An Empirical Investigation of Incident Triage for Online Service Systems

Junjie Chen[1,2], Xiaoting He[3], Qingwei Lin[3], Yong Xu[3], Hongyu Zhang[4], Dan Hao[1,2]
Feng Gao[5], Zhangwei Xu[5], Yingnong Dang[5], Dongmei Zhang[3]
[1]Key Laboratory of High Confidence Software Technologies (Peking University), MoE
[2]Institute of Software, EECS, Peking University, Beijing, 100871, China
{chenjunjie,haodan}@pku.edu.cn
[3]Microsoft Research, Beijing 100080, China,{v-xiah,qlin,yox,dongmeiz}@microsoft.com
[4]The University of Newcastle, NSW 2308, Australia, hongyu.zhang@newcastle.edu.au
[5]Microsoft, Redmond, WA 98052, USA,{fgao,zhangxu,yidang}@microsoft.com

***Abstract*—Online service systems have become increasingly popular. During operation of an online service system, incidents (unplanned interruptions or outages of the service) are inevitable. As an initial step of incident management, it is important to be able to automatically assign an incident report to a suitable team. We call this step *incident triage*, which can significantly affect the efficiency and accuracy of overall incident management. To better understand the incident-triage practice in industry, we perform an empirical study of incident triage on 20 large-scale online service systems in Microsoft. We find that incorrect assignment of incident reports occurs frequently and incurs unnecessary cost, especially for the incidents with high severity. For example, about 4.11% to 91.58% of incident reports are reassigned at least once and the average increment in incident-triage time caused by the reassignments is up to 10.16X. Considering the similarity between bug triage (automatically assigning bug reports to software developers) and incident triage, we then explore the applicability of typical bug-triage techniques to incident triage for online service systems. The results demonstrate that these bug-triage techniques are able to correctly assign incident reports to a certain extent, but still need to be further improved, especially for the incident reports that are assigned incorrectly at the first time. We further discuss possible ways to improve the accuracy of incident triage based on the empirical study. To our best knowledge, we are the first to investigate incident triage in industrial practice. Our results are useful for both practitioners and researchers to develop methods and tools to improve the current incident-triage practice for online service systems.**

I. Introduction

Online service systems, such as Microsoft Office 365, have become increasingly popular. Despite of various quality control measures, during actual operation of an online service system, there are always incidents, which are unplanned interruptions and outages of the service [1]. These incidents can lead to huge economic loss and serious consequences. For example, the estimated cost of the one-hour downtime for `Amazon.com` on Prime Day this year (its biggest sale event of the year) is up to $100 million[1]. Also, according to a study conducted on 63 data center organizations in the U.S., the average cost of service downtime has steadily increased from $505,502 in 2010 to $740,357 in 2016[2].

Once an incident of an online service system occurs, it needs to be mitigated as soon as possible. The goal is to minimize the service downtime and to ensure high quality of the provided service. Currently, incident management has become a critical task for online service systems. A typical procedure of incident management is as follows. When an incident is detected by engineers or machine-based alerts (also called monitors), a corresponding incident report is created and submitted to the incident management system. Then, the incident report is assigned to the responsible team and an incident investigation process is triggered. Given an incident report, the engineers in the responsible team need to understand what the problem is and then mitigate it. More details about incident management will be presented in Section II-A.

As an initial step of incident management, it is important to be able to automatically assign an incident report to a responsible team. We call this step *incident triage*[3]. In particular, incident triage can significantly affect the follow-up steps and the efficiency and accuracy of overall incident management. If an incident report is assigned to a wrong team, the mitigation of the incident could be delayed and more cost could be incurred. Therefore, accurate incident triage is essential.

In the literature, there are a large number of studies on bug triage, which focuses on the automatic assignment of bug reports to software developers [2]–[4]. However, incident triage for online service systems is still unexplored. As we will show in this paper (Section IV), incident triage and bug triage have some different characteristics, although they have much in common. For example, bug reports are reported and treated individually in the bug-triage context, while many incident reports tend to have correlations, i.e., time correlation (reported around similar time) or location correlation (reported from close locations). One major reason is that an incident of an online service system can lead to a series of other

[1] https://www.businessinsider.com/amazon-prime-day-website-issues-cost-it-millions-in-lost-sales-2018-7.

[2] https://www.ponemon.org/blog/2016-cost-of-data-center-outages.

[3]This concept is analogous to traditional bug triage.

incidents, which can be detected by different monitors. All these monitors can create and submit unique incident reports, although these incidents actually have the same root cause. In this case, the incident reports caused by the same root cause tend to have similar reporting time or be reported from close locations. Therefore, it is yet to be verified if the existing bug-triage techniques can be applied to the incident-triage practice.

To better understand the incident-triage practice in industry, we performed an empirical study of incident triage on 20 real-world, large-scale online service systems in Microsoft, including several widely-used Microsoft products such as Office 365, Skype, Visual Studio, etc. In this study, we used over 30GB incident reports[4], and the total number of teams involved in these incident reports is over 20K. Here a team refers to a group of engineers responsible for investigating, mitigating, and resolving incidents. There can be a large number of teams supporting the same online service system. In the study, we investigated the problem of incident reassignment during the incident-triage process, which refers to reassigning an incorrectly assigned incident report at the first time to another potentially responsible team, and the cost associated with reassignment. Through the empirical study, we find that incident reassignment occurs frequently for online service systems, i.e., the reassignment rate ranges from 4.11% to 91.58% for the 20 studied online service systems. The reassignment largely incurs unnecessary cost, especially for the incidents with high severity. For example, the average increment in incident-triage time due to reassignment is up to 10.16X on the 20 studied online service systems. Therefore, accurate incident triage is definitely desired by online service systems in industry. To our best knowledge, this is the first empirical study to investigate incident triage in industrial practice.

Considering the similarities and differences between bug triage and incident triage, we further investigated the effectiveness of typical bug-triage techniques [2]–[8] on incident triage for online service systems. More specifically, we evaluated six typical bug-triage techniques using five kinds of technical aspects, i.e., machine-learning based techniques [9], [10][5], deep-learning based technique [11], topic-model based technique [12], tossing-graph based technique [3], and fuzzy-set based technique [13], based on 10 online service systems in Microsoft. The experimental results demonstrate that the studied bug-triage techniques are able to assign incident reports to responsible teams to a certain extent. For example, the studied bug-triage techniques are able to accurately identify the responsible teams for 32% – 71% incident reports. However, the accuracy still needs to be further improved, especially for the incident reports that are assigned incorrectly at the first time. More specifically, the studied bug-triage techniques can identify the responsible teams for only 17% – 52% incident reports that are assigned incorrectly at the first time.

