A Study on the Use of Runtime Files in Handling Crash Reports in a Large Telecom Company

Komal Panchal
ECE, Concordia University
Montreal, Canada
komal.panchal@concordia.ca

Fatima Ait-Mahammed
ECE, Concordia University
Montreal, Canada
fatima.aitmahammed@mail.concordia.ca

Abdelwahab Hamou-Lhadj
ECE, Concordia University
Montreal, Canada
wahab.hamou-lhadj@concordia.ca

Zhongwen Zhu
Ericsson
Montreal, Canada
zhongwen.zhu@ericsson.com

Salman Memon
Ericsson
Montreal, Canada
salman.memon@ericsson.com

Alka Isac
Ericsson
Montreal, Canada
alka.isac@ericsson.com

Pragash Krishnamoorthy
Ericsson
Ottawa, Canada
pragash.krishnamoorthy@ericsson.com

***Abstract*—To minimize service downtime and ensure high system availability, Telecom companies must react quickly to failures and crashes of network services and applications. In this study, the focus is on the crash reporting process that normally goes through different levels of service support in a telecom company. To speed up this this repairing or recovering process, service support engineers and product developers rely on the analysis of runtime files (logs, traces, performance metrics, etc.) that are attached to crash reports submitted when an incident occurs. However, there is no clear understanding how these files are used and what their impact on the crash fixing time is. In this paper, we conduct an empirical study at Ericsson to study the use of runtime files in the crash resolution process. We tackle various research questions that revolve around the proportion of runtime files in a selected set of crash reports, the relationship between the severity of crashes and the type of files they contain, and the impact of different file types on the time to fix the crashes. We also study the prediction of the attachment of runtime files to crash reports during the creation of the reports. Our ultimate goal is to figure out how to collect enough information in the crash report system for support engineers and product developers to resolve the failures occurring in telecom network quickly and efficiently.**

***Keywords*—Network services and applications, Telecom systems, crash reports, empirical study in industry, quality of service.**

I. Introduction

The handling of system crashes efficiently is a critical activity for ensuring high quality of service and system availability. This is particularly important for large Telecom companies such as Ericsson where system operations tasks are inherently complex because of the large client base the company serves. The company needs to react quickly when crashes are reported. Failing to do so may lead to increased operational costs.

In a typical larger telecom company, the support service is provided through a multi-level support organization with a three-level support as the common practice. The handling of system crashes is triggered when a crash occurs, at which point a Level 1 support operator creates a crash report (CR) that contains information about the incident such as a description, the affected system components, the severity of the crash, and runtime data that are generated during the crash. The CR is saved in a CR database. The Level 1 support operator sends a notification to a Level 2 support engineer who retrieves and reviews the CR and performs root cause analysis to identify the reasons behind the incident and potentially provide a solution. A Level 2 support engineer may realize that the CR does not contain sufficient information and sends a request back to Level 1 support to generate the missing data. This cycle is repeated several times until the support engineer is satisfied with the detailed description of the incident. If the solution requires modifications to the source code, the CR is forwarded to a Level 3 support line to be reviewed by a product designer. Similar to the previous situation, the software designer may engage in a cycle of interactions with the service support teams to get more detailed information about the incident that would help uncover the root cause of the problem and provide a fix.

These repeated cycles of interactions between the various support lines often delay the provision of fixes, which may cause inefficiencies and high costs. With the help of Telecom domain experts from Ericsson, it shows that key information that product designer and service support engineer often request consists of files that contain runtime data about the system. These files are attached to CRs and used to perform root cause analysis. There are different type of runtime files including logs, traces, and profiling metrics [1]. However, it is not always clear what types of files are needed and what the impact of including these files on the CR fixing time is.

