An Annotated Dataset of Stack Overflow Post Edits

Sebastian Baltes
sebastian.baltes@adelaide.edu.au
University of Adelaide
Adelaide, South Australia, Australia

Markus Wagner
markus.wagner@adelaide.edu.au
University of Adelaide
Adelaide, South Australia, Australia

ABSTRACT

To improve software engineering, software repositories have been mined for code snippets and bug fixes. Typically, this mining takes place at the level of files or commits. To be able to dig deeper and to extract insights at a higher resolution, we hereby present an annotated dataset that contains over 7 million edits of code and text on Stack Overflow. Our preliminary study indicates that these edits might be a treasure trove for mining information about fine-grained patches, e.g., for the optimisation of non-functional properties.

CCS CONCEPTS

• **Software and its engineering → Search-based software engineering**; **Software maintenance tools**.

KEYWORDS

Software documentation, software evolution, patches, mining software repositories, stack overflow

ACM Reference Format:
Sebastian Baltes and Markus Wagner. 2020. An Annotated Dataset of Stack Overflow Post Edits. In *Proceedings of* . ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 MOTIVATION

Data-Driven Search-Based Software Engineering (DSE) [9] combines insights from Mining Software Repositories (MSR) and Search-based Software Engineering (SBSE). While MSR formulates software engineering problems as data mining problems, SBSE reformulates SE problems as optimisation problems and use meta-heuristic algorithms to solve them. Both MSR and SBSE share the common goal of providing insights to improve software engineering. In this present paper, we suggest to improve software engineering – in particular the search for code and text patches – by mining the edit histories of Stack Overflow posts.

We are, of course, not the first to propose to mine corpora for patches. In particular in the field of automated program repair [3], this has been a popular approach. For example, development histories of Eclipse JDT have been mined to find bug-fixing patches [4], and so have been GitHub projects [6, 7], but again for fixing bugs. Moving on from bug fixing to the optimisation of non-functional properties, Petke [10] suggested in 2017:

> "[...] to mine changes made by software developers [...] with particular focus on improvement of the software property of interest, such as runtime efficiency. The results can then be used to devise new mutation operators in the form of templates. [...]"

Interestingly, we are not aware of any work towards this, and one reason for this might be the scarcity of data. Having said this, there has been the work of Moura et al. [8], who focused on mining energy-saving commits from GitHub that "had the explicit intention of saving energy". They manually interpreted 290 commits and presented among other two code transformations for very specialised cases that might be translatable into patches for automatic search. While said article has been cited 28 times at the time of writing, none of the citing works and none of the other works that we are aware of had a broader focus on non-functional properties in general.

With this article, we present a dataset of annotated Stack Overflow post edits for future extraction of patches. We are of the opinion that the edits of Stack Overflow posts are by nature more fine-grained than, e.g., commits in Github repositories, and hence they are more amenable to the extraction of patches that can result in new mutation operators – we would go as far to saying that this is because Stack Overflow post edits are less formal (due to the forum-like style of the platform) than a software repository commit where each commit is, e.g., expected to fix a bug or to extend functionality.[1]

Before we can start to extract patches, we first need to set the expectations by characterising the dataset. As we have a particular interest in code-optimisation, e.g., for the purpose of genetic improvement of software [5, 11], this greatly influences our research questions:

(1) **RQ1:** Which aspects do Stack Overflow users mention in their edit comments?
(2) **RQ2:** Which non-functional properties do users reference in edit comments?

2 STACK OVERFLOW POST EDITS

To derive Stack Overflow code edits, we utilised the *SOTorrent* dataset that we developed and maintain [1]. Based on version *2020-01-24* of the dataset we retrieved all 7,459,778 post edits where the user provided an (optional) description of the edit. Those edits, which are not limited to a particular programming language, are the foundation of our dataset [2]. Of all edits, 1,305,323 (17.5%) modified only a code block, 4,792,777 (64.2%) only a text block, and 1,361,678 (18.3%) both text and code blocks of a particular Stack Overflow post – Figure 1 shows an example of a typical post.

[1] As anecdotal evidence, we point at the two discussions https://softwareengineering.stackexchange.com/q/74764/ and https://stackoverflow.com/q/107264 which have been viewed over 70k at the time of writing; accessed on 14 April 2020.

, ,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Figure 1: Example of a Stack Overflow post that contains text and code blocks and that has been edited after its initial publication. Source: https://stackoverflow.com/a/23480549, accessed on 17 April 2020.

