An Empirical Study of Bugs in Build Process

Xiaoqiong Zhao[1], Xin Xia[1], Pavneet Singh Kochhar[2], David Lo[2], and Shanping Li[1]
[1]College of Computer Science and Technology, Zhejiang University, China
[2]School of Information Systems, Singapore Management University, Singapore
{zhaoxiaoqiong, xxkidd}@zju.edu.cn, {kochharps.2012, davidlo}@smu.edu.sg,
shan@zju.edu.cn

ABSTRACT

Software build process translates source codes into executable programs, packages the programs, generates documents, and distributes products. In this paper, we perform an empirical study to characterize build process bugs. We analyze bugs in build process in 5 open-source systems under Apache namely CXF, Camel, Felix, Struts, and Tuscany. We compare build process bugs and other bugs across 3 different dimensions, i.e., bug severity, bug fix time, and the number of files modified to fix a bug. Our results show that the fraction of build process bugs which are above major severity level is lower than that of other bugs. However, the time effort required to fix a build process bug is around 2.03 times more than that of a non-build process bug, and the number of source files modified to fix a build process bug is around 2.34 times more than that modified for a non-build bug.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

Keywords

Build Process, Bugs, Empirical Study

1. INTRODUCTION

Software build process, which compiles source code into binary code, packages the binary code, executes test cases, runs static analysis, generates documents, and distributes products, is a critical task in software development and maintenance. This automated process enhances product quality, reduces redundant tasks and aids in efficient management of a project. Despite these benefits, build system maintenance increases the cost of software development by 12%-36% [1].

In this paper, we perform an empirical study on build process bugs and their potential impact on software development. Our empirical study provides insights about the importance of build process bugs and differentiates these bugs with other bugs which we refer to as *non-build bugs*. We analyze build process bugs from 5 projects under The Apache Software Foundations including CXF, Camel, Felix, Struts, and Tuscany.

We investigate the following research questions:

1. What are the distributions of severity levels assigned to build process bugs and non-build bugs?
2. How long does it take to fix build process bugs and non-build bugs?
3. How many source files are modified to fix build process bugs and non-build bugs?

Table 1: Size of Analyzed Projects.

Project	LOC	# Files	# Components
CXF	410,809	4,326	25
Camel	466,797	8,272	102
Felix	438,456	4,836	54
Struts2	274,360	1,973	12
Tuscany	237,075	3,006	41

2. METHODOLOGY & BASIC STATISTICS

To collect build process bugs we follow a two step approach: bug report collection and build process bug identification.

Bug Report Collection. In this study, we consider 5 large open source projects each containing more than 100 kLOC. Table 1 shows some statistics of these 5 projects. We collect bug reports from the JIRA bug tracking systems of these projects. In total, we collect more than 10,286 bug reports from the 5 projects. The number of bug reports collected for each project is shown in Table 2.

Table 2: Collected Bug Reports.

Projects	# Bug Reports
CXF	2,402
Camel	1,168
Felix	1,213
Structs	3,642
Tuscany	1,861
Total	**10,286**

Build Process Bug Identification. Identifying build process bugs from more than 10,000 bug reports is an arduous task. In our identification step, we first semi-automatically reduce the number of bug reports that we need to manually analyze to identify build process bugs. Next, we manually analyze these bug reports and categorize them as build process bugs or non-build bugs.

Build tools such as Ant and Maven use build files i.e., build.xml, and pom.xml, respectively, to automate the software build processes. These build files contain information about different modules and the relationships between source code, packages, and modules. The modules closely correspond to the components in bug reports, and not all components appear in the build files. For example,

SAC'14 March 24-28, 2014, Gyeongju, Korea.

http://dx.doi.org/10.1145/2554850.2555142

for CXF, component "Documentation" does not appear in the build file. Bug reports related to documentation are unlikely to be build process bugs. For CXF, only 8 modules are included in the build file, however there are 25 different components in the bug reports. So, we reduce the search space by only considering bug reports related to the modules present in the build files. The name of a module (in the build file) and the name of its corresponding component (in the bug reports) might not be exactly the same. For example, for CXF, the module "tools" corresponds to component "Tooling". Also, the module "services" corresponds to components "Services" and "Services Model". We manually map the names of the modules to the names of their corresponding components by analyzing the lexical and semantic similarities of their names. After we filter out bug reports that are not related to the modules in the build files, we are left with 3,205 bug reports (642, 836, 811, 343, and 573 for CXF, Camel, Felix, Struts2, and Tuscany, respectively).