[4]Due to the policy of Microsoft, we cannot disclose the actual number of incident reports in this paper.

[5]In this work, machine learning refers to traditional machine learning algorithms.

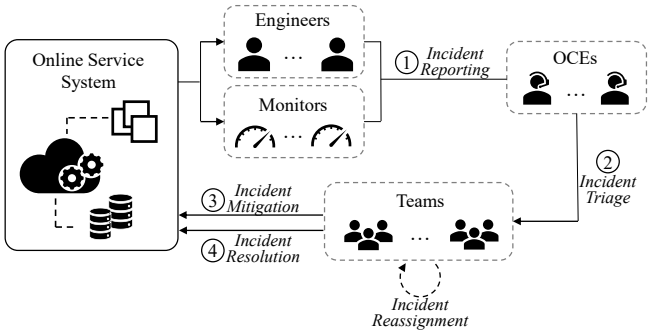

Fig. 1: The overview of incident management

By analyzing the studied bug-triage techniques in the industrial incident-triage context, we obtain some insights that can be used to better solve the incident-triage problem for online service systems. For example, as a large number of incidents are detected by monitors, we can leverage the monitoring data to further improve incident triage, such as service-level logs, performance counters, and machine/process/service-level events. Also, instead of treating each incident report individually during triage, we can leverage past incident reports that were submitted around the similar time or from close locations.

To sum up, this work has the following major contributions:

- We reported the first empirical study of incident triage on real-world, large-scale online service systems.
- We conducted the first empirical evaluation of bug-triage techniques for the incident-triage practice.
- We discussed a series of implications that can better solve the incident-triage problem for online service systems in practice.

The remaining sections of this paper are organized as follows. Section II presents the empirical study to investigate incident triage for online service systems in practice. Section III presents the empirical evaluation of the existing bug-triage techniques for incident triage. Section IV and Section V discuss the implications for improving incident triage and the threats to validity in our studies, respectively. Section VI presents the related work and Section VII concludes our work.

II. An Empirical Study on Incident Triage

In this section, we report the first empirical study of incident triage for real-world online service systems. In this study, we used 20 online service systems in Microsoft as subjects, including several widely-used Microsoft products such as Office 365, Skype, Visual Studio, etc. Due to the policy of Microsoft, we only report rough data and hide the time period of these incident reports. The total size of the incident reports for the 20 subjects used in this study is over 30GB and the total number of teams is over 20K. To our best knowledge, this is the most large-scale study that explores the triage problem (including bug triage) in the literature. In the following, we first introduce the incidents of an online service system in Section II-A, and present the empirical investigation on incident reassignment during the incident-triage process in Section II-B.

A. Incidents

An online service system involves many components, such as virtual machines, database, network, etc. All these components can become the sources of the incidents in the daily operation of an online service system. That is, the incidents could be caused by many factors, such as source code bugs, misconfigurations, network traffic, and hardware failures. To provide 24x7 highly available service for millions of users around the world, any unplanned incident for an online service system is serious. Currently, incident management has become a critical task for online service systems. In particular, Figure 1 shows the overview of incident management for an online service system. A typical procedure of incident management goes through four steps: incident reporting, incident triage, incident mitigation, and incident resolution.

1) Incident Reporting: For an online service system, incidents are detected by either engineers or machine-based alerts (also called monitors). Engineers could observe incidents during their daily operation, and then they can manually submit the incident reports through an incident report portal. Monitors can detect incidents by monitoring the data at the runtime of the online service system, such as service-level logs, performance counters, and machine/process/service-level events. These monitoring data typically contains the information that reflects the runtime state and behavior of the online service system. Based on these data, incidents of the online service system can be detected by monitors in a timely way. Then, the monitors can automatically submit the corresponding incident reports by rendering certain templates.

In particular, each incident has a severity level, which is set based on its potential impact on the users. In Microsoft, there are five types of incident severity levels, i.e., 0 – 4, where 0 represents the highest severity level and 4 represents the lowest severity level.

2) Incident Triage: Once an incident is reported, the incident management system first makes a phone call to a set of On-Call Engineers (OCEs) to trigger the investigation process of the incident, in order to restore the service as soon as possible. Ideally, OCEs can identify the root cause of the incident based on the information in the incident report and resolve it quickly. However, mostly, OCEs are unable to diagnose the root cause within a short time. Therefore, they have to manually assign the incident report to a team that they think is the most suitable to handle it. This process is called *incident triage*. Sometimes, the engineers in the assigned team may find that they are actually not responsible for the incident, and thus they can reassign the incident report to another team that is potentially responsible. This process is called *incident reassignment*. It is not trivial to identify the correct team that an incident report should be assigned to. Section II-B will further explain the situation of incident triage in detail.

3) Incident Mitigation: When the incident report is assigned to the correct team, the incident investigation process is triggered. That is, the engineers in the team start to investigate what the problem is and mitigate it as soon as possible. In a large-scale online service system, completely resolving an incident tends to take quite some time that the service cannot afford. Therefore, the best way is to first quickly mitigate and bring the service back to normal and then do deeper investigation later. An example of mitigation is that, if an incident is related to SQL servers, the mitigation method may be to reboot the abnormal SQL servers.

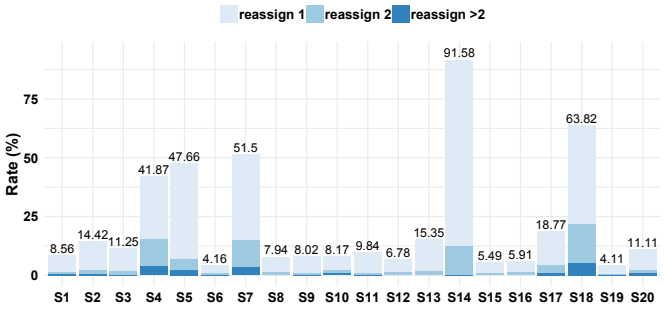

Fig. 2: Incident reassignment rate for each subject

4) Incident Resolution: After mitigating the incident, the engineers in the team then identify and fix the underlying root cause for the incident through offline postmortem analysis. In particular, all the relevant information used for resolving the incident could be recorded for future usage. After resolving the incident, the engineers close the incident report in the incident management system.

B. An Empirical Investigation of Incident Reassignment

Based on the 20 online service systems in Microsoft, we investigated the practice of incident triage from the following two aspects:

- $\mathcal{A}$**1**: What is the *rate* of incident reassignment?
- $\mathcal{A}$**2**: What is the *cost* of incident reassignment?

