Automatically Identifying Bug Entities and Relations for Bug Analysis

Dingshan Chen[1], Bin Li[1*], Cheng Zhou[1,2], Xuanrui Zhu[1]

[1]School of Information Engineering, Yangzhou University, Yangzhou, China
[2]Information Technology Research Base of Civil Aviation Administration of China,
Civil Aviation University of China, Tianjin,China

***Abstract*—During the bug fixing process, developers usually analyze the historical relevant bug reports in bug repository to support various bug analysis activities, i.e., bug understanding, bug localization, bug fixing, etc. There are rich semantics and relations in the bug reports, which are useful for bug analysis. Entity recognition and relation extraction are useful to represent semantics and relations in the text. However, the text in the bug reports are often in a free-style form, and include lots of noisy information. Therefore, how to extract and express rich semantics and relations in the bug reports is difficult, but important for bug analysis. To address this challenge, we propose an approach, which incorporates the neural networks RNN with dependency parser to automatically extract bug entities and their relations from bug reports. A preliminary empirical evaluation demonstrates that our approach is effective to extract bug entities and their relations from bug reports in the bug repository.**

***Index Terms*—Bug analysis, bug entity recognition, relation extraction, neural networks**

I. Introduction

Software bugs are continuously reported during software development and maintenance. Many open source software projects use Bug Tracking Systems (*BTS*) to store and manage bug reports. In recent years, lots of work utilized the Information Retrieval (IR) technology to explore these massive bug repositories to help understand, localize and fix bugs [1]–[3].

However, bug reports are submitted by different users, with free grammar and different structures, including a large number of phrases and short text. Traditional IR technology is based primarily on statistical methods that treat bug reports as a collection of words, which breaks the context and does not fully explore the semantic information in bug reports. Accurately extracting and expressing rich semantics and relations from bug reports is important for various bug analysis activities, such as bug understanding, bug localization and bug fixing [4].

Knowledge graph is shown to be effective for knowledge representation and management, which effectively expresses rich semantics in heterogeneous information [5]. Entities and their relations are key elements on the knowledge graph [5]. An entity is the smallest unit of knowledge and can be a single token or a combination of tokens, and the entity relation is a semantic relationship between two entities, usually represented by a triplet structure, such as *(E1, R, E2)*, where *Ei* represents an entity and *R* represents a relationship name or relationship type [6]. In this paper, we aim to express the information in bug reports by entities and their relations to help developers for better bug understanding and analysis. However, we have to overcome the following challenges for bug entity and relation extraction:

- The data in bug reports are usually with a free-style, semi-structured or unstructured, which makes it difficult to extract and process them.
- There are different types of information in the bug reports, such as software project itself, general software engineering knowledge and bug characteristics. It is difficult to accurately extract and categorize them in bug entities.
- There are a large volume of short text in the bug repository. For a short text, relation instances have no obvious relational words, often ambiguous, or the available relation features are seriously insufficient, which leads to the sparsity of relation features.

To address the above challenges, we manually examined textual content of bug reports, attempted to understand the semantics relations among the bug entities and define the classification categories of entity relations. In addition, we refer to the work of Zhou et al. [7] to define the bug entity categories in this paper. Then, we propose an automatic approach for bug entity and relation extraction based on neural networks and dependency parser. The major contributions of our work are shown as follows:

- The first work to define the categories of entity relations in bug reports;
- The first attempt in the software engineering community for automatic extraction of both bug entities and relations using the Bi-LSTM-RNN model and dependency parser; and
- A preliminary evaluation to show the effectiveness of the proposed approach.

The remainder of this paper is organized as follows. Section II presents the background. Section III elaborates on the details of our approach. Section IV presents the preliminary

*Corresponding Author.
E-mail: lb@yzu.edu.cn (Bin Li)

evaluation. We discuss the related work in Section V. Finally, Section VI concludes our work.

II. Background

In this section, we first analyze the classification criteria for bug entities and entity relations, and then introduce the used Recurrent Neural Networks and dependency parser used to extract bug entities and relations.