Next, we keep bug reports with status "closed" and "fixed". Then, we search these reports by using keywords like "build", "xml", "pom", etc. For all the bug reports which match one or more keywords, we manually decide if it is a build process bug or a non-build bug. To decide whether a bug report is a build process bug or a non-build bug, we investigate the bug report and the commits that fix the bug. JIRA links a bug report to the commits (made to a version control system) that address it by a unique bug identifier which is automatically inserted to the logs of the relevant commits. Bissyande et al. have shown that these links are reliable [2]. We investigate the files that are changed by the commits and decide whether a bug report is a build process bug or not. For a bug report, if we judge it to be a build process bug, we also consider whether there would be more keywords, in the newly identified build process bug, which can be used to search other build process bugs. We iterate these steps until there is no new bug report which matches the keywords. At the end of this process, we identified 121 build process bugs (27, 33, 12, 11, and 38 for CXF, Camel, Felix, Struts2, and Tuscany, respectively).

3. EMPIRICAL STUDY RESULTS

RQ1: Severity Distribution of Build Process Bugs. To answer this question, we investigate the severity distributions for the build process bugs and non-build bugs. Severity represents the importance of that bug as compared to other bugs. The higher the severity level, the more attention a developer would pay to that bug. There are five severity levels that a bug reporter can assign in JIRA: Blocker, Critical, Major, Minor, and Trivial. Blocker represents the most severe bugs while trivial represents the least severe bugs. In JIRA, the default severity for a bug report is Major. If a bug report's severity is Critical or Blocker, it means the bug could cause fatal system problems (e.g., system crash, data corruption, etc) and developers would fix these kinds of bugs first.

Table 3: Number of Build and Non-Build Bugs of Different Severity Levels.

Projects	Types	Blocker	Critical	Major	Minor	Trivial
CXF	Build.	0	2	23	2	0
	Non-build	13	26	505	66	5
Camel	Build.	0	3	24	6	0
	Non-build	2	31	588	170	12
Felix	Build.	0	0	9	2	1
	Non-build	9	16	636	122	16
Structs	Build.	0	0	9	2	0
	Non-build	9	92	188	30	13
Tuscany	Build.	1	1	26	10	0
	Non-build	29	33	406	64	3

Table 3 presents the severity distribution for the build process bugs and other bugs in CXF, Camel, Felix, Structs, and Tuscany. We notice that the number of bug reports with severity above major are small for both build process bugs and non-build bugs. For example, in CXF, there are 0 and 2 build process bugs with severity blocker and critical, respectively. Also, there are only 13 and 26 non-build bugs with severity blocker and critical, respectively. Among the 121 build process bugs, only 7 bugs (5.8%) are assigned severity critical or blocker. On the other hand, among the 3,084 non-build bugs, 260 bugs (8.4%) are assigned severity critical or blocker. Thus, the fraction of critical and blocker bugs in build process bugs is lower than that in non-build bugs.

We use the Pearson's chi-squared test [3] to check whether the distribution of severity levels of build process bugs is different from that of non-build bugs. The test returns a p-value of less than $2.2e^{-16}$. Thus, considering a significance level of 0.05, there is a significant difference in the *proportions* of bug reports of various severity levels for build process bugs and non-build bugs.

