Information and Software Technology xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Information and Software Technology

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

Not all bug reopens are negative: A case study on eclipse bug reports

Qing Mi[*,a], Jacky Keung[a], Yuqi Huo[b], Solomon Mensah[a]

[a] Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong
[b] School of Information, Renmin University of China, Beijing, China

ARTICLE INFO

Keywords:
Non-negative bug reopen
Bug report
Reopen cycle
Data quality
Open source software
Empirical software engineering

ABSTRACT

Context: We observed a special type of bug reopen that has no direct impact on the user experience or the normal operation of the system being developed. We refer to these as *non-negative bug reopens*.
Objective: Non-negative bug reopens are novel and somewhat contradictory to popular conceptions. Therefore, we thoroughly explored these phenomena in this study.
Method: We begin with a novel approach that preliminarily characterizes non-negative bug reopens. Based on bug reports extracted from Eclipse Bugzilla, we then examined a case study to compare non-negative and regular bug reopens using the Wilcoxon-Mann-Whitney test.
Results: The results show that non-negative bug reopens are statistically significantly different than regular bug reopens, based on their survival times and the number of developers involved in the entire debugging process.
Conclusion: Taking into account the significant differences, we suggest that the effects of non-negative bug reopens should be considered in future research in related areas, such as bug triage and reopened bug prediction.

1. Introduction

A software bug that has been REOPENED for further processing after being RESOLVED/VERIFIED/CLOSED is known as a*reopened bug*. Approximately 6%–10% of total bugs are eventually reopened [1], but the problem of bug reopening has only recently attracted the attention of the software engineering community. Zimmermann et al. [2] characterized the reopening process using a mixed approach. Shihab et al. [3] constructed decision trees to predict whether a bug will be reopened. Xia et al. [4] used a combination of three classifiers to improve the performance of reopened bug prediction.

Research on bug reopens is important to the software engineering community as it provides capabilities needed to: 1) Plan maintenance efforts taking into account the reopening rate [1]; 2) Characterize the actual quality of the bug fixing process [2]; 3) Prepare practitioners to think twice before closing a bug [3,4]. Our work here is complementary to previous studies.

Bug reopens are normally considered to be negative since they take longer to be resolved [1,3,4], cause rework for already-busy developers [1,4], and degrade the user-perceived quality of the software product [3]. However, the results of this study indicate otherwise. Table 1 lists a variety of reopen reasons determined in our previous investigation [1]. Further analysis revealed that a significant proportion of bug reopens have no direct effect on the user experience or the normal operation of the system being developed. For instance, R4, R6, and R7 in Table 1 may be considered to be either *non-essential* or *non-negative* bug reopens. A case study on Eclipse bug reports is examined; we show, statistically, that non-negative bug reopens are significantly different than regular bug reopens, based on their survival times and the number of developers involved in the entire debugging process. Thus, we propose that non-negative bug reopens should not be treated as equivalent to regular bug reopens.

2. Identifying non-negative bug reopens

We begin by presenting an overview of the reopen cycle and its corresponding taxonomy. Three patterns are then proposed to characterize non-negative bug reopens.

2.1. Reopen cycle and its representation

Our research is based on Bugzilla,[1] a popular system used to track reported software bugs [5]. Fig. 1 shows the typical timeline for a Bugzilla bug. Note that the lifetime for a regular bug (the upper arrow in Fig. 1) is considered to be completed when it first reaches the status RESOLVED/VERIFIED/CLOSED, whereas a reopened bug (the lower arrow in Fig. 1) survives for at least one more reopen cycle as depicted in the dotted rectangles in Fig. 1.

* Corresponding author.
E-mail addresses: Qing.Mi@my.cityu.edu.hk (Q. Mi), Jacky.Keung@cityu.edu.hk (J. Keung), bnhony@ruc.edu.cn (Y. Huo), smensah2-c@my.cityu.edu.hk (S. Mensah).
[1] https://www.bugzilla.org.

https://doi.org/10.1016/j.infsof.2018.03.006
Received 4 October 2016; Received in revised form 29 January 2018; Accepted 5 March 2018

Table 1
Root causes for bug reopens. The reason IDs are in accordance with those used in our previous work [1]. The eighth category of reopen reasons, rare causes, is intentionally omitted from this study.

