2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)

Cleaning StackOverflow for Machine Translation

Musfiqur Rahman, Peter C Rigby, Dharani Palani
Department of Computer Science and Software Engineering
Concordia University, Montréal, Canada
first.last@concordia.ca

Tien Nguyen
Department of Computer Science
The University of Texas at Dallas
tien.n.nguyen@utdallas.edu

***Abstract*—Generating source code API sequences from an English query using Machine Translation (MT) has gained much interest in recent years. For any kind of MT, the model needs to be trained on a parallel corpus. In this paper we clean StackOverflow, one of the most popular online discussion forums for programmers, to generate a parallel English-Code corpus from Android posts. We contrast three data cleaning approaches: standard NLP, title only, and software task extraction. We evaluate the quality of the each corpus for MT. To provide indicators of how useful each corpus will be for machine translation, we provide researchers with measurements of the corpus size, percentage of unique tokens, and per-word maximum likelihood alignment entropy. We have used these corpus cleaning approaches to translate between English and Code [22, 23], to compare existing SMT approaches from word mapping to neural networks [24], and to re-examine the "natural software" hypothesis [29]. After cleaning and aligning the data, we create a simple maximum likelihood MT model to show that English words in the corpus map to a small number of specific code elements. This model provides a basis for the success of using StackOverflow for search and other tasks in the software engineering literature and paves the way for MT. Our scripts and corpora are publicly available on GitHub [1] as well as at https://search.datacite.org/works/10.5281/zenodo.2558551.**

I. Introduction

The process of translating between two languages automatically is known as Machine Translation (MT). Recent advances and computational power have increased the popularity of MT. MT techniques can be broadly classified into three classes: Statistical Machine Translation (SMT)[2, 16, 19], Example-based Machine Translation (EBMT)[33], and Neural Machine Translation (NMT)[3, 21]. Application of MT algorithms is not limited to translation between natural languages. In recent years, MT techniques have been used to translate from natural language to programming languages[10, 22, 23, 24, 28, 40]. Although MT approaches differ in terms of theory, algorithms, and efficiently, all approaches require a high volume and low noise *parallel corpus*[5, 15] or *bitext*[11]. The goal of this work is to clean the StackOverflow developer question and answer form, as a corpus for future use in MT.

StackOverflow is a rich, bilingual corpus because it discusses programming in both English and code. For example, in Figure 1 we see a post the answers in both English and code the question "How can I refresh the cursor from a CursorLoader?" English words in the post, such as "data will be discarded" can be aligned with code elements such as `restartLoader()`. These alignments can then be used in machine translation. Unfortunately, StackOverflow posts are noisy because people write posts in an informal manner. Examples of noise in StackOverflow includes incorrect spelling, inappropriate use of punctuation, use of acronyms without elaboration, and grammatical mistakes. This results in a degradation of the quality of corpus texts. We found while applying existing SMT such as Phrase based MT and Recurrent Neural Network (RNN) based MT that the noise reduced the quality of translation [24].

Removing the noise from the StackOverflow data without any significant loss of relevant information is challenging. As a result, we spent substantial time and effort cleaning and evaluating the quality of various cleaning approaches. In this paper, we experiment with different data cleaning techniques ranging from general Natural Language Processing (NLP)[13] to Software Engineering specific techniques[35], and determine which techniques yield a corpus with a high potential for MT tasks. We hope others will use our cleaned corpus and we make our scripts and data publicly available [1]. This corpus has great promise and we have used it in T2API to translate from textual descriptions of code to API graph usages [22, 23], to evaluate existing SMT approaches including neural networks [24], and, in basic research, to examine the impact of code and English representations on their repetitiveness or "naturalness" [29].

A. Data Cleaning Approaches

We prepare Android posts on StackOverflow using the following four cleaning approaches (C0 to C3):

C0: Raw data

We prepare this corpus extracting raw text and source code from StackOverflow posts. We do not perform any kind of processing on the English text so this serves as a baseline corpus for comparison purposes. We extract code elements using Rigby and Robillard's [30] publicly available *ACE* tool, which was built for extracting code elements from freeform StackOverflow text. The alignment is at the post level with the English being aligned with the extracted code elements.

