9.6 Million Links in Source Code Comments: Purpose, Evolution, and Decay

Hideaki Hata*, Christoph Treude†, Raula Gaikovina Kula* and Takashi Ishio*
*Nara Institute of Science and Technology
{hata, raula-k, ishio}@is.naist.jp
†University of Adelaide
christoph.treude@adelaide.edu.au

***Abstract*—Links are an essential feature of the World Wide Web, and source code repositories are no exception. However, despite their many undisputed benefits, links can suffer from decay, insufficient versioning, and lack of bidirectional traceability. In this paper, we investigate the role of links contained in source code comments from these perspectives. We conducted a large-scale study of around 9.6 million links to establish their prevalence, and we used a mixed-methods approach to identify the links' targets, purposes, decay, and evolutionary aspects. We found that links are prevalent in source code repositories, that licenses, software homepages, and specifications are common types of link targets, and that links are often included to provide metadata or attribution. Links are rarely updated, but many link targets evolve. Almost 10% of the links included in source code comments are dead. We then submitted a batch of link-fixing pull requests to open source software repositories, resulting in several of our fixes being merged successfully. Our findings indicate that links in source code comments can indeed be fragile, and our work opens up avenues for future work to address these problems.**

I. Introduction and Motivation

When Ted Nelson started Project Xanadu[1] in 1960, he envisioned "an entire form of literature where links do not break as versions change; where documents may be closely compared side by side and closely annotated; where it is possible to see the origins of every quotation; and in which there is a valid copyright system—a literary, legal and business arrangement—for friction-less, non-negotiated quotation at any time and in any amount" [26]. Links were supposed to be visible and could be followed from all endpoints, with permission to link to a document explicitly granted by the act of publication [2]. Decades later, Nelson witnessed the birth of the World Wide Web, which in his words "trivialized this original Xanadu model, vastly but incorrectly simplifying these problems to a world of fragile ever-breaking one-way links, with no recognition of change or copyright, and no support for multiple versions or principled re-use" [26]. As predicted by Nelson, the Internet and its implementation of links have afforded us countless opportunities since, but also experienced issues such as link decay [19], [23], digital plagiarism [9], and the need to rely on external services to keep historical copies of web content [25].

In this work, we investigate the role of links contained in source code comments from the perspective of these opportunities and challenges: what purposes do they serve, how do they and their targets evolve, and how often do they break? Our work is related to and inspired by recent research on source code comments in terms of documentation, traceability, licensing, and attribution. For example, source code comments have been found to document technical debt [29] and to support articulation work [36]. They are fragile with respect to identifier renaming, i.e., traceability between comments and code is easily lost [33]. Source code comments located at the beginning of a file often include a text or a link indicating the copyright and license information of the file [13]. Those comments are updated during the evolution of a product by the copyright holders [42]. Links in source code comments are sometimes used for attribution when source code has been taken from elsewhere—however, the vast majority of code snippets is copied without attribution [7], [8]. Despite these research efforts, to the best of our knowledge, the role of links in source code comments has not been studied comprehensively so far.

To fill this gap, in this paper, we first lay the foundation for understanding the role of links in source code comments by collecting 9,654,702 links from source code comments in 25,925 GitHub repositories. Our parser is able to extract comments from source code written in 7 programming languages. We find that links in source code comments are common: more than 80% of the repositories in our study contained at least one link. Through a qualitative study of a stratified sample of 1,146 links, we establish the kinds of link targets that are referenced in source code comments. To understand how links are used to indicate issues related to attribution, technical debt, copyright, and licensing, our qualitative study also uncovers the various purposes for including links in source code comments. We find that licenses, software homepages, and specifications are among the most prevalent types of link targets, and links are often used to provide metadata or attribution.

Link decay has the potential of making documentation in source code comments fragile and buggy. We investigate this issue from two perspectives: we analyze the evolution of the links in the repositories' commit histories and we examine how often link targets referenced in source code comments change. We find that links are rarely updated, but their targets evolve, in almost 10% of all cases leading to dead links. We then submit fixes to a subset of these broken links as pull

[1] http://www.xanadu.com/

requests, several of which were successfully merged by the maintainers of the corresponding open source projects.

In summary, this paper's contributions are three-fold:

- a large-scale and comprehensive study of around 9.6 million links to establish the prevalence of links in source code comments,
- a mixed-methods study to identify targets, purposes, and evolutionary aspects of links in source code comments, and
- an analysis of the extent to which links in source code comments are affected by link decay, with five of our link-fixing pull requests submitted to open source projects already merged by the projects' maintainers.

II. Research Method

In this section, we present our research questions and data collection methodology, and we introduce the data contained in our online appendix.

A. Research Questions

The main goal of the study is to gain insights into the purposes, evolution and decay of links in source code comments. Based on this goal, we constructed seven research questions to guide our study. We now present each of these questions, along with the motivation for each.

The motivation of **RQ1** is to understand whether the use of links in source code is a common practice in the wild. Furthermore, we would like to quantitatively explore the distribution, diversity, and spread of these links across different types of software projects.

(RQ1): *How prevalent are links in source code comments?*

RQ2 and **RQ3** require a deeper analysis of the repositories, where we would like to understand the nature and purpose that the links serve. The key motivation for **RQ2** is to identify the types of link targets that developers are likely to refer to in source code comments. Furthermore, we would like to characterize the most common types of linked domains. The key motivation for **RQ3** is to determine the reasons why developers use links.

(RQ2): *What kind of link targets are referenced in source code comments?*

(RQ3): *What purpose do links in source code serve?*

RQ4, **RQ5**, and **RQ6** investigate the phenomenon of links in source code comments from an evolutionary and maintenance standpoint. We would like to understand whether developers are updating or maintaining these links after introducing them to the source code, whether the targets evolve, and how many of the links are affected by link decay.

(RQ4): *How do links in source code comments evolve?*

(RQ5): *How frequently do link targets references in source code comments change?*

(RQ6): *How many links in source code comments are dead?*

Finally, our aim for the final research question is two-fold. First, we would like to show that dead links can be restored and fixed. Second, the results can serve as a validation that developers care about the maintenance of links in their code.

(RQ7): *Can we fix dead links in source code comments?*

TABLE I: Collected repositories and links

	# candidate repo	# obtained repo (%)	# links
C	2,771	2,482 (90%)	1,602,156
C++	3,563	3,211 (90%)	1,686,575
Java	4,995	4,472 (90%)	2,925,909
JavaScript	7,130	6,224 (87%)	1,533,219
Python	5,263	4,715 (90%)	412,020
PHP	3,279	2,827 (86%)	1,405525
Ruby	2,233	1,994 (89%)	88,298
sum	**29,234**	**25,925 (89%)**	**9,654,702**

B. Data Collection

We now describe our methods for repository preparation, comment extraction, and link identification.