Classification	ID	Reopen reason	Description	Type
Unsuccessful fix	R1	Bug reporter's unclear description	The bug-related information (e.g., steps to reproduce) provided by the reporter may be incomplete, ambiguous, or inaccurate.	T2
	R2	Developer's negligence	The assigned developers do not treat the reported bug seriously.	T2
	R3	Introduction of additional problems	The bug has been resolved, yet some unexpected problems (e.g., instability) are introduced into the system.	T1
	R5	Incomplete fix	The bug is thought to be resolved, yet someone reproduces the issue on a later version.	T1
Further improvements and other reasons	R4	A better solution	The bug has been resolved, yet a better solution is found afterwards (typically by the same developer).	T1
	R6	Misapprehension	The bug has been resolved, yet the reporter mistakenly considers that the issue still exists.	T1, T4
	R7	Inappropriate status	The bug has been resolved, yet the accompanying attributes (e.g., resolution) may be improperly or incorrectly completed.	T2, T3, T4

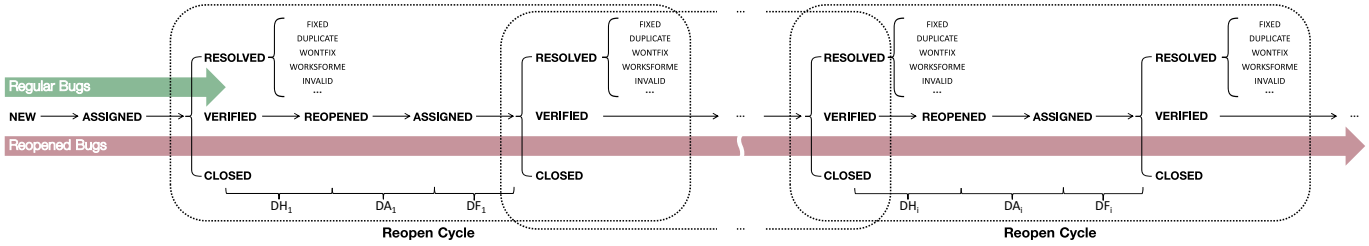

Fig. 1. Typical timeline for a Bugzilla bug.

A typical reopen cycle consists of four stages: RESOLVED/VERIFIED/CLOSED, REOPENED, ASSIGNED, and then back to RESOLVED/VERIFIED/CLOSED. Every time a bug goes through the RESOLVED/VERIFIED/CLOSED stage, one of the possible resolutions (FIXED, DUPLICATE, WONTFIX, WORKSFORME, INVALID, NOT_ECLIPSE, LATER, and REMIND) is used to describe what happened to the bug. Ideally, all resolutions should be properly verified, yet many bugs end up with the status RESOLVED in practice.

To represent the reopen cycle, we followed the method detailed by Jongyindee et al. [6], which is designed to symbolize bug history in the form of status changes (e.g., NEW ⇒ ASSIGNED ⇒ RESOLVED (FIXED)). Because the objective of our study differed from that of Jongyindee et al., we used a relatively concise format that ignored bug status and was restricted only to resolutions. Accordingly, each reopen cycle can be characterized as a pair of resolutions, for instance, WORKSFORME ⇒ FIXED. In this scheme, "⇒" stands for sequence rather than causality.

2.2. Taxonomy of reopen cycles

Given that reopen cycles served as the basis for this study, a classification framework was developed to facilitate further analysis.

Based on whether a check-in to a code repository is involved, resolutions fall broadly into two groups: Fixed (FIXED) and Non-Fixed (DUPLICATE, WONTFIX, WORKSFORME, INVALID, NOT_ECLIPSE, LATER, and REMIND). Accordingly, all reopen cycles can be divided into the following four types: **T1.** Fixed ⇒ Fixed; **T2.** Non-Fixed ⇒ Fixed; **T3.** Fixed ⇒ Non-Fixed; **T4.** Non-Fixed ⇒ Non-Fixed.

As shown in Table 1, R1 and R2 are the typical scenarios for T2, whereas R3, R4, and R5 fall into T1. However, R6 and R7 cannot be classified into any single type; these were the main targets for our identification as non-negative bug reopens.

2.3. Patterns of non-negative bug reopens

The primary criteria for non-negative bug reopens are that neither the user experience nor the system execution performance is significantly affected. Therefore, we focused solely on reopen cycles without code modification; thus, only T3 and T4 were within the scope of our research. Accordingly, a total of three patterns were proposed to identify non-negative bug reopens:

- **P1.** Fixed ⇒ INVALID/WORKSFORME/NOT_ECLIPSE (T3)
  The pattern is proposed to partially identify R7. In some circumstances, reported bugs are erroneously labeled as FIXED when no code churns have actually been made. The assigned developer realizes his or her oversight and reopens the bug to provide a proper resolution.
- **P2.** Non-Fixed ⇒ The same resolution (T4)
  The pattern is comprised of a pair of consistent resolutions (e.g., INVALID ⇒ INVALID) and is used to partially identify: 1) R6 (where the bug is closed with the same resolution after being mistakenly reopened) and 2) R7 (where the bug maintains the same resolution after being reopened to correct the accompanying attributes (e.g., severity and priority)).
- **P3.** INVALID/WORKSFORME/NOT_ECLIPSE ⇒ INVALID/WORKSFORME/NOT_ECLIPSE (T4)
  The pattern is used to partially identify R7. In practice, some bug reopens result from developers' incomplete understanding of statuses such as INVALID, WORKSFORME, and NOT_ECLIPSE, that all represent false positives to varying degrees.

3. Case study

We first present a brief introduction to the studied systems and dataset. After that, we describe the case study on Eclipse bug reports