There exist studies that aim to improve the CR handling process by examining CR attributes including runtime data. Bettenburg et al. [3] conducted a user study and found that developers consider execution traces to be one of the most useful information that developers request when fixing a crash. Sabor et al. [11] introduced an approach to predict the severity of CRs using a combination of execution traces and other CR features. Koopaei et al. [7] proposed a method based on Hidden Markov Models (HMMs) and execution traces to predict duplicate CRs. Existing studies, however, do not specifically examine the role of runtime files in the CR resolution process. In addition, most studies are evaluated using open source systems. In this paper, we conduct an empirical study at Ericsson to understand the role of runtime files in the CR handling process. More particularly, we aim to answer the following four research questions:

- RQ1: What is the distribution of various types of runtime files that are attached to CRs?
- RQ2: What is the relationship between the severity of CRs and the type of files they contain?
- RQ3: What is the impact of different file types on the CR fixing time?
- RQ4: Can we predict if a CR should have a file attached to it before the CR is relayed to other lines of support?

Answering RQ1, RQ2, and RQ3 help us understand the files that are attached and set up the priority to the type of files that are required for the crash report. The answer to RQ4 may lead to the development of a recommendation system that would suggest to the first support line the files they should include when submitting a CR. The goal would be to reduce the number of interactions between the distinct lines of support, which should result in a more optimized CR resolution process. To our knowledge, this is the first study that focuses solely on the importance on various runtime files in the processing of CRs of an industrial Telecom system. The outcome of this study should shed light on the role of runtime files in resolving crashes from an industrial perspective, contributing further to the advancement of the field of software maintenance, evolution, and operations.

II. Study Setup

A. *Procedure and Dataset*

Figure 1 shows the process of collecting the data and addressing the research questions RQ1, RQ2, and RQ3. RQ4, the prediction of runtime file attachments, is discussed in Section III. First, we extract data from the Ericsson CR database. We then mine the reports to extract the files attached to the CRs. The input of domain experts is needed throughout the process to help understand the data and interpret the results.

The dataset we used in this study consists of CRs that cover almost two years of development of an Ericsson product (The exact number of CRs is not provided for confidentiality reasons). These CRs went through different lines of support until they were fixed. With the help of Ericsson experts, we wrote queries to extract CR information such as the CR heading, description, severity, the submitter, as well as the attached files. For each file, we retrieved the file name, the date at which the file was attached to the CR, the user who submitted the file, and the file size. Note that one CR may contain more than one file attached to it. In our dataset, the total number of files is five times larger than the number of CRs. We identified five types of files:

- Node Dumps (NDs): A ND file is required whenever a system crash is reported in a CR. It contains the output of a set of commands that are executed on the node, providing with a snapshot of the state of the node during the crash. This file is important for understanding what went wrong. The content of a ND may, for example, help spot configurations issues by the first line of support, eliminating the need to relay the CR to other lines of support.
- Key Performance Indicators (KPIs): A KPI file provides information on the performance of the system through performance counters. For example, a KPI file may show the degradation of the network, which can help network operators identify the cause of the problem.
- Execution traces: They are used to find a causal relationship between the system artifacts. Examples includes traces of function calls, inter-process communication traces, distributed traces, etc. Tracing requires instrumentation of the system, which consists of insertion of probes in places of interest. Using a trace, an analyst can replay the execution of the system to understand and diagnose the problem.
- User-defined Logs: Complex telecom systems such as the ones developed at Ericsson are composed of many hardware and software platforms. It is common for developers to insert logging statements that would help them later diagnose challenging problems. A logging statement typically contains a timestamp, a process id, a verbosity level, a logging function, a log message, and variables. Developers go through log messages to debug the system and perform root cause analysis.
- Post-mortem Dumps (PMDs): They contain whatever data is available in memory during the crash, including the processes that were executing at the moment of the incident, the data exchanged between processes, etc. PMDs are rarely structured and may contain data related to multiple processes or even components.