In particular, the first aspect expects to motivate the problem of incident triage for online service systems by exploring the frequency of incident reassignment, and the second aspect aims to investigate the impact of incident reassignment.

1) The Reassignment of Incident Reports: Figure 2 shows the reassignment rate of incident reports for each subject system. In this figure, the x-axis represents the used subjects. The y-axis represents the reassignment rates, which is computed by dividing the number of incident reports involving reassignment by the total number of incident reports. Here we considered three cases of reassignment, i.e., the cases in which the number of reassignment is 1, the number of reassignment is 2, and the number of reassignment is larger than 2. The value over each bar represents the reassignment rate (considering all the three reassignment cases) of incident reports for the corresponding subject. Figure 2 shows that, for the 20 subjects (S1 to S20), the reassignment rate of incident reports ranges from 4.11% to 91.58%. The result demonstrates that incorrect assignments for incident reports are ubiquitous. In other words, every subject has a non-negligible ratio of incident reports that are assigned incorrectly at the first time. In particular, there are five subjects for which more than 40% incident reports have to be reassigned. Also, for some subjects such as *S4*,

S7, and *S18*, a number of incident reports are even reassigned several times (i.e., more than twice). These results indicate that incident reassignment indeed frequently occurs and is quite serious for some of the online service systems, which could cause unnecessary delays in incident mitigation and resolution.

TABLE I: Average reassignment rate for the incident reports of the 20 subjects at each severity level (%)

Severity	**0**	**1**	**2**	**3**	**4**
#reassignment=1	1.17	**35.85**	20.70	20.28	11.77
#reassignment=2	0.16	3.39	**4.78**	4.51	2.14
#reassignment>2	0.06	2.18	**2.35**	1.46	0.90
All	1.39	**41.42**	27.82	26.25	14.81

We further analyzed the average reassignment rate for the incident reports of the 20 subjects at each severity level (level 0 to level 4). The results are shown in Table I, where each cell represents the average reassignment rate for the incident reports at certain severity level in the corresponding reassignment case. The bold value in each row represents the largest value of the row, referring to the largest reassignment rate across all severity levels in the corresponding reassignment case. From the last row in this table, we can see that except the highest severity level (i.e., level 0), the incident reports at higher severity levels have larger reassignment rates, indicating that it is harder to accurately assign more severe incidents at the first time. Since more severe incidents tend to lead to more serious impacts, it further motivates the necessity of accurate incident triage.

2) The Cost of Reassignment: Figure 3 shows the analysis results about reassignment cost, i.e., the increment in incident-triage time (in terms of multiples of time) due to reassignment. Here the incident-triage time refers to the time from incident reporting to the incident being assigned to the correct team. Due to the company policy, we can only report the relative time instead of the absolute time. In this figure, each bar represents the average increment in incident-triage time due to reassignment for the corresponding subject. The increment ranges from 1.09X to 24.32X, and the average increment on all the subjects is up to 10.16X. This result demonstrates that reassignment could incur huge cost, showing the strong demand for accurate incident triage.

We also analyzed the cost of reassignment for incident reports of the 20 subjects at each severity level (level 0 to level 4). The results are shown in Table II. In this table, each cell represents the average increment in incident-triage time (in terms of multiples of time) due to reassignment on all the subjects at the corresponding severity level. The bold value refers to the largest increment across the severity levels. From this table, we can see that for the incident reports at the lowest severity level (level 4), the increment is the highest (i.e., up to 27.09X). It may be still acceptable to spend relatively longer time to assign the less severe incident reports. However, for the rest four severity levels (level 0 to level 3), the increment at a higher severity level is larger than that at a lower severity level. That is, the reassignment cost tends to be more significant for more severe incidents. It is very harmful since it may prolong the incident diagnosis time and could lead to more serious consequences. Therefore, it is very necessary to improve the accuracy of incident triage.

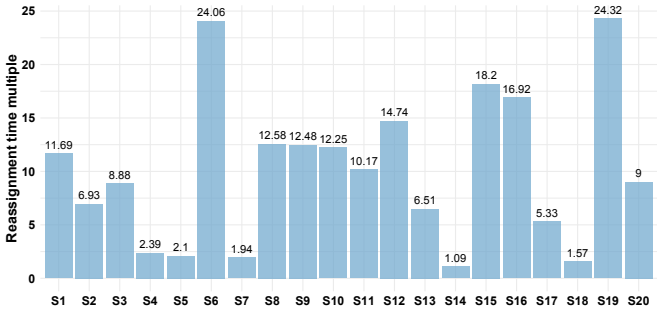

Fig. 3: Average increment in incident-triage time (in terms of multiples of time) caused by reassignment

TABLE II: Average increment in incident-triage time (in terms of multiples of time) due to reassignment on all the subjects at each severity level

Severity	**0**	**1**	**2**	**3**	**4**
Multiple	15.17	11.40	7.15	7.51	**27.09**

In summary, by exploring the two aspects on the 20 online service systems in Microsoft, we obtain the following two major findings:

- Incident reassignment frequently occurs, i.e., the reassignment rates range from 4.11% to 91.58% for the 20 studied online service systems, and incidents at a higher severity level tend to have a larger reassignment rate;
- Incident reassignment largely aggravates the incident-triage cost, especially for the incidents at higher severity levels. For example, the average increment in incident-triage time due to reassignment is up to 10.16X on the 20 studied online service systems.

Therefore, accurate incident triage is definitely desired by online service systems in industry.

III. An Empirical Evaluation of Bug-Triage Techniques for Incident Triage

To solve the incident-triage problem for online service systems, we conducted the first empirical study to investigate **how traditional bug-triage techniques perform in this context**. In this section, we first introduce the studied typical bug-triage techniques in our incident-triage context in Section III-A, then present our study design in Section III-B, and finally present the results and analysis in Section III-C.

A. Studied Bug-Triage Techniques

In the literature, a large number of bug triage techniques have been proposed [3], [9]–[16]. According to the technical aspects they use, we classified them into five categories. We then selected one or two typical techniques from each category as the representatives to evaluate the effectiveness

on incident triage for online service systems. Table III shows the studied techniques in this work (i.e., six techniques from five categories in total). In this table, the second and third columns represent the name of each studied technique called in this paper for the ease of presentation and the technical aspect used in the corresponding technique, respectively.

