Abstract

The information that a developer seeks to aid in the completion of a task typically exists across different kinds of natural language software artifacts. In the artifacts that a developer consults, only some portions of the text will be useful to a developer's current task. Locating just the portions of text useful to a given task can be time-consuming as the artifacts can include substantial text to peruse organized in different ways depending on the type of artifact. For example, artifacts structured as tutorials might be easier to locate information given their structured headings whereas artifacts consisting of developer conversations might need to be read in detail.

To aid developers in this activity, given the limited time they have to spend on any given task, researchers have proposed a range of techniques to automate the identification of relevant text. However, this prior work is generally constrained to one or only a few types of artifacts. Integrating such artifact-specific approaches to allow developers to seamlessly search for information across the multitude of artifact types they find relevant to a task is challenging, if not impractical.

In this dissertation, we propose a set of generalizable techniques to aid developers in locating the portion of the text that might be useful for a task. These techniques are based on semantic patterns that arise from the empirical analysis of the text relevant to a task in multiple kinds of artifacts, leading us to propose techniques that incorporate the semantics of words and sentences to automatically identify text likely relevant to a developer's task.

We evaluate the proposed techniques assessing the extent to which they identify text that developers deem relevant in different kinds of artifacts associated with Android development tasks. We then investigate how a tool that embeds the most promising semantic-based technique might assist developers while they perform a task. Results show that semantic-based techniques perform equivalently well across multiple artifact types and that a tool that automatically provides task-relevant text assists developers effectively complete a software development task.

Glossary

DS$_{Python}$ Python tasks dataset, containing 28 natural language artifacts where 24 participants indicated text containing information that assisted them in writing a solution for three programming tasks involving well-known Python modules

TARTI Automatic Task-Relevant Text Identifier, our proof-of-concept semantic-based tool, which uses BERT to automatically identify and show text relevant to an input task in a given web page

Chapter 7

Discussion and Future Work

In this chapter, we discuss questions and decisions not addressed in the previous chapters of this dissertation. We also use this discussion to describe potential venues for future work.

7.1 Limitations & Trade-offs

When is tool support needed? A notable trade-off of answering which approaches can identify task-relevant text (Chapter **??**) and whether developers benefit from them (Chapter **??**), is that our studies did not focus on when tool support is needed and for whom. For example, certain participants in the TARTI experiment indicated that they focus more on code snippets and that they did not make use of the highlights provided by our tool:

> "*I realized going through [the experiment] that I read very little free-form text when looking for solutions. Mainly code samples with clear, succinct examples or type-/method definitions.*"

> "*I'm not going to bother reading the text if I don't have to, especially when the code snippets are easy to understand.*"

While other participants indicated that textual highlights were essential to complete a task, even when the task required writing code:

> "*The highlights were super useful, without them I would definitely had not been able to rapidly do the task.*"

> "*With the highlighted references, I was able to move much quicker. I quickly glanced at each resource, reading just the highlights to determine how valuable that resource was. The highlights allowed me to focus on the most relevant resources, gathering the necessary information to complete the task.* "

This feedback suggests that we could have observed different usage behaviours if participants had the chance to decide when to invoke TARTI. However, our experimental design focused on comparing tasks using or not TARTI and thus, participants did not have the option to request TARTI to automatically identify task-relevant text on the fly.

To understand who would use TARTI in which types of tasks or what factors influence the tool's usage, we would have to consider a different experiment with challenges and risks of its own. For example, designing a longitutional study investigating how developers with different expertise and background of some open source community would use TARTI.

Precision vs Recall. In Chapter **??**, we discuss that our semantic-based techniques were designed favoring locating all relevant text within and artifact (*i.e.,* recall) instead of correctly determining the text's relevance (*i.e.,* precision). As we motivated in the introduction of this thesis (Chapter **??**), missing relevant text means that a developer might have an incomplete or partial view of the information needed, and thus the reason why we favor recall.

Nonetheless, when favoring recall, we may obtain more false positives—non-relevant text indicated as relevant—and due to the limited time developers spend inspecting a natural language artifact [31], there is a chance that a developer could discard reading an artifact due to a false positive. Although abandoning reading an artifact and moving to another artifact could lead to non-efficient work (i.e., a developer might have to perform further searches and perhaps revisit artifacts that they have already inspected), we believe that the benefits of automatically identifying relevant text outweighs these risks. That is, inspecting an artifact without tool support is more time-consuming than inspecting each sentence retrieved to judge their relevance to the task at hand.

Training Data. In Chapter **??**, we have shown that semantic-based approaches identify task-relevant textual information across different types of artifacts without relying on assumptions about the nature of an artifact or its meta-data. However, we observe that the techniques with the best accuracy require training data and this could be considered as an impediment to the using or improving of such techniques.

