SOTorrent: Reconstructing and Analyzing the Evolution of Stack Overflow Posts

Sebastian Baltes
Lorik Dumani
research@sbaltes.com
dumani@uni-trier.de
University of Trier, Germany

Christoph Treude
christoph.treude@adelaide.edu.au
University of Adelaide, Australia

Stephan Diehl
diehl@uni-trier.de
University of Trier, Germany

ABSTRACT

Stack Overflow (SO) is the most popular question-and-answer website for software developers, providing a large amount of code snippets and free-form text on a wide variety of topics. Like other software artifacts, questions and answers on SO evolve over time, for example when bugs in code snippets are fixed, code is updated to work with a more recent library version, or text surrounding a code snippet is edited for clarity. To be able to analyze how content on SO evolves, we built *SOTorrent*, an open dataset based on the official SO data dump. *SOTorrent* provides access to the version history of SO content at the level of whole posts and individual text or code blocks. It connects SO posts to other platforms by aggregating URLs from text blocks and by collecting references from GitHub files to SO posts. In this paper, we describe how we built *SOTorrent*, and in particular how we evaluated 134 different string similarity metrics regarding their applicability for reconstructing the version history of text and code blocks. Based on a first analysis using the dataset, we present insights into the evolution of SO posts, e.g., that post edits are usually small, happen soon after the initial creation of the post, and that code is rarely changed without also updating the surrounding text. Further, our analysis revealed a close relationship between post edits and comments. Our vision is that researchers will use *SOTorrent* to investigate and understand the evolution of SO posts and their relation to other platforms such as GitHub.

CCS CONCEPTS

• **Software and its engineering** → **Software evolution**;

KEYWORDS

stack overflow, software evolution, open dataset, code snippets

ACM Reference Format:
Sebastian Baltes, Lorik Dumani, Christoph Treude, and Stephan Diehl. 2018. SOTorrent: Reconstructing and Analyzing the Evolution of Stack Overflow Posts. In *MSR '18: 15th International Conference on Mining Software Repositories , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3196398.3196430

ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196430

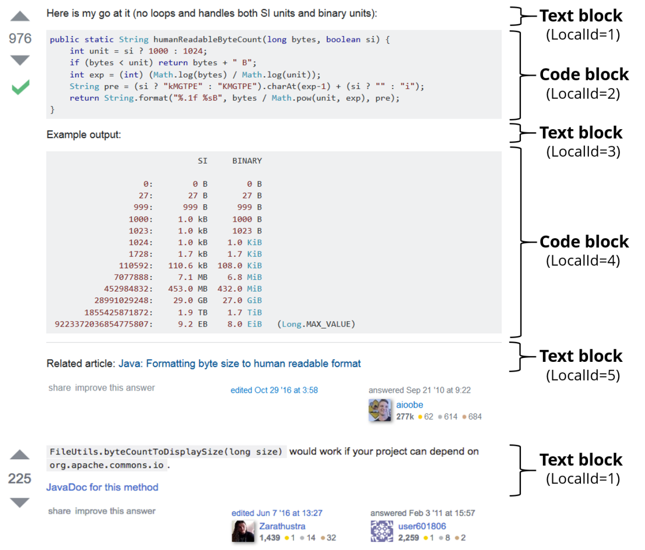
1 INTRODUCTION

Stack Overflow (SO) is the most popular question-and-answer website for software developers. As of December 2017, its public data dump [49] lists over 38 million posts and over 8 million registered users. Many answers contain code snippets together with explanations [60]. Similar to other software artifacts such as source code files and documentation [17, 29, 34, 38], text and code snippets on SO evolve over time, e.g., when the SO community fixes bugs in code snippets, clarifies questions and answers, and updates documentation to match new API versions. Since the inception of SO in 2008, a total of 13.9 million SO posts have been edited after their creation—19,708 of them more than ten times. While many SO posts contain code, the evolution of code snippets on SO differs from the evolution of entire software projects: Most snippets are relatively short (on average 12 lines, see Section 6.1) and many of them cannot compile without modification [60]. In addition, SO does not provide a version control or bug tracking system for post content, forcing users to rely on the commenting function or additional answers to voice concerns about a post.

Recent studies have shown that developers use SO snippets in their software projects, regardless of maintainability, security, and licensing implications [1, 2, 4, 14, 25, 27, 59, 61]. Assuming that developers copy and paste snippets from SO without trying to thoroughly understand them, maintenance issues arise. For instance, it may later be more difficult for developers to refactor or debug code that they did not write themselves. Moreover, if no link to the SO post is added to the copied code, it is not possible to check the SO thread for a corrected or improved solution in case problems occur.

The SO data dump keeps track of different versions of entire posts, but does not contain information about differences between versions at a more fine-grained level. In particular, it is not trivial to extract different versions of the same code snippet from the history of a post to analyze its evolution. To address these challenges, we present the open dataset *SOTorrent*, which enables researchers to analyze the version history of SO posts at the level of whole posts and individual post blocks, and their relation to corresponding source code in GitHub repositories. We also use this dataset to answer three research questions about the evolution of post content on SO, which are, to the best of our knowledge, not sufficiently answered yet: *How do Stack Overflow posts evolve?* (RQ1), *Which posts get edited?* (RQ2), and *What is the temporal relationship between edits and comments?* (RQ3).

While answering the first two questions will further our understanding of the phenomenon of SO post evolution, the third question aims at finding a connection between post edits and other events on the SO platform. We found that SO posts grow over time



976

Here is my go at it (no loops and handles both SI units and binary units):

```
public static String humanReadableByteCount(long bytes, boolean si) {
    int unit = si ? 1000 : 1024;
    if (bytes < unit) return bytes + " B";
    int exp = (int) (Math.log(bytes) / Math.log(unit));
    String pre = (si ? "kMGTPE" : "KMGTPE").charAt(exp-1) + (si ? "" : "i");
    return String.format("%.1f %sB", bytes / Math.pow(unit, exp), pre);
}
```

Example output:

```
                              SI     BINARY

                   0:        0 B        0 B
                  27:       27 B       27 B
                 999:      999 B      999 B
                1000:     1.0 kB     1000 B
                1023:     1.0 kB     1023 B
                1024:     1.0 kB    1.0 KiB
                1728:     1.7 kB    1.7 KiB
              110592:   110.6 kB  108.0 KiB
             7077888:     7.1 MB    6.8 MiB
           452984832:   453.0 MB  432.0 MiB
         28991029248:    29.0 GB   27.0 GiB
       1855425871872:     1.9 TB    1.7 TiB
 9223372036854775807:     9.2 EB    8.0 EiB   (Long.MAX_VALUE)
```

Related article: Java: Formatting byte size to human readable format

share improve this answer edited Oct 29 '16 at 3:58 answered Sep 21 '10 at 9:22 aioobe 277k ●62 ●614 ●684

225

`FileUtils.byteCountToDisplaySize(long size)` would work if your project can depend on `org.apache.commons.io`.

JavaDoc for this method

share improve this answer edited Jun 7 '16 at 13:27 Zarathustra 1,439 ●1 ●14 ●32 answered Feb 3 '11 at 15:57 user601806 2,259 ●1 ●8 ●2

Figure 1: Exemplary Stack Overflow answers with code blocks (top, 3758880) and with inline code (bottom, 4888400). The `LocalId` represents the position in the post.

in terms of their number of text and code blocks, but the size of the individual blocks is relatively stable. Most edits (86.6%) just modify a single line of text or code, but only in 6.1% of the cases are code blocks changed without also changing text content; post edits usually happen shortly after the creation of the post. Our research suggests that comments and post edits are closely related: Some comments might trigger edits, others might be made in response to the edits. The contribution of this work consists of the publicly available dataset *SOTorrent*, the algorithms and techniques used for its construction, and an initial analysis of SO post evolution.

2 THE SOTORRENT DATASET

To answer our research questions, and to support other researchers in answering similar questions, we build *SOTorrent*, an open dataset based on data from the official SO data dump [49] and the Google BigQuery GitHub (GH) dataset [30]. *SOTorrent* provides access to the version history of SO content at the level of whole posts and individual post blocks. A post block can either be a text or a code block, depending on how the author formatted the content (see Figure 1 for an example). Beside providing access to the version history, the dataset links SO posts to external resources in two ways: (1) by extracting linked URLs from text blocks of SO posts and (2) by providing a table with links to SO posts found in the source code of GitHub projects. This table can be used to connect *SOTorrent* and GH datasets such as *GHTorrent* [31]. Our dataset is available on Zenodo as a database dump [9] together with instructions on how to import the dataset. We also published the source code of the software that we used to build [7, 11] and analyze [6, 8] *SOTorrent*.

The current release *2018-02-16* of the dataset contains the version history of all 38,394,895 questions and answers in the official SO data dump. It contains 60,235,289 post versions and 186,924,947 post block versions, ranging from the creation of the first post on July 31, 2008 until the last edit on December 1, 2017. We extracted links to 11,019,477 distinct URLs from 19,453,365 different post block versions and further identified 5,816,307 links to SO posts in 430,521 public GH repositories. In the following sections, we provide information about *SOTorrent*'s data storage and collection process, before we use the dataset to answer our research questions.

3 DATABASE SCHEMA

SOTorrent contains all tables from the official Stack Overflow data dump, published December 1, 2017 [49] (see database schema in Figure 2). However, that dump does only provide the version history at the level of whole posts as Markdown-formatted text. To analyze how individual text or code blocks evolve, we needed to extract individual blocks from that content. This extraction also enabled us to collect links to external sources from the identified text blocks. In the SO dump, one version of a post corresponds to one row in the table `PostHistory`. However, that table does not only document changes to the content of a post, but also changes to metadata such as tags or title. Since our goal was to analyze the evolution of SO posts at the level of whole posts and individual post blocks, we had to filter and process the available data. First, we selected edits that changed the content of a SO post, identified by their `PostHistoryTypeId` [48] (2: *Initial Body*, 5: *Edit Body*, 8: *Rollback Body*). We linked each filtered version to its predecessor and successor and stored it in table `PostVersion`.