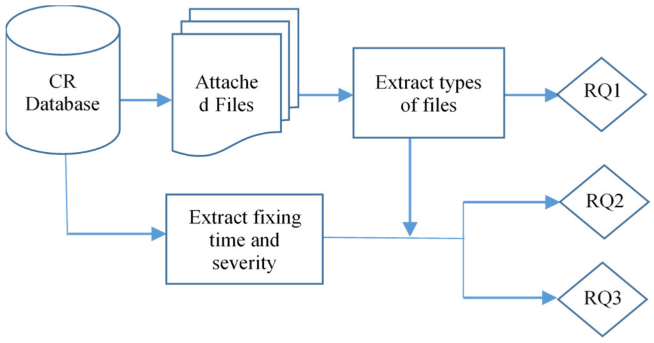

Fig. 1. Data collection and analysis process

B. *RQ1: What is the distribution of various types of runtime files that are attached to CRs?*

Objective: The answer to this question can help product design and support team understand the type of files that are used the most and should therefore be prioritized. This is because logs and traces are known to be large files [9] and generating and processing these files infer overhead and cost. Collecting all attached file types for each incident may turn out to be unproductive. One needs to develop parsers and analysis tools that are tailored towards a particular type of file.

Variables: For each file type $T_i \in \{ND, KPI, Log, Trace, PMD\}$, we use as variables the number of files of type T_i, the number of CRs that contain files of type T_i. We also need the total number of files and CRs to compute the ratios.

Method: To answer this research question, we use descriptive statistics [14]. More precisely, we measure the ratio of the number of files of type T_i to the total number of files and the ratio of the number of CRs that contain logs of file type T_i to the total number of CRs.

C. *RQ2. What is the relationship between the severity of CRs and the file types they contain?*

Objective: In this research question, we examine if there is a relationship between the severity of the CRs and the type of files they contain. The severity of a CR reflects the impact of the fault on the system functionality. The severity can be "A", "B", or "C" with "A" being the most severe CRs. We want to establish whether the severity of a CR impacts the type of files that are attached to this CR. We state the following null hypothesis:

- H_{01}: The type of files that is attached to a CR is not dependant on the CR severity?

Variables: We use as variables the number of CRs with file type T_i and the severity level of the CRs (A, B, C). Note that we exclude node dumps from this analysis because node dumps appear in all CRs.

Method: To answer the research question, we build the contingency table with two qualitative variables, the CR severity (A, B, C) and the CR file type. In each cell, we include the number of CRs that contain only files of that type. The last column contains the number of CRs that contain more than one file type. We use the Pearson's Chi-squared independence test [14] to accept or reject the null hypothesis H_{01}. The test is commonly used to examine the relationship between two qualitative variables, which are in our study, the CR severity and the CR attached file type. The result is considered statistically significant at alpha = 0.05. If $p\text{-value} < 0.05$ then we reject the null hypothesis H_{01} and conclude that the severity of the CRs and the types of files attached to the CRs are dependent.

D. *RQ3: What is the impact of file types on the time to fix the CRs?*

Objective: For this question, we analyze the impact of file types on the time it takes to solve the CR. The fixing time of a CR is measured in days. The answer to this question is useful to understand whether including a certain file type would improve the resolution process of CRs. For example, knowing that a CR that contains a PMD takes less time than a CR that does not have a PMD would encourage Ericsson designers to collect PMDs if they are available when a crash occurs. To answer this question, we run a statistical test for each file type $T_i \in \{ND, KPI, Log, Trace, PMD\}$. We state the following null hypothesis:

- H_{02}: There is no statistically significant difference between the fixing time of CRs with file type T_i and that of CRs that do not have files of type T_i.

Variables: We use as variables the fixing time of a CR, which is in days and the file type $T_i \in \{ND, KPI, Log, Trace, PMD\}$. Unlike the previous question, we do include node dumps in this question. In other words, we want also to know if the presence of a node dump file impacts the fixing time.

Method: We compute the non-parametric Mann-Whitney test [14] to compare the CR fixing time with respect to the file type T_i attached to the CR and analyze. We use the Mann-Whitney test because we cannot assume that the data follows a normal distribution. The result of the test is considered as statistically significant at alpha = 0.05. Therefore, if $p\text{-value} < 0.05$, we reject the null hypothesis H_{02} and conclude that the fixing time of CRs with file type T_i is significantly different from the fixing time of CRs without file type T_i.

