Empirical Software Engineering (2024) 29:5
https://doi.org/10.1007/s10664-023-10397-6

Detecting outdated code element references in software repository documentation

Wen Siang Tan[1] · **Markus Wagner**[2] · **Christoph Treude**[3]

Accepted: 25 September 2023 / Published online: 21 November 2023

Abstract
Outdated documentation is a pervasive problem in software development, preventing effective use of software, and misleading users and developers alike.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 In this work, we analysed over 3,000 GitHub projects and found that most projects contain at least one outdated code element reference at some point in their history. We submitted GitHub issues to real-world projects containing outdated references detected by our approach, some of which have already led to documentation fixes.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repositories up-to-date, we have made our implementation available for developers to scan their GitHub projects for outdated code element references.

Keywords Software repositories · Outdated documentation · Outdated references · Code elements

Communicated by: Romain Robbes

✉ Wen Siang Tan
wensiang.tan@adelaide.edu.au

Markus Wagner
markus.wagner@monash.edu

Christoph Treude
christoph.treude@unimelb.edu.au

1 University of Adelaide, Adelaide, Australia

2 Monash University, Melbourne, Australia

3 University of Melbourne, Melbourne, Australia

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Outdated documentation is a common and well-known problem in software development (Lee et al. 2019).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 et al. 2018). In a recent study on software documentation issues, (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 (de Souza et al. 2005; Lethbridge et al. 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.

As a step toward helping developers to keep their documentation up-to-date, we propose an automated approach that detects outdated references in README file and wiki pages of a GitHub project. 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 noticed and fixed by project maintainers.

The remainder of the paper is structured as follows: We motivate our work through a real-world example of outdated documentation in Section 2, explain our approach in Section 3, and introduce the research questions in Section 4. We report our findings in Section 5, present our publicly available implementation in Section 6, and interpret our findings in Section 7. We discuss the threats to validity of our approach in Section 8 before we conclude the paper with related and future work in Sections 9 and 10.

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] However, the documentation[5] was not updated to reflect the changes. In the same project, another code element `fPIC` was found 21

[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.

[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

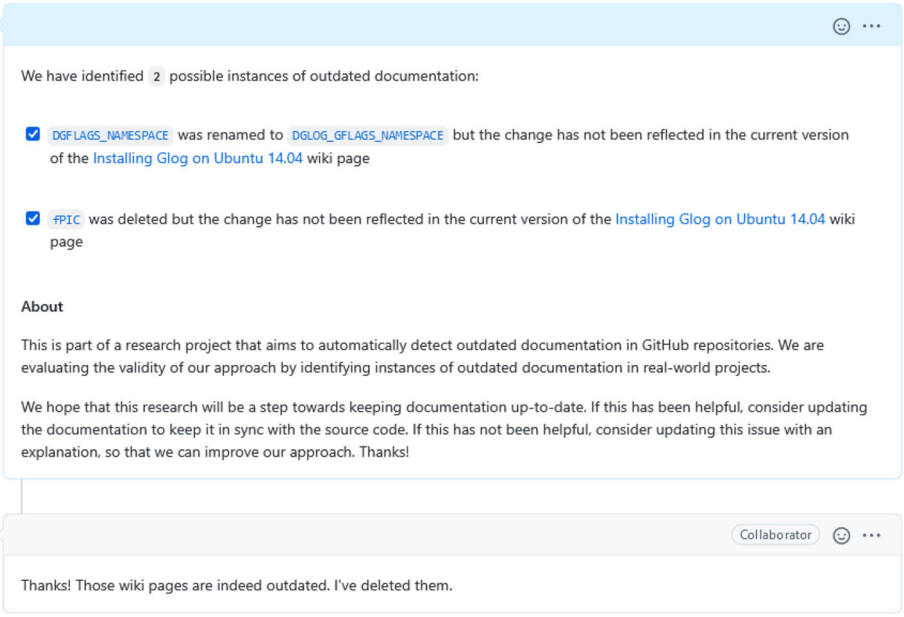

Fig. 1 Screenshot of the GitHub issue submitted

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 (Fig. 1). Following our report, the project maintainer fixed the outdated documentation by deleting the document containing the two outdated references.

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 section 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.

3 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

[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

source code, we consider the documentation outdated. The rest of this section describes this process in detail.