The content of a post version is available as Markdown-formatted text. We split the content of each version into text and code blocks (see Section 4) and extracted the URLs from all text blocks using a regular expression (table `PostVersionUrl`). To reconstruct the version history of individual post blocks (table `PostBlockVersion`), we established a linear predecessor relationship between the post block versions using a string similarity metric that we selected after a thorough evaluation (see Section 5.4). For each post block version, we computed the line-based difference to its predecessor, which is available in table `PostBlockDiff`.

One row in table `PostReferenceGH` represents one link from a file in a public GH repository to a post on SO. To extract those references, we utilized Google BigQuery, which allows to execute SQL queries on various public datasets, including a dataset with all files in the default branch of GitHub projects [30]. To find references to SO, we applied the following regular expression to each line of each non-binary file in the dataset:

```
(?i:https?://stackoverflow\.com/[^\s)\.\"]*)
```

Because there are different ways of referring to questions and answers on SO, i.e. using full URLs or short URLs, we mapped all extracted URLs to their corresponding sharing link (ending with `/q/<id>` for questions and `/a/<id>` for answers) and stored that link together with information about the file and the repository in which the link was found in table `PostReferenceGH`. We ignored other links referring to, e.g., users or tags on SO.

4 POST BLOCK EXTRACTION

Our goal was to analyze the evolution of individual text and code blocks, for example to trace changes to particular code snippets or to identify bug fixes for code on SO. Moreover, the differentiation between the two post block types allowed us to extract

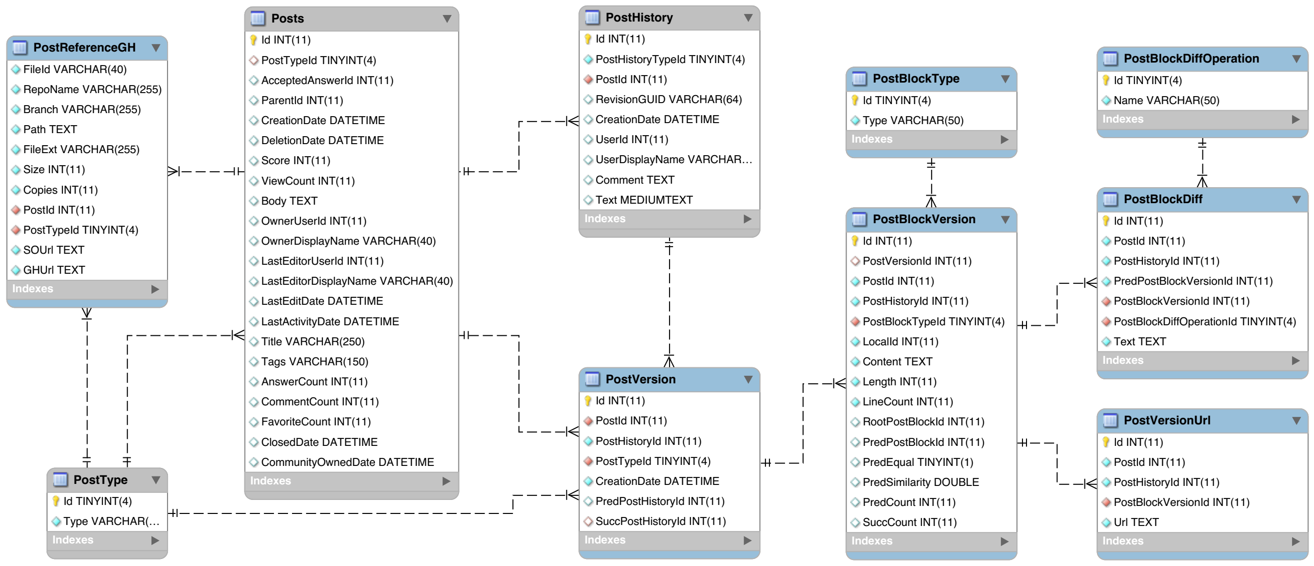


Figure 2: Database schema of *SOTorrent*: The tables from the offical SO dump [48] are marked gray, the additional tables are marked blue. Not all tables from the official SO dump and not all foreign key constraints are shown.

links to external resources only from text blocks, not from code blocks. The latter may, for example, contain XML namespace links or links to stylesheet files, which we do not consider to be external sources of the post. The first step towards reconstructing the version history of individual post blocks is their extraction from the Markdown-formatted text that SO uses for the content of posts. In our notion, a code block is not a short inline code fragment embedded into a text block (see Figure 1 for an example), but a continuous code snippet. We consider inline-code to be part of the surrounding text block. According to SO's Markdown specification [50], code blocks are indented by four spaces and inline code is framed by backtick characters. However, as we found during our research, users are free to use other Markdown specifications or HTML tags, which are not officially supported, but correctly parsed and displayed on the SO website. We iteratively tested and refined our post block extraction approach using a random sample of over 100,000 SO posts (s_{large}). We ran the extraction, randomly checked the extracted posts blocks, and added a new test case if the result differed from the rendering on the SO website (class `PostVersionHistoryTest` [11]). We then updated the extraction such that all test cases passed and re-ran the extraction on the test data. The final version of our post block extraction method was able to detect various notations that SO authors used to mark code blocks, including SO Markdown (indented by 4 spaces), code fencing Markdown (enclosed by three backticks), SO stack snippets (enclosed by `<!--begin/end snippet-->`), stack snippet language tags (prepended by `<!--language:...-->`), HTML code tags (enclosed by `<pre><code>`), and HTML script tags (enclosed by `<script>`).

5 POST BLOCK MATCHING

After successfully extracting the post blocks from a post version, we had to map the extracted post blocks to their predecessors in the previous post version to reconstruct their version history. Since this mapping had to work for text and code content, the latter in various programming languages, we decided to utilize syntax-based similarity metrics. We implemented 134 different string similarity metrics (see Section 5.1), which we evaluated regarding their correctness and performance using the manually validated version history of 600 SO posts (see Sections 5.2 and 5.4). In case of multiple matches, we had to choose between different predecessor candidates. Thus, we developed a matching strategy that considers the location and context of a post block (see Section 5.3).

5.1 Similarity Metrics

A similarity metric maps two input strings to a value in $[0, 1]$, where 0 corresponds to inequality and 1 corresponds to equality. We implemented five different types of similarity metrics: *edit-based metrics* (e.g., Levenshtein), *set-based metrics* (e.g., n-grams with Jaccard coefficient), *profile-based metrics* (e.g, cosine similarity), *fingerprint-based metrics* (Winnowing), and *equality-based metrics*, which served as a baseline in the metrics evaluation (see Section 5.4). Our Java implementation of all metrics is available on GitHub [12]. Table 1 shows all metrics that we implemented and evaluated.

The *edit-based metrics* define the similarity of two strings based on the number of edit operations needed to transform one string into the other. Optimal string alignment (OA) allows the two operations 'insertion of one character' and 'deletion of one character'. The Levenshtein distance further allows 'substitution of one character'. The Damerau-Levenshtein distance is similar to Levenshtein, but additionally allows the operation 'swap two neighboring characters'. The longest common subsequence (LCS) of two strings is the longest sequence of characters (order irrelevant) that can be found in both strings. It can be interpreted as a variant of Damerau-Levenshtein with the additional restriction that each character can only be modified once (e.g., swapping two characters and then

Table 1: Overview of all evaluated similarity metrics ($n = 134$).

Type	Metric		Variants
edit	levenshtein longestCommonSubsequence (LCS)	damerauLevenshtein optimalAlignment (OA)	with/without normalization

set	nGram{Jaccard|Dice|Overlap}
	token{Jaccard|Dice|Overlap}
		nShingle{Jaccard|Dice|Overlap}	nGram : $n \in \{2, 3, 4, 5\}$, nShingle : $n \in \{2, 3\}$
			with/without normalization, padding (nGram)

profile	cosineNGram{Bool|TF|NormalizedTF}
	cosineNShingle{Bool|TF|NormalizedTF}
	cosineToken{Bool|TF|NormalizedTF}
		manhattanNGram
		manhattanNShingle
		manhattanToken
			nGram : $n \in \{2, 3, 4, 5\}$, nShingle : $n \in \{2, 3\}$
			with normalization (both) and without (cosine)

fingerprint	winnowingNGram{Jaccard|Dice|Overlap|LCS|OA}		nGram : $n \in \{2, 3, 4, 5\}$, with/without normalization
equal	equal	tokenEqual	with/without normalization

replacing one of them is not possible). To derive a similarity metric from the number of edit operations and the longest common subsequence, we used the following approaches:

Definition 5.1 (Edit/LCS Similarity). Let S_1, S_2 be two strings, d be the edit distance and LCS be the longest common subsequence between the two strings: $(S_1, S_2) \longrightarrow \mathbb{R}_0^+$. The edit- and LCS-based similarity functions $sim\colon (S_1, S_2) \longrightarrow [0, 1]$ are then defined as

$$sim_{\text{edit}}(S_1, S_2) = \frac{max(|S_1|, |S_2|) - d(S_1, S_2)}{max(|S_1|, |S_2|)}$$

$$sim_{\text{lcs}}(S_1, S_2) = \frac{LCS(S_1, S_2)}{max(|S_1|, |S_2|)}$$

The *profile-based metrics* consider each distinct token, n-gram, or n-shingle in the two input strings as one dimension of a vector space. Tokens can be extracted from a string by a tokenization with whitespaces as delimiter, *n*-grams split the string in sequences of *n* consecutive characters, *n*-shingles split the string in sequences of *n* consecutive words or tokens. One input string is then characterized as one vector in the vector space. In the simplest form (bool), the values of the dimensions can either be 1 (token, n-gram, or n-shingle present in the string) or 0 (not present). Alternatively, one can consider the number of occurrences of each token, n-gram, or n-shingle as the value of the dimensions (term frequency). We also considered the BM15 weighting scheme ($k = 1.5$) [35], which intends to lower the effect of very frequent terms skewing the comparison. The similarity of the two strings is then defined as the cosine or Manhattan distance between the two vectors that have been derived from the strings using one of the three approaches described above.

