Noname manuscript No.
(will be inserted by the editor)

18 Million Links in Commit Messages: Purpose, Evolution, and Decay

Tao Xiao ✉ · Sebastian Baltes · Hideaki Hata · Christoph Treude · Raula Gaikovina Kula · Takashi Ishio · Kenichi Matsumoto

Received: date / Accepted: date

Abstract Commit messages contain diverse and valuable types of knowledge in all aspects of software maintenance and evolution. Links are an example of such knowledge. Previous work on "9.6 million links in source code comments" showed that links are prone to decay, become outdated, and lack bidirectional traceability. We conducted a large-scale study of 18,201,165 links from commits in 23,110 GitHub repositories to investigate whether they suffer the same fate. Results show that referencing external resources is prevalent and that the most frequent domains other than `github.com` are the external domains of `Stack Overflow` and `Google Code`. Similarly, links serve as source code context to commit messages, with inaccessible links being frequent. Although repeatedly referencing links is rare (4%), 14% of links that are prone to evolve become unavailable over time; e.g., tutorials or articles and software homepages become unavailable over time. Furthermore, we find that 70% of the distinct links suffer from decay; the domains that occur the most frequently are related to Subversion repositories. We summarize that links in commits share the same fate as links in code, opening up avenues for future work.

✉ Corresponding author - Tao Xiao
Nara Institute of Science and Technology, Japan
E-mail: tao.xiao.ts2@is.naist.jp

Sebastian Baltes
University of Adelaide, Australia
E-mail: sebastian.baltes@adelaide.edu.au

Hideaki Hata
Shinshu University, Japan
E-mail: hata@shinshu-u.ac.jp

Christoph Treude
University of Melbourne, Australia
E-mail: christoph.treude@unimelb.edu.au

Raula Gaikovina Kula, Takashi Ishio, Kenichi Matsumoto
Nara Institute of Science and Technology, Japan
E-mail: {raula-k,ishio,matumoto}@is.naist.jp

Keywords Commit Messages · Software Documentation · Link Sharing · Link Decay

1 Introduction

Developers use commit messages as a means to summarize introduced code changes in natural language (Mockus and Votta, 2000; Buse and Weimer, 2010). These descriptions can be used to validate changes, locate and triage bug reports, and trace changes to address software maintenance tasks (Girba et al., 2005; Hassan, 2008; D'Ambros et al., 2010). It is also known that commit messages usually contain diverse types of useful knowledge in terms of how they facilitate the understanding of code changes, e.g., issue, feature, and rationale (Mockus and Votta, 2000; Fu et al., 2015; Sarwar et al., 2020).

Inspired by the Internet, useful information or knowledge has been represented by links. However, the growth of links has brought on more challenges of link decay (Kehoe et al., 1998), digital plagiarism (Barrie and Presti, 2000), and fragile historical web content (Murphy et al., 2007). Recently, Hata et al. (2019) conducted a study to understand the purposes, evolution, and decay of links in source code comments. They observed decay, insufficient versioning, and lack of bidirectional traceability. Link decay is also a common problem in other software artifacts, e.g., Stack Overflow posts (Liu et al., 2021). Liu et al. (2022b) also observed that external links are repeatedly referenced in Stack Overflow posts, and they found that repeated external links increase the maintenance effort. As an important communication channel, commit messages become critical for communicating effectively. Among the knowledge embedded in commit messages, links are special containers that provide additional knowledge for developers in commit messages. Commit messages may play a similar role as source code comments, which communicate information indirectly between code authors and reviewers. Thus, these issues in source code comments may also apply to commit messages. Figure 1 shows the motivating example of how the link decay in the commit messages causes knowledge loss in the code review process. Such knowledge loss will not only increase review time but also make confusion for future developers.

This paper is an extension of Hata et al. (2019), where we formulate six research questions to establish the understanding of the role of links in commit messages. We collect 18,201,165 links from commits in 23,110 GitHub repositories. From our quantitative analyses, we find that links are occurring in commit messages, accounting for at least 83% of GitHub repositories in our study and the internal domain github.com is the most frequently occurring domain in commit messages, followed by stackoverflow.com. Then, we identify the kinds of link targets that are referenced and their purposes served in commit messages through qualitative studies of a stratified sample of 1,145 links. We observe that (i) inaccessible links and patch links are the most frequently occurring link target types in commit messages; and (ii) links are often used to serve as source code context or to keep files in sync between versions. To

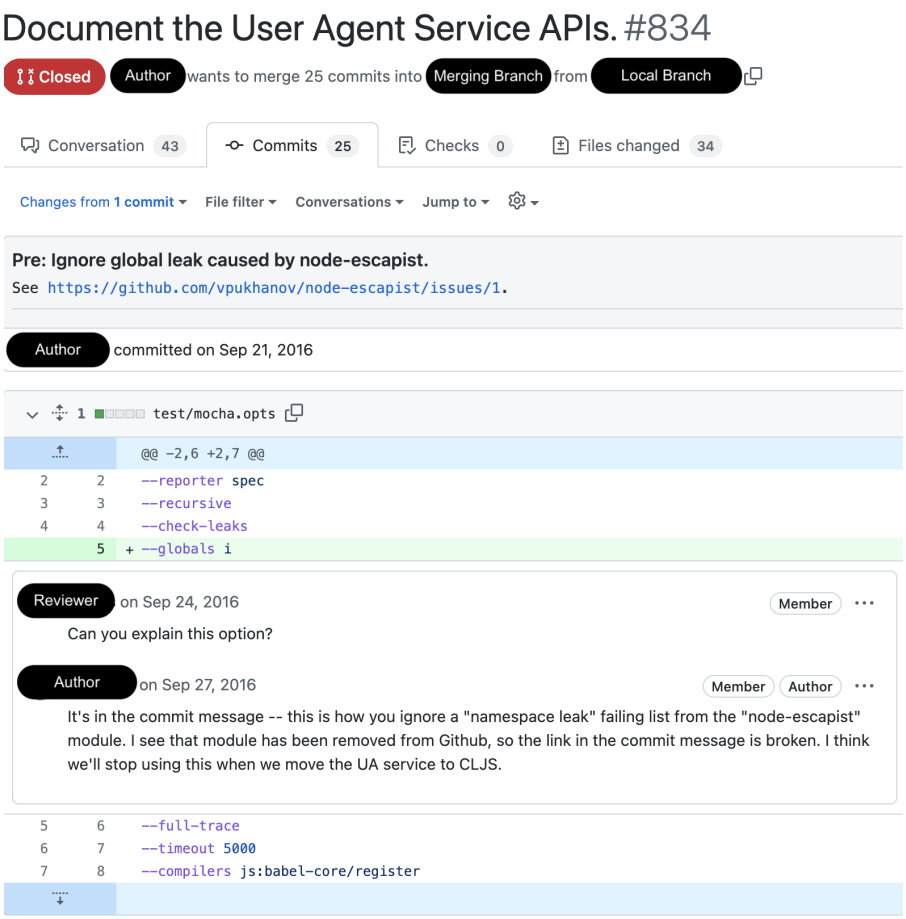


Fig. 1: Motivating example of link decay in the commit message.

investigate the phenomenon of links (e.g., repeated link reference, link evolution, and link decay), we conducted a mixed-methods study. Our results show that (i) the behavior of repeatedly referenced links rarely happens in commit messages; (ii) 14% of the links are prone to evolve to become unavailable over time, e.g., tutorial or article and software homepage are temporarily available software artifacts; and (iii) link decay is a common issue in commit messages. This extension makes the following main contributions:

- large-scale and comprehensive studies of around 18 million links to establish the prevalence, link decay, and case study of Stack Overflow link target evolution in commit messages,
- a mixed-methods study to identify types of link targets and purposes that links served, links evolution, and reasons why links are repeatedly referenced in commit messages,

- a comparison study of the role of links between source code comments and commit messages (i.e., RQs 1, 2, 3, 6, and a case study of Stack Overflow in RQ5), and
- a full replication package of our study, including the scripts and data set.[1]

The rest of the paper is organized as follows. Section 2 structures our six research questions and their motivations. Section 3 details our data collection process. Section 4 describes our methods in qualitative and quantitative analyses. Section 5 presents our research findings by answering the six aforementioned research questions. Section 6 discusses the comparison of the role of links between source code comments and commit messages and recommendations. Section 7 acknowledges threats to the validity of our study. Section 8 situates our work in relation to the literature on commit messages, knowledge sharing, and link sharing. Section 9 draws conclusions and highlights opportunities for future work.

2 Research Questions

In this section, we present our six research questions with the motivation to gain insight on how links are used in commit messages.

(RQ1) *How prevalent are links in commit messages?*

To gain an initial intuition about links and understand the usage of links in commit messages, we set out to quantitatively explore the distribution, diversity, and spread of these links across different types of software projects.

(RQ2) *What kinds of link targets are referenced in commit messages?*

(RQ3) *What purpose do links in commit messages serve?*

RQ2 and **RQ3** involve a more qualitative approach to analyze what role links play in what developers use in commit descriptions. Answering these RQs will help characterize why and how developers make references in a commit message.

(RQ4) *To what extent do commit message links get repeatedly referenced?*

(RQ5) *How prone are link targets to evolve over time?*

(RQ6) *How frequent are inaccessible links in commit messages?*

RQ4, **RQ5**, and **RQ6** investigate the phenomenon of links from the evolutionary and maintenance point of view. We would like to understand whether developers are repeating links, how these links evolve after introducing them, and how many of the links are affected by link decay.

3 Data Collection

To answer the research questions mentioned above, we focus on the same stratified sample of repositories as the previous study used in our earlier study of links in source code files (Hata et al., 2019). They collected active non-forked

[1] `https://doi.org/10.5281/zenodo.7536500`

repositories for the seven languages from the GHTorrent data set[2] (Gousios, 2013) using the following criteria: (i) having more than 500 commits in their entire history, and (ii) having at least 100 commits in the most active two years to remove long-term less active repositories and short-term projects that have not been maintained for long. The repository list is part of our replication package.