A. Category of Bug Entities and Relations

For category of bug entities, we refer to the work of Zhou et al. [7]. They classify bugs from the perspective of localization of a bug, i.e., component, as component is unique and always indicated in the bug report. Software can be divided into different kinds such as application software and operating software, and components in different kinds of software are inherently different. In this work, we also focus on extracting entities from the bug reports. So we directly use their classification approach for bug entities. In their work, they classify bug entities into sixteen categories, including *Core*, *GUI*, *Network (NW)*, *I/O (IO)*, *Driver(Dri)*, *FileSystem (FS)*, *Hardware (HD)*, *Language (LA)*, *API*, *Standard (SD)*, *Platform (PF)*, *Framework (FW)*, *common verb (CV)*, *defect test (TEST)*, *common adjective (CA)*, and *Mobile (MOB)*.

Based on the bug entity corpus, we manually examined the common relation words used to express relations between entities in bug reports. We define eight types of relations between bug entities. The specific categories and descriptions for relations between bug entities are shown in Table I.

Figure 1 shows an example of labeled sentence for the title of the Bug#1368216[1] in Mozilla. We use the marking strategy of Zheng et al. [8], where *B* and *I* indicate the Begin and Inside of the text chunk. Other meaningless tokens outside of entities such as "in" are labeled with *O*. Numbers "1" means that the token belongs to the first entity in the relation and "2" means that the token belongs to the second entity. Figure 1 shows five word entities (*Python*, *code*, *browser*, *flake8*, *convention*) and two relations (*be_in*, *should_follow*) for the text in this bug report.

B. Techniques for Entity and Relation Extraction

In this paper, we incorporate the Bi-LSTM-RNN (bi-directional long-short-term-memory recurrent-neural-network) [9] model with dependency parser to extract bug entities and their relations. The model can learn the representations of entities and their surrounding context for more accurate classification. The Bi-LSTM-RNN model firstly performs bi-directional recurrent computation along all the tokens in a sentence to generate the sequence of token representations. Then, the token representations are concatenated and fed into a softmax layer for classification. Finally, the softmax layer calculates the probabilities of all labels for an entity or entity relation. During this process, Long Short-Term memory (LSTM) [10] is used to alleviate the gradient vanishing problem when two entities are distant in text.

[1]https://bugzilla.mozilla.org/show_bug.cgi?id=1368216

Dependency parser extracts the grammatical structure tree of a sentence to derive relations between entities by their shortest dependency path (SDP) [11]. We use the Stanford CoreNLP toolkit [12] for dependency parsing.

III. Approach

Figure 2 shows the framework of the proposed approach. Given bug reports in the bug repository, we preprocess the text in the bug reports to extract words in them. Then, we select some of bug reports, and construct a baseline corpus to manually label the categories of bug entities and relations as defined in Section II. For other bug reports, we will transform them into a matrix for the following joint extraction model to analyze. Finally, we use the Bi-LSTM-RNN model and dependency parser to predict the categories of entities and relations for the given text from bug reports, and generate a set of triples for each pair of entities as well as relations between them.

A. Bug Data Preprocessing and Labeling

Given a bug report, we split the text in the bug report into sentences and then tokenize these sentences into words [13]. However, for bug data, tokenization is performed using not only white-spaces but also punctuation, since we might not find the node for an entity in the dependency tree if it is not separated from a piece of text. This process is implemented using the tool called *Stanford CoreNLP* for tokenization, POS (part of speech) Tagging and Lemmatization [12]. For tokenization, each sentence is divided into a series of tokens and the tokenizer uses regular expression to match the joint structure of the bug entity. The tokenizer does not split the name of a bug entity, e.g., the name of *flake8*. It does not split valid programming operators, such as "+=" and "++". In this way, the text in bug reports is represented by meaningful words that can be used for analysis.

In addition, to recognize categories of entities and relations, we need a baseline labeled corpus for training of the prediction model. We manually labeled the preprocessed text in bug reports using the strategy of Zheng et al. [8] as shown in Figure 1. The manual annotation process includes two steps and is performed by four participants who are all with software engineering background. We first defined the labeling method and trained the participants' annotations about entities and relationships. In Step 1, each participant is assigned with some bug sentences splitted from bug reports. Then, we use the feedback from the participants to improve the quality of annotations. In Step 2, participants cross validate the annotation data. In this way, we can build a baseline corpus with categorization labels of entities and relations for sampled bugs.