For the *set-based metrics*, we considered all distinct tokens, n-grams and n-shingles in the strings as elements of sets. We used three coefficients to compare the resulting sets:

Definition 5.2 (Similarity Coefficients). Let S_1, S_2 be sets of tokens, n-grams, or n-shingles.

$$Jaccard(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \qquad Dice(S_1, S_2) = \frac{2 \cdot |S_1 \cap S_2|}{|S_1| + |S_2|}$$

$$Overlap(S_1, S_2) = \frac{|S_1 \cap S_2|}{min(|S_1|, |S_2|)}$$

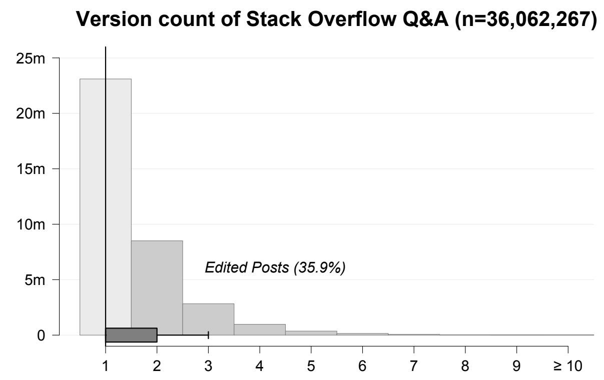
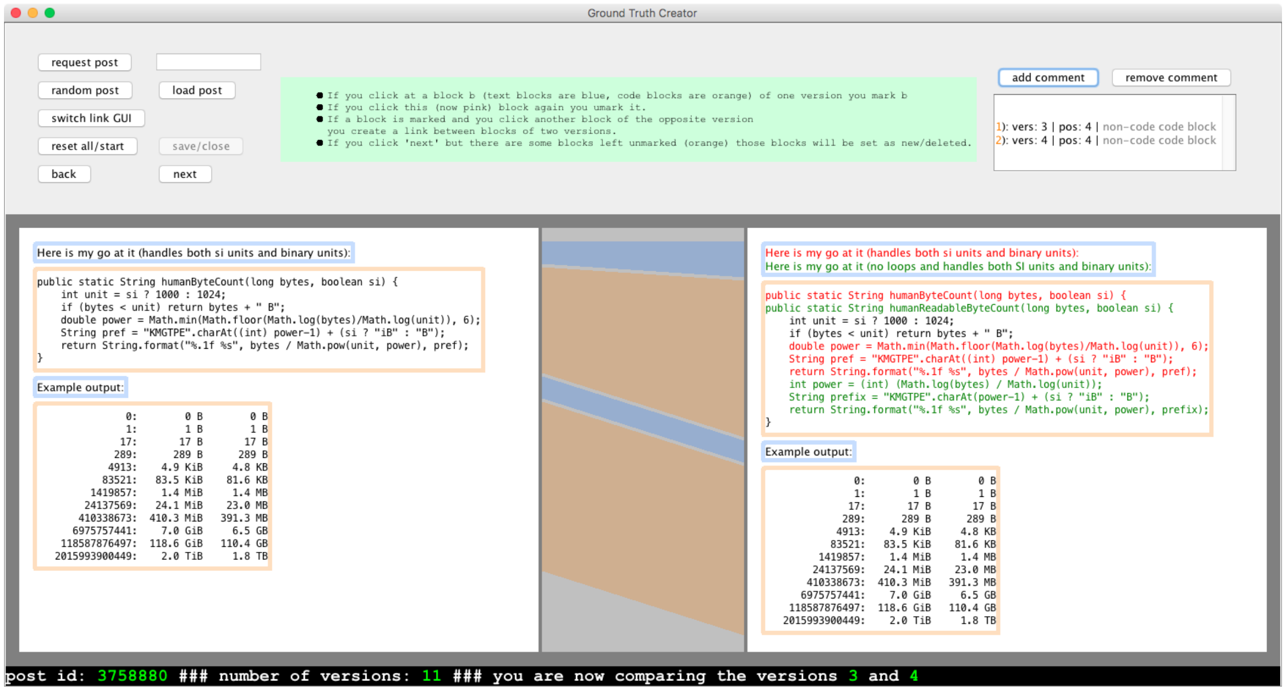


Figure 3: Histogram and boxplot showing the number of Stack Overflow questions and answers with a certain version count (PostHistoryTypeIds 2, 5, 8); based on the SO data dump 2017-06-12; vertical line is median.

The *fingerprint-based metrics* apply a hash function to substrings of the input strings and then use the computed hash values to determine the similarity. The Winnowing algorithm is one approach to calculate and compare the fingerprints of two strings [24, 45]. Winnowing is often used for plagiarism detection, e.g., in the source code comparison software *MOSS* [15, 33, 36]. We implemented different variants of the algorithm described by Schleimer et al. [45], e.g., using different n-grams sizes and different approaches to compare the fingerprints.

We implemented each metric in different variations. In the variants with normalized input strings, we used different approaches for different metric types: For the edit metrics, we unified the whitespace characters, i.e. reduced them to a single space, and converted all characters to lower case. For the n-gram metrics, we converted all characters to lower case, removed all whitespace, and removed some special characters (`{};`). For the shingle metrics, we again converted all characters to lower case, unified the whitespace characters, and removed all non-word characters (`[^a-zA-Z_0-9]`). We used common n-gram and shingle sizes [15] and also implemented an optional n-gram padding that emphasizes the beginning and the end of the input strings. All these variations lead to a total number of 134 different similarity metrics.



- If you click at a block b (text blocks are blue, code blocks are orange) of one version you mark b
- If you click this (now pink) block again you umark it.
- If a block is marked and you click another block of the opposite version you create a link between blocks of two versions.
- If you click 'next' but there are some blocks left unmarked (orange) those blocks will be set as new/deleted.

1): vers: 3 | pos: 4 | non-code code block
2): vers: 4 | pos: 4 | non-code code block

Here is my go at it (handles both si units and binary units):

```
public static String humanByteCount(long bytes, boolean si) {
    int unit = si ? 1000 : 1024;
    if (bytes < unit) return bytes + " B";
    double power = Math.min(Math.floor(Math.log(bytes)/Math.log(unit)), 6);
    String pref = "KMGTPE".charAt((int) power-1) + (si ? "iB" : "B");
    return String.format("%.1f %s", bytes / Math.pow(unit, power), pref);
}
```

Example output:

```
            0:        0 B        0 B
            1:        1 B        1 B
           17:       17 B       17 B
          289:      289 B      289 B
         4913:    4.9 KiB     4.8 KB
        83521:   83.5 KiB    81.6 KB
      1419857:    1.4 MiB     1.4 MB
     24137569:   24.1 MiB    23.0 MB
    410338673:  410.3 MiB   391.3 MB
   6975757441:    7.0 GiB     6.5 GB
 118587876497:  118.6 GiB   110.4 GB
2015993900449:    2.0 TiB     1.8 TB
```

Here is my go at it (handles both si units and binary units):
Here is my go at it (no loops and handles both SI units and binary units):

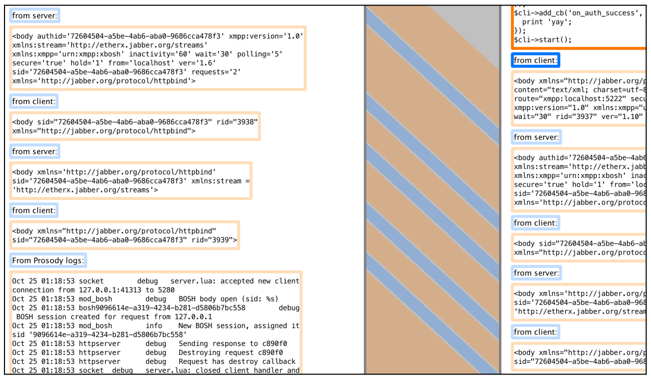
```
public static String humanByteCount(long bytes, boolean si) {
public static String humanReadableByteCount(long bytes, boolean si) {
    int unit = si ? 1000 : 1024;
    if (bytes < unit) return bytes + " B";
    double power = Math.min(Math.floor(Math.log(bytes)/Math.log(unit)), 6);
    String pref = "KMGTPE".charAt((int) power-1) + (si ? "iB" : "B");
    return String.format("%.1f %s", bytes / Math.pow(unit, power), pref);
    int power = (int) (Math.log(bytes) / Math.log(unit));
    String prefix = "KMGTPE".charAt(power-1) + (si ? "iB" : "B");
    return String.format("%.1f %s", bytes / Math.pow(unit, power), prefix);
}
```

Example output:

```
            0:        0 B        0 B
            1:        1 B        1 B
           17:       17 B       17 B
          289:      289 B      289 B
         4913:    4.9 KiB     4.8 KB
        83521:   83.5 KiB    81.6 KB
      1419857:    1.4 MiB     1.4 MB
     24137569:   24.1 MiB    23.0 MB
    410338673:  410.3 MiB   391.3 MB
   6975757441:    7.0 GiB     6.5 GB
 118587876497:  118.6 GiB   110.4 GB
2015993900449:    2.0 TiB     1.8 TB
```

```
post id: 3758880 ### number of versions: 11 ### you are now comparing the versions 3 and 4
```

Figure 4: App developed to create ground truth for similarity metric evaluation.



from server:

```
<body authid='72604504-a5be-4ab6-aba0-9686cca478f3' xmpp:version='1.0'
xmlns:stream='http://etherx.jabber.org/streams'
xmlns:xmpp='urn:xmpp:xbosh' inactivity='60' wait='30' polling='5'
secure='true' hold='1' from='localhost' ver='1.6'
sid='72604504-a5be-4ab6-aba0-9686cca478f3' requests='2'
xmlns='http://jabber.org/protocol/httpbind'>
```

from client:

```
<body sid="72604504-a5be-4ab6-aba0-9686cca478f3" rid="3938"
xmlns="http://jabber.org/protocol/httpbind">
```

from server:

```
<body xmlns='http://jabber.org/protocol/httpbind'
sid='72604504-a5be-4ab6-aba0-9686cca478f3' xmlns:stream =
'http://etherx.jabber.org/streams'>
```

from client:

```
<body xmlns="http://jabber.org/protocol/httpbind"
sid="72604504-a5be-4ab6-aba0-9686cca478f3" rid="3939">
```