3.1 Commit Message Collection

To extract all links from the commit messages in these repositories, we used a bash script to retrieve all commit messages from the default branch of these repositories, exporting them to CSV files along with the commit metadata.[3] Since some repositories were not available because they had been deleted or made private, we obtained 27,263 (93%) repositories from the given repository list, created in 2018 (see Table 1 for details on the different strata). We then imported those CSV files into Google BigQuery tables, one per programming language.

3.2 Link Identification

Using the following regular expression in SQL queries, we extracted HTTP(S) URLs, which is the most common way of hosting or sharing resources, from collected commit messages stored in Google BigQuery tables:

```
(https?:\/\/(?:www\.)?[-a-zA-Z0-9@:%.\+~#=]{1,256}\.[a-zA-Z0-9()]{1,6}\b(?:[-a-
    zA-Z0-9()@:%\+.~#?&\/\/=]*))
```

We identified a total of 18,201,190 links from commit messages, as seen in Table 1. Since we will conduct a quantitative study on aspects of link domains (RQ1), we exclude 25 false-positive links in this analysis, whose domain is empty (e.g., "The learn more link should go to http://...answer=185277."). These false-positive links are malformed and served as an example of links in commit messages. As a result, we obtained 18,201,165 links that are used in this study. The bash and SQL scripts, commit messages, and resulting CSV files are available as part of our replication package.

4 Method

In this section, we describe the mixed-methods procedure that includes quantitative analysis (Section 4.1 for RQs 1, 5, and 6) and qualitative analysis (Section 4.2 for RQs 2, 3, and 4).

[2] MySQL database dump 2019-02-01 from `http://ghtorrent.org/downloads.html`.

[3] `https://github.com/sbaltes/git-log-extractor`

Table 1: Collected repositories and links.

	# repositories		# commits		# links
	candidate	obtained (%)	All	w/ links (%)	
C	2,771	2,607 (94%)	122.5M	7,256,770 (5.9%)	8,067,201
C++	3,563	3,391 (95%)	21.2M	4,188,042 (19.7%)	4,779,742
Java	4,995	4,701 (94%)	30.4M	1,758,495 (5.8%)	1,891,739
JavaScript	7,130	6,542 (92%)	13.8M	634,715 (4.6%)	778,667
Python	5,263	5,007 (95%)	15.9M	766,324 (4.8%)	859,396
PHP	3,279	2,941 (90%)	11.0M	1,335,934 (12.1%)	1,503,529
Ruby	2,233	2,074 (93%)	5.9M	214,647 (3.6%)	320,916
sum	**29,234**	**27,263 (93%)**	**220.8M**	**16,154,927 (7.3%)**	**18,201,190**

4.1 Quantitative Analysis

To get an intuition of links and their usages in commit messages (**RQ1**), and investigate the phenomenon of the evolution of link targets (**RQ5**) and link decay (**RQ6**), we conduct quantitative analyses of 18,201,165 links and a statistically representative and stratified sample.

Link Prevalence (RQ1) For RQ1, we conducted three quantitative studies on aspects of link existence, domain popularity, and popular domains in our data set. For link existence, we calculate the ratio of repositories with at least one link among seven programming languages. For domain popularity, we calculate the distribution of the number of different domains per repository. Median values are used to measure popularity. For popular domains, we calculate the top ten frequent repositories by counting only once in each repository.

Link Target Evolution (RQ5) To address RQ5, we conducted a quantitative study to investigate the evolution of link targets. We use the Wayback Machine JSON API[4][5] to obtain the closest snapshots of links from our 1,145 samples that are used in qualitative analysis from RQs2–4 in the next two years when we retrieved our data set (e.g., March 16, 2020, March 16, 2021, and March 16, 2022). This API will return a JSON object if the given link is archived and currently accessible in the Wayback Machine, and it will return an empty JSON object if the given link is not archived or currently not accessible. Then, we compare these availability statuses to the coding results from RQ2.

Link Decay (RQ6) For RQ6, we conducted a quantitative study on aspects of link decay in our data set. Out of the obtained 18,201,165 links, there are 6,667,207 distinct links. To investigate the number of inaccessible links in commit messages, we accessed all web content of the 6,667,207 unique links by using Perl modules `LWP::UserAgent` and `LWP::RobotUA` as in the previous

[4] `https://archive.org/help/wayback_api.php`

[5] `https://github.com/sbaltes/wayback-machine-retriever`

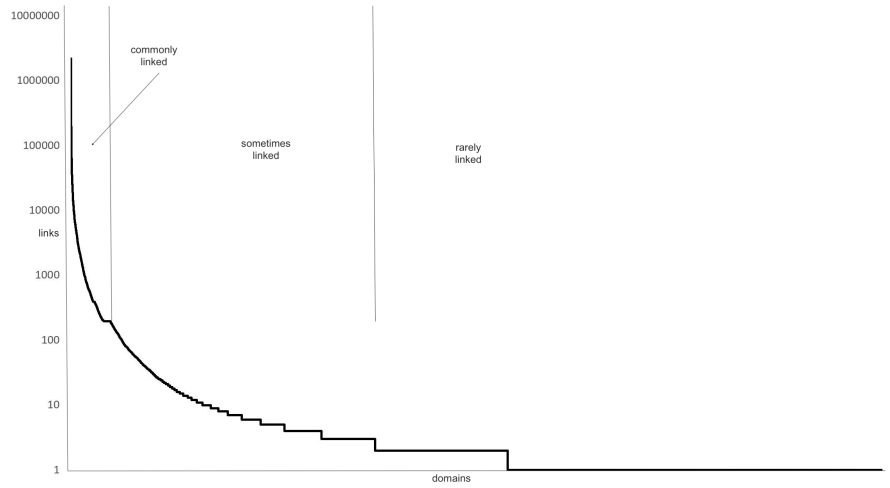


Fig. 2: Distribution of links per domain.

study (Hata et al., 2019). Then, we identified their HTTP status codes and considered `2xx` codes accessible links. Unlike in the previous study, we excluded error status codes (i.e., `4xx` and `5xx`) because error details are not the focus of this study. Finally, redirection status codes (i.e., `3xx`) occurred infrequently in the previous study—only 0.7% of the analyzed links were redirected. For this study, we consider such redirect links not accessible in terms of the original resource and thus excluded them. Additional retrieval and verification of their redirection targets would be required, with a marginal effect on the results. They could, however, be further analyzed in future work to better understand the evolution of software documentation.

4.2 Qualitative Analysis

To understand what kinds of link targets are referenced in commit messages (**RQ2**), understand what purpose do links in commit messages serve (**RQ3**), and investigate the phenomenon of repeated links in commit messages (**RQ4**), we conduct qualitative analyses of a statistically representative and stratified sample.

Link Target (RQ2) In RQ2, we conducted a qualitative study of a statistically representative and stratified sample of all links in our data set, to understand what kind of link targets are referenced in commit messages. Similar to the previous study of Hata et al. (2019), we divided the data into three strata: 1) links to commonly linked domains; 2) links to domains sometimes linked; and 3) links to rarely linked domains. To decide on thresholds for distinguishing domains into three strata, we conducted a visual analysis of the distribution

Table 2: Construction of the stratified sample.

strata	# domains	# links	# links in sample
common	2,270	17,870,470	384
sometimes	22,897	309,120	384
rare	21,575	21,575	377
sum	**46,742**	**18,201,165**	**1,145**

of links per domain in our data set. Figure 2 presents this distribution using a log scale with a cutoff frequency between 'common' and 'sometimes' of 194. We consider domains that account for exactly one link in our data set to be rarely linked, similar to the previous study of Hata et al. (2019).

Table 2 shows the number of domains and the number of links in each stratum. We randomly sampled a statistically representative number of links with a confidence level of 95% and a confidence interval of 5 from each stratum. In the results, we obtained 384 (from commonly linked domains), 384 (from domains sometimes linked), and 377 links (from rarely linked domains).

Hata et al. (2019) introduced the coding guide for link targets in comments of source code files. In this paper, we will reuse the coding guide to classify link targets. However, we found that their coding guide did not cover all link targets in commit messages. Hence, the following three additional codes emerged from our manual analysis in the first iteration:

- **repository**: online storage location of software packages.
- **pull request**: request to merge changes back into the main branch.
- **patch**: set of changes to a repository.

The first, third, and fourth authors of this paper independently coded 30 links from the sample and then calculated the Fleiss's kappa agreement (Fleiss, 1971) of this iteration between all three raters. The kappa agreement of the link target was 0.72 or "Substantial agreement" (Viera and Garrett, 2005). Based on this encouraging result, the remaining data was then coded by the first author of this paper. If the first author was unable to identify the code of the links (i.e., 13 cases), we engage in discussions to determine the appropriate type. This collaborative approach ensured that all links were accurately and consistently coded.

Link Purpose (RQ3) In RQ3, we use the same sample and process as RQ2. We conducted a qualitative analysis of our statistically representative sample, focusing on the purpose of the link. We also found that their coding guide (Hata et al., 2019) did not cover all link purposes in commit messages. Hence, the following three additional codes emerged from our manual analysis in the first iteration:

- **version control sync**: the link explicitly indicates that it is used for keeping files in sync between versions (e.g., merge branch and git-svn-id).
- **related issue**: the link relates to the issue report.

- **unknown (404)**: exception which is the original GitHub commit link cannot be accessed. Note that creating this code was likely not the purpose of adding the link, but since we cannot reliably access its content anymore, we code all such instances as unknown (404).

After coding 30 links from the sample, the kappa agreement reached 0.69, which is also "Substantial agreement" (Viera and Garrett, 2005). The remaining data was coded by the first author of this paper.