B. Input Representation for Bugs

After preprocessing bug reports, we obtain their word tokens. To utilize the Bi-LSTM-RNN model for entity and relation recognition, we need to transform the incoming bug reports into the forms fit to the model. We use two vectors

TABLE I
CATEGORIES FOR RELATIONS BETWEEN BUG ENTITIES

Relation Categories	Description	Anno.Tag
Brother	Both entity *a* and entity *b* are derived from the same parent class	Bro
Consensus	Different entity representations of unified bug knowledge	Con
Inclusion	Entity *a* is contained in the entity *b*	Inc
Opposition	Both entities should not exist at the same time	Opp
Inheritance	Entity *a* directly has another entity's properties and methods	Inh
Explanation	Entity *a* is a description of entity *b*	Exp
Cause	Entity *a* is the reason for another entity's conclusion	Cau
Semantic	There is a semantic connection between entity *a* and entity *b*	Sem

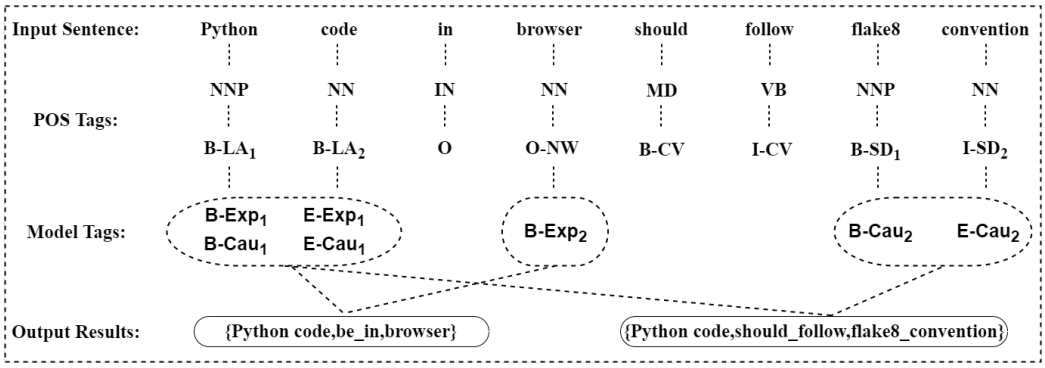

Fig. 1. An example of expressing entities and relations for bug information

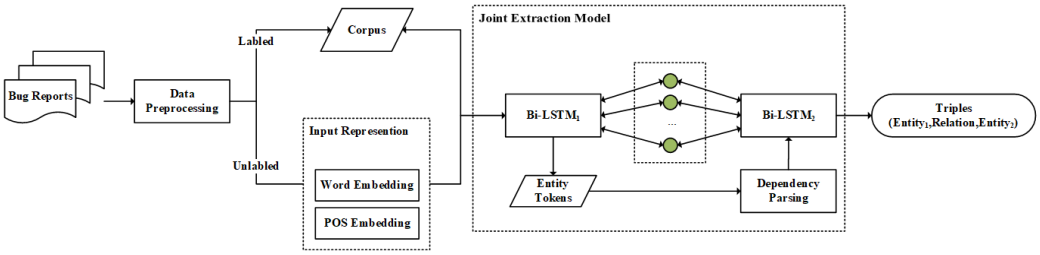

Fig. 2. The process of our approach

to represent each input token: word embedding and POS embedding. The two vectors are concatenated and transformed by a matrix representation and fed to the rectified layer of the Bi-LSTM-RNN model for analysis.

C. Joint Model for Bug Entity and Relation Extraction

With the labeled corpus and the transformed matrix representation for the text in bug reports, we first use the Bi-LSTM-RNN model to learn and identify bug entity pairs in a sentence. The softmax output layer in the Bi-LSTM-RNN model will calculate the probabilities of all entity labels (bug entity categories) for each pair of entities of bugs, and the entity category with the maximum probability is selected as their label.

With the recognized bug entities, we can further identify the relations between each pair of bug entities. We first use the dependency parser to extract the grammatical structure of the sentence to derive relations between each pair of entities by finding the shortest dependency path (SDP) of two target entities in the dependency tree. Then, we use the Bi-LSTM-RNN to model relation representations between two target entities along their SDP. Finally, with the softmax output layer in the Bi-LSTM-RNN model, we calculate the probabilities of all relation labels (categories of relations) between each pair of bug entities, and the relation category with the maximum probability is selected as the relation label for this pair of bug entities. In this way, we can obtain a set of triples for each pair of entities in the form of *(Entity1, Relation, Entity2)*, where *Entity1* and *Entity2* represent bug entities, and *Relation* represents the relation type.