From Prosody logs:

```
Oct 25 01:18:53 socket        debug   server.lua: accepted new client
connection from 127.0.0.1:41313 to 5280
Oct 25 01:18:53 mod_bosh        debug   BOSH body open (sid: %s)
Oct 25 01:18:53 bosh9096614e-a319-4234-b281-d5806b7bc558        debug
 BOSH session created for request from 127.0.0.1
Oct 25 01:18:53 mod_bosh        info    New BOSH session, assigned it
sid '9096614e-a319-4234-b281-d5806b7bc558'
Oct 25 01:18:53 httpserver      debug   Sending response to c890f0
Oct 25 01:18:53 httpserver      debug   Destroying request c890f0
Oct 25 01:18:53 httpserver      debug   Request has destroy callback
Oct 25 01:18:53 socket  debug   server.lua: closed client handler and
```

```
$cli->add_cb('on_auth_success',
  print 'yay';
});
$cli->start();
```

from client:

```
<body xmlns="http://jabber.org/p
content="text/xml; charset=utf-8
route="xmpp:localhost:5222" secu
xmpp:version="1.0" xmlns:xmpp="u
wait="30" rid="3937" ver="1.10"
```

from server:

```
<body authid='72604504-a5be-4ab6
xmlns:stream='http://etherx.jabb
xmlns:xmpp='urn:xmpp:xbosh' inac
secure='true' hold='1' from='loc
sid='72604504-a5be-4ab6-aba0-968
xmlns='http://jabber.org/protoco
```

from client:

```
<body sid="72604504-a5be-4ab6-ab
xmlns="http://jabber.org/protoco
```

from server:

```
<body xmlns='http://jabber.org/p
sid='72604504-a5be-4ab6-aba0-968
'http://etherx.jabber.org/stream
```

from client:

```
<body xmlns="http://jabber.org/p
sid="72604504-a5be-4ab6-aba0-968
```

Figure 5: Post with multiple equal predecessors (13064858).

5.2 Ground Truth

To evaluate the correctness of the post block mappings retrieved using different string similarity metrics, we created a set of 600 manually validated post version histories. Figure 4 shows a screenshot of the tool we developed to create those manually validated histories (available on GitHub [23]). It visualizes a post version (right) and its predecessor (left). Post blocks with equal content and type that are unique in the two versions are automatically connected. For the other post blocks, the user has to choose a match by clicking on a post block of the same type in each version; the tool then visualizes the line-based difference between the connected blocks. It is also possible to add comments for individual post blocks, e.g., in case the user is not confident in his or her mapping, or in case the post block extraction failed.

We drew four different samples from the SO data dump released June 12, 2017. The first sample with 200 posts (s_{rand}) was randomly drawn from all SO questions and answers with at least two versions (otherwise no mapping is needed). Since there are many posts with only two versions (see Figure 3), we decided to draw another sample of 200 posts from SO questions and answers with at least seven versions (99% quantile). We denote this sample s_{rand+}. As the initial focus of our research was on Java, we also drew a sample with 200 Java posts (s_{java}) from all SO questions tagged with <java> or <android>, and the corresponding answers. The last sample (s_{mult}), which contains 100 posts with multiple possible predecessors, was not used to evaluate the metrics, but to evaluate our matching strategy (see Section 5.3). In this sample, we included posts which had at least two possible matches (two post blocks of the same type with identical content) in two adjacent versions.

The validated version histories of the samples were created by a graduate student, and then later discussed with two of the authors. The student was introduced to the app and told to comment all post blocks where he is not sure about the mapping. Together, we looked at all post blocks with comments indicating an unclear mapping ($n = 38$) and tried to find a mapping we all agreed on. If that was not possible, we moved the post to a new sample $s_{unclear}$, which we separately analyzed. After discussing all 38 posts, $s_{unclear}$ contained 17 posts (4 from s_{rand}, 8 from s_{rand+}, and 5 from s_{java}). All samples are available on Zenodo [13].

5.3 Matching Strategy

Our goal was to establish a linear predecessor relationship for all post block versions, thus each post block version can only have one predecessor. The reason for this decision was the fact that we rarely observed splits and merges in the post version histories we manually analyzed. Moreover, even if multiple predecessors have equal or similar content, usually only one of them is the actual predecessor (see Figure 5 for an example). To correctly choose the predecessor from different candidates, we had to develop a matching strategy for post block versions, which we present in this section. In the database, we not only store the matched predecessor, but also the number of possible predecessors and successors, to be later able to identify post version histories that could contain splits or merges. For our analysis (see Section 6), we consider *post block lifespans*, i.e. chains of connected post block versions that are predecessors of each other. Those lifespans can be easily retrieved from the database, because each post block version has a `RootPostBlockId`, which is the id of the first post block version in the chain. As mentioned above, we utilized a dedicated sample s_{mult} to evaluate how well our matching strategy can handle posts with multiple possible connections. In case of differences between the ground truth and the results of our approach, we wrote unit tests replicating the issue and then updated the strategy until all unit tests passed. We further used the sample s_{large} to test the strategy's scalability. To be able to describe our matching strategy, we define our notation for post versions, post block versions, and possible predecessors:

Definition 5.3 (Post Version). Let p be a post with n versions. Then p_i denotes one post version and $|p_i|$ denotes the number of post blocks in p_i for $i \in \{1 \dots n\}$.

Definition 5.4 (Post Block Version). Let p_i be one post version and $\tau \in \{\text{text}, \text{ code}\}$ be a post block type. Then $b^{\tau}_{(i,l)}$ denotes one post block of type τ with local id l for $l \in \{1 \dots |p_i|\}$. The function $id^{\tau} : p_i \longrightarrow \{1 \leq l \leq |p_i|\}$ maps a post version to the local ids of the post blocks of type τ in that version.

Definition 5.5 (Possible Predecessors). Let $b^{\tau}_{(i-1,l)}, b^{\tau}_{(i,j)}$ be post blocks of the same type in subsequent post versions,

$$equal(b^{\tau}_{(i-1,l)}, b^{\tau}_{(i,j)}) \longrightarrow \{\text{true}, \text{false}\}$$

be a function that tests if the post blocks' contents are equal, and

$$sim^{\tau}(b^{\tau}_{(i-1,l)}, b^{\tau}_{(i,j)}) \longrightarrow [0, 1]$$

be the similarity of the two post blocks' contents according to the similarity metric sim^{τ}. Let $\vartheta^{\tau} \in [0, 1]$ be a threshold for sim^{τ}. Then, we define the set of equal predecessors as

$$PredEqual(b^{\tau}_{(i,j)}) = \{\beta^{\tau}_{(i-1,k)} \mid equal(\beta^{\tau}_{(i-1,k)}, b^{\tau}_{(i,j)}) = \text{true}, \\ k \in id^{\tau}(p_{i-1}),\ j \in id^{\tau}(p_i)\}$$

We define the maximum predecessor similarity as

$$maxSim^{\tau} = max(\{sim^{\tau}(\beta^{\tau}_{(i-1,k)}, b^{\tau}_{(i,j)}) \mid sim^{\tau} \geq \vartheta^{\tau}, \\ k \in id^{\tau}(p_{i-1}),\ j \in id^{\tau}(p_i)\})$$

In case no predecessor with a similarity above the threshold exists, we define $maxSim^{\tau}(\emptyset) = 0$. We define the set of similar predecessors as

$$PredSim(b^{\tau}_{(i,j)}) = \{\beta^{\tau}_{(i-1,k)} \mid sim^{\tau}(\beta^{\tau}_{(i-1,k)}, b^{\tau}_{(i,j)}) \geq maxSim^{\tau}, \\ k \in id^{\tau}(p_{i-1}),\ j \in id^{\tau}(p_i)\}$$

Finally, we define the set of possible predecessors as

$$Pred(b^{\tau}_{(i,j)}) = \begin{cases} PredEqual(b^{\tau}_{(i,j)}), & \text{if } PredEqual(b^{\tau}_{(i,j)}) \neq \emptyset, \\ PredSim(b^{\tau}_{(i,j)}), & \text{if } PredEqual(b^{\tau}_{(i,j)}) = \emptyset. \end{cases}$$

The set of possible successors $Succ(b^{\tau}_{(i,j)})$ is defined analogously.

As it can be seen in the above definition, we need two different similarity metrics (sim^{text} and sim^{code}) and two different similarity thresholds (ϑ^{text} and ϑ^{code}). We only compute the similarity if the content of the post blocks is not equal, because we want to be able to distinguish equal post block versions from post block versions with similarity 1 according to the metric. Before we describe our matching strategy, we present two methods that we use in case of multiple possible predecessors. Both methods iterate over all post blocks $b^{\tau}_{(i,j)}$ in a post version $p_{2 \leq i \leq n}$ that do not have a predecessor yet. They follow different strategies for selecting a predecessor:

setPredContext(p_i, *BOTH*) tries to select a predecessor using the post blocks before and after $b^{\tau}_{(i,j)}$, i.e. the blocks with local ids $j-1$ and $j+1$. Please note that those blocks usually have a different post block type than $b^{\tau}_{(i,j)}$. In case the predecessors of those neighboring blocks are already set and one post block $b^{\tau}_{(i-1,l)} \in Pred(b^{\tau}_{(i,j)})$ has the predecessors of those two post blocks as neighbors (local ids $l-1$ and $l+1$ in version p_{i-1}), the function sets $b^{\tau}_{(i-1,l)}$ as predecessor of $b^{\tau}_{(i,j)}$ and returns *true*. If no predecessor has been set, it returns *false*. In case of parameter *ABOVE*, only the post block above (local id $j-1$) is taken into account; in case of parameter *BELOW*, only the post block below (local id $j+1$) is taken into account. Examples for posts that motivated this strategy are answer 32841902 (mapping of version 2 to 1) and answer 37196630 (mapping of version 2 to 1).

setPredPosition(p_i) sets the post block $b^{\tau}_{(i-1,l)} \in Pred(b^{\tau}_{(i,j)})$ with $\Delta_{\text{pos}} = min(|l-j|)$, i.e. the post block with the local id closest to j, as predecessor of $b^{\tau}_{(i,j)}$. If two possible predecessors have the same Δ_{pos}, the method chooses the one with the smallest local id. This approach is based on our observation that the order of post blocks rarely changes (see Section 6.1). Examples for posts that motivated this strategy are question 18276636 (mapping of version 2 to 1) and answer 2581754 (mapping of version 3 to 2).