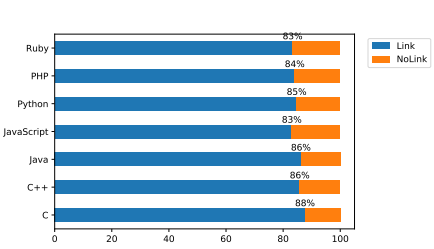
Repeated Links (RQ4) The corresponding research question of the previous study (Hata et al., 2019) is "How do links in source code comments evolve?" Unlike links in source code comments, it is unlikely that links in commit messages are edited in practice. Therefore, this research question, "To what extent do commit message links get repeatedly referenced?", investigates whether the same links appear in commit messages when the same files are modified.

We use the same sample from RQ2 and RQ3. For each link, we extract a commit history of the files modified by the commit, including the link. Then, we count the frequency of the link in the commit history. If the same link appears more than once, we consider the link as repeated. We manually read the commit messages to analyze relationships among the commits. We classify them into groups based on potential reasons in a bottom-up manner.

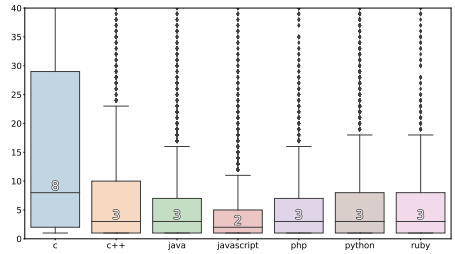
- **same data source**: includes links that appear in commits importing updates from external repositories. In particular, we classify a link into this group if:
  - the commit message is generated by a tool such as `Dependabot` and `Greenkeeper`, or
  - the link points to a language translation platform such as `translatewiki.net` and `Weblate`.

  In those cases, we consider that the links point to external data sources of the project.
- **same purpose**: includes the link pointing to development issues, including feature requests and bug reports. Links repeatedly appear in the commit history, probably because the developers needed a number of changes to resolve the issues. We classify a link into this group if:
  - the link target is an issue on an issue tracking system, or
  - commits including the link that has exactly the same message.
- **same reference**: includes the link that refers to either specification documents or API documents outside of the projects. We classify a link into this group based on the link target analyzed in RQ2.
- **other**: includes the link that could not be categorized due to limited information.

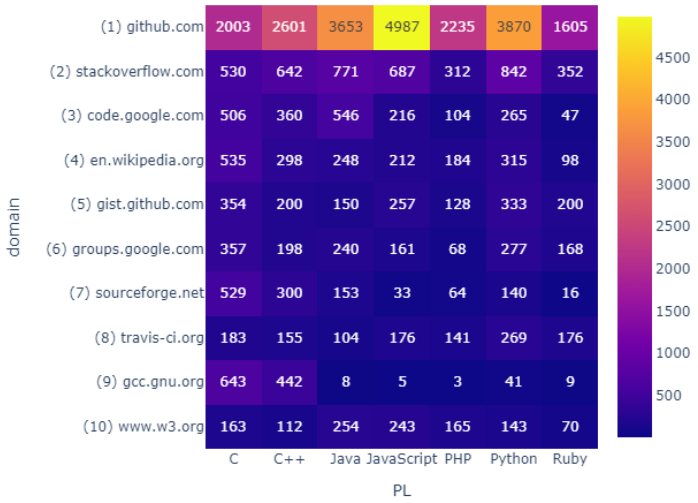
While the codes and rules emerged from manual analysis, the sample is analyzed by the above rules.



(a) Ratio of repositories with links.



(b) Distribution of the number of different domains per repository.



(c) Heatmap of repository languages shared by the top ten most referenced domains.

Fig. 3: Analysis of links by (a) languages, (b) domain diversity, and (c) top domains.

5 Results

In this section, we present the results of each research question.

5.1 Prevalence of Links (RQ1)

To explore the prevalence of links referenced in commit messages, we conducted quantitative analyses of our collected data set in terms of the existence of links, diversity of domains, and popularity of domains.

Table 3: Links to domain github.com by language. Same owner: for link pattern `github.com/([^/]+)`, the group matches the owner of the repository the commit belongs to; same repo: in addition, the second group matches the owner name and the repository name: `github.com/([^/]+)/([^/]+)`.

language	all	same owner		same repo	
C	178,250	42,310	(23.7%)	36,225	(20.3%)
C++	107,308	70,491	(65.7%)	54,544	(50.8%)
Java	194,086	141,324	(72.8%)	84,502	(43.5%)
JavaScript	318,043	170,478	(53.6%)	110,255	(34.7%)
PHP	116,368	61,632	(53.0%)	45,469	(39.1%)
Python	154,938	84,491	(54.5%)	52,805	(34.1%)
Ruby	128,686	28,700	(22.3%)	20,030	(15.6%)
sum	**1,197,679**	**599,426**	**(50.0%)**	**403,830**	**(33.7%)**

Table 4: Links to domain github.com pointing to the same repository (n=403,830), ten most frequent path segments following the repository name; none: link pointed to the repository root path.

path segment	link count	
none	187,400	(46.4%)
issues	107,479	(26.6%)
pull	82,051	(20.3%)
commit	14,667	(3.6%)
blob	8,389	(2.1%)
compare	1,637	(0.4%)
wiki	983	(0.2%)
tree	341	(0.08%)
releases	197	(0.05%)
projects	68	(0.02%)

Link existence. Figure 3(a) presents the percentages of studied repositories. Many repositories have at least one link in their commit messages, e.g., 83% for Ruby repositories. We observe that the percentages slightly vary by language, accounting for five percentage points. Additionally, for repositories written in Java, C++, and C, more than 85% of the repositories contain links to provide additional knowledge.

Domain diversity. We obtained 46,742 distinct domains or Internet hostnames, out of the obtained 18,201,165 links. Figure 3(b) shows the distribution of the number of different distinct domains per repository, with median values. We find that the median of different domains of repositories written in C is relatively greater than other studied languages, accounting for at least five different domains.

Popular domains. Figure 3(c) depicts the heatmap of repositories in each language shared by the top ten most referenced domains. To rank the top ten domains, we only counted once even if the domain has appeared several times, and used the number of repositories instead of the number of links.

The github.com domain is the most frequently occurring domain in our studied data set, accounting for 20,954 repositories across seven languages referencing content on github.com. As one of the most popular social coding platforms, Dabbish et al. (2012) found that transparency in GitHub allowed work to evolve with collaboration. Not surprisingly, the github.com domain is used frequently in commit messages. Hence, we decided to look at those links in detail. Table 3 shows how many links in the particular programming languages pointed to resources hosted under the same owner as the one owning the repository the commit belongs to. The table further shows how many links pointed to resources in the same repository (matching the owner name and the repository name). Overall, half of the github.com links were internal to the repository owner, and about a third of the links were internal to the repository, i.e., pointed to resources within the same repository. Focusing on the latter, we see that almost half of those links point to the repository root path, 26.6% point to issues, and 20.3% point to pull requests (see Table 4). This indicates that almost half of the repository-internal links are used in cases where GitHub's hash notation for linking issues or pull requests (`#issue_id` or `#pull_request_id`) could have been used.

Another commonly occurring domain is the stackoverflow.com domain, accounting for 4,136 repositories across seven languages referencing content on stackoverflow.com. This finding confirms the results of Vasilescu et al. (2013), they found Stack Overflow activities accelerate GitHub committing. Developers share external knowledge through links from Stack Overflow to GitHub to encourage committing since active developers in GitHub are also active questioners (Xiong et al., 2017).

The distribution of the top ten domains differs by language. But in summary, most domains are frequently referenced in C repositories, e.g., en.wikipedia.org, sourceforge.net, and gcc.gnu.org. The github.com domain is commonly referenced in 4,987 JavaScript repositories. Moreover, Java repositories contain more links with the code.google.com domain and www.w3.org domain. Repositories written in C referenced many links to the domains of en.wikipedia.org, gist.github.com, groups.google.com, sourceforge.net, and gcc.gnu.org.

RQ1 Summary: We observe that links in commit messages are prevalent. Most of the repositories have at least one link in their commit messages, e.g., 83% for Ruby. The top three most frequently referenced domains per repository are github.com, stackoverflow.com, and code.google.com.

5.2 Link Targets (RQ2)

Table 5 shows the result of our qualitative analysis. For all types of domains, in many of the cases we could not determine the link target type because of a

Table 5: Frequency of link target types in our sample. The bold target categories are complemented by this paper from the previous work (Hata et al., 2019). To prevent any confusion regarding the inclusion of "404" as a link target, we have decided to rename "404" (Hata et al., 2019) to "unknown (404)".

	Commit messages						Source code comments (Hata et al., 2019)					
	common		**sometimes**		**rare**		**common**		**sometimes**		**rare**	
unknown (404)	161	(42%)	132	(34%)	162	(43%)	27	(7%)	122	(32%)	138	(37%)
patch	110	(29%)	9	(2%)	2	(1%)	-	-	-	-	-	-
bug report	61	(16%)	23	(6%)	2	(1%)	9	(2%)	10	(3%)	3	(1%)
other	20	(5%)	49	(13%)	51	(14%)	5	(1%)	23	(6%)	45	(12%)
repository	14	(4%)	2	(1%)	1	(0%)	-	-	-	-	-	-
software homepage	6	(2%)	26	(7%)	23	(6%)	55	(14%)	65	(17%)	28	(7%)
organization homepage	4	(1%)	13	(3%)	15	(4%)	16	(4%)	41	(11%)	24	(6%)
tutorial or article	3	(1%)	34	(9%)	42	(11%)	16	(4%)	21	(5%)	31	(8%)
pull request	3	(1%)	1	(0%)	0	(0%)	-	-	-	-	-	-
forum thread	2	(1%)	28	(7%)	4	(1%)	0	(0%)	5	(1%)	6	(2%)
API documentation	0	(0%)	22	(6%)	12	(3%)	14	(4%)	20	(5%)	10	(3%)
blog post	0	(0%)	14	(4%)	23	(6%)	1	(0%)	10	(3%)	22	(6%)
specification	0	(0%)	12	(3%)	4	(1%)	21	(5%)	33	(9%)	32	(8%)
application	0	(0%)	7	(2%)	15	(4%)	0	(0%)	11	(3%)	13	(3%)
code	0	(0%)	5	(1%)	7	(2%)	6	(2%)	2	(1%)	5	(1%)
research paper	0	(0%)	4	(1%)	4	(1%)	0	(0%)	9	(2%)	13	(3%)
personal homepage	0	(0%)	2	(1%)	6	(2%)	4	(1%)	8	(2%)	4	(1%)
Q&A thread	0	(0%)	1	(0%)	1	(0%)	0	(0%)	0	(0%)	1	(0%)
license	0	(0%)	0	(0%)	2	(1%)	208	(54%)	4	(1%)	1	(0%)
book content	0	(0%)	0	(0%)	1	(0%)	0	(0%)	0	(0%)	2	(1%)
GitHub profile	0	(0%)	0	(0%)	0	(0%)	1	(0%)	0	(0%)	0	(0%)
Stack Overflow	0	(0%)	0	(0%)	0	(0%)	1	(0%)	0	(0%)	0	(0%)
sum	**384(100%)**		**384(100%)**		**377(100%)**		**384(100%)**		**384(100%)**		**378(100%)**	

