Automatic Detection and Analysis of Outdated Documentation in GitHub Repositories

Wen Siang Tan

A thesis submitted for the degree of
MASTER OF PHILOSOPHY
The University of Adelaide

June 30, 2023

Contents

List of Figures

List of Tables

University of Adelaide

Abstract

Automatic Detection and Analysis of Outdated Documentation in GitHub Repositories

by Wen Siang Tan

Outdated documentation is a pervasive problem in software development, preventing effective use of software, and misleading users and developers alike. We posit that one possible reason why documentation becomes out of sync so easily is that developers are unaware of when their source code modifications render the documentation obsolete. Ensuring that the documentation is always in sync with the source code takes considerable effort, especially for large codebases. To address this situation, we propose an approach that can automatically detect code element references that survive in the documentation after all source code instances have been deleted. In this work, we analysed more than 3,000 GitHub projects and found that most projects contain at least one outdated code element reference at some point in their history. We submitted GitHub issues to real-world projects containing outdated references detected by our approach, some of which have already led to documentation fixes. As an initiative toward keeping documentation in software repositories up-to-date, we have made our implementation available and created a tool for developers to scan their GitHub projects for outdated code element references. Lastly, we extended our approach to detect outdated references to code elements in over 2,000 images present in software documentation.

Declaration of Authorship

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I acknowledge that copyright of published works contained within this thesis resides with the copyright holder(s) of those works. I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Wen Siang Tan

JUNE 2023

Acknowledgements

I would like to express my deepest thanks to my supervisors, A/Prof Markus Wagner and Dr Christoph Treude, for their continuous support and guidance. This thesis would not have been possible without them. I am also extremely grateful to my friends who have accompanied me through the ups and downs over the course of this degree. Most importantly, I would like to express my gratitude to my parents for their relentless love, support and encouragement.

Chapter 1

Introduction

Outdated documentation is a common and well-known problem in software development (Lee et al., 2019). It hinders the effectiveness of documentation (Forward and Lethbridge, 2002), prevents developers from using APIs and libraries efficiently (Uddin and Robillard, 2015), contributes to software ageing (Parnas, 1994) and confusion (Kajko-Mattsson, 2005), and it demotivates newcomers (Steinmacher, Treude, and Gerosa, 2018). In a recent study on software documentation issues, Ahgajani et al. (Aghajani et al., 2019) found that "up-to-dateness problems" account for 39% of documentation content issues. Previous studies also revealed that more than two-thirds of participants surveyed believe that their system documentation is outdated (Souza, Anquetil, and Oliveira, 2005; Lethbridge, Singer, and Forward, 2003). Despite these findings, outdated documentation has remained an issue in the software engineering community due to the efforts needed to ensure that the documentation is in sync with the source code. Unlike source code, software documentation gets outdated "silently", i.e., there are no crashes or error messages to indicate that documentation is no longer up-to-date.[1] In many cases, developers are not aware that the source code changes they made have rendered the documentation outdated.

1.1 Related work

In this section, we review related work on the impact of outdated documentation, efforts in the area of code element resolution, and work on detecting inconsistencies between source code and documentation. Our work is the first to detect outdated documentation based on references to code elements that are no longer in sync.

1.1.1 Impact of outdated documentation

According to the Open Source Survey (Zlotnick, 2017), "incomplete or outdated documentation is a pervasive problem, observed by 93% of respondents, yet 60% of contributors say they rarely or never contribute to the documentation." In Sholler et al.'s 'Ten simple rules for helping newcomers become contributors to open projects' (Sholler et al., 2019), the authors include "Keep knowledge up-to-date and findable" as one of their rules, arguing that "outdated documentation may lead newcomers to a wrong understanding of the project, which is also demotivating. While it may be hard to keep material up-to-date, community members should at least remove or clearly mark outdated information. Signalling the absence or staleness of material can save newcomers time and also suggest opportunities for them to make contributions that they themselves would find useful."

[1] This is a well-known problem in software development, e.g., the documentation of tda-api states 'TDA might change them at any time, at which point this document will become silently out of date', see `https://tda-api.readthedocs.io/en/latest/client.html`.

Outdated software documentation is a form of technical debt (Kruchten, Nord, and Ozkaya, 2012) often referred to as documentation debt (Aldaeej, 2021). Rios et al. (Rios et al., 2020) list a number of effects of documentation debt, including low maintainability, delivery delay, rework, and low external quality, concluding that documentation debt affects several software development areas but especially requirements. With a similar focus on requirements, Mendes et al. (Mendes et al., 2016) report an extra maintenance effort caused by documentation debt of about 47% of the total effort estimated for developing a project and an extra cost of about 48% of the initial cost of the development phase. Compared to other types of technical debt, Liu et al. (Liu et al., 2021) found that documentation debt is less commonly and more slowly removed.

Motivated by these findings, the goal of our work is the automated detection of outdated documentation, based on the intuition that documents can be considered outdated if they contain references to code elements that used to be part of a project but are no longer contained in a repository.

1.1.2 Code element resolution

Code element resolution refers to techniques that resolve a general (typically ambiguous) mention of a potential code element (e.g., a class or a method) to its definition (Robillard et al., 2017). Code element resolution has been employed in the context of emails (Bacchelli, Lanza, and Robbes, 2010), tutorials (Dagenais and Robillard, 2012), or Stack Overflow (Rigby and Robillard, 2013), to name a few examples, often with the goal of linking relevant learning resources to code elements. Related work has also focused on automatically determining the importance of a code element mentioned in its context (e.g., in tutorial pages (Petrosyan, Robillard, and De Mori, 2015)) or on detecting errors in API documentation (Zhong and Su, 2013).

Supervised machine learning approaches are often used for code element resolution, usually aiming at a balance of precision and recall. In this work, we rely on an improved version of the regular expressions used for code element detection by Treude et al. (Treude, Robillard, and Dagenais, 2014) and then use a very strict filter (exact match) to find instances of the mentioned code element in the source code. While this may underestimate the number of actually outdated code element references, we err on the side of caution to not establish traceability links that we are not confident about.

1.1.3 Code-documentation inconsistencies

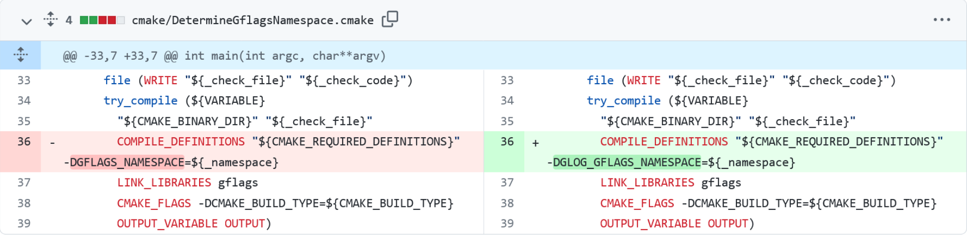
Inconsistencies between source code and its documentation have been the target of various research efforts over the past years, with a particular focus on source code comments. Wen et al. (Wen et al., 2019) presented a large-scale empirical study of code-comment inconsistencies, revealing causes such as deprecation and refactoring. In one of the first attempts to detect and fix such inconsistencies, Tan et al. (Tan et al., 2012) presented @tcomment for determining the correctness of Javadoc comments related to null values and exceptions. DocRef by Zhong and Su (Zhong and Su, 2013) was designed to detect inconsistencies between source code and API documentation, based on the use of island parsing to extract code elements and reporting mismatched code elements as errors. AdDoc by Dagenais and Robillard (Dagenais and Robillard, 2014) is a technique to identify code patterns in documentation using traceability links that can report new changes that do not conform to the code patterns of existing documentation. Also aimed at inconsistencies between source code and documentation, Ratol and Robillard (Ratol and Robillard, 2017) presented Fraco, a tool to detect source code comments that are fragile with respect to identifier renaming.

Zhou et al. (Zhou et al., 2020) presented DRONE, a framework that can automatically detect defects in Java API documentation and generate meaningful natural language recommendations. This is achieved through a combination of static program analysis, part-of-speech tagging, and constraint solving. Another related work is FreshDoc, which is an approach proposed by Lee et al. (Lee et al., 2019) to automatically update class, method, and field names in the API documentation. This is done by extracting code elements with a grammar parser and analysing different versions of the source code. More recently, Panthaplackel et al. (Panthaplackel et al., 2020) proposed an approach to automatically update existing comments when the source code is modified. This is accomplished by tokenising the comments and source code, and then modifying the comment tokens associated with the changes in source code.

In contrast to these related works, our approach detects outdated references to code elements in the documentation. To the best of our knowledge, there are currently no similar contributions for automatically detecting outdated documentation in software repositories when source code and documentation go out of sync.

1.2 Motivating example

The google/glog project[2] is one of the projects we found to contain outdated documentation. We detected an instance of the code element `DGFLAGS_NAMESPACE` in the source code[3] when the documentation was last updated. On 1 June 2018, the code element was renamed to `DGLOG_GFLAGS_NAMESPACE` in one of the commits[4] (Figure 1.1). However, the documentation[5] was not updated to reflect the changes. In the same project, another code element `fPIC` was found 21 times in the source code[6] when the documentation was last updated, but the document was not updated when all source code instances of the code element were deleted in this commit[7]. We reported the discrepancies by submitting a GitHub issue[8] to the project's repository (Figure 1.2). Following our report, the project maintainer fixed the outdated documentation by deleting the document containing the two outdated references.



```
@@ -33,7 +33,7 @@ int main(int argc, char**argv)
        file (WRITE "${_check_file}" "${_check_code}")
        try_compile (${VARIABLE}
          "${CMAKE_BINARY_DIR}" "${_check_file}"
-         COMPILE_DEFINITIONS "${CMAKE_REQUIRED_DEFINITIONS}"
 -DGFLAGS_NAMESPACE=${_namespace}
          LINK_LIBRARIES gflags
          CMAKE_FLAGS -DCMAKE_BUILD_TYPE=${CMAKE_BUILD_TYPE}
          OUTPUT_VARIABLE OUTPUT)
```

```
        file (WRITE "${_check_file}" "${_check_code}")
        try_compile (${VARIABLE}
          "${CMAKE_BINARY_DIR}" "${_check_file}"
+         COMPILE_DEFINITIONS "${CMAKE_REQUIRED_DEFINITIONS}"
 -DGLOG_GFLAGS_NAMESPACE=${_namespace}
          LINK_LIBRARIES gflags
          CMAKE_FLAGS -DCMAKE_BUILD_TYPE=${CMAKE_BUILD_TYPE}
          OUTPUT_VARIABLE OUTPUT)
```

Figure 1.1. Code element renamed to `DGLOG_GFLAGS_NAMESPACE`

[2] https://github.com/google/glog

[3] https://github.com/google/glog/blob/921651e97c3892e656287f1cfa923319f0799729/cmake/DetermineGflagsNamespace.cmake#L36

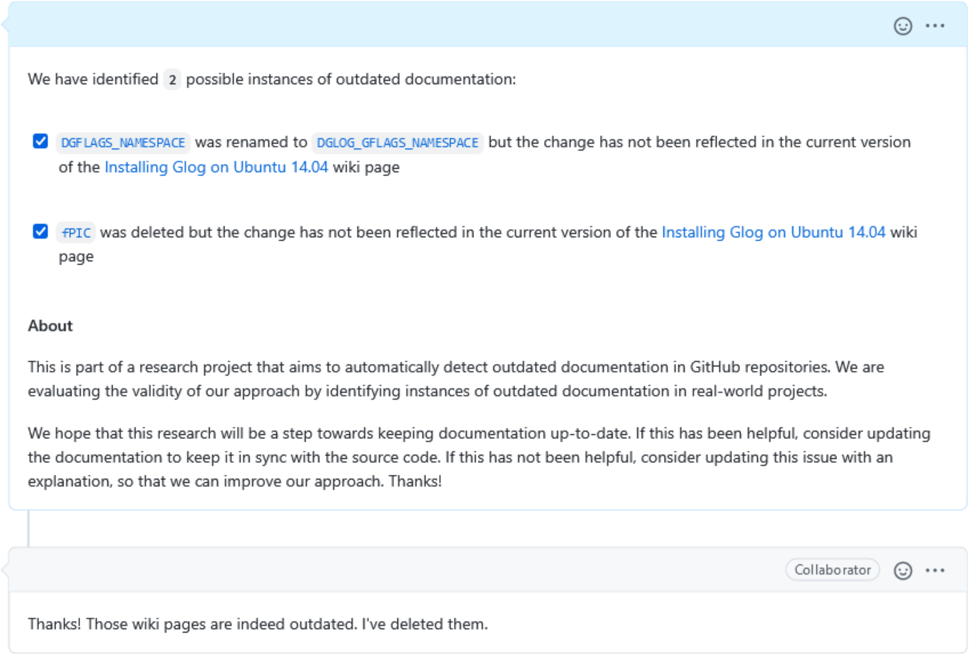
[4] https://github.com/google/glog/commit/abce78806c8a93d99cf63a5a44ff09873f46b56f

[5] https://github.com/google/glog/wiki/Installing-Glog-on-Ubuntu-14.04/aa4fc07826bca7edf4aae57acd53119e515f9963

[6] https://github.com/google/glog/blob/921651e97c3892e656287f1cfa923319f0799729/m4/libtool.m4#L3905

[7] https://github.com/google/glog/commit/b539557b3692c9c68d4e91d3cc920e8d14490d46

[8] https://github.com/google/glog/issues/750



We have identified 2 possible instances of outdated documentation:

☑ DGFLAGS_NAMESPACE was renamed to DGLOG_GFLAGS_NAMESPACE but the change has not been reflected in the current version of the Installing Glog on Ubuntu 14.04 wiki page

☑ fPIC was deleted but the change has not been reflected in the current version of the Installing Glog on Ubuntu 14.04 wiki page

About

This is part of a research project that aims to automatically detect outdated documentation in GitHub repositories. We are evaluating the validity of our approach by identifying instances of outdated documentation in real-world projects.