3.1 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. (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 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.[9]

We consider two datasets in this paper. The first dataset consists of the 1,000 most popular projects on GitHub, ranked by the number of stars.[10] The second dataset consists of all 2,279 GitHub projects from Google.[11] Figures 2, 3, 4 to 5 show some key statistics of the projects. The ***top1000*** projects are generally larger in size compared to ***google*** projects (median of 31.7MiB compared to 1.5MiB), existed longer (median of 7.6 years compared to 4.4 years), and have more developers contributing (median of 181 compared to 4).[12] Meanwhile, programming languages such as JavaScript, Python, Java and Go are popular choices for both ***top1000*** and ***google*** projects. This roughly matches the top programming languages used on GitHub in 2022.[13] The list of project names for both datasets can be found in our online appendix.[14]

3.2 Extracting Code Elements

In Section 3.1, 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. (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.[15] For the scope of this paper, we define code elements as syntactic compo-

[9] https://github.com/github/markup

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

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

[12] Number of contributors collected at a later date on 5 May 2023 for paper revision

[13] https://octoverse.github.com/2022/top-programming-languages

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

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

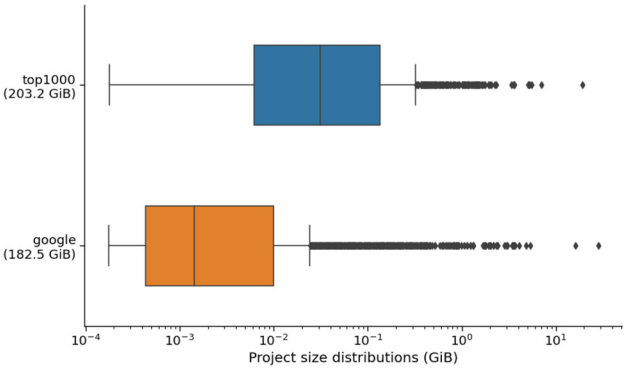

Fig. 2 Project size distributions (GiB) for ***top1000*** and ***google*** projects in log scale

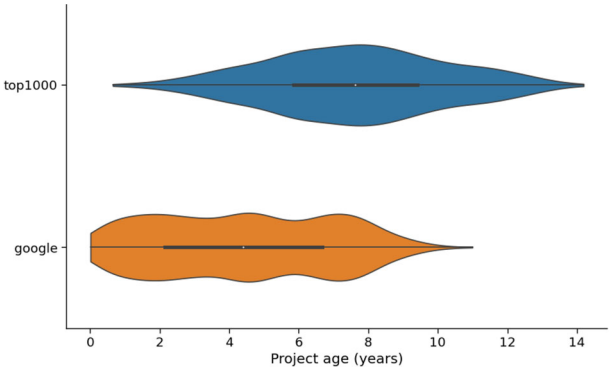

Fig. 3 Project age in years for ***top1000*** and ***google*** projects

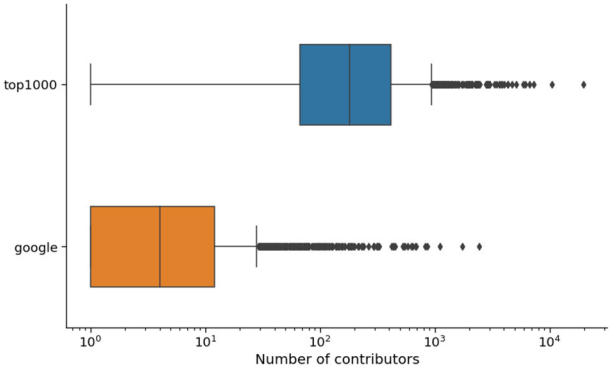

Fig. 4 Number of contributors for ***top1000*** and ***google*** projects in log scale

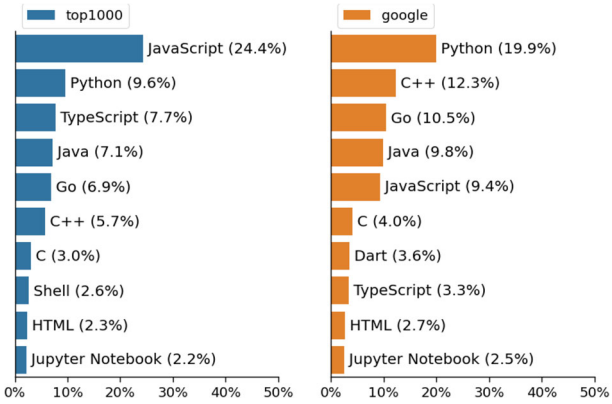

Fig. 5 Top 10 programming languages used in ***top1000*** and ***google*** projects

nents of programming languages and URLs matched by regular expressions as well as other tokens such as file names present in the repository.