missing link, accounting for 42%, 34%, and 43%, respectively. For commonly-linked domains, `patch` is the second most frequent type of link target, accounting for 29%. For domains that are sometimes and rarely linked, `tutorial or article` is the third most common type of link target. Moreover, this table reveals that the remaining link targets are distributed similarly (i.e., not more than 7%). The prevalence of the code `other` in the results for links to all linked domains is an indicator of the diversity of links present in commit messages.

RQ2 Summary: Inaccessible links are the most prevalent target type in commit messages, whereas various other types, such as `patch`, `bug report`, and `tutorial or article`, are also common.

5.3 Link Purpose (RQ3)

Table 6 shows the result of our qualitative analysis. For commonly-linked domains, `version control sync` is the most frequent purpose, accounting for 44% of links, followed by `metadata` (22%). For domains that are sometimes and rarely linked, `source code context` is the most common purpose (59–71%), followed by `source/attribution` (11–16%) and `related issue` (3–14%), respectively. This indicates that most links are used for keeping files in sync or adding additional information to fill the context in commit messages. We also

Table 6: Frequency of link purposes in our sample. The bold target categories are complemented by this paper from the previous work (Hata et al., 2019).

	Commit messages						Source code comments (Hata et al., 2019)					
	common		**sometimes**		**rare**		**common**		**sometimes**		**rare**	
version control sync	169	(44%)	18	(5%)	1	(0%)	-	-	-	-	-	-
metadata	85	(22%)	11	(3%)	12	(3%)	288	(75%)	131	(34%)	43	(11%)
related issue	70	(18%)	54	(14%)	12	(3%)	-	-	-	-	-	-
source code context	28	(7%)	225	(59%)	267	(71%)	18	(5%)	60	(16%)	80	(21%)
source/attribution	16	(4%)	44	(11%)	62	(16%)	27	(7%)	62	(16%)	75	(20%)
unknown (404)	11	(3%)	2	(1%)	1	(0%)	-	-	-	-	-	-
see-also	3	(1%)	14	(4%)	12	(3%)	28	(7%)	59	(15%)	51	(13%)
commented-out source code	1	(0%)	11	(3%)	9	(2%)	1	(0%)	17	(4%)	70	(19%)
link-only	1	(0%)	2	(1%)	1	(0%)	6	(2%)	24	(6%)	40	(11%)
self-admitted technical debt	0	(0%)	2	(1%)	0	(0%)	11	(3%)	16	(4%)	13	(3%)
@see	0	(0%)	1	(0%)	0	(0%)	5	(1%)	15	(4%)	6	(2%)
sum	**384(100%)**		**384(100%)**		**377(100%)**		**384(100%)**		**384**	**(100%)**	**378(100%)**	

observe a few false-positive cases (0–3%), which indicate the original GitHub commit link cannot be accessed.

Patterns in the relationship between link targets and purposes. Based on the qualitative analysis conducted to answer RQ2 and RQ3 about the targets and purposes of links in source code comments, we can now investigate the relationships between the different types of link targets and the different purposes which emerged from our qualitative analysis. To do so, we applied association rule learning using the apriori algorithm (Agrawal et al., 1994) as implemented in the R package arules[6] to our data, treating each link as a transaction containing two items: its target type and its purpose. We used four as the support threshold and 0.7 as the confidence threshold, i.e., all the rules that we extracted are supported by at least four data points, and we have at least a 70% confidence that the left-hand side of the rule implies the right-hand side.

Table 7 shows the association rules extracted from our data with these settings, separately for each stratum in our sample. Especially in commonly-linked domains, we observe a tight connection between keeping files in sync and links no longer being available, i.e., `unknown (404)`. 85% of the inaccessible links are related to the purpose of keeping files in sync, and the 78% of links that were added in commit messages for the purpose of keeping files in sync are found to be inaccessible. Other than inaccessible links being tightly connected with the purpose of keeping files in sync, we identified more relationships (e.g., between bug report links and the purpose of providing related issues, and links to software, organization, application, API documentation, and forum thread are associated with source code context).

After examining commit messages containing 171 Subversion-related links (links containing 'svn'), we found that five cases were inaccessible on GitHub, 164 messages contained the keyword 'git-svn-id', and two contained the keyword 'svnmerge'. The 'git-svn-id' is a keyword that appears when migrating from Subversion to Git, so the fact that this link is currently inaccessible does not mean that any important information is missing. The keyword 'svnmerge'

[6] `https://cran.r-project.org/web/packages/arules/index.html`

Table 7: Associations between link target type and link purpose.

strata	association rule			conf.	supp.
common	unknown (404)	⇒	version control sync	0.85	132
common	version control sync	⇒	unknown (404)	0.78	132
common	metadata	⇒	patch	0.85	72
common	bug report	⇒	related issue	0.97	56
common	related issue	⇒	bug report	0.80	56
sometimes	software homepage	⇒	source code context	0.81	21
sometimes	bug report	⇒	related issue	0.87	20
sometimes	version control sync	⇒	unknown (404)	0.94	17
sometimes	organization homepage	⇒	source code context	0.85	11
sometimes	application	⇒	source code context	0.86	6
rare	unknown (404)	⇒	source code context	0.77	125
rare	other	⇒	source code context	0.80	41
rare	software homepage	⇒	source code context	0.86	19
rare	application	⇒	source code context	1.00	15
rare	API documentation	⇒	source code context	0.75	9
rare	commented-out source code	⇒	unknown (404)	0.89	8
rare	forum thread	⇒	source code context	1.00	4

Table 8: Categories of links that repeatedly appear in the commit history.

Category	**common**	**sometimes**	**rare**	frequency
same data source	12 (3.1%)	6 (1.6%)	0 (0%)	2–260
same purpose	9 (2.3%)	8 (2.1%)	0 (0%)	2–8
same reference	0 (0.0%)	4 (1.0%)	0 (0%)	2–5
other	1 (0.3%)	6 (1.6%)	0 (0%)	2–28
sum	**22(5.7%)**	**24 (6.3%)**	**0(0%)**	**-**

is related to merging in Subversion, and not being able to access this link can be problematic for understanding merge details. However, the number of such commit messages is small.

RQ3 Summary: For domains that are sometimes and rarely linked, `source code context` is the most prevalent purpose in commit messages. The purpose `version control sync` is particularly common for commonly linked domains.

5.4 Repeated Links (RQ4)

Table 8 shows the number of categories we have identified in repeated links. We observed that only 5.7% and 6.3% of sample from commonly linked domains and domains sometimes linked appear more than once in their commit histories and no link repeatedly appears in rarely linked domains. The result shows that, in most cases, developers use external documents only once to complete a task.

Table 9: Evolution of the link status of our sample in Wayback Machine and our coding in RQ2.

20200316	**20210316**	**20220316**	**link target in RQ2**	**# of links**
200	200	200	Not unknown (404)	389 (34%)
200	200	200	unknown (404)	128 (11%)
200	200	Not Found	unknown (404)	1 (0%)
200	Not Found	200	unknown (404)	3 (0%)
Not Found	200	200	unknown (404)	1 (0%)
200	200	Not Found	Not unknown (404)	4 (0%)
200	Not Found	200	Not unknown (404)	3 (0%)
Not Found	200	200	Not unknown (404)	4 (0%)
Not Found	Not Found	Not Found	unknown (404)	322 (28%)
Not Found	Not Found	Not Found	Not unknown (404)	290 (25%)
sum				**1145**(100%)

Not unknown (404) represents the link targets that are not identified as unknown (404) in our coding. Not Found represents the links that are unavailable or not archived in Wayback Machine.

For commonly linked domains, `same data source` is the most frequent reason, accounting for 12 links, followed by `same purpose` (9). For domains that are sometimes linked, `same purpose` is the most common reason, accounting for 8 links, followed by `same data source` (6). Besides that, the repeated links due to `same data source` could vary from 2–260 times. The most frequent link is pointed to translatewiki.net. The result indicates that developers continuously use the platform to update their documentation.

This analysis examined links in commit messages when the same files were edited, whereas the previous study (Hata et al., 2019) examined how links in code comments were edited. Therefore, the results are not comparable.

RQ4 Summary: Repeatedly referenced links in commit messages occur infrequently. Four percent of our sample is repeatedly referenced in the history of commit messages. The `same data source` is the most common reason for developers repeatedly referencing links in commit messages. The frequency of the links being repeatedly referenced varies from 2–260 times for `same data source`.