C1: Thread title

Many researchers report that StackOverflow thread titles contain valuable information in a brief yet precise manner [7, 26, 32]. We prepare the English corpus with StackOverflow titles only. We remove stopwords and stem the text. For the code corpus we extract code from only the accepted answer posts using *ACE*.

C2: Standard NLP

For this corpus, we use NLP including keyword extraction, stopword removal, and stemming [31]. For code extraction we use *ACE*.

C3: Software engineering task extraction

Treude *et al.* developed *TaskNavigator* [35] for extracting development tasks for software documentation. We use this tool to extract tasks from StackOverflow posts and use

2574-3864/19/$31.00 ©2019 IEEE
DOI 10.1109/MSR.2019.00021

these tasks as our English corpus after stemming and removing stopwords. We extract code elements from the posts using *ACE*.

B. Evaluation Metrics and Criteria

After preparing the corpora we evaluate each corpus to provide researchers with an indication of how useful it may be for various tasks using the following two metrics.

E1: Size of corpus

All statistical MT requires a large corpus. The cleaning process will eliminate text, code, and entire posts that are noisy. We measure the size of each corpus in terms of number of posts, number of English words, number of code elements, and frequency of element code usage.

E2: Per-Word Maximum likelihood Alignment Entropy

To evaluate the relative potential of each cleaned corpus for MT, we train a simple maximum likelihood alignment model for each English word to code elements. For each corpus we measure the distribution of per-word alignment entropy between the English and code. Low entropy indicates that the expectation maximization algorithm has converged to a limited, precise set of mappings [16]. If there is no convergence, it will be difficult to create a good MT model. This simple measure of entropy indicates the future potential of a corpus for use in more complex MT models.

The rest of the paper is structured as follows. In Section II we detail our data and data cleaning approaches. In Section III we evaluate each corpus for MT. In Section IV, we conclude the paper by discussing the lessons learned and the potential for future work.

II. Corpus Cleaning

StackOverflow is a popular Q&A forum where developers discuss programming issues ranging from how to solve a problem in a particular language to the best programming practices [20]. One of the key features of StackOverflow which makes it suitable as bilingual parallel corpus is the presence of both the textual description of how to solve a problem and code showing how to implement the solution.

Figure 1 shows an example of a typical StackOverflow post. The post discusses refreshing the cursor from CursorLoader in an Android application. A step-by-step solution has been described in English which is followed by a sample code snippet showing how to implement the solution. The solution is described in both English and in code making StackOverflow a potential English-code parallel corpus.

In this work, we process posts that are tagged with "android" between September 2011 to September 2016.[1] We perform corpus preparation in multiple experimental settings. The outcome of each setting is then processed by an n-gram language model and a word-based aligner to determine the per-word alignment entropy for each corpus.

A. C0: Raw data approach

We create a corpus with raw data extracted from StackOverflow. For each post in our English corpus we take the title of the thread and the text from the post body. We remove

[1]Depending on the research questions, the researcher can process the latest StackOverflow dump with the provided scripts.

You just need to move your code around a little and call `restartLoader`. That is,

1. When the user clicks a list item, you somehow change the private instance variable that is returned in `getChosenDate()` (I am being intentionally vague here because you didn't specify what exactly `getChosenDate()` returns).
2. Once the change is made, call `getLoaderManager().restartLoader()` on your `Loader`. Your old data will be discarded and `restartLoader` will trigger `onCreateLoader` to be called again. This time around, `getChosenDate()` will return a different value (depending on the list item that was clicked) and that should do it. Your `onCreateLoader` implementation should look something like this:

```
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    Uri baseUri = SmartCalProvider.CONTENT_URI;

    makeProviderBundle(
        new String[] { "_id, event_name, start_date, start_time, end_date, end_time, loca
        "date(?) >= start_date and date(?) <= end_date",
        new String[]{ getChosenDate(), getChosenDate() },
        null);

    return new CursorLoader(
            this,
            baseUri,
            args.getStringArray("projection"),
            args.getString("selection"),
            args.getStringArray("selectionArgs"),
            args.getBoolean("sortOrder") ? args.getString("sortOrder") : null);
}
```

Fig. 1: An example of a StackOverflow post that answers in both English and code the question, "How can I refresh the cursor from a CursorLoader?" English, such as "data will be discarded", can be aligned with code elements, such as `restartLoader()`, for use in MT.
Post available at: *https://stackoverflow.com/a/11092861/1055441*

the code elements and code snippets from the body. We extract the code element for the code using *ACE*. After filtering out posts that do not contain both English and code we have a corpus of 149,476 Android posts.