We hope that this research will be a step towards keeping documentation up-to-date. If this has been helpful, consider updating the documentation to keep it in sync with the source code. If this has not been helpful, consider updating this issue with an explanation, so that we can improve our approach. Thanks!

Thanks! Those wiki pages are indeed outdated. I've deleted them.

FIGURE 1.2. Screenshot of the GitHub issue submitted

Much like this motivating example, source code and documentation often remain out of sync for some time before getting discovered. Our approach can automatically detect such discrepancies and enable project maintainers to monitor how source code and documentation evolve. The next chapter will discuss our approach in detail: (1) the criteria used to select documentation such as the README file and wiki pages in the project, (2) the method used to detect code elements such as `DGFLAGS_NAMESPACE` and `fPIC` in the motivating example, (3) the steps needed to match code element references to actual instances in the source code, and (4) how the approach can be generalised to study the state of a project over time.

1.3 Contribution

This thesis proposes an automated approach that detects outdated references in README file and wiki pages of a GitHub project to help developers keep their documentation up-to-date. Other kinds of outdated documentation are beyond the scope of the thesis. We focus our analysis on GitHub since it gives us access to the documentation of a large number of projects in a consistent format. We analysed the current state and full history of documentation of more than 3,000 GitHub projects and found that 28.9% of the most popular projects on GitHub currently contain at least one outdated reference, with 82.3% of the projects being outdated at least once during the project's history. These references were typically outdated for years before they were fixed by project maintainers. To promote change in the software engineering community, we created a tool using GitHub Action that can automatically scan for outdated references whenever a new pull request is submitted to the repository. Finally, as images generally require more effort to continuously keep up-to-date, we examined over 2,000 unique images in software documentation and found that 14 projects contain outdated code element references in at least one image.

Chapter 2

Detecting Outdated Code Element References

The content of this chapter has been submitted to Empirical Software Engineering (EMSE) and is currently under review.[1]

2.1 Approach

To detect outdated code element references in software repositories, relevant pieces of documentation need to be identified first. We extract from the documentation a list of potentially outdated references to code elements and match them to actual instances in the source code. If a reference remains in the documentation after all instances have been deleted from the source code, we consider the documentation outdated. Figure 2.1 shows the overview of the approach, with the rest of this section describing this process in detail.

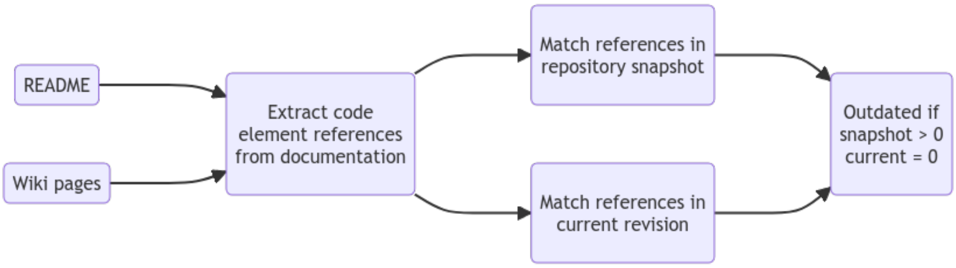

README
Wiki pages
Extract code element references from documentation
Match references in repository snapshot
Match references in current revision
Outdated if snapshot > 0 current = 0

FIGURE 2.1. Overview of the approach

2.1.1 Identifying documentation

GitHub provides two main forms of documentation for project maintainers to document their projects. The README file is a convenient way to introduce the project to users and contributors. In a study by Prana et al. (Prana et al., 2019) to categorise different types of content found in README files, the authors report that the majority of the README files from 393 randomly sampled projects contain some form of introduction or project background. In addition, README files often contain information for issues that may be encountered while using the project such as setup guides and API documentation. Project maintainers may also opt to make use of the wiki section for hosting documentation, which typically describes the project in more detail. One of the main differences between README and wiki is that the wiki may

[1] https://arxiv.org/abs/2212.01479

contain many pages while README is a single file. As any file types can be stored in GitHub wiki, only documentation written in file formats recognised by GitHub are considered in this work.[2]

We consider two datasets in this work. The first dataset consists of the 1,000 most popular projects on GitHub, ranked by the number of stars.[3] The second dataset consists of all 2,279 GitHub projects from Google.[4] Having two datasets allows us to gain insights into the differences between the state of documentation in popular projects maintained by the public and those maintained by a specific company. Figures 2.2 and 2.3 show the size distributions and the top programming languages of *top1000* and *google* projects. The list of project names for both datasets can be found in our online appendix.[5]

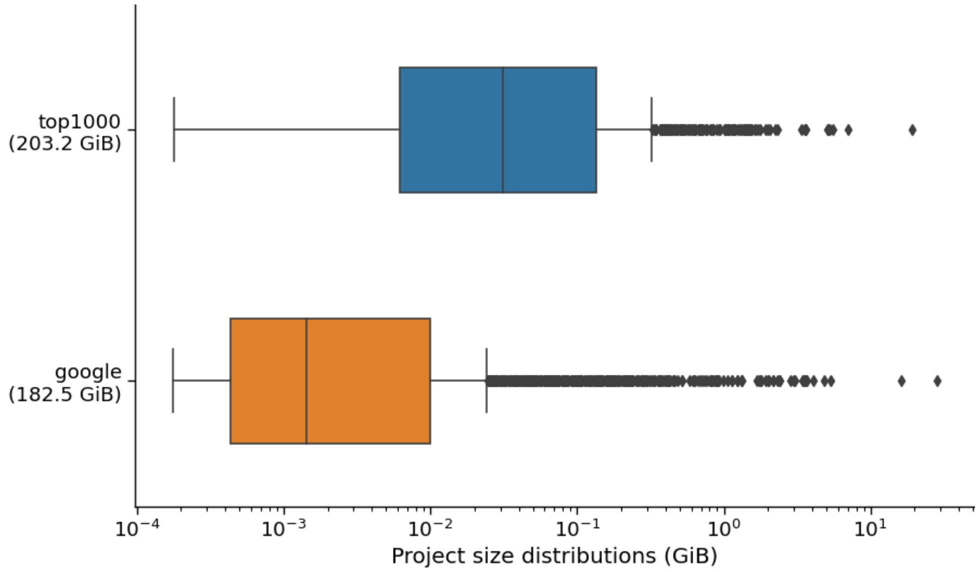


FIGURE 2.2. Project size distributions (GiB) for *top1000* and *google* projects in log scale

2.1.2 Extracting code elements

In Section 2.1.1, we identified a list of relevant documents from which we can extract potential outdated code element references. In this subsection, we outline the steps needed to extract such references from the documentation. These outdated references include variables, functions and class names found in the documentation. In this work, we use regular expressions to extract references to code elements in the documentation. Unlike parsers that are language-dependent, regular expressions can be used to extract possible candidates of outdated references in the documentation and matched to any source code files. We build on the work of Treude et al. (Treude, Robillard, and Dagenais, 2014) to extract code elements from the documentation using regular expressions, in which the authors have created a list of regular expressions to detect code elements.[6] As an example, one of the regular expressions `[A-Z][a-zA-Z]+?<[A-Z][a-zA-Z]*>` in that list is used to detect class templates such as the following:

[2] `https://github.com/github/markup`

[3] `https://gitstar-ranking.com/repositories`, project names collected on 20 June 2022

[4] `https://github.com/orgs/google/repositories`, project names collected on 20 June 2022

[5] `https://zenodo.org/record/7384588`

[6] `https://www.cs.mcgill.ca/~swevo/tasknavigator/`

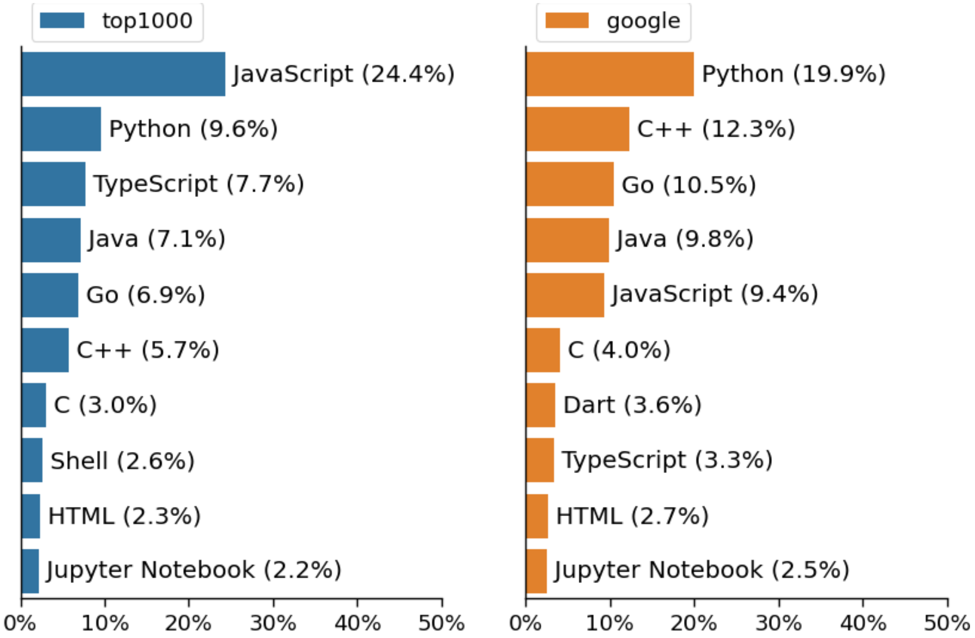


FIGURE 2.3. Top 10 programming languages used in *top1000* and *google* projects

- `Worker<T>`
- `ArrayList<String>`
- `Callback<SimpleResponse>`

To help improve the quality of the list of code element references extracted from the documentation, i.e. code elements that are also found in the source code, we extracted a list of code elements using the original regular expression list and manually annotated if the reference is outdated. Each author[7] annotated the same 50 randomly selected code elements[8] detected from the *google* projects to measure the inter-rater agreement. We achieved a free-marginal kappa of 0.92 when deciding whether the case is a true positive.

1. We consider a code element reference as not outdated (false positive) if it fits any of the following criteria:
   (a) The source code file and documentation have identical content, e.g. one of the projects in our dataset contained their entire documentation corpus twice: once in the wiki and once as .md files in the source code repository.
   (b) The code element reference extracted is a common word within the project (e.g. project name), a capitalised common word (PRIMARY, INACTIVE), an abbreviation (API, iOS), or a word that is not specific to the project (Data, User).
   (c) The code element reference extracted from the documentation is a URL or URL alt text.
   (d) The source code file is a text file that supposedly documents the project, e.g., an HTML file.

[7] Each author of the paper `https://arxiv.org/abs/2212.01479`

[8] Previous work have used lesser than 50 data points to measure inter-rater agreement such as `https://link.springer.com/article/10.1007/s10664-021-10058-6`

(e) The code element matched in the source code is part of a source code comment.

2. A reference is considered outdated (true positive) if the code element was found in a previous revision but has since been deleted:

   (a) The source code file exists in the current revision but the code element instance is deleted.

   (b) The source code file is deleted in the current revision.