III. Results

A. *RQ1: What is the proportion of each type of runtime files (i.e., traces, logs, profiling metrics, etc.) in our dataset?*

Figure 2 shows the percentage of each file type in our dataset. As we can see, NDs are the most files that are collected. This is expected since NDs contain information about nodes after a crash occurs. It is also common to have multiple ND files for one CR.

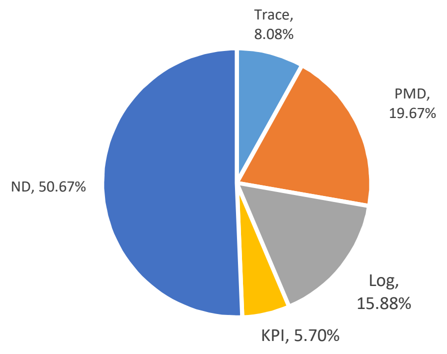

Fig. 2. Percentage of file types in our dataset

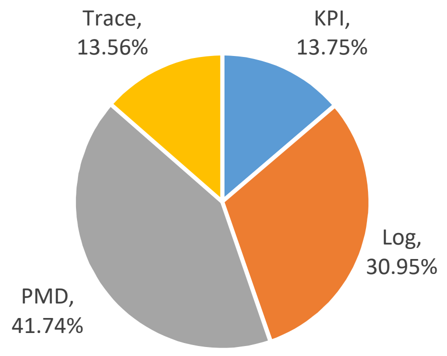

Fig. 3. Percentage of CRs regarding the enclosed file type.

PMDs and Logs occupy the second and third position with 19.67% and 15.88% of all the file types. Traces and KPI files are used the least. This could be due to the fact that these two types of files incur an additional overhead to generate then. For example, tracing requires an external instrumentation tool and settings, which may deter users from generating this type of file.

The same applies to KPI files that contain profiling metrics, and necessitate the use of profiling tools.

Figure 3 shows the percentage of CRs regarding the type of files they contain. Note that we excluded Node Dumps from this analysis since Node Dumps are collected for every crash. The figure shows that PMDs and Logs are the most used files, which clearly supports the idea that traces and KPIs receive less attention in problem diagnosis tasks.

B. *RQ2. What is the relationship between the severity of CRs and the file types they contain?*

Table 1 shows the percentage of each file type with respect to severity. Note that we show the percentages instead of the real values for confidentiality reasons. To compute the Chi-square test, we used the real value and not the percentages. We found that the p-value is < 0.00001. The result is significant at p-value < 0.05. Therefore, we reject the null hypothesis H_{01} and conclude that there is a relationship between the CR severity and the file types.

TABLE 1. CONTINGENCY TABLE FOR RQ2

Severity	**KPI**	**Log**	**Trace**	**PMD**	**More than one type**
A	3.57%	37.50%	7.14%	32.14%	19.64%
B	7.78%	14.39%	7.34%	41.56%	28.93%
C	19.64%	21.43%	5.36%	19.64%	33.93%

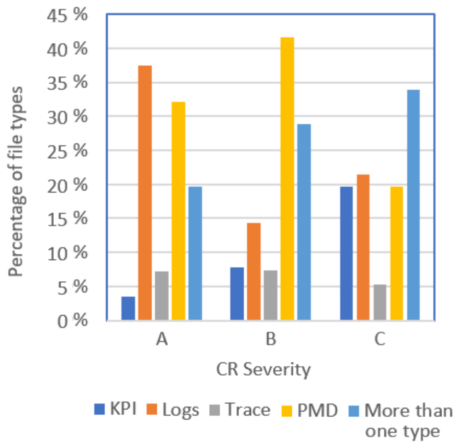

Fig. 4. Percentage of file types attached to CRs with respect to CR severity.

Figure 4 shows the distribution of the file types based on the CR severity. The data varies depending on the severity. The figure shows that the most severe CRs (CR with severity "A") have logs as the most important files. This may be explained by the fact that logs contain information introduced by developers and therefore can be very useful in debugging these systems. It should be noted that CRs with severity "A" are almost always sent to developers to provide fixes. CRs of severity "B" rely on PMDs and more than one type to help diagnose the problems. The least severe CRs contain a combination of all file types. We found that in all CRs, traces seem to receive the least attention.