IV. PRELIMINARY EVALUATION

To evaluate our approach, we randomly selected 500 bug reports from the Mozilla project[2]. Then, we extracted the title, description, comments from each bug report to construct our corpus. The statistics of extracting entities and relations from 500 bug reports are shown in Table II. We do 10-fold cross-validation to evaluate our model, where 20% of the data is used as the test data and the remaining is used as the training

[2]https://bugzilla.mozilla.org/

data. Meanwhile, we also split 10% samples out of the training set for validation. As shown in Table II, we have a total of 7126 entity relationships, and the specific distribution of the eight relation categories is shown in Figure 3.

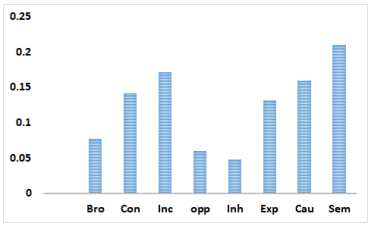

Fig. 3. Distribution of different categories of bug relation extracted in the corpus

TABLE II
STATISTICS OF EXTRACTING ENTITIES AND RELATIONS FROM 500 BUG REPORTS

Corpus	Sentences	Bug Entities	Relations
Training set	4568	8206	5580
Testing set	1142	2114	1546
Total	5710	10320	7126

We employ the Precision (P), Recall (R) and $F1$ to evaluate our approach for bug entity and relation extraction. For each category of bug entity and relation, P measures to what extent the output labels are correct. R measures to what extent the entities and relations in the golden dataset (testing data from the corpus) are labeled correctly. $F1$ is the harmonic mean of P and R. For entity extraction, we also compare our approach with the work of Zhou et al. (BNER) [7].

Table III shows the empirical results. For bug entity recognition, we compare our bug entity recognition result with BNER, which extracts bug-specific entity recognition with the Conditional Random Fields (CRF) model and word embedding technique. As shown in Table III, the first row shows the performance of entity recognition by BNER based on our corpus. For bug entity recognition, our approach improves the accuracy over BNER with precision from 81.5% to 83.7%, recall from 74.1% to 75.2%. Therefore, in terms of entity extraction, our joint model is better than the original bug named entity recognition method BNER.

Meanwhile, we also compare our approach on bug relation extraction with SDP-LSTM [11], which is a novel neural network to classify the relation of two entities in a sentence. SDP-LSTM architecture also leverages the shortest dependency path (SDP) between two entities, picking up heterogeneous information along the SDP. Table III shows the results of comparing these two methods. For bug relation extraction, our approach improves the accuracy over SDP-LSTM with precision from 53.74% to 66.3%, recall from 51.7% to 61.5%. To summarize, our joint model performs well for both the bug entity recognition and their relations extraction from the bug reports.

TABLE III
EVALUATION RESULTS

Technique	Entity Recognition			Relation Extraction		
	P(%)	R(%)	F1(%)	P(%)	R(%)	F1(%)
BNER	81.5	74.1	77.6	-	-	-
SDP-LSTM	-	-	-	53.74	51.7	52.7
Our Approach	83.7	75.2	79.2	66.3	61.5	63.8

V. RELATED WORK

Extraction of entities and relations have been widely studied in many communities, such as social media, agriculture and biology [14] [15] [16]. Meanwhile, there are also some work to mine the knowledge of the community of software engineering [17]. Recently, some work on extracting entities have been done in the software engineering community. Ye et al. proposed an S-NER system using CRF model and Brown clustering to recognize software specific entities such as programming languages, platforms and libraries on Stackoverow posts [18]. Zhou et al. presented a BNER system using CRF model and word embedding to recognize software bug-specific entities from bug reports [7]. Different from these work, we not only identify the entities, but also the relations between them. Specifically, we incorporate the neural networks (Bi-LSTM-RNN) with dependency parser to automatically extract bug entities and their relations from the bug reports.

