Title

Daniel E. Krutz and Wei Le
Rochester Institute of Technology
1 Lomb Memorial Drive
Rocheser, NY, USA
{dxkvse, weil.le}@rit.edu

ABSTRACT

Developing software is hard. Developing bug free software is even more difficult. Buggy software often leads to crashes. A byproduct of these crashes are crash stacks, which is a list of all function or procedure calls which are currently in the application stack. Call stacks have become an important source for software developers to learn quality issues in released software. Since a same bug can be repeatedly triggered by different users, an overwhelming number of crash dumps are returned daily. Grouping call stacks has other advantages in the software development life cycle. Some of which include determining the cause of the issue, who should fix it, priority of a bug and even assist in intrusion detection.

In order to assist in the crash grouping process, we propose a new crash similarity measurement tool, Mozilla Call Stack Similarity (MCSS) **[name??]** This analyzes a selected group of all stacks in a chosen application and measures the similarity of all the analyzed call stacks using several similarity measurements as defined by previous research. In the following paper, we describe this crash measurement tool along with several potential future applications of the tool. **[Really work on this abstract]**

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Keywords

Crash Dumps, Call Stacks, Grouping, Similarity

1. INTRODUCTION

Due to the complexity of software, it is unreasonable to assume that it will be created bug free in its initial release. Thus, an important and challenging task during software maintenance is to capture and diagnose failures triggered at the client side. Examples of such feedback systems include Microsoft Dr. Watson [16] and Mozilla Crash Reporter [11]. In these systems, crash dumps, which typically contain call stacks recorded at the crash site, are returned for locating bugs in software. Crash or call stacks are are list of methods which have been invoked during the execution of the application which exist at the moment of the crash occurring. The most recent function call is at the top of the stack [4] [15] [**?**] [13]

Since a same bug can be repeatedly triggered by different users and under different execution environments, the number of crash dumps sent in daily can reach millions. Incorrectly grouping unrelated crash dumps or failing in identifying a new crash that actually belongs to an already fixed group can incur unacceptable manual overhead and potentially delay critical patches [9] [13].

In order to measure the similarity of crash dumps groups from Mozilla [1], we developed an innovative crash measurement tool known as Mozilla Call Stack Measurment (MCSM). We decided to create a tool which analyzed Mozilla crash stacks for several reasons. First of all, there is a substantial amount of Mozilla crash data which is freely available for analysis. While there are other applications which offer significant amounts of crash information, we felt that the Mozilla suite of applications was the best to analyze because it also offered the opportunity to examine several applications, not merely a single application from a single company. Secondly, a wide range of research has been done using the Mozilla suite of applications [6] [1] [8] [14] [17]. We wanted to develop a tool for an application suite which other researchers not only had access to a substantial amount of crash data, but also a suite which a large amount of research was also being conducted in. This would allow others to use the proposed tool for their own research.

Self managing systems that properly form crash grouping make bug fixing easier and cheaper largely due to the need for less human intervention and manual work. Determining how related two crashes are may assist with this process [4]. Additionally, a crash similarity measurement tool may be used to place crashes into buckets, which are groups of related call stacks [13]. These groups of related stacks are likely indicative of related crashes and related crash symptoms. Related crashes may mean that if one is fixed, all of them will be repaired as well [5].

A call stack measurement tool such as the one being proposed is beneficial for several reasons. First of all, it may

[1]http://www.mozilla.org

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICSE '2014 Hyderabad, India
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

be used to evaluate the effectiveness of an employed call stack grouping mechanism. There are many possible ways to group call stacks. Some of which include grouping using only the signature of the stack, using heuristics for string matching such as subsequences, edit distance and prefix matching [3] [10] or even machine learning techniques [2]. Additionally, a tool such as the one being proposed could be used to augment existing techniques by providing an additional data set.