During the manual annotation, we noticed that developers often use backticks (`) in Markdown to indicate code elements. We also observed that extracting URLs from the documentation produced many code element references that are not matched to source code instances in a later stage. With the manual annotation data, we made a few modifications to the regular expression list:

1. A regular expression to capture text enclosed in backticks is added. Code blocks (```) are not added as they often contain longer texts that are less likely to be matched.

2. A regular expression used to detect URLs in the original list is removed, URLs enclosed in backticks are still extracted.

3. Many regular expression groupings in the original list are modified to extract only the code element, preventing additional spaces that are not part of the code element from getting extracted.

The updated regular expression list used in this work can be found in our online appendix.[9]

2.1.3 Matching code elements

In the previous step, a list of potentially outdated references was extracted from the documentation using regular expressions. This subsection will describe the process of how these references are matched to actual instances in the source code to determine if they are outdated. In this work, a reference is considered outdated if the code element was found in both source code and documentation when the documentation was last updated, but the reference remains in the latest version of the documentation after all source code instances have been deleted (Table 2.1).

TABLE 2.1. What is outdated?

	Before	**After**
Documentation	✓	✓
Source code	✓	✗

To determine if a reference is currently outdated, we compare the number of instances found in two repository revisions. The first revision is the snapshot of the repository of when the documentation was last updated, and the second revision corresponds to the current revision of the repository. An instance is counted if it is a whole word, case-sensitive, and exact string match of the code element reference. If

[9] `https://zenodo.org/record/7384588`

the number of source code instances goes from a positive integer (i.e. at least one code element instance was found in the source code when the documentation was updated) to a zero (i.e. all source code instances have been deleted in the current revision), we flag the reference as outdated. Going back to the motivating example, the two code element references flagged as outdated have the following number of instances found in the snapshot and the current repository revision (Table 2.2).

Table 2.2. Number of source code instances for the two code element references from the motivating example

Code element	Repository snapshot	Current revision
DGFLAGS_NAMESPACE	1	0
fPIC	21	0

Linking references

On GitHub, a project's source code and wiki are stored separately in different Git repositories. We can get the snapshot of a project by interleaving the commit histories of both Git repositories: given a particular version of the documentation that is under investigation, we retrieve the most recent source code repository revision that was committed prior (Figure 2.4). In cases where the documentation is updated after the current repository revision, the snapshot refers to the current repository revision; this means that the number of instances found in both revisions are the same and the reference will not be flagged as outdated. This process is repeated for each code element reference extracted from the documentation to determine if the reference is currently outdated. Note that, as each page in the documentation may be updated at different times, code element references extracted from different pages may have a different repository snapshot.

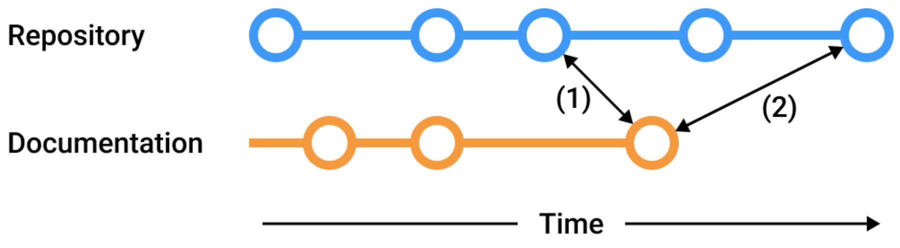


Figure 2.4. Linking the current documentation version to (1) repository snapshot and (2) current repository revision

File references

A code element reference may be incorrectly flagged as outdated when documentation references a file in the source code because file paths are often not explicitly written in the source code. To avoid flagging these cases as outdated, each variant of the file path that is an exact match of a code element reference is treated as an additional source code instance. In our implementation, a file path is considered a variant if it is a component of the file path including an optional slash at the beginning. For example, if the source code contains a file named `path/to/file.py`, all of the following variants are added to the list of code elements:

- /path/to/file.py
- path/to/file.py
- /to/file.py
- to/file.py
- /file.py
- file.py

2.1.4 Extending the analysis

The approach outlined in the previous subsections can be generalised to analyse the state of documentation throughout a project's entire history. To help describe the state of a reference to code element C at the time of revision R and in document D, we designed a symbolic representation for the extended analysis:

- **. (dot)** In revision R of the source code, document D did not exist.
- **- (dash)** In revision R of the source code, document D existed and it did not contain any references to C.
- **0** In revision R of the source code, document D existed and contained at least one reference to C and the source code did not contain any instances of C.
- **N** In revision R of the source code, document D existed and contained at least one reference to C and the source code contained an instance of C N times.

Table 2.3. Summary of symbolic representation used in the extended analysis

	Document existed in revision R	**Document has at least one reference**	**Number of source code instances**
. (dot)	✗		
- (dash)	✓	✗	
0	✓	✓	0
N	✓	✓	N

The symbolic representation can be summarised in Table 2.3. As an example, the first 50 revisions of the code element `renderFiles('./files')` in the README file from the vuejs/vue-cli project[10] have the following symbolic representation:

Table 2.4. Example of symbolic representation

. . . . - - - - - - - - - - - - - - - - - - 3 3 3 3 3 3 3 0 0 0 0 - - - - - - - -

[10] `https://github.com/vuejs/vue-cli`

- In the first 13 revisions, there is a dot (.) indicating that the README file did not yet exist.
- From revisions 14 to 31, there is a dash (-) indicating that the reference to the code element did not exist in the documentation (i.e., could not possibly be outdated).
- From revisions 32 to 38, there is a three (3) indicating that the reference to the code element existed in the documentation and was matched to three instances in the source code.
- From revisions 39 to 42, there is a zero (0) indicating that the reference to the code element existed in the documentation, but was no longer found in the source code (i.e., documentation was outdated).
- From revision 43 onward, there is a dash (-) again, indicating that the reference to the code element does not exist in the documentation anymore (i.e., documentation is no longer outdated).

Extending the linking process

To analyse the state of documentation throughout a project's history, we link each repository revision in the main branch to the next version of the documentation. Consistent with the method in Section 2.1.3, the current version of the documentation is linked to the same repository revisions. Figure 2.5 shows the links between repository revisions and their corresponding documentation versions.

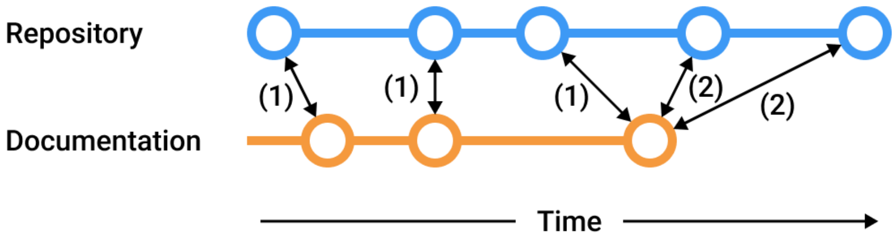


FIGURE 2.5. Linking each repository revision to a corresponding documentation version for repository commits made (1) before and (2) after the current documentation version

Flagging as outdated

Consider a scenario where the symbolic representation of a particular code element in seven consecutive revisions is `2 0 0 . 0 0 0`. Two source code instances were found in the first revision and subsequently removed. The documentation was accidentally deleted in the fourth revision (indicated by the dot) and then restored (back to zero). Following the definition of outdated in Section 2.1.3 (positive integer followed immediately by a zero) will fail to flag this code element as outdated. Even though no source code instances are found in the latest revision, the reference still remains in the documentation. Using the symbolic representation, we can more accurately define 'outdated' in the extended analysis. A code element is considered outdated if a positive integer is somewhere in front of a zero, even if it is not directly before the zero.

Creating a report

To make observing the trend of a code element throughout the project's history easier, we can record the number of code element instances found in each revision of the repository in a tabular form, grouped by their names and the documents from which they were extracted. Table 2.5 shows a small section of the report from the vuejs/vue-cli project. We can see that three instances of the code element `renderFiles('./files')` were found in revisions 37 and 38 followed by four zeros, which indicates that the code element reference was outdated from revisions 39 to 42. This was fixed in revision 43 when the outdated reference was deleted.

TABLE 2.5. A small section of the report generated from analysing the vuejs/vue-cli project (revision 37 to 43 for five code element references)

code element	R37	R38	R39	R40	R41	R42	R43
projectOptions	-	-	-	-	-	-	7
render('./template')	-	-	-	-	-	-	3
renderFiles('./files')	3	3	0	0	0	0	-
vue	198	205	205	205	205	210	210
vue-cli-service	14	14	14	14	14	15	15

2.2 Research questions

RQ1: What is the current state of documentation? Our first research question investigates the current state of documentation in open-source projects on code element, document and project levels. This includes the number of code element references that are currently outdated and the duration for which they have been outdated.

RQ2: What was the state of documentation during the projects' history? This research question aims to further explore the state of documentation by analysing the entire history of open-source projects. Similar to RQ1, we investigate the number of code element references that were outdated at some point in the project's history and the duration for which the outdated references typically survived in the documentation before getting fixed.

RQ3: How is outdated documentation resolved in projects? After investigating the state of documentation in RQ1 and RQ2, we ask RQ3 to gain insights on how outdated documentation is typically fixed in real-world open-source projects by comparing the number of outdated references resolved by updating the source code, deleting the outdated code element reference, or by deleting the documentation.

RQ4: How do open source projects respond to issues about outdated documentation? Our final research question examines how open-source project maintainers respond to our approach by creating GitHub issues highlighting the potentially outdated code element references detected in their projects.

2.3 Results

This section will discuss the research questions raised in the previous section: (1) the current state of documentation, (2) the state of documentation over time, (3) how outdated documentation is commonly fixed, (4) and the responses of open source projects to our approach.

We ran our analysis on projects in the two datasets introduced in Section 2.1.1. When cloning the repositories, one project[11] failed due to a large number of files. In the *top1000* dataset, the analyses of 8 projects were terminated after failing to finish in a day. Among the 991 successfully analysed projects, 265 projects contained at least one outdated reference in their current version, 653 projects did not contain any outdated references and the documentation of 73 projects did not contain any matches to any code element in the source code. In addition, 90.4% (896/991) of the *top1000* projects contained a README.md file and 60.0% (595/991) had at least one wiki page at the time of analysis. In the *google* dataset, the analysis of 1 project[12] was terminated after three days, leaving 2277 projects. The documentation of 101 projects was found to contain at least one outdated reference to a code element, the documentation of 1778 projects was up-to-date and the documentation of 398 projects did not contain code element references that were matched to the source code. 88.7% (2019/2277) projects used a README.md file and 13.0% (297/2277) used the wiki. Figure 2.6 shows the breakdown of the projects' statuses.

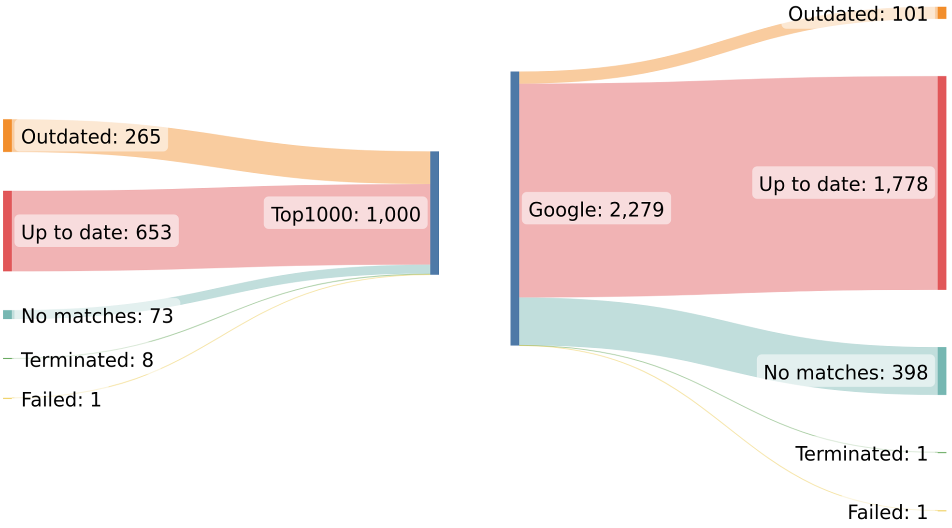


Figure 2.6. Analysis status of *top1000* and *google* projects, indicating whether a repository's documentation is currently out of date

2.3.1 RQ1: What is the current state of documentation?

To investigate the current state of documentation in open-source projects, we scanned projects using the approach described in Section 2.1 and counted the number of projects for which the documentation contained at least one outdated code element reference (see Figure 2.6). The same process is repeated at the document level to calculate the percentage of outdated documents. In addition, we can calculate the duration each code element reference is outdated for using the project's commit history.