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 annotated the same 50 randomly selected code elements detected from the ***google*** projects to measure the inter-rater agreement. We achieved an 'almost-perfect' (McHugh 2012) 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.
   (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

Table 1 List of updated regular expressions and examples of whole word, case-sensitive, and exact string matched text

Regular expression	Example matched text
`(?<=(?<!`)`)[!-_a-~]+(?=`)`	`` `abcdef` ``
`[A-Z][a-zA-Z]+ ?<[A-Z][a-zA-Z]*>`	`ArrayList<String>`
`[a-zA-Z0-9\.]+[(][a-zA-Z_,\.]*[)]`	`Promise.reject(err)`
`[a-zA-Z]+\.[A-Z]+`	`Pattern.MULTILINE`
`[@][a-zA-Z]+`	`@types`
`([a-zA-Z]{3,}\.[A-Za-z]+_[a-zA-Z_]+)`	`request.iter_idx`
`([A-Z]{2,})`	`JSON`
`([A-Z]+_[A-Z0-9_]+)`	`GIT_DIR`
`([a-z]+_[a-z0-9_]+)`	`node_modules`
`\w{3,}:\w+[a-zA-Z0-9:]*`	`sdk:stable`
`([A-Z]+[a-z0-9]+[A-Z][a-z0-9]+\w*)`	`KeyboardEvent`
`([A-Z]{3,}[a-z0-9]{2,}\w*)`	`IOException`
`([a-z0-9]+[A-Z]+\w*)`	`querySelector`
`(\w+\([^)]*\))`	`createElement('div')`
`([A-Z][a-z]+[A-Z][a-zA-Z]+)`	`HttpClient`
`([a-z]+[A-Z][a-zA-Z]+)`	`addEventListener`
`\{\{[^\}]*\}\}`	`{{ end }}`
`\{\%[^\%]*\%\}`	`{% endif %}`
`'[^']*'`	`'unknown'`
`__[^_]*__`	`__init__`
`\$[A-Za-z\_]+`	`$working_dir`

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 paper (Table 1) can also be found in our online appendix.[16]

3.3 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

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

Table 2 What is outdated?

	Before	After
Documentation	✓	✓
Source code	✓	✗

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).

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 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 3).

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 (Fig. 6). 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.

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

Table 3 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

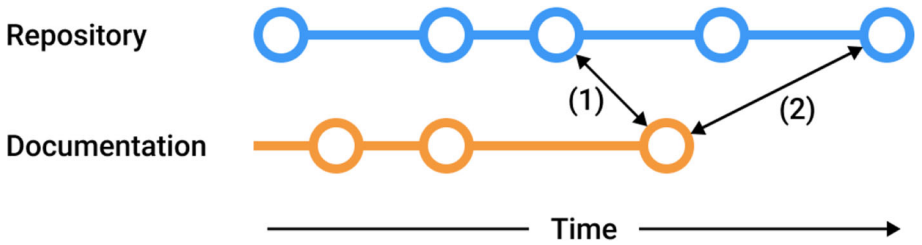

Fig. 6 Linking the current documentation version to (1) repository snapshot and (2) current repository revision

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

3.4 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 N instances of C.

Table 4 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

Table 5 Example of symbolic representation

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

The symbolic representation can be summarised in Table 4. As an example, the first 50 revisions of the code element `renderFiles('./files')` in the README file from the vuejs/vue-cli project[17] have the following symbolic representation (Table 5):

- 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 3.3, the current version of the documentation is linked to the same repository revisions. Figure 7 shows the links between repository revisions and their corresponding documentation versions.

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 3.3 (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 6 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.

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

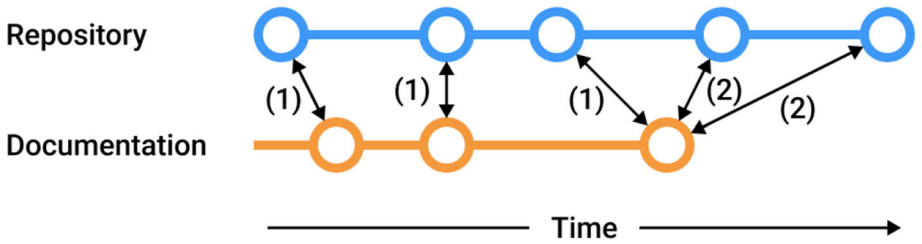

Fig. 7 Linking each repository revision to a corresponding documentation version for repository commits made (1) before and (2) after the current documentation version

4 Research Questions

RQ1 What is the current state of documentation?

RQ1.1 In terms of references to code elements, how many projects/documents are currently outdated?

RQ1.2 In terms of references to code elements, how long have these projects/documents been outdated?

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 projects/documents/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?

RQ2.1 In terms of references to code elements, how many projects/documents were outdated at some point in their history?

RQ2.2 In terms of references to code elements, how long have these projects/documents been outdated?

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 projects/documents/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

Table 6 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

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.

5 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 3.1. When cloning the repositories, one project[18] 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[19] 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 8 shows the breakdown of the projects' statuses.

5.1 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 3 and counted the number of projects for which the documentation contained at least one outdated code element reference (see Fig. 8). 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 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 (Fig. 9). 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 (Fig. 10).