C. *RQ3: What is the impact of file types on the time to fix the CRs?*

Table 2 shows the results of the statistical tests. As we can see from the table, there is a statistically significant difference between CRs that contain Node Dumps and those that do not in terms of the fixing time. The same applies to KPI, Log, and Trace file types. The corresponding p-values are less than 0.05. However, we did not find a statistical difference between CRs with PMDs and those that do not.

TABLE 2. RESULTS OF STATISTICAL ANALYSIS ON THE IMPACT OF FILE TYPES ON CR FIXING TIME

Log Type	**p-value**	**H_0**
Node Dumps	0.006273 < 0.05	Reject
KPI	0.000293 < 0.05	Reject
Log	8.761203e-05 < 0. 05	Reject
Trace	0.0102392 < 0.05	Reject
PMD	**0.6895507 > 0.05**	**Do not reject**

In other words, the fact that a CR contains a PMD does not necessarily result in a faster resolution. This may be due to the fact that PMDs may contain a large amount of unstructured data related to different parts of the system. Interpreting this data is usually a challenging task. Ironically, in RQ1, we found that PMDs are the second most collected runtime data after node dumps. Based on this finding, we suggest to review the relevance of PMDs for fixing CRs for improved efficiency and cost saving. The current practice of collecting PMDs whenever a crash occurs may not be productive.

IV. PREDICTION OF ATTACHMENT OF RUNTIME FILES

In this section, we answer RQ4: Can we predict if a CR should have a file attached to it before the CR is relayed to other lines of support? Predicting if a file needs to be attached during the creation of a CR is an important step towards reducing the time and effort it takes to solve the CR. We can implement a recommendation system that suggests to the CR reporter whether a file should be attached or not. This will reduce the number of interactions between the various lines of supports.

To answer this question, we use machine learning techniques. We build a training model that learns from past CRs that can later be used in the second phase to predict the inclusion of attached files (the inference phase). We discuss the feature extraction, training and testing phase, as well as the evaluation metrics in the next subsection.

A. *Feature Extraction*

With the help of Ericsson domain experts, we selected the following features of crash reports to use for classification.

- Submitter Priority: It is the severity of the defect and the CR's priority (i.e., A, B, or C).
- Priority Rate: This is an internal priority assigned to the CR.
- Observation Fault: This field represents the fault type.
- Faulty Product Design Responsible Office: This field represents the team within Ericsson that is responsible for

the defective product. This field changes as the CR is reassigned over the course of its life.

- Node Level Name: The node level product name.
- Node Level Product Number: The product number at the node level that is affected by the failure.
- Faulty product: This fields refers to the faulty product.
- Product affected system area: This field represents the functionality that is affected by the failure.

B. *Training and Testing*

We use three machine learning algorithms to predict whether a CR should have a file attached to it. These are Support Vector Machine (SVM), Random Forest (RF), and K-Nearest Neighbor (KNN). SVM is a classification algorithm that uses a hyperplane to differentiate different classes of examples in high-dimension space. KNN is a lazy learning algorithm based on instances. KNN returns the K most similar instances to a feature vector when given one. As a result, the method provides the K closest relevant instances based on the value of (K), which is a fixed variable that defines the number of returned neighbours. RF is a machine learning technique based on the decision tree algorithm. The model was created using a logic-based approach.

We use ten-fold cross-validation to test the model. This method randomly shuffles the dataset and divides it into 10 folds. The training is done using 90% of the data, which results in a model that is later tested with the remaining 10%. The approach is repeated 10 times by choosing different folds each time. The accuracy of the classification algorithm is the average accuracy obtained using the 10-fold cross-validation.

C. *Evaluation Metrics*

We used precision, recall, and F1-score to assess the performance of the algorithms. These metrics are used extensively in the literature, which are calculated as follows:

- Precision = TP/(TP + FP).
- Recall = TP/(TP + FN).
- F1_score = 2*Precision*Recall / (Precision + Recall)

Where TP (True Positives) is the number of CRs that are classified properly. FP (False Positives) represents the number of CR for which the algorithm wrongly predicted that an attachment is needed. FN (False Negatives) represents the number of CRs that have file attached files, but the algorithm missed.