There are also some work focusing on exploring the entities and relations in software to construct the knowledge graph for applications in software engineering tasks. Zhao et al. aimed to discover domain specific concepts and their relation triples from the content of webpages by a machine learning algorithm [19]. Han et al. proposed a translation-based, description-embodied knowledge representation learning method to embed both software weaknesses and their relations in the knowledge graph used for knowledge acquisition and inference [20]. Wang et al. proposed to utilize bug knowledge graph for bug resolution [21]. In this paper, with the identified bug entities and relations, we can also construct the bug knowledge graph, which will be our future work. Currently, we only use our approach on Mozilla, and the follow-up work will be considered on multiple projects, such as Github [22], [23], Eclipse and Stack Overflow [4], [19], [24].

VI. CONCLUSION AND FUTURE WORK

In this paper, we use the bug category defined in the work of Zhou et al. [7] for bug entity recognition, and define a category of relations between these entities. Then, we propose to use a neural joint model, which incorporates the neural networks (Bi-LSTM-RNN) with dependency parser, to automatically extract bug entities and their relations. The preliminary empirical evaluation demonstrates that our approach can effectively extract the defined bug entities and relations from bug reports.

We believe that our work can facilitate the research in bug localization and fixing with the identified bug entities and relations, which we have being studied as our work. In addition, we aim to provide a public baseline platform for

bug entities and relations extraction with more bug reports in our future work to support bug fixing studies in the software engineering community. We currently only train and test on the mozilla project. There may be some over-fitting, and we will conduct cross-project analysis for further evaluation.

ACKNOWLEDGMENT

We express our gratitude to all those who participated in this work, especially the data labelling process. This work is supported partially by Natural Science Foundation of China under Grant No. 61472344, No. 61872312, No. 61402396, and No. 61611540347, partially by the Jiangsu Qin Lan Project, partially by the Jiangsu "333" Project, partially by the Jiangsu Overseas Visiting Scholar program, partially by the Open Project Foundation of Information Technology Research Base of Civil Aviation Administration of China (CAAC-ITRB201704), partially by the Natural Science Foundation of Yangzhou City under Grant No. YZ2017113, and partially by the 2018 College Students Academic Science and Technology Innovation Fund Project of Yangzhou University under Grant No. x20180304.

REFERENCES

[1] T. B. Le, F. Thung, and D. Lo, "Will this localization tool be effective for this bug? mitigating the impact of unreliability of information retrieval based bug localization tools," *Empirical Software Engineering*, vol. 22, no. 4, pp. 2237–2279, 2017.

[2] S. Khatiwada, M. Tushev, and A. Mahmoud, "Just enough semantics: An information theoretic approach for ir-based software bug localization," *Information & Software Technology*, vol. 93, pp. 45–57, 2018.

[3] X. Sun, H. Yang, H. Leung, B. Li, H. J. Li, and L. Liao, "Effectiveness of exploring historical commits for developer recommendation: an empirical study," *Frontiers Comput. Sci.*, vol. 12, no. 3, pp. 528–544, 2018. [Online]. Available: https://doi.org/10.1007/s11704-016-6023-3

[4] X. Sun, X. Peng, K. Zhang, Y. Liu, and Y. Cai, "How security bugs are fixed and what can be improved: an empirical study with mozilla," *Science China Information Sciences*, vol. 62, no. 1, p. 19102, Dec 2018. [Online]. Available: https://doi.org/10.1007/s11432-017-9459-5

[5] L. Dietz, A. Kotov, and E. Meij, "Utilizing knowledge graphs for text-centric information retrieval," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, 2018, pp. 1387–1390. [Online]. Available: http://doi.acm.org/10.1145/3209978.3210187

[6] G. Zhou, J. Su, J. Zhang, and M. Zhang, "Exploring various knowledge in relation extraction," in *ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 25-30 June 2005, University of Michigan, USA*, 2005, pp. 427–434. [Online]. Available: http://aclweb.org/anthology/P/P05/P05-1053.pdf

[7] C. Zhou, B. Li, X. Sun, and H. Guo, "Recognizing software bug-specific named entity in software bug repository," in *Proceedings of the 26th Conference on Program Comprehension, ICPC 2018, Gothenburg, Sweden, May 27-28, 2018*, 2018, pp. 108–119. [Online]. Available: http://doi.acm.org/10.1145/3196321.3196335

[8] S. Zheng, F. Wang, H. Bao, Y. Hao, P. Zhou, and B. Xu, "Joint extraction of entities and relations based on a novel tagging scheme," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, 2017, pp. 1227–1236. [Online]. Available: https://doi.org/10.18653/v1/P17-1113

