ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 5, Issue 12, December 2016**

# A Novel Technique for Predicting Software System Crashes

Dr. Madhavi Karanam

Professor, Department of Computer Science and Engineering, Gokaraju Rangaraju Institute of Engineering and Technology, Hyderabad, India

**ABSTRACT**: Currently, most crash reporting systems sort crashes based on the number of their crash reports. These systems can easily be used to recognize the top crashes—but only in hindsight. To identify the top crashes, one must wait and see until enough crash reports have been submitted.This implies that users have to suffer many crashes before getting a fix, leading to possible data loss and frustration.One of the central hypotheses in work is that a small number of top crashes account for a majority of crash reports. To address this problem of current practice, the need to advocate a prediction-based approach that does not rely on hindsight to identify top crashes. With the current approach, it becomes feasible to identify top crashes during pre-release testing (i.e., alpha or beta testing), and also to react as soon as the first crash reports are received. Rather than waiting for a number of crashes to occur, developers can identify and address the most pressing problems without delay.This result is encouraging because it shows that the prediction-based approach holds promise in improving the current practice. Motivated by this improvement, thus, proposed our approach toward top-crash prediction.

**KEYWORDS**: Crash reports, alpha testing, prediction.

## I. INTRODUCTION

In recent days there are many software systems emerging, most of them which instantly reports failure back to the individual with the intention that developer gets a glance over the most encountering troubles from the software system. However, this may consumes certain instance of time to access those failures which are frequently reported and therefore predicting these "top crashes" which eventually enhance the quality of the software product. The existing work which reveals that most of the top crashes can be analysed and recognized only after the enough crash reports are occurred and the concept behind existing work is “Wait and see”. Therefore this leads the individual to suffer many crashes before the solution to fix that bug and along with loss of data.

Each crash report includes a crash point, that is, the program location where the crash occurred. Crashes that have the same crash point are considered to be the same [34]. In addition, a crash report includes other information relevant to crash occurrence, such as user comments, hardware and software configuration, as well as thread stack traces—stacks of methods that were active at the moment of the crash. The number of crash reports thus submitted can be large: Everyday, Firefox users submit thousands of crash reports; the number spikes considerably right after a new software release. The crashes summarized by these crash reports, however, are not equally frequent.it can well be that only a small number of crashes account for the vast majority of crash reports calledas top crashes

## II. RELATED WORK

Literature survey is mainly carried out in order to analyse the background of the existing work which helps to find out flaws in the existing work& guides on which unsolved problems can easily worked out. So, the following topics not only illustrate the background of the project but also uncover the problems and flaws which motivated to propose solutions. A variety of research has been done on learning of collective behaviour. Following sections explores different references that discuss about several topics related to collective behaviour.In this section, let’s have a brief review research areas related to our proposed approach.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 5, Issue 12, December 2016**

### *A. CRASH REPORTS ANALYSES*

Automatic crash reporting facilities have long been integratedinto commercial software (e.g., the Dr. Watson system used by Microsoft [15]). Compared to manually written bug reports [7], crash reports always contain accurate information and require little human effort to submit. However, the analysis of crash reports can be tedious and time consuming if done manually because the raw data they contain (e.g., the heap state and stack frames) are not amenable to human examination. For this reason, researchers have proposed many techniques to automate crash report analysis.

### *B. BUG REPORT*

Bug reports are vital for any software development. They allow users to provide crucial information to developers of the problems encountered while using software. Bug reports typically contain a detailed description of a failure and occasionally hint at the location of fault in the code (in form of patches or stack traces). However, bug reports vary in their quality of content as they often provide inadequate or incorrect information in software development. However, these reports widely differ in their quality. Most developers consider steps to reproduce, stack traces, and test cases as helpful, which are at the same time most difficult to provide for users. Such insight is helpful to design new bug tracking tools that guide users at collecting and providing more helpful information.

### *C.CUEZILLA*

A prototype called CUEZILLA is such a tool and measures the quality of new bug reports; it also recommends which elements should be added to improve the quality. CUEZILLA was able to predict the quality of 31–48% of bug reports accurately. There expected several factors to impact the quality of bug reports such as the length of descriptions, formatting, and presence of stack traces and attachments. Hence, to enable swift fixing of bugs, this mismatch should be bridged with the help of a prototype tool called CUEZILLA, which gauges the quality of bug reports and suggests to reporters what should be added to make a bug report better and also provides incentives to reporters.