D. *Results*

Table 3 shows the results of applying SVM, RF, and KNN. All algorithms perform relatively well. However, Random Forest yields best results with an F1-score of 84%. The precision and recall are 81% and 87.5%, respectively. Our comparative study of the different classifications techniques reveals that RF outperforms the other techniques when predicting the presence of files in CRs. These results are very promising and suggest that we can develop a recommendation system that predict if a file should be attached to a CR, which may reduce the number of interactions between the various lines of support.

TABLE 3. PREDICTION OF CR RUNTIME FILE ATTACHMENTS

Evaluation Metric	**SVM**	**RF**	**KNN**
Precision	75%	81%	76.8%
Recall	88.5%	87.5%	88%
F1-Score	81%	**<u>84%</u>**	82%

V. RELATED WORK

Maiga et al. [8] conducted an empirical study using Ericsson data to better understand how internal and external CRs are handled. Internal CRs refer to CR that are reported internally by testing teams, whereas external CRs refer to crashes that occur in the released products. The study focused on the percentage of internal vs. external CRs, time taken to fix each type of CR, and time to assign the CRs to developers.

Zimmermann et al. [13] showed that the main issue that developers face when addressing bug and crash reports is incomplete information. By surveying developers and users, the authors synthesized the information that makes a good report. The authors went on creating the CUEZILLA recommendation tool to assess the quality of submitted reports and make recommendations on how to enhance them.

Bhattacharya et al. [4] conducted an empirical study to learn more about the bug fixing procedure in Android apps and platforms and found that most Android app bug reports are of high quality, with lengthy textual descriptions, steps to reproduce errors, and explanations of expected output. A lengthy description usually indicates a high-quality bug report, which aids in the fast resolution of the bug.

An et al. [2] proposed an approach for mining information from bug and crash reports while highlighting the challenges of existing techniques. They looked at the problem of user identification. The authors also discussed the challenges of mapping crash reports to their corresponding bugs to understand the distribution of crash-related bugs in the user base. Rastkar et al. [10] examined if current conversation-based automatic summarizers can be used to summarize bug and crash reports. Karim [5] performed a research study to identify CR features that submitters often overlook. Then, the authors went on creating an automated key feature prediction model to recommend features to CR submitters based on historical reports.

Sabor et al. [11] introduced an approach to predict the severity of CRs using a combination of execution traces and CR categorical features (description, product, component, etc.). Koopaei et al. [6] introduced CrashAutomata to detect duplicate CRs using finite-state machines. The authors created a representative model using historical CRs and a combination of n-grams of varying lengths and automata to depict stack traces. In a follow-up study, Koopaei et al. [7] improved CrashAutomata by introducing the use of HMMs. The new approach resulted in an increase of the true positive rate by 10% over CrashAutomata. Sabor et al. [12] presented Durfex, a feature reduction technique for automatic prediction of duplicate reports using execution traces. The authors used a trace abstraction mechanism that replaces stack traces of function calls with traces of packages.

The analysis of crash/bug reports has been an active research topic for the last decade. Techniques that can reduce or predict the time it takes to fix crashes are needed to reduce the maintenance and operations overhead. Existing techniques focus on various aspects of bug and CRs. This paper complements these techniques by examining how runtime files including execution traces, logs, and profiling metrics, are used in an industrial setting to reduce the time and effort of solving crashes.