In the *top1000* dataset, 3.9% (7910/201852) of the code element references detected are currently outdated. We found that 19.2% (1880/9784) of the documents

[11] https://github.com/google/material-design-icons
[12] https://github.com/google/swiftshader

contain at least one outdated reference to a code element, and 28.9% (265/918) of the projects contain at least one outdated document. In the *google* dataset, 2.7% (1283/48078) code element references, 9.7% (287/2947) documents, and 5.4% (101/1879) projects are currently outdated (Figure 2.7). On average, the references are currently outdated for 4.7 years for projects in the *top1000* dataset and 4.2 years for the *google* dataset (Figure 2.8).

RQ1 Summary Documentation of 28.9% *top1000* projects and 5.4% *google* projects were out of date at the time of analysis, with the references outdated for 4.7 and 4.2 years on average respectively.

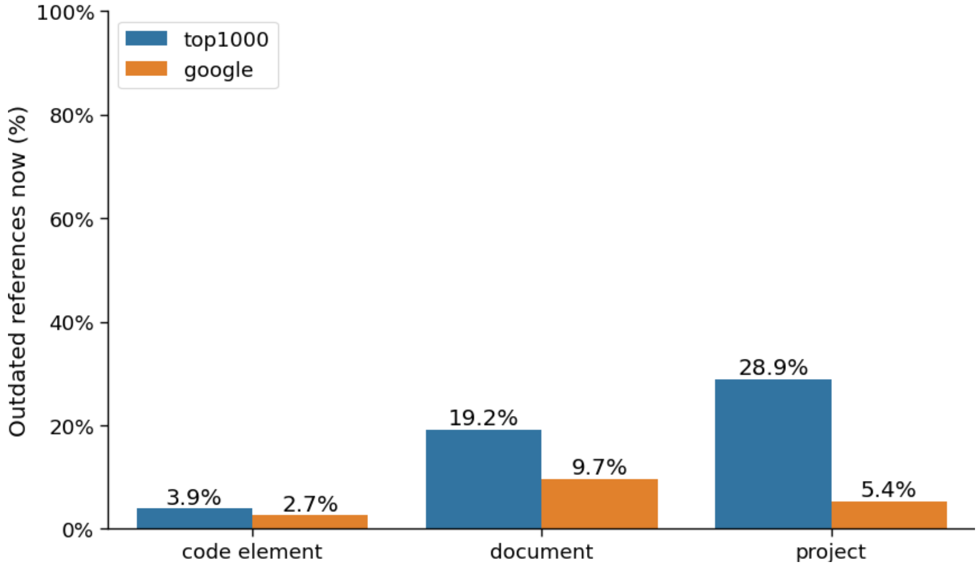


Figure 2.7. Percentage of references outdated at the time of analysis on code element, document and project levels

2.3.2 RQ2: What was the state of documentation during the projects' history?

To study how documentation evolves, we analysed the entire history of 800 projects from the *top1000* dataset. 82.3% (658/800) of the projects, 40.7% (2878/7071) of the documents, and 12.3% (23588/191849) of the code element references are found to be outdated at some point in history. In addition, 1.3% (2431/191849) of the code element references were outdated once again at some point in time after they were fixed. In addition, we analysed the full history of 1907 *google* projects. 29.7% (567/1907) projects, 30.6% (925/3018) documents and 7.1% (4176/58805) code elements were outdated sometime during the project's history (Figure 2.9). 0.4% (210/58805) code element references were outdated again at least once after they were fixed. Note that the number of analysed projects for the extended analysis is different from the normal analysis (Figure 2.10).

In addition to calculating the percentage of outdated documentation across project, document and code element levels, we calculated the duration of which outdated references survive in the documentation before getting fixed by project maintainers. Figure 2.11 contains only outdated code elements references that project

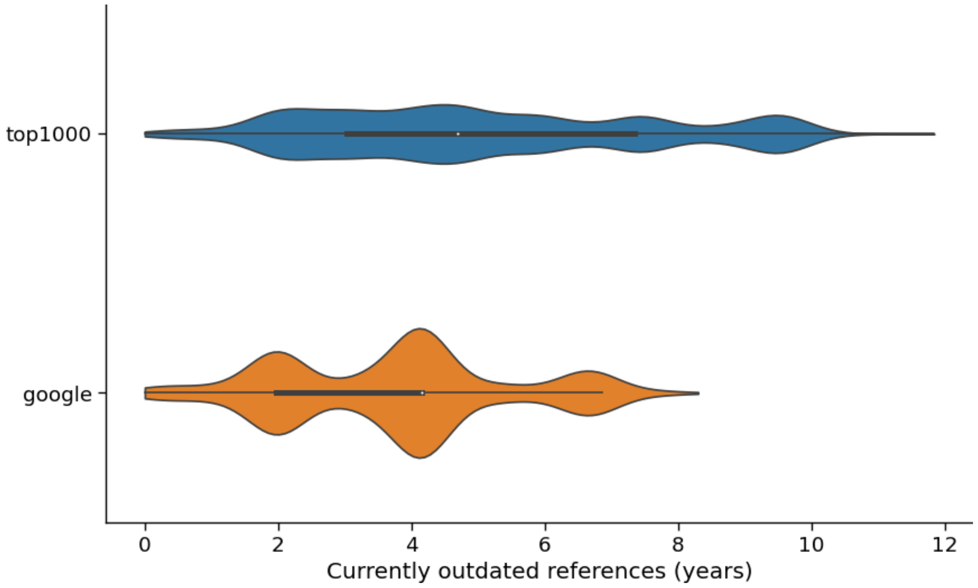


FIGURE 2.8. Distribution of duration that code element references have been outdated for at the time of analysis in *top1000* and *google* projects

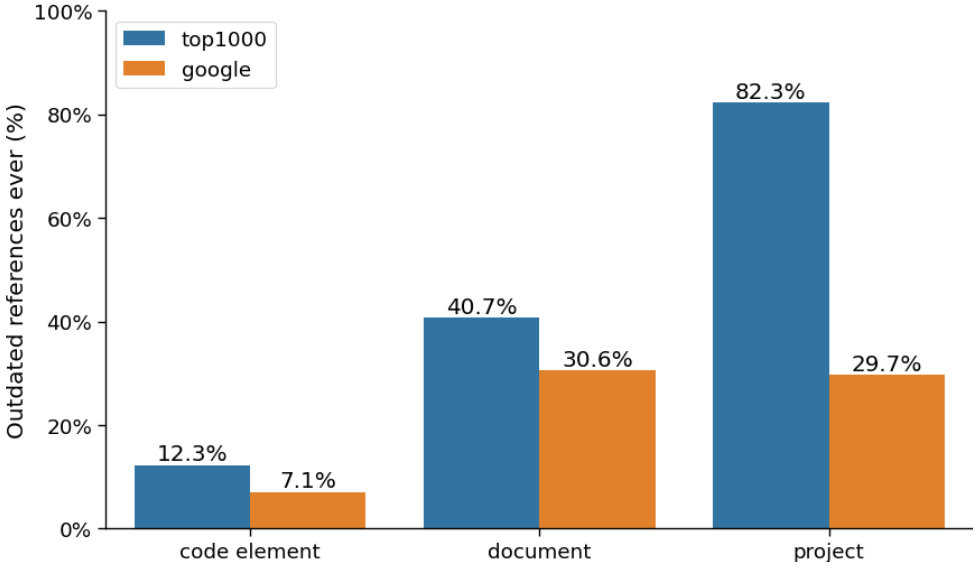


FIGURE 2.9. Percentage of references outdated at least once at some point during its history on code element, document and project levels

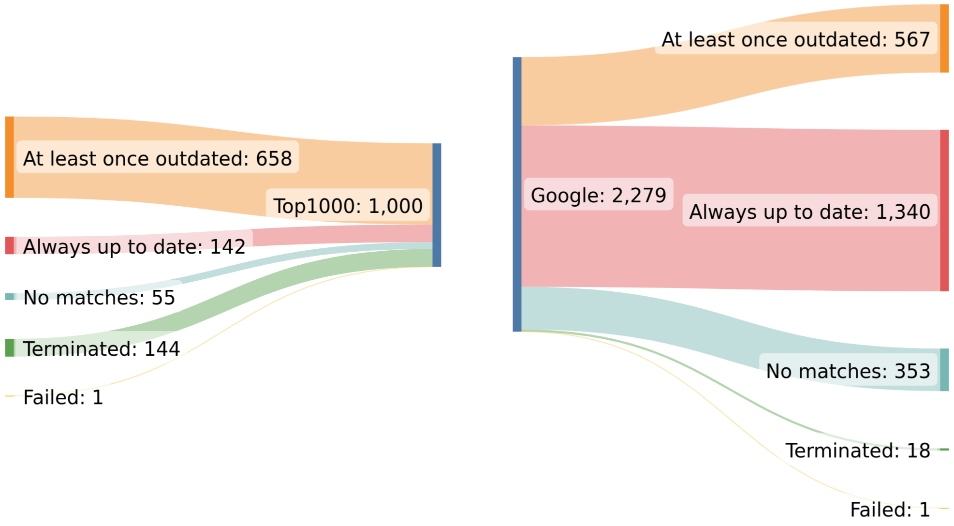


FIGURE 2.10. Extended analysis status of *top1000* and *google* projects, indicating whether a repository's documentation was outdated at some point during its history

maintainers have already fixed with a timestamp difference greater than zero.[13] The probability of surviving is calculated by the percentage of outdated code element references that were still present in the documentation after the duration indicated by the x-axis has passed. For example, outdated references have around 55% chance of surviving in *top1000* projects and 45% in *google* projects after a month.

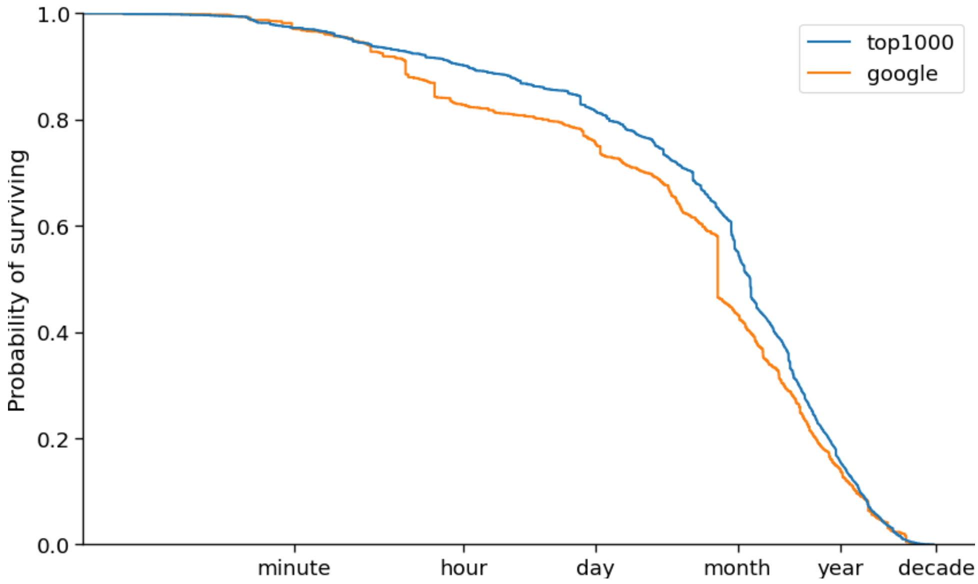


FIGURE 2.11. Time taken to fix outdated references in documentation for the *top1000* and *google* dataset in log scale

[13] The babel/babel project had 7 negative timestamp differences caused by reverting README.md to an earlier version.

RQ2 Summary Documentation of 82.3% *top1000* projects and 29.7% *google* projects were outdated at some point in history, with 1.3% and 0.4% references outdated once again respectively after they were fixed.

2.3.3 RQ3: How is outdated documentation resolved in projects?

There are three ways in which an outdated document can be resolved:

1. Source code is changed to reintroduce code element instances referenced by the documentation, making the documentation in sync again.
2. Documentation containing the outdated reference is updated to remove the outdated reference.
3. Documentation containing the outdated reference is deleted, thereby removing the outdated reference.

The three cases can be represented using the symbolic representation introduced in Section 2.1.4:

TABLE 2.6. Types of documentation fixes

	Before	**After**
Documentation delete	0	. (dot)
Documentation update	0	- (dash)
Source code change	0	N

Using the reports generated, we can study how the documentation was typically fixed throughout the project's history. For the *top1000* projects, we found that 73.6% (17368/23588) outdated references to code elements were resolved throughout the projects' histories, with 47.6% (8271/17368) fixed by changing the source code, 39.1% (6783/17368) by updating the documentation, and 13.3% (2314/17368) by deleting the documentation. For *google* projects, 55.5% (2319/4176) code element references were fixed by project maintainers. 50.2% (1164/2319) were fixed by code changes, 43.3% (1004/2319) by updating the documentation, and 6.5% (151/2319) by deleting the documentation.