[remember that we are not grouping stacks, only measuring similarity]

[remember that right now, I am only measuring call stack similarity]

MCSM works with the user first selecting the target application and version. For example, the user may choose to analyze Firefox version 19.1. MCSG then scans the crash stacks for the top 300 crashes for the application. Once the call stacks are collected, they are compared against one another in a round robin fashion using call stack comparison techniques derived from the works of Modani *et al.* [10], Brodie *et al.* [4] and Bartz *et al.* [2]. The final results are output to a final report file.

MCSM is the first third party tool which we are aware of which measures Mozilla crash stack similarity. Presently, Mozilla groups crashes based on the the top method signature of the crash stack. However, identical bugs may not share the same method signature, thus making such a grouping process inaccurate [6]. The proposed tool is not only useful for measuring Mozilla crash stack similarity, but is extensible for future research. The tool is designed in such a way that new or updated call stack measurement functionality may easily be added to the system.

The remainder of this paper is organized as follows. Section 2 describes the created tool. Section 3 describes future work on this project and applications of the tool. Section 4 discusses related works to this paper. Section 5 provides a summary of this work.

2. TOOL

The two primary components to MCSG are the call stack data collection and call stack similarity measurement mechanisms. While the tool has been developed in Java, the described process is language agnostic and may be applied by other researchers to develop a similar tool.

2.1 How Tool Works

The first step of MCSG is the data collection phase. For this step, the user selects a group of crash stacks to be analyzed. This may be for a specific Mozilla application and version. For example, the user may choose to analyze all call stacks for Mozilla 22.0. MCSG will then begin to pull the emphtop crasher call stacks for the specified application and version from the Crash Stats Mozilla website[2]. A top crasher is a crash type with the maximum number of crash reports [6].

The data collection process begins by gathering the call stacks for these top crashers. This is done using a tool known as HttpUnit[3]. Each of the collected call stacks are stored locally in a .txt file in the folder with all the other call stacks from the same top crasher group.

Once all of the folders and call stacks have been collected, the comparison process may begin. There are several existing techniques for measuring cal stack similarity [3] [2] [10]. MCSG measures call stack similarity using the call stack similarity metrics as defined in these works. If a researcher wishes to add, modify or remove a call stack similarity measurements algorithm to the tool, it is designed to make this as easy as possible.

Figure 1

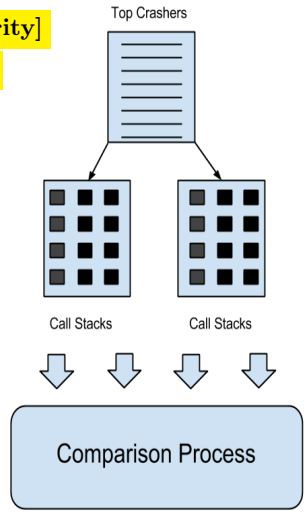

Figure 1: Overview of the tool

2.2 Initial Results

Initial Results

3. FUTURE WORK

Future Work

While MCSG works well, there is room for improvement and future work.

4. RELATED WORK

Brodie *et al.* [3] developed the techniques of normalizing strings based on length before comparing them. They applied metrics commonly used in string matching algorithms, including edit distance, longest common subsequence and prefix match.

Brodie et al. [4], proposed that similar bugs are likely to produce stacks which resemble one another. To determine if a new failure is originated from the same cause documented in the database, they developed the metrics of Brodie weight for determining similarities between call stacks. The idea is that when measuring similarity, a higher weight is placed upon items that match between the top of two stacks. The assumption is that the closer to the top of a stack a function call is, the more relevant it is to the matching process [3].

Several crash grouping mechanisms have also been proposed. Bartz *et al.* [2] defined a similar-failure search engine to find similar failures across a large global data set. The primary use of this was to allow a developer to see if a similar failure had already been resolved. This mechanism worked from Windows failure reports in the Windows Error Reporting(WER) system.

[2]https://crash-stats.mozilla.com/

[3]http://httpunit.sourceforge.net

Modani *et al.* [10] proposed and evaluated algorithms for using a weighted metric for efficiently and accurately matching call stacks. Recursive and uninformative function calls were removed from the stacks before comparison. Dang *et al.* [5] proposed a method of improving the grouping of grouping of Windows crash information received by WER called ReBucket. This used a method for clustering crash reports and was based on call stack matching.