The work most closely related to failure clustering, an approach introduced by Podgurski et al. [19]. There the technique applies feature selection, clustering, and multivariate visualization to group failures with similar causes.

### *D. FAILURE CLUSTERING*

A transmitted failure report includes information characterizing the state of the software at the time the failure was detected, which is intend to serve the purpose of being automated support for classifying reported software failures in order to facilitate prioritizing them and diagnosing their causes.

The presence of multiple faults in a program can inhibit the ability of fault-localization techniques to locate faults. This problem occurs for two reasons: when a program fails and the number of faults is unknown and certain faults may mask or obfuscate other faults.

### *E. SEQUENTIAL DEBUGGING*

A developer can inspect a single failed test case to attempt to find its cause using an existing debugging technique, after a fault is found and fixed, the program must be retested to determine whether previously failing test cases now pass. If failures remain, the debugging process is repeated. This is called one-fault-at-a-time mode of debugging and retesting sequential debugging.

### *F. PARALLEL DEBUGGING*

A technique that enables more effective debugging in the presence of multiple faults and a methodology that enables multiple developers to simultaneously debug multiple faults and this technique parallel debugging that is an alternative to sequential debugging. The main benefit for parallel debugging is that it can result in decreased time to a failure-free program

### *G. FAULT-FOCUSING CLUSTER*

The current technique automatically partitions the set of failing test cases into clusters that target different faults, called fault-focusing clusters, using behaviour models and fault-localization information created from execution data.

Each fault-focusing cluster is then combined with the passing test cases to get a specialized test suite that targets a single fault. Consequently, specialized test suites based on fault-focusing clusters can be assigned to developers who

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 5, Issue 12, December 2016**

can then debug multiple faults in parallel. The resulting specialized test suites provide a prediction of the number of current, active faults in the program. Another benefit is that the fault localization effort within each cluster is more efficient than without clustering. A third benefit is that the number of clusters is an early estimate of the number of existing active faults. A final benefit is that our technique automates a debugging processthat is already naturally occurs.

### *H. TRACE PROXIMITY*

Recent software systems usually feature an automated failure reporting system, with which a huge number of failing traces are collected every day. In order to prioritize fault diagnosis, failing traces due to the same fault are expected to be grouped together. By hypothesizing that similar failing traces imply the same fault, cluster failing traces based on the literal trace similarity, which is called trace proximity. However, since a fault can be triggered in many ways, failing traces due to the same fault can be quite different.

Liblit and Aiken introduced a technique that supports automatic reconstruction of completeexecution paths based on partial execution information such as backtracks and execution profiles. Their technique analyzes control flow graphs and derives a set of plausible paths that agree with available information. Manevich et al. [2] proposed the PSE technique, which improved on Liblit's work by incorporating a data flow analysis to reduce infeasible execution paths. These techniques are especially important for debugging predicted top crashes because, at the time of prediction, there are only a small number of crash reports available to developers.

### *I. POST-MORTEM SYMBOLIC EVALUATION*

PSE(Post-mortem Symbolic Evaluation), is a static analysis algorithm that can be used by programmers to diagnose software failures. The algorithm requires minimal information about a failure, namely its kind (e.g. NULL dereference), and its location in the program's source code. It produces a set of execution traces along which the program can be driven to the given failure. PSE tracks the flow of a single value of interest from the point in the program where the failure occurred back to the points in the program where the value may have originated.

### *J. FREQUENCY PREDICTION*

Frequency prediction refers to the problem of estimatinghow frequently a program entity (e.g., branch, path, function, or declarative rules) will be exercised in program execution. This is applicable in many domains such as program optimization [18] and reliability assessment [4]. Research in this area can be divided into profile-based [17], program-based [5], and evidence-based [12] approaches.