Repository preparation. In this work, we analyzed *active* software development repositories on GitHub written in *common* programming languages. As common programming languages, we selected seven languages: C, C++, Java, JavaScript, Python, PHP, and Ruby. These languages have been ranked consistently in the top 10 languages on GitHub from 2008 to 2017 (based on the number of repositories from 2008 to 2015 [21], the number of pull requests from 2014 to 2017 [10], and the number of pull requests in 2017 [14]).

Using the GHTorrent dataset[2] [17], we collected active repositories for the seven languages using the following criteria: (i) having more than 500 commits in their entire history (the same threshold used in previous work [4]), and (ii) having at least 100 commits in the most active two years. We designed the second criterion to remove long-term less active repositories and short term projects that have not been maintained for long (and may not be software development projects). For example, we were able to exclude `software-engineering-amsterdam/sea-of-ql`, which is a repository of a collaboration space for students in a particular university course, and was reported as a false positive of software project identification [24]. We determine repositories' languages based on the GHTorrent information. Forked repositories are excluded if repositories are recorded in GHTorrent as forks of other repositories.

With the above criteria, we prepared the candidate list of target repositories for the seven languages as shown in Table I. When we collected these candidate repositories (from May to June 2018), some repositories were not available because they had been deleted or made private. In total, we obtained more than 25,000 repositories, which is almost 90% of the candidate repositories.

Comment extraction. From each Git repository, we extract source files of the labeled language in the `HEAD` commit (the latest snapshot of a cloned repository). For example, only

[2]MySQL database dump 2018-04-01 from http://ghtorrent.org/downloads.html.

`.java` files are extracted from a Java repository. To process source files, we employ ANTLR4 lexical analyzers for six languages other than Ruby because their grammar definitions are available in the official example repository[3]. For Ruby, we use a standard library, Ripper parser.

We extract all single line comments (e.g., `//` in C) and multiline comments (`/* ... */`) according to the grammars. In the case of Python, string literals (`''' ... '''`) are also regarded as comments because they include documents (known as *docstring*). In the case of PHP, both HTML comments and PHP code comments are extracted.

Link identification. From the extracted comments, links are identified using the regular expression `/http\S+/` (*localhost* and IP addresses, which are mainly used for private addresses, are excluded) and validated with the Perl module `Data::Validate::URI`. We identified a total of 9,654,702 links from the collected repositories as seen in Table I. All links are recorded with the information of the corresponding file, repository identifiers (pairs of account and repository names), commit hashes, and the line number where the surrounding comment starts. Considering the number of repositories, we find that repositories written in C, C++, and Java tend to contain more links compared to repositories in Python and Ruby.

C. Online Appendix

Our online appendix contains our 9,654,702 links associated with the information of languages and comment location (GitHub links including account names, repository names, commit hashes, file paths, and line numbers). The appendix is available at https://tinyurl.com/allLinksData.

III. Findings

In this section, we present our findings for each research question.

A. Prevalence of Links (RQ1)

To understand the prevalence of links referenced in source code comments (**RQ1**), we conducted a quantitative analysis of our collected dataset in terms of link existence, domain diversity, and domain popularity.

Link existence. Figure 1a show the percentages of repositories that have at least one link in their source code comments. We see that, in every language, more than 80% of the repositories contain links in source code comments. Especially for repositories written in C, C++, and PHP, more than 90% of the repositories refer to external sources via links.

Domain diversity. In the obtained 9,654,702 links, there are 57,039 distinct domains (Internet hostnames). Figure 1b shows the distribution of the number of distinct domains per repository, for repositories that have at least one link in their source code comments. Median values are presented in the figure. We found that there is a diversity of links in a single repository even when summarized by their domains. Especially in repositories written in C, C++, JavaScript, and PHP, source code comments link to 10 or more different domains (median).

Popular domains. Figure 1c shows the top 10 most referenced domains. The percentages for each language are shown to illustrate the characteristics of referencing links in different languages. Note that this ranking is based on the number of repositories instead of the number of links. If links belonging to a domain appear in a small number of repositories, the domain will be low-ranked even if those repositories contain many links.

The `github.com` domain is the top referenced domain in our dataset. More than 14,000 repositories across seven languages referenced content on `github.com`. As we will describe in detail in Section III-B, such content includes *software homepage*, *code*, and *profile of a GitHub contributor*. However, we find in Section III-F that many links to `github.com` are no longer available. We also found many links to `code.google.com` (7th rank) redirected to `github.com`. In a statistically representative sample of common domains (sampling described in Section III-B), two out of three links to `code.google.com` are redirected to `github.com`, and one links to `code.google.com/archive/`.

The `stackoverflow.com` domain is the second most referenced domain and has been linked to from 8,189 repositories. As identified in previous work, Stack Overflow is widely used as a knowledge exchange platform between programmers [38], where programmers can obtain knowledge of good practices of coding from code examples [30], [35], for example. The large number of links to `stackoverflow.com` in source code comments can be another piece of evidence of developers' needs of knowledge acquisition from external resources. We study how code could be obsolete by not being updated when external sources change in Section III-E.

The top domains differ by programming language: The `www.apache.org` domain is frequently linked from Java repositories, and the `www.gnu.org` domain is referenced from C and C++ repositories. Repositories written in JavaScript have many links to the Web-related domains of `www.w3.org` and `developer.mozilla.org`.

> **Summary**: We revealed that links in source code comments are prevalent. In more than 80% of the 25,925 active repositories written in seven common languages, there exists at least one link in each repository. The top three most frequently referenced domains per repository are `github.com`, `stackoverflow.com`, and `en.wikipedia.com`.

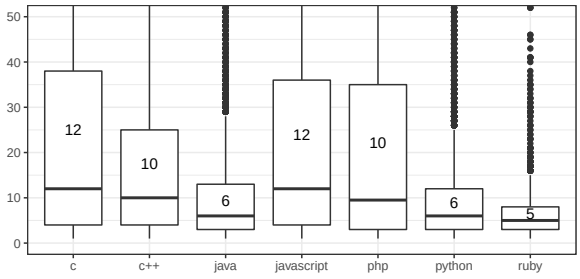
B. Link Targets (RQ2)

To understand what kind of link targets are referenced in source code comments (**RQ2**), we conducted a qualitative study of a statistically representative and stratified sample of all links in our dataset.