TABLE III: Studied bug-triage techniques

Work	Name	Technical Aspect
Anvik et al. [9]	ML_I	machine learning (individual)
Jonsson et al. [10]	ML_E	machine learning (ensemble)
Lee et al. [11]	DL	deep learning
Naguib et al. [12]	TM	topic model
Jeong et al. [3]	TG	tossing graph
Tamrawi et al. [13]	FS	fuzzy set

We selected these techniques based on the following criteria: 1) The information leveraged by a bug-triage technique can be acquired in our incident-triage context. For example, summary and description in a report can be acquired, while developers' social network can hardly be acquired in the context since incident triage targets at teams instead of developers; 2) The source-code-based techniques are excluded, since besides code bugs, incidents are caused by various other factors, such as service unavailable and misconfigurations; 3) For each category we selected the state-of-the-art or widely-studied techniques as the representatives. Due to the differences between bug triage and incident triage, we adapted the studied traditional bug-triage techniques in our incident-triage context, e.g., using teams to replace developers. Next, we introduce the adapted bug-triage techniques in detail. More traditional bug-triage techniques will be presented in Section VI-A.

1) Machine-Learning based Techniques: Machine-learning based bug triage regards the problem as a supervised classification problem. It uses previously resolved bug reports to train a classifier, which is able to recommend assignments for new bug reports. In this category, we chose two typical techniques, i.e., ML_I, which is based on an individual machine learning algorithm, and ML_E, which integrates several machine learning algorithms.

ML_I first transforms the text (i.e., summary and description) in each previously resolved incident report to a feature vector by counting the term frequency so as to apply a machine learning algorithm. Then, ML_I labels each feature vector as the team that resolved the incident. Finally, ML_I uses Support Vector Machines (SVM) [17] to train a classifier for recommending assignments of new incident reports.

ML_E uses an ensemble learning technique to integrate several machine learning algorithms. Following ML_I, ML_E first uses each individual machine learning algorithm to train a classifier. Then, it uses Stacked Generalization (SG) [18] to combine the results of these classifiers for recommending assignments of new incident reports. In particular, based on the results in the existing work [10], we selected four machine learning algorithms that achieve the best effectiveness to combine, including SVM [17], Random Forest [19], Decision Tree [20], and K-Nearest Neighbors (KNN) [21].

2) Deep-Learning based Technique: Deep-learning based bug triage also regards the problem as a supervised classification problem. Different from machine-learning based techniques, DL first converts each word in a previously resolved incident report (i.e., summary and description) to a vector representation using Word2Vec [22]. In this way, the semantic information of an incident report can be considered. Then, DL adopts Convolutional Neural Network (CNN) [23] to train a classifier for recommending assignments of new incident reports.

3) Topic-Model based Technique: TM recommends teams based on their expertise relevant to the topic of an incident report. It first uses Latent Dirichlet allocation (LDA) [24] to cluster previously resolved incident reports into topics based on the summary and description of incident reports. Then, it extracts the activities (including resolving and assigning) of each team based on these incident reports to construct the association between topics and teams. For a new incident report, TM first identifies its topics, and then produces a ranking list of recommended teams based on the constructed association.

4) Tossing-Graph based Technique: TG aims to construct a graph model to capture the tossing (also called reassigning) probability between teams from the tossing history via Markov chains. In particular, we construct the *goal oriented model*, since the existing study [3] demonstrated that this model performs the best. The goal oriented model encodes the relationship between intermediate teams and the resolving team. Same to the existing work [3], TG first uses Naive Bayesian [25] to train a classifier to produce a list of recommended teams. Then, it adjusts the list of recommended teams based on the constructed graph model.

5) Fuzzy-Set based Technique: FS aims to model the resolving association between teams and technical aspects via fuzzy sets [26]. It constructs the association based on the technical terms in previously resolved incident reports (i.e., summary and description) and the past resolving activities of teams via fuzzy sets. Based on the association, FS recommends the teams with the highest resolving expertise for new incident reports.

B. Study Design

1) Subjects: In this study, we selected 10 (out of 20) online service systems in Microsoft, including several widely-used Microsoft products (e.g., Office 365, Skype, and Visual Studio) as subjects, to evaluate the effectiveness of these typical bug-triage techniques for incident triage. The total number of teams for the 10 subjects is about 9K. In particular, we used the incident reports from the first half period as the training data to construct the model for incident triage, and used the remaining incident reports as the testing data to evaluate the effectiveness of each technique based on the model.

2) Tools and Implementations: As the implementations of the studied bug-triage techniques are unavailable, we reimplemented them strictly following the description of these techniques in the papers. The first two authors carefully checked

the understanding about the description of these techniques and the implementations with each other. In particular, for all the involved machine learning algorithms (including SVM, Random Forest, Decision Tree, KNN, and Naive Bayesian), we adopted the implementations provided by scikit-learn[6], which is a popular tool for data mining and data analysis in Python. Also, the implementation of LDA used in TM is provided by scikit-learn. For DL, we adopted the implementation of Word2Vec provided by gensim[7], and implemented CNN based on Apache MXNet[8], a scalable deep learning framework. In particular, we used the default parameters of these provided implementations. For the parameters to be set additionally, we set them based on a small dataset. More specifically, the kernel used in SVM is the linear kernel, the number of topics in LDA is set to be 60, and the word-embedding dimension is set to be 128. For DL, the CNN uses three sets of convolution kernels with different sizes (i.e., 3, 4, 5), each of which has 50 channels, and the used epoch is 20.

3) Metrics: To measure the effectiveness of these studied techniques, following the existing work [13], [27] we adopted the widely-used metric: *accuracy@n*, which means that the responsible team is identified in the top-n list of the returned results. Here we consider n = {1, 3, 5}.

Besides, we also measure the efficiency of these studied techniques in the incident-triage context. That is, for each studied technique, we recorded the time spent on the phase of training, which is to construct a model for incident triage, and the average time spent on the phase of recommending, which is to recommend the responsible team for a new incident report. For the ease of presentation, we call the former *training time* and the latter *recommendation (or testing) time*.