RQ3 Summary Project maintainers most commonly resolve outdated documentation by changing the source code, followed by updating and deleting the document to remove the outdated reference.

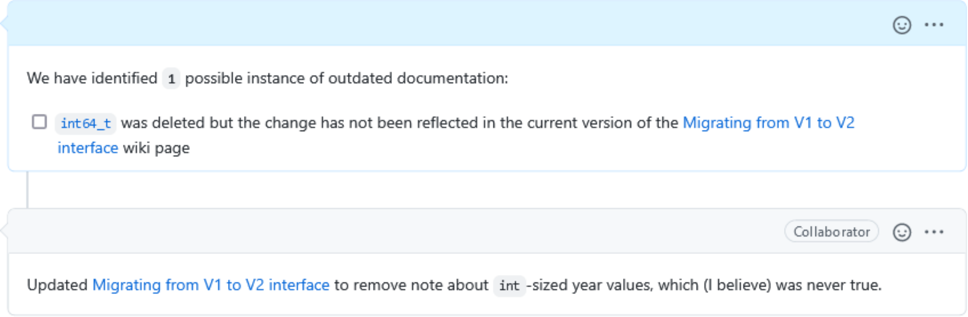
2.3.4 RQ4: How do open source projects respond to issues about outdated documentation?

To examine the usefulness of our approach in real-world projects, we submitted GitHub issues to projects containing outdated references detected by our approach. In contrast to pull requests, creating an issue allows project maintainers to decide whether to delete the outdated reference in the documentation or update the documentation to reflect the changes made in the source code. Based on the manual annotation in Section 2.1.2, we filtered projects from the *google* dataset with at least one true positive and further narrowed them down to 15 actively maintained projects that have had new commits within the past year.

In the issues, we listed the outdated references with links to the documentation and an instance of the code element found in the source code. At the time of writing, 4 projects have responded positively, while the other 4 reported the issues as false positives. 7 projects have not yet responded to our GitHub issues. Across the 15 projects, we reported 19 instances of outdated documentation, 5 of which have been fixed by project maintainers. The following subsections will discuss two true positives and two false positives.

True positives

The cctz project was one of the projects that responded positively to our GitHub issue.[14] In one of the commits, the code element instance `int64_t` was removed entirely from the source code but the reference to the code element remained in the documentation. The project maintainer responded to our GitHub issue and updated the documentation to reflect the changes in the source code (Figure 2.12). In the hs-portray project, the function `prettyShow` was renamed to `showPortrayal` in the source code, but the README file was not updated (Figure 2.13). We alerted the developers of this discrepancy, and the issue was fixed subsequently.[15]

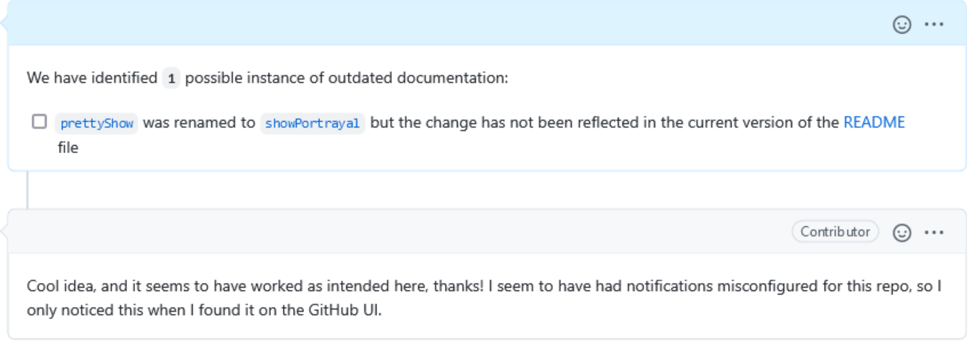


We have identified 1 possible instance of outdated documentation:

□ int64_t was deleted but the change has not been reflected in the current version of the Migrating from V1 to V2 interface wiki page

Updated Migrating from V1 to V2 interface to remove note about int -sized year values, which (I believe) was never true.

FIGURE 2.12. True positive: data type updated in the documentation



We have identified 1 possible instance of outdated documentation:

□ prettyShow was renamed to showPortrayal but the change has not been reflected in the current version of the README file

Cool idea, and it seems to have worked as intended here, thanks! I seem to have had notifications misconfigured for this repo, so I only noticed this when I found it on the GitHub UI.

FIGURE 2.13. True positive: function name updated in the documentation

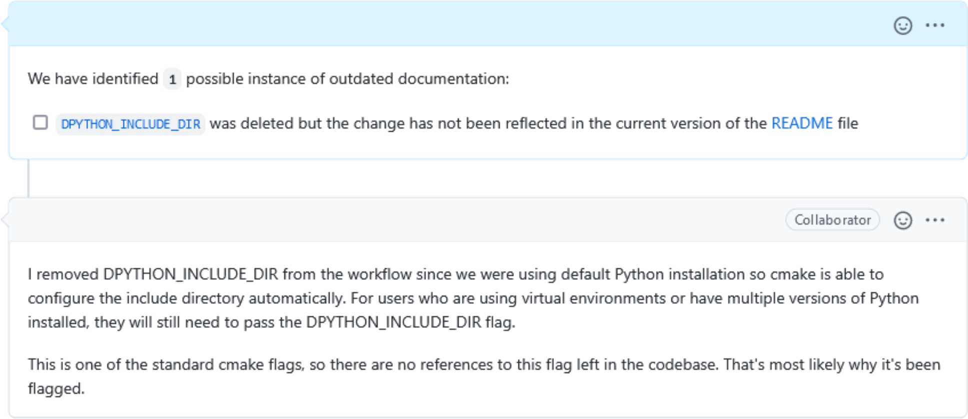
[14] https://github.com/google/cctz/issues/210

[15] https://github.com/google/hs-portray/issues/7

False positives

In one of the projects (Figure 2.14), a CMake flag was removed from the source code but the reference was not updated in the documentation. The project maintainers responded that the flag is no longer required in the source code but the documentation is still relevant for users that have installed multiple Python versions to configure the installation directory correctly.[16] A false positive was reported in another project (Figure 2.15) where the code element instance `text_out` was deleted from the source code. Although the code element reference is not explicitly written in the source code, the functionality remains in the program logic which results in the code element reference getting falsely flagged as outdated.[17]

> **RQ4 Summary** Several project maintainers responded positively to our GitHub issues and resolved the outdated references by updating or deleting the corresponding documents.



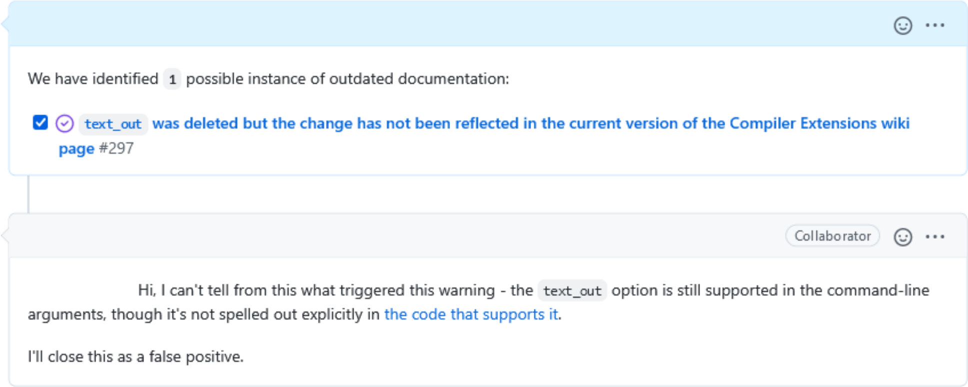
We have identified 1 possible instance of outdated documentation:

□ `DPYTHON_INCLUDE_DIR` was deleted but the change has not been reflected in the current version of the README file

I removed DPYTHON_INCLUDE_DIR from the workflow since we were using default Python installation so cmake is able to configure the include directory automatically. For users who are using virtual environments or have multiple versions of Python installed, they will still need to pass the DPYTHON_INCLUDE_DIR flag.

This is one of the standard cmake flags, so there are no references to this flag left in the codebase. That's most likely why it's been flagged.

FIGURE 2.14. False positive: still relevant for users with multiple Python versions



We have identified 1 possible instance of outdated documentation:

☑ `text_out` was deleted but the change has not been reflected in the current version of the Compiler Extensions wiki page #297

Hi, I can't tell from this what triggered this warning - the `text_out` option is still supported in the command-line arguments, though it's not spelled out explicitly in the code that supports it.

I'll close this as a false positive.

FIGURE 2.15. False positive: functionality remains in the program logic

[16] https://github.com/google/clif/issues/52

[17] https://github.com/google/gnostic/issues/273

2.4 Discussion

In this section, we will discuss our findings and the interesting differences between the two datasets used in this work. We investigated the current state of documentation in open-source software repositories and found that, on average, the *top1000* projects contain more outdated references than *google* projects at the time of analysis. The references have also been outdated longer in the *top1000* projects (4.7 years) compared to *google* projects (4.2 years). In the *top1000* dataset, 28.9% of the projects were found to contain at least one outdated code element reference in contrast to 5.4% of the *google* projects. We hypothesise that this is because *google* projects are generally smaller in size (median of 31.7 MiB for *top1000* projects and 1.47 MiB for *google* projects), and hence easier for project maintainers to keep their documentation up-to-date.

In RQ2, we reviewed the full history of 800 *top1000* projects and 1907 *google* projects. We found that 12.3% and 7.1% of the references to code elements detected respectively were outdated at some point in history, with the proportion higher on document and project levels. We investigated the sudden drops in survival probability for *google* projects (Figure 2.11) and discovered that the biggest drop around the one month mark was caused by project maintainers deleting[18] and restoring[19] large amount of source code files.

Next in RQ3, we looked into how open-source project maintainers usually resolve their outdated documentation. In our findings, approximately half of the fixes were attributed to source code changes. This is because the action of mass deleting and restoring source code files was interpreted as a fix caused by source code changes. We can also observe in various reports that the number of code element instances found in the source code suddenly drops to 0 and back to the original count.

Finally in RQ4, we examined the usefulness of our approach in real-world projects by alerting developers from 15 different Google projects of potential outdated references in their documentation where several project maintainers have responded positively to our GitHub issues. By using the implementation available in our online appendix, developers can scan for code element references that are potentially outdated in their GitHub project's documentation.

Although the content of this thesis is centred around detecting outdated code element references in documentation hosted on GitHub, our approach can be generalised to other version control platforms. In the next chapter of the thesis, we will present our publicly available tool that developers can utilise to scan for outdated references in their documentation.

[18] `https://github.com/google/j2objc/commit/f9ff221f9eb8aacaecf057e3e9a1ca7c4e8a5beb`

[19] `https://github.com/google/j2objc/commit/592382e0bf314134fac9bfee862dacca50fccdb1`

Chapter 3

Automated Tool for Outdated Documentation Detection

3.1 Motivation

Although documentation gets outdated without warnings, developers can take steps to keep their documentation up-to-date by checking if the documentation needs to be updated whenever changes are made to the source code. The implementation of our approach in Chapter 2 called DOCER (Detecting Outdated Code Element References) available in our online appendix[1] allows developers to avoid manually checking whether the source code modifications they made will lead to outdated documentation. Running the script extracts code element references from the documentation and reports the number of code element instances found in the source code. The generated report includes additional information such as URLs to the source code, commit timestamps and SHAs to help developers investigate why a reference was flagged as outdated. However, running the script whenever new changes are proposed may be mundane and repetitive. To simplify this process further, we created a workflow that is automatically triggered when a pull request is submitted to the repository. Figure 3.1 shows the automated steps carried out by the workflow.