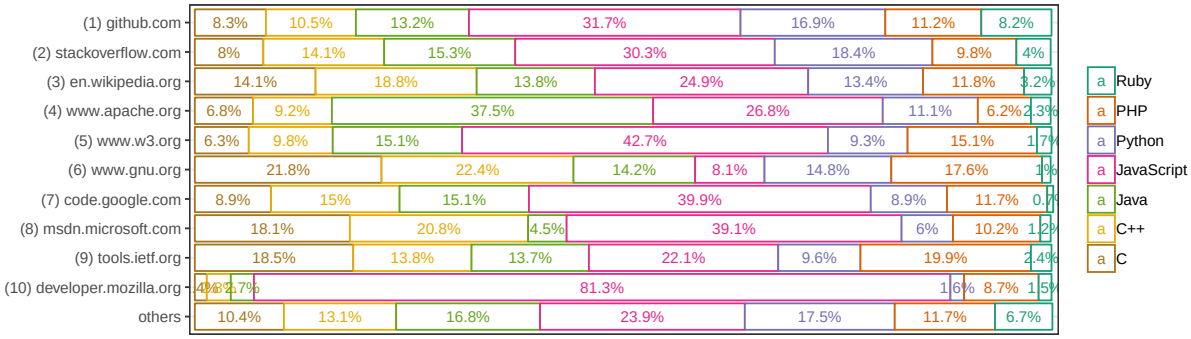
After an initial analysis of the link data, it quickly became obvious that some domains account for many links while other

[3]https://github.com/antlr/grammars-v4/

(a) Ratio of repositories with links



(b) Distribution of number of different domains per repository



(c) Top 10 most referenced domains by language based on the number of repositories.

Fig. 1: Analysis of links by (a) languages, (b) domain diversity, and (c) top domains for RQ1

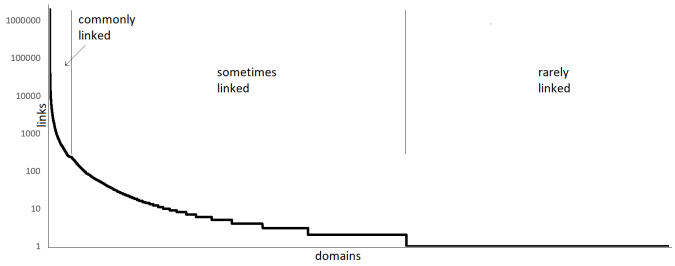


Fig. 2: Distribution of links per domain

TABLE II: Construction of the stratified sample

strata	# domains	# links	# links in sample
common	2,013	9,128,444	384
sometimes	30,851	502,083	384
rare	24,175	24,175	378
sum	**57,039**	**9,654,702**	**1,146**

domains are rare. Based on this observation and to ensure diversity of our sample, 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 those that are commonly, sometimes, and rarely linked, we conducted a visual analysis of the distribution of links per domain in our dataset. Figure 2 shows this distribution using a log scale. While content from the most commonly linked domain was linked more than a million times, many domains appeared in our dataset with a much lower frequency. We used the "step" in the distribution on the left-hand side of Figure 2 to distinguish between domains that are commonly linked and domains that are sometimes linked, with a cutoff frequency of 230. We consider domains which account for exactly one link in our dataset to be rarely linked. Table II shows the number of domains and the number of links in each strata. We then drew a statistically representative sample from each bucket. The required sample size was calculated so that our conclusions about the ratio of links with a specific characteristic would generalize to all links in the same bucket with a confidence level of 95% and a confidence interval of 5.[4]

The qualitative analysis was conducted in multiple iterations: in the first iteration, the first two authors independently coded 20 links from the sample, discussed a common coding guide, and tested this coding guide on another 20 links from the sample, refining the guide, merging codes, and adding codes which had been missed. The initial codes were informed by those used by Aniche et al. [5] to categorize content posted on news aggregators, however, we found that their codes did not cover all types of link targets present in our dataset. In the second iteration, the four authors of this paper then independently coded another 30 links from the sample, using the coding guide designed by the first two authors. We then calculated the kappa agreement of this iteration between all

[4]https://www.surveysystem.com/sscalc.htm

four raters, for 30 cases and all 19 codes that emerged from the qualitative analysis.[5] The kappa agreement was 0.81 or "almost perfect" [39]. Based on this encouraging result, the remaining data was then coded by a single author.

The following list shows the 19 codes that emerged from our analysis along with a short description which was available in the coding guide:

- *404*: link target does not exist (anymore) or cannot be accessed
- *licence*: licence of a software project
- *software homepage*: main web presence of a library or software project
- *specification*: anything that resembles a requirements document or a technical standard
- *organization homepage*: main web presence of an organization or company
- *other*: anything that does not fit the other codes, including if sign-in is required
- *tutorial or article*: technical article or tutorial, without commenting section (blog post otherwise)
- *API documentation*: documentation of an API element
- *blog post*: technical content with a commenting section
- *application*: interactive application (e.g., web application, online utility)
- *bug report*: bug report or issue in an online bug/issue tracker
- *research paper*: academic paper
- *personal homepage*: personal homepage of one individual
- *code*: a source code file
- *forum thread*: thread in a forum or entire forum
- *GitHub profile*: profile of a GitHub contributor
- *book content*: chapter/section of a book or entire book
- *Q&A thread*: question-and-answer thread, but not Stack Overflow
- *Stack Overflow*: question-and-answer thread on Stack Overflow

Taxonomy of link targets. Table III shows the result of our qualitative analysis. For commonly-linked domains, license is the most common type of link target, accounting for more than half of the links in our sample, followed by software homepages, i.e., the main web presence of a library or software project. For domains that are linked sometimes from source code comments, the most common type of link target was 404, a non-existing link target. This is a first indicator of the decay of links in source code comments, which we will analyze in detail in the next sections. Software homepages are also prevalent, as are organization homepages, both accounting for more than 10% of all links in our sample. Finally, for links from domains which are rarely linked, the problem of decay is even more serious, affecting 37% of the links in this sample. In other words, we can conclude with a 95% confidence that between 32 and 42% of all links to domains which are rarely linked from source code comments are dead or inaccessible. The prevalence of the code "other" in the results for links to rarely linked domains is an indicator of the diversity of links present in source code comments.

[5]Kappa agreement was calculated using http://justusrandolph.net/kappa/.

