software Repositories)*, pages 227–231. IEEE/ACM, 2022.
- 721 68 Jirateep Tantisuwankul, Yusuf Sulisty Nugroho, Raula Gaikovina Kula, Hideaki Hata, Arnon
722 Rungsawang, Pattara Leelaprute, and Kenichi Matsumoto. A topological analysis of commu-
723 nication channels for knowledge sharing in contemporary GitHub projects. *Journal of Systems
724 and Software*, 158:110416, 2019.
- 725 69 Telegram. Telegram. URL: <https://telegram.org/>.
- 726 70 The Matrix.org Foundation C.I.C. Matrix. URL: <https://matrix.org/>.
- 727 71 Mike Thelwall and Liwen Vaughan. A fair history of the web? Examining country balance in
728 the Internet Archive. *Library & Information Science Research*, 26(2):162–176, 2004.
- 729 72 Yuan Tian, Palakorn Achananuparp, Ibrahim Nelman Lubis, David Lo, and Ee-Peng Lim.
730 What does software engineering community microblog about? In *Proceedings of MSR 2012
731 (Working Conference on Mining Software Repositories)*, pages 247–250. IEEE, 2012.
- 732 73 Travis CI. Travis CI. URL: <https://www.travis-ci.com/>.

- 733 **74** Christoph Treude and Margaret-Anne Storey. Effective communication of software development
734 knowledge through community portals. In *Proceedings of ESEC/FSE 2011 (European Software*
735 *Engineering Conference and Symposium on the Foundations of Software Engineering)*, pages
736 91–101. ACM, 2011.
- 737 **75** Twitter, Inc. Twitter. URL: <https://twitter.com/>.
- 738 **76** Jed R Wood and Larry E Wood. Card sorting: Current practices and beyond. *Journal of*
739 *Usability Studies*, 4(1):1–6, 2008.
- 740 **77** Zhou Yang, Chenyu Wang, Jieke Shi, Thong Hoang, Pavneet Kochhar, Qinghua Lu, Zhenchang
741 Xing, and David Lo. What do users ask in open-source AI repositories? An empirical study of
742 GitHub issues. *arXiv preprint arXiv:2303.09795*, 2023.
- 743 **78** Liguu Yu, Srinu Ramaswamy, Alok Mishra, and Deepti Mishra. Communications in global
744 software development: An empirical study using GTK+ OSS repository. In *Proceedings of*
745 *OTM 2011 (On the Move to Meaningful Internet Systems)*, pages 218–227. Springer, 2011.
- 746 **79** Junji Zhi, Vahid Garousi-Yusifoglu, Bo Sun, Golaru Garousi, Shawn Shahnewaz, and Guenther
747 Ruhe. Cost, benefits and quality of software development documentation: A systematic
748 mapping. *Journal of Systems and Software*, 99:175–198, 2015.
- 749 **80** Carlos Zimmerle, Kiev Gama, Fernando Castor, and José Murilo Mota Filho. Mining the
750 usage of reactive programming APIs: A study on GitHub and Stack Overflow. In *Proceedings*
751 *of MSR 2022 (International Conference on Mining Software Repositories)*, pages 203–214.
752 ACM, 2022.
- 753 **81** Zyte. Scrapy. URL: <https://scrapy.org>.