Information and Software Technology 86 (2017) 37–53

Contents lists available at ScienceDirect

Information and Software Technology

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

Towards an understanding of change types in bug fixing code

Yangyang Zhao[a], Hareton Leung[b], Yibiao Yang[a], Yuming Zhou[a,*], Baowen Xu[a]

[a] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[b] Department of Computing, Hong Kong Polytechnic University, Hong Kong

ARTICLE INFO

Article history:
Received 29 June 2016
Revised 11 February 2017
Accepted 13 February 2017
Available online 20 February 2017

Keywords:
Change
Bug fixing code
Empirical study
Understanding
Software quality

ABSTRACT

Context: As developing high quality software becomes increasingly challenging because of the explosive growth of scale and complexity, bugs become inevitable in software systems. The knowledge of bugs will naturally guide software development and hence improve software quality. As changes in bug fixing code provide essential insights into the original bugs, analyzing change types is an intuitive and effective way to understand the characteristics of bugs.

Objective: In this work, we conduct a thorough empirical study to investigate the characteristics of change types in bug fixing code.

Method: We first propose a new change classification scheme with 5 *change types* and 9 *change subtypes*. We then develop an automatic classification tool CTforC to categorize changes. To gain deeper insights into change types, we perform our empirical study based on three questions from three perspectives, i.e. across project, across domain and across version.

Results: Based on 17 versions of 11 systems with thousands of faulty functions, we find that: (1) across project: the frequencies of change subtypes are significantly similar across most studied projects; interface related code changes are the most frequent bug-fixing changes (74.6% on average); most of faulty functions (65.2% on average) in studied projects are finally fixed by only one or two change subtypes; function call statements are likely to be changed together with assignment statements or branch statements; (2) across domain: the frequencies of change subtypes share similar trends across studied domains; changes on function call, assignment, and branch statements are often the three most frequent changes in studied domains; and (3) across version: change subtypes occur with similar frequencies across studied versions, and the most common subtype pairs tend to be same.

Conclusion: Our experimental results improve the understanding of changes in bug fixing code and hence the understanding of the characteristics of bugs.

1. Introduction

1.1. Motivation

With the increasing size and complexity of software systems, bugs become inevitable in software development. Consequently, to enhance software quality, it is essential that bugs be understood and fixed as early as possible. As bugs are usually caused by specific issues and involve fixes in the related source code, the types of bugs highly correlate with the types of source code changes for bug fixing. For example, bugs caused by data issues are naturally fixed by modifying data-related code. Similarly, bugs manifested by interface errors are usually fixed by correcting interface-related code. In other words, changes in bug fixing code provide essential insights into the original bugs. Therefore, analyzing change types in bug fixing code should be an intuitive and effective way to understand the characteristics of bugs.

The study on changes in bug fixes could enable practitioners to better understand the general characteristics like the nature of changes, the most common kind of change types, and the frequencies of change types across versions, projects, or domains. Since bug-fixing change types are associated with the specific kinds of source code that are faulty [7–9], the most common change types indicate which kinds of code are most likely to be faulty. When developing a new software project, this knowledge could guide developers to pay more attention to the statements with the highest chance of being faulty. In particular, if the frequencies of bug-fixing change types are found similar across versions, projects, or domains, developers could learn from the previous bug fixing activities and avoid making similar mistakes again. Besides, when a code bug occurs, this knowledge can also guide developers to prioritize source code to be reviewed, debugged, or tested.

* Corresponding author.
E-mail address: cs.zhou.yuming@gmail.com (Y. Zhou).

http://dx.doi.org/10.1016/j.infsof.2017.02.003
0950-5849/

Despite the above benefits, little effort has been devoted to thoroughly investigate changes in bug fixing code with a broad range of projects, mainly due to the following challenges: (1) lack of adequate bug-fixing data. To draw meaningful conclusion, it is necessary to collect a large number of bug fixes for empirical study. However, as many projects did not provide adequate public history data for analysis, it tends to be difficult to identify bug-fixing changes in these projects; (2) lack of a well-accepted taxonomy for bug fixing changes. There have been a number of studies on categorizing bugs, such as cause-driven or document-driven fault classification [1–3], and fault trigger based classification [4,5]. However, most of them are not associated with source code, thus offering little help on code review; and (3) lack of automatic tool support. Manual change categorization is subjective and unreliable. If taxonomies do not provide concrete guidelines on classification, a change is likely to be categorized differently by different practitioners based on their own understanding and background. Besides, manual change categorization is time-consuming, thus it is infeasible to study a broad range of systems. Consequently, to comprehensively investigate bug fixing changes, it is necessary to develop automatic tools to overcome the problems of manual categorization.