TABLE III: Frequency of link target types in our sample

	common	**sometimes**	**rare**
404	27 (7%)	122 (32%)	138 (37%)
license	208 (54%)	4 (1%)	1 (0%)
software homepage	55 (14%)	65 (17%)	28 (7%)
specification	21 (5%)	33 (9%)	32 (8%)
organization homepage	16 (4%)	41 (11%)	24 (6%)
other	5 (1%)	23 (6%)	45 (12%)
tutorial or article	16 (4%)	21 (5%)	31 (8%)
API documentation	14 (4%)	20 (5%)	10 (3%)
blog post	1 (0%)	10 (3%)	22 (6%)
application	0 (0%)	11 (3%)	13 (3%)
bug report	9 (2%)	10 (3%)	3 (1%)
research paper	0 (0%)	9 (2%)	13 (3%)
personal homepage	4 (1%)	8 (2%)	4 (1%)
code	6 (2%)	2 (1%)	5 (1%)
forum thread	0 (0%)	5 (1%)	6 (2%)
GitHub profile	1 (0%)	0 (0%)	0 (0%)
book content	0 (0%)	0 (0%)	2 (1%)
Q&A thread	0 (0%)	0 (0%)	1 (0%)
Stack Overflow thread	1 (0%)	0 (0%)	0 (0%)
sum	**384(100%)**	**384(100%)**	**378(100%)**

Summary: We identified more than a dozen different kinds of link targets, with dead links, licenses, and software homepages being the most prevalent. Dead links are particularly common for rarely linked domains.

C. Link Purpose (RQ3)

To understand the purpose of links referenced in source code comments (**RQ3**) and similar to (**RQ2**), we again employed a qualitative analysis of our statistically representative and stratified sample of 1,146 links, only this time focusing on the origin of a link (in a source comment) rather than the target of the link. We used the same iterative approach to design a coding guide, and validated the coding guide by having the first four authors code 30 links independently, this time leading to a kappa agreement of 0.70 which indicates "substantial" agreement [39]. The somewhat lower agreement can be explained by the need to extrapolate the purpose of a link from its context in the source code alone, without being able to interview the contributor who added the link.

The following list shows all 8 codes that emerged from our analysis for link purpose, along with a short description which was available in the coding guide. The coding guide was informed by work on source code comments (e.g., [36]), self-admitted technical debt (e.g., [29]), and attribution (e.g., [7]).

- *link-only*: the comment only contains the link
- *self-admitted technical debt*: bug-related, like workaround, under development, and so on
- *metadata*: the link relates to the author of the source, a related organization, or the license

TABLE IV: Frequency of link purposes in our sample

	common	sometimes	rare
metadata	288 (75%)	131 (34%)	43 (11%)
source/attribution	27 (7%)	62 (16%)	75 (20%)
source code context	18 (5%)	60 (16%)	80 (21%)
see-also	28 (7%)	59 (15%)	51 (13%)
commented-out source code	1 (0%)	17 (4%)	70 (19%)
link-only	6 (2%)	24 (6%)	40 (11%)
self-admitted technical debt	11 (3%)	16 (4%)	13 (3%)
@see	5 (1%)	15 (4%)	6 (2%)
sum	**384(100%)**	**384(100%)**	**378(100%)**

- *commented-out source code*: the link is part of the source code, e.g., as a parameter value, but has been commented out
- *source/attribution*: the comment explicitly indicates that the link is a source of some aspect of the source code (e.g., algorithm)
- *see-also*: the comment explicitly indicates that the link points to additional reading material (usually accompanied by a phrase such as "see also").
- *@see*: the link is accompanied by "`@see`", but no further explanation
- *source code context*: the link adds additional information to the source code (use this code for things that do not obviously fit into any of the previous)

Note that our coding guide required the indicators of *see-also* and *source/attribution* to be explicit, thus reducing the guesswork required as part of the qualitative analysis.

Taxonomy of link purpose. Table IV shows the results of the qualitative analysis. For links to commonly linked domains, providing metadata, e.g., in the forum of licenses or author information, is by far the most common purpose of a link, covering three quarters of the links in our sample. For links to domains which are only sometimes linked, metadata only accounts for one third of the data, followed by links included for the purpose of attribution, providing context, or see-also information. The results for links to rarely linked domains are even more diverse: we can see from the table that these links are used for context, attribution, and as part of the source code functionality (albeit commented out), to name the top three. Six of the eight codes account for at least 10% of the links in this part of our sample.

Matching link target with purpose. Based on the qualitative analysis conducted for answering **RQ2** and **RQ3** about the targets and purposes of links in source code comments, we are now able to 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 [1] 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 4 as threshold for support and 0.7 as threshold for confidence, i.e., all 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.

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

TABLE V: Associations between link target type and link purpose

strata	association rule			conf.	supp.
common	license	⇒	metadata	1.00	208
common	metadata	⇒	license	0.72	208
common	software homepage	⇒	metadata	0.75	41
common	organization homepage	⇒	metadata	0.88	14
common	bug report	⇒	satd	0.78	7
common	satd	⇒	bug report	0.64	7
common	personal homepage	⇒	metadata	1.00	4
sometimes	software homepage	⇒	metadata	0.65	42
sometimes	organization homepage	⇒	metadata	0.80	33
sometimes	personal homepage	⇒	metadata	1.00	8
sometimes	license	⇒	metadata	1.00	4
rare	personal homepage	⇒	metadata	1.00	4

Table V shows the association rules extracted from our data with these settings, separately for each stratum in our sample. Unsurprisingly, the link target type *license* and the purpose of providing *metadata* are tightly connected, in particular for links referring to commonly linked domains. In fact, all links to licenses were found to have been included for the reason of providing metadata, and 72% of the metadata is license information. Links to software, organization, and personal homepages are also associated with metadata, across all strata. Although with a relatively low support of seven instances, it is also interesting to note the tight coupling of the link target type *bug report* and the purpose of admitting technical debt.

> **Summary**: We identified different purposes for the inclusion of links in source code comments, with providing metadata and attribution being the most common. Links are also included for background information, to provide context, or to admit technical debt. In some cases, the link is part of source code which has been commented out.

D. Link Evolution (RQ4)

To understand how links evolve (**RQ4**), we investigated the revision history of repositories using the samples from (**RQ2**). For each sample link, we searched an old version of the link that has been revised by a commit that introduced the link. We extracted such a commit introducing a link by using the git log command (`-S` option with tracking file renaming). We searched http(s) links removed from the code location where the sample link has been added. We identified 88 revised links out of 1,146 samples, including 24 (6.3%) in common, 31 (8.1%) in sometimes, and 33 (8.7%) in rare.