B. C1: Thread title approach

Rosen *et al.* [32] state that "titles summarize and identify the main concepts being asked in the post." The title is brief and in contrast to the entire post contains little noise. In this experimental setting we align the English in the title with the code elements that are contained in positively voted answer posts. We apply the following additional criteria:

- Stopwords are removed from the title.
- The title text is stemmed.
- There are at least three and at most twenty code elements extracted from the answer post.

For example, the title of the post in Figure 1 is "How can I refresh the cursor from a CursorLoader?" For this post extracted English and code will be as follows:

- English: refresh cursor CursorLoader
- Code: Bundle.getString, Uri.baseUri, SmartCallProvider.CONTENT_URI, Bundle.getString, Bundle.getBoolean, String.getLoaderManager, String.getLoaderManager.restartLoader, LoaderCallbacks.onCreateLoader

The final corpus comprises of 106,359 question titles and corresponding posts.

C. C2: Standard NLP approach

For this corpus, we extract English keywords from the StackOverflow title and post body. The English is processed by RAKE[31] which is a general purpose keyword extractor. For our code corpus we use ACE[30] to extract the *unique*

code elements present in the post body. Code elements can be either inside a code snippet or can be embedded within free-form English.

Rapid Automatic Keyword Extraction (RAKE): RAKE splits a text into word groups separated by sentence separators or words from a provided list of stopwords. For our implementation we use the list of English stopwords provided in Python's NLTK[18] library. Each of the resulting word groups is a keyword candidate. In the next step the algorithm scores each keyword according to the word co-occurrence graph. The word co-occurrence graph assigns the frequency of unique word-pairs across the set of candidate keywords. The details of the algorithm can be found in[31]. The stopword list plays an important role in the effectiveness of the RAKE. Some works use manually curated stopwords list for domain specific application. An example is [14] in which the authors initially carried out their experiments with standard information retrieval stopwords list and report that this yielded poor results for the domain specific case of Polish legal texts. They further report that this approach generated a lot of very long keywords, containing many words that are not very informative. To deal with this issue they create their own list of stopwords and achieved better result. However, in our experiment we deal with this problem in a different way. As mentioned before we do not curate any stopwords lists rather we go with the stopwords provided by NLTK. Besides observing the issue of long keywords containing 'uninformative' words, we also observe other issues associated with the software engineering keywords extracted by RAKE. We briefly discuss these issues here.

- Keywords with long sequence of tokens containing 'uninformative' words.
- Presence of stopwords in the multi-token keywords.
- Association of high scores with keywords having very long sequence of tokens.

To deal with these issues we add an additional level of filtering after the keywords are returned by RAKE. This filtering includes constraining the length of keywords and the associated score. It also includes a stopword removal performed on the extracted keywords to remove any stopwords that may be embedded within the keywords. On the code side of the parallel corpus we add the constraint of having at least 3 code elements extracted by ACE. Steps/constraints related to the final level of filtering are as follows:

1) Keywords must be of length between 1 and 4 in terms of number of tokens to be accepted as a valid keyword.
2) The score associated with a keyword must be greater than 5 and less than 50.
3) There must be at least 3 code elements extracted from the post.
4) Stopwords are removed for the extracted keywords.
5) Stemming is performed using Porter Stemmer [27].

After performing all these five steps (first three steps for constraints checking and last two for post-processing) the extracted keywords and extracted *unique* code elements become a part of our *Keyword-Code* parallel corpus. For the same example in Figure 1 the following English (without stemming) and code pair is returned.

- English: move code, private instance variable, user clicks, list item, old data, different value, discarded, trigger, cursor, refresh
- Code: Bundle.getString, Uri.baseUri, SmartCallProvider.CONTENT_URI, Bundle.getString, Bundle.getBoolean, String.getLoaderManager, String.getLoaderManager.restartLoader, LoaderCallbacks.onCreateLoader

After applying these constraints this corpus contains 130,988 Android posts.