[9] F. Li, M. Zhang, G. Fu, T. Qian, and D. Ji, "A bi-lstm-rnn model for relation classification using low-cost sequence features," *CoRR*, vol. abs/1608.07720, 2016. [Online]. Available: http://arxiv.org/abs/1608.07720

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[11] Y. Xu, L. Mou, G. Li, Y. Chen, H. Peng, and Z. Jin, "Classifying relations via long short term memory networks along shortest dependency paths," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, 2015, pp. 1785–1794. [Online]. Available: http://aclweb.org/anthology/D/D15/D15-1206.pdf

[12] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*, 2014, pp. 55–60. [Online]. Available: http://aclweb.org/anthology/P/P14/P14-5010.pdf

[13] X. Sun, X. Liu, J. Hu, and J. Zhu, "Empirical studies on the nlp techniques for source code data preprocessing," in *Proceedings of the 2014 3rd International Workshop on Evidential Assessment of Software Technologies*, ser. EAST 2014, 2014, pp. 32–39.

[14] X. Liu, S. Zhang, F. Wei, and M. Zhou, "Recognizing named entities in tweets," in *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, 2011, pp. 359–367. [Online]. Available: http://www.aclweb.org/anthology/P11-1037

[15] C. Qi, Q. Song, P. Zhang, and H. Yuan, "Cn-makg: China meteorology and agriculture knowledge graph construction based on semi-structured data," in *17th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2018, Singapore, Singapore, June 6-8, 2018*, 2018, pp. 692–696. [Online]. Available: https://doi.org/10.1109/ICIS.2018.8466485

[16] N. Kang, B. Singh, Q. Bui, Z. Afzal, E. M. van Mulligen, and J. A. Kors, "Knowledge-based extraction of adverse drug events from biomedical text," *BMC Bioinformatics*, vol. 15, p. 64, 2014. [Online]. Available: https://doi.org/10.1186/1471-2105-15-64

[17] X. Sun, H. Yang, X. Xia, and B. Li, "Enhancing developer recommendation with supplementary information via mining historical commits," *Journal of Systems and Software*, vol. 134, pp. 355–368, 2017. [Online]. Available: https://doi.org/10.1016/j.jss.2017.09.021

[18] D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li, and N. Kapre, "Software-specific named entity recognition in software engineering social content," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*, 2016, pp. 90–101. [Online]. Available: https://doi.org/10.1109/SANER.2016.10

[19] X. Zhao, Z. Xing, M. A. Kabir, N. Sawada, J. Li, and S. Lin, "HDSKG: harvesting domain specific knowledge graph from content of webpages," in *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, 2017, pp. 56–67. [Online]. Available: https://doi.org/10.1109/SANER.2017.7884609

[20] Z. Han, X. Li, H. Liu, Z. Xing, and Z. Feng, "Deepweak: Reasoning common software weaknesses via knowledge graph embedding," in *25th International Conference on Software Analysis, Evolution and Reengineering, SANER 2018, Campobasso, Italy, March 20-23, 2018*, 2018, pp. 456–466. [Online]. Available: https://doi.org/10.1109/SANER.2018.8330232

[21] L. Wang, X. Sun, J. Wang, Y. Duan, and B. Li, "Construct bug knowledge graph for bug resolution: poster," in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume*, 2017, pp. 189–191. [Online]. Available: https://doi.org/10.1109/ICSE-C.2017.102

[22] X. Sun, W. Xu, X. Xia, X. Chen, and B. Li, "Personalized project recommendation on github," *SCIENCE CHINA Information Sciences*, vol. 61, no. 5, pp. 050 106:1–050 106:14, 2018. [Online]. Available: https://doi.org/10.1007/s11432-017-9419-x

[23] X. Sun, T. Zhou, G. Li, J. Hu, H. Yang, and B. Li, "An empirical study on real bugs for machine learning programs," in *24th Asia-Pacific Software Engineering Conference, APSEC 2017, Nanjing, China, December 4-8, 2017*, 2017, pp. 348–357. [Online]. Available: https://doi.org/10.1109/APSEC.2017.41

[24] X. Sun, B. Li, H. Leung, B. Li, and J. Zhu, "Static change impact analysis techniques: A comparative study," *Journal of Systems and Software*, vol. 109, pp. 137–149, 2015. [Online]. Available: https://doi.org/10.1016/j.jss.2015.07.047