We manually analyzed the old and new paths of the links and identified the following evolution types:

- *license*: A new link refers to a new software license. For example, a link to GNU GPL has been replaced with a link to the Apache License.

TABLE VI: Link Evolution Types

	common		sometimes		rare	
license	12	(50%)	1	(3%)	1	(3%)
organization	7	(29%)	10	(32%)	3	(9%)
https	2	(8%)	1	(3%)	5	(15%)
moved content	0	(0%)	2	(6%)	6	(18%)
updated content	1	(4%)	3	(10%)	4	(12%)
relevant content	0	(0%)	6	(19%)	2	(6%)
other	2	(8%)	8	(26%)	12	(36%)
Total	24	(100%)	31	(100%)	33	(100%)

- *organization*: A project or an organization changed its name or website. For example, a project that acquired their own domain updated links to their project website.
- *https*: A new link uses HTTPS instead of HTTP for the same location as the previous link.
- *moved content*: A new link refers to a slightly different location (e.g. the same path on a different server, the same document name on a different wiki), which is likely the same content.
- *updated content*: A new link refers to different content from the previous link, but the new content is likely updated. For example, the Apache Jackrabbit project replaced a link pointing to a draft version of a document[7] with a link to an RFC version[8].
- *relevant content*: A new link refers to relevant but different content from the previous link. For example, the Pi4J project replaced a link related to the usage of a serial port of Raspberry Pie[9] with another similar document[10].
- *other*: We could not identify types for some links whose contents are no longer available. It should be noted that the contents for 20 updated links are 404 Not Found.

Reasons for link evolution. Table VI shows the numbers of link evolution in the three strata.

> **Summary**: Links are rarely updated (less than 9%). Common modifications are updating licenses and organization homepages.

E. Link Target Evolution (RQ5)

After understanding the evolution of links, our next research question **(RQ5)** asks about the evolution of their targets. To investigate whether link targets referenced in source code comments evolve, we attempted to download all link targets in our sample of 1,146 links using the `curl` command with a timeout of 60 seconds. As already discussed as part of **(RQ2)**, not all link targets are available. We were able to download a total of 1,034 link targets (90%). We then repeated the same download process exactly ten days later, to see how many of the link targets had changed within this short time frame and what kind of changes had happened.

TABLE VII: Link target evolution within a 10-day timeframe

evolution	# of links
no content returned on both dates	112
no changes between the two dates	879
content not available anymore	6
auto-generated changes (e.g., # of visitors)	125
content changed	7
"latest" updated	7
design change (e.g., new banner added)	6
different error (e.g., 502 ⇒ 301)	3
page title changed	1
sum	1,146

Changes to the link target. Table VII summarizes the results of this analysis: out of the 1,034 link targets for which `curl` returned a result, 879 (85%) had not changed at all in the ten-day time frame. We manually analyzed the 155 cases in which the content had changed by opening both versions in a web browser and conducting a visual comparison. The majority of the changes in the remaining 15% can be attributed to automatically generated changes, such as the display of a visitor count or the current date in a footer.

However, a non-negligible number of link targets underwent more significant changes in the ten-day time window: For six links for which we were able to retrieve data on the first download date, there was no content available anymore ten days later. For three links which had displayed an error message when we first attempted to download their content, the specific error message changed. Some link targets changed their website design, and for a few links, the content changed. For example, the download page of TaskWarrior[11] included the following notice when we first downloaded its content: "(For those of you wishing to build task from source on Cygwin, you will need some components installed (make, g++/clang, GnuTLS, libuuid, libreadline), but don't forget - task is a standard part of the Cygwin distribution, so you do not need to build from source, unless you want the latest development snapshot)." Ten days later, this notice was replaced with: "(Please note, that Cygwin is not supported anymore. Please use the Windows Subsystem for Linux to use Taskwarrior on Windows)." We argue that this kind of change is relevant to software developers.

Stack Overflow case study. To investigate this phenomenon in more detail, we conducted a case study with the subset of links pointing to Stack Overflow. As seen in Section III-A, `stackoverflow.com` is the second most referenced domain.

In all 9,654,702 obtained links, there are 32,197 links belonging to `stackoverflow.com`. Among those Stack Overflow links, there are varieties of expressions: an abbreviated path to an answer (`/a/(answer id)`), an abbreviated path to a question (`/q/(question id)`), and a full path to

[7] http://greenbytes.de/tech/webdav/draft-ietf-webdav-bind-21.html
[8] http://greenbytes.de/tech/webdav/rfc5842.html
[9] http://www.irrational.net/2012/04/19/using-the-raspberry-pis-serial-port/
[10] https://www.cube-controls.com/2015/11/02/disable-serial-port-terminal-output-on-raspbian/
[11] https://taskwarrior.org/download/

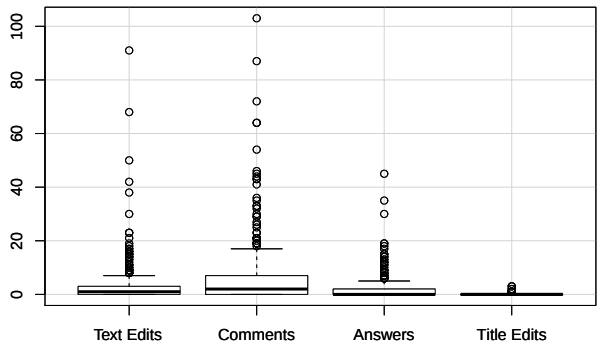


Fig. 3: Number of changes to the linked Stack Overflow threads after they were first referenced

a question (`/questions/(question id)/(title)`). Older links start with '`http://`' and newer links start with '`https://`'. Despite such varieties, we found duplicate link expressions, which are mainly because of the same developers' commenting of the same links in the same repositories or file reusing across different repositories, which is known as *clone-and-own* [12], [18]. For each Stack Overflow link, we identified the timestamp of when the link was added to a repository by using the same git log command (`-S` option with tracking file renaming) used in Section III-D. For duplicate links, we consider only the oldest timestramp. Consequently, we obtained a list of 11,464 distinct links with their timestamps.

We then made use of the SOTorrent dataset [8] to investigate the extent to which Stack Overflow content had changed since the link to the question or answer had been added to a source code comment on GitHub. We created a statistically representative sample of 372 links from the population of all unique links to Stack Overflow content in our dataset, and 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.