Although we acknowledge that the need for training data is a limitation inherent to supervised techniques, we mitigated some of the challenges of requiring large amounts of training data by using pre-trained models [14]. Furthermore, the fact that artifact-specific techniques, such as AnswerBot, have accuracy comparable to semantic-based approaches provides a unique opportunity for the creation of synthetic data that one might use for training purposes. In a similar manner to how researchers created word embeddings for the software engineering domain [13], one might gather significantly large amounts of data from online resources and use techniques that apply to such resources to create the data needed for training. A requirement to using such a bootstrapping mechanism is that we must show that models trained on synthetic data do not simply replicate the underlying heuristics of the techniques used for bootstrapping. That is, we have to show that a technique trained in this manner correlates data from a task and an artifact in ways that extend beyond what a technique used for bootstrapping already provides. For example, showing that it can identify task-relevant textual information in a different type of artifact or showing that its accuracy is significantly better than the technique used for bootstrapping.

Costs. A second limitation of the semantic-based techniques we explored relates to the cost associated with these techniques. First, the hardware needed by the neural embeddings and the neural networks often requires dedicated servers with significant memory and high-throughput computational power. Due to the high demand for commercial servers with such properties, deploying our tools for usage by software development communities might incur significant financial burdens. The need for dedicated servers also means that our proposed techniques and our web browser plug-in cannot run offline. If we consider a standard client-server architecture, such as the one of TARTI, it means that one must send data about the task and artifacts that they are working to a remote server for processing. This

might not always be possible due to privacy reasons, i.e., outside the public domain, organizations would not be willing to our tool and, depending on their size, organizations might not have the resources needed for in-house solutions. Due to these limitations, the techniques and tools described in this dissertation must be treated as proofs-of-concept.

7.2 Semantics in Software Development Artifacts

In Chapter **??**, we used frame semantics [16] to infer the meaning of the sentences that developers deemed relevant to a software task. As one potential alternative to this approach, we could have considered taxonomies available in other software engineering studies. For example, the knowledge types in API documents [23] or the information types in Open Source GitHub issues [2] or in development mailing lists [12].

We decided to not use such taxonomies because the categories available in these and other studies are often based on a need for access to the meaning of the text and we were interested in assessing the applicability of a general linguistic approach for this purpose. To the best of our knowledge, there have been only a few uses of frame semantics in software engineering research [1, 17, 19] and these approaches have largely focused on text associated with software requirements, leaving open the question of the applicability of the approach to text in different kinds of natural language artifacts.

We address the question of whether semantic frames can help identify the meaning of software engineering text in a study orthogonal to this dissertation [24]. In this study, we assessed the applicability of generic semantic frame parsing to software engineering text aimed at supporting program comprehension activities. First, we assessed how the tool we used in Chapter **??** to analyze the meaning of the text considered relevant (i.e., SEMAFOR [10]) applies to text sampled from 1,802 documents drawn from a set of datasets [2, 6, 23, 34]. Based on the results from this analysis, we proposed *SEFrame*, a tool that tailors frame parsing to natural language text in software engineering artifacts. We assessed the correctness and robustness of *SEFrame* in a second evaluation where we found that the approach was correct in between 73% and 74% of the cases and that it can parse text

from a variety of software artifacts used to support program comprehension. These results motivated our decision to apply *SEFrame* in the design of the techniques we explored in Chapter **??**. Nonetheless, future research could consider investigating other frame semantic parsing tools (e.g., [7, 33]) to classify the meaning of software engineering text.

7.3 Deep Learning Models for Software Engineering

In Chapter **??** we used a deep learning neural network (BERT [11]) to automatically identify text relevant to a particular input task. There are many considerations that affect a neural network and, in this section, we discuss decisions we took with regards to how we applied BERT to our domain problem.

A first decision on using a pre-trained model is on the selection of the base model itself, i.e., what data has been used to pre-train the model. There are base models trained on text from Wikipedia [11], online news [26], or source code [15], to cite a few, and we could have explored how usage of these different models could impact the accuracy or our techniques. Nonetheless, we decided to use the BERT base model as published on the model's original paper [11].

Our rationale for this decision is motivated by the fact that at the time we designed our techniques, BERT had not been used in task-artifact sentence pairs. Hence, using the base model without modifications was helpful for establishing a baseline for future comparisons. Furthermore, we observed that BERT outperformed the word2vec technique with embeddings from the software engineering domain [13] and thus, using this base model did not represent a risk to our study.

