Original software publication

Predicting issue types on GitHub

Rafael Kallis [a,*], Andrea Di Sorbo [b], Gerardo Canfora [b], Sebastiano Panichella [c]

[a] *Valdon Group, Seilergraben 53, 8001 Zurich, Switzerland*
[b] *University of Sannio, Piazza Guerrazzi, 82100 Benevento, Italy*
[c] *Zurich University of Applied Sciences, Obere Kirchgasse 2, 8400 Winterthur, Switzerland*

ARTICLE INFO

ABSTRACT

Software maintenance and evolution involves critical activities for the success of software projects. To support such activities and keep code up-to-date and error-free, software communities make use of issue trackers, i.e., tools for signaling, handling, and addressing the issues occurring in software systems. However, in popular projects, tens or hundreds of issue reports are daily submitted. In this context, identifying the type of each submitted report (e.g., bug report, feature request, etc.) would facilitate the management and the prioritization of the issues to address. To support issue handling activities, in this paper, we propose TICKET TAGGER, a GitHub app analyzing the issue title and description through machine learning techniques to automatically recognize the types of reports submitted on GitHub and assign labels to each issue accordingly. We empirically evaluated the tool's prediction performance on about 30,000 GitHub issues. Our results show that the Ticket Tagger can identify the correct labels to assign to GitHub issues with reasonably high effectiveness. Considering these results and the fact that the tool is designed to be easily integrated in the GitHub issue management process, Ticket Tagger consists in a useful solution for developers.

---

* Corresponding author.
E-mail addresses: rk@rafaelkallis.com (R. Kallis), disorbo@unisannio.it (A. Di Sorbo), canfora@unisannio.it (G. Canfora), panc@zhaw.ch (S. Panichella).

Software metadata

Software metadata description	
Current software version	2.1.4
Permanent link to executables of this version	https://github.com/ScienceofComputerProgramming/SCICO-D-20-00067
Legal Software License	GNU General Public License (GPL)
Computing platform / Operating System	macOS, Linux
Installation requirements & dependencies	nodejs 12
If available Link to user manual - if formally published include a reference to the publication in the reference list	https://github.com/rafaelkallis/ticket-tagger/blob/master/README.md
Support email for questions	rk@rafaelkallis.com

Code metadata

Code metadata description	
Current code version	2.1.4
Permanent link to code/repository used of this code version	https://github.com/ScienceofComputerProgramming/SCICO-D-20-00067
Legal Code License	GNU General Public License (GPL)
Code versioning system used	git
Software code languages, tools, and services used	javascript, nodejs, heroku, fasttext
Compilation requirements, operating environments & dependencies	nodejs 12
If available Link to developer documentation/manual	https://github.com/rafaelkallis/ticket-tagger
Support email for questions	rk@rafaelkallis.com

1. Introduction

Software maintenance involves tasks for mitigating potential defects in the code, as well as for evolving it according to the users' emerging needs [1]. Thus, it is crucial for the success of software projects. Issue tracking systems are tools to support these tasks by providing facilities to efficiently signal, manage, and address tickets or potential problems arising in software systems. In this context, software developers are required to timely react to issues reported in issue trackers and solve such issues by investing the lowest possible effort, to keep the costs related to software maintenance low [2]. However, especially in popular projects, tens or hundreds of issues are reported daily. This complicates the issues management activities, resulting in heavier workloads for developers [3,4].

In projects hosted on GitHub, issue submitters report new issues by simply providing a title and an optional description of the issue. As issues of different types (e.g., asking questions, proposing features, signaling bugs) and quality could be submitted, GitHub also offers a customizable labeling system that can be used by developers to tag issue reports (e.g., by specifying the issue category or the related development tasks). Such labeling has positive effects on issues processing [5], making it easier for their management and prioritization [6]. More specifically, labels assigned to issues help to classify and filter the reports, allowing more efficient issue handling processes. However, the manual labeling of issues may be labor-intensive, error-prone and time-consuming for project managers [7] and, for this reason, labels are barely used on GitHub [8,3].