5.5 Link Target Evolution (RQ5)

To investigate the evolution of link targets, we conducted a quantitative analysis of our sample. Table 9 shows the results of this analysis. Approximately a third of the links in our sample exist permanently, representing 34% of our sample. Among these permanent links, `tutorial or article` (67), `other` (54), and `software homepage` (47) are the most frequent link targets. In detail, most link targets are available that are considered official (e.g., 82% of

`repository` links, 85% of `software homepage` links, 86% of `application` links, 88% of `personal homepage` links, and 94% of `organization homepage` links) or documentation (e.g., 85% of `tutorial or article` links, 86% of `blog post` links, 88% of `API documentation` links, 100% of `book content` links, 100% of `license` links, and 100% of `Q&A thread` links) web pages. In contrast, we find the minority of link targets are available that are considered as temporarily available software artifacts, e.g., only 9% of `patch` links, 25% of `pull request` links, 38% of `research paper` links, 41% of `bug report` links, and 41% of `forum thread` links.

We further investigate a total of 133 links (the third row to the sixth row of Table 9) that exist at least in one closest snapshot but we identify as `unknown (404)` in RQ2. We code link targets of these links in Wayback snapshots following our coding guide in RQ2. We find that 24 links are still coded as `unknown (404)`, accounting for 18% of these links. In addition to that, `tutorial or article` and `software homepage` are frequently occurring link targets, accounting for 20 links (15%) and 19 links (14%), respectively. We also code 11 links (the seventh row to the ninth row of Table 9) that have at least existed in one closest snapshot and we identified them as `Not unknown (404)` in RQ2. We observe that the link targets are not changed.

28% of the links in our sample are unavailable or not archived in snapshots and our coding. Besides that, a quarter of links in our sample are unavailable or are not archived in snapshots but are not identified as `unknown (404)` in our coding. To investigate whether these 290 links are unavailable or are not archived in snapshots, we attempted to request these 290 links with a maximum of ten retries. We find that only 24 of 290 links are unavailable (i.e., 15 links are identified as `other` in RQ2). Thus, we find 14% of links ($\frac{24+133}{1145} \approx 14\%$) that evolve and become unavailable over time.

These results indicate that links in commit messages are fragile, unstable, and easily inaccessible; these links could cause knowledge loss from this communication channel between commit authors and developers.

To investigate the evolution of link targets in more detail, we conducted one quantitative case study with the subset of links pointing to Stack Overflow. Of the 18,201,165 links, there are 9,696 links pointing to 7,315 distinct link targets on stackoverflow.com. Among those Stack Overflow links, there are varieties of expressions, e.g., an abbreviated path to an answer (`/a/(answer id)`), an abbreviated path to a question (`/q/(question id)`), and a full path to a question (`/questions/(question id)/(title)`). Older links start with '`http://`' and newer links start with '`https://`'.

We manually removed 122 commit-link pairs where the link was not referring to a specific Stack Overflow thread (e.g., Stack Overflow homepage), which left us with links to 6,973 unique post IDs. For each Stack Overflow link, we identified the timestamp of when the link was referenced in a commit message. For duplicate links, we consider only the oldest timestamp, leaving us with 6,973 distinct links and the oldest commit that referred to them. We created a statistically representative random sample of 364 links (confidence

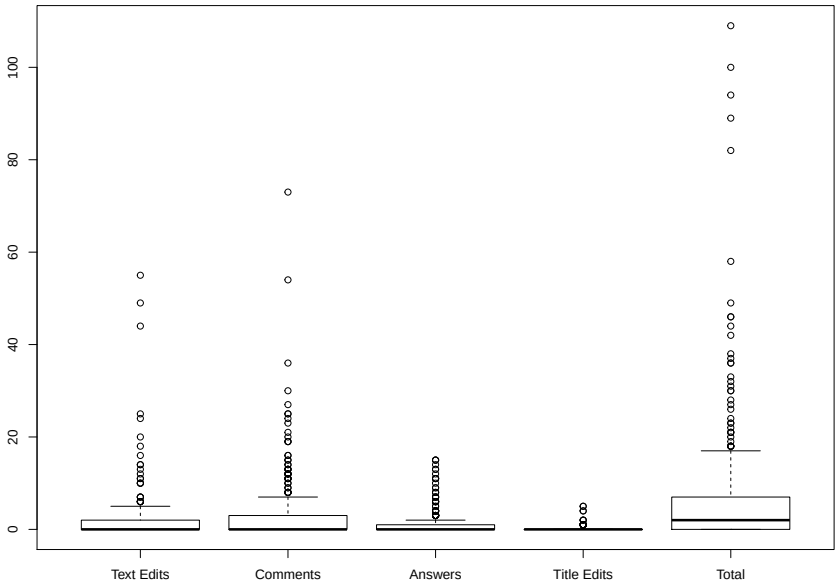


Fig. 4: Number of changes to the linked Stack Overflow threads after they were first referenced.

interval: 5%; confidence level: 95%) of this data for our analysis.[7] The calculation of statistically significant sample sizes based on population size, confidence interval, and confidence level is well established by Krejcie and Morgan (1970). We excluded three commits that were no longer available on GitHub at the time of analysis.

We then used the SOTorrent data set (Baltes et al., 2018) to investigate the extent to which Stack Overflow content had changed since the link to the question or answer had been referenced in a commit message. We queried SOTorrent to determine the following metrics for each link:

- the number of text edits on any post (i.e., question or answer) in the same thread,
- the number of new comments on any post (i.e., question or answer) in the same thread,
- the number of new answers in the same thread, and
- the number of edits to the thread title.

Figure 4 shows the results of this analysis. More than half of all Stack Overflow threads experienced at least one change (median: 2, third quartile:

[7] `https://www.surveysystem.com/sscalc.htm`

7) after they were referenced in a commit message, and more than a quarter of these links attracted at least one new comment in the meantime (median: 0, third quartile: 3). Although the number of new answers to a thread was zero in the median case, a quarter of the Stack Overflow threads attracted at least one new answer after the link was referenced in a commit message (median: 0, third quartile: 1). In total, only 142 (39%) of the 361 Stack Overflow threads in our sample did not undergo any change after being included in a commit message.

The most extreme example in our sample—a Stack Overflow thread titled "C++ Singleton design pattern"[8]—had 11 new answers added (out of a total of 24) and attracted 73 new comments and 25 edits after being first referenced in a commit message in early July 2014.

RQ5 Summary: In summary, 14% of links are prone to evolve and become unavailable over time. Moreover, in a case study of Stack Overflow, we find that more than half of Stack Overflow threads linked in commit messages attracted at least one change (text edit, new comment, new answer, or title edit) after they were referenced.

5.6 Link Decay (RQ6)

We identified 4,659,236 inaccessible links from the web content of the 6,667,207 unique links (70%). Table 10 shows the domains with more than 30,000 inaccessible links. The svn.apache.org is the domain that occurs most frequently in these inaccessible links, accounting for 585,149 of the 607,873 links in commit messages (96%).

Upon closer inspection, most domains with a large number of inaccessible links are related to SVN (SubVersioN) repositories; for instance, svn.apache.org is the domain for Apache Software Foundation Subversion Server. This result confirms our finding in RQ3 and RQ5, which inaccessible links and the purpose of keeping files in sync are tightly connected in our sample (RQ3), and patch is temporarily available software artifacts (RQ5). The gerrit.instructure.com domain is used for Gerrit Code Review, which requires sign-in to be accessed. Besides that, the links of github.com domain are unable to be accessed since they are no longer available or pointing to private repositories. Moreover, only 8% of the links belonging to github.com domain are inaccessible in our data set.

Most of the inaccessible links in commit messages belong to domains for SVN repositories (e.g., svn.apache.org) instead of github.com domain in source code comments. These SVN repository domains have a high chance to be inaccessible, i.e., at least 96%. We also observed these SVN-related links are inaccessible in RQ3, usually these links are introduced in commit messages

[8] `https://stackoverflow.com/q/1008019`

Table 10: Domains with large number of inaccessible links

domain	# inaccessible links	total links	% inaccessible links
svn.apache.org	585,149	607,873	(96%)
llvm.org	408,888	410,396	(99%)
svn.code.sf.net	370,339	370,351	(99%)
www.virtualbox.org	159,144	159,178	(99%)
svn.osgeo.org	89,719	89,721	(99%)
anonsvn.ncbi.nlm.nih.gov	76,725	76,725	(100%)
root.cern.ch	75,289	75,568	(99%)
svn.wxwidgets.org	68,954	68,954	(100%)
svn.aros.org	52,884	52,884	(100%)
dev.geogebra.org	46,975	46,988	(99%)
svn.blender.org	46,030	46,067	(99%)
develop.svn.wordpress.org	41,321	41,325	(99%)
core.svn.wordpress.org	41,213	41,213	(100%)
svn.codehaus.org	41,202	41,202	(100%)
svn.parrot.org	39,973	39,973	(100%)
svn.erp5.org	38,292	38,322	(99%)
source.sakaiproject.org	36,614	36,684	(99%)
v8.googlecode.com	36,013	36,013	(100%)
gerrit.instructure.com	34,884	36,708	(95%)
svn.opendtect.org	33,977	33,977	(100%)
svn.jboss.org	33,799	33,799	(100%)
github.com	33,398	405,784	(8%)

with the keyword 'git-svn-id' that appears in the transition from Subversion to Git, so the fact that this link is currently inaccessible does not mean that any important information is missing.

RQ6 Summary: We observe that 70% of links are not available, considering all unique links. The most frequently occurring domains accounting for inaccessible links are related to SVN repositories.

6 Discussion

In this section, we first compare the different roles that links play between source code comments and commit messages. Second, we make recommendations to developers and researchers based on our results.

6.1 Comparisons of the roles of links between source code comments and commit messages

Based on our results, we make the following comparison in each RQ.

Prevalence of Links (RQ1). On the one hand, we find that the average ratio of repositories with links decreases from 89% (Hata et al., 2019) to 85% in Figure 3(a). Especially, PHP has the highest ratio of repositories with links

in source code comments (92%), has decreased to the third lowest ratio of repositories with links in commit messages (84%) in Figure 3(a), accounting for eight percentage points.