VI. Conclusion and Future Work

We conducted an empirical study on the runtime files attached to Ericsson CRs. We used a dataset of CRs covering two years of product development. We showed that there are five type of files that are used to diagnose system failures. We found that logs and PMDs are the two runtime file types that are collected the most when excluding Node Dumps. We also found that all file types have an impact on the CR fixing time except PMDs. This raises questions on the usefulness of PMDs in diagnosing crashes. Perhaps, the time and effort spent on collecting and processing PMDs should be invested in activities that have a direct impact on crash fixing time. Additionally, we found that traces and KPIs are used the least despite the extensive literature (e.g., [7][11]) that demonstrates their usefulness in crash diagnosis and repair tasks. We attributed this to the fact that these types of runtime data require external tools to generate and process their content. Further, we investigated whether we can predict if a CR should contain a file during the creation of the CR. This is important for reducing the time it takes to fix the CR. We used three machine learning algorithms – SVM, KNN and RF – and found that RF performs best with an F1-score of 84%. For future work, we intend to experiment with more CRs of different products. We will also work towards developing precise guidelines for the collection of runtime files that add value to the CR handling process. We will also set the ground for building a recommendation system for file attachment to optimize the diagnosis of crashes and failures.

VII. Acknowledgment

We would like to thank the domain experts at Ericsson for their assistance in collecting the data and examining the results. We are also grateful to MITACS, and Ericsson Global AI Accelerator (GAIA) for supporting this study.

References

[1] L. Alawneh, A. Hamou-Lhadj, "Execution Traces: A New Domain that Requires the Creation of a Standard Metamodel," Book Series on Communications in Computer and Information Science, Book on Advances in Software Engineering, Springer Berlin/ Heidelberg, pp. 253-263, 2009.

[2] L. An and F. Khomh, "Challenges and Issues of Mining Crash Reports," in Proc. of the IEEE 1st International Workshop on Software Analytics (SWAN), pp. 5-8, 2015.

[3] N. Bettenburg, S. Just, A. Schroter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?," in Proc. of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'16), pp. 308-318, 2008.

[4] P. Bhattacharya, L. Ulanova, I. Neamtiu and S. C. Koduru, "An Empirical Analysis of Bug Reports and Bug Fixing in Open Source Android Apps," in Proc. of the 17th European Conference on Software Maintenance and Reengineering (CSMR'13), pp. 133-143, 2013.

[5] M. R. Karim, "Key Features Recommendation to Improve Bug Reporting," in Proc. of the International Conference on Software and System Processes (ICSSP), pp. 1-4, 2019.

[6] N. E. Koopaei, A. Hamou-Lhadj, "CrashAutomata: An Approach for the Detection of Duplicate Crash Reports Based on Generalizable Automata," in Proc. of the 25th Annual International Conference on Computer Science and Software Engineering (CASCON'15), pp. 201–210, 2015.

[7] N. E. Koopaei, S. Islam, A. Hamou-Lhadj, and M. Hamdaqua, "An Effective Method for Detecting Duplicate Crash Reports Using Crash Traces and Hidden Markov Models," in Proc. of the 26th nnual International Conference on Computer Science and Software Engineering (CASCON'16), pp. 75–84, 2016.

[8] A. Maiga, A. Hamou-Lhadj, M. Nayrolles, K. K. Sabor and A. Larsson, "An empirical study on the handling of crash reports in a large software company: An experience report," in Proc. of the 31st IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 342-351, 2015.

[9] A. V. Miranskyy, A. Hamou-Lhadj, E. Cialini, A. Larsson, "Operational-Log Analysis for Big Data Systems: Challenges and Solutions," IEEE Software, vol. 33, no. 2, pp. 52-59, 2016.

[10] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," IEEE Transactions on Software Engineering, vol. 40, no. 4, pp. 366–380, 2014.

[11] K. K. Sabor, M. Hamdaqa, and A. Hamou-Lhadj, "Automatic prediction of the severity of bugs using stack traces and categorical features," Information and Software Technology, vol. 123, 2020.

[12] K. K. Sabor, A. Hamou-Lhadj, and A. Larsson, "DURFEX: A feature extraction technique for efficient detection of duplicate bug reports," in Proc. of the 2017 IEEE International Conference on Software Quality and Reliability (QRS'17), pp. 240–250, 2017.

[13] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter and C. Weiss, "What Makes a Good Bug Report?" IEE Transactions on Software Engineering, vol. 36, no. 5, pp. 618-643, 2010.

[14] Peter C. Bruce. Introductory Statistics and Analytics: A Resampling Perspective. Wiley; 1st edition, 2014.