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

[19] https://github.com/google/swiftshader

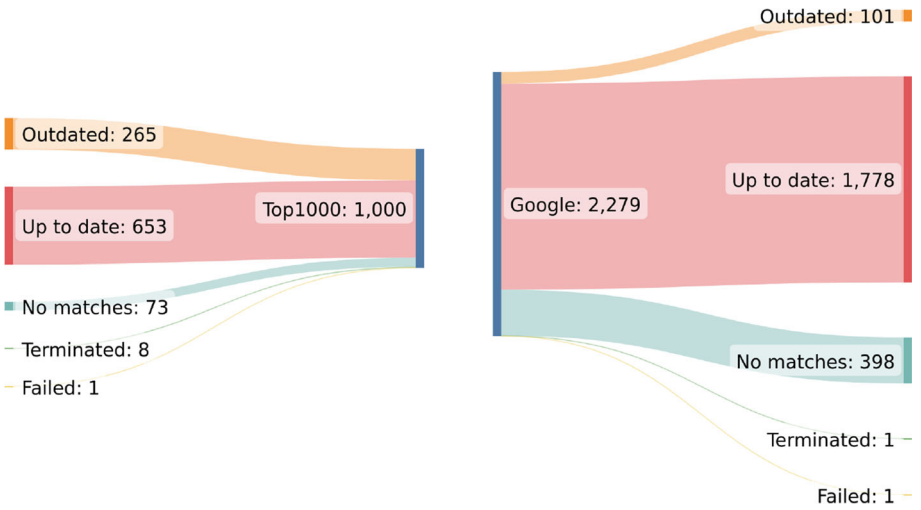

Fig. 8 Analysis status of ***top1000*** and ***google*** projects, indicating whether a repository's documentation is currently out of date

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.

5.2 RQ2: What was the State of Documentation During the Projects' History?

To study how documentation evolves, we analysed all commits made to the main branch as well as merge commits on the main branch containing changes from feature branches for 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

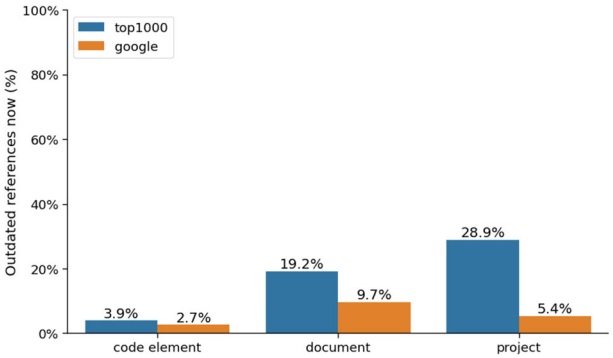

Fig. 9 Percentage of references outdated at the time of analysis on code element, document and project levels

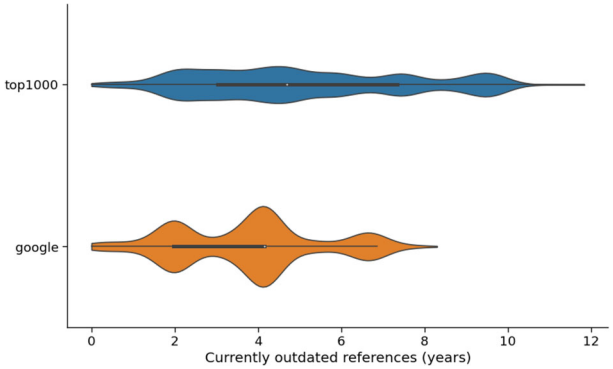

Fig. 10 Distribution of duration that code element references have been outdated at the time of analysis in ***top1000*** and ***google*** projects

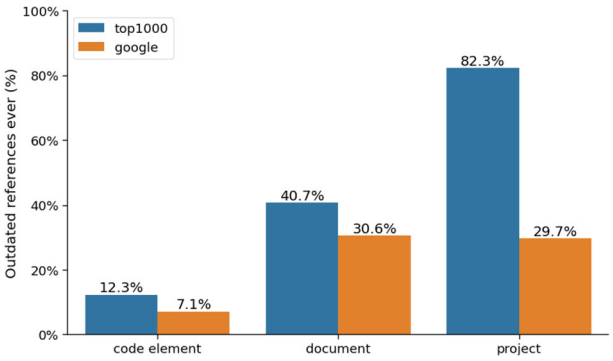

Fig. 11 Percentage of references outdated at least once at some point during its history on code element, document and project levels

