The Journal of Systems and Software 106 (2015) 82–101

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

An exploratory study on exception handling bugs in Java programs

Felipe Ebert[a,*], Fernando Castor[a], Alexander Serebrenik[b]

[a] Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE), Recife 50670-901, Brazil
[b] Eindhoven University of Technology, Eindhoven, The Netherlands

ARTICLE INFO

Article history:
Received 22 August 2014
Revised 29 March 2015
Accepted 20 April 2015
Available online 24 April 2015

Keywords:
Exception handling
Bugs
Repository mining

ABSTRACT

Most mainstream programming languages provide constructs to throw and to handle exceptions. However, several studies argue that exception handling code is usually of poor quality and that it is commonly neglected by developers. Moreover, it is said to be the least understood, documented, and tested part of the implementation of a system. Nevertheless, there are very few studies that analyze the actual exception handling bugs that occur in real software systems or that attempt to understand developers' perceptions of these bugs. In this work we present an exploratory study on exception handling bugs that employs two complementary approaches: a survey of 154 developers and an analysis of 220 exception handling bugs from the repositories of Eclipse and Tomcat.

Only 27% of the respondents claimed that policies and standards for the implementation of error handling are part of the culture of their organizations. Moreover, in 70% of the organizations there are no specific tests for the exception handling code. Also, 61% of the respondents stated that *no* to *little* importance is given to the documentation of exception handling in the design phase of the projects with which they are involved. In addition, about 40% of the respondents consider the quality of exception handling code to be either *good* or *very good* and only 14% of the respondents consider it to be *bad* or *very bad*. Furthermore, the repository analysis has shown (with statistical significance) that exception handling bugs are ignored by developers less often than other bugs. We have also observed that while overly general `catch` blocks are a well-known bad smell related to exceptions, bugs stemming from these `catch` blocks are rare, even though many overly general `catch` blocks occur in the code. Furthermore, while developers often mention empty `catch` blocks as causes of bugs they have fixed in the past, we found very few bug reports caused by them. On top of that, empty `catch` blocks are frequently used as part of bug fixes, including fixes for exception handling bugs.

Based on our findings, we propose a classification of exception handling bugs and their causes. The proposed classification can be used to assist in the design and implementation of test suites, to guide code inspections, or as a basis for static analysis tools.

1. Introduction

Modern software systems must include provisions to handle errors at runtime. An error is "part of the system internal state which is liable to lead to subsequent failure", while "a system *failure* occurs when the service delivered by the system deviates from what the system is aimed at" (Garcia et al., 2001, p. 198). Errors may stem from application logic-related erroneous conditions, *e.g.*, an invalid bank account number, undetected bugs, *e.g.*, null dereferences and arithmetic overflow, or environmentally triggered erroneous conditions, *e.g.*, impossibility to open a file or communicate via network. Exception handling mechanisms promote separation of concerns between the normal execution flow of an application and the execution flow in which errors are handled.

At the beginning, exceptions were handled just by returning error codes (success or failure) (Cabral and Marques, 2007). ML (Milner et al., 1990) was the first programming language that implemented typed exceptions, *i.e.*, it allowed developers to define a new type (within the language) for each different type of error. Prior to that, different types of errors were defined by values in the language. The use of types is important because it promotes static checking of exception usage. The throw-catch style of exception signaling and handling was introduced by LISP (McCarthy, 1978). More recently, several modern programming languages, like Java, Ruby, C#, C++ and Scala, implement exception handling and a considerable part of the system source code is often dedicated to error detection and handling (Cabral and Marques, 2007; Weimer and Necula, 2008). Nonetheless, developers have a tendency to focus on the normal behavior of the applications, *i.e.*, what it should do when no errors occur, and deal

* Corresponding author. Tel.: +558196491241.
E-mail addresses: fe@cin.ufpe.br, felipe.ebert@gmail.com (F. Ebert), castor@cin.ufpe.br (F. Castor), a.serebrenik@tue.nl (A. Serebrenik).

http://dx.doi.org/10.1016/j.jss.2015.04.066
0164-1212/© 2015 Elsevier Inc. All rights reserved.

with error handling only during the system implementation, in an *ad hoc* manner (Cristian, 1989; Reimer and Srinivasan, 2003).

Several studies (Cristian, 1989; Reimer and Srinivasan, 2003; Shah et al., 2010) argue that the quality of exception handling code is usually poor and that this part of the code is commonly neglected by developers. Moreover, the exception handling code is often hard to test due to both the numerous exceptional conditions that might occur in a non-trivial application and to the need to stimulate all possible causes for exceptions during testing (Coelho et al., 2011). Quality of exception handling code, lack of developers' attention, and testing-related challenges specific to exception handling code can therefore be expected to create a fertile ground for bugs. Nevertheless, very few studies analyze exception handling bugs (EH-bugs) occurring in real software systems and no study has attempted to understand developers' perceptions about these bugs. We consider an EH-bug to be a bug whose cause is related to exception handling. It may be a problem related to the definition, throwing, propagation, handling, or documentation of exceptions, to situations where an exception should be throw or handled but is not, and to the use of clean-up actions associated to regions of the code that may throw exceptions.