D. C3: Software engineering task extraction

The previous approaches do not consider software engineering specific keywords and concepts. We use Treude *et al.*'s [35] software engineering task extractor to provide concise descriptions of the tasks contained in STACKOVERFLOW posts. The *TaskNavigator*[35] extracts development tasks from documentation based on syntactic dependencies in the sentences. Verbs associated with a direct object and/or a prepositional phrase are identified as tasks. They consider four syntactic dependencies for extracting tasks: **direct object**, **prepositional modifier**, **passive nominal subject**, and **relative clause modifier**. Definitions of the these grammatical dependencies are available in[9] and the implementation details along with the performance of *TaskNavigator* are available in[35].

TaskNavigator was designed to extract development tasks from *formal documentation.* However, STACKOVERFLOW is an informal and noisy discussion forum. Unlike formal documentation where the text is grammatically correct with precise descriptions, STACKOVERFLOW posts contain additional grammatical errors and other erroneous task information. For example, posters tend to give context before giving the precise step by step solution. This results in *TaskNavigator* identifying incorrect and irrelevant tasks because it only examines sentence structure and the associated dependencies.

In examining these false tasks, we observed true tasks tend to contain a code element in the sentence. For example, short tasks with fewer than three tokens are usually false positives unless one of the tokens is a code element. Therefore, we remove extracted tasks if they contain fewer than three tokens and no code element. The following five steps are an additional filtering on *TaskNavigator*:

1) Stopword removal and stemming on the extracted tasks.
2) At least three tasks must be extracted from a post.
3) There must be at least two tokens after stopword removal and one of the tokens has to be a code element.
4) There has to be at least three code elements extracted from the post.

For the example shown in Figure 1 the following English-code alignment attained:

- English: refresh cursor, move code around, change private instance variable, discard old data, return different value.
- Code: Bundle.getString, Uri.baseUri, SmartCallProvider.CONTENT_URI, Bundle.getString, Bundle.getBoolean, String.getLoaderManager, String.getLoaderManager.restartLoader, LoaderCallbacks.onCreateLoader

The final corpus contains 110,009 posts.

TABLE I: Simple size measures of the corpora

Corpus	Posts	Unique English	Unique code	Median code usage
Raw	149,476	46,067	30,188	3
Title	106,359	10,477	12,227	6
Standard NLP	130,988	15,266	15,949	6
Task	110,009	9,738	14,971	6

III. Evaluation

One of the limitations of MT is that there is no straightforward mechanism to judge the quality of the corpus. In order to determine how well a corpus performs one has to train a model on the corpus and test its performance. However, for any MT system training the model takes a considerable amount of time. For example, it took 6 days to train a recurrent neural network (RNN) MT model on our raw corpus on a Google cloud virtual machine with 4 Nvidia GPUs each having 12GB of memory and 2496 processor cores [24]. We found that the translation quality was poor because the corpus contained many noisy alignments. Our goal is to provide simple and computationally inexpensive evaluations of a corpus for use in MT.

A. E1. Size of corpus

Our first evaluation criterion is size of the corpora. Statistical learning requires a large number of aligned English and code posts. Table I shows that number of posts, unique English tokens, unique code elements, and the number of times each code element has been used. For our calculation we consider tokens that occur more than once. We see that the raw corpus has the largest size and the largest number of unique tokens. However, to do prediction, we need repetitive use of tokens to identify pattern and we can see that most code elements are only used once. In contrast, the other approaches contain similar numbers of tokens and the median usage of code is two. From simple size measures, we can conclude that the raw corpus likely still contains too much noise, however, the other corpora are difficult to differentiate purely based on size.

B. E2: Per-Word Alignment Entropy

We create simple maximum-likelihood machine translation models [16]. Each English word is aligned to one or more code elements based on maximum-likelihood. The lower the alignment entropy for each English word the stronger the mapping to the specific code elements and the better the model. We plot the per-word alignment distribution in Figure 2. A box plot, is also contained within the distribution with the bottom and top of the box showing the 25th and 75th percentiles, respectively. The vertical line shows the median.