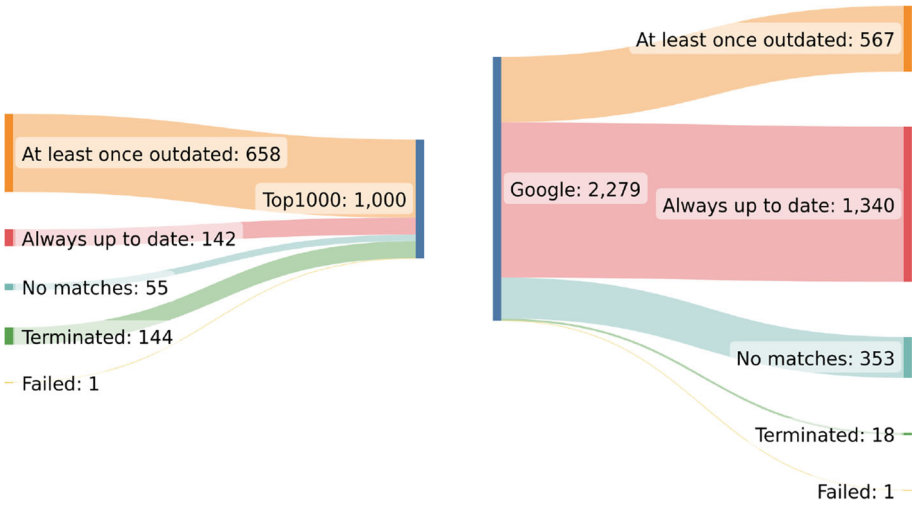

Fig. 12 Extended analysis status of ***top1000*** and ***google*** projects, indicating whether a repository's documentation was outdated at some point during its history

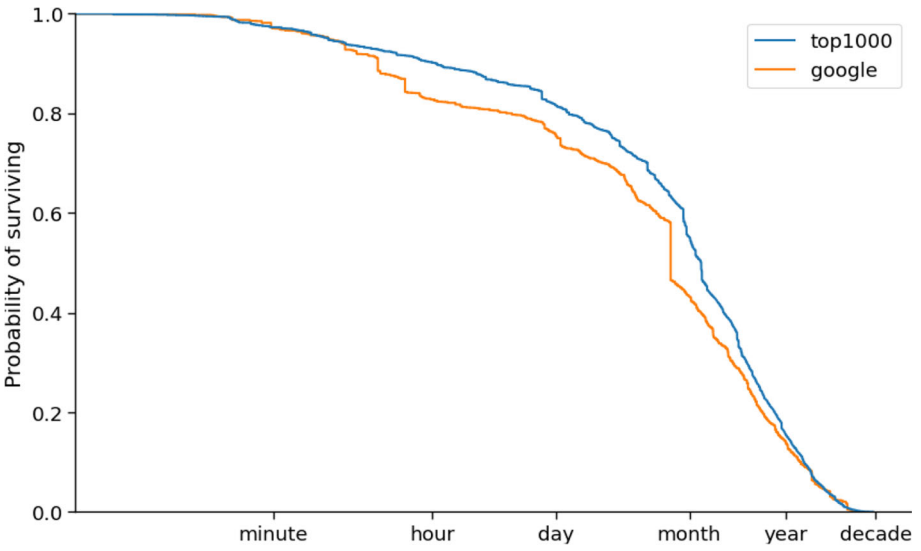

Fig. 13 Probability of outdated references surviving in the documentation of ***top1000*** and ***google*** projects after the indicated duration (out of all references fixed by project maintainers at the time of analysis), e.g. after one month, 45% (***google***) and 55% (***top1000***) of the references (that were eventually fixed) were still outdated

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 (Fig. 11). 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 as more projects exceeded the time limit when analysing all commits in the main branch (Fig. 12).

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 13 contains only outdated code elements references that project maintainers have already fixed with a timestamp difference greater than zero.[20] The probability of surviving is calculated by the percentage of outdated code element references that survived in the documentation after the duration indicated by the x-axis has passed. For example, after one month, 45% (***google***) and 55% (***top1000***) of the references (that were eventually fixed) were still outdated.

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.

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

Table 7 Types of documentation fixes

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

5.3 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, 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 3.4 as shown in Table 7.

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. Figure 14 shows the distribution of the differences between the number of instances found between each commit for ***top1000*** and ***google*** projects, with 0.29% of ***top1000*** and 0.51% of ***google*** commits having a difference of more than 1000.

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.

5.4 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 3.2, we filtered 35 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