To help maintainers dealing with issue processing, we developed TICKET TAGGER [9], a tool able to automatically label issue reports. Differently from previous approaches aimed at automatically identifying issue types [10,11], since GitHub (according to its lightweight structure) does not provide any structured information about such issues, our tool exclusively relies on the textual features contained in the titles and descriptions of the reports to enable the automated labeling of them, immediately after they are submitted. This is beneficial for developers interested to handle new issues [6].

In this paper we briefly illustrate Ticket Tagger, a GitHub app that can easily work on any software repository hosted on GitHub and automatically marks new issues submitted to target repositories with a relevant label. Besides, we assess the classification performance achieved by using different machine learning strategies and investigate the extent to which confounding factors of different types can degrade classification results.

Paper structure. The paper is organized as follows: Section 2 describes Ticket Tagger's approach and briefly presents the tool's main features, while in Section 3 we assess the tool's classification performance. Section 4 discusses the threats that could affect the validity of our work and, finally, Section 5 concludes the paper outlining future research directions.

2. Approach and tool's overview

To classify an issue report, Ticket Tagger processes the report's title and body to represent the textual information (extracted from the issue) in a vectorial space. By inspecting the resulting components, the tool can assign a relevant label to the mentioned report.

The Machine Learning Model. Different Machine Learning (ML) algorithms can be adopted to efficiently classify textual information [12–14]. However, complex ML strategies may require a long time for training and consume a lot of memory.
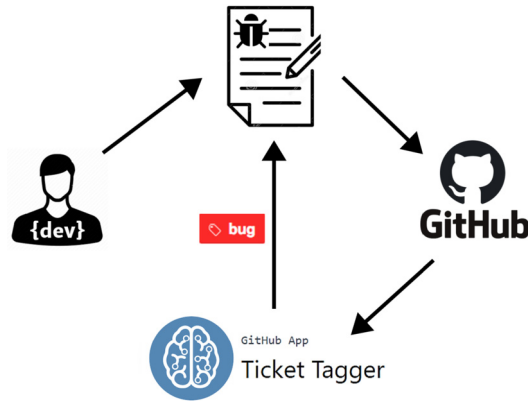
Fig. 1. Ticket Tagger issue labeling process.

Since we wanted to deploy the model on low-end server hardware,[1] we opted for *fastText*, a tool using linear models with a rank constraint and fast loss approximation, able to achieve comparable classification results to several deep learning-based approaches [12].

Issue Reports pre-processing and Vectorial Representation. For allowing the fastText linear classifier to make issue type predictions, the title and body of the reports are concatenated into a single textual paragraph. The resulting text is then tokenized and the tokenized text represents the source for obtaining the bag of words representation of the issue. This bag of words representation, in which each word is represented by a vector of *character n-grams*, is the input of the fastText based classifier.

Issues Classification. The fastText model classifies issues by minimizing the following objective function over N possible labels:

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n))$$

where x_n is a bag of features, A represents the weight dictionary of the average text embeddings, B is the weight dictionary that converts the embedding to pre-softmax values for each class, and f is the hierarchical softmax function used to minimize computational complexity [9].

We set fastText by using the default values for most of the parameters[2] and applied the following customization:

- word n-gram features are not captured, i.e., `wordNgrams` parameter;
- we only consider words that occur at least 14 times in the dataset, i.e., `minCount` parameter.

Both settings have been applied according to the disk constraints of our server hardware. Indeed, these decisions allowed us to obtain a trained model requiring less than 5 MB of disk space whilst only imposing a <10% performance penalty.

Ticket Tagger is currently able to classify issues according to three categories reflecting the intent [13,15] of the writer: *bug report*, *enhancement*, and *question*. These labels are included by default in every GitHub repository and they are the three labels most used on GitHub [8]. Obviously, our model is designed to be easily re-trained to adapt Ticket Tagger to specific projects' needs, enabling the prediction of additional issue types.