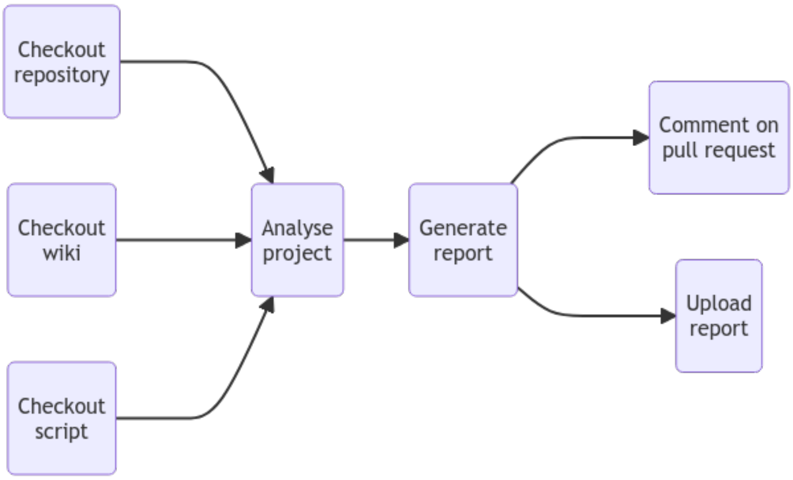


FIGURE 3.1. Overview of the automated workflow

[1] https://zenodo.org/record/7384588

3.2 Implementation

GitHub Action[2] is a feature on GitHub that allows developers to automate workflows based on events, commonly used for building a pipeline for Continuous Integration and Continuous Delivery (CI/CD). We created the tool using GitHub Action because it provides developers a convenient way to integrate the tool with existing GitHub projects. Developers can also configure their projects to automatically run the tool to scan for outdated code element references whenever there is a new pull request.

The workflow is defined by a YAML file[3] containing a series of actions that gets executed when the workflow is triggered. At the start of the YAML file, we list the name of the workflow, the events that will trigger the workflow, followed by the name of the GitHub-hosted runner[4] to use. In our case, the workflow is named DOCER, triggered only by pull requests and specified to run on the latest Long Term Support (LTS) version of Ubuntu.

```yaml
name: DOCER

on: pull_request

jobs:
  run:
    runs-on: ubuntu-latest
    steps:
```

The rest of the file defines the steps to execute in the workflow. Three repositories are cloned on the runner (repositories containing the source code, wiki pages, and scripts for the analysis) using a GitHub action named checkout.[5]

```yaml
- name: Checkout repository
  uses: actions/checkout@v3
  with:
    repository: ${{ github.repository }}
    ref: ${{ github.event.pull_request.head.sha }}
    path: repo
    fetch-depth: 0

- name: Checkout wiki
  continue-on-error: true
  uses: actions/checkout@v3
  with:
    repository: ${{ github.repository }}.wiki
    path: wiki

- name: Checkout tool
  uses: actions/checkout@v3
  with:
```

[2] https://github.com/features/actions

[3] https://yaml.org/

[4] https://docs.github.com/en/actions/using-github-hosted-runners/about-github-hosted-runners

[5] https://github.com/actions/checkout

```
    repository: wesleytanws/DOCER_tool
    path: tool
```

After cloning the repositories, the runner has all the files required to scan for outdated references. The workflow then commences the analysis, installs the necessary Python packages, generates the report and stores it in an environment variable.

```
- name: Run tool
  run: |
    bash tool/analysis.sh

    pip install pandas
    pip install numpy

    echo 'report<<EOF' >> $GITHUB_ENV
    python tool/report.py ${{ github.repository }} \
        ${{ github.run_id }} >> $GITHUB_ENV
    echo 'EOF' >> $GITHUB_ENV
```

In the case where merging the pull request may result in outdated documentation, the workflow uses a GitHub action named github-script[6] to post a comment on the pull request listing the potentially outdated references.

```
- name: Comment on pull request
  if: ${{ env.report }}
  uses: actions/github-script@v6
  env:
    report: ${{ env.report }}
  with:
    script: |
      github.rest.issues.createComment({
        issue_number: context.issue.number,
        owner: context.repo.owner,
        repo: context.repo.repo,
        body: process.env.report
      })
```

Depending on the number of modifications in the pull request, it may be difficult to figure out why a code element reference has been flagged as potentially outdated. This final step uploads the report and summary files to GitHub using a GitHub action named upload-artifact[7], allowing developers to view the full report.

```
- name: Upload artifact
  if: ${{ env.report }}
  uses: actions/upload-artifact@v3
  with:
    name: report
    path: |
      output/report.csv
```

[6] https://github.com/actions/github-script

[7] https://github.com/actions/upload-artifact

```
      output/summary.csv
      output/summary.md
```

The repository including the workflow introduced above and source code for the tool named DOCER_tool is publicly available on GitHub.[8]

3.3 Adding to GitHub projects

To demonstrate how the GitHub Action tool works, we will integrate the tool with an example repository with three files (Figure 3.2):

- `README.md` documents the functions defined in arithmetic.py
- `arithmetic.py` defines the arithmetic functions
- `main.py` calls the functions defined in arithmetic.py

Integrating the tool to a repository is as convenient as copying the YAML file defining the workflow[9] to the .github/workflows folder. Suppose a pull request as shown in Figure 3.3 is submitted to the repository.

Looking at the the pull request submitted, two files in the repository have been modified. In arithmetic.py, the subtract and divide functions were removed and a new power function was added. Similarly, the main.py file was modified to remove the subtract function and the chained multiply functions were refactored into a power function. Notice that the tool reports that continuing to merge the pull request may result in two outdated references in the documentation (Figure 3.4). This is because the divide and subtract functions were deleted from the source code but the README file was not updated to mention the changes.

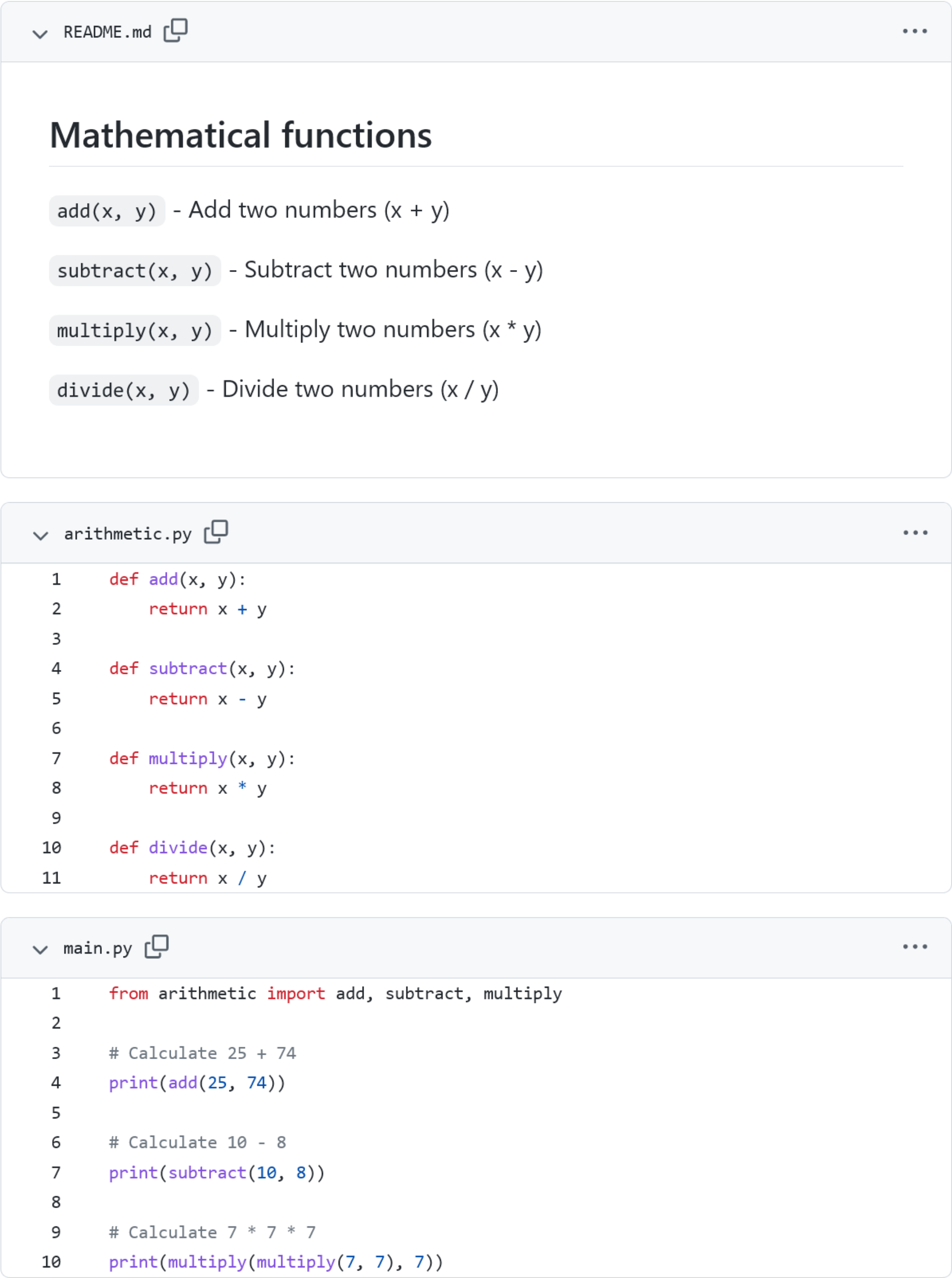
To keep the documentation up-to-date, we can simply remove the two outdated references in the README file. Better still, we can document the new function and mention that the two functions are now deprecated as shown in Figure 3.5.

3.4 Excluding code elements

One useful feature that we added to the tool is the ability to exclude certain code elements from the report, which allows developers to stop keeping track of code elements that have been determined to be false positives. Developers can add a list of code elements separated by newlines in a file named `.DOCER_exclude` located at the root of the repository. Code elements in the exclude list will be ignored by the tool when scanning for outdated references. The next chapter will discuss how the approach can be extended to detect outdated references in images that may be more prone to being outdated.

[8] `https://github.com/wesleytanws/DOCER_tool/tree/v1.0.0`

[9] `https://github.com/wesleytanws/DOCER_tool/blob/v1.0.0/DOCER.yml`



README.md

Mathematical functions

`add(x, y)` - Add two numbers (x + y)

`subtract(x, y)` - Subtract two numbers (x - y)

`multiply(x, y)` - Multiply two numbers (x * y)

`divide(x, y)` - Divide two numbers (x / y)

arithmetic.py

```python
def add(x, y):
    return x + y

def subtract(x, y):
    return x - y

def multiply(x, y):
    return x * y

def divide(x, y):
    return x / y
```

main.py

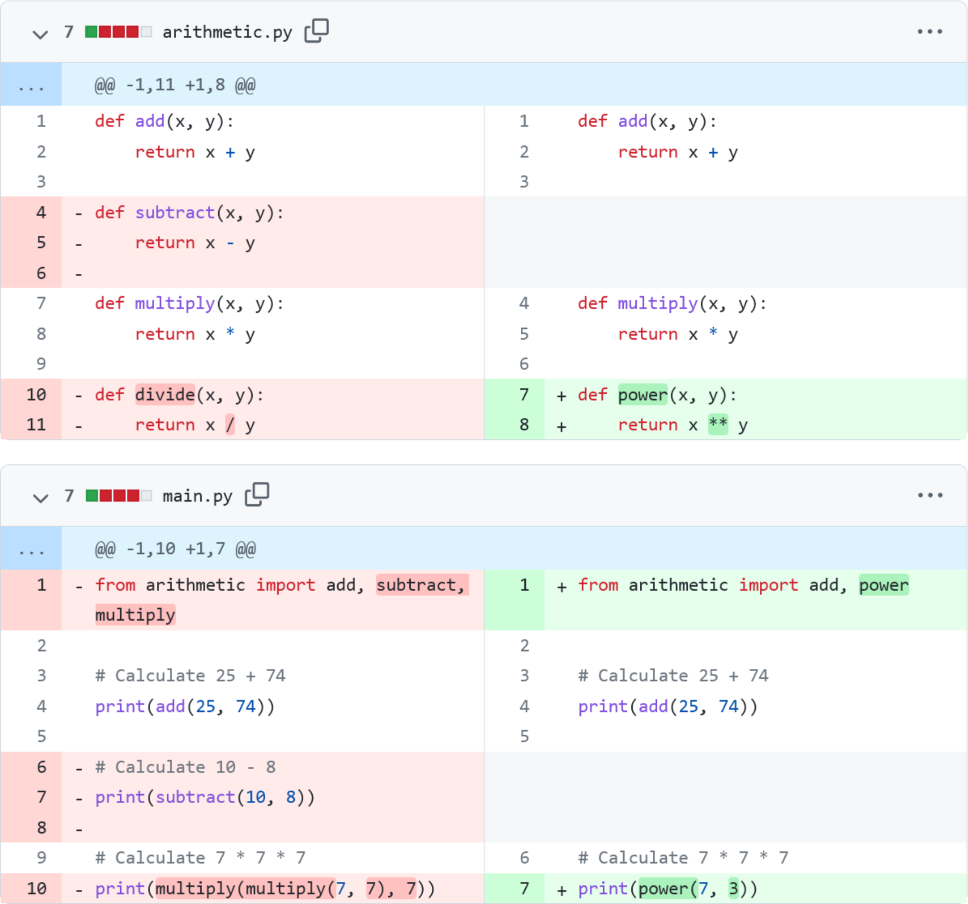
```python
from arithmetic import add, subtract, multiply

# Calculate 25 + 74
print(add(25, 74))

# Calculate 10 - 8
print(subtract(10, 8))

# Calculate 7 * 7 * 7
print(multiply(multiply(7, 7), 7))
```