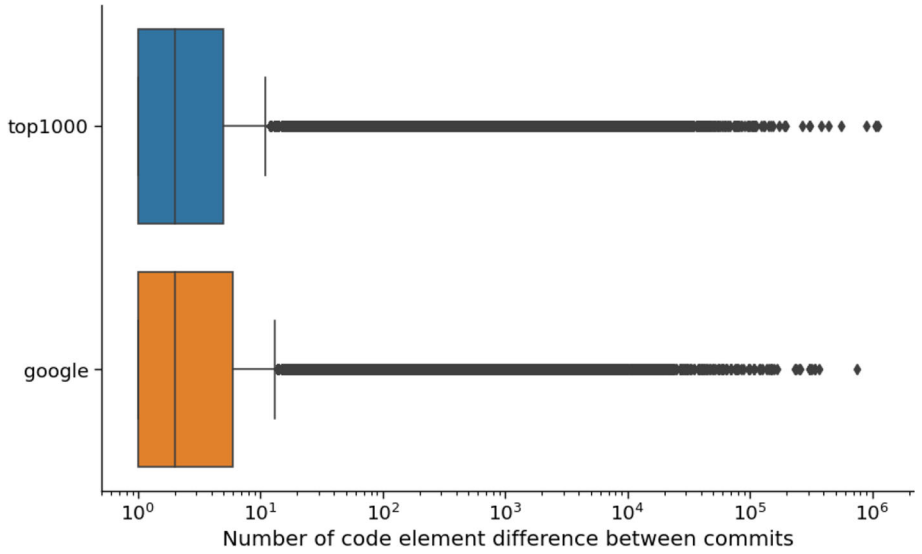

Fig. 14 The differences between the number of instances found in consecutive commits for ***top1000*** and ***google*** projects. 0.29% of ***top1000*** and 0.51% of ***google*** commits have a difference of more than 1000

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.[21] 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 (Fig. 15). In the hs-portray project, the function **`prettyShow`** was renamed to **`showPortrayal`** in the source code, but the README file was not updated (Fig. 16). We alerted the developers of this discrepancy, and the issue was fixed subsequently.[22]

False positives In one of the projects (Fig. 17), 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.[23] A false positive was reported in another project (Fig. 18) 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.[24]

