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 ∼12K 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

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

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

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

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

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.

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

We show the most representative values in different dimensions characterizing the landscape. We then proceed more in-depth with the history of modern communication platforms. We show how some platforms have seen increasing adoption, reached a plateau, and finally started their decline. Our analysis provides insights into the many implications of this

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.
ongoing phenomenon for software documentation. Finally, we discuss possible features that future platforms should have to mitigate some of the perils introduced by these continuous shifts. We use the following icons to highlight salient points:

- 📚 Insight
- ✨ Idea/Future Work
- ⚠️ Threat

![Figure 1 DwarvenMail and Analyses Overview.](image)

## 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.

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

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

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

### Table 1 Projects and Represented Languages.

<table>
<thead>
<tr>
<th>Language</th>
<th>Projects</th>
<th>All</th>
<th>CP</th>
<th>CP %</th>
<th>( \Delta_{CP} \ % )</th>
<th>IM</th>
<th>IM %</th>
<th>( \Delta_{IM} \ % )</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1,240</td>
<td>518</td>
<td>41.2%</td>
<td>-13.1%</td>
<td>218</td>
<td>20.0%</td>
<td>-9.2%</td>
<td></td>
</tr>
<tr>
<td>C#</td>
<td>543</td>
<td>378</td>
<td>69.6%</td>
<td>+12.3%</td>
<td>214</td>
<td>39.4%</td>
<td>+10.2%</td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td>1,707</td>
<td>926</td>
<td>54.2%</td>
<td>-3.1%</td>
<td>469</td>
<td>27.5%</td>
<td>-1.7%</td>
<td></td>
</tr>
<tr>
<td>Go</td>
<td>677</td>
<td>443</td>
<td>65.4%</td>
<td>+8.1%</td>
<td>276</td>
<td>40.8%</td>
<td>+11.6%</td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td>1,510</td>
<td>860</td>
<td>57.0%</td>
<td>-0.4%</td>
<td>432</td>
<td>28.6%</td>
<td>-0.6%</td>
<td></td>
</tr>
<tr>
<td>JavaScript</td>
<td>1,528</td>
<td>899</td>
<td>58.8%</td>
<td>+1.5%</td>
<td>440</td>
<td>28.8%</td>
<td>-0.4%</td>
<td></td>
</tr>
<tr>
<td>PHP</td>
<td>733</td>
<td>379</td>
<td>51.7%</td>
<td>-5.6%</td>
<td>165</td>
<td>22.5%</td>
<td>-6.7%</td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td>1,806</td>
<td>1,094</td>
<td>60.6%</td>
<td>+3.3%</td>
<td>557</td>
<td>30.8%</td>
<td>+1.7%</td>
<td></td>
</tr>
<tr>
<td>Ruby</td>
<td>406</td>
<td>238</td>
<td>58.6%</td>
<td>+1.3%</td>
<td>103</td>
<td>25.4%</td>
<td>-3.8%</td>
<td></td>
</tr>
<tr>
<td>Rust</td>
<td>244</td>
<td>163</td>
<td>66.8%</td>
<td>+9.5%</td>
<td>112</td>
<td>45.9%</td>
<td>+16.7%</td>
<td></td>
</tr>
<tr>
<td>Shell</td>
<td>205</td>
<td>93</td>
<td>45.1%</td>
<td>-11.9%</td>
<td>35</td>
<td>17.1%</td>
<td>-12.1%</td>
<td></td>
</tr>
<tr>
<td>TypeScript</td>
<td>895</td>
<td>575</td>
<td>64.2%</td>
<td>+6.9%</td>
<td>314</td>
<td>35.1%</td>
<td>+5.9%</td>
<td></td>
</tr>
<tr>
<td>Other / Unspecified</td>
<td>587</td>
<td>328</td>
<td>55.9%</td>
<td>-1.4%</td>
<td>160</td>
<td>27.3%</td>
<td>-1.9%</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>12,081</td>
<td>6,924</td>
<td>57.3%</td>
<td></td>
<td>3,525</td>
<td>29.2%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.2 Tool Support: DwarvenMail