The complete matching strategy that selects (at most) one predecessor for each post block in a post version can be found as pseudo code in Algorithm 1. The actual source code can be found in method `processVersionHistory` of class `PostVersionList` in the corresponding GitHub project [11].

5.4 Metrics Evaluation

The matching strategy described above depends on the results of the similarity metrics sim^{text} and sim^{code} and the thresholds ϑ^{text} and ϑ^{code}. To select the best metrics for reconstructing the version history of post blocks, we evaluated all 134 metrics in different combinations with different thresholds using our ground truth samples s_{rand}, s_{rand+}, and s_{java}. Please note that the correctness of sim^{text} and sim^{code} cannot be evaluated independently, because the neighboring post blocks that *setPredContext* takes into account usually have different types. To assess the performance, we measured the runtime of the post history extraction for each configuration. To assess the correctness of the extracted post block history, we regarded

Algorithm 1 Matching Strategy

```
for all $p_{2 \leq i \leq n}$ do
    // set predecessors where only one candidate exists
    for all $b^{\tau}_{(i, 1 \leq j \leq |p_i|)}$ do
        if $|Pred(b^{\tau}_{(i,j)})| = 1$ then
            Let $pred$ be the equal or similar predecessor
            if $|Succ(pred)| = 1$ then
                Set $pred$ as predecessor of $b^{\tau}_{(i,j)}$
                continue
            end if
        end if
    end for
    // set predecessors using context
    $predSet$ = true
    while $predSet$ do
        $predSet = setPredContext(p_i, BOTH)$
    end while
    while $predSet$ do
        $predSet = setPredContext(p_i, BELOW)$
    end while
    while $predSet$ do
        $predSet = setPredContext(p_i, ABOVE)$
    end while
    // set predecessors using position
    $setPredPosition(p_i)$
end for
```

each metric configuration as a binary classifier that either assigns the predecessor of a post block version correctly or not (compared to the ground truth). To calculate the number of true/false positives/negatives, we consider the set of *predecessor connections*, i.e. all $(b^{\tau}_{(i-1,l)}, b^{\tau}_{(i,j)})$ from $p_{2 \leq i \leq n}$ that have been connected with a certain metric configuration. We then compare those connections with the connections from the ground truth:

Definition 5.6 (Metric Evaluation)**.** Let GT^{τ} be the set of predecessor connections of type τ in the ground truth, C^{τ} be the set of predecessor connections of type τ determined using a certain metric configuration, and $n^{\tau}_{pos} = \sum_{2 \leq i \leq n} |id^{\tau}(p_i)|$ be the number of possible predecessor connections of type τ. We define the number of true positives tp^{τ}, false positives fp^{τ}, true negatives tn^{τ}, and false negatives fn^{τ} as:

$$tp^{\tau} = |\mathrm{C} \cap \mathrm{GT}| \qquad fp^{\tau} = |\mathrm{C} \setminus \mathrm{GT}|$$
$$tn^{\tau} = n^{\tau}_{pos} - |\mathrm{C} \cup \mathrm{GT}| \qquad fn^{\tau} = |\mathrm{GT} \setminus \mathrm{C}|$$

After each comparison run, we ranked the configurations according to their Matthews correlation coefficient (*MCC*) [37], which takes tp^{τ}, fp^{τ}, tn^{τ}, and fn^{τ} into account. If two configurations had the same *MCC* value, we ranked them according to their runtime. *MCC* is the preferred measure when evaluating binary classifiers [19] and should be chosen over evaluation measures such as recall, precision, or F-measure [43]. In our case, it correlates the connections from the ground truth and the connections set by a certain metric configuration. The *MCC* values are in range $[-1, 1]$; a total disagreement is represented by -1, a perfect agreement by 1. The source code of the tool we used for the metrics evaluation is available on GitHub [10].

In the first comparison run, we configured $sim^{text} = sim^{code}$ and chose $\vartheta^{\{text,\ code\}} \in \{0.0, 0.1, 0.2, \ldots, 1.0\}$. This resulted in 1,474 different configurations. The first run took about 24 hours on a regular desktop PC (Intel Core i7-7700, 64 GB RAM, 512 GB SSD).

For the second run, we selected the metrics which, for a particular threshold, achieved a *MCC* value in the 95% quantile of all three samples either for text or for code blocks. Some metrics cannot be applied to very short strings (e.g., if string length < n-gram size). For the final implementation, we wanted to have a backup metric that works for all input strings. We filtered edit- and token-based metrics and selected the best candidates according to the criterion described above. Finally, we selected 27 regular and 4 backup metrics for the second run. We also added the *equal* metric as a baseline. We tested those metrics again with $sim^{text} = sim^{code}$, but this time we chose $\vartheta^{\{text,\ code\}} \in \{0.0, 0.01, 0.02, \ldots, 1.0\}$ Thus, the second run tested 3,232 different configurations, which took about 20 hours.

As motivated above, the results of the text and code metrics depend on each other. In the third and last run, we tested all combinations of the best (99% quantile) text and code configurations together with the best backup configurations. This was the first run with $sim^{text} \neq sim^{code}$ and with a backup metric for text and code blocks. Those backup metrics were only used if the input strings were too short for the configured metrics. The run, which took about 14 hours, tested all combinations of 13 text configurations, 3 text backup configurations, 15 code configurations, and 2 code backup configurations, resulting in 1,170 combinations in total. For the final selection, we ranked the combinations according to the sum of their *MCC* scores for text and code blocks. The selected configuration was:

$$\begin{aligned}
sim^{text} &= manhattanFourGramNormalized & (\vartheta^{text} &= 0.17)\\
sim^{code} &= winnowingFourGramDiceNormalized & (\vartheta^{\mathrm{code}} &= 0.23)\\
sim^{text}_{backup} &= cosineTokenNormalizedTermFrequency & (\vartheta^{text} &= 0.36)\\
sim^{code}_{backup} &= cosineTokenNormalizedTermFrequency & (\vartheta^{code} &= 0.26)
\end{aligned}$$

Figure 6 shows the performance of the selected metrics for different thresholds with $sim^{text} = sim^{code}$, compared to the baseline metric *equals.* The final configuration achieved a *MCC* value of 0.86 for text (true positive rate 0.99, false positive rate 0.14) and 0.92 for code (true positive rate 0.99, false positive rate 0.07).

6 DATA ANALYSIS

After describing how we reconstructed the version history for individual text and code blocks, we come back to our initial research questions. We first characterize the phenomenon of SO post evolution, and in particular the evolution of individual post blocks (RQ1). To find out if edited posts share common characteristics, we analyzed if certain measures such as score or number of comments correlate with the number of edits (RQ2). We also investigated if those measures have a temporal relationship with the edits, in particular if comments happen immediately before or after edits (RQ3). As descriptive statistics, we use mean (M), standard deviation (SD), median (Mdn), and the first and third quartiles (Q_1, Q_3). To test for significant differences, we applied the nonparametric two-sided

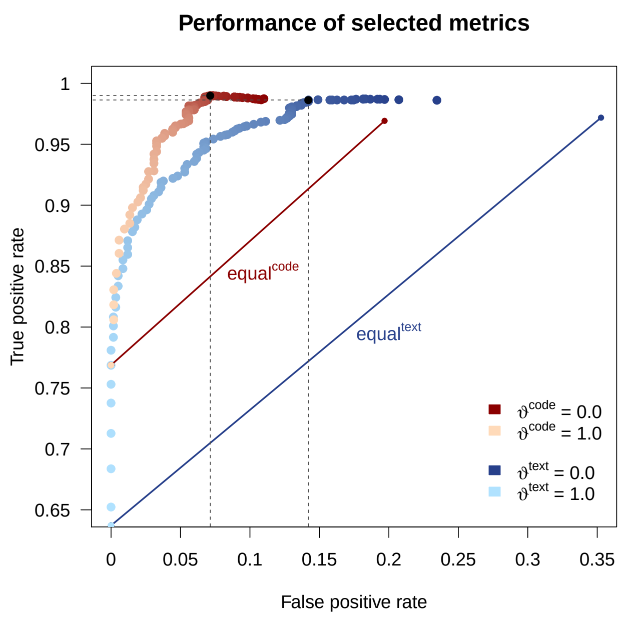


Figure 6: Performance of selected metrics: *manhattanFourGramNormalized* for text (blue) and *winnowingFourGramDiceNormalized* for code (red); selected thresholds: 0.17 for text and 0.23 for code (dotted lines).

Wilcoxon rank-sum test [58] and report the corresponding p-value (p_w). To measure the effect size, we used *Cohen's d* [20, 28]. Our interpretation of d is based on the thresholds described by Cohen [21]: negligible effect ($|d| < 0.2$), small effect ($0.2 \le |d| < 0.5$), medium effect ($0.5 \le |d| < 0.8$), otherwise large effect. We used the non-parametric *Spearman's rank correlation coefficient* (ρ) [47] to test the statistical dependence between two variables. Our interpretation of ρ is based on Hinkle et al.'s scheme [32]: low correlation ($0.3 \le |\rho| < 0.5$), moderate correlation ($0.5 \le |\rho| < 0.7$), high correlation ($0.7 \le |\rho| < 0.9$), and very high correlation ($0.9 \le |\rho| \le 1$).