Tool's Overview. When a new issue report is submitted to a GitHub repository on which Ticket Tagger is installed, the tool automatically assigns a relevant label to the new report. In particular, Ticket Tagger is a Node.js-based GitHub app, that automatically (i) gathers issue reports information from a GitHub repository, and (ii) labels the newly reported issues, by leveraging the pre-trained fastText model previously discussed. The app is freely accessible and can be easily installed onto any existing GitHub repositories. By navigating to the Ticket Tagger app webpage,[3] to install Ticket Tagger on a target repository, the repository administrator has to click on the "Install" button, specify the repository, and that's it. From this moment on, as depicted in Fig. 1, when a user opens a new issue ticket on the repository, GitHub calls the hook endpoint exposed by Ticket Tagger and references the information related to the newly created issue. Such information is used by the app to classify the ticket. In order to automatically label the issue report, GitHub provides a temporary access token to Ticket

---

[1] AWS EC2 `t2.nano` (1 vCPU, 512 MB RAM, 20 GB SSD).
[2] For further details, see https://fasttext.cc/docs/en/options.html.
[3] https://github.com/apps/ticket-tagger.

Tagger, which is consumed by assigning the predicted label to the issue. The automated issue labeling performed by Ticket Tagger allows the developers to (i) timely react to urgent issues, (ii) postpone less impelling tasks (such as enhancement requests), or (iii) assign the questions to specific users.

3. Performance evaluation

In this section, we describe the datasets and baseline approach used to assess the classification performance of the fastText model integrated into Ticket Tagger (described in Section 2).

Datasets Construction. For assessing Ticket Tagger's effectiveness in classifying GitHub issues we collected two datasets. The first dataset, $D_{balanced}$, contains 30,000 issues.[4] This dataset was obtained by first collecting issues from 12,112 heterogeneous projects, this by querying the GitHub Archive[5] using Google BigQuery.[6] After this initial step, we randomly sampled issues from the set of all GitHub issues closed during February 2018, thus selected all issues having label matching the following strings: *bug*, *enhancement* or *question*. With this random selection process, we selected, on average, 2.48 issues for each project (median $= 1$ and standard deviation $= 15.78$). One third of the 30,000 issues had the bug label assigned; one third issues had the enhancement label[7] assigned; while the remaining 10,000 issues had the question label assigned. To construct the second dataset, $D_{unbalanced}$, we ran a query[8] over the GitHub Archive using Google BigQuery. We queried for issues containing any of the three labels, i.e., bug, enhancement and question, between the 1st and 9th of March 2018 in the GitHub Archive, obtaining approximately 34,000 issues.[9] The resulting distribution of issue types in $D_{unbalanced}$ is as follows: 16,355 (48%) tickets labeled as bug, 14,228 (41.8%) tickets marked as enhancement, and 3,458 (10.2%) question issues. While the first dataset, $D_{balanced}$, contains an identical number of tickets from each category, the second one, $D_{unbalanced}$, presents an unbalanced distribution of labels and is more representative of reality.

Evaluation Methodology. The goal of our experiments is twofold. On the one hand we compare Ticker Tagger against a baseline approach, to observe whether more simple ML-based approaches are able to achieve comparable or better results than Ticker Tagger. On the other hand, we evaluate the extent to which Ticket Tagger is able to automatically identify the correct labels to assign to issue reports in a realistic scenario. More specifically, we compare Ticket Tagger with the J48 machine learning (ML) algorithm that was successfully used in previous work concerning the assessment of ML strategies for textual classification problems [14,15]. To perform such a comparison, a 10-fold cross validation strategy [16] on $D_{balanced}$ is used for evaluating the classification performance achieved by both Ticket Tagger and the baseline J48 ML algorithm.

For training the J48 model, we leverage all the terms contained in both titles and descriptions of issues in our dataset to build a document-term matrix M, where each row represents an issue of our dataset, and each column represents a term. Every entry M_{ij} of the aforementioned matrix represents the weight or importance of the *j-th* term in the *i-th* issue, computed according to the *tf-idf* weighting scheme [17] that has been successfully used in recent work concerning the classification of GitHub issues [7] and vulnerabilities [18]. It is worth noticing that, for ensuring a fair comparison between the two models, in applying J48, we do not perform any model tuning and pre-processing of the data, since also fastText is used in the same way. In the future, we are interested in investigating the pre-processing steps and parameters tuning required to achieve better results. Furthermore, the evaluation is performed without the custom settings used for reducing fastText's disk space (described in Section 2).

With the aim of assessing the Ticket Tagger's capability of recognizing issue types in a realistic setting, i.e., unbalanced distribution of issue types, we carry out a further experiment in which Ticket Tagger is trained on the whole balanced dataset, $D_{balanced}$, and the unbalanced dataset, $D_{unbalanced}$, is used for evaluating the classification performance. This particular setting, i.e., balanced training set and unbalanced test set, is motivated by the need to avoid that the resulting model is biased towards the majority class(es). Well-known information retrieval metrics, namely precision, recall, and F-measure [17], are adopted to evaluate the classification performance in our experiments.

Results. Table 1 reports the classification performance achieved by both Ticket Tagger and the baseline approach (J48) using 10-fold cross validation on $D_{balanced}$. In particular, Table 1 shows how Ticker Tagger obtained F-measure values above 0.80 for each considered label, confirming the practical usefulness of the proposed approach for improving the issue management practices on GitHub. In addition, we can observe how for $D_{balanced}$, Ticket Tagger always outperforms the baseline approach (J48) for all labels and in all precision, recall, and F-measure metrics.

Table 2 shows the performance of Ticket Tagger in identifying bug, enhancement and question issues, when trained on $D_{balanced}$ and tested on $D_{unbalanced}$. The results of this second experiment highlight that our tool automatically identifies issues of the bug and enhancement types with reasonably high effectiveness, i.e., F-measure of about 0.75, while lower classification performance is obtained for the question category. On the one hand, these findings confirm the practical

---

[4] https://tinyurl.com/y23kgdro.

[5] https://gharchive.org.

[6] https://cloud.google.com/bigquery.

[7] This label refers to improvements and new features.

[8] https://tickettagger.blob.core.windows.net/scripts/github-labels-top3-34k.sql.

[9] https://tickettagger.blob.core.windows.net/datasets/github-labels-top3-34k.csv.

Table 1

Precision, Recall and F-measure of *bug*, *enhancement* and *question* labels for Ticker Tagger and the baseline J48 model, obtained using a 10-fold cross validation over $D_{balanced}$. Differences with the baseline approach are highlighted in **bold**.

Approach	Metrics	Bug	Enhancement	Question
Ticker Tagger	Precision	0.82 (+**0.24**)	0.89 (+**0.29**)	0.78 (+**0.13**)
	Recall	0.84 (+**0.25**)	0.76 (+**0.13**)	0.87 (+**0.26**)
	F-measure	0.83 (+**0.24**)	0.82 (+**0.20**)	0.83 (+**0.20**)
J48	Precision	0.58	0.60	0.65
	Recall	0.59	0.63	0.61
	F-measure	0.59	0.62	0.63

Table 2

Precision, Recall and F-measure of *bug*, *enhancement* and *question* labels, when Ticker Tagger is trained on $D_{balanced}$ and tested on $D_{unbalanced}$. The proportion of tickets is 48%, 41.8% and 10.2%, respectively.

Metrics	Bug	Enhancement	Question
Precision	0.79	0.73	0.44
Recall	0.72	0.74	0.53
F-measure	0.75	0.74	0.48

usefulness of our tool, as it achieves reasonably high performance in automatically recognizing issues reporting bugs or requesting features. These are the most important feedback for developers interested in performing software maintenance and evolution activities [15]. On the other hand, we believe that further efforts and tunings are required to improve the tool's capability of recognizing issues of the question type.

In recent work, Herbold et al. [19] considered Ticker Tagger in a quantitative comparison, showing that fastText outperforms the competition concerning the issue labeling problem, this without particular tuning. Herbold et al.'s approach achieves slightly higher precision results than our model because it leverages the auto-tuning feature, a feature that we did not use in Ticket Tagger. Thus, such small improvements in prediction performance are due to structural information about the issues used.

Discussion of confounding factors. There are several factors that can potentially influence Ticker Tagger's performance, as discussed below.

(i) *Impact of function words*: For issues belonging to the bug and enhancement classes both precision and recall are above 0.70, while Ticket Tagger produces higher numbers of false positives and false negatives for the question category, i.e., a lower precision and a lower recall are achieved for this class. We believe that the strong use of function words, e.g., "how" or "what" that typically introduce questions, in the issue title or description could lead the classifier to erroneously assign the question label to issues that actually belong to different classes and, consequently, this degrades the precision achieved for the question category. In addition, the lower recall obtained for this class could be connected with the fact that developers (and users) ask questions about a wide range of topics [20], making it hard to learn all the patterns that could lead to the assignment of this label.

(ii) *Impact of Language Consistency in Issue Tickets*: we observe whether the ticket's language affects the performance of our model. Thus, we generated two datasets, one containing 24,600 English tickets and one baseline dataset of 24,600 tickets with random tickets sampled using the same strategy described in Section *Dataset Construction*. To generate the dataset comprising 24,600 English tickets, we used a javascript port of *guess language*,[10] a tool using heuristics based on character sets and trigrams for automatically detecting the language of the text. Results in Table 3 suggest that language consistency in issue tickets has a positive effect on the classification performance.

(iii) *Presence of Code Snippets in Issue Tickets*: we observe whether the presence of code snippets in tickets affects the performance of our model. Thus, we generated two datasets, one characterized by 6,000 tickets containing code snippets and one baseline dataset of 6,000 tickets sampled at random using the previously mentioned method. In particular, the presence of code snippets is recognized by detecting pieces of text enclosed in triple backticks, which is the special syntax recommended by the *GitHub Flavored Markdown* language[11] to highlight code snippets. Results in Table 3 show that the presence of snippets does not significantly impact classification performance.

4. Threats to validity

Threats to construct validity. We compared Ticket Tagger with a baseline approach (J48) on a dataset comprising equal numbers of bugs, enhancements and questions. This could represent a threat to construct validity as in real scenarios the

---

Table 3

Precision, Recall and F-measure of *bug, enhancement* and *question* labels for Ticker Tagger are computed using a 10-fold cross validation. Only differences among the various treatments (CONSISTENT LANGUAGE and CODE SNIPPET PRESENCE) are reported for the sake of brevity.

Approach	Metrics	Bug	Enhancement	Question
CONSISTENT	Precision	−2.5%	+6.3%	+2.0%
LANGUAGE	Recall	+9.4%	−1.8%	+1.7%
	F-measure	+4.0%	+2.4%	+1.9%
CODE SNIPPET	Precision	+0.6%	−2.0%	−0.4%
PRESENCE	Recall	−3.1%	+2.5%	+0.5%
	F-measure	−0.3%	+0.1%	+0.0%

distributions of the different types of issues may be unbalanced. To counteract this issue, we also assessed Ticket Tagger on a second unbalanced dataset where the proportion between the different classes is close to reality.

Threats to internal validity. Our results could be misleading if a significant percentage of collected issues would be subject to re-labeling. To mitigate this concern and reduce the likelihood of re-labeling for the considered samples, we collected GitHub issues having the *closed* status assigned.

Threats to external validity. The main threat to external validity is related to the potential specificity of our datasets. The collected issues could not be adequately representative of all the issues present on GitHub. However, to increase the heterogeneity of data, we selected issues from projects (i) having different natures, (ii) implemented through different programming languages, and (iii) developed by different developers' communities. To further confirm the low specificity of our datasets and the quality of our results, in recent work Ticker Tagger was considered in a quantitative comparison [19], which demonstrated that fastText outperforms state-of-the-art approaches addressing the issue labeling problem.

5. Conclusion

In this work, we presented Ticket Tagger, an app that we released on the GitHub marketplace, that automatically assigns suitable labels to issues opened on GitHub projects. The core of Ticket Tagger is represented by a machine learning model that analyzes the title and the textual description of issues in order to determine whether such an issue can be labeled as a bug report, a feature request or a question.

With the aim of assessing the classification performance achieved by our tool, we conducted four main evaluation experiments. The results of such evaluation showed that Ticket Tagger allows to automatically assign labels with reasonably high levels of precision and recall, outperforming results of a baseline approach. Our findings have also shown that the use of a consistent language can improve Ticket Tagger classification performance, while the presence of code snippets does not affect the results significantly.

Future work will be aimed (i) at comparing Ticket Tagger's accuracy and functionality with other existing solutions, as well as (ii) at investigating its usefulness through the analysis of direct feedback from end-users.

CRediT authorship contribution statement

Rafael Kallis: Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Andrea Di Sorbo:** Conceptualization, Formal analysis, Investigation, Methodology, Validation, Visualization, Writing – original draft, Writing – review & editing. **Gerardo Canfora:** Supervision, Writing – review & editing. **Sebastiano Panichella:** Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

References

[1] A. Di Sorbo, G. Grano, C. Aaron Visaggio, S. Panichella, Investigating the criticality of user-reported issues through their relations with app rating, J. Softw. Evol. Process (2020) e2316, https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2316.

[2] P. Floris, H. Vogt Harald, How to save on software maintenance costs. Omnext white paper, Source Value 2 (2010).

[3] T.F. Bissyandé, D. Lo, L. Jiang, L. Réveillere, J. Klein, Y. Le Traon, Got issues? Who cares about it? A large scale investigation of issue trackers from github, in: International Symposium on Software Reliability Engineering, 2013, pp. 188–197.

[4] S. Panichella, G. Bavota, M.D. Penta, G. Canfora, G. Antoniol, How developers' collaborations identified from different sources tell us about code changes, in: ICSME, IEEE, 2014, pp. 251–260.

[5] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, S. Liu, Exploring the characteristics of issue-related behaviors in github using visualization techniques, IEEE Access 6 (2018) 24003–24015.

[6] J.L.C. Izquierdo, V. Cosentino, B. Rolandi, A. Bergel, J. Cabot, Gila: Github label analyzer, in: International Conference on Software Analysis, Evolution, and Reengineering, SANER, 2015, pp. 479–483.

[7] Q. Fan, Y. Yu, G. Yin, T. Wang, H. Wang, Where is the road for issue reports classification based on text mining?, in: International Symposium on Empirical Software Engineering and Measurement, ESEM 2017, 2017, pp. 121–130.

[8] J. Cabot, J.L.C. Izquierdo, V. Cosentino, B. Rolandi, Exploring the use of labels to categorize issues in open-source software projects, in: International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2015, pp. 550–554.

[9] R. Kallis, A. Di Sorbo, G. Canfora, S. Panichella, Ticket tagger: machine learning driven issue classification, in: International Conference on Software Maintenance and Evolution, ICSME, 2019, pp. 406–409.

[10] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, Y. Guéhéneuc, Is it a bug or an enhancement?: a text-based approach to classify change requests, in: Conference of the Centre for Advanced Studies on Collaborative Research, 2008, p. 23.

[11] Y. Zhou, Y. Tong, R. Gu, H.C. Gall, Combining text mining and data mining for bug report classification, J. Softw. Evol. Process 28 (3) (2016) 150–176.

[12] A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of tricks for efficient text classification, in: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3–7, 2017, in: Short Papers, vol. 2, 2017, pp. 427–431.

[13] A. Di Sorbo, S. Panichella, C.A. Visaggio, M. Di Penta, G. Canfora, H.C. Gall, DECA: development emails content analyzer, in: International Conference on Software Engineering, ICSE 2016 - Companion Volume, 2016, pp. 641–644.

[14] S. Panichella, A. Di Sorbo, E. Guzman, C.A. Visaggio, G. Canfora, H.C. Gall, How can I improve my app? Classifying user reviews for software maintenance and evolution, in: International Conference on Software Maintenance and Evolution, ICSME, 2015, pp. 281–290.

[15] A. Di Sorbo, S. Panichella, C.V. Alexandru, J. Shimagaki, C.A. Visaggio, G. Canfora, H.C. Gall, What would users change in my app? Summarizing app reviews for recommending software changes, in: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, 2016, pp. 499–510.

[16] D.M. Allen, The relationship between variable selection and data agumentation and a method for prediction, Technometrics 16 (1) (1974) 125–127, https://doi.org/10.1080/00401706.1974.10489157.

[17] R.A. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[18] E.R. Russo, A.D. Sorbo, C.A. Visaggio, G. Canfora, Summarizing vulnerabilities' descriptions to support experts during vulnerability assessment activities, J. Syst. Softw. 156 (2019) 84–99.

[19] S. Herbold, A. Trautsch, F. Trautsch, On the feasibility of automated prediction of bug and non-bug issues, Empir. Softw. Eng. 25 (6) (2020) 5333–5369.

[20] X. Yang, D. Lo, X. Xia, Z. Wan, J. Sun, What security questions do developers ask? A large-scale study of stack overflow posts, J. Comput. Sci. Technol. 31 (5) (2016) 910–924.