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

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

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

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

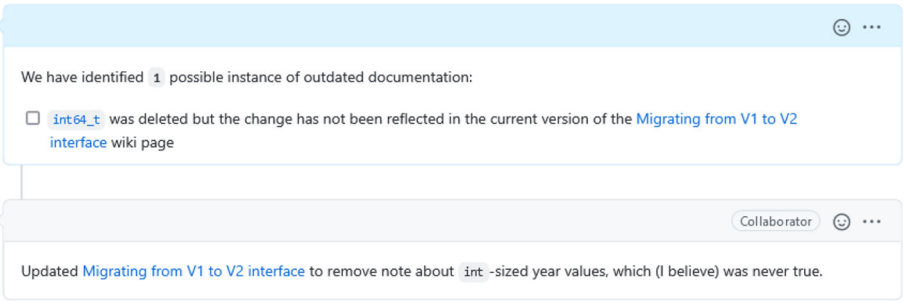

Fig. 15 True positive: data type updated in the documentation

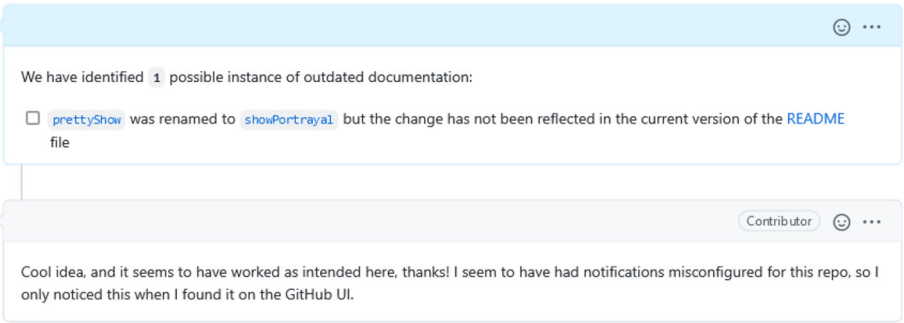

Fig. 16 True positive: function name updated in the documentation

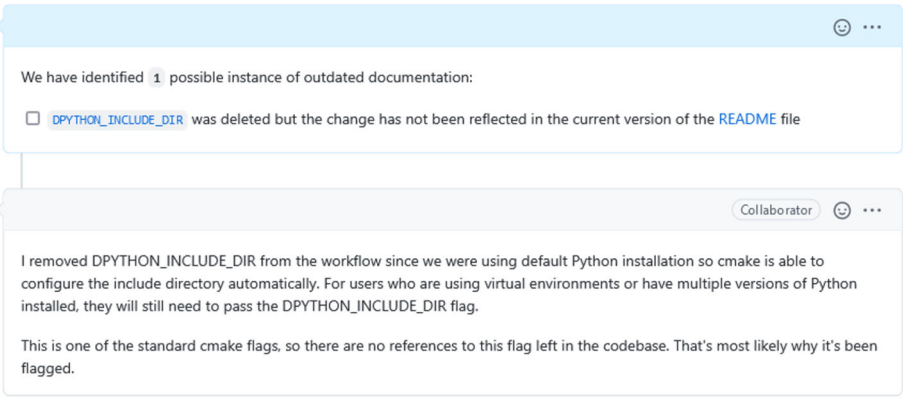

Fig. 17 False positive: still relevant for users with multiple Python versions

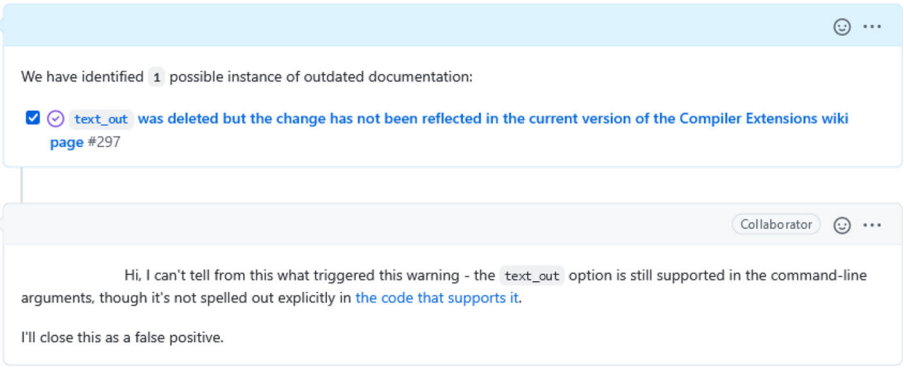

Fig. 18 False positive: functionality remains in the program logic

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

6 Implementation

The implementation of our approach called DOCER (Detecting Outdated Code Element References) is available in our online appendix.[25] To get started, specify the project name in the new_projects.txt file and run clone_projects.sh to clone the required project files. Running the normal_analysis.sh or extended_analysis.sh file extracts code element references from the documentation and reports the number of code element instances found in the source code. Additionally, a report can be generated in the format of Table 8 using normal_report.py or extended_report.py depending on the type of analysis. 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.

7 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 posit 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.

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

Table 8 Format of the report generated by the implementation

Column name	Description
code_element	the code element captured by the regular expression
page_type	type of the documentation page (repo or wiki)
page_name	file name of the documentation page
rev_N	number of code element instances in revision N
rev_SHA_N	SHA of revision N
rev_timestamp_N	commit timestamp of revision N
doc_SHA	SHA of the documentation
doc_timestamp	commit timestamp of the documentation
doc_link	link to the documentation
source_link	link to the first source code instance

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 (Fig. 13) and discovered that the biggest drop around the one month mark was caused by project maintainers deleting[26] and restoring[27] large amounts 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 paper is centred around detecting outdated code element references in documentation hosted on GitHub projects, our approach can be generalised to other version control platforms. The next section of the paper will discuss the potential threats to validity of our approach.

8 Threats to Validity

8.1 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

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

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

containing outdated information such as images or videos cannot be detected. Even though regular expressions allow 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.

Our approach focuses on the main branch, not analysing outdated code element references across parallel or feature branches. The rationale for this design choice is GitHub's default behaviour, which displays only the README file from the main branch. Moreover, in branches with interleaved commits, code elements exclusive to a particular branch can appear and disappear intermittently, causing our method to mistakenly identify them as outdated. While adapting our approach to encompass all branches, yielding a comprehensive list of outdated elements for each, would be straightforward, a significant challenge would be the design of a user interface which effectively communicates this information in scenarios with many parallel branches. This is especially important for situations where developers are willing to accept temporary outdatedness during ongoing modifications.

8.2 Internal Validity

The manual annotation conducted in Section 3.2 to assess 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. Using more than 50 randomly selected code elements for the manual annotation process might have resulted in different modifications to the regular expression list and the number of repositories that fit the criteria for RQ4.

8.3 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.

9 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 and/or fixing 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.

9.1 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."

Outdated software documentation is a form of technical debt (Kruchten et al. 2012) often referred to as documentation debt (Aldaeej 2021). 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. (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. (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.

9.2 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 et al. 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 et al. 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. (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.

9.3 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. (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. (2012) presented @tcomment for determining the correctness of Javadoc comments related to null values and exceptions. DocRef by 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 (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 (2017) presented Fraco, a tool to detect source code comments that are fragile with respect to identifier renaming.

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. (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. (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 work, 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.

10 Conclusion/Future Work

In this paper, 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 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.

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. Using our current implementation, developers have to manually scan their repository with each commit which may be repetitive and time consuming. A possible direction for future work is to create a workflow that automatically clones the repository, runs the analysis, and outputs the potentially outdated references. Using

a tool such as GitHub Action[28] to automate the workflow simplifies the process considerably as it allows developers to configure their repository to automatically scan for outdated references whenever there is a new commit or pull request.

Another potential direction for future work is expanding our approach to other forms of documentation, such as images. We imagine that texts in images are more likely to be outdated as they generally require more effort to update. These texts may be extracted using methods such as Optical Character Recognition, allowing us to detect more potentially outdated references. Additionally, the approach can be extended to other documentation files other than README.md files and wiki pages. Future work could also build on the current approach to handle multiple parallel branches simultaneously. This will allow project maintainers to quickly scan for outdated documentation without running the analysis separately on the branches. Applying customised sets of regular expressions for files written in different programming languages may also be another direction to help with more accurate matches in the source code, e.g. avoiding matching source code comments. We hope that this research will be a step toward keeping documentation in software repositories up-to-date.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions.