In a next step, we normalised the edit messages by converting all characters to lower case and by replacing all whitespace sequences with a single space character. This yielded 3,291,268 unique edit messages. We then ranked the edit messages according to their frequency. Starting with the most frequent messages, we manually extracted characteristic keywords to build regular expressions matching similar messages. We stopped the manual analysis as soon as we were able to cluster all messages with at least 1,000 occurrences. After this process, we were able to assign edit messages to 25 categories using customised regular expressions. Thirteen categories were related to actions that the user performed (*adding, updating, deleting, fixing, improving, clarifying, simplifying, explaining, editing, copy-editing, active reading, refactoring*), eleven were nouns related to the target of the edit (*formatting, typo, grammar, spelling, code, bug, link, image, example, syntax, solution, tag*). One category was more on a meta-level and captures *sarcasm* in edit messages. To provide an example, we present the final regular expression that we used to match edits *adding* a certain information:

```
.*\\b((add|expand|more|extend)[a-z0-9_-]*).*
```

Please note that one edit can belong to more than one of these edit categories. Overall, we were able at assign 6,704,541 of the 7,459,778 edits (89.9%) to at least one category (see Figure 2). We provide the our retrieval and analysis scripts as part of our dataset [1].

Figure 2 shows the results of the above-mentioned annotation process, thus answering **RQ1**. Most post edits are concerned with *formatting* or *additions*. Many edits referred to *fixing* and to *code*, which indicates that our dataset is indeed suitable for mining patches. While the term *bug* was less frequently mentioned, it still occurred in 24,775 edit messages.

In the following, to explore the potential for patch-extraction, we focus on post edits that only edited code blocks, because this allows us to unambiguously link the edit message to the code edit. Of the 1,305,323 edits that only modified code blocks, 933,340 (71.5%) were assigned to at least one of the 25 categories. Figure 3 shows the

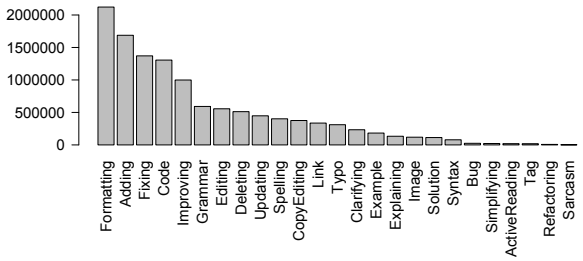

Figure 2: Number of edits assigned to the 25 categories we iteratively derived (n=6,704,541).

pair	count
formatting, code	152,721
improving, formatting	98,339
fixing, code	76,026
fixing, formatting	65,544
adding, code	50,795
fixing, typo	32,711
improving, code	31,463
editing, code	28,844
updating, code	24,910
deleting, code	20,106

Table 1: Top 10 pairs of tags (pairs ordered only for presentation purposes).

corresponding distribution. As we can see, about one third of these edits are about *formatting* and *code*, but *fixing* and *adding* were also frequently mentioned. Overall, the edits that are interesting for us are the hundreds of thousands of edits that are concerned with *fixing*, *editing*, *updating*, or *improving*.

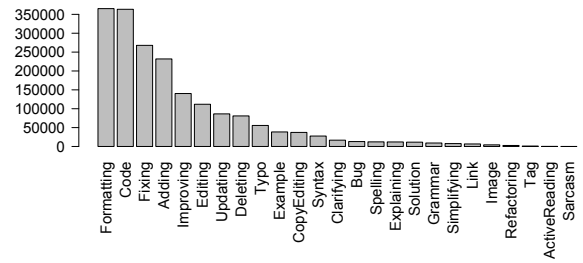

Figure 3: Number of code edits assigned to the 25 categories we iteratively derived (n= 933,340).

Next, we dig a little deeper. In Table 1, we list the most frequent pairs of tags (of edits that were assigned to at least two categories). While a large proportion of the post edits is about the *formatting* if *code*, over 230,000 edits possibly target the improvement of a code aspect of a post, i.e., by *fixing*, *adding*, *improving*, *editing*, *updating*, or *deleting* code.

property	count
Performance	2,658
Size	2,284
Memory	1,084
Energy	10

Table 2: Number of code edits where the user mentioned one of the four non-functional properties we have considered (n=7,024).

Finally, we focus on edits targeting non-functional properties, as this appears to be an area that is currently under-explored in the literature. While non-functional properties were not among the most frequently mentioned terms that formed the 25 categories we used to tag edits, we want to show that our dataset nevertheless contains edits related to such properties. We derived four categories capturing non-functional properties based on our own judgement and information taken from the Wikipedia page on non-functional requirements.[2] We again built custom regular expressions to match edits related to these categories (see scripts attached to our dataset [1]). Table 2 shows the results of applying the regular expressions to the edit messages of all code-only edits, thus answering **RQ2**. We found a few thousand edits that appear to target non-functional properties such as *performance* and *memory*. Interestingly, the small number of edits targeting *energy* is in line with the small number of commits that the aforementioned research [8] has been able to find.