C. Results and Analysis

1) Effectiveness: Table IV shows the effectiveness of the studied bug-triage techniques for all incident reports in testing data. In this table, the last row presents the average accuracy of each studied technique on the 10 subjects, and the bold value for each metric in each row refers to the largest one among these techniques. From this table, we find that all the studied bug-triage techniques are able to assign incident reports to corresponding teams to a certain extent, e.g., the average accuracy@1 values of them range from 0.32 to 0.71. On average, the DL technique performs the best while the TM technique performs the worst for incident triage in terms of accuracy@1, accuracy@3, and accuracy@5. Moreover, for the most important metric accuracy@1, the DL technique performs the best for the largest number of subjects. That is, among these studied techniques, the DL technique is the most effective one for incident triage of online service systems. One possible reason is that the DL technique considers the semantic information of incident reports via word embedding, which is suitable in the incident-triage context. However, the accuracy of these studied techniques (including the DL technique) still has room to further improve for incident triage of online service systems. For example, the accuracy@1 value of the most effective technique for subject *P3* is only 0.42. In particular, as demonstrated in the existing studies also using industrial subjects [10], [11], the average accuracy@1 value of the ML_E technique achieves 0.71 on five industrial subjects (while its value is 0.58 in our context), and the average accuracy@5 value of the DL technique achieves 0.92 on four industrial subjects (while its value is 0.85 in our context). That further demonstrates the effectiveness of these bug-triage techniques is discounted in our incident-triage context, and thus the accuracy of incident triage should be further improved.

We further analyzed the effectiveness of the studied bug-triage techniques for the incident reports involving reassignment in the testing data. These incidents are incorrectly assigned at the first time, and thus are able to incur larger cost as presented in Section II-B2. Therefore, accurately assigning these incident reports are more important. Table V shows the effectiveness of the studied techniques on these incident reports. From this table, all the studied bug-triage techniques perform significantly worse than their effectiveness shown in Table IV on average. That indicates that the effectiveness of these studied techniques is largely discounted for assigning the incident reports that are incorrectly assigned at the first time. In particular, the accuracy@1 values of them range from 0.17 to 0.52 on average, which are relatively low, demonstrating that these studied techniques still need to be further largely improved for incident reports involving reassignment. Among these techniques, the DL technique still performs the best, which further shows that the DL technique is more suitable for the incident-triage context.

To sum up, in general, the studied bug-triage techniques (except the TM technique) perform relatively well on incident triage for online service systems, while they perform relatively poorly for the incident reports involving reassignment, indicating that there is still considerable room for improving the accuracy of incident triage, especially for the incident reports involving reassignment. Also, the DL technique performs the best among these techniques for incident triage.

2) Efficiency: We also investigated the efficiency of the studied bug-triage techniques, including the training time and the recommendation (testing) time. Table VI shows the efficiency results of the studied bug-triage techniques. From the second row of this table, we can see that much time is spent on the phase of training for all the studied bug-triage techniques, ranging from 91.93 seconds to 13,607.86 seconds. However, the time consumption is acceptable since the phase of training is offline, which cannot lead to the delay of incident triage. Considering the whole computation resources, the DL, TM, TG, and FS techniques seem to be better than the other two techniques in terms of the training efficiency. From the last row in this table, we can see that the time spent on recommending the responsible team for an incident report is little for all the studied bug-triage techniques, ranging from 0.0002 seconds to 0.8267 seconds.

Overall, the results demonstrate that all these techniques are

[6] http://scikit-learn.org/stable/.

[7] https://radimrehurek.com/gensim/.

[8] https://mxnet.incubator.apache.org/.

TABLE IV: Accuracy of the studied bug-triage techniques for all incident reports in testing data

Sub	Accuracy@1						Accuracy@3						Accuracy@5					
	ML_I	ML_E	DL	TM	TG	FS	ML_I	ML_E	DL	TM	TG	FS	ML_I	ML_E	DL	TM	TG	FS
P1	0.62	0.63	**0.70**	0.28	0.64	0.43	0.75	**0.79**	0.78	0.45	0.72	0.64	0.79	**0.81**	0.81	0.54	0.81	0.71
P2	0.65	0.65	**0.69**	0.50	0.68	0.63	0.83	0.81	**0.84**	0.71	0.81	0.78	0.85	0.84	**0.85**	0.78	0.85	0.83
P3	0.38	**0.42**	0.41	0.18	0.41	0.29	0.60	0.57	0.61	0.37	**0.62**	0.52	0.66	0.64	**0.72**	0.45	0.68	0.62
P4	0.67	0.45	0.72	0.16	**0.75**	0.64	0.85	0.87	0.88	0.47	**0.90**	0.84	0.91	0.90	0.91	0.55	**0.93**	0.90
P5	0.56	0.55	0.68	0.33	**0.68**	0.48	0.61	0.76	0.77	0.46	**0.77**	0.73	0.64	0.81	0.81	0.53	**0.81**	0.83
P6	0.52	0.61	**0.83**	0.45	0.78	0.58	0.62	0.83	**0.89**	0.66	0.85	0.78	0.71	0.85	**0.90**	0.69	0.88	0.83
P7	0.35	0.32	0.64	0.35	**0.67**	0.55	0.50	0.78	0.73	0.45	**0.80**	0.72	0.60	0.80	0.77	0.53	**0.83**	0.78
P8	0.66	0.69	**0.82**	0.26	0.75	0.48	0.79	0.85	0.87	0.41	**0.88**	0.65	0.85	0.87	0.89	0.46	**0.92**	0.72
P9	0.77	0.78	**0.81**	0.39	0.77	0.64	0.83	0.89	**0.92**	0.59	0.87	0.79	0.88	0.91	**0.94**	0.73	0.90	0.88
P10	0.53	0.70	**0.77**	0.29	0.72	0.42	0.77	0.84	**0.84**	0.47	0.77	0.64	0.88	0.86	**0.88**	0.54	0.83	0.73
Avg.	0.57	0.58	**0.71**	0.32	0.68	0.51	0.71	0.80	**0.81**	0.50	0.80	0.71	0.78	0.83	**0.85**	0.58	0.84	0.78

TABLE V: Accuracy of the studied bug-triage techniques for incident reports involving reassignment in testing data