As a result, the primary motivation of this study is to overcome the above challenges and perform a thorough investigation with a relatively large number of projects to improve the understanding on change types in bug fixing code.

1.2. Contributions

As the change types in bug fixing code are essential reflections of the original kinds of bugs, we attempt to study bugs from the view of fine-grained code changes in the bug fixing process. To classify code changes for faulty functions, we develop a taxonomy of *change types*: Data, Computation, Interface, Logic/control, and Others, which are language independent. To support a detailed analysis in the follow-up experiments, we further subdivide the five change categories into 9 *change subtypes,* i.e. changes on data declaration or initialization statements (CDDI), changes on assignment statements (CAS), changes on function declaration/definition statements (CFDD), changes on function call statements (CFC), changes on loop statements (CLS), changes on branch statements (CBS), changes on return/goto statements (CRGS), changes on preprocessor directives (CPD), and other changes (CO).

In this study, we study the code change types based on projects developed in C language. The primary reasons are that: (1) there is no previous work devoted to studying the bug fixing code changes in these systems; and (2) there is currently no automatic change type classification tools for C language (to the best of our knowledge). We develop an automatic tool CTforC based on Coccinelle [12,13]. CTforC has three main components, i.e. change location, change pattern detection, and change type classification. It can automatically identify faulty functions, match changed code, and finally output change types. The evaluation results show that CTforC can achieve accuracies consistently above 98% when compared to manual change classification.

With the defined change types, firstly, we study 11 well-known open-source systems to explore the general characteristics of change types across projects; Secondly, we investigate 3 domains, i.e GNU, Apache, and Tool, to examine whether the frequencies of change subtypes are domain specific; Finally, we examine two systems with multiple versions (i.e. Apr versions and Libav versions), to study whether change subtypes occur with similar frequencies in different versions of the same system.

Based on 17 versions of 11 C systems with thousands of faulty functions, we have the following key findings.

- Across projects. There are general characteristics of change types across studied projects. More specifically, interface related code changes are usually the most frequent bug-fixing changes, accounting for 74.6% on average (Finding1); The frequencies of change types are significantly similar across most studied systems (Finding 2); A significant number of faulty functions (65.2% on average) are fixed by only one or two change types (Finding 3); Besides, function call statements are very likely to be changed together with assignment statements or branch statements in most studied systems (Finding 4).
- Across domains. The frequencies of change subtypes do not vary substantially, but share significantly similar trends across studied domains (Finding 6), and changes on function-call statements are the most commonly observed changes regardless of the domain of the studied projects (Finding 5).
- Across versions. The frequencies of change subtypes are similar across different versions of the studied system (Finding 7), and the most common subtype pairs (i.e. a pair of change subtypes which occur together in the same function) are always the same across studied versions (Finding 8).

The above findings provide a comprehensive view on the characteristics of change types, improve the visualization of how bugs were repaired, and gain deeper insights into the nature of bugs from source code perspective. We believe these results are valuable to guide both software development and future researches in this direction.

Paper outline: The remainder of the paper is organized as follows. Section 2 describes the fundamental concepts, our proposed change classification scheme, and the classification methods for special cases. Section 3 presents the experimental methods of our study, including the research questions, studied systems, and our automatic classification tool. Section 4 reports the evaluation results of our automatic tool and presents experimental results in detail for each research question. Section 5 analyzes the threats to validity of our study, followed by the introduction of related work in Section 6. Finally, in Section 7, we give the conclusions and outline the directions for future work.

2. Change types

In this section, we first introduce the concepts used in our study. Then, we propose a new classification scheme and introduce each change type in detail. Finally, we illustrate the classification methods for special cases.

2.1. Concepts

For better understanding, it is necessary to first define the special concepts used in our study. It is noteworthy that, in this study, we use “bug”, “defect”, and “fault” interchangeably, since “faults” and “defects” are synonyms of bugs [6].

- Bug: errors in a software system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.
- Bug version: a released version with bugs.
- Fix version: a subsequent version of the bug version, which is released purely for fixing bugs.
- Faulty function: a function in the bug version which is changed in its corresponding fix version.
- Fixing Code Change: code change between the bug version and fix version, like adding, deleting, modifying, or moving source code. (Note that, the fixing code changes are occasionally abbreviated to changes in this study)
- Change type: the type of a fixing code change.