Thread updates. Figure 3 shows the results of this analysis. More than half of all Stack Overflow threads had at least one change made to the text of a question or answer in the same thread (median: 1, third quartile: 3) after they were committed to a GitHub repository as part of a source code comment, and more than half of these links attracted at least one new comment in the meantime (median: 2, third quartile: 7). While the number of new answers to a thread was zero in the median case, a quarter of the Stack Overflow threads attracted at least 2 new answers after the link was added in a source code comment (median: 0, third quartile: 2). In total, only 91 (24%) of the 372 Stack Overflow threads in our sample did not undergo any changes after they were added to a GitHub repository.

The most extreme example in our sample—a Stack Overflow thread titled "How do you split a list into evenly sized chunks?"[12]—had 45 new answers added (out of a total of 58) and attracted 103 new comments, 50 edits, and one change to the thread title after being first referenced in a source code comment in early September 2010.

> **Summary**: We find that even within a short ten-day time window, a non-negligible portion of link targets referenced in source code comments evolve, in some cases adding or modifying pertinent information. In our case study on links pointing to Stack Overflow, we find that more than three quarters of all Stack Overflow threads linked in source code comments attracted at least one change (edit, new answer, or new comment) after being first referenced in a source code comment.

F. Link Decay (RQ6)

To investigate the amount of dead links in source code comments (**RQ6**), we accessed all Web contents from the 382,650 unique links by using the Perl module `LWP`[13]. We found that among the obtained 9,654,702 links, there are 382,650 distinct links.

To answer our sixth research question **(RQ6):** *How many links in source code comments are dead?*, we investigate the number of links that returned an unsuccessful web retrieval of the web content (i.e., 404 HTTP status).

Link retrieval responses. Table VIII shows obtained HTTP Status Codes for all links. More than 80% of the links were successfully reachable including redirecting to new locations. Although some HTTP status codes do not clearly mean links are dead, we found that about 9% of the links are not available now (404 Not Found). Among these 404 links, we saw some patterns: Internet host lost, Web content move/deletion, fake URL (for showing examples of parameters or data structures), and link typo. The domain with the largest number of 404s is `github.com`, with 3,346 links no longer available.

TABLE VIII: HTTP Status Codes for All Unique Links

status	**# links**	**(%)**
2xx success	310,592	(81.2%)
404 not found	34,689	(9.1%)
500 internal server error	18,955	(5.0%)
405 method not allowed	10,104	(2,6%)
403 forbidden	4,068	(1.0%)
others	4,242	(1.1%)
sum	**382,650**	**100%**

[12]https://stackoverflow.com/q/312443

[13]We used `LWP::UserAgent` and `LWP::RobotUA`.

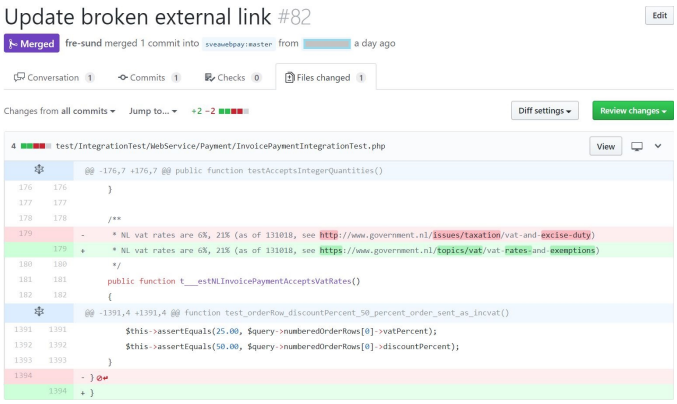


Fig. 4: Example of a pull request sent to sample of projects with dead links that are fixable.

Summary: We find that 9.1% of links are not available (404 Not Found) in all unique links.

G. Fixing Dead Links (RQ7)

To fix dead links (**RQ7**), we collected fixable dead links and submitted pull requests to fix. We select dead links that are not metadata (need multiple files to be fixed) nor commented-out source code. Personal blog articles were avoided because they tend to be no longer available. Consequently we obtained dead links of API documentation, research papers, and so on. After checking the original contents in `Wayback Machine`[14], we manually investigated new links by searching specific keywords in the original contents.

Figure 4 shown an example of a selected project that we submitted a pull request for developers to review. As such, we sent a total of 14 pull requests to projects, requesting to fix broken links. Our fixing process included first forking a personal copy of the project, fixing the link, and then later submission of a pull request to the project.

To answer our last research question **(RQ7):** *Can we fix dead links in source code comments?*, we created pull requests to fix real-world projects, with special attention to their comments as well as reaction to these updates.

Developer Comments. We found that developers were positive, with six projects responsive. Five of these projects readily accepted our fixes (two projects still pending tests before merging). Interestingly, there is a case where developers react positively to our fixes, even if a security issue caused the link to be broken, showing that they are adamant to keep the link alive:

> *"Well, the original link still works, there's just a certificate issue on the server side (old SHA-1 certificate). Not sure how optimistic I am that it will ever be fixed, so we can probably just fall back to http as suggested."*

[14]https://web.archive.org/

Since the link itself is a comment, we speculate that it has almost no conflicts to existing code, so our pull requests to pass all tests and be merged immediately. In other cases, developers responded with *"LGTM (looks good to me)"* and *"Thanks for spotting the broken link"*. However, there was one case where the developer kindly rejected our pull request as the code itself was regarded obsolete:

> *"Thanks, but this is just a automated downstream mirror of OpenBSD which includes an ancient gcc (4.2.1). So, won't fix.."*

Overall, the responses from developers provide sufficient motivation for tool support to assist with fixing any broken links. We argue that such comments indicate that developers are concerned with keeping their links alive.

Summary: Developers generally responded positively to the request to fix the dead links. Five out of six responsive projects accepted our pull requests (i.e., two merge pending) to fix dead links.

IV. Recommendations

Our findings can be summarized into recommendations for developers and researchers.

Recommendations for software developers referencing links in source code comments are:

- *Try referencing permanent links*, as it is reported that more than 30% of links will not work after a 4 years period [20]. Referencing research papers with DOI is preferable instead of researchers' personal Web pages. Explicitly mentioning tags or commit hashes to referencing code in GitHub would be recommended, as software structure can be changed (we found many dead links to GitHub in Section III-F).
- *Check link targets for new information on a regular basis*, as referencing external resources can be considered to be software documentation to support comprehension and maintenance activities. In addition, link target updates can be triggers of improving and updating code (as seen in Section III-E).

We can also consider future work with the following possible challenges.