FIGURE 3.2. Files in the example repository for tool demonstration



7 arithmetic.py

```
@@ -1,11 +1,8 @@
 def add(x, y):
     return x + y

-def subtract(x, y):
-    return x - y
-
 def multiply(x, y):
     return x * y

-def divide(x, y):
-    return x / y
+def power(x, y):
+    return x ** y
```

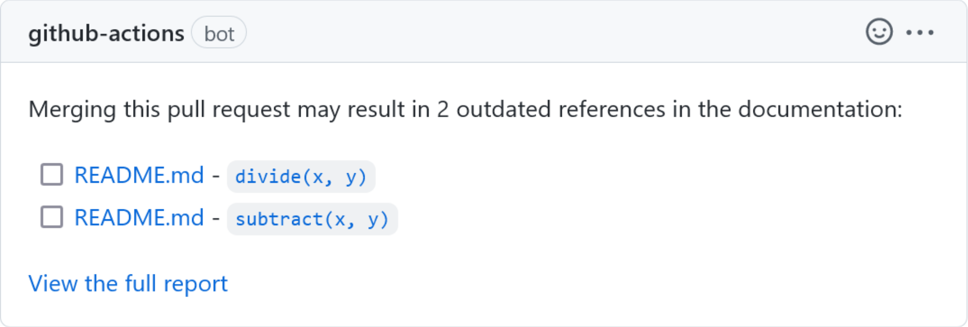
7 main.py

```
@@ -1,10 +1,7 @@
-from arithmetic import add, subtract, multiply
+from arithmetic import add, power

 # Calculate 25 + 74
 print(add(25, 74))

-# Calculate 10 - 8
-print(subtract(10, 8))
-
 # Calculate 7 * 7 * 7
-print(multiply(multiply(7, 7), 7))
+print(power(7, 3))
```

FIGURE 3.3. Pull request showing the incoming changes



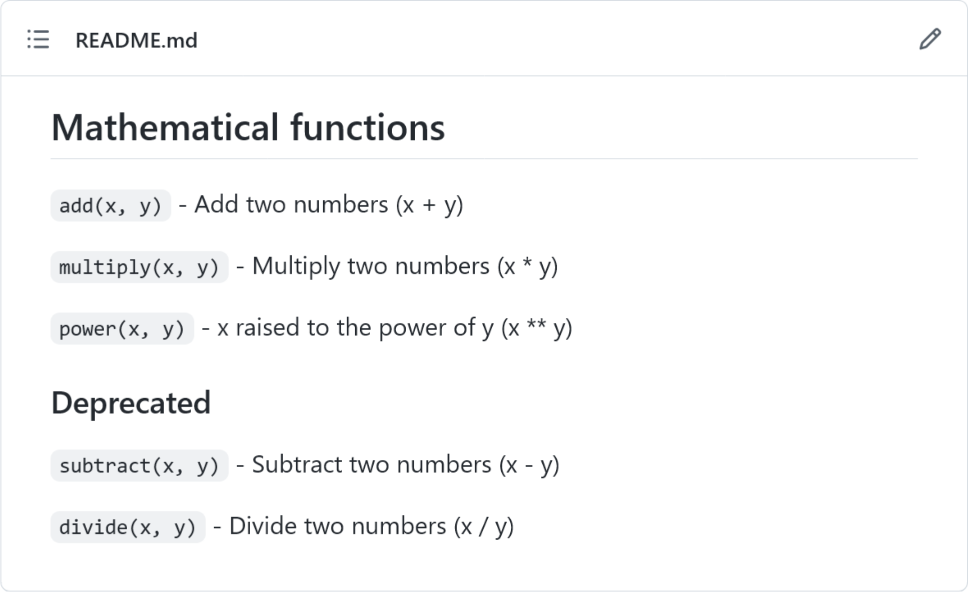
github-actions bot

Merging this pull request may result in 2 outdated references in the documentation:

- □ README.md - `divide(x, y)`
- □ README.md - `subtract(x, y)`

View the full report

FIGURE 3.4. Comment on the pull request listing the potentially outdated code element references



README.md

Mathematical functions

`add(x, y)` - Add two numbers (x + y)

`multiply(x, y)` - Multiply two numbers (x * y)

`power(x, y)` - x raised to the power of y (x ** y)

Deprecated

`subtract(x, y)` - Subtract two numbers (x - y)

`divide(x, y)` - Divide two numbers (x / y)

FIGURE 3.5. Updated README file including the new power function and listing the deleted functions as deprecated

Chapter 4

Outdated References in Images

4.1 Approach

References in images may be more prone to being outdated as they generally require more effort to continuously keep up-to-date. Similar to our approach in Chapter 2, we extract code elements from the documentation and match them to the source code, but with texts extracted from images. To get a collection of images that are in README and wiki pages, we used a markdown parser and a HTML parser to extract image links from the documents.

As our approach relies on text matching, we use Optical Character Recognition (OCR) to extract texts from images to detect potentially outdated references. We experimented with various OCR services and found that among the services that provide an API, OCRSpace[1] extracted most keywords and contained least noise. Table 4.1 shows the comparison between the texts extracted by different OCR services using an example image (Figure 4.1) hosted on GitHub wiki.[2]

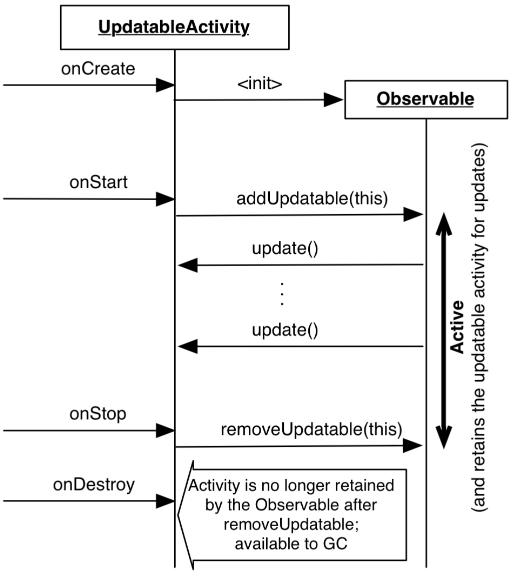

UpdatableActivity
onCreate
<init>
Observable
onStart
addUpdatable(this)
update()
update()
onStop
removeUpdatable(this)
onDestroy
Activity is no longer retained by the Observable after removeUpdatable; available to GC
Active
(and retains the updatable activity for updates)

FIGURE 4.1. Example image hosted on GitHub wiki

[1] https://ocr.space/ocrapi

[2] https://github.com/google/agera/wiki/Observables-and-updatables/ca03cd96fa90986fbe555c91d5fd426175fa3793

Table 4.1. Text extraction comparison between different OCR services

ocr.space	**newocr.com**	**convertio.co/ocr**
UpdatableActivity	UpdatableActivity	UpdatableActivity
onCreate	onCreate -	onCreate
onStart	<init>	onStart
onStop	Observable	onStop
onDestroy	a	onDestroy
<init>	2	<init>
Observable	onStart . 3	Observable
addUpdatable(this)	addUpdatable(this) a.	addllpdatable(this)
update()	$$ ___ii———_	update()
update()	2	removellpdatable(this)
removeUpdatable(this)	update() 2	/Activity is no longer retained by the Observable after removellpdatable; available to GC
Activity is no longer retained	a >	0
by the Observable after	oO	0
removeUpdatable;	© ©	H—’
available to GC	. ®	03
o	> —	"O
o	update() 33	Q_
0	$yY—_ w	=3
	<5	O
	Qo	03
	=>	2J
	@	” -Q
	—	o B
	onStop *	< 03 "O Q_ =3
	removeUpdatable(this) c	0
	O	0
	®	_C
	onDestroy Activity is no longer retained 3S	03
	by the Observable after &	0
	removeUpdatable;	"O
	available to GC	c
		0

4.2 Research question

RQ1: What is the current state of images in documentation? This research question investigates the current state of images in documentation. This includes the number of code element references found in images that are currently outdated and the duration for which they have been outdated on code element, document and project levels.

4.3 Results

4.3.1 RQ1: What is the current state of images in documentation?

To analyse the current state of images in documentation, we used 2279 projects from the same *google* dataset introduced in Section 2.1.1 and extracted images from the most recent version of README.md and wiki pages at the time of analysis. We were not able to clone one project[3] as there were too many files in the repository history. 1542 projects did not contain any image references, while the remaining 736 projects contained at least one image reference in their documentation. From the documentation of 736 projects, we extracted a total 2119 image references. There were 2098 unique images: 1726 unique images extracted from README and 372 from wiki pages. Figure 4.2 shows the breakdown of *google* projects' statuses.

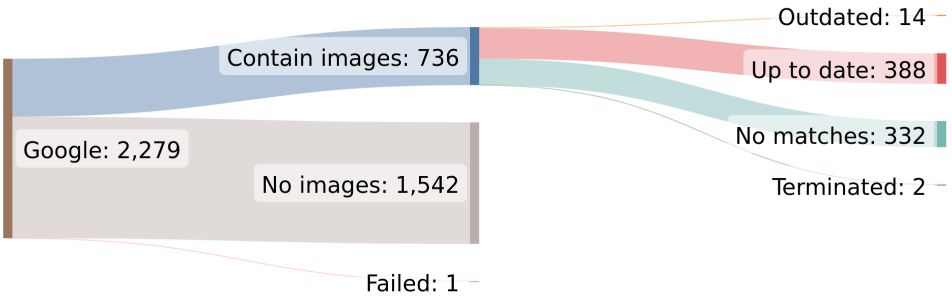


FIGURE 4.2. Analysis status of images in *google* projects, indicating whether a repository's documentation is currently out of date

We ran the analysis on all 736 projects that contained at least one image reference for a day. Due to the time constraint, two projects[4][5] did not finish the analysis on time. 402 projects contained at least one reference to code element and 332 projects did not contain any references. Altogether, the 402 projects contained 435 documents with 3186 matched references to the source code. We found that 1.9% (14/736) projects, 3.4% (15/435) documents and 1.1% (35/3186) code element references extracted from images were outdated at the time of analysis. With the projects' commit histories, we calculated that the code element references were outdated for a median duration of 2.2 years.

RQ1 Summary 1.9% (14/736) projects, 3.4% (15/435) documents and 1.1% (35/3186) code element references extracted from images belonging to 736 *google* projects were found to be outdated for a median duration of 2.2 years.