Sub	Accuracy@1						Accuracy@3						Accuracy@5					
	ML_I	ML_E	DL	TM	TG	FS	ML_I	ML_E	DL	TM	TG	FS	ML_I	ML_E	DL	TM	TG	FS
P1	0.28	0.26	**0.28**	0.03	0.28	0.17	0.38	0.38	**0.38**	0.16	0.33	0.25	0.42	0.41	**0.85**	0.24	0.40	0.30
P2	0.65	0.64	**0.69**	0.50	0.67	0.63	0.83	0.81	**0.84**	0.71	0.71	0.78	0.85	0.83	**0.94**	0.78	0.82	0.83
P3	0.19	0.17	**0.26**	0.04	0.21	0.14	0.38	0.29	**0.44**	0.18	0.32	0.34	0.45	0.36	**0.57**	0.28	0.45	0.46
P4	0.75	0.42	0.75	0.17	**0.76**	0.65	**0.95**	0.92	0.91	0.35	0.86	0.90	**0.96**	0.94	0.80	0.42	0.93	0.94
P5	0.57	0.54	**0.64**	0.17	0.64	0.33	**0.85**	0.68	0.84	0.22	0.70	0.77	**0.87**	0.85	0.65	0.38	0.75	0.87
P6	0.56	0.46	**0.64**	0.13	0.57	0.30	**0.85**	0.82	0.84	0.38	0.67	0.58	**0.89**	0.86	0.84	0.40	0.86	0.75
P7	0.41	0.31	**0.47**	0.14	0.45	0.34	0.56	**0.65**	0.59	0.20	0.50	0.50	0.66	**0.70**	0.42	0.29	0.64	0.57
P8	0.21	0.24	**0.33**	0.18	0.26	0.24	0.44	0.36	**0.53**	0.20	0.47	0.32	0.62	0.50	**0.86**	0.22	0.64	0.40
P9	0.47	0.57	0.57	0.33	**0.61**	0.45	0.69	0.70	**0.82**	0.44	0.66	0.59	0.75	0.74	**0.89**	0.59	0.75	0.73
P10	0.48	0.38	0.53	0.03	**0.64**	0.09	0.65	**0.73**	0.71	0.23	0.65	0.42	0.71	**0.76**	0.57	0.30	0.73	0.51
Avg.	0.46	0.40	**0.52**	0.17	0.51	0.33	0.66	0.63	**0.69**	0.31	0.59	0.55	0.72	0.69	**0.74**	0.39	0.70	0.63

TABLE VI: Efficiency of the studied techniques (seconds)

Tech.	ML_I	ML_E	DL	TM	TG	FS
Train	8579.18	13,607.86	426.97	91.93	160.31	242.19
Test	0.0201	0.0804	0.0010	0.8267	0.0002	0.3367

practical for incident triage in terms of time efficiency.

IV. Implications for Incident Triage

A. Incident Triage and Bug Triage

Over the years, many bug-triage techniques have been proposed [2]–[8]. Considering the similarity between bug triage and incident triage, intuitively, these existing bug-triage techniques may be applied to incident triage. Indeed, the results presented in Section III-C demonstrate that the studied bug-triage techniques are able to assign incident reports to the responsible teams to a certain extent, but their effectiveness still needs to be further improved, especially for the incident reports involving reassignment. In general, incident triage for online service systems has the following different characteristics with the traditional bug triage:

- Traditional bug reports are written manually by submitters, while incident reports for online service systems tend to be created and submitted automatically by monitors, although there is also a small portion of incident reports that are manually written by engineers. More specifically, for the 20 online service systems used in Section II, around 9% incident reports are manually written while around 91% incident reports are automatically created and submitted by monitors on average. The reason is that an online service system heavily depends on a large number of monitors to guarantee its quality, i.e., timely check the runtime state and behavior of the online service system. Once a monitor detects an incident, it could automatically create and submit an incident report based on certain template.
- Traditional bugs are reported and treated individually in the bug-triage context, while many incidents tend to have correlations. i.e., time correlation (reported around the similar time) and location correlation (reported from similar locations). There are two major reasons: 1) A monitor checks the runtime state and behavior of an online service system at regular time intervals, and thus a series of incident reports caused by the same root cause could be created and submitted continuously by the monitor; 2) Due to the propagation of failures, a number of monitors are able to detect incidents caused by the same root cause, and then each of these monitors could create and submit a separate incident report. In these cases, the incident reports caused by the same root cause tend to have similar reporting time or be reported from close locations. Figure 4 shows examples of time correlation among incident reports assigned to three teams of an online service system in Microsoft. In the figure, the x-axis represents a period of time and the y-axis represents the number of incident reports

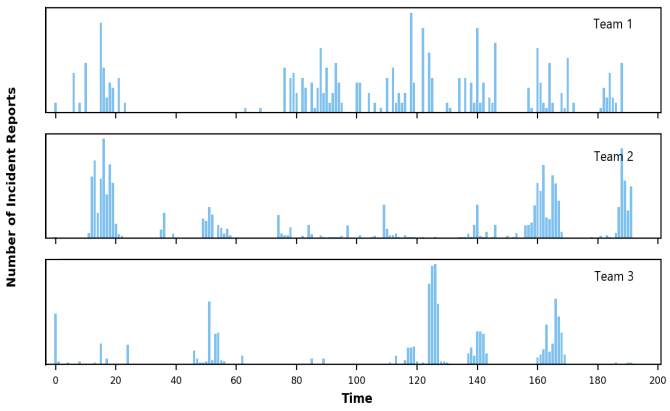

Fig. 4: Examples of time correlation of incident reports

that are assigned to the team at a time unit[9]. Higher bars mean larger numbers, indicating that more incidents are assigned to the team within the time unit (therefore stronger time correlation among the incident reports). In particular, many bars could be clustered together, indicating that the time correlation also exists within a larger time interval (i.e., several continuous time units). According to the number and distribution of incident reports, we observe that there is indeed time correlation among incident reports assigned to a team.

- For an online service system, incidents occur more frequently than bugs. This is because incidents not only can be caused by source code bugs, but also can be caused by various other factors, e.g., misconfigurations, hardware failures, service unavailable, etc. Therefore, in the incident-triage context, when there are incidents that are assigned incorrectly, it is more likely to lead to the backlog of incidents and delay the process of incident management due to reassignment. Therefore, it is desirable to have a more accurate incident triage method.
- Traditional bug triage tends to identify the responsible developer for each bug report, while incident triage for online service systems aims to find the responsible team for each incident report. The reason is that an online service system is often a large-scale system-of-systems, which requires a number of service teams to maintain. Within each team, the work is distributed internally, which could improve the efficiency for restoring the online service system.