### *K. PROFILE-BASED APPROACH*

The profile-based approach samples a few actual executionsof the program and generalizes them to predict other executions. It has been successfully applied to predict branch frequency [17]. The profile-based approach requires a workload generator that is capable of simulating the real worldusage of the program. Such a workload generator can be difficult to design.

### *L. PROGRAM-BASED APPROACH*

The program-based approach relies on source code analysis and does not require the generation of a realistic workload. Ball and Larus [5] were among the first to use this approach for branch frequency prediction purposes. Specifically, used a loop analysis to predict branches that control the iteration of loops and some simple heuristic rules (e.g., comparing a pointer with NULL usually fails) to predict nonloop branches. One limitation of the program-based approach is that prediction rules have to be derived from experience. However, currently no such rules have been observed for crash stacks.

### *M. EVIDENCE-BASED APPROACH*

In [1], Calder et al. proposed the evidence-based approach, which addresses the limitation of the program based approach while retaining its benefit. The main idea is to first learn prediction rules from a corpus of programs, and then use the learned rules to predict the frequency for new programs. They applied this approach to predict the branch frequency. Recently, Buse and Weimer [11] used a similar approach to predict path frequency. The current approach shares the same principle as the evidence-based approach. However, there are some facts which considered for

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 12, December 2016

predicting the crash frequency, which is different from branch or path frequency. In addition to it here introduced the use of social network metrics in our predictions.

## III. PROPOSED WORK

The description contains several topics which are worth to discuss and also highlight some of their limitation that encourage going on finding solution as well as highlights some of their advantages for which reason these topics and their features are used in this paper.
Many of today's software systems include automated problem reporting: As soon as a problem occurs, the system reports the problem details back to the vendor, who can then leverage these details to fix the problem. As an example of automated problem reporting, consider the well-known Firefox Internet browser.

1. This paper presents a novel technique to predict whether a crash will be frequent (a "top crash") or not.
2. Then evaluate the current carried out approach on the crash report repositories of Thunderbird and Mozilla, demonstrating that it scales to real-life software.
3. Moreover the need to show that approach is efficient, as it requires only a small training set from the previous release. This implies that it can be applied at an early stage of development, e.g., during alpha or beta testing.
4. Moreover to show that approach is effective, as it predict stop crashes with high accuracy. This means that effort on addressing the predicted problems is well spent.
5. Here the need to discuss and investigate under which circumstances the current approach works best; in particular, the purpose investigate which features of crash reports are most sensitive for successful prediction.
6. Overall, the need to thus allow for quick resolution of the most pressing bugs, increasing software stability, along with user satisfaction.

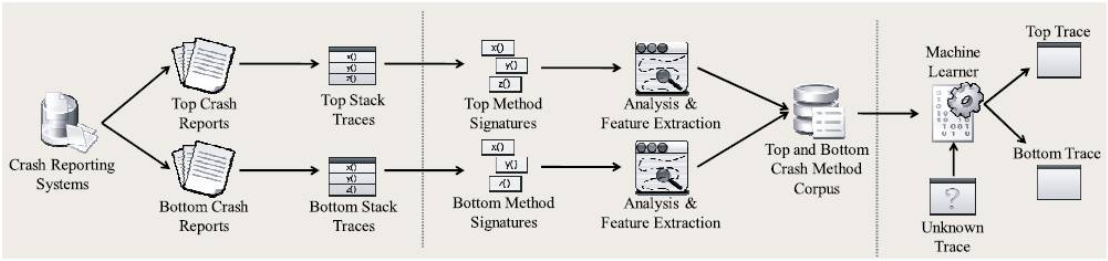


**Fig 1 Overview description of Approach**

The approach consists of three steps: extracting traces from top and bottom crash reports, creating training data from the traces, and predicting unknown crashes.

(a) Extracting crash traces: -The first step classifies top and bottom crashes and extracts stack traces from their reports.
(b) Creating corpus: - the second step extracts methods from the stack traces and characterizes these methods using feature data, which are extracted from source code repositories. Feature values are then accumulated per trace. These are used for training a machine learner.
(c) Prediction: - In the prediction step, the machine learner takes an unknown crash stack trace and classifies it as a top or bottom trace.