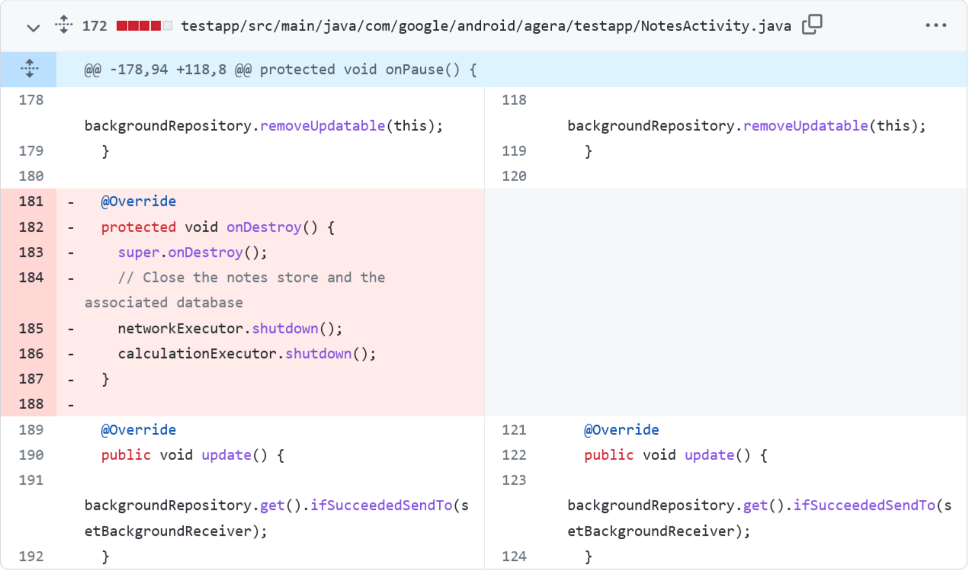
[3] https://github.com/google/material-design-icons

[4] https://github.com/google/wikiloop-doublecheck

[5] https://github.com/google/wmt-mqm-human-evaluation

4.4 Discussion

The google/agera[6] project is one of the 14 projects that was detected by our approach that contains outdated references in an image. One of the wiki pages[7] in the project contained the example image used to compare the text extraction between the various OCR services (Figure 4.1). Using OCR, we extracted the code element reference `onDestroy` from the image. When the documentation was last updated, the source code contained the code element `onDestroy` twice.[8] However when the source code instances were deleted in this commit[9] (Figure 4.3), the image was not updated to remove the code element reference. This could potentially be confusing for users of the project as the image now references a function that has been deleted from the source code. Using our approach, we were able to identify that the image contains an outdated code element reference and the image should be updated to avoid confusion.



```
172 testapp/src/main/java/com/google/android/agera/testapp/NotesActivity.java

@@ -178,94 +118,8 @@ protected void onPause() {

backgroundRepository.removeUpdatable(this);          backgroundRepository.removeUpdatable(this);
  }                                                    }

-  @Override
-  protected void onDestroy() {
-    super.onDestroy();
-    // Close the notes store and the
associated database
-    networkExecutor.shutdown();
-    calculationExecutor.shutdown();
-  }
-
   @Override                                           @Override
   public void update() {                              public void update() {

backgroundRepository.get().ifSucceededSendTo(s       backgroundRepository.get().ifSucceededSendTo(s
etBackgroundReceiver);                               etBackgroundReceiver);
  }                                                    }
```

FIGURE 4.3. Code element `onDestroy` deleted from the source code

Although 2119 images were considered in the analysis, only 35 out of 3186 code element references were found to be outdated. This low number may be attributed to a few factors. Firstly, many images in the README file included repository badges which often only contain a word. Secondly, images often contain English words instead of code elements which are less likely to be matched to the source code. Text incorrectly extracted by the OCR also contribute to the low number of matched code element references as the approach look for an exact match in the source code. The next chapter will discuss the threats to validity of our approach.

[6] https://github.com/google/agera/

[7] https://github.com/google/agera/wiki/Observables-and-updatables/ca03cd96fa90986fbe555c91d5fd426175fa3793

[8] https://github.com/google/agera/blob/3570c4167388fcd7b70bfb25e098b96cefca6db7/testapp/src/main/java/com/google/android/agera/testapp/NotesActivity.java#L182

[9] https://github.com/google/agera/commit/4711c2fff23254389b8486cb81c60dfb918f6d2c#diff-bf04f1c6a2813c126d1d793240e2ada18a5ab20ae297815f47625a9681fb0a0bL182

Chapter 5

Threats to Validity

5.1 Construct validity

In this work, our approach has identified many documents that are potentially outdated in software repositories but it does not detect all kinds of outdated documentation. As our approach relies on regular expressions for text extraction and matching, other forms of documentation containing outdated information such as videos cannot be easily detected. Even though regular expressions allows us to easily extract code element references, they may sometimes lead to references being falsely categorised as outdated, e.g. deleting the final instance of a code element that is part of a source code comment.

A project's change log may occasionally be incorrectly flagged as outdated as it may contain references to code elements that are no longer in the source code. However, these references should not be considered outdated as they only serve as a notice for users that the referenced class or function has been deprecated. In addition, our approach also cannot detect outdated relationships between the repository and documentation if the code elements are still present in the source code, i.e. documentation could be considered outdated even if all code element references are matched. These false positives are difficult to eliminate and require project maintainers to verify individually.

5.2 Internal validity

The manual annotation conducted in Section 2.1.2 to improve the quality of the code element references extracted by regular expressions may introduce bias. To minimise bias when determining if a reference was outdated, the annotation process was done separately by three annotators. We also ensured that our inter-rater agreement was high so that the annotations were reliable.

5.3 External validity

While the findings are based on the analysis of over 3,000 projects, we cannot claim that the findings can be generalised to other GitHub repositories that are not in the datasets considered, i.e. the top 1,000 most popular GitHub repositories and those owned by Google. We also cannot make claims of the generalisability of our findings for projects hosted on other version control platforms.

Chapter 6

Conclusions and Future Work

In this thesis, we proposed an approach that can automatically detect outdated references to code elements caused by removing all source code instances. We investigated the current state of documentation in software repositories, extended the approach to analyse the state of documentation throughout projects' history, explored how outdated documentation is resolved in open source projects, and with the results, we alerted Google developers of potentially outdated code element references in their projects. In addition, we created a publicly available tool that enables developers to scan for outdated references and used OCR to detect images containing outdated references in software documentation.

In detail, we found that the majority of the most popular projects on GitHub contained at least one outdated reference to a code element at some point during their history and these outdated references usually survived in the documentation for years before they were fixed. By analysing the full history of projects, we discovered that outdated references are more likely fixed by updating the source code or document than deleting the entire document. Moreover, our GitHub issues have led to instances of outdated documentation getting fixed in real-world projects. To assist developers with discovering outdated documentation in their projects, we created a GitHub Action tool that can automatically scan for outdated code element references whenever a pull request is submitted to a repository. Finally, we were able to use OCR to extract text from images to detect outdated references in real-world projects.

One of the potential directions for future work is to investigate how traceability links can be established for renamed code elements. Our approach currently only detects if a code element reference is outdated by performing an exact match to find source code instances, but it does not know what the code element has been renamed to. Being able to automatically establish links means that the tool can automatically suggest how to update the documentation. Other future work may come in the form of small improvements to the current approach. Currently, deleting the final code element instance that is part of a source code comment may lead to falsely flagged references. Applying customised sets of regular expressions for files written in different programming languages may be one such improvement to help with more accurate matches in the source code, e.g. avoiding matching code elements that are part of a source code comment. Another small useful extension to the tool could be a feature where the project developer can reply to the tool's comment for code elements they do not want to keep track of. The tool will then automatically add the code elements to the project's exclude list. Finally, more investigations on outdated references in images, e.g. extending the analysis to the *top1000* dataset, will help the software engineering community better understand the state of images in documentation.

We hope that this research will be a step toward keeping documentation in software repositories up-to-date.

Bibliography

Aghajani, Emad, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza (2019). "Software documentation issues unveiled". In: *Proceedings of the International Conference on Software Engineering*, pp. 1199–1210.

Aldaeej, Abdullah (2021). "Towards Effective Technical Debt Decision Making in Software Startups: A Multiple Case Study of Web and Mobile App Startups". PhD thesis. University of Maryland, Baltimore County.

Bacchelli, Alberto, Michele Lanza, and Romain Robbes (2010). "Linking e-mails and source code artifacts". In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pp. 375–384.

Dagenais, Barthélémy and Martin P Robillard (2012). "Recovering traceability links between an API and its learning resources". In: *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, pp. 47–57.

Dagenais, Barthélémy and Martin P Robillard (2014). "Using traceability links to recommend adaptive changes for documentation evolution". In: *IEEE Transactions on Software Engineering* 40.11, pp. 1126–1146.

Forward, Andrew and Timothy C Lethbridge (2002). "The relevance of software documentation, tools and technologies: a survey". In: *Proceedings of the Symposium on Document Engineering*, pp. 26–33.

Kajko-Mattsson, Mira (2005). "A survey of documentation practice within corrective maintenance". In: *Empirical Software Engineering* 10.1, pp. 31–55.

Kruchten, Philippe, Robert L Nord, and Ipek Ozkaya (2012). "Technical debt: From metaphor to theory and practice". In: *IEEE Software* 29.6, pp. 18–21.

Lee, Seonah, Rongxin Wu, Shing-Chi Cheung, and Sungwon Kang (2019). "Automatic detection and update suggestion for outdated API names in documentation". In: *IEEE Transactions on Software Engineering*.

Lethbridge, Timothy C, Janice Singer, and Andrew Forward (2003). "How software engineers use documentation: The state of the practice". In: *IEEE Software* 20.6, pp. 35–39.

Liu, Jiakun, Qiao Huang, Xin Xia, Emad Shihab, David Lo, and Shanping Li (2021). "An exploratory study on the introduction and removal of different types of technical debt in deep learning frameworks". In: *Empirical Software Engineering* 26.2, pp. 1–36.

Mendes, Thiago Souto, Mário André de F. Farias, Manoel Mendonça, Henrique Frota Soares, Marcos Kalinowski, and Rodrigo Oliveira Spínola (2016). "Impacts of agile requirements documentation debt on software projects: a retrospective study". In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pp. 1290–1295.

Panthaplackel, Sheena, Pengyu Nie, Milos Gligoric, Junyi Jessy Li, and Raymond J Mooney (2020). "Learning to Update Natural Language Comments Based on Code Changes". In: *arXiv preprint arXiv:2004.12169*.

Parnas, David Lorge (1994). "Software aging". In: *Proceedings of International Conference on Software Engineering*, pp. 279–287.

Petrosyan, Gayane, Martin P Robillard, and Renato De Mori (2015). "Discovering information explaining API types using text classification". In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. IEEE, pp. 869–879.

Prana, Gede Artha Azriadi, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo (2019). "Categorizing the content of github readme files". In: *Empirical Software Engineering* 24.3, pp. 1296–1327.

Ratol, Inderjot Kaur and Martin P Robillard (2017). "Detecting fragile comments". In: *Proceedings of the International Conference on Automated Software Engineering*, pp. 112–122.

Rigby, Peter C and Martin P Robillard (2013). "Discovering essential code elements in informal documentation". In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, pp. 832–841.

Rios, Nicolli, Leonardo Mendes, Cristina Cerdeiral, Ana Patrícia F Magalhães, Boris Perez, Darío Correal, Hernán Astudillo, Carolyn Seaman, Clemente Izurieta, Gleison Santos, et al. (2020). "Hearing the voice of software practitioners on causes, effects, and practices to deal with documentation debt". In: *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, pp. 55–70.

Robillard, Martin P, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro, Neil Ernst, Marco Aurélio Gerosa, Michael Godfrey, Michele Lanza, Mario Linares-Vásquez, et al. (2017). "On-demand developer documentation". In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, pp. 479–483.

Sholler, Dan, Igor Steinmacher, Denae Ford, Mara Averick, Mike Hoye, and Greg Wilson (2019). "Ten simple rules for helping newcomers become contributors to open projects". In: *PLoS computational biology* 15.9, e1007296.

Souza, Sergio Cozzetti B de, Nicolas Anquetil, and Káthia M de Oliveira (2005). "A study of the documentation essential to software maintenance". In: *Proceedings of the International Conference on Design of Communication: Documenting & Designing for Pervasive Information*, pp. 68–75.

Steinmacher, Igor, Christoph Treude, and Marco Aurélio Gerosa (2018). "Let me in: Guidelines for the successful onboarding of newcomers to open source projects". In: *IEEE Software* 36.4, pp. 41–49.

Tan, Shin Hwei, Darko Marinov, Lin Tan, and Gary T Leavens (2012). "@tcomment: Testing Javadoc comments to detect comment-code inconsistencies". In: *Proceedings of the International Conference on Software Testing, Verification and Validation*, pp. 260–269.

Treude, Christoph, Martin P Robillard, and Barthélémy Dagenais (2014). "Extracting development tasks to navigate software documentation". In: *IEEE Transactions on Software Engineering* 41.6, pp. 565–581.

Uddin, Gias and Martin P Robillard (2015). "How API documentation fails". In: *IEEE Software* 32.4, pp. 68–75.

Wen, Fengcai, Csaba Nagy, Gabriele Bavota, and Michele Lanza (2019). "A large-scale empirical study on code-comment inconsistencies". In: *Proceedings of the International Conference on Program Comprehension*, pp. 53–64.

Zhong, Hao and Zhendong Su (2013). "Detecting API documentation errors". In: *Proceedings of the International Conference on Object Oriented Programming Systems Languages & Applications*, pp. 803–816.

Zhou, Yu, Changzhi Wang, Xin Yan, Taolue Chen, Sebastiano Panichella, and Harald C Gall (2020). “Automatic detection and repair recommendation of directive defects in Java API documentation”. In: *IEEE Transactions on Software Engineering* 46.9, pp. 1004–1023.

Zlotnick, Frances (June 2017). *GitHub Open Source Survey 2017*. DOI: `10.5281/zenodo.806811`. URL: `https://doi.org/10.5281/zenodo.806811`.