On the other hand, we obtained around two times as many links as in source code comments (9.6 million links). Furthermore, domain diversity in commit messages (46,742 distinct domains of 18 million links) is lower than links in source code comments (57,039 distinct domains of 9.6 million links). The median values of different domains per repository have decreased by a margin of 2–10 domains in Figure 3(b). The code.google.com domain is more frequently referenced in commit messages (from 7th rank to 3rd rank in Figure 3(c)). The ranks of licensing-related domains decreased in commit messages, e.g., www.apache.org (4th rank in source code comments) and www.gnu.org (6th rank in source code comments). However, the ranks of committing related domains increased, e.g., travis-ci.org.

These comparison results indicate that developers tend to reference more links in commit messages to provide commit-related information to reviewers or other developers, however, the diversity of domains is lower than for links in source code comments.

Link Targets (RQ2). Table 5 shows the results of link types in commit messages and source code comments. The decay of links (inaccessible) in commit messages is worse than in source code comments. In detail, the inaccessible links have increased from 7–37% to 34–43%. Even for the domains that are commonly linked, link decay has become a common issue in commit messages (from 7% in source code comments to 42% in commit messages).

Due to the different nature of code comments and commit messages, the license information has changed from the majority of link targets in source code comments (0–54%) to the minority of link targets in commit messages (0–1%). This finding confirms the result of RQ1, that the license information domains are not in the top ten popular domains in commit messages. On the one hand, there are two link targets (`GitHub profile` and `Stack Overflow`), which are not found in commit messages. On the other hand, we also find three new link targets, such as `repository`, `pull request`, and `patch`.

These comparison results indicate that developers tend to reference links that are temporarily available in commit messages. These links are short-lived in the commit process even though they are linked to common domains. In commit messages, developers are more likely to reference commit process-related links, e.g., bug reports, pull requests, and patches.

Link Purpose (RQ3) Table 6 shows the results of link purposes in commit messages and source code comments. Providing metadata purpose has been dramatically reduced to 3–22% in commit messages from 11–75% in source code comments, as commit messages focus on the context of changes, rather than licenses or author information.

On the contrary, providing context purpose has been indirectly increased to 7–71% in commit messages from 5–21% in source code comments (especially for domains that are rarely linked). On the one hand, four link purposes (`commented-out source code`, `link-only`, `self-admitted technical debt`,

and `@see`) become rare or even do not appear in commit messages. On the other hand, we also observe two new purposes, such as `version control sync` and `related issue`.

These comparison results indicate that developers tend to reference short-lived link targets to keep files in sync. Since the main purpose of the commit message is to identify changes in this commit, developers tend to reference diverse link targets with the purpose of providing additional information related to the commit.

Link Target Evolution (RQ5) We can compare the results of the Stack Overflow case study to links in source code comments (Hata et al., 2019). In Figure 4, the median values of each change in commit messages share the same trend in source code comments, i.e., Comments > Text Edits > Answers > Title Edits. However, only 91 (24%) of the 372 Stack Overflow threads in source code comments did not undergo any change. These comparison results indicate that Stack Overflow links referenced in commit messages tend to be less prone to change than those referenced in source code comments.

Link Decay (RQ6) When comparing this result to links in source code comments (Hata et al., 2019), the link decay in commit messages is heavily worse than in source code comments from this quantitative analysis, i.e., 70% of links in commit messages and 18.8% of links in source code comments are inaccessible. These comparison results indicate that commit messages often contain links to old systems or services and are in danger of information loss due to broken links.

As we have discussed above, the behaviors that developers follow to reference external resources in the two different software artifacts are different. Source code comments and commit messages are two important software artifacts in the development process that developers can use to share knowledge related to code or commits with their communities. It is important to understand the role of links in these two software artifacts, since compared to natural language, links contain richer information that could be useful in understanding code or commits.

In our paper, we find that developers tend to reference more links from fewer domains (RQ1) in commit messages; however, these links are more likely to be short-lived and related to the commit process with the purpose of keeping files in sync (e.g., bug report, pull request, and patch) (RQs1–3). To include complete information to describe the commit to the extent possible, developers reference diverse links (e.g., software, organization, application, API documentation, and forum thread) (RQ3). The link decay in commit messages is substantially worse than in source comments, especially for links to SVN repositories (RQ6). This issue could be a major cause of knowledge loss related to commit messages. In a case study of links pointing to Stack Overflow, we find that Stack Overflow links referenced in commit messages tend to be less prone to change than similar links in source code comments (RQ5).

In summary, we can see that links in source code comments are complementary to the source code, they provide relatively longer-lived information to help developers easily understand the metadata of code (i.e., license or soft-

ware homepage as basic information for this code). In terms of the commit messages, links are more likely to be used as additional information related to changes or as a sign for syncing files to explain changes.

6.2 Recommendations

Based on our findings, we make the following recommendations for developers and researchers. First, we recommend to developers:

- **Pay special attention to maintaining commit related links**. We find that developers tend to use links to reference software artifacts that are short-lived and related to the commit process (e.g., bug report, pull request, and patch). Developers should pay special attention to maintaining these links, since these software artifacts are important for understanding a commit. Moreover, these short-lived software artifacts can cause knowledge loss in the community.
- **Reference permanent links in commit messages**. We find that inaccessible links are prevalent in commit messages (34–43% of links in our sample and 70% of the distinct links in our data set). Commit messages are a critical means of communication between developers and reviewers, and links in commit messages are special containers for providing external and internal knowledge. Building stable traceability links between diverse software artifacts (e.g., bug reports, patchs, and API documentation) and commits could support knowledge sharing in the community.
- **Fix inaccessible links when generating release notes from commit messages**. We find that link decay is a common issue in commit messages. Commit messages can be used to generate release notes. Wu et al. (2022) analyzed 1,731 GitHub issues that are related to release notes, they found that 20.81% of these issue reports complained about wrong or broken links in the release note. Thus, to avoid suffering from the link decay issue in release notes, developers should fix or omit these inaccessible links from the release notes generation.

Second, we recommend to researchers:

- **Further studies of traceability links between other software artifacts and commits**. We find various link targets exist in commit messages, e.g., patch, bug report, and tutorial or article. Bug reports have been studied, for example, Sun et al. (2017) recover missing issue-commit links by revisiting file relevance. We suggest that researchers should investigate traceability links between other software artifacts and commits.
- **Estimate the impact of the link decay in commit messages.** In **RQ2** and **RQ3**, we find that role of the inaccessible links is associated with the purpose of keeping files in sync. We argue that link decay is indeed a problem, as it renders the information resource useless (i.e., a reviewer or user cannot access the link that holds supplementary information). Since

this breakdown in information will result in a breakdown in communication and knowledge acquisition in the project, our plan is to detect cases where a inaccessible link generates a discussion among interested parties. Depending on the situation, these removals of informal may cause delays in review time, increased discussion, and overall acceptance of code commits. Analyzing such characteristics, for example, could shed light on the negative impact of inaccessible links on software development.

– **Tools to support common commit message templates**. We find that links in commit messages point to a smaller range of different domains compared to links in source code comments, suggesting that there are common use cases for links in commit messages. A promising research direction could be the creation of templates for commit messages to increase the probability that developers remember to add links where relevant, for example, following the work on using stereotypes to characterize commits (Dragan et al., 2011).

7 Threats to Validity

In this section, we discuss the threats to the validity of our study.

Construct Validity. To compare the role of links between source code comments and commit messages, we used the same stratified sample of repositories as the previous study (Hata et al., 2019). However, the use of the same list of repositories may introduce an inconsistency in the comparison due to the presence of private or unavailable repositories. To keep the consistent comparison between links in source code comments and commit messages, we only consider http(s) links as URLs in commit messages by the regular expression. For work on other kinds of links in the context of software development, we refer readers to Schermann et al. (2015) who studied the interlinking characteristics in commits and issues. Future work is needed to gain a global picture of links in commit messages for different URL protocols, e.g., FTP and SSH.

Content Validity. In our study, we manually classified a sample of 1,145 links in commit messages, which carries the risk of undiscovered link targets and link purposes in commit messages. Since we are unable to reliably infer the type and the intended purpose of links for which the target is no longer accessible, we decided to code such links consistently as `unknown (404)` similar to previous study (Hata et al., 2019). Trying to guess the type and intended purpose for a subset of these links (e.g., `https://svn.apache.org/repos/asf/subversion/trunk@851235` could be coded as a `patch`) would likely have had an impact on the results reported for link types and purposes since the target content is not always obvious from the link alone and since not all links were archived in a consistent manner in Wayback Machine or similar services. Finally, as part of the qualitative analysis, we defined three strata of samples generated visually from the distribution of the data. Although subjective, we are confident in this sampling, especially since this data does not follow a normal Gaussian distribution.

Internal Validity. To answer RQs2–4, we conducted qualitative studies of a sample of all links in our data set. These qualitative studies are manual analyses that were conducted according to our coding guides. These codes may be inadequate due to the subjective nature of understanding the coding guides. To mitigate this threat, we require Fleiss's kappa agreements (Fleiss, 1971) of at least "Substantial agreement" for the understanding of the coding guides in RQs2–3, following previous work (Wang et al., 2021; Xiao et al., 2021). In RQ4, we define the coding guides as rules that automatically classify links. The rules are dependent on the link target domains and the coding results of RQ2.

External Validity Although we analyzed a large number of commit messages from GitHub repositories, our results may not generalize to industry or other open-source artifacts, in general. Some open-source repositories are hosted outside of GitHub, e.g., on GitLab or private servers.

8 Related Work

In this section, we discuss existing work related to commit messages, knowledge sharing, and link sharing.

8.1 Commit Messages

In modern software development, developers submit commits to version control systems to integrate new features or fix bugs. Commit messages are required to document or summarize such changes. While the size and complexity of software systems grow with an increasing number of commits, those commit messages become critical to understanding code changes, especially if issues occur.