The classifier is first trained on information found in historical changes, and once trained, can be used to classify a new impending change as being either buggy or clean. If the change is predicted to be buggy, a software engineer could perform a range of actions to find the latent bug, including writing more unit tests, performing a code inspection, or

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 12, December 2016

examining similar changes made elsewhere in the project. A bug prediction service must also provide precise predictions, A possible approach (from the machine learning literature) for handling large feature sets is to perform a feature selection process to identify that subset of features providing the best classification results. A reduced feature set improves the scalability of the classifier, and can often produce substantial improvements in accuracy, precision, and recall.

To classify software changes using machine learning algorithms, a classification model must be trained using features of buggy and clean changes. In order to find bug-introducing changes, bug fixes must first be identified by mining change log messages. The two approaches ate used in searching for keywords in log messages suchas “Fixed”, “Bug”, or other keywords likely to appear in a bug fix, and searching for references to bug reports like “#42233”. This allows us to identify whether an entire code change transaction contains a bug fix.

### A. *IMPROVING THE ACCURACY OF PREDICTION*

In [1], they have not considered environmental factors like the execution environment, OS virtual memory snap, network bandwidth available etc. But most of the crashes occur due to this kind of problems. So for training of the machine learning model consider following additional features

**Table 1 Additional Features**

| Features | Description |
|---|---|
| Virtual memory available | Availability of virtual memory in terms of snap |
| Network bandwidth | The current bandwidth used at the time of crash |
| No of instances | Number of instance of application running |
| CPU usage | CPU usage of the application |

For all crashes occurred during the alpha and beta testing these parameters are also gathered and the machine model is trained. Due the consideration of environmental factors, the accuracy of prediction increases.

### *B.AUTO LABELLING OF CRASHES*

The software can be split into different functional groups. Each functional group can consist of one or more modules. For each defect identified in the alpha and beta testing process, the functional area in which the crash belongs is found by developer.

Based on this dataset of crash features with the functional area identified, a feed forward neural network is trained. The input is the features and the output is the functional area of crash. Any defects occurring during the testing and the live usage scenarios, these defect features are extracted and passed to the neural network to identify the functional area in which the crash belongs. The advantage of this auto labelling is enormous. It allows managers to plan their resources on the functional area in which the top crashes occur. Indirectly defects grouped in same functional area may have same source. So it helps the developer to categorize the related crashes together and analyze it.

This results shows that the dataset contain ‘how many correctly classified instances’ and ‘how many incorrectly classified instances’. It also shows that ‘what are the attributes’ that it selected for finding the defects. Moreover this window displays the other kappa static, mean absolute error, root mean squared error, relative absolute error and root relative squared error.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 12, December 2016

**Performance Graph**

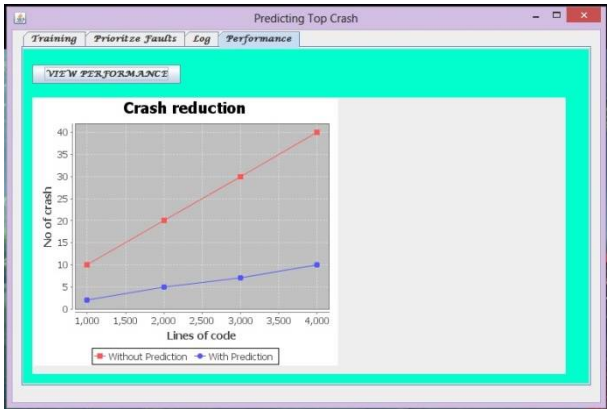


**Fig.2 Performance Comparison graph**

Fig 2 shows the comparison graph in between with the prediction of crash and without prediction results. This crash reduction graph is drawn based on lines of codes in bluecolour on x- axisand the no of crashes occurred represented in red colour on the y- axis.This graph is generated between no of crashes and lines of code.