6.1 Evolution of Stack Overflow Posts

In the following, we describe different properties of post blocks and post block versions either for their latest version in the dataset, or for different versions over time:

Post Block Count: Half of all posts in the *SOTorrent* dataset contain between one and two text blocks and between zero and two code blocks ($Q_{1,3}$). There are only few posts without text blocks (1.0%), but over a third of all posts do not have code blocks (36.6%). Examples for such posts include conceptual questions and answers, but also posts with inline code that we considered to be part of the text blocks. If we compare the first and the last version of edited posts, we can observe a statistically significant difference in the number of text and code blocks ($p_w^{text,\ code} < 2.2\text{e}{-16}$); posts tend to grow over time. However, the effect is only small ($d^{text} = 0.21,\ d^{code} = 0.23$).

Post Block Length: Code blocks tend to be larger than text blocks. Figure 7 visualizes the difference measured in number of lines. The

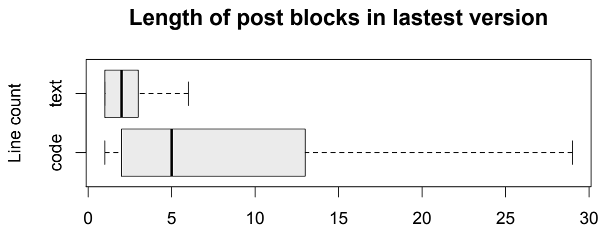


Figure 7: Boxplots showing the line count of text and code blocks in the latest version of Stack Overflow posts (n = 69, 940, 599 for text and n = 42, 568, 011 for code).

average text block contains 2.5 lines ($Mdn = 2$, $SD = 3.1$) and 247.5 characters ($Mdn = 153$, $SD = 319.1$); the average code block contains 12.0 lines ($Mdn = 5$, $SD = 23.4$) and 455.9 characters ($Mdn = 194$, $SD = 989.3$). We compared the length of post blocks in the first and the last version and found no effect. Thus, we can conclude that posts tend to become longer over time in terms of their number of post blocks, but the length of individual post blocks is relatively stable.

Post Block Versions: For our analysis of post block versions, we retrieved all post block lifespans in the dataset, but only considered the initial versions and later versions where the content of the blocks changed (not all blocks are edited in all versions). We found that about half of all post blocks were edited after their creation (see Figure 8). On average, text blocks have 4.8 and code blocks 4.1 versions. We analyzed the line-based differences between post block versions and found that 86.6% of all edits modify only one line (92.5% for text blocks and 71.4% for code blocks). There is a significant difference in the size of changes when comparing text and code blocks ($p_w < 2.2\text{e}{-16}$) with a small to medium effect ($d = 0.46$ for the number of added lines and $d = 0.51$ for the number of deleted lines): Changes in code blocks are larger, which is expectable due to the larger size of code compared to text blocks.

Post Block Co-change: We were also interested in the co-change of text and code blocks, i.e. if text and code is edited together. On average, 1.5 text blocks and 0.9 code blocks were edited in each post version ($Mdn = 1$ and $SD = 1.1$ for both types). We found that text and code blocks were either edited together (49.3% of all post versions), or just the text blocks were edited (44.6%). Only in 6.1% of all post versions, code blocks were changed without also editing text blocks. This could indicate that SO authors document changes to their code snippets in the text blocks or update the description of the modified code.

Order of Post Blocks: To check our assumption that the order of post blocks rarely changes, we computed the difference between the local ids of all post blocks versions and their predecessors. We found that 95.5% of all post block versions have the same local id as their predecessor. Of all absolute differences, two was the most common one (3.1%), which is expectable, because text and code blocks usually alternate. Thus, e.g., swapping two blocks of the same type leads to a local id difference of two in the next version.

Timespan Between Edits: For the posts that have been edited after their creation, we analyzed the timespan between the edits. 80.6%

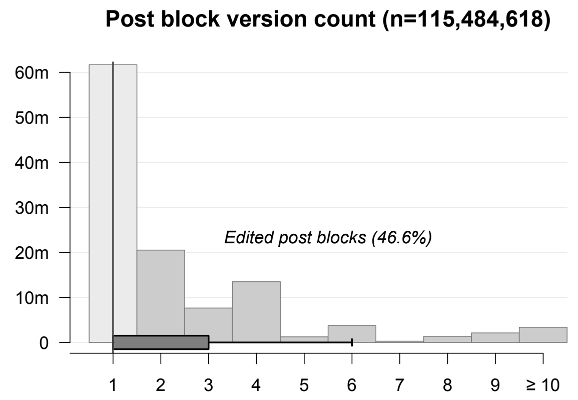


Figure 8: Histogram and boxplot showing the number of post block versions (vertical line is median).

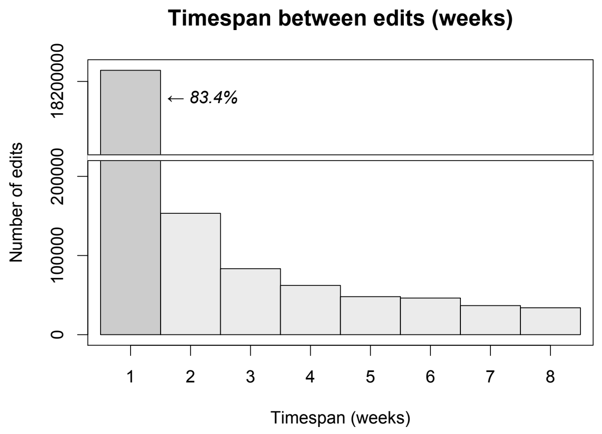


Figure 9: Bar chart visualizing all edit timespans between one and eight weeks (85.5% of all values, $n = 18,677,709$); the other 14.5% are spread over a range of 475 weeks.

of the first post edits happen on the same day as the creation of the post, 4.6% within one week (> 1 and ≤ 7 days), 5.1% within one year (> 7 and ≤ 365 days), and 9.7% more than one year after the creation. If we only consider the second or later edits, not much changes: 74.2% of them happen on the same day, 6.2% within one week, 7.9% within one year, and 11.7% more than one year after the creation. Overall, 78.2% of all edits happen on the same day, i.e. soon after the creation of the post, and 83.4% happen on the same day or within the first week after the creation (see Figure 9).

Post Editors: On SO, either the author of a post or a moderator, i.e. a SO user with a reputation of at least 2,000, can make edits. We found that 87.4% of all edits were conducted by the post authors themselves and 12.6% by moderators. We found no effect of the authors' reputation on the fact that a moderator edits the post. We consider an analysis of typical moderator changes to be an interesting direction for future work.

Table 2: Correlation table with Spearman's correlation coefficients ρ for different properties of Stack Overflow posts (p-value < 0.001 for all combinations).

ρ	Versions	Age	Score	Comments	GHMatches
Versions		−0.03	0.09	**0.26**	0.09
Age	−0.03		0.25	−0.03	0.10
Score	0.09	0.25		0.08	0.23
Comments	**0.26**	−0.03	0.08		0.09
GHMatches	0.09	0.10	0.23	0.09	
n	38.4m	38.4m	38.4m	38.4m	137k

6.2 Properties of Edited Posts

To investigate which properties edited posts possess, we searched for monotonic relationships between the version count of a post and other properties such as the age of the post, its score, comment count, or the number of distinct files on GH referring to the post. Table 2 shows the correlation coefficients (ρ) for those relationships. There was no correlation that exceeded the threshold for a low correlation (0.3). However, the relationship between the version count and the number of comments drew our attention as it had the highest correlation coefficient in the table. We decided to explore that relationship using a quasi-experiment: We compared the number of comments of all posts with only one version to all posts with more than one version (version count over all posts: $Mdn = 1$, $M = 1.6$, $SD = 1.0$). The difference was significant ($p_w < 2.2\text{e}{-16}$) and the effect size was medium ($d = 0.52$). We also compared the opposite relationship, i.e. the number of versions of all posts with at most one comment to all posts with more than one comment (comment count over all posts: $Mdn = 1$, $M = 1.6$, $SD = 2.5$). Again, the difference was significant ($p_w < 2.2\text{e}{-16}$), but the effect size was small ($d = 0.49$).

6.3 Edits and Comments

To further explore the relationship between comments and post edits, we looked at their temporal connection, i.e. if comments usually happen before or after edits. First, we aggregated all edits (including post creation) and all comments per post id and day. Thus, our units of observation were all days where posts were either created, edited or commented. We found that in 32.3% of the cases, the posts were created or edited and commented; in 33.3% of the cases they were only created, in 9.1% of the cases only edited, in 7.5% of the cases only created and edited, and in 17.8% of the cases only commented. If we focus on the comments, we see that 64.4% of them happened on a day where the post had either been created or edited. We then further focused on those days and calculated the time difference between a comment and the closest edit. If a comment was closer to the creation then to an edit, we assigned the comment to the creation. We found that 34.7% of the edits were related to the creation of the post and 65.3% were related to an edit. Of the latter, 47.9% were made before an edit and 52.1% afterwards. Moreover, the comments were usually made right before ($M = -1.2$ hours, $Mdn = -0.3$, $SD = 2.6$) or soon after the edits ($M = +1.3$ hours, $Mdn = +0.3$, $SD = 2.7$).

7 DISCUSSION

The *SOTorrent* dataset has allowed us to study the phenomenon of post editing on SO in detail (RQ1). We found that a total of 13.9 million SO posts (36.1% of all posts) have been edited at least once. Many of these edits (86.6%) only modify a single line of text or code, and while posts grow over time in terms of the number of text and code blocks they contain, the size of these individual blocks is relatively stable. Interestingly, only in 6.1% of all cases are code blocks changed without corresponding changes in text blocks of the same post, suggesting that SO users typically update the textual description accompanying code snippets when they are edited. Studying the exact nature of such edits will be part of our future work. We also found that edits are mostly made shortly after the creation of a post (78.2% of all edits are made on the same day when the post was created), and the vast majority of edits are made by post authors (87.4%)—although the remaining 12.6% will be of particular interest for our future work. The number of comments on posts without edits is significantly smaller than the number of comments on posts with edits, suggesting an interplay of these two features (RQ2). We find evidence which suggests that commenting a post on SO helps to bring attention to it (RQ3). Of the comments that were made on the same day as an edit, 47.9% were made before an edit and 52.1% afterwards, typically (median value) only 18 minutes before or after the edit. Comments before edits might trigger them, comments after edits might be feedback.