With regards to software-specific models (e.g., [15, 21]), we also emphasize that these models have been trained on either source code or source code and method level text documentation. These datasets might not align with how developers discuss text in natural language artifacts [3], such as Stack Overflow posts or web tutorials. Therefore, future research must consider how models trained on text originating from various kinds of natural language software artifacts compare to the base model that we used.

A second decision relates to the model fine-tuning procedures and how we divided our data for training and evaluation purposes. In Chapter **??**, we split the

Android tasks and their associated artifacts in two equal portions using 25 tasks for training—with 10% of these tasks being used for validation—and 25 other tasks for evaluation. This division might have affected BERT's fine-tuning and it would have been interesting to explore other data splits. For example, at early iterations in the design of our techniques, we used 10-fold cross validation [32] and other approaches could consider randomly selecting a smaller portion of the data for evaluation purposes. We refrained from further pursuing such data splits because we were interested in testing our approach over a number of tasks with Stack Overflow artifacts (Section **??**) and other splits could have affected this evaluation.

Despite using only half of the tasks for our initial assessment, in Chapter **??**, TARTI used a model trained on all the 50 Android tasks, allowing us to verify that the model applies to unseen tasks and artifacts drawn from an entirely new dataset (DS_{python}). For this dataset, we found that the BERT technique with no filters has accuracy similar to the one in our previous assessment (Figure **??**).

7.4 Empirical Studies on Determining Relevancy

At multiple points in this dissertation, participants produced annotated data indicating the text that they deemed relevant to a software task. These annotations reflect their *explicit* reasoning about the information available in the text and what they consider relevant to the tasks that we presented them. However, asking participants to indicate what was relevant might have introduced deviations from what information they would implicitly use and deem relevant when performing a task [20].

Due to differences that might arise from explicit and implicit reasoning, we argue for the need for more empirical data gathered in a non-obstructive way and originating from software developers performing daily tasks. Conducting empirical experiments in a more realistic environment is challenging [18]. This effort is worthwhile as the richness of collected data can provide valuable insights to provide a foundation for tool development.

One potential method for data collection considers how recent studies with eye-trackers [9, 27, 30] have shown that the technology is not as disruptive as other methods such as think aloud protocols or manual input. Hence, we believe that eye-tracking could be used in a developer's working environment, leading to more

realistic data on which text developers perceive as task-relevant. For example, one could extend the work done by Kevic and colleagues on tracing developers' eye for change tasks [18] to also consider tracing data outside a developer's IDE for such a purpose.

A second venue to enrich datasets on the text that is relevant to a task is to identify who deemed what relevant. In Chapter **??**, we described a series of studies that mine text from natural language artifacts. These studies provide annotated data resulting from coding procedures that usually do not take into account differences in what might be relevant; or disagreements are resolved during the annotation process. However, in our formative study (Chapter **??**), we have observed that there is variability in what text may be relevant to a software task, a fact that other researchers have also brought to light [4, 29]. Therefore, if future empirical studies provide more data about who perceived which text as relevant, it could lead to benchmarks for evaluating techniques focused on a certain population (e.g., expert versus novice developers [5, 8]).

We leave the investigation of other data collection methods and the creation of benchmarks with different properties as part of future research.

7.5 Presenting Task-Relevant Text

In Chapter **??**, we described how TARTI highlights the text in a natural language artifact that it identifies as relevant to an input task. Our idea to highlight text draws from related work which suggests that textual highlights are a simple approach to surface the most important information in an artifact [25, 29]. Although simple, participants shared limitations of this strategy:

> "*It was much easier to follow with previously highlighted text. However, it would have been nicer to have some sort of side bar/index of highlighted snippets where I could know and scroll directly through the highlighted parts of a page.*"

> "*There was a resource page which was super long, and I found it very difficult to locate which sentences were highlighted, therefore, making that resource useless to me because I didn't have the motivation to scroll through it to find all highlights. An interface that gathers highlight locations would make a difference.*"

This feedback made us question other potential ways to present the text identified by TARTI. For example, we could have followed design principles adopted by Unakite [22]—a tool that collects, organizes and keeps track of information—to make TARTI display the text identified on its context menu. Figure 7.1 shows a mock up of TARTI bundling the highlights identified for one of the artifacts in the `titanic` task (Section **??**). Using this context menu, a user could click on the text identified and navigate to the part of the documentation originally containing the text.