## IV. CONCLUSION AND FUTURE WORK

There is no reason why developers should have to wait for crash reports to pile up before knowing where to start. By learning from past crash reports, thus, can automatically and effectively predict whether a new crash report is the first of many similar ones to come (and hence deserves immediate attention) or whether it is likely to be an isolated event.
By automatically classifying incoming crash reports, developer scan quickly fix the most pressing problems, which increases software stability and improving the user's experience. By applying the current approach on the crash databases of Firefox and Thunderbird, traced out a prediction accuracy of 75%-90% despite having learned from only a small number of crash reports. This approach is fully automated and easily applicable for any software system where crash data are collected and aggregated in a central database.
The future scope of this work is on Non-crashing problems i.e. the need to investigate which specific runtime features can be used to predict problem frequency. Software failures may not manifest themselves as a crash—the end result may be invalid without the operating or runtime system detecting a problem and, in particular, without a stack trace characterizing the problem.

## REFERENCES

[1] Which crash should fix first? Dongsunkim IEEE transactions on software Engineering MAY 2011
[2] E. Alpaydin, Introduction to Machine Learning. MIT Press, 2004.
[3] BreakPad, http://code.google.com/p/google-breakpad/, 2009.
[4] A. Avritzer, J.P. Ros, and E.J. Weyuker, "Reliability Testing ofRule-Based Systems," IEEE Software, vol. 13, no. 5, pp. 76-82, Sept.1996.
[5] T. Ball and J.R. Larus, "Branch Prediction for Free," Proc. ACMSIGPLAN Conf.Programming Language Design and Implementation,pp. 300-313, 1993.
[6] CrashReporter(MacOSX),http://developer.apple.com/technotes/tn2004/tn2123.html, 2009.
[7] N. Bettenburg, S. Just, A. Schro¨ ter, C. Weiss, R. Premraj, and T.Zimmermann, "What Makes a Good Bug Report?" Proc. 16th ACMSIGSOFT Int'l Symp.Foundations of Software Eng., pp. 308-318,2008.
[8] J. Ekanayake, J. Tappolet, H.C. Gall, and A. Bernstein, "TrackingConcept Drift of Software Projects Using Defect PredictionQuality," Proc. Sixth IEEE Int'l Working Conf. Mining SoftwareRepositories, pp. 51-60, 2009.N.E. Fenton and M. Neil, "Software Metrics: Roadmap," Proc.Conf. The Futureof Software Eng., pp. 357-370, 2000.
[9] GNU Binutils, http://www.gnu.org/software/binutils/, 2009.
[10] R.P.L. Buse and W. Weimer, "The Road Not Taken: EstimatingPath Execution Frequency Statically," Proc. IEEE 31st Int'l Conf.Software Eng., pp. 144-154, 2009.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 5, Issue 12, December 2016**

[11] B. Calder, D. Grunwald, M. Jones, D. Lindsay, J. Martin, M.Mozer, andB. Zorn, "Evidence-Based Static Branch PredictionUsing Machine Learning," ACM .Software Eng.AndMethodology, vol. 19, no. 1, pp. 188-222, 1997.
[12] P. Godefroid et al., "Automated White box Fuzz Testing," Proc.Network Distributed Security Symp., 2008.
[13] M. Hamill and K. Goseva-Popstojanova, "Common Trends inSoftware Fault and Failure Data," IEEE Trans. Software Eng.,vol. 35, no. 4, pp. 484-496, July/Aug. 2009.
[14] ConnectingwithCustomers,http://www.microsoft.com/mscorp/execmail/2002/10- 02customers.mspx, 2006.
[15] R.A. Hanneman and M. Riddle, Introduction to Social NetworkMethods. Univ. of California, 2005.
[16] J.A. Fisher and S.M. Freudenberger, "Predicting ConditionalBranch Directions from Previous Runs of a Program," Proc. FifthInt'l Conf. Architectural Support for Programming Languages andOperating Systems, pp. 85-95, 1992.
[17] R. Gupta, E. Mehofer, and Y. Zhang, Profile-Guided CompilerOptimizations,pp. 143-174. CRC Press, 2002.
[18] A. Podgurski, D. Leon, P.A. Francis, W. Masri, M. Minch, J. Sun,and B. Wang, "Automated Support for Classifying Software Failure Reports," .Software Eng., pp. 465-475,2003.
[19] J.A. Jones, J.F. Bowring, and M. Harrold, "Debugging in Parallel, "Software Testing and Analysis, pp. 16-26, 2007.