To investigate the connection between post edits and comments made immediately before or after edits, we conducted a preliminary qualitative analysis. We drew a random sample of 50 posts, 25 posts for which at least one comment had been made at most 10 minutes before an edit and 25 posts for which at least one comment had been made at most 10 minutes after an edit. We qualitatively analyzed the posts and found that, in the majority of cases, the comments and edits were clearly related (34 of 50 posts in our sample) and that the edit added or modified a code block (30/50). We classified a small set of comments as bug reports (10/50) and found that in some cases, the edit was explicitly documented in the post (11/50, e.g., by prefixing content with "*EDIT:*"). Comments often asked for additional information (22/50), and in cases where comments happened shortly before the edits, the comment was often a clarifying question (14/25). Answer 15437937[1] represents a typical example: In a timespan of 35 minutes, a user answered a question, edited the answer three times, and commented on it once in response to three comments from the user asking the question. Analyzing such communication structures, e.g., to learn how comments are used for feedback on posts, is part of our future work.

8 RELATED WORK

Over the past years, there have been various research papers on leveraging knowledge from SO, e.g., to support post edits [18], to automate the search [16, 41], or to augment API documentation [53]. Regarding the population of SO users, studies described properties such as gender [55] and age [39]. Wang et al. [57] analyzed the asking and answering behavior of SO users and found that most of them only answer or ask one question. We complement those results with our finding that post edits happen soon after post creation and that comments are closely linked to edits. Xia et al. [59] describe that it is common for developers to search for reusable code snippets on the web. Yang et al. [60] found that SO Python and JavaScript snippets are more usable in terms of parsability, compilability and runnability, compared to Java and C#. Yang et al. [61] analyzed code clones between Python snippets from SO and Python projects on GH and found a considerable number of non-trivial clones, which may have a negative impact on code quality [1]. Other studies aimed at identifying API usage in SO code snippets [51], describing characteristics of effective code examples [40], investigating whether SO code snippets are self-explanatory [54], or analyzing the impact of copied SO code snippets on application security [2, 25]. There has also been work on the interplay between user activity on SO and GH [5, 46, 56]. *SOTorrent* enables researchers to further investigate this connection by collecting links from public GH projects to SO posts. To describe topics of SO questions and answers, different methods like manual analysis [52] and Latent Dirichlet Allocation [3, 57] have been used. Automatically identifying high-quality posts has been another research direction, where metrics based on the number of edits on a question [62], author popularity [42], and code readability [22] yielded good results. With our dataset, the evolution of such high-quality posts can easily be analyzed. German et al. [26] investigated how code siblings, code clones that evolve in a different system than the original code, flow between systems with different licenses. Tracing the flow of siblings between GH projects, posts on SO, and external sources is another possible direction for future work that *SOTorrent* can support. Two fields related to our research are source code plagiarism detection [33] and code clone detection [44], which both rely on determining the similarity of code fragments.

9 CONCLUSION

In this paper, we presented *SOTorrent*, an open dataset that enables researchers to analyze the evolution of SO content at the level of whole posts and individual text and code blocks. We described how we evaluated 134 different string similarity metrics regarding their suitability to match text and code blocks to their predecessor versions. For text blocks, a profile-based metric using the Manhattan distance yielded the best results; for code blocks, a fingerprint-based metric using the Winnowing algorithm [24, 45] outperformed the other metrics. Since multiple predecessor candidates may exist, we also developed a matching strategy that we iteratively refined using random samples of SO posts. First analyses using the dataset provided new insights into the evolution of SO posts. In future work, we want to deepen our understanding of how code snippets are maintained on SO. To this end, we want to identify bug-fixing edits. Moreover, as *SOTorrent* also collects links from SO posts to other websites and from public GH projects to SO posts, we can explore how code flows from and to external sources like blog posts and open source software projects. Beside the investigation of new research questions, we will improve and maintain the dataset, for example by developing means to automatically detect code blocks that are not used for code, but for markup (see, e.g., second code block in Figure 1). Our vision is that *SOTorrent* will help other researchers to further investigate the evolution of SO posts and their connection to other platforms and resources.

[1] https://stackoverflow.com/a/15437937

REFERENCES

[1] Rabe Abdalkareem, Emad Shihab, and Juergen Rilling. 2017. On code reuse from StackOverflow: An exploratory study on Android apps. *Information and Software Technology* 88 (2017), 148–158.

[2] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2016. You Get Where You're Looking For: The Impact Of Information Sources on Code Security. In *2016 IEEE Symposium on Security and Privacy (S&P 2016)*, Michael Locasto, Vitaly Shmatikov, and Úlfar Erlingsson (Eds.). IEEE Computer Society, San Jose, CA, USA, 289–305.

[3] Miltiadis Allamanis and Charles Sutton. 2015. Why, when, and what: Analyzing Stack Overflow questions by topic, type, and code. In *12th Working Conference on Mining Software Repositories (MSR 2015)*, Massimiliano Di Penta, Martin Pinzger, and Romain Robbes (Eds.). IEEE Computer Society, Florence, Italy, 53–56.

[4] Le.. An, Ons Mlouki, Foutse Khomh, and Giuliano Antoniol. 2017. Stack Overflow: A Code Laundering Platform?. In *24th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2017)*, Martin Pinzger, Gabriele Bavota, and Andrian Marcus (Eds.). IEEE Computer Society, Klagenfurt, Austria, 283–293.

[5] Ali Sajedi Badashian, Afsaneh Esteki, Ameneh Gholipour, Abram Hindle, and Eleni Stroulia. 2014. Involvement, Contribution and Influence in GitHub and Stack Overflow. In *24th International Conference on Computer Science and Software Engineering (CASCON 2014)*, Joanna Ng, Jin Li, and Ken Wong (Eds.). IBM / ACM, Markham, ON, Canada, 19–33.

[6] Sebastian Baltes. 2018. SOTorrent: Reconstructing and Analyzing the Evolution of Stack Overflow Posts — Supplementary Material. (2018). http://doi.org/10.5281/zenodo.1201553

[7] Sebastian Baltes. 2018. sotorrent/db-scripts on GitHub. (2018). https://doi.org/10.5281/zenodo.1116346

[8] Sebastian Baltes. 2018. sotorrent/r-scripts on GitHub. (2018). https://doi.org/10.5281/zenodo.1048185

[9] Sebastian Baltes and Lorik Dumani. 2018. SOTorrent Data Set Version 2018-02-16. (2018). http://doi.org/10.5281/zenodo.1196296

[10] Sebastian Baltes and Lorik Dumani. 2018. sotorrent/metrics-comparison on GitHub. (2018). https://doi.org/10.5281/zenodo.1045823

[11] Sebastian Baltes and Lorik Dumani. 2018. sotorrent/so-posthistory-extractor on GitHub. (2018). https://doi.org/10.5281/zenodo.835046

[12] Sebastian Baltes and Lorik Dumani. 2018. sotorrent/string-similarity on GitHub. (2018). https://doi.org/10.5281/zenodo.835044

[13] Sebastian Baltes, Lorik Dumani, and Tobias Zeimetz. 2017. Dataset with manually validated version histories of Stack Overflow posts. (2017). http://doi.org/10.5281/zenodo.884909

[14] Sebastian Baltes, Richard Kiefer, and Stephan Diehl. 2017. Attribution required: Stack overflow code snippets in GitHub projects. In *39th International Conference on Software Engineering (ICSE 2017), Companion Volume*, Sebastián Uchitel, Alessandro Orso, and Martin P. Robillard (Eds.). IEEE Computer Society, Buenos Aires, Argentina, 161–163.

[15] Steven Burrows, Seyed M. M. Tahaghoghi, and Justin Zobel. 2007. Efficient plagiarism detection for large code repositories. *Software—Practice and Experience* 37, 2 (2007), 151–176.

[16] Brock Angus Campbell and Christoph Treude. 2017. NLP2Code: Code Snippet Content Assist via Natural Language Tasks. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME 2017)*, Hong Mei, Lu Zhang, and Thomas Zimmermann (Eds.). IEEE Computer Society, Shanghai, China, 628–632.

[17] Ned Chapin, Joanne E. Hale, Khaled Md Khan, Juan F. Ramil, and Wui-Gee Tan. 2001. Types of software evolution and software maintenance. *Journal of Software Maintenance* 13, 1 (2001), 3–30.

[18] Chunyang Chen, Zhenchang Xing, and Yang Liu. 2017. By the Community & For the Community: A Deep Learning Approach to Assist Collaborative Editing in Q&A Sites. *Proceedings of the ACM on Human-Computer Interaction* 1 (2017), 32:1–32:21.

[19] Davide Chicco. 2017. Ten quick tips for machine learning in computational biology. *BioData mining* 10, 1 (2017), 35.

[20] Jacob Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences* (2nd ed.). Routledge, Mahwah, NJ, USA.

[21] Jacob Cohen. 1992. A power primer. *Psychological bulletin* 112, 1 (1992), 155.

[22] Maarten Duijn, Adam Kucera, and Alberto Bacchelli. 2015. Quality Questions Need Quality Code: Classifying Code Fragments on Stack Overflow. In *12th Working Conference on Mining Software Repositories (MSR 2015)*, Massimiliano Di Penta, Martin Pinzger, and Romain Robbes (Eds.). IEEE Computer Society, Florence, Italy, 410–413.