- *Further studies of source code comments* to understand how knowledge (related to knowledge-based theory of the firm [44] and human capital [27], [40]) is summarized and shared via source code comments. Further analyses of source code comment contents [28] would be required.
- *Further understanding of external sources*. We found several sources as shown in Figure 1c and Table III. Although some sources have been already studied, for example, licenses [13], self-admitted technical debt [29], and Stack Overflow [38], other sources have not been well-studied with regard to their impact and influence on software development, such as research papers and Wikipedia articles.

- *Tool support for external source referencing, tracking, and updating*. Although we recommend developers to maintain links and associated code, it is not always possible. Tools or systems to help developers fix link issues and maintain code, and some automated support could be practically useful.

V. Threats to Validity

Threats to the *construct validity* exist in our approach to link identification. Since we identified links per line in source code comments, links located across multiple lines cannot be extracted. Note that we did not encounter any such multiple-line links in our representative sample of 1,146 links. Hence we consider that the impact of incorrect link identification because of multiple-line links is small.

Threats to the *external validity* exist in our repository preparation. Although we analyzed a large amount of repositories on GitHub, we cannot generalize our findings to industry nor open source projects in general; some of open source repositories are hosted outside of GitHub, e.g., on GitLab or private servers.

To mitigate threats to *reliability*, we prepared an online appendix of our 9,654,702 links with associated information (see Section II-C).

VI. Related Work

We have discussed complementary work throughout the paper in the relevant sections; here, we discuss literature related to source code comments.

One of the most related studies is that by Xia et al. [43]. They investigated what developers search for on the Web, and found that developers search for explanations on unknown terminologies, explanations for exceptions/error messages (e.g., HTTP 404), reusable code snippets, solutions to common programming bugs, and suitable third-party libraries/services. Furthermore, they found that searching for solutions to performance bugs, solutions to multi-threading bugs, public datasets to test newly developed algorithms or systems, reusable code snippets, best industrial practices, database optimization solutions, solutions to security bugs, and solutions to software configuration bugs are the most difficult search tasks that developers consider.

Many researchers have made use of code comments in their work. Tan et al. [37] automatically identify bugs by analyzing inconsistencies between code and comments. Ratol and Robillard [33] used code comments to assist refactoring activities. Wong et al. [41] used code comments to map between source code and Stack Overflow. German et al. [13] developed the ninka tool that automatically identifies a software license in code comments. Goldman and Miller [15] developed the tool CodeTrail, that demonstrates how the developer's use of web resources can be improved by connecting the Eclipse integrated development environment (IDE) and the Firefox web browser.

Self-admitted technical debt is a commenting activity that has been well-studied in recent years [29]. Maldonado et al. [22] and Zampetti et al. [45] studied the removal of self-admitted technical debt based on the modification of comments. Our finding of referencing bug reports for self-admitted technical debt could be another opportunity to study development activities around technical debt.

There are also studies which analyze link sharing occurring in other software artifacts. Gomez et al. [16] investigated link sharing on Stack Overflow to gain insights into how software developers discover and disseminate innovations. Rath et al. [32] investigated links to issue tracking systems in commit comments. They reported that developers often do not provide external links to issues. They evaluated several methods to automatically recover links by searching issues related to a given commit. Alqahtani et al. [3] proposed a tool to automatically link dependent components in a system to online resources for analyzing their vulnerabilities. Chen et al. [11] proposed a tool to link problematic source code to relevant Stack Overflow questions using similarity of source code fragments.

Traceability links between source code and documents is another related research topic. Scanniello et al. [34] reported that developers can understand source code effectively if design models include source code element names. Antoniol et al. [6] proposed a tool to identify links between source files and design documents. It supports to keep them consistent because developers may update source file names without updating related documents. Rahimi et al. [31] proposed a tool to update links between source files and requirements documents according to semantic differences of source code.

However, none of the related work provides a comprehensive study of the role of links in source code comments, which is the goal of this paper.

VII. Conclusion

To understand purposes, evolution, and decay of links in source code comments, we conducted (i) a quantitative study of 9,654,702 links from source code comments in 25,925 GitHub repositories to establish the prevalence of links in source code comments; (ii) a qualitative study of a stratified sample of 1,146 links to determine the kinds of link targets and purposes for including links present in our dataset; (iii) a quantitative and qualitative study to investigate the evolution of links in source code comments and their targets; and (iv) a quantitative study to determine the extent to which links in source code comments are affected by link decay.

Our work has shown that links in source code comments indeed suffer from decay, from insufficient versioning (when link targets evolve), and from lack of bidirectional traceability (which could help avoid decay). Based on this work which has established the prevalence of links in source code comments, their multiple purposes and targets, issues of decay, and practical needs of fixing dead links, there are many open avenues for future work: further studies of source code comments, understanding the role of external sources for software development, and tool support for external source referencing, to name a few.

REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of 20th International Conference on Very Large Data Bases (VLDB 1994)*, pages 487–499, 1994.

[2] M. Aiello. *The Web Was Done by Amateurs: A Reflection on One of the Largest Collective Systems Ever Engineered*. Springer, 2018.

[3] S. S. Alqahtani, E. E. Eghan, and J. Rilling. Recovering semantic traceability links between APIs and security vulnerabilities: An ontological modeling approach. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, mar 2017.

[4] M. Aniche, G. Bavota, C. Treude, M. A. Gerosa, and A. Deursen. Code smells for model-view-controller architectures. *Empirical Softw. Engg.*, 23(4):2121–2157, Aug. 2018.

[5] M. Aniche, C. Treude, I. Steinmacher, I. Wiese, G. Pinto, M.-A. Storey, and M. A. Gerosa. How modern news aggregators help development communities shape and share knowledge. In *Proceedings of the 40th International Conference on Software Engineering*, pages 499–510, 2018.

[6] G. Antoniol, B. Caprile, A. Potrich, and P. Tonella. Design-code traceability for object-oriented systems. *Annals of Software Engineering*, 9(1-4):35–58, Jan. 2000.

[7] S. Baltes and S. Diehl. Usage and attribution of stack overflow code snippets in github projects. *Empirical Softw. Engg.*, pages 1–37, 2018.

[8] S. Baltes, L. Dumani, C. Treude, and S. Diehl. Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts. In *Proceedings of the International Conference on Mining Software Repositories*, pages 319–330, 2018.

[9] J. M. Barrie and D. E. Presti. Digital plagiarism-the web giveth and the web shall taketh. *Journal of medical Internet research*, 2(1), 2000.

[10] F. Beuke. Github Language Statistics – GitHut 2.0. https://madnight.github.io/githut/. [Online; accessed Aug 2018].

