Do Stack Traces Help Developers Fix Bugs?

Adrian Schröter
University of Victoria
Canada
schadr@uvic.ca

Nicolas Bettenburg
Queens University
Canada
nicbet@cs.queensu.ca

Rahul Premraj *
VU University
The Netherlands
rpremraj@cs.vu.nl
* *contact author*

***Abstract*—A widely shared belief in the software engineering community is that stack traces are much sought after by developers to support them in debugging. But limited empirical evidence is available to confirm the value of stack traces to developers. In this paper, we seek to provide such evidence by conducting an empirical study on the usage of stack traces by developers from the ECLIPSE project. Our results provide strong evidence to this effect and also throws light on some of the patterns in bug fixing using stack traces. We expect the findings of our study to further emphasize the importance of adding stack traces to bug reports and that in the future, software vendors will provide more support in their products to help general users make such information available when filing bug reports.**

***Keywords*-debugging, stack traces, empirical study, bug tracking, collaboration**

I. Introduction

Stack traces are a useful programming construct to support developers in debugging tasks. Software debugging is difficult and often involves searching through millions of lines of code to identify the cause of a defect – akin to finding a needle in a haystack. But stack traces can potentially narrow down the list of candidate files that are likely to contain the defect to speed up debugging.

Some evidence exists to suggest that stack traces indeed help debugging. In a survey, developers from three open source projects: APACHE, ECLIPSE, and MOZILLA were asked which information items they prefer in bug reports to help them resolve bugs [1]. Their responses indicated a strong preference for stack traces. Also, software vendors including Microsoft, Apple, and Mozilla are improving in-built support in their products to send stack traces back to developers when the software crashes. Furthermore, the online documentation of JAVA has an entire chapter dedicated to educating developers on how to analyze stack traces [2], which signifies their importance (Section II).

But to the best of our knowledge, no systematic investigation has been conducted to check whether software defects are actually fixed in one or more methods listed in the stack traces. This paper aims to fill this gap by examining the development and defect history of the ECLIPSE project (Section III), to answer key research questions that verify if indeed stack traces are of much worth, while resolving bugs. For the purpose of our study we consider that a stack trace contributed to fixing the bug if changes were made in one or more methods in the stack trace.

We consider our investigation important so as to encourage bug reporters to submit stack traces in their reports. In the same survey as above [1], bug reporters from the same projects were asked which information items they had previously submitted in bug reports. Stack traces were selected by only a handful of reporters and were rated as one of the most difficult information items to provide. Quantitative evidence demonstrating the value of stack traces may help motivate reporters to go the extra mile and more frequently provide stack traces in their bug reports. Our results are also insightful for developers, since we present our findings on the typical locations of defects in stack traces that may help focus their search in the future.

In order to provide evidence to show the value of stack traces, we answer the following four research questions in this paper (Section IV):

RQ1. Are bugs fixed in methods in stack traces?
RQ2. How far down the stack to trace?
RQ3. Are two (or more) stack traces better than one?
RQ4. Do stack traces help speed up debugging?

After presenting our results for the above questions, we bring the paper to a close with a discussion on our findings and presenting our conclusions (Section V).

II. Related work

Although, to our best knowledge, there is no study directly investigating the usefulness of stack traces, there are a number of studies that assume stack traces are useful. For instance, articles written by John Goerzen [3] suggest how stack traces can used be used developer to debug programs but provide no empirical evidence to this end. Assuming the importance of stack traces, Shah et al. [4] investigated how developers use exception handling. Noting that most developers take little care of exception handling, they argued that better exception handling is important to produce more meaningful stack traces in the event of a failure.