In this paper, we address this challenge by conducting an empirical study of EH-bugs, *i.e.*, bugs caused by the definition, throwing, propagation, handling, or documentation of exceptions. We aim to gain a better understanding of the causes of these bugs, their frequency, severity, and difficulty of fixing them. Such understanding can be beneficial not only for developers but also for tool designers aiming at supporting software developers in their daily tasks. We complement the objective information about EH-bugs with an investigation of the developers' perceptions about exception handing, in general, and EH-bugs, in particular. The combination of the empirical study of EH-bugs with the investigation of their perceptions allows us to triangulate our findings, such a triangulation being considered an important step in empirical software engineering research (Runeson and Hst, 2009).

The study involved, therefore, analysis of two data sources: (i) 220 bug reports related to error handling from the Bugzilla repositories of two large systems, Tomcat and Eclipse; and (ii) 154 responses to a survey conducted with software developers from industry and academia. Furthermore, we have inspected the source code of patches attached to the aforementioned bug reports, when available. We have considered the following four research questions:

RQ1: Do organizations and developers take exception handling into account? Our findings indicate that developers do pay attention to exception handling, even though their organizations do not. Only 27% of the respondents claimed that policies and standards for the implementation of error handling are part of the culture of their organizations. In 70% of the organizations there are no specific tests for the exception handling code. Furthermore, 61% of the respondents stated that their organizations give little to no importance to the documentation of exception handling in the design phase. In contrast, 66% of the respondents claim that they employ exception handling to create ways to tolerate faults and 63% do it to improve the system functionality (they could select multiple answers). Only 17% use exception handling mainly for debugging and 21% because of organizational policies. In addition, the repository analysis has shown that exception handling bugs are ignored by developers less often than other bugs. For example, for Eclipse, 96.65% of EH-bugs have the "Fixed" resolution whereas only 3.26% have "Wontfix" or "Worksforme" as resolutions.

RQ2: How common are EH-bugs? Developers seem to overestimate the frequency of occurrence of EH-bugs. On the average, they believe that 9.72% of the bugs in a system are EH-bugs. Analysis of the bug repositories of Eclipse and Tomcat yielded much smaller percentages, 0.35 and 1.87%, respectively.

RQ3: Are EH-bugs harder to fix than other bugs? Following the suggestion of Fonseca et al. (2010) we employed the bug fixing time and the number of discussion messages as proxies for the difficulty of fixing a bug. The analysis of the bug repositories of Tomcat revealed that there is no significant difference for both proxies. For Eclipse there was a statistically significant difference only for the number of discussion messages: it is greater for EH-bugs. This could be an evidence that EH-bugs are as hard to fix as any other bug but they might generate lengthier discussions.

Table 1
Classification of EH-bugs.

Lack of a handler that should exist
Exception not thrown
Error in the handler
Error in the clean-up action
Exception caught at the wrong level
General `catch` block
Wrong exception thrown
Exception that should not have been thrown
Wrong encapsulation of exception cause
Lack of a `finally` block that should exist
Error in the exception assertion
Inconsistency between source code and API documentation
Empty `catch` block
Error in the definition of exception class
`catch` block where only a `finally` would be appropriate

RQ4: What are the main causes of EH-bugs? We discovered that bug reports describing bugs stemming from overly general `catch` blocks, a well-known bad smell in programs that use exceptions (Cabral and Marques, 2007; Robillard and Murphy, 2003), are rare, even though there are many opportunities for them to occur and developers report that they have encountered this kind of bug in the past. Empty `catch` blocks, another well-known bad smell, are not only prevalent, as previously reported in literature (Cabral and Marques, 2007), but also commonly used as part of bug fixes, including fixes for EH-bugs. Moreover, developers often state in the code, by means of comments, that these `catch` blocks do not capture exceptions in practice. However, we found very few bug reports (only 2 among 220) whose causes are empty `catch` blocks, although developers often mention empty `catch` blocks as causes of bugs they have fixed in the past.

In addition to the aforementioned findings, we present a classification of EH-bugs, on Table 1, and their causes—reported during the survey or obtained by analyzing bug reports. The proposed classification can be used as a checklist to design test cases and to assist during code reviews, as well as a basis for static analysis tools for code defect detection, *e.g.*, similar to FindBugs (Ayewah et al., 2008).

The remainder of this paper is organized as follows. Section 2 presents the methodology used in this work and also the threats to the validity. Section 3 presents the results from the survey and the analysis of the bug repositories and also our proposed classification for causes of EH-bugs. Section 5 discusses related work and Section 6 summarizes the contributions and conclusions of this work and discusses future work. Finally, Appendix A presents a comprehensive explanation of the comparison between ours and Barbosa et al. (2014) EH-bug classification.

2. Methodology

To explore the reality of both EH-bugs and developers' perceptions, we combined investigation of bug repositories with a survey of developers' opinions. Our study focuses on two large and mature open source applications: Tomcat[1] and Eclipse.[2] Both systems are written in Java (which comprise Java versions from 1 to 7, but our analysis did not make any distinction based on the Java version), the language that arguably popularized exception handling, and use

[1] http://tomcat.apache.org

[2] http://www.eclipse.org