Figure 7.1: Mock up of the highlights identified by TARTI as navigational cues; by clicking on a highlight, a user could be directed to the part of a document containing that highlight

A second approach to organizing the text identified could consider the semantic meaning of each sentence, as extracted through frame semantics or other semantic-based approaches. Semantic frames could assist a user in comprehending the content of the text automatically identified without the need to read it. As done by Libra [28], semantic frames could also be used to group the text identified in bubbles or a hierarchical representation helping a developer in deciding what portions of the text they would inspect first. Figure 7.2 shows a mock up with some of

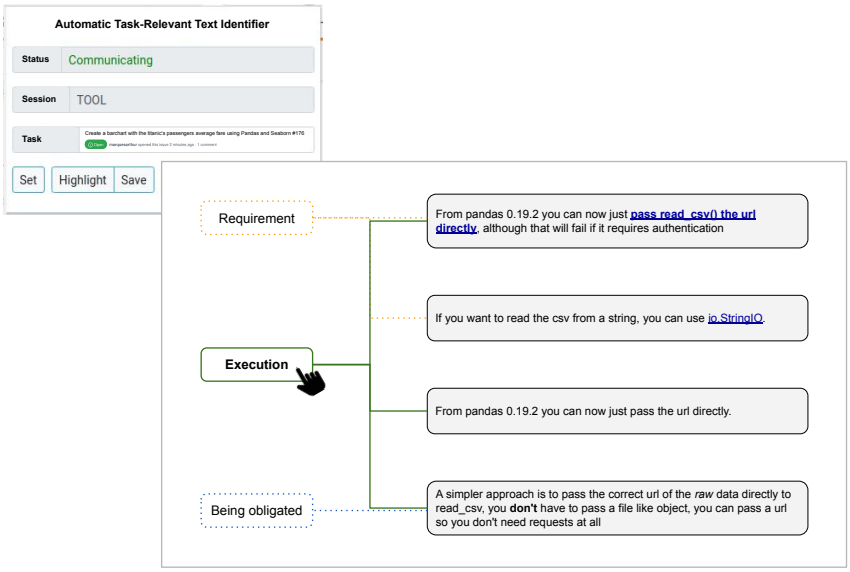

Figure 7.2: Mock up of the highlights grouped by semantic frames; by hovering over a semantic frame, a user could quickly identify which of the text identified is associated with a certain meaning

the semantic frames extracted for the sentences in Figure 7.1. Using the semantic frames identified, a developer could decide whether they would read sentences describing some coding procedure (*execution*), or sentences with warnings or requirements (*requirement* and *being obligated*) about the Pandas API. To be useful, future research must consider how to identify from all the frames available in a sentence, which frames better summarize the meaning of the text.

We leave the investigation of other ways to present the text identified as relevant to future research.

Bibliography

[1] W. Alhoshan, R. Batista-Navarro, and L. Zhao. Using frame embeddings to identify semantically related software requirements. In *REFSQ Workshops*, 2019. → page 4

[2] D. Arya, W. Wang, J. L. Guo, and J. Cheng. Analysis and detection of information types of open source software issue discussions. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 454–464, 2019. doi:10.1109/ICSE.2019.00058. → page 4

[3] D. M. Arya, J. L. Guo, and M. P. Robillard. Information correspondence between types of documentation for apis. *Empirical Software Engineering*, 25(5):4069–4096, 2020. → page 5

[4] G. Bavota. Mining unstructured data in software repositories: Current and future trends. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 5, pages 1–12, 2016. doi:10.1109/SANER.2016.47. → page 7

[5] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm. Eye movements in code reading: Relaxing the linear order. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 255–265, 2015. → page 7

[6] O. Chaparro, J. Lu, F. Zampetti, L. Moreno, M. Di Penta, A. Marcus, G. Bavota, and V. Ng. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 396–407, 2017. ISBN 9781450351058. doi:10.1145/3106237.3106285. → page 4

[7] X. Chen, C. Zheng, and B. Chang. Joint multi-decoder framework with hierarchical pointer network for frame semantic parsing. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2570–2578, 2021. → page 5

[8] M. E. Crosby and J. Stelovsky. How do we read algorithms? a case study. *Computer*, 23(1):25–35, 1990. → page 7

[9] E. Cutrell and Z. Guan. What are you looking for? an eye-tracking study of information usage in web search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI'07, page 407416, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595935939. doi:10.1145/1240624.1240690. → page 6

[10] D. Das, D. Chen, A. F. Martins, N. Schneider, and N. A. Smith. Frame-semantic parsing. *Computational linguistics*, 40(1):9–56, 2014. → page 4

[11] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. → page 5

[12] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, and H. C. Gall. Development emails content analyzer: Intention mining in developer discussions (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 12–23, 2015. doi:10.1109/ASE.2015.12. → page 4