B. Insights for Improving Incident Triage

Our investigation shows that incident management has become a critical task for online service systems, and incident triage plays an important role during the incident management process. Based on our empirical evaluation, we have obtained the following insights for improving incident triage:

- *Leveraging more incident data*: Since a large portion of incidents is detected by monitors for an online service system, there are actually a lot of monitoring data that can be used for incident triage, such as service-level logs, performance counters, and machine/process/service-level events. These monitoring data is relevant to the reported incidents to a certain extent, and thus leveraging them may further improve the accuracy of incident triage. For example, we can propose frequent-pattern-mining based techniques to analyze the service log information and get the suspicious execution patterns for incidents. The characteristics of these suspicious execution patterns may facilitate the recommendation of the responsible teams.
- *Leveraging the time and location correlations among incidents:* Since a series of incidents caused by the same root cause tend to be reported around similar time or from close locations, the characteristics of time and locations may facilitate incident triage. For example, instead of individually recommending the responsible team for each incident report, we can consider the characteristics of the incidents reported around the similar time as the current incident or the incidents reported around the nearby locations, which may further improve the accuracy of incident triage.
- *Considering the workload of teams:* Since incidents occur more frequently than traditional bugs, it is easier to lead to the backlog of incident reports for a team. That is, a team is more likely to be overloaded in the incident context. Therefore, it is necessary to consider the workload of teams when assigning incident reports. To make incident management more practical, cost-aware incident triage techniques are desired. In this direction, we may adapt the existing cost-aware bug-triage techniques [4], [28] for incidents, and then further improve these techniques by considering more domain-specific knowledge of incidents for online service systems.

V. Threats to Validity

The *internal* threat to validity mainly lies in the implementations of the studied bug-triage techniques. To reduce this threat, we reimplemented these techniques strictly following the description about them in the papers, and the first two authors carefully checked the implementations. For many algorithms (e.g., machine learning algorithms) used in these techniques, we adopted the implementations provided by mature tools such as scikit-learn, which have been presented in Section III-B2.

The *external* threats to validity mainly lie in the used subjects and the studied bug-triage techniques in the study. For the used subjects, we used 20/10 real-world online service systems in Microsoft in the two studies, respectively. To our best knowledge, this is the most large-scale study targeting at the triage problem (including bug triage) in industry. Even so, they may not represent other online service systems from other companies. In the future, we will perform a larger scale evaluation on the systems from different companies, as well as on open source systems. For the studied bug-triage techniques, we selected six techniques as the representatives following the criteria presented in Section III-A. These studied techniques

[9]Due to the policy of Microsoft, we cannot report the exact time unit.

may not represent other techniques. To reduce this threat, we first divided existing techniques into five categories based on their used technical aspects, and then selected one or two typical techniques from each category. That is, we try to consider the diversity of techniques. In the future, we will evaluate more techniques for incident triage.

The *construct* threats to validity mainly lie in the used parameters in the techniques and the adopted metrics. To reduce the threat from the parameters, we adopted the default parameters provided by the mature tools. For the parameters to be set additional, we set them based on a small dataset. In the future, we will explore the impact of various parameters. To reduce the threat from the used metrics to evaluate existing triage techniques, we considered the effectiveness and efficiency metrics. In particular, for the effectiveness metric, we adopted the *accuracy@n* metric, which has been widely used in the existing work [3], [13], [27]. For the metric to measure the reassignment cost, we use the time spent on reassignments, but this metric may be bias. In the future, we will consider more metrics to sufficiently measure them.

VI. Related Work

A. Bug Triage Techniques

Besides the six studied techniques in our work, there are many other bug-triage techniques. For example, Xia et al. [27] proposed a specialized topic model based bug triage technique, which considers the product and component information of bug reports to construct the mapping from term space to topic space for bug triage. Xuan et al. [5] proposed a machine-learning based bug triage technique by using data reduction techniques. More specifically, this technique first conducts the instance selection and feature selection for reducing training data before training a classifier using certain machine learning algorithm. Bhattacharya and Neamtiu [29] proposed to use a fine-grained incremental learning method and multi-feature tossing graphs (labeling the edges of tossing graphs using developer expertise and the nodes using developer activity on the basis of the original tossing graphs [3]) for bug triage. Besides the five categories, there are some bug-triage techniques based on other technical aspects. Hu et al. [6] proposed a bug-triage technique based on historical bug-fix information. More specifically, this technique models the relationship between developers and source code components, and also the relationship between source code components and the associated bugs. Badashian et al. [30] proposed a bug-triage technique based on the *Stack Overflow* platform. More specifically, it extracts the expertise of developers from the social software-development platforms for bug triage. Different from the above-mentioned related work, our work conducted an empirical study to evaluate the effectiveness of typical bug-triage techniques on incident triage for online service systems in industry.

B. Empirical Studies on Bug Triage

Apart from proposing various novel bug-triage techniques, there are also a number of empirical studies on the area of bug triage. Baysal et al. [31] conducted a study to revisit bug triage practices interviewed with Mozilla Core and Firefox developers. Goyal and Sardana [32] conducted an empirical study to compare machine-learning based bug-triage techniques and information-retrieval based bug-triage techniques based on four open-source projects. Dedík and Rossi [33] conducted an empirical study to explore the differences of machine-learning based bug-triage techniques between open-source projects and industrial projects. Lin et al. [14] conducted an empirical study to explore bug triage automation based on Chinese bug data. Karim et al. [34] conducted an empirical investigation for the single-objective and multi-objective evolutionary algorithms for bug triage. Jonsson et al. [10] conducted an empirical study to investigate the effectiveness of various machine learning algorithms based on industrial projects. Different from them, our work is the first work to explore incident triage rather than bug triage. In particular, we conducted an empirical study to better understand the incident-triage practice in industry, and an empirical evaluation of typical bug-triage techniques for incident triage of online service systems. Furthermore, we considered the bug-triage techniques from different technical aspects (e.g., machine-learning based techniques, topic-model based technique, and tossing-graph based technique).

C. Incident Management

Our work is also related to incident management since incident triage is an initial step of the process. Most of incident-management work focuses on the identification of *incident beacons* [35], [36], which are formed from a combination of system metrics with unusual values that produce symptoms. For example, Cohen et al. [36] proposed a Tree-Augmented-Network (TAN) approach to deducing a TAN model, and the model is able to predict system SLO (Service Level Objective) states based on a few system metrics. Here the approach identifies the metrics used by the model as service-issue beacons. Besides, some work aims to associate a new incident with a previous known incident [37], [38]. For example, Duan and Babu [37] proposed an active-learning based approach to improving the accuracy, which maximizes the benefits gained from new unknown instances to facilitate manual labeling efforts. In addition, Lou et al. [1], [39], [40] presented an experience report on applying software analytics in incident management of online service systems, including incident diagnosis and mitigation. Different from them, our work is the first one to explore incident triage for online service systems, which is an initial step of incident management.

VII. Conclusion

Incident triage is a critical task for maintaining a large-scale online service system. To better understand the incident-triage practice in industry, we perform an empirical study of incident triage on 20 real-world, large-scale online service systems in Microsoft. Through the empirical study, we find that incident reassignment occurs frequently, which incurs unnecessary cost, especially for the incident reports with high severity. We also evaluate six typical automated bug-triage

techniques for incident triage and find that the recent deep-learning based technique is the most effective one. However, the effectiveness of the existing bug-triage techniques still needs to be further improved for incident triage, especially for the incident reports involving reassignment. We further discuss the possible improvements in the paper.