Also tools (such as compilers) with built-in ability to create stack traces assume the usefulness of stack traces. Allwood et al. [5] implemented the feature of creating stack traces into Haskell, which only pays off if stack traces are actually helpful in debugging. Further, Microsoft (and

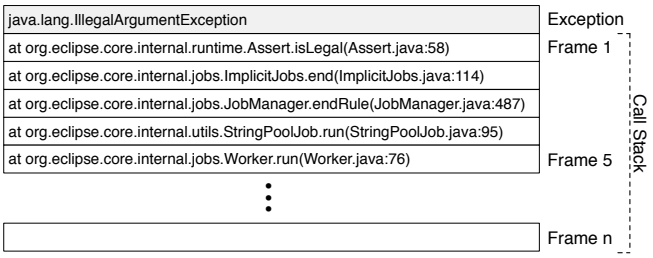

Figure 1. Sample stack trace extracted from ECLIPSE bug #111106.

other large software vendors) is collecting crash reports from a large number of computers [6] to help their developers resolve their crashes. While their efforts in this direction are considerable, the question whether the stack traces indeed help remains to be answered.

III. Data collection

Our study has been conducted using the development data from the ECLIPSE project developed in JAVA. The following summarizes how we extracted the relevant data to perform the study.

A. Stack Traces

A typical JAVA stack trace consists of an ordered list of methods or stack frames that were active on the call stack before an *exception* or *error* occurred (Figure 1). Each frame contains the full-qualified name of the method and the exact location of the execution inside the source code through a file name and line number. We consider the first (top-most) frame in the stack as the method that caused the crash, because the exception occurred in this method. To exemplify, in the stack trace presented in Figure 1, the `isLegal()` method, which is defined in class `Assert` in file `Assert.java`, crashed when executing the program at line `58`.

The ECLIPSE project uses the BUGZILLA bug tracking system to organise their maintenance activities. We obtained a copy of this data containing 161,500 bug reports filed until October 2006. We then parsed these bug reports and the associated comments to extract all available stack traces using the infoZilla tool that uses a set of complex regular expressions to identify and extract JAVA stack traces [7]. The extracted information from the stack traces includes the bug id, exception thrown, method frames, and order of the stack trace (many bug reports contained multiple stack traces). Note that we extracted stack traces from the bug reports by only parsing the main description and the following comments. Stack traces that were perhaps reported in attachments submitted by reporters were not included in the study.

B. Location of bug fixes

In order to determine the locations changed in the source code to fix bugs, we mined the version repository of the ECLIPSE project. Many developers use the commit feature of version archives (`CVS` in this case) to annotate each change to the source code with a log message that describes the reason for that change. We scanned these messages for references to bug reports such as "Fixed 4223" or "bug #23444". Every such number is a potential reference to a bug report, however they have a low trust at first. We increase the trust level when the message contains keywords such as "fixed" or "bug" or matched patterns like "# and a number", as described in the work by Śliwersky et al. [8].

For each change found to describe a fix for a bug with sufficiently high confidence, we performed a syntactical analysis to retrieve information about which methods, classes and packages were changed. Thereafter, information regarding changed locations collected for bug reports found to contain stack traces were mapped to the methods listed in the respective stack traces. In case the bug report was resolved as a DUPLICATE, its stack traces were mapped to the fix locations of the original bug report.

IV. Results

Our findings from the investigations into the research questions are presented in this section.

A. RQ1: Are bugs fixed in methods in stack traces?

Our first research question is to find whether bug reports that contain stack traces are fixed in any of the constituent frames. We address this question by presenting a summary of the data collected in Section III, which reflects on the usefulness of stack traces and answers our research question.