[23] Lorik Dumani and Sebastian Baltes. 2017. sotorrent/so-posthistory-gt on GitHub. (2017). https://doi.org/10.5281/zenodo.1045935

[24] Zoran Duric and Dragan Gasevic. 2013. A source code similarity system for plagiarism detection. *The Computer Journal* 56, 1 (2013), 70–86.

[25] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. 2017. Stack Overflow Considered Harmful? The Impact of Copy&Paste on Android Application Security. In *2017 IEEE Symposium on Security and Privacy (S&P 2017)*, Kevin R. B. Butler, Úlfar Erlingsson, and Bryan Parno (Eds.). IEEE Computer Society, San Jose, CA, USA, 121–136.

[26] Daniel M. German, Massimiliano Di Penta, Yann-Gael Gueheneuc, and Giuliano Antoniol. 2009. Code siblings: Technical and legal implications of copying code between applications. In *6th International Working Conference on Mining Software Repositories (MSR 2009)*, Michael W. Godfrey and Jim Whitehead (Eds.). IEEE Computer Society, Vancouver, BC, Canada, 81–90.

[27] Mohammad Gharehyazie, Baishakhi Ray, and Vladimir Filkov. 2017. Some From Here, Some From There: Cross-Project Code Reuse in GitHub. In *14th International Conference on Mining Software Repositories (MSR 2017)*, Jesus M. Gonzalez-Barahona, Abram Hindle, and Lin Tan (Eds.). IEEE Computer Society, Buenos Aires, Argentina, 291–301.

[28] Robert D. Gibbons, Donald R. Hedeker, and John M. Davis. 1993. Estimation of effect size from a series of experiments involving paired comparisons. *Journal of Educational Statistics* 18, 3 (1993), 271–279.

[29] Michael W. Godfrey and Daniel M. German. 2008. The past, present, and future of software evolution. In *Frontiers of Software Maintenance (FoSM 2008)*, Hausi Muller, Scott Tilley, and Kenny Wong (Eds.). IEEE, Beijing, China, 129–138.

[30] Google Cloud Platform. 2018. GitHub Data. (2018). https://cloud.google.com/bigquery/public-data/github

[31] Georgios Gousios. 2013. The GHTorrent dataset and tool suite. In *10th International Working Conference on Mining Software Repositories (MSR 2013)*, Thomas Zimmermann, Massimiliano Di Penta, and Sunghun Kim (Eds.). IEEE, San Francisco, CA, USA, 233–236.

[32] Dennis E. Hinkle, William Wiersma, and Stephen G. Jurs. 1979. *Applied statistics for the behavioral sciences.* Rand McNally College Publishing, Skokie, IL, USA.

[33] Thomas Lancaster and Fintan Culwin. 2004. A comparison of source code plagiarism detection engines. *Computer Science Education* 14, 2 (2004), 101–112.

[34] Meir M. Lehman. 1980. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE* 68, 9 (1980), 1060–1076.

[35] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. 2008. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA.

[36] Vitor T. Martins, Daniela Fonte, Pedro Rangel Henriques, and Daniela da Cruz. 2014. Plagiarism Detection: A Tool Survey and Comparison. In *3rd Symposium on Languages, Applications and Technologies (SLATE 2014) (OpenAccess Series in Informatics (OASIcs))*, Maria João Varanda Pereira, José Paulo Leal, and Alberto Simoes (Eds.), Vol. 38. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Bragança, Portugal, 143–158.

[37] Brian W. Matthews. 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) – Protein Structure* 405, 2 (1975), 442–451.

[38] Tom Mens and Serge Demeyer (Eds.). 2008. *Software Evolution.* Springer, Berlin, Germany.

[39] Patrick Morrison and Emerson Murphy-Hill. 2015. Is programming knowledge related to age? An exploration of Stack Overflow. In *12th Working Conference on Mining Software Repositories (MSR 2015)*, Massimiliano Di Penta, Martin Pinzger, and Romain Robbes (Eds.). IEEE Computer Society, Florence, Italy, 69–72.

[40] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. 2012. What makes a good code example? A study of programming Q&A in StackOverflow. In *28th IEEE International Conference on Software Maintenance (ICSM 2012)*, Paolo Tonella, Massimiliano Di Penta, and Jonathan I. Maletic (Eds.). IEEE Computer Society, Trento, Italy, 25–34.

[41] Luca Ponzanelli, Alberto Bacchelli, and Michele Lanza. 2013. Seahawk: Stack Overflow in the IDE. In *35th International Conference on Software Engineering (ICSE 2013)*, David Notkin, Betty H. C. Cheng, and Klaus Pohl (Eds.). IEEE Computer Society, San Francisco, CA, USA, 1295–1298.

[42] Luca Ponzanelli, Andrea Mocci, Alberto Bacchelli, and Michele Lanza. 2014. Understanding and classifying the quality of technical forum questions. In *14th International Conference on Quality Software (QSIC 2014)*, W. Eric Wong and Bruce McMillin (Eds.). IEEE, Allen, TX, USA, 343–352.

[43] David Martin Powers. 2011. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies* 2, 1 (2011), 37–63.

[44] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. 2009. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming* 74, 7 (2009), 470–495.

[45] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. 2003. Winnowing: Local algorithms for document fingerprinting. In *2003 ACM SIGMOD International Conference on Management of Data (SIGMOD 2003)*, Alon Y. Halevy, Zachary G. Ives, and AnHai Doan (Eds.). ACM, San Diego, CA, USA, 76–85.

[46] Giuseppe Silvestri, Jie Yang, Alessandro Bozzon, and Andrea Tagarelli. 2015. Linking Accounts across Social Networks: The Case of StackOverflow, GitHub and Twitter. In *1st International Workshop on Knowledge Discovery on the WEB (KDWeb 2015) (CEUR Workshop Proceedings)*, Giuliano Armano, Alessandro Bozzon, and Alessandro Giuliani (Eds.). CEUR-WS.org, Cagliari, Italy, 41–52.

[47] Charles Spearman. 1904. The proof and measurement of association between two things. *American Journal of Psychology* 15, 1 (1904), 72–101.

[48] Stack Exchange Community Wiki. 2018-02-27. Database schema documentation for the public data dump and SEDE. (2018-02-27). https://meta.stackexchange.com/a/2678

[49] Stack Exchange Inc. 2017. Stack Exchange Data Dump 2017-12-01. (2017). https://archive.org/details/stackexchange/

[50] Stack Exchange Inc. 2018. Markdown help. (2018). https://stackoverflow.com/editing-help

[51] Siddharth Subramanian and Reid Holmes. 2015. Making sense of online code snippets. In *12th Working Conference on Mining Software Repositories (MSR 2015)*, Massimiliano Di Penta, Martin Pinzger, and Romain Robbes (Eds.). IEEE Computer Society, Florence, Italy, 85–88.

[52] Christoph Treude, Ohad Barzilay, and Margaret-Anne D. Storey. 2011. How do programmers ask and answer questions on the web?. In *33rd International Conference on Software Engineering (ICSE 2011)*, Richard N. Taylor, Harald C. Gall, and Nenad Medvidovic (Eds.). ACM, Waikiki, Honolulu, 804–807.

[53] Christoph Treude and Martin P. Robillard. 2016. Augmenting API Documentation with Insights from Stack Overflow. In *38th International Conference on Software Engineering (ICSE 2016)*, Laura Dillon, Willem Visser, and Laurie Williams (Eds.). ACM, Austin, TX, USA, 392–403.

[54] Christoph Treude and Martin P. Robillard. 2017. Understanding Stack Overflow Code Fragments. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME 2017)*, Hong Mei, Lu Zhang, and Thomas Zimmermann (Eds.). IEEE Computer Society, Shanghai, China, 509–513.

[55] Bogdan Vasilescu, Andrea Capiluppi, and Alexander Serebrenik. 2012. Gender, Representation and Online Participation: A Quantitative Study of StackOverflow. In *4th International Conference on Social Informatics (SocInfo 2012) (Lecture Notes in Computer Science)*, Karl Aberer, Andreas Flache, Wander Jager, Ling Liu, Jie Tang, and Christophe Gueret (Eds.). Springer, Lausanne, Switzerland, 332–338.

[56] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. 2013. StackOverflow and GitHub: Associations between Software Development and Crowdsourced Knowledge. In *2013 International Conference on Social Computing (SocialCom 2013)*, L. W. Chang, Jaideep Srivastava, and Justin Zhan (Eds.). IEEE Computer Society, Washington, DC, USA, 188–195.

[57] Shaowei Wang, Lo David, and Lingxiao Jiang. 2013. An empirical study on developer interactions in StackOverflow. In *28th Annual ACM Symposium on Applied Computing (SAC 2013)*, Sung Y. Shin and José Carlos Maldonado (Eds.). ACM, Coimbra, Portugal, 1019–1024.

[58] Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics* 1, 6 (1945), 80–83.

[59] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E. Hassan, and Zhenchang Xing. 2017. What do developers search for on the web? *Empirical Software Engineering* 22, 6 (2017), 3149–3185.

[60] Di. Yang, Aftab Hussain, and Cristina Videira Lopes. 2016. From Query to Usable Code: An Analysis of Stack Overflow Code Snippets. In *13th International Conference on Mining Software Repositories (MSR 2016)*, Miryung Kim, Romain Robbes, and Christian Bird (Eds.). ACM, Austin, TX, USA, 391–402.

[61] Di. Yang, Pedro Martins, Vaibhav Saini, and Cristina V. Lopes. 2017. Stack Overflow in Github: Any Snippets There?. In *14th International Conference on Mining Software Repositories (MSR 2017)*, Jesus M. Gonzalez-Barahona, Abram Hindle, and Lin Tan (Eds.). IEEE Computer Society, Buenos Aires, Argentina, 280–290.

[62] Jie Yang, Claudia Hauff, Alessandro Bozzon, and Geert-Jan Houben. 2014. Asking the right question in collaborative Q&A systems. In *25th ACM Conference on Hypertext and Social Media (HT 2014)*, Leo Ferres, Gustavo Rossi, Virgilio A. F. Almeida, and Eelco Herder (Eds.). ACM, Santiago, Chile, 179–189.