In the study of Alali et al. (2008), they found that commit messages are coupled with three size-based characteristics of commits (number of files, lines, and hunks). As external documentation of source code changes, commit messages play a critical role in open-source projects. Open-source projects often have rules about commit messages to ensure governance, but as a community based on voluntary contributions, it is difficult to enforce such rules (O'mahony and Ferraro, 2007). Therefore, Dyer et al. (2013) observed that there have been around 14% of commit messages in more than 23,000 open-source SourceForge Java projects that are completely empty. Santos and Hindle (2016) used the n-gram cross entropy of text in commit messages to successfully identify commits that were likely to make a build fail. Commit messages are also used as a link between issue reports and commits (Xie et al., 2019), recommend refactoring opportunities (Rebai et al., 2020), detect and classify refactoring descriptions (Krasniqi and Cleland-Huang, 2020), identify security issues (Zhou and Sharma, 2017), and identify whether this commit can be skipped by continuous integration (Abdalkareem et al., 2020). However, due to the voluntary contribution of open-source projects and time pressures of developers (O'mahony

and Ferraro, 2007; Maalej and Happel, 2009; Murphy, 2009; D'Ambros et al., 2010), commit messages can be non-informative, lack information, meaningless, or completely empty (Maalej and Happel, 2010; Tian et al., 2022; Dyer et al., 2013; Liu et al., 2020). Therefore, many researchers have proposed commit message generation tools to facilitate the understanding of commit changes. For example, Liu et al. (2020) described ATOM, a tool to encode abstract syntax tree paths of diffs to generate commit messages. Similarly, Huang et al. (2020) presented ChangeDoc, an approach to generate commit messages from existing messages. In this study, we recommend that developers include permanent links in their commit messages to prevent knowledge loss during the commit process. However, it is possible that developers may not always follow this recommendation. Therefore, we suggest implementing an approach to automatically generate commit messages and archive links for changes, which would facilitate the preservation of knowledge during the commit process.

Moreover, commit messages are critical for knowledge transfer during project maintenance. Fu et al. (2015) presented a semi-supervised Latent Dirichlet Allocation based approach to automatically classify change messages (i.e., Corrective, Perfective, and Adaptive). In validation surveys, they found that this approach can be applied to cross-projects; automatic classification results can reach around 70% in agreements with developers. Sarwar et al. (2020) also presented an approach using transfer learning to classify commit messages into the same categories. Commit messages help developers understand changes and their underlying rationale with less effort, process influences, and less bias between communication. Mockus and Votta (2000) identified four types of changes based on commit messages: adding new functionality, repairing faults, restructuring the code to accommodate future changes, and code inspection rework. The messages help to reduce efforts on other software development tasks: generate release notes (Moreno et al., 2014). In this study, we did not examine the relationship between types of links and types of changes. However, we plan to address this aspect in future research.

Unlike the aforementioned studies, our study investigates links as one representation of knowledge in commit messages. Investigating links in commit messages enables us to understand the role they play as well as potential issues with links in commit messages. In our notion, links form an important communication channel between the initial code authors and later reviewers and maintainers.

8.2 Knowledge Sharing

Knowledge sharing is a central aspect of software development. Knowledge is transferred or exchanged among people and communities on online platforms, for example, Wikipedia (Kittur and Kraut, 2010; Forte et al., 2012; Nagar, 2012) and GitHub (Dabbish et al., 2012). Dabbish et al. (2012) interviewed light and heavy GitHub users, and showed that developers made a variety of social inferences about other developers and projects to collaborate, learn and

manage projects using the networked activity information. This study also suggested that transparency of such the networked activity information can support knowledge sharing, new ideas, and community. Wattanakriengkrai et al. (2022) investigated the referencing of academic papers in README files of GitHub repositories. They found that this knowledge sharing is rarely occurring (0.4%), however, the majority of these referenced academic papers are open access (98.5%). We also found a few links referenced to academic papers in commit messages, which indicate such knowledge sharing is also rarely occurring in commit messages.

In addition to GitHub, Aniche et al. (2018) surveyed the Reddit programming subreddit and analyzed Reddit and Hacker News posts. They found that development posts are the highly occurring theme, and these posts are mostly related to programming or markup languages. Developers on the r/programming subreddit aim at learning, but on Hacker News, they focus more on publicizing news. They also suggested researchers expand Reddit and Hacker News as sources for developer knowledge. Furthermore, other research focuses on knowledge sharing on Q&A sites (Movshovitz-Attias et al., 2013; Vasilescu et al., 2014; Baltes et al., 2022), between Q&A sites and GitHub (Vasilescu et al., 2013), and GitHub discussions (Hata et al., 2022). Inspired by these works, we expand links in commit messages as a source for developer knowledge.

Although plenty of studies widely investigate knowledge sharing in software engineering, there is no study that focuses on links as special containers that provide additional knowledge for developers in commit messages. In this study, we focus on the different link targets and purposes that they served on commit messages, rather than natural language, to study knowledge sharing.

8.3 Link Sharing

As one convenient way of knowledge sharing, link sharing has been widely adopted and explored in developer communities (i.e., Q&A sites, GitHub, code review). Gómez et al. (2013) found that a significant proportion of links shared on Stack Overflow, in particular, were used to transfer knowledge about software development innovations such as libraries and tools. Ye et al. (2017) also analyzed link sharing activities in Stack Overflow to study the structural and dynamic properties of the emergent knowledge network in Stack Overflow. They discovered that developers share links for diverse purposes, e.g., reference information for problem solving is the most occurring purpose. In addition to that, external links are investigated on aspects of broken links (Liu et al., 2021) and repeatedly referenced links Liu et al. (2022b) in Stack Overflow posts.

A previous study (Hata et al., 2019) analyzed 9.6 million links that exist in source code comments. They explored that link sharing is common in source code comments, more than 80% of the repositories contained at least one link. They also identified the kinds of link targets (i.e., licenses, software homepages, and specifications) and link sharing purposes (providing meta-

data or attribution). Zampetti et al. (2017) investigated to what extent and for which purpose developers refer to external online resources in pull requests. The findings of their investigation suggest that developers frequently consult external resources for the purpose of acquiring new knowledge or resolving specific issues. Aghajani et al. (2019) presented an empirical study which shows "Outdated/Obsolete references" are one of the issues in software documentation. Furthermore, Baltes and Diehl (2019) found that 40% of their survey participants added a source code comment in GitHub projects with a Stack Overflow link to the corresponding question or answer. In addition to the Stack Overflow link to a question or answer, issue report links were studied. Zhang et al. (2018) showed that developers tend to link more cross-project or cross-ecosystem issues over time. Zhang et al. (2020) proposed an approach called iLinker for issue knowledge acquisition in GitHub projects, it can improve the development efficiency of GitHub projects. Similar to previous work, we also analyzed links in GitHub artifacts. We found that link decay is a frequent phenomenon in commit messages. Moreover, developers reference links for purpose of providing context.

Rath et al. (2018) addressed missing links between commits and issues, and proposed an approach to generate the missing links by a combination of process and text-related features. The practice of link sharing was also studied in the context of code review. Wang et al. (2021) performed a mixed-method approach to highlight the role that shared links play in the review discussion. Their results show that the link is served as an important resource to fulfill various information needs for patch authors and review teams. Liu et al. (2022a) proposed an approach to identify references between projects by extracting links from pull requests, issues, and commits. These links were obtained by patterns of URL, ID of issue, pull request or commit. Unlike these works in the code review process, we focus on the comparison between the roles of links in commit messages and source code comments to obtain an understanding of informal and initial knowledge sharing in the software development process.

Inspired by these past studies of link sharing, we conduct the first study on links in commit messages. Similar to prior work, we investigate the prevalence, targets, purposes of links and the phenomenon of links (i.e., repeated link reference, link evolution, and link decay) in commit messages.

9 Conclusion

In this paper, we conducted: (i) a quantitative study of 18 million links from commit messages in 23,110 Git repositories to investigate the prevalence of links in commit messages; (ii) qualitative studies of a stratified sample of 1,145 links to determine the kinds of link targets, the purposes of referencing these links and their repeated links; (iii) a quantitative study to investigate the evolution of link targets; and (iv) a quantitative study to investigate how the

link decay issue is common in commit messages and investigate which domains frequently affect link decay.

We observed that (i) links are frequently occurring in commit messages, accounting for at least 83% of GitHub repositories in our study; (ii) 34–43% of links in our sample are unavailable, other than that, links to patch are common in commit messages; (iii) the purpose of adding additional information is the most prevalent in commit messages; (iv) repeated links are rarely occurring in commit messages, accounting for four percent of our sample; (v) 14% of the links are prone to evolve to become unavailable over time; and (vi) link decay is a common issue in commit messages, around 70% of the distinct links in our study are inaccessible. We foresee many promising avenues for future work, such as tool support to fix inaccessible links, expanding our coded corpus to other software artifacts, and further studies of commit messages.

Acknowledgements This work was inspired by the International Workshop series on Dynamic Software Documentation, held at McGill's Bellairs Research Institute, and was supported by JSPS KAKENHI under Grants JP18H04094, JP20K19774, JP20H05706, and JP22K11970.

Data Availability The datasets generated during and/or analysed during the current study are available in the Zenodo repository, `https://doi.org/10.5281/zenodo.7536500`.

Funding

Declarations

Conflict of Interests The authors declare that Sebastian Baltes, Hideaki Hata, Christoph Treude, and Raula Gaikovina Kula are members of the EMSE Editorial Board. All co-authors have seen and agree with the contents of the manuscript and there is no financial interest to report.