3 EXAMPLES

Our dataset can either be downloaded from Zenodo [1] as a CSV file or accessed via Google BigQuery.[3] The table `PostEdits`, which we provide as part of our dataset, gives researchers access to all 7,459,778 post edits extracted from the *SOTorrent* dataset, identified by their `PostHistoryId`. We further provide the edit messages and binary flags for the categories mentioned throughout this paper, which can be used to filter the edits. The table can be joined with table `PostBlockVersion` of the *SOTorrent* dataset to retrieve the content of the modified text and code blocks before and after the edits. A corresponding query is attached to our dataset [1].

As a first investigation to explore the potential of our dataset, we have manually explored the subset of the code-block edits that we had tagged as being *performance*-related. For this proof-of-concept, a total of 15 minutes was spent on the exploration of the edits and on the assessment of the respective Stack Overflow posts. Among others, we have found the following edits:[4]

(1) *"using john saunders tip for more performance"* (23481309): the edit replaced a String with a StringBuilder (see Figure 1).
(2) *"added debounce to improve performance when app scales"* (44000037): the edit added a JavaScript debounce function.
(3) *"evaluating x 0 first solves for type errors and gives better performance than if"* (19400435): the edit updated an if-statement – interestingly, there is a brief discussion on the performance attached to this post.
(4) *"some small performance improvements always a good idea to have a fast primality test"* (8539774): the edit added a few hard-coded scenarios for a particular problem.
(5) *"Improved performance, by getting [...] outside the loop"* (11535593): the edit lifted code outside of a loop, which is an approach that is commonly taught in undergraduate courses.

[2]https://en.wikipedia.org/w/index.php?title=Non-functional_requirement&oldid=947189406, accessed on 13 April 2020
[3]https://bigquery.cloud.google.com/table/sotorrent-org:2020_01_24_edits.PostEdits
[4]structure: edit comment from the post editor (post ID): our interpretation

4 OUTLOOK

The particular value of our dataset is that the post edits are most likely much smaller in scope and much more fine-grained than, e.g., commits in project repositories. With his hypothesis in mind, it might be possible to reveal insights on software engineering in practice at a higher resolution. Moreover, our preliminary study indicates that the Stack Overflow edits might be a treasure trove for manually mining information about fine-grained code patches, e.g., for the optimisation of non-functional properties. Lastly, while our focus has been on code edits, we envision that our dataset – which also contains text edits and their comments – can be of use for mining other insights as well, including typical grammar fixes or frequent formatting improvements.

We are open for feedback from the community on potential improvements to the dataset and we are happy to provide support for researcher who want to use or adapt our data.

REFERENCES

[1] Sebastian Baltes, Lorik Dumani, Christoph Treude, and Stephan Diehl. 2018. SOTorrent: reconstructing and analyzing the evolution of stack overflow posts. In *Proceedings of the 15th Int. Conf. on Mining Software Repositories* (Gothenburg, Sweden) *(MSR '18)*. ACM, 319–330.
[2] Sebastian Baltes and Markus Wagner. 2020. *An Annotated Dataset of Stack Overflow Post Edits.* https://doi.org/10.5281/zenodo.3754159
[3] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. 2019. Automated program repair. *Commun. ACM* 62, 12 (2019), 56–65.
[4] Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim. 2013. Automatic Patch Generation Learned from Human-Written Patches. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, USA) *(ICSE '13)*. IEEE Press, 802–811.
[5] William B. Langdon. 2015. Genetically Improved Software. In *Handbook of Genetic Programming Applications*. Springer.
[6] X. B. D. Le, D. Lo, and C. L. Goues. 2016. History Driven Program Repair. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 213–224.
[7] Fan Long, Peter Amidon, and Martin Rinard. 2017. Automatic Inference of Code Transforms for Patch Generation. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Klagenfurt, Austria) *(ESEC/FSE 2017)*. ACM, 727–739.
[8] Irineu Moura, Gustavo Pinto, Felipe Ebert, and Fernando Castor. 2015. Mining Energy-Aware Commits. In *Proceedings of the 12th Working Conference on Mining Software Repositories* (Florence, Italy) *(MSR '15)*. IEEE Press, 56–67.
[9] Vivek Nair, Amritanshu Agrawal, Jianfeng Chen, Wei Fu, George Mathew, Tim Menzies, Leandro Minku, Markus Wagner, and Zhe Yu. 2018. Data-driven Search-based Software Engineering. In *Proceedings of the 15th Int. Conf. on Mining Software Repositories* (Gothenburg, Sweden) *(MSR '18)*. ACM, 341–352.
[10] Justyna Petke. 2017. New Operators for Non-Functional Genetic Improvement. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17)*. ACM, 1541–1542.
[11] Justyna Petke, Saemundur O. Haraldsson, Mark Harman, William B. Langdon, David R. White, and John R. Woodward. 2018. Genetic Improvement of Software: a Comprehensive Survey. *Transactions on Evolutionary Computation* 22, 3 (June 2018), 415–432.