Although all the distribution are highly left skewed indicating that each English word maps to a limited set of code elements, there are outlier English words that map to a large number of possible elements. The raw corpus has the highest median entropy indicating that each English word maps imprecisely to many code elements. The median values for the raw, title, standard NLP, and SE task corpora are 0.69, 0.00, 0.09, and 0.38, respectively. The corresponding values for the 75th percentile are 1.66, 0.84, 1.06, .88, respectively. For the title and standard NLP corpora we see that most English words map with a high probability to few code elements. The SE tasks shows a higher median value but a lower 75 percentile than the standard NLP corpus. This distribution shows a consist mapping that allows for more diversity compared to the narrow focus of the title corpus. The narrow focus of the title corpus may be appropriate for a code summary, while the diversity of SE task corpus would allow the researcher to translate more complex developer tasks.

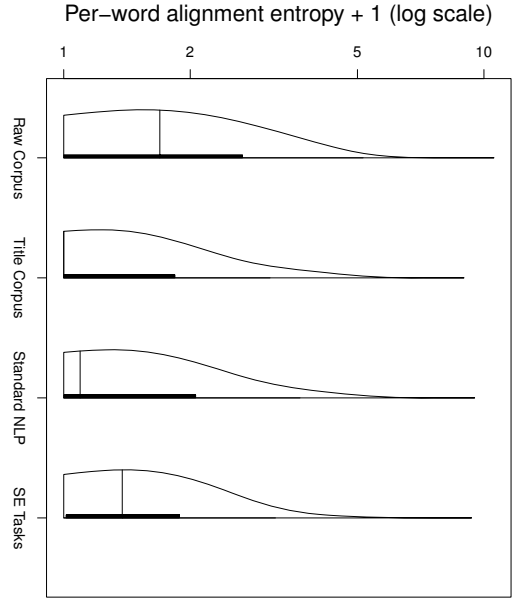

Fig. 2: Distribution of the per-word maximum likelihood alignment entropy. The more left skewed a distribution, the lower the entropy, and the higher predictive power.

IV. Conclusion

In this paper, we provide an aligned English to code corpus and show the potential of StackOverflow as a bilingual machine translation corpus. We show that the Raw StackOverflow posts contain substantial noise and do not lead to good MT models. In contrast, the cleaned corpora: the StackOverflow title, standard NLP including keyword extraction, and extracting SE tasks lead to low entropy alignments between English and code. Many works have shown that code is repetitive and predictable [12, 36], we show that the English discussions of code are also repetitive. Creating a maximum likelihood MT model, we find that English words map to a small number of specific code elements which partially explains the success of using StackOverflow for search [4, 34] and various SE tasks [6, 8, 17, 25, 37, 38, 39] as well as paving the way for advances in English to code MT [10, 22, 23, 24, 28, 40].

We make our scripts, corpora, and corpus preparation approaches available in the hope that other researches will use these StackOverflow alignments to help software engineers search for code and translate from English tasks to working code [1].

REFERENCES

[1] Replication package, 2018. https://github.com/mrsumitbd/SOParallelCorpusReplication.

[2] Y. Al-Onaizan, J. Curin, M. Jahr, K. Knight, J. Lafferty, D. Melamed, F.-J. Och, D. Purdy, N. A. Smith, and D. Yarowsky. Statistical machine translation. In *Final Report, JHU Summer Workshop*, volume 30, 1999.

[3] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] A. Barua, S. W. Thomas, and A. E. Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, 19(3):619–654, 2014.

[5] R. Barzilay and K. R. McKeown. Extracting paraphrases from a parallel corpus. In *Proceedings of the 39th annual meeting on Association for Computational Linguistics*, pages 50–57. Association for Computational Linguistics, 2001.

[6] A. Bosu, C. S. Corley, D. Heaton, D. Chatterji, J. C. Carver, and N. A. Kraft. Building reputation in stackoverflow: an empirical investigation. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 89–92. IEEE Press, 2013.

[7] B. A. Campbell and C. Treude. Nlp2code: Code snippet content assist via natural language tasks. *arXiv preprint arXiv:1701.05648*, 2017.

[8] S. A. Chowdhury and A. Hindle. Mining stackoverflow to filter out off-topic irc discussion. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 422–425. IEEE Press, 2015.