We identified 12,947 bug reports from the ECLIPSE bug database in which at least one stack trace was submitted. These bug reports amount to a little less than 10% of all bugs in the database. Of these bug reports, 8,580 were fixed (i.e., their status was FIXED) and 3,940 could be linked to their fixes in the version control system. Another 4,050 bug reports were identified with links to their changes but their status was other than FIXED. From the 3,940 linked and fixed bug reports, 2,321 were observed to be fixed in one of the stack frames from the traces submitted in the report. Thus, almost 60% fixed bugs reports with stack traces and could be linked to their fixes were fixed in one of the stack frames. Note that this is a conservative estimate in that we chose to consider only those bug reports that are fixed and could be linked using the commit logs to their fixes. It is likely that other bugs that could be linked to their changes in the repository but did not have a FIXED status (4,050 bugs) involved changes in one of the stack frames. Overall, these numbers suggest that developers favour stack traces in bug reports because they can indeed help identify candidate locations that must be changed to resolve the bug and support the same findings from a previous survey [1].

We also identified a total of 25,127 unique stack traces in the database. Some bug reports contained multiple stack traces, which were submitted in the reports' comments or

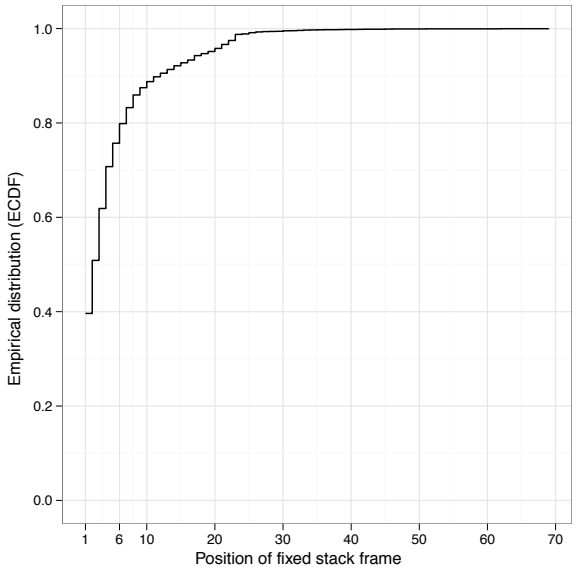

Figure 2. ECDF of the position of the top-most frame in the stack trace changed to fix the bug.

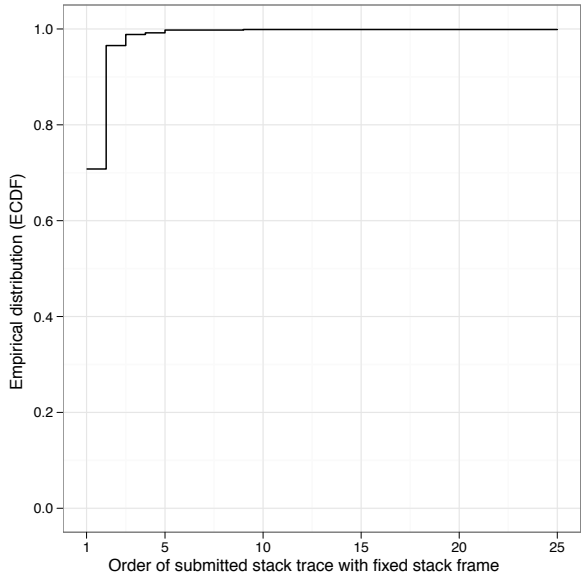

Figure 3. ECDF of the order of the most important stack trace with a fix location.

their duplicates. As many as 7,968 stack traces belonged to bug reports that were fixed and could be linked to the fixes and 3,809 of these stack traces contained at least one frame that was changed in order to fix a bug.

B. RQ2: How far down the stack to trace?

Stack traces can vary substantially in length. In the ECLIPSE project, we observed stack traces ranging from a single frame to 1,024 frames (bug #18625) and a median length of 25 frames. The length of the stack traces raises practical concern for debugging purposes when there is no certain way to know which frame contains the defect. At some point in time, marginal returns kick in and the hope to find the defect in one of the lower frames begins to rapidly fade.