[11] F. Chen and S. Kim. Crowd debugging. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. ACM Press, 2015.

[12] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki. An exploratory study of cloning in industrial software product lines. In *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, CSMR '13, pages 25–34, Washington, DC, USA, 2013. IEEE Computer Society.

[13] D. M. German, Y. Manabe, and K. Inoue. A sentence-matching method for automatic license identification of source code files. In *Proceedings of the 25th IEEE/ACM international conference on Automated software engineering*, pages 437–446. ACM, 2010.

[14] GitHub. The State of the Octoverse 2017. https://octoverse.github.com/, 2017. [Online; accessed Aug 2018].

[15] M. Goldman and R. C. Miller. Codetrail: Connecting source code and web resources. *Journal of Visual Languages & Computing*, 20(4):223 – 235, 2009.

[16] C. Gómez, B. Cleary, and L. Singer. A study of innovation diffusion through link sharing on stack overflow. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 81–84, Piscataway, NJ, USA, 2013. IEEE Press.

[17] G. Gousios. The ghtorent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[18] T. Ishio, Y. Sakaguchi, K. Ito, and K. Inoue. Source file set search for clone-and-own reuse analysis. In *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR '17, pages 257–268, Piscataway, NJ, USA, 2017. IEEE Press.

[19] M. Klein, H. Van de Sompel, R. Sanderson, H. Shankar, L. Balakireva, K. Zhou, and R. Tobin. Scholarly context not found: one in five articles suffers from reference rot. *PloS one*, 9(12):e115253, 2014.

[20] W. Koehler. Web page change and persistence—a four-year longitudinal study. *J. Am. Soc. Inf. Sci. Technol.*, 53(2):162–171, Jan. 2002.

[21] A. La. Language Trends on GitHub – The GitHub Blog. https://blog.github.com/2015-08-19-language-trends-on-github/, 2015. [Online; accessed Aug 2018].

[22] E. D. S. Maldonado, R. Abdalkareem, E. Shihab, and A. Serebrenik. An empirical study on the removal of self-admitted technical debt. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 238–248, Sept 2017.

[23] J. Markwell and D. W. Brooks. Broken links: The ephemeral nature of educational www hyperlinks. *Journal of Science Education and Technology*, 11(2):105–108, 2002.

[24] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan. Curating github for engineered software projects. *Empirical Softw. Engg.*, 22(6):3219–3253, Dec. 2017.

[25] J. Murphy, N. H. Hashim, and P. OConnor. Take me back: validating the wayback machine. *Journal of Computer-Mediated Communication*, 13(1):60–75, 2007.

[26] T. H. Nelson. Xanalogical structure, needed now more than ever: parallel documents, deep links to content, deep versioning, and deep re-use. *ACM Computing Surveys (CSUR)*, 31(4es):33, 1999.

[27] S. Onoue, H. Hata, R. G. Kula, and K. Matsumoto. Human capital in software engineering: A systematic mapping of reconceptualized human aspect studies. *CoRR*, abs/1805.03844, 2018.

[28] L. Pascarella and A. Bacchelli. Classifying code comments in java open-source software systems. In *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR '17, pages 227–237, Piscataway, NJ, USA, 2017. IEEE Press.

[29] A. Potdar and E. Shihab. An exploratory study on self-admitted technical debt. In *Proceedings for the International Conference on Software Maintenance and Evolution*, pages 91–100. IEEE, 2014.

[30] S. Radevski, H. Hata, and K. Matsumoto. Towards building api usage example metrics. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 619–623, March 2016.

[31] M. Rahimi and J. Cleland-Huang. Evolving software trace links between requirements and source code. *Empirical Software Engineering*, nov 2017.

[32] M. Rath, J. Rendall, J. L. C. Guo, J. Cleland-Huang, and P. Mder. Traceability in the wild. In *Proceedings of the 40th International Conference on Software Engineering - ICSE '18*. ACM Press, 2018.

[33] I. K. Ratol and M. P. Robillard. Detecting fragile comments. In *Proceedings of the International Conference on Automated Software Engineering*, pages 112–122, 2017.

[34] G. Scanniello, C. Gravino, M. Genero, J. A. Cruz-Lemus, G. Tortora, M. Risi, and G. Dodero. Do software models based on the UML aid in source-code comprehensibility? aggregating evidence from 12 controlled experiments. *Empirical Software Engineering*, feb 2018.

[35] J. Sillito, F. Maurer, S. M. Nasehi, and C. Burns. What makes a good code example?: A study of programming q&a in stackoverflow. In *Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM)*, ICSM '12, pages 25–34, Washington, DC, USA, 2012. IEEE Computer Society.

[36] M.-A. Storey, J. Ryall, R. I. Bull, D. Myers, and J. Singer. Todo or to bug. In *Proceedings of the International Conference on Software Engineering*, pages 251–260. IEEE, 2008.

[37] L. Tan, D. Yuan, G. Krishna, and Y. Zhou. /*icomment: Bugs or bad comments?*/. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 145–158, New York, NY, USA, 2007. ACM.

[38] C. Treude, O. Barzilay, and M.-A. Storey. How do programmers ask and answer questions on the web? (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 804–807, New York, NY, USA, 2011. ACM.

[39] A. J. Viera, J. M. Garrett, et al. Understanding interobserver agreement: the kappa statistic. *Fam Med*, 37(5):360–363, 2005.

[40] C. Wohlin, D. mite, and N. B. Moe. A general theory of software engineering: Balancing human, social and organizational capitals. *Journal of Systems and Software*, 109:229 – 242, 2015.

[41] E. Wong, J. Yang, and L. Tan. Autocomment: Mining question and answer sites for automatic comment generation. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ASE'13, pages 562–567, Piscataway, NJ, USA, 2013. IEEE Press.

[42] Y. Wu, Y. Manabe, T. Kanda, D. M. German, and K. Inoue. Analysis of license inconsistency in large collections of open source projects. *Empirical Software Engineering*, 22(3):1194–1222, dec 2016.

[43] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing. What do developers search for on the web? *Empirical Software Engineering*, 22(6):3149–3185, Dec 2017.

[44] M. Zahedi and M. A. Babar. Why does site visit matter in global software development: A knowledge-based perspective. *Information and Software Technology*, 80:36 – 56, 2016.

[45] F. Zampetti, A. Serebrenik, and M. Di Penta. Was self-admitted technical debt removal a real removal?: An in-depth perspective. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, pages 526–536, New York, NY, USA, 2018. ACM.