To our best knowledge, we are the first to investigate incident triage in industrial practice. We believe that our work is useful for practitioners and researchers to improve the current incident-triage practice for online service systems.

VIII. Acknowledgement

We would like to thank the Microsoft groups that develop the incident management system, who helped us learn the incident management process and reviewed our work. This work is supported by NSFC Grant Nos. 61828201 and 61872008.

References

[1] J.-G. Lou, Q. Lin, R. Ding, Q. Fu, D. Zhang, and T. Xie, "Software analytics for incident management of online services: An experience report," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, 2013, pp. 475–485.

[2] D. Cubranic and G. C. Murphy, "Automatic bug triage using text categorization," in *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, 2004, pp. 92–97.

[3] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 111–120.

[4] J. Park, M. Lee, J. Kim, S. Hwang, and S. Kim, "Costriage: A cost-aware triage algorithm for bug reporting systems," in *Proceedings of the National Conference on Artificial Intelligence*, 2011, p. 139.

[5] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, and X. Wu, "Towards effective bug triage with software data reduction techniques," *IEEE transactions on knowledge and data engineering*, vol. 27, no. 1, pp. 264–280, 2015.

[6] H. Hu, H. Zhang, J. Xuan, and W. Sun, "Effective bug triage based on historical bug-fix information," in *IEEE 25th International Symposium on Software Reliability Engineering*, 2014, pp. 122–132.

[7] S. Wang, W. Zhang, and Q. Wang, "Fixercache: Unsupervised caching active developers for diverse bug triage," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014, p. 25.

[8] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation," in *10th IEEE Working Conference on Mining Software Repositories*, 2013, pp. 2–11.

[9] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *28th International Conference on Software Engineering*, 2006, pp. 361–370.

[10] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts," *Empirical Software Engineering*, vol. 21, no. 4, pp. 1533–1578, 2016.

[11] S.-R. Lee, M.-J. Heo, C.-G. Lee, M. Kim, and G. Jeong, "Applying deep learning based automatic bug triager to industrial projects," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 926–931.

[12] H. Naguib, N. Narayan, B. Brügge, and D. Helal, "Bug report assignee recommendation using activity profiles," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013, pp. 22–30.

[13] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Fuzzy set and cache-based approach for bug triaging," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 365–375.

[14] Z. Lin, F. Shu, Y. Yang, C. Hu, and Q. Wang, "An empirical study on bug assignment automation using chinese bug data," in *Proceedings of the 3rd international symposium on Empirical software engineering and measurement*, 2009, pp. 451–455.

[15] G. Bortis and A. v. d. Hoek, "Porchlight: A tag-based approach to bug triaging," in *Proceedings of the 2013 International Conference on Software Engineering*, 2013, pp. 342–351.

[16] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 3, p. 10, 2011.

[17] S. R. Gunn *et al.*, "Support vector machines for classification and regression," *ISIS technical report*, vol. 14, no. 1, pp. 5–16, 1998.

[18] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[19] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[20] R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[21] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine learning*, vol. 6, no. 1, pp. 37–66, 1991.

[22] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[23] C. dos Santos and M. Gatti, "Deep convolutional neural networks for sentiment analysis of short texts," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, 2014, pp. 69–78.

[24] M. Steyvers and T. Griffiths, "Probabilistic topic models," *Handbook of latent semantic analysis*, vol. 427, no. 7, pp. 424–440, 2007.

[25] E. Alpaydin, *Introduction to machine learning*. MIT press, 2009.

[26] G. Klir and B. Yuan, *Fuzzy sets and fuzzy logic*. Prentice hall New Jersey, 1995, vol. 4.

[27] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, "Improving automated bug triaging with specialized topic model," *IEEE Transactions on Software Engineering*, vol. 43, no. 3, pp. 272–297, 2017.

[28] M. Alenezi, K. Magel, and S. Banitaan, "Efficient bug triaging using text mining," *Journal of Software*, vol. 8, no. 9, pp. 2185–2190, 2013.

[29] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in *IEEE International Conference on Software Maintenance*, 2010, pp. 1–10.

[30] A. S. Badashian, A. Hindle, and E. Stroulia, "Crowdsourced bug triaging: Leveraging q&a platforms for bug assignment," in *International Conference on Fundamental Approaches to Software Engineering*, 2016, pp. 231–248.

[31] O. Baysal, R. Holmes, and M. W. Godfrey, "Revisiting bug triage and resolution practices," in *Proceedings of the First International Workshop on User Evaluation for Software Engineering Researchers*, 2012, pp. 29–30.

[32] A. Goyal and N. Sardana, "Machine learning or information retrieval techniques for bug triaging: Which is better?" *e-Informatica Software Engineering Journal*, vol. 11, no. 1, 2017.

[33] V. Dedík and B. Rossi, "Automated bug triaging in an industrial context," in *Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on*, 2016, pp. 363–367.

[34] M. R. Karim, G. Ruhe, M. M. Rahman, V. Garousi, and T. Zimmermann, "An empirical investigation of single-objective and multiobjective evolutionary algorithms for developer's assignment to bugs," *Journal of Software: Evolution and Process*, vol. 28, no. 12, pp. 1025–1060, 2016.

[35] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: automated classification of performance crises," in *Proceedings of the 5th European conference on Computer systems*, 2010, pp. 111–124.

[36] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons, "Correlating instrumentation data to system states: A building block for automated diagnosis and control." in *OSDI*, vol. 4, 2004, pp. 16–16.

[37] S. Duan and S. Babu, "Guided problem diagnosis through active learning," in *International Conference on Autonomic Computing*, 2008, pp. 45–54.

[38] M. H. Lim, J. G. Lou, H. Zhang, F. Qiang, A. Teoh, Q. Lin, J. Ding, and D. Zhang, "Identifying recurrent and unknown performance issues," in *IEEE International Conference on Data Mining*, 2015.

[39] J.-G. Lou, Q. Lin, R. Ding, Q. Fu, D. Zhang, and T. Xie, "Experience report on applying software analytics in incident management of online service," *Automated Software Engineering*, vol. 24, no. 4, pp. 905–941, 2017.

[40] Q. Lin, J.-G. Lou, H. Zhang, and D. Zhang, "How to tame your online services," in *Perspectives on Data Science for Software Engineering*. Elsevier, 2016, pp. 63–65.