We study the 2,321 bugs from the ECLIPSE project that were fixed in one of the stack frames to investigate how far down the stack trace is it worthwhile to examine and potentially locate the defect. Figure 2 is the empirical distribution frequency (ECDF) plot of the position of the earliest frame in stack trace that was fixed. In the plot, the *x*-axis represents the position of the fixed frame and the *y*-axis represents the percentage of bugs that were fixed in a frame at that or an earlier position. Thus, any point *(x,y)* on the curve denotes that *y*% bugs were fixed in a frame by the *x*th position.

Figure 2 gives us several insights into where defects were located and fixed in stack traces. Firstly, 40% of bugs were fixed in the very first frame, while 80% of bugs are fixed within the top-6 stack frames. Close to 90% bugs were fixed within the top-10 stack frames. Thereafter, only a small percentage of additional bugs were fixed in frames in between the 10th and 20th position. We can draw from these results that when a stack trace is reported, defects typically lie within the top-10 frames. Examining the stack trace further is unlikely to yield results. Having said that, in an exceptional case in our study, we observed a fix for a bug in the 69th stack frame (ECLIPSE bug #63898).

C. RQ3: Are two (or more) stack traces better than one?

In many cases, we observed several stack traces submitted to a single bug report in their comments and together with their duplicates (e.g., bug #3128 together with its 14 duplicates contained 52 stack traces). In a previous study, we found that information in duplicates can often be helpful because they provide developers with multiple perspectives on the same bug [7]. Since many reports also contain multiple stack traces, we investigate if this helps debugging by providing several locations as starting points for a code inspection.

Among the 12,947 bug reports with stack traces, 4,206 reports (32.5%) contained more than one stack trace. Of these 4,206 reports, 3,049 bug reports (72%) have been marked as fixed. This is a significantly higher rate of fixed reports than the 5,531 fixed bugs among the 8,741 reports (63%) with one stack trace ($p < .00001$ using Chi square test, $\chi^2 = 107.4667$).

We believe the above comparison of rates of fixed bugs paints only half the picture on the value of multiple stack traces. We carry our investigation further by checking which of the multiple stack traces are of more value to developers. For our purpose, we consider a stack trace more important if the position of fixed stack frame is higher than those in other stack frames. For each bug report (combined with its duplicates), we then note the order of the most important stack trace. The results are plotted as an ECDF in Figure 3. The *x-axis* of the bug report denotes the order of the most important stack trace and the *y-axis* denotes the percentage of bugs fixed in one of the stack frames from the corresponding stack trace. Thus, a position *x,y* on the

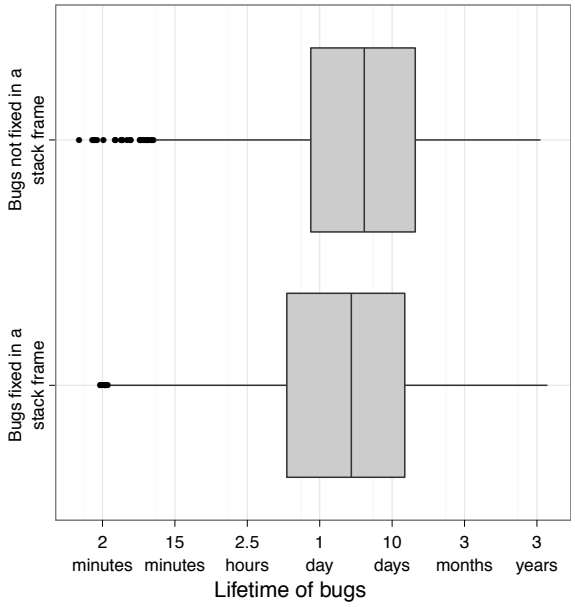

Figure 4. Boxplots comparing lifetimes of bugs reports with stack traces that were either fixed in a stack frame or not.

curve denotes that y% bugs with multiple stack traces were fixed in a frame from up to the xth stack trace. The ECDF shows that 70% of the bugs are fixed in a frame from the first stack trace already. While close to 95% of the bugs are fixed using the first three stack traces. Thus, the first stack trace submitted appears to be most helpful in fixing the bug. In some sense, this contradicts our result above that bug reports with multiple stack traces have a higher rate of fixes. We expect that this may still be the case because multiple stack traces indeed provide developers with additional information about the bug and also, we noted in the ECDF that more than 20% bugs were fixed in the second and third stack traces. Again, in an exceptional case, we noted a ECLIPSE bug fixed in the 25th stack trace (ECLIPSE bug #64358).