Wang *et al.* [15] discussed three rules which they found could be used to automatically correlate crash types. These three criteria were a crash signature comparison, top frame comparison and a frequent closed ordered sub-set comparison. Using these criteria, the authors were able to identify crash correlation groups with a precision of between 79% and 100% and a recall of between 65% and 90%.

Stack traces have also been examined for a variety of other uses. Feng *et al.* [7] examined how call stack information could be used to assist in anomaly detection. Schroter emphet al. [12] concluded that stack traces in the bug reports indeed help developers fix bugs. Dhaliwal *et al.* [6] performed an empirical analysis of crash reports for Mozilla Firefox to determine the impact of crash grouping and evaluvate the efficiency of such grouping.

5. CONCLUSION

Conclusion

6. REFERENCES

[1] S. Ahsan, J. Ferzund, and F. Wotawa. Are there language specific bug patterns? results obtained from a case study using mozilla. In *Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on*, pages 210–215, 2009.

[2] K. Bartz, J. W. Stokes, J. C. Platt, R. Kivett, D. Grant, S. Calinoiu, and G. Loihle. Finding similar failures using callstack similarity. In *Proceedings of the Third conference on Tackling computer systems problems with machine learning techniques*, SysML'08, pages 1–1, Berkeley, CA, USA, 2008. USENIX Association.

[3] M. Brodie, S. Ma, G. M. Lohman, L. Mignet, N. Modani, M. Wilding, J. Champlin, and P. Sohn. Quickly finding known software problems via automated symptom matching. In *ICAC*, pages 101–110, 2005.

[4] M. Brodie, S. Ma, L. Rachevsky, and J. Champlin. Automated problem determination using call-stack matching. *Journal of Network and Systems Management*, 13(2):219–237, 2005.

[5] Y. Dang, R. Wu, H. Zhang, D. Zhang, and P. Nobel. Rebucket: a method for clustering duplicate crash reports based on call stack similarity. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 1084–1093, Piscataway, NJ, USA, 2012. IEEE Press.

[6] T. Dhaliwal, F. Khomh, and Y. Zou. Classifying field crash reports for fixing bugs: A case study of mozilla firefox. In *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance*, ICSM '11, pages 333–342, Washington, DC, USA, 2011. IEEE Computer Society.

[7] H. Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong. Anomaly detection using call stack information. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 62–75, 2003.

[8] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams. Do faster releases improve software quality? an empirical case study of mozilla firefox. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 179–188, 2012.

[9] J. Lerch and M. Mezini. Finding duplicates of your yet unwritten bug report. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 69–78, 2013.

[10] N. Modani, R. Gupta, G. Lohman, T. Syeda-Mahmood, and L. Mignet. Automatically identifying known software problems. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 433–441, 2007.

[11] M. C. Reporter. Mozilla crash reporter. https://support.mozilla.org/en-US/kb/Mozilla%20Crash%20Reporter. [Online; accessed 2013-07-25].

[12] A. Schroter, N. Bettenburg, and R. Premraj. Do stack traces help developers fix bugs? In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 118–121, 2010.

[13] H. Seo and S. Kim. Predicting recurring crash stacks. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 180–189, New York, NY, USA, 2012. ACM.

[14] J. Wang and J. Carroll. Behind linus's law: A preliminary analysis of open source software peer review practices in mozilla and python. In *Collaboration Technologies and Systems (CTS), 2011 International Conference on*, pages 117–124, 2011.

[15] S. Wang, F. Khomh, and Y. Zou. Improving bug localization using correlations in crash reports. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 247–256, Piscataway, NJ, USA, 2013. IEEE Press.

[16] M. D. Watson. Microsoft dr. watson. https://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/drwatson overview.mspx?mfr=true. [Online; accessed 2013-07-12].

[17] S. Zaman, B. Adams, and A. Hassan. A qualitative study on performance bugs. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 199–208, 2012.