Data availibility statement The datasets generated during and/or analysed during the current study are available in the Zenodo repository, https://zenodo.org/record/7384588.

Declarations

Conflicts of interest Christoph Treude is a member of the Empirical Software Engineering Editorial Board. The authors have no other conflict of interest to declare.

References

Aghajani E, Nagy C, Vega-Márquez OL, Linares-Vásquez M, Moreno L, Bavota G, Lanza M (2019) Software documentation issues unveiled. In: Proceedings of the International Conference on Software Engineering, pp 1199–1210

Aldaeej A (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 A, Lanza M, Robbes R (2010) Linking e-mails and source code artifacts. In: Proceedings of the 32nd ACM/IEEE International conference on software engineering vol 1, pp 375–384

Dagenais B, Robillard MP (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 B, Robillard MP (2014) Using traceability links to recommend adaptive changes for documentation evolution. IEEE Trans Software Eng 40(11):1126–1146

Forward A, Lethbridge TC (2002) The relevance of software documentation, tools and technologies: a survey. In: Proceedings of the symposium on document engineering, pp 26–33

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

Kajko-Mattsson M (2005) A survey of documentation practice within corrective maintenance. Empir Softw Eng 10(1):31–55
Kruchten P, Nord RL, Ozkaya I (2012) Technical debt: From metaphor to theory and practice. IEEE Softw 29(6):18–21
Lee S, Wu R, Cheung SC, Kang S (2019) Automatic detection and update suggestion for outdated API names in documentation. IEEE Transactions on Software Engineering
Lethbridge TC, Singer J, Forward A (2003) How software engineers use documentation: The state of the practice. IEEE Softw 20(6):35–39
Liu J, Huang Q, Xia X, Shihab E, Lo D, Li S (2021) An exploratory study on the introduction and removal of different types of technical debt in deep learning frameworks. Empir Softw Eng 26(2):1–36
McHugh ML (2012) Interrater reliability: the kappa statistic. Biochem Med (Zagreb) 22(3):276–282
Mendes TS, de F Farias MA, Mendonça M, Soares HF, Kalinowski M, Spínola RO (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 S, Nie P, Gligoric M, Li JJ, Mooney RJ (2020) Learning to update natural language comments based on code changes. arXiv preprint arXiv:2004.12169
Parnas DL (1994) Software aging. In: Proceedings of international conference on software engineering, pp 279–287
Petrosyan G, Robillard MP, De Mori R (2015) Discovering information explaining api types using text classification. In: 2015 IEEE/ACM 37th IEEE international conference on software engineering, IEEE, vol 1, pp 869–879
Prana GAA, Treude C, Thung F, Atapattu T, Lo D (2019) Categorizing the content of github readme files. Empir Softw Eng 24(3):1296–1327
Ratol IK, Robillard MP (2017) Detecting fragile comments. In: Proceedings of the international conference on automated software engineering, pp 112–122
Rigby PC, Robillard MP (2013) Discovering essential code elements in informal documentation. In: 2013 35th international conference on software engineering (ICSE), IEEE, pp 832–841
Rios N, Mendes L, Cerdeiral C, Magalhães APF, Perez B, Correal D, Astudillo H, Seaman C, Izurieta C, Santos G, 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 MP, Marcus A, Treude C, Bavota G, Chaparro O, Ernst N, Gerosa MA, Godfrey M, Lanza M, Linares-Vásquez M, et al. (2017) On-demand developer documentation. In: 2017 IEEE International conference on software maintenance and evolution (ICSME), IEEE, pp 479–483
Sholler D, Steinmacher I, Ford D, Averick M, Hoye M, Wilson G (2019) Ten simple rules for helping newcomers become contributors to open projects. PLoS Comput Biol 15(9):e1007296
de Souza SCB, Anquetil N, de Oliveira KM (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 I, Treude C, Gerosa MA (2018) Let me in: Guidelines for the successful onboarding of newcomers to open source projects. IEEE Software 36(4):41–49
Tan SH, Marinov D, Tan L, Leavens GT (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 C, Robillard MP, Dagenais B (2014) Extracting development tasks to navigate software documentation. IEEE Trans Software Eng 41(6):565–581
Uddin G, Robillard MP (2015) How API documentation fails. IEEE Software 32(4):68–75
Wen F, Nagy C, Bavota G, Lanza M (2019) A large-scale empirical study on code-comment inconsistencies. In: Proceedings of the international conference on program comprehension, pp 53–64
Zhong H, Su Z (2013) Detecting API documentation errors. In: Proceedings of the international conference on object oriented programming systems languages & applications, pp 803–816
Zhou Y, Wang C, Yan X, Chen T, Panichella S, Gall HC (2020) Automatic detection and repair recommendation of directive defects in Java API documentation. IEEE Trans Software Eng 46(9):1004–1023
Zlotnick F (2017) Github open source survey 2017. https://doi.org/10.5281/zenodo.806811