[9] M.-C. De Marneffe and C. D. Manning. Stanford typed dependencies manual. Technical report, Technical report, Stanford University, 2008.

[10] X. Gu, H. Zhang, D. Zhang, and S. Kim. Deep api learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 631–642. ACM, 2016.

[11] B. Harris. Interlinear bitext. *Language Technology*, page 12, 1988.

[12] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu. On the naturalness of software. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 837–847, 2012.

[13] K. S. Jones. Natural language processing: a historical review. In *Current issues in computational linguistics: in honour of Don Walker*, pages 3–16. Springer, 1994.

[14] M. Jungiewicz and M. Łopuszyński. *Unsupervised Keyword Extraction from Polish Legal Texts*, pages 65–70. Springer International Publishing, Cham, 2014.

[15] P. Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86, 2005.

[16] P. Koehn and K. Knight. Statistical machine translation, Nov. 24 2009. US Patent 7,624,005.

[17] O. Kononenko, D. Dietrich, R. Sharma, and R. Holmes. Automatically locating relevant programming help online. In *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*, pages 127–134. IEEE, 2012.

[18] E. Loper and S. Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[19] A. Lopez. Statistical machine translation. *ACM Computing Surveys (CSUR)*, 40(3):8, 2008.

[20] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann. Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 2857–2866. ACM, 2011.

[21] G. Neubig. Neural machine translation. 2016.

[22] A. Nguyen, P. Rigby, T. Nguyen, D. Palani, M. Karanfil, and T. Nguyen. Statistical translation of english texts to api code templates. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 194–205, Sep. 2018.

[23] T. Nguyen, P. C. Rigby, A. T. Nguyen, M. Karanfil, and T. N. Nguyen. T2api: Synthesizing api code usage templates from english texts with statistical translation. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 1013–1017, New York, NY, USA, 2016. ACM.

[24] D. K. Palani. *Statistical Machine Translation of English Text to API Code Usages: A comparison of Word Map, Contextual Graph Ordering, Phrase-based, and Neural Network Translations*. PhD thesis, Concordia University, 2018.

[25] G. Pinto, F. Castor, and Y. D. Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 22–31. ACM, 2014.

[26] L. Ponzanelli, A. Bacchelli, and M. Lanza. Seahawk: Stack overflow in the ide. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1295–1298. IEEE Press, 2013.

[27] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[28] M. Raghothaman, Y. Wei, and Y. Hamadi. Swim: synthesizing what i mean: code search and idiomatic snippet synthesis. In *Proceedings of the 38th International Conference on Software Engineering*, pages 357–367. ACM, 2016.

[29] M. Rahman, P. Dharani, and P. C. Rigby. Natural Software Revisited. In *Proceedings of the 2019 International Conference on Software Engineering*, ICSE '19, 2019.

[30] P. C. Rigby and M. P. Robillard. Discovering essential code elements in informal documentation. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 832–841. IEEE Press, 2013.

[31] S. Rose, D. Engel, N. Cramer, and W. Cowley. Automatic keyword extraction from individual documents. *Text Mining: Applications and Theory*, pages 1–20, 2010.

[32] C. Rosen and E. Shihab. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering*, 21(3):1192–1223, 2016.

[33] H. Somers. Example-based machine translation. *Machine Translation*, 14(2):113–157, 1999.

[34] K. T. Stolee, S. Elbaum, and M. B. Dwyer. Code search with input/output queries: Generalizing, ranking, and assessment. *Journal of Systems and Software*, 116:35–48, 2016.

[35] C. Treude, M. P. Robillard, and B. Dagenais. Extracting development tasks to navigate software documentation. *IEEE Transactions on Software Engineering*, 41(6):565–581, 2015.

[36] Z. Tu, Z. Su, and P. Devanbu. On the localness of software. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 269–280, New York, NY, USA, 2014. ACM.

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

[38] S. Wang, D. Lo, and L. Jiang. An empirical study on developer interactions in stackoverflow. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1019–1024. ACM, 2013.

[39] 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*, pages 562–567. IEEE Press, 2013.

[40] P. Yin, B. Deng, E. Chen, B. Vasilescu, and G. Neubig. Learning to mine aligned code and natural language pairs from stack overflow. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, pages 476–486, New York, NY, USA, 2018. ACM.