[13] V. Efstathiou, C. Chatzilenas, and D. Spinellis. Word embeddings for the software engineering domain. In *Proceedings of the 15th International Conference on Mining Software Repositories (MSR)*, pages 38–41, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450357166. doi:10.1145/3196398.3196448. → pages 3, 5

[14] D. Erhan, A. Courville, Y. Bengio, and P. Vincent. Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 201–208. JMLR Workshop and Conference Proceedings, 2010. → page 3

[15] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online, Nov. 2020. Association for Computational Linguistics. doi:10.18653/v1/2020.findings-emnlp.139. → page 5

[16] C. J. Fillmore. Frame semantics and the nature of language. *Annals of the New York Academy of Sciences*, 280(1):20–32, 1976. → page 4

[17] N. Jha and A. Mahmoud. Mining user requirements from application store reviews using frame semantics. In *International working conference on requirements engineering: Foundation for software quality*, pages 273–287, 2017. → page 4

[18] K. Kevic, B. M. Walters, T. R. Shaffer, B. Sharif, D. C. Shepherd, and T. Fritz. Tracing software developers' eyes and interactions for change tasks. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 202–213, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336758. doi:10.1145/2786805.2786864. → pages 6, 7

[19] M. Kundi and R. Chitchyan. Use case elicitation with framenet frames. In *2017 IEEE 25th international requirements engineering conference workshops (REW)*, pages 224–231, 2017. → page 4

[20] J. Lazar, J. H. Feng, and H. Hochheiser. *Research Methods in Human Computer Interaction*. Morgan Kaufmann, Boston, second edition edition, 2017. ISBN 978-0-12-805390-4. → page 6

[21] H. Li, S. Kim, and S. Chandra. Neural code search evaluation dataset. *arXiv preprint arXiv:1908.09804*, 2019. → page 5

[22] M. X. Liu, N. Hahn, A. Zhou, S. Burley, E. Deng, J. Hsieh, B. A. Myers, and A. Kittur. UNAKITE: Support developers for capturing and persisting design rationales when solving problems using web resources. 2018. → page 8

[23] W. Maalej and M. P. Robillard. Patterns of knowledge in api reference documentation. In *IEEE Transactions on Software Engineering*, volume 39, pages 1264–1282, 2013. doi:10.1109/TSE.2013.12. → page 4

[24] A. Marques, G. Viviani, and G. C. Murphy. Assessing semantic frames to support program comprehension activities. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pages 13–24, 2021. doi:10.1109/ICPC52881.2021.00011. → page 4

[25] S. Nadi and C. Treude. Essential sentences for navigating stack overflow answers. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 229–239, 2020. doi:10.1109/SANER48275.2020.9054828. → page 7

[26] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi:10.18653/v1/N18-1202. → page 5

[27] R. Petrusel and J. Mendling. Eye-Tracking the Factors of Process Model Comprehension Tasks. In C. Salinesi, M. C. Norrie, and Ó. Pastor, editors, *Advanced Information Systems Engineering*, pages 224–239, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-38709-8. → page 6

[28] L. Ponzanelli, S. Scalabrino, G. Bavota, A. Mocci, R. Oliveto, M. Di Penta, and M. Lanza. Supporting software developers with a holistic recommender system. In *Proc. of the 39th Int'l Conf. on SE*, pages 94–105, Piscataway, NJ, USA, 2017. IEEE Press. ISBN 978-1-5386-3868-2. → page 8

[29] M. P. Robillard and Y. B. Chhetri. Recommending reference api documentation. *Empirical Software Engineering*, 20(6):1558–1586, Dec. 2015. ISSN 1382-3256. doi:10.1007/s10664-014-9323-y. → page 7

[30] Z. Sharafi, B. Sharif, Y.-G. Guéhéneuc, A. Begel, R. Bednarik, and M. Crosby. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering*, 25(5):3128–3174, 2020. → page 6

[31] J. Starke, C. Luce, and J. Sillito. Searching and skimming: An exploratory study. In *2009 IEEE International Conference on Software Maintenance*, pages 157–166, 2009. doi:10.1109/ICSM.2009.5306335. → page 2

[32] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society: Series B (Methodological)*, 36(2): 111–133, 1974. → page 6

[33] S. Swayamdipta, S. Thomson, C. Dyer, and N. A. Smith. Frame-Semantic Parsing with Softmax-Margin Segmental RNNs and a Syntactic Scaffold. *arXiv preprint arXiv:1706.09528*, 2017. → page 5

[34] B. Xu, Z. Xing, X. Xia, and D. Lo. AnswerBot: Automated generation of answer summary to developers' technical questions. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 706–716, 2017. doi:10.1109/ASE.2017.8115681. → page 4