> *The fraction of build process bugs whose severity levels are critical or blocker is lower than that of non-build bugs. The distribution of severity levels for build process bugs is different from that for non-build bugs.*

RQ2: Bug Fix Time. In this research question, we analyse the time to fix build and non-build bugs. We record the time duration as the difference between the bug reporting timestamp and bug closing (i.e., the bug report is assigned status "closed") timestamp. Although not a perfect measure, this time duration is related to the effort required to fix a bug [4, 5]. Table 4 shows the mean fix time for build process bugs and non-build bugs in CXF, Camel, Felix, Structs, and Tuscany, respectively.

Table 4: Mean Fix Time (Hours) for Build Process Bugs and Non-Build Bugs.

Projects	Build Process Bugs	Non-Build Bugs
CXF	1,587	898
Camel	430	228
Felix	705	681
Structs	6,014	2,293
Tuscany	1,146	766
Average	**1,976**	**973**

We notice the mean fix time for build process bugs is much longer than that of other bugs, i.e., on average across the 5 projects, the mean fix time for build process bugs is 1,976 hours, while that of non-build bugs is 973 hours. To fix a bug, the time effort for a build process bug is around 2.03 times the time required to fix a non-build bug. Moreover, the mean fix time of bug reports whose severity levels are either Major, Critical, or Blocker is much longer for build process bugs than for other bugs, i.e., on average across the 5 projects, the mean fix time of build process bugs whose severity levels are either Major, Critical, or Blocker is 2,175 hours, while that of other bugs is 1,043 hours.

We perform a Mann-Whitney-Wilcoxon (MWW) test [6] to compare the bug fix time of build and non-build bugs and find that the difference is statistically significant with a p-value of 0.004345. Thus, we can conclude that build process bugs take more time to fix than non-build bugs.

> *The time required to fix a build process bug is almost double that of a non-build bug.*

RQ3: Number of Modified Source Files. Here, we investigate the number of modified source files for build process bugs and non-build bugs. Table 5 presents the number of source files modified to fix build process bugs and non-build bugs in CXF, Camel, Felix, Structs, and Tuscany, respectively.

We notice that the number of source files modified for build pro-

cess bugs is much more than that of non-build bugs, i.e., on average across the 5 projects, the number of source files modified for build process bugs is 14.9, while that of other bugs is 6.36. To fix a bug, the number of source files modified for a build process bug is around 2.34 times for that of a non-build bug. Moreover, the number of source files modified for bug reports whose severity levels are either Major, Critical, or Blocker is much more for build process bugs than for other bugs, i.e., on average across the 5 projects, the number of source files modified to fix bug process bugs whose severity levels are either Major, Critical, or Blocker is 16.65, while that of other bugs is 6.79.

Table 5: Number of Source Files Modified for Build Process Bugs and Non-Build Bugs.

Projects	Build Process Bugs	Non-Build Bugs
CXF	19.89	9.55
Camel	16.21	7.03
Felix	11.08	3.54
Structs	14.82	4.10
Tuscany	12.50	7.56
Average	**14.90**	**6.36**

We perform a Mann-Whitney-Wilcoxon (MWW) test to compare the number of files modified to fix build and non-build bugs and find that the difference is statistically significant with a p-value of less than $2.2e^{-16}$. Thus, we can conclude that the number of source files modified to fix build process bugs is significantly larger than that of other bugs.

The number of source files modified to fix a build process bug is more than twice the number required to fix a non-build bug.

Threats to Validity. Threats to internal validity relates to experimenter bias and errors. We use a heuristic method to identify build process bugs. Some build process bugs might be missed. We manually examine 3,205 bug reports. We might make some mistakes in categorizing a bug report as build process bugs or not. Threats to external validity refers to the generalizability of our findings. In this preliminary study, we only analyze 10,286 bugs (121 build process bugs and 10,165 non-build bugs) in 5 software systems. In the future, we plan to reduce this threat to external validity by analyzing more bugs in more software systems.

4. RELATED WORK

Seaman et al. investigate bugs in various NASA projects and categorize bugs depending on the location of the defects: requirement documents, code, and test plans [7]. Li et al. categorize bugs in Mozilla and Apache Web Server based on their root causