On the Characteristics and Applicability of Topic Modeling in Predicting Lifetime of GitHub Issues

Syed Fatiul Huq*, Asadullah Hill Galib†, Abu Rafe Md. Jamil‡ and Ahmedul Kabir§
Institute of Information Technology, University of Dhaka
Dhaka, Bangladesh
Email: *bsse0732@iit.du.ac.bd, †bsse0712@iit.du.ac.bd, ‡bsse0722@iit.du.ac.bd, §kabir@iit.du.ac.bd

***Abstract*—Topic modeling is the process of extracting keywords from documents to characterize and distinguish them from other documents. It is a process applied to summarize, compare and analyze large corpus of text. In the software engineering domain, it has been applied to mine repositories and extract valuable insight into the important properties and aspects of the project and its developers. One such important aspect of current project management efforts is the prediction of issue lifetime. This study conducts topic modeling on GitHub issues to observe patterns in the extracted topics and their performance as a feature for predicting the lifetime of issues. It is observed that issues from a large collection of projects can yield distinguishable and comprehensible topics. In terms of predictive performance, the prediction model with topic modeling performs better than the previous approach, with a high increase in precision and f1-measure. Evaluating these findings helps establish topic modeling as a viable feature in issue-based software development processes.**

***Index Terms*—topic modeling, issue lifetime prediction, mining software repositories**

I. Introduction

Information Retrieval approaches have become a crucial and frequently utilized aspect in the field of software engineering. Topic modeling is one such information retrieval approach that uses statistical modeling to discover "topics" that define a document. Topic modeling has been used as a general purpose tool across multiple domains, such as text summarization [1]–[3], text similarity [4], [5], sentiment analysis [6], [7], topic generation [8], [9], software traceability [10] etc.

The domain of software engineering has also seen the adoption of topic modeling. It has been incorporated into the field of mining software repositories, such as in source code analysis [11], cataloging repositories [12], project recommendations [13], topic evolution [14], identification of developers' characteristics [15], etc. The effectiveness of topic modeling in these aspects induces its potential in the context of project issues. Issues are an important artifact of online software project management systems, like GitHub[1], where bugs are reported, requests for new features are posted and development processes are discussed and tracked. Topic extracted from the text and comments in issues can provide insight into issues and be applied to predict their properties.

An important aspect of issues is their lifetime, which signifies the amount of time and effort exhausted for resolving the task. Predicting the issue resolution time is helpful in project management processes as it affects decisions for prioritization, characterization, developer assignment and more. There have been several approaches for predicting the resolution time using features like text similarity [16], affective emotions [17], dynamic and contextual features [18] and cross-project data [19]. However, topic modeling has yet been utilized for predicting issue lifetime, which would further augment its applicability in the software engineering domain.

This study aims to answer three Research Questions (RQ) to understand the characteristics and applicability of topic modeling in GitHub issues:

- **RQ1: What are the characteristics of topics in issues?** This RQ observes the comprehensibility of topics for combined and cross-project issues.
- **RQ2: How do issue topics perform in predicting issue resolution time?** This RQ generates a prediction model on issue lifetime incorporating issue topics and observes its performance compared to the existing approach.
- **RQ3: Which topics are important in predicting issue resolution time?** This RQ analyzes the individual predictability and significance of the extracted topics.

II. Related Work

Several studies previously dealt with GitHub issue lifetime and some works incorporated topic modeling in mining software repositories.

Weiss et al. [16] used past issue reports to predict effort for new issues. They identified similar past issues by measuring text similarity. They used KNN for effort estimation based on text similarity. They evaluated their approach on a single project, JBoss, with a high enough accuracy regarding actual effort. However, they did not adopt other static and dynamic features of issues and considered a single project only.

Rees-Jones et al. [19] incorporated cross-project data of ten projects instead of single project data. They used static features of projects and employed the decision tree algorithm for predicting issue lifetime. Their approach obtained high precision and low false alarms concerning existing works. However, they dealt with limited projects which may induce threats to external validity.

In addition to using static features, Kikas et al. [18] also incorporated dynamic and contextual features to forecast the issue's lifespan. They investigated issues from more than 4000

[1]https://github.com/

projects at several phases to take into account the evolving nature of the issues and the overall state of a project. They used the Random Forest algorithm by incorporating 21 features to anticipate the issue's lifetime within a fixed timeline. Their work obtained AUC scores ranging from 0.64 to 0.707. Besides, they also provided a ranking of features based on the features' importance for prediction.

None of the existing lifetime prediction studies used topic modeling that was used in the context of repositories for mining software. Antoniol et al. [20] used topic modeling for distinguishing Linguistic features which are used for classifying issue. To evaluate the naming convention in source code, Markovtsev et al. [11] integrated topic modeling into source code, and presented insights on topic modeling. Sharma et al. [12] used topic modeling for automatically cataloging GitHub repositories. Orii et al. [13] applied collaborative topic modeling (LDA) on source code for recommending similar projects. Thomas et al. [14] analyzed the evolution of topics across GitHub repositories by incorporating topic modeling into source code. Linstead et al. [15] used an LDA-based Author-Topic Model on Eclipse source code to identify the developer's characteristics and performance. These studies establish the viability of topic modeling in software development artifacts, motivating its application in predicting issue lifetime.

III. Methodology

The three research questions posed by this study contain independent approaches along with common processes. These include extracting topics from the issue documents and the use of dataset, which are described in the following subsections.

A. Topic Modeling

Latent Dirichlet Allocation (LDA) [21] is a common topic modeling technique which provides probability distributions of topic per document and word per topic. LDA deals with two concerns: how prevalent is a particular word across topics, and how prevalent are topics in the document? For this study, LDA's implementation would yield probability distributions of topics extracted from issues and the words that populate those topics. Each issue is treated as a single document, with its title, body and optionally its comments providing the text.

For this study, LDA is tuned with the following values, based on [22]:

- alpha = 0.5
- beta = 0.1
- iterations = 100
- fitting = Gibbs Sampling[2]

As a preprocessing step, code snippets and markdown are removed from the text. This is done because the language specific keywords in code e.g., "for", "if" etc. can divide topics based on the language used in the project. This would hamper the concern-wise categorization of issues, for instance, "debugging", "feature requests" etc.

[2]https://github.com/yangliuy/LDAGibbsSampling

Furthermore, documents with less than 20 words are excluded from the final dataset as small documents have the potential to skew the extracted topic.

B. Dataset

Two types of datasets are used in this study for extracting topics and predicting issue resolution time. Description of the two sets is provided as follows.

1) Combined Project Data: This study adopts the data used by Kikas et al. [18] (KDP), to utilize the features used in the literature and conduct a comparative evaluation. The dataset contains a total of 967,037 issues from 4,024 projects in GitHub. The project data was originally extracted from GHTorrent [23]. The issues contain contextual, static and dynamic information in the form of 50 features.

Along with feature information, the dataset provides cleaned text of the issues' title, body and comments. These artifacts are necessary to conduct topic modeling.

2) Cross-Project Data: A shortcoming of the dataset by KDP is that it does not distinguish the issues based on the projects these are contained in. This, therefore, hinders the opportunity for cross-project topic analysis. In this study, 3 GitHub projects, as listed in Table I, are mined. The repositories are mined using the GitHub API.

TABLE I
Repositories for cross project analysis

Project	Issues	Comments
Google Guava	2433	12630
Mockito	726	3553
Facebook Android SDK	295	1023

IV. Experimentation and Results

This section describes the approaches for answering the Research Questions (RQs) and their associated findings.

A. RQ1: Issue Topic Characteristics

This RQ qualitatively analyzes the characteristics of topics extracted from issues. The aim of this RQ is to understand the characteristics of the topics extracted, for instance, the comprehensibility of the containing words, ease of labeling and how differentiable the topics are.

Procedure: First, ten topics are extracted from KDP's combined set of issues. The extracted topics are sorted based on popularity. Popularity is measured by the average of each topic's probabilities in the individual documents. These are derived from LDA's document to topic probability distribution. Then, for the ten topics, labels are assigned based on the top words of the topics. The top words are collected from LDA's topic to word probability distribution. Table II lists the ten sorted topics, complete with top words and assigned labels.

As a separate step, the issues from the three repositories are used for topic modeling. Since the corpus is significantly smaller than the combined dataset, only five topics are extracted for each project. Table III shows the top words for each topic under the three projects.

TABLE II
COMBINED PROJECT TOPICS

Topic	Top words	Label
1	user, page, add, http, link, creat, list, search, show, post	Feature Request
2	time, make, issue, work, way, test, seem, chang, know	Management
3	function, work, valu, code, call, event, set, class, method	Code
4	imag, button, click, icon, show, http, menu, screen, window	GUI
5	file, error, http, instal, run, build, test, version, packag	Deployment
6	server, connect, run, file, version, support, error, instal, start	Deployment
7	line, error, file, string, object, type, id, http, return, valu	Code
8	de, http, la, le, en, se, el, silli, para, verbos, na	Ambiguous
9	info, debug, error, sourc, unknown, null, crash, sever	Bug
10	avail, initi, construct, unload, preiniti, postiniti, impserver	Initialization

TABLE III
CROSS-PROJECT TOPICS

Topic	Google Guava	Mockito	Facebook Android SDK
1	method, case, methods, make, code, way, implementation	release, version, mockito, make, versions, [x], beta, time, java	facebook, app, sdk, android, user, application, code, api, issue, access
2	_[original, posted, pm_, am_, **status.**, comment, entered	api, mockito, junit, mock, public, code, powermock, static	pc, method, dscount, prio, scount, sched, nice, group
3	guava, java, version, gwt, problem, maven, issue, class, fix	mockito, version, java, build, issue, byte, fix, seems, thanks, test	i/debug, dialog, login, screen, webview, permissions, page, click
4	interfaces, issue, file, google, classpath, jar, coming, unknown	mockito, error, package, mock, class, issue, version, android, illegal	i/debug, pc, info/debug, key, url, heap, hash, debug
5	public, static, extends, type, null, method, return, string	method, test, mock, class, code, mockito, type, return, methods	sdk, facebook, issue, android, thanks, supported, developer

Findings: In combined project topics, as displayed in Table II, topics can be successfully categorized. Each of the ten issue topics contains words that are distinct and, in most cases, interpretable. Despite some words omitting prefix and suffixes, the core word conveys their context. Other than the eighth topic, which mostly contains words that cannot be identified and aptly named *Ambiguous*, all the topics are categorized. For instance, due to words like "imag", "button", "click", "icon" and more, the fourth topic is named *GUI*.

Among these topics, the most popular topic in GitHub's issues is *Feature Request*. This is similar to the findings of Liao et al. [24] where new feature related issues have a majority. Other important aspects of software development like *GUI*, *Bug*, *Deployment* etc. are also present in the topic list. This signifies the applicability of topic modeling in GitHub issues for automating the categorization of new issues.

For the cross-project issue topics, categorization was not possible. This was due to very little inter-project topic similarity, repetitions of top words and the saturation of project-specific keywords. Due to this ambiguity, cross-project issues were deemed unfit for issue lifetime prediction.

B. RQ2: Issue Lifetime Prediction

This RQ analyzes the applicability of topic modeling in issues by predicting issue lifetime with the extracted topics and comparatively evaluating with KDP, the base model.

Procedure: To understand the predictive property of topic features, prediction models are executed for two feature sets. In the first setup, only the topic-associated feature set is considered. For the ten extracted topics from the combined issue set, a document to topic probability distribution is calculated using LDA. However, for this dateset, comments are excluded from the issue text as comments are time-dependent dynamic features. In this paper, only static features – issue texts (title and description) – are considered for lifetime prediction.

TABLE IV
PREDICTIVE PERFORMANCE OF THE TWO MODELS – ONE WITH ONLY TOPICS AS FEATURES AND THE OTHER INCLUDING STATIC ISSUE FEATURES – COMPARED TO KDP

Days	Metric	KDP	Topic-only	Topic+static
1	precision	N/A	0.24	**0.3**
	recall	N/A	0.52	**0.64**
	f1	N/A	0.33	**0.41**
7	precision	0.26	0.41	**0.48**
	recall	**0.82**	0.5	0.67
	f1	0.39	0.45	**0.56**
14	precision	0.13	0.47	**0.55**
	recall	**0.8**	0.57	0.67
	f1	0.23	0.5	**0.59**
30	precision	0.16	0.54	**0.62**
	recall	**0.8**	0.58	0.67
	f1	0.27	0.55	**0.64**
90	precision	0.25	0.62	**0.69**
	recall	**0.79**	0.55	0.67
	f1	0.38	0.57	**0.68**
180	precision	0.2	0.66	**0.72**
	recall	**0.81**	0.51	0.67
	f1	0.32	0.6	**0.7**
365	precision	0.21	0.69	**0.74**
	recall	**0.89**	0.5	0.68
	f1	0.34	0.55	**0.71**

*(The bold values represent the best result for that metric)

In the second setup, Rees-Jones et al's [19] feature set is adopted along with the ten topics. Their feature set of seven static features is a reduced version of KDP's. The rationale for the feature reduction is that the original feature set

TABLE V
INDIVIDUAL TOPIC IMPORTANCE

Days	Feature Request Topic 1	Management Topic 2	Code Topic 3	GUI Topic 4	Deployment Topic 5	Deployment Topic 6	Code Topic 7	Ambiguous Topic 8	Bug Topic 9	Initialization Topic 10
1	0.21	0.03	0.08	0.04	0.12	0.05	0.08	**0.23**	0.08	0.07
7	**0.2**	0.11	0.08	0.04	0.05	0.07	0.19	0.11	0.06	0.08
14	0.19	0	**0.15**	0.1	**0.15**	0.08	0.08	0.06	0.13	0.06
30	0.07	0.03	0.07	0.08	0.13	0.09	**0.16**	0.14	**0.16**	0.07
90	0.06	0.02	0.12	0.18	0.13	0.03	0.13	0.14	**0.21**	0
180	0.1	0.06	0.1	0.08	**0.15**	0.14	0.13	0.12	0.11	0.03
365	0.08	0.12	0.08	0.11	0.12	0.1	0.1	**0.16**	0.13	0

*(The bold values represent the most important topic for a horizon)

contains many dynamic features as well as unspecified feature (i.e., textScore). They also demonstrate in their approach that the reduced static features perform better than all of KDP's features. In this work, these seven static features are considered and incorporated with the ten topic features.

Seven prediction horizons (1, 7, 14, 30, 90, 180 and 365 days) are considered according to KDP. A prediction horizon *n* is used for predicting if an issue closes between 0 and *n* days. The starting range remains at 0 for all horizons since only static features are taken for this study. Each range provides a common time-frame: a day, a week, two weeks, a month, three months, half a year and a year respectively.

The adoption of prediction horizons converts this prediction process into an N-binary classification problem, which can easily be interpreted. Random Forest algorithm is performed on the dataset for model training and testing. Here the parameters of the Random Forest algorithm is set as per KDP's work.

Before conducting the classification, preprocessing steps are conducted on the dataset. Firstly, the dataset is balanced, since the target value is unbalanced for the different prediction horizons. For instance, for the 1 day horizon, approximately 70% issues are open. The inverse is true for the 365 days horizon. Secondly, the features are normalized, since the topic features and static features contain different ranges.

Findings:

From the numeric results displayed in Table IV, the following findings can be observed.

Firstly, this model with topic-only features generates consistently higher precision than KDP. Conversely, however, KDP's recall is higher for all seven prediction horizons. To balance out the conflict between precision and recall, the f1 score is considered for understanding the overall correctness of the model. In this case, the topic-based model contains a better f1 score than KDP throughout the seven horizons. This is true for both topic-only and topic+static features.

To expand on the results, the payoff between precision and recall can be inferred from what they signify in this context. A low recall model will predict issues that close before n days to be finished after the horizon n. A low precision model will predict issues that close over the horizon n to be finished before n days. In other words, this model would classify some short issues as long and KDP the opposite.

While this model does overestimate lifetime for some issues, longer issues would less likely be misclassified as short. As such, developers will not be assigned less time than what is required, potentially preventing unwanted adverse effects like bugs to be inserted due to an unrealistic and tight deadline.

It can be concluded that topic-modeling is an effective tool for classifying issues and predicting their lifetime.

C. RQ3: Topic Feature Importance

Each of the ten topics represents a category of issues. From the prediction model, individual numeric significance of the topics can be extracted. This RQ aims to infer the relation between the topics or categories and the lifetime of issues.

Procedure: Using the Random Forest Classifier, the feature importance of the topic associated features is calculated for all the seven prediction horizons.

Findings: Feature importance of the 10 topics in different horizons is listed in Table V, with the following observations.

1) The topic *Feature Request* plays a significant role in predicting the issue lifetime in the 1 and 7 days horizon. As *Feature Request* issues relate to adding a whole new feature to the software, these issues can take a higher number of days to be completed. Among the three types of common changes – corrective, perfective and adaptive – feature requests, which are adaptive changes, take the most amount of time to complete [25] [26]. Hence, these issues cannot usually be closed within a day or a week, enabling it to be an important feature when predicting whether issues can be closed early.
2) Compared to adaptive changes, corrective ones – bug fixing, error handling – take less time to complete [25] [26]. This is congruent with the findings of this study, where the topic *Bug* emerges a significant feature in the 30 and 90 days horizons. This can be inferred as the tendency of bug fixing issues being closed within a month or at most three months.
3) A similar trend is present for the topics titled *Code*. Issues that heavily mention the source code can be interpreted as perfective maintenance, since these focus on the code instead of any product-level concerns. Such issues, like refactoring requests, deal with modifications to the code for better efficiency or comprehensibility without altering the output of the end product. The findings show that these are significant features in the 14 and 30 day horizons, earlier than the corrective ones.

This result conforms to perfective maintenance being the shortest of the three development tasks [25].

4) Other than significant topics, two have shown zero importance as features for different horizons: *Management* and *Initialization*. The latter is unimportant for two horizons while having low importance for the rest. As the topics are sorted based on popularity, it can be interpreted that the least popular topic tends to be unimportant in predicting issue lifetime.

V. Threats to Validity

For RQ1, the analysis on cross-project topics was only conducted on three projects. This lessens generalizability, threatening the external validity. Also, for the combined projects' topics, the labels are manually assigned, which is susceptible to bias and can be interpreted differently by different analysts. However, these do not change the outcome of the prediction model and consequently, the applicability of topic modeling.

For the prediction model, the dataset is adopted from KDP's study [18]. Hence, the threats to validity of their dataset also apply to this study.

VI. Future Work

With its promising results, this paper can be expanded upon.

Firstly, the breadth of the cross-project analysis will be increased. While the three projects provide an idea of how topics would behave, a dataset comparable to the combined project one can better justify the claims.

Secondly, the implementation of LDA will be expanded, by considering its non-deterministic properties.

Lastly, the intrinsic significance of individual topics in the predictive model will be further inferred with the inclusion of empirical studies with GitHub issues.

VII. Conclusion

This study conducted topic modeling on GitHub issues and analyzed the patterns of the extracted topics and its applicability for predicting issue lifetime. It is seen that, when combined from multiple projects, the issues generate comprehensible topics that enable concern-specific labeling. Next, these were used for predicting issue resolution time. With an N-binary classification using Random Forest, it was observed that the inclusion of topics as features improves precision and f1-score, with respect to KDP's. Lastly, analyzing individual importance of the topics showed that the context and popularity of topics can be used to interpret the issue longevity.

References

[1] G. Ercan and I. Cicekli, "Lexical cohesion based topic modeling for summarization," in *International Conference on Intelligent Text Processing and Computational Linguistics*. Springer, 2008, pp. 582–592.

[2] W. Gao, P. Li, and K. Darwish, "Joint topic modeling for event summarization across news and social media streams," in *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 2012, pp. 1173–1182.

[3] A. Haghighi and L. Vanderwende, "Exploring content models for multi-document summarization," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2009, pp. 362–370.

[4] Z. Wang, M. He, and Y. Du, "Text similarity computing based on topic model lda," *Computer science*, vol. 40, no. 12, pp. 229–232, 2013.

[5] J. Lin and W. J. Wilbur, "Pubmed related articles: a probabilistic topic-based model for content similarity," *BMC bioinformatics*, vol. 8, no. 1, p. 423, 2007.

[6] T. H. Nguyen and K. Shirai, "Topic modeling based sentiment analysis on social media for stock market prediction," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2015, pp. 1354–1364.

[7] B. Lu, M. Ott, C. Cardie, and B. K. Tsou, "Multi-aspect sentiment analysis with topic models," in *2011 IEEE 11th international conference on data mining workshops*. IEEE, 2011, pp. 81–88.

[8] H. M. Wallach, "Topic modeling: beyond bag-of-words," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 977–984.

[9] Y. Gao, Y. Xu, Y. Li, and B. Liu, "A two-stage approach for generating topic models," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2013, pp. 221–232.

[10] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1. IEEE, 2010, pp. 95–104.

[11] V. Markovtsev and E. Kant, "Topic modeling of public repositories at scale using names in source code," *arXiv preprint arXiv:1704.00135*, 2017.

[12] A. Sharma, F. Thung, P. S. Kochhar, A. Sulistya, and D. Lo, "Cataloging github repositories," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2017, pp. 314–319.

[13] N. Orii, "Collaborative topic modeling for recommending github repositories," *Inf. Softw. Technol.*, vol. 83, no. 2, pp. 110–121, 2012.

[14] S. W. Thomas, "Mining software repositories using topic models," in *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 2011, pp. 1138–1139.

[15] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, "Mining eclipse developer contributions via author-topic models," in *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 2007, pp. 30–30.

[16] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 2007, pp. 1–1.

[17] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli, "Are bullies more productive? empirical study of affectiveness vs. issue fixing time," 05 2015.

[18] R. Kikas, M. Dumas, and D. Pfahl, "Using dynamic and contextual features to predict issue lifetime in github projects," in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 291–302.

[19] M. Rees-Jones, M. Martin, and T. Menzies, "Better predictors for issue lifetime," *arXiv preprint arXiv:1702.07735*, 2017.

[20] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests." in *CASCON*, vol. 8, 2008, pp. 304–318.

[21] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[22] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National academy of Sciences*, vol. 101, no. suppl 1, pp. 5228–5235, 2004.

[23] G. Gousios and D. Spinellis, "Ghtorrent: Github's data from a firehose," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 2012, pp. 12–21.

[24] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, and S. Liu, "Exploring the characteristics of issue-related behaviors in github using visualization techniques," *IEEE Access*, vol. 6, pp. 24 003–24 015, 2018.

[25] A. Murgia, G. Concas, R. Tonelli, M. Ortu, S. Demeyer, and M. Marchesi, "On the influence of maintenance activity types on the issue resolution time," in *Proceedings of the 10th international conference on predictive models in software engineering*, 2014, pp. 12–21.

[26] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 2007, pp. 1–1.