References

Abdalkareem R, Mujahid S, Shihab E (2020) A machine learning approach to improve the detection of ci skip commits. IEEE Transactions on Software Engineering

Aghajani E, Nagy C, Vega-Márquez OL, Linares-Vásquez M, Moreno L, Bavota G, Lanza M (2019) Software documentation issues unveiled. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, pp 1199–1210

Agrawal R, Srikant R, et al. (1994) Fast algorithms for mining association rules. In: Proc. 20th int. conf. very large data bases, VLDB, Citeseer, vol 1215, pp 487–499

Alali A, Kagdi H, Maletic JI (2008) What's a typical commit? a characterization of open source software repositories. In: 2008 16th IEEE international conference on program comprehension, IEEE, pp 182–191

Aniche M, Treude C, Steinmacher I, Wiese I, Pinto G, Storey MA, Gerosa MA (2018) How modern news aggregators help development communities shape and share knowledge. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), IEEE, pp 499–510

Baltes S, Diehl S (2019) Usage and attribution of stack overflow code snippets in github projects. Empirical Software Engineering 24(3):1259–1295

Baltes S, Dumani L, Treude C, Diehl S (2018) Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts. In: Proceedings of the 15th international conference on mining software repositories, pp 319–330

Baltes S, Treude C, Robillard MP (2022) Contextual documentation referencing on stack overflow. IEEE Trans Software Eng 48(2):135–149, DOI 10.1109/TSE.2020.2981898, URL `https://doi.org/10.1109/TSE.2020.2981898`

Barrie JM, Presti DE (2000) Digital plagiarism-the web giveth and the web shall taketh. Journal of medical Internet research 2(1):e6

Buse RP, Weimer WR (2010) Automatically documenting program changes. In: Proceedings of the IEEE/ACM international conference on Automated software engineering, pp 33–42

Dabbish L, Stuart C, Tsay J, Herbsleb J (2012) Social coding in github: transparency and collaboration in an open software repository. In: Proceedings of the ACM 2012 conference on computer supported cooperative work, pp 1277–1286

D'Ambros M, Lanza M, Robbes R (2010) Commit 2.0. In: Proceedings of the 1st Workshop on Web 2.0 for Software Engineering, pp 14–19

Dragan N, Collard ML, Hammad M, Maletic JI (2011) Using stereotypes to help characterize commits. In: 2011 27th IEEE International Conference on Software Maintenance (ICSM), IEEE, pp 520–523

Dyer R, Nguyen HA, Rajan H, Nguyen TN (2013) Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In: 2013 35th International Conference on Software Engineering (ICSE), IEEE, pp 422–431

Fleiss JL (1971) Measuring nominal scale agreement among many raters. Psychological bulletin 76(5):378

Forte A, Kittur N, Larco V, Zhu H, Bruckman A, Kraut RE (2012) Coordination and beyond: social functions of groups in open content production. In: Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work, pp 417–426

Fu Y, Yan M, Zhang X, Xu L, Yang D, Kymer JD (2015) Automated classification of software change messages by semi-supervised latent dirichlet allocation. Information and Software Technology 57:369–377

Girba T, Kuhn A, Seeberger M, Ducasse S (2005) How developers drive software evolution. In: Eighth international workshop on principles of software evolution (IWPSE'05), IEEE, pp 113–122

Gómez C, Cleary B, Singer L (2013) A study of innovation diffusion through link sharing on stack overflow. In: 2013 10th Working Conference on Mining Software Repositories (MSR), IEEE, pp 81–84

Gousios G (2013) The ghtorent dataset and tool suite. In: 2013 10th Working Conference on Mining Software Repositories (MSR), IEEE, pp 233–236

Hassan AE (2008) The road ahead for mining software repositories. In: 2008 Frontiers of Software Maintenance, IEEE, pp 48–57

Hata H, Treude C, Kula RG, Ishio T (2019) 9.6 million links in source code comments: Purpose, evolution, and decay. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, pp 1211–1221

Hata H, Novielli N, Baltes S, Kula RG, Treude C (2022) Github discussions: An exploratory study of early adoption. Empirical Software Engineering 27(1):1–32

Huang Y, Jia N, Zhou HJ, Chen XP, Zheng ZB, Tang MD (2020) Learning human-written commit messages to document code changes. Journal of Computer Science and Technology 35(6):1258–1277

Kehoe C, Pitkow J, Rogers J (1998) Gvu's ninth www user survey report. Office of

Kittur A, Kraut RE (2010) Beyond wikipedia: coordination and conflict in online production groups. In: Proceedings of the 2010 ACM conference on Computer supported cooperative work, pp 215–224

Krasniqi R, Cleland-Huang J (2020) Enhancing source code refactoring detection with explanations from commit messages. In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, pp 512–516

Krejcie RV, Morgan DW (1970) Determining sample size for research activities. Educational and psychological measurement 30(3):607–610

Liu B, Zhang L, Jiang J, Wang L (2022a) A method for identifying references between projects in github. Science of Computer Programming 222:102858

Liu J, Xia X, Lo D, Zhang H, Zou Y, Hassan AE, Li S (2021) Broken external links on stack overflow. IEEE Transactions on Software Engineering

Liu J, Zhang H, Xia X, Lo D, Zou Y, Hassan AE, Li S (2022b) An exploratory study on the repeatedly shared external links on stack overflow. Empirical Software Engineering 27(1):1–32

Liu S, Gao C, Chen S, Yiu NL, Liu Y (2020) Atom: Commit message generation based on abstract syntax tree and hybrid ranking. IEEE Transactions on Software Engineering

Maalej W, Happel HJ (2009) From work to word: How do software developers describe their work? In: 2009 6th IEEE International Working Conference on Mining Software Repositories, IEEE, pp 121–130

Maalej W, Happel HJ (2010) Can development work describe itself? In: 2010 7th IEEE working conference on mining software repositories (MSR 2010), IEEE, pp 191–200

Mockus A, Votta LG (2000) Identifying reasons for software changes using historic databases. In: icsm, pp 120–130

Moreno L, Bavota G, Di Penta M, Oliveto R, Marcus A, Canfora G (2014) Automatic generation of release notes. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp 484–495

Movshovitz-Attias D, Movshovitz-Attias Y, Steenkiste P, Faloutsos C (2013) Analysis of the reputation system and user contributions on a question answering website: Stackoverflow. In: 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013), IEEE, pp 886–893

Murphy G (2009) Attacking information overload in software development. In: 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), IEEE, pp 4–4

Murphy J, Hashim NH, O'Connor P (2007) Take me back: validating the wayback machine. Journal of Computer-Mediated Communication 13(1):60–75

Nagar Y (2012) What do you think? the structuring of an online community as a collective-sensemaking process. In: Proceedings of the ACM 2012 conference on computer supported cooperative work, pp 393–402

O'mahony S, Ferraro F (2007) The emergence of governance in an open source community. Academy of Management Journal 50(5):1079–1106

Rath M, Rendall J, Guo JL, Cleland-Huang J, Mäder P (2018) Traceability in the wild: automatically augmenting incomplete trace links. In: Proceedings of the 40th International Conference on Software Engineering, pp 834–845

Rebai S, Kessentini M, Alizadeh V, Sghaier OB, Kazman R (2020) Recommending refactorings via commit message analysis. Information and Software Technology 126:106332

Santos EA, Hindle A (2016) Judging a commit by its cover. In: Proceedings of the 13th International Workshop on Mining Software Repositories-MSR, vol 16, pp 504–507

Sarwar MU, Zafar S, Mkaouer MW, Walia GS, Malik MZ (2020) Multi-label classification of commit messages using transfer learning. In: 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), IEEE, pp 37–42

Schermann G, Brandtner M, Panichella S, Leitner P, Gall H (2015) Discovering loners and phantoms in commit and issue data. In: 2015 IEEE 23rd International Conference on Program Comprehension, IEEE, pp 4–14

Sun Y, Wang Q, Yang Y (2017) Frlink: Improving the recovery of missing issue-commit links by revisiting file relevance. Information and Software Technology 84:33–47

Tian Y, Zhang Y, Stol KJ, Jiang L, Liu H (2022) What makes a good commit message? p To be appear

Vasilescu B, Filkov V, Serebrenik A (2013) Stackoverflow and github: Associations between software development and crowdsourced knowledge. In: 2013 International Conference on Social Computing, IEEE, pp 188–195

Vasilescu B, Serebrenik A, Devanbu P, Filkov V (2014) How social q&a sites are changing knowledge sharing in open source software communities. In: Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing, pp 342–354

Viera A, Garrett J (2005) Understanding interobserver agreement: The kappa statistic. Family medicine

Wang D, Xiao T, Thongtanunam P, Kula RG, Matsumoto K (2021) Understanding shared links and their intentions to meet information needs in modern code review. Empirical Software Engineering 26(5):1–32

Wattanakriengkrai S, Chinthanet B, Hata H, Kula RG, Treude C, Guo J, Matsumoto K (2022) Github repositories with links to academic papers: Public access, traceability, and evolution. Journal of Systems and Software 183:111117

Wu J, He H, Xiao W, Gao K, Zhou M (2022) Demystifying software release note issues on github. In: 2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC), pp 602–613, DOI 10.1145/3524610.3527919

Xiao T, Wang D, Mcintosh S, Hata H, Kula RG, Ishio T, Matsumoto K (2021) Characterizing and mitigating self-admitted technical debt in build systems. IEEE Transactions on Software Engineering

Xie R, Chen L, Ye W, Li Z, Hu T, Du D, Zhang S (2019) Deeplink: A code knowledge graph based deep learning approach for issue-commit link recovery. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, pp 434–444

Xiong Y, Meng Z, Shen B, Yin W (2017) Mining developer behavior across github and stackoverflow. In: SEKE, pp 578–583

Ye D, Xing Z, Kapre N (2017) The structure and dynamics of knowledge network in domain-specific q&a sites: a case study of stack overflow. Empirical Software Engineering 22(1):375–406

Zampetti F, Ponzanelli L, Bavota G, Mocci A, Di Penta M, Lanza M (2017) How developers document pull requests with external references. In: 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC), IEEE, pp 23–33

Zhang Y, Yu Y, Wang H, Vasilescu B, Filkov V (2018) Within-ecosystem issue linking: a large-scale study of rails. In: Proceedings of the 7th International Workshop on Software Mining, pp 12–19

Zhang Y, Wu Y, Wang T, Wang H (2020) ilinker: a novel approach for issue knowledge acquisition in github projects. World Wide Web 23(3):1589–1619

Zhou Y, Sharma A (2017) Automated identification of security issues from commit messages and bug reports. In: Proceedings of the 2017 11th joint meeting on foundations of software engineering, pp 914–919