D. RQ4: Do stack traces help speed up debugging?

In a previous study [1], we noted that bug reports that contained stack traces had significantly shorter lifetimes than other bug reports, i.e., they got resolved sooner. This is likely because stack traces reliably indicate which parts of the code could contain the defect and this helps speed up debugging. We take our analysis further by examining whether bug reports that are fixed in a stack frame have a shorter lifetime than those, which were not fixed in any stack frame.

For our analysis, we consider all 3,940 identified bug reports that contained at least one stack trace, had status FIXED and could be linked to their fixes in the version repository. We then compared the lifetimes of the 2,321 bug reports fixed in one of the stack frames with the lifetimes of the remaining bug reports. Figure 4 is a boxplot visualising and comparing the distribution of lifetimes of the two groups of bug reports. The median and mean lifetimes of bugs fixed in a stack frame are 2.73 days and 26.44 days respectively, while same for bugs that were not fixed in a stack frame are 4.13 days and 32.88 days respectively. A statistical test (the non-parametric Wilcoxson rank sum test with α set to .05) comparing the lifetimes of the bug reports also confirmed that the differences in the distributions are statistically significant with $p < .00001$. The median and mean lifetimes of all fixed bug reports in the ECLIPSE project are 6.9 and 48.5 days. The results show strong evidence to suggest that bug reports with a stack trace have shorter lifetimes and even more so when the bug is fixed in one of the stack frames.

V. Conclusion and consequences

Stack traces are generally regarded as helpful to developers when debugging programs. With this study, we have aimed to provide empirical evidence in support of the usefulness of stack traces to ECLIPSE developers by examining key patterns in the the resolution of bug reports that contained stack traces. Our study showed that up to 60% FIXED bug reports that contained stack traces involved changes to one of the stack frames. Also, the average lifetime of these bug reports is significantly lower than that of other reports. Furthermore, we found that a defect is typically to be found in one of the top-10 stack frames. We expect the findings emphasize the importance of encouraging bug reporters to provide this information using tool support or other means.

In the future, we seek to expand our investigation by studying multiple projects. We have also planned to investigate other aspects which exceptions are commonly submitted, which ones get fixed, and also explore opportunities to support reporters to add stack traces to their reports.

References

[1] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Procs of FSE*. ACM, 2008, pp. 308–318.

[2] Java, "Analyzing stack traces," http://java.sun.com/developer/onlineTraining/Programming/JDCBook/stack.html, last accessed 2009-12-21.

[3] J. Goerzen, "Finding stubborn bugs with meaningful debug info," *Linux J.*, vol. 2005, no. 129, p. 7, 2005.

[4] H. Shah, C. Görg, and M. J. Harrold, "Why do developers neglect exception handling?" in *Procs. of the Int. Workshop on Exception Handling*. ACM, 2008, pp. 62–68.

[5] T. O. Allwood, S. Peyton Jones, and S. Eisenbach, "Finding the needle: stack traces for ghc," in *Procs. of the Symposium on Haskell*. ACM, 2009, pp. 129–140.

[6] K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle, and G. Hunt, "Debugging in the (very) large: ten years of implementation and experience," in *Procs. of the Symposium on Operating systems principles*. ACM, 2009, pp. 103–116.

[7] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful?" in *Procs. of ICSM*. Beijing: IEEE, September 2008, pp. 337–345.

[8] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in *Procs. of MSR*. ACM, 2005, pp. 1–5.