To support our analyses, we developed DwarvenMail, a Python application to scrape GitHub and extract information about projects’ README files and their history. It features an object-oriented domain model to facilitate the extraction of insights from exploration. DwarvenMail also supports manual inspection, link extraction, and classification from README files (see Section 2.3). DwarvenMail takes the list of projects in the dataset and uses the REST API of GitHub and web scraping [81] to extract the information needed to build its internal domain model. It is implemented as a multiprocess application to speed up the scraping. Each process uses a different API key to access GitHub in parallel through
PyGitHub [23]. Parallelization is handled at project level: Each process gets a project from a queue and starts to fetch the data. Processes are also responsible for not exceeding GitHub rate limits associated with their API keys.

2.3 Manual Annotation

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

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

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

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

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

The project annotation page uses two side-by-side panes to present the README. The left one represents the raw Markdown version of the README. The right pane shows a partially rendered version (i.e., similar to what a user sees on GitHub).
2.4 Parsing Links: Strategy & Heuristics

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

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

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

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

2.5 Parsing README Histories

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

2.6 Community Size

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

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

2.7 Data Availability and Replication Package

We provide a replication package, publicly available on Figshare [51], containing the source code of DwarvenMail, the input dataset, the manually annotated projects, the serialized 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].

扮 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.

扮 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.

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

扮 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.

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

扮 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.

<table>
<thead>
<tr>
<th>Projects</th>
<th>Content Type</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>36 (60%)</td>
<td>General Community Hub</td>
<td>141</td>
</tr>
<tr>
<td>29 (46%)</td>
<td>Official Documentation</td>
<td>97</td>
</tr>
<tr>
<td>28 (47%)</td>
<td>License</td>
<td>68</td>
</tr>
<tr>
<td>25 (42%)</td>
<td>Contributing</td>
<td>56</td>
</tr>
<tr>
<td>23 (38%)</td>
<td>Issues</td>
<td>52</td>
</tr>
<tr>
<td>23 (38%)</td>
<td>CI/CD</td>
<td>50</td>
</tr>
<tr>
<td>21 (35%)</td>
<td>Project Repository</td>
<td>76</td>
</tr>
<tr>
<td>20 (33%)</td>
<td>Relevant Projects</td>
<td>152</td>
</tr>
<tr>
<td>20 (33%)</td>
<td>Dependency/Environment</td>
<td>84</td>
</tr>
<tr>
<td>19 (32%)</td>
<td>Releases</td>
<td>60</td>
</tr>
<tr>
<td>16 (27%)</td>
<td>In-Repository Resource</td>
<td>67</td>
</tr>
<tr>
<td>16 (27%)</td>
<td>Package Repository</td>
<td>47</td>
</tr>
<tr>
<td>14 (23%)</td>
<td>CI/CD &gt; Testing</td>
<td>24</td>
</tr>
<tr>
<td>13 (22%)</td>
<td>Installation Instructions</td>
<td>24</td>
</tr>
<tr>
<td>11 (18%)</td>
<td>Code Coverage</td>
<td>22</td>
</tr>
</tbody>
</table>

(b) Source Type.

<table>
<thead>
<tr>
<th>Projects</th>
<th>Source Type</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>55 (92%)</td>
<td>Homepage/Website</td>
<td>436</td>
</tr>
<tr>
<td>41 (64%)</td>
<td>Collaborative Platform</td>
<td>188</td>
</tr>
<tr>
<td>36 (60%)</td>
<td>Third Party Service</td>
<td>169</td>
</tr>
<tr>
<td>34 (57%)</td>
<td>Wiki</td>
<td>125</td>
</tr>
<tr>
<td>32 (53%)</td>
<td>Repository</td>
<td>166</td>
</tr>
<tr>
<td>28 (47%)</td>
<td>Sourcefile/Sourcefolder</td>
<td>151</td>
</tr>
<tr>
<td>25 (42%)</td>
<td>Instant Messaging</td>
<td>84</td>
</tr>
<tr>
<td>11 (18%)</td>
<td>Auxiliary README</td>
<td>21</td>
</tr>
<tr>
<td>10 (17%)</td>
<td>Readme Section/Anchor</td>
<td>44</td>
</tr>
<tr>
<td>9 (15%)</td>
<td>Mailing List</td>
<td>27</td>
</tr>
<tr>
<td>9 (15%)</td>
<td>Forum</td>
<td>20</td>
</tr>
<tr>
<td>8 (13%)</td>
<td>Image/GIF</td>
<td>21</td>
</tr>
<tr>
<td>8 (13%)</td>
<td>Blog</td>
<td>20</td>
</tr>
<tr>
<td>7 (12%)</td>
<td>Email Address</td>
<td>12</td>
</tr>
<tr>
<td>6 (10%)</td>
<td>Video</td>
<td>13</td>
</tr>
</tbody>
</table>

(c) Source Instance.

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

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 headquarters 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.png)

**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.

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.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discord</td>
<td>Voice, video, text messaging multimedia platform</td>
</tr>
<tr>
<td>Facebook</td>
<td>Social media and social networking service</td>
</tr>
<tr>
<td>Forum</td>
<td>General category for web based discussion sites</td>
</tr>
<tr>
<td>GitHub</td>
<td>GitHub infrastructure for project development</td>
</tr>
<tr>
<td>Gitter</td>
<td>Voice, video, text messaging multimedia platform</td>
</tr>
<tr>
<td>IRC</td>
<td>Text-based instant messaging chat system</td>
</tr>
<tr>
<td>LinkedIn</td>
<td>Business social media &amp; professional networking</td>
</tr>
<tr>
<td>Mailing List</td>
<td>E-mail based communication among recipients</td>
</tr>
<tr>
<td>Matrix</td>
<td>Communication protocol implemented by clients</td>
</tr>
<tr>
<td>Medium</td>
<td>Online publishing platform and social journalism</td>
</tr>
<tr>
<td>Reddit</td>
<td>Social news aggregation, rating, discussion, and multimedia sharing</td>
</tr>
<tr>
<td>Slack</td>
<td>Voice, video, text messaging multimedia platform</td>
</tr>
<tr>
<td>Spectrum</td>
<td>Text-based web instant messaging chat system</td>
</tr>
<tr>
<td>StackOverflow</td>
<td>Question and Answer website</td>
</tr>
<tr>
<td>Telegram</td>
<td>Voice, video, text messaging multimedia platform</td>
</tr>
<tr>
<td>Twitter</td>
<td>Social media and social networking service</td>
</tr>
<tr>
<td>Youtube</td>
<td>Video hosting and sharing platform</td>
</tr>
</tbody>
</table>

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

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

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

5 Coexistence and Competition

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

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

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

⚠️ 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.

Overall, 2,105 out of 3,280 projects (66%) using GitHub Issues, also have other communication platforms referenced in the README. Similar ratios are found, for example, for Discord with 801 out of 1,187 projects (67%).

多方通信平台可以有不同特性和针对不同的受众。为了覆盖开发或用户的需要，项目可以选择使用多种媒体在同一时间。核心开发人员在 README 中选择的数量和平台的多样性有什么？

图 6 显示了通信平台（在第 2.4 节中提供的 RE 生成的示例）的非穷尽列表，这些平台用于单独或结合使用。

约 38% 的项目在使用 Discord 或 Slack 时也包含 GitHub Issues。图 6a 中显示了这些 READMEs。

⚠️ GitHub Issues 可以在 README 中使用，而无需显式链接，只是项目的一个标签，如果允许的话。一些平台可能被隐含地假设为可用，即使它们不在 README 中。

总的来说，2,105 个（66%）使用 GitHub Issues 的项目，也包含其他通信平台的引用在 README 中。相似的比率是发现的，例如，对于 Discord，801 个（67%）1,187 个项目。

多方通信平台可以有不同类型的存在，并且可以共存。
GitHub has significant overlaps with the whole category of instant messaging, and with specific asynchronous platforms (e.g., StackOverflow, see Figure 6b). However, 1,270 projects rely only on the integrated support provided by GitHub.

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

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.

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

6 Instant Messaging: A Deep Dive

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

Instant messaging platforms fulfill a very specific role: Providing communication in real-time, possibly with rich media sharing capabilities (e.g., links, videos, files), and Voice over IP conferencing (i.e., VoIP). Two instances of these platforms are seldom found together. Similar characteristics, audiences, and usages make competition the prevailing paradigm.
Figure 6d shows that 97% of projects opting for these platforms choose between one of the three alternatives. Three projects include links (see Section 2.4) to all the platforms and also other instant messaging (e.g., Spectrum), but only PowerShell/PowerShell has a significant Discord community (more than 10k members).

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.

6.1 Gitter, Discord, and Slack: A Timeline

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

![Figure 7](image-url)

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

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

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

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

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.

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

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

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

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

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

6.3 Community Sizes

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

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.

6.4 Different Projects, Same Community

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

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

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.

6.5 Technical, Social, and Ethical Challenges

Some community platforms are public, some allow anonymous access, some require a form of registration or access permission. We found communities with (automatic) procedures to accept new members but, in general, it is hard to devise a general automatic “agent” to explore all of them.
The sheer amount of customization that is possible, even in a simple invite landing page of Slack, has been an obstacle to getting reliable data about communities lying behind those pages. Exploring a larger and more varied sample could shed light on platform dependent similarities and differences.

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.

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

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

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

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

7 Threats to Validity

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

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

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.

Limiting the extraction of the documentation landscape to what is reachable from the main README file (i.e., ignoring links to other READMEs in submodules of a project) poses a threat to construct validity. This threat is partially mitigated by the magnitude of the phenomenon we highlighted, emerging despite the limited scope, and calling for discussion and further investigation (i.e., also considering auxiliary documentation sources as a starting point to map the landscape).
Our analysis benefits from verifying the validity of links whenever possible (i.e., if the resource referred by the link is still available we expect an HTTP 200 OK response). When mining GitHub we verified the links we found in a two step process. The time interval between the first pass for scraping and the second pass for verification was short enough to guarantee that most links were in their intended state. Obsolete links may be possible and are part of the present study.

⚠️ 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.

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

💡 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.

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

💡 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.

8 Related Work

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

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

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

Nugroho et al. [40] studied how Eclipse developers utilize project forums, concluding that forums are essential platforms for linking various resources in the Eclipse ecosystem besides representing an important source of expert knowledge.
Modern social media are becoming another information source for development activities. Mezouar et al. [38] studied how tweets can improve the bug fixing process. They observed how issues for Firefox and Chrome are usually reported earlier through Twitter than on tracking systems. This can potentially decrease the lifespan of a bug. Guzman et al. [26] analyzed the usage characteristics, content, and automatic classification of tweets about software applications. They found that tweets contain useful information for software companies but stressed the need for automatic filtering of irrelevant information.

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

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

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

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

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

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

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


9 Conclusions and Future Work

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

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

References

Codecov.Codecov.URL:https://about.codecov.io/.


Discord, Inc. Discord. URL:https://discord.com/.


GitHub, Inc. PyGithub. URL:https://github.com/PyGithub/PyGithub.

Google, LLC. YouTube. URL:https://www.youtube.com/.


LinkedIn Corporation. LinkedIn. URL: https://www.linkedin.com.


Python Software Foundation. Python Package Index. URL: https://pypi.org/.


Marco Raglianti, Csaba Nagy, Roberto Minelli, Bin Lin, and Michele Lanza. Replication package. URL: https://figshare.com/s/33c8af53d6ba61d7c41.

Reddit. Reddit. URL: https://www.reddit.com/.

On the Rise of Modern Software Documentation


81 Zyte. Scrapy. URL: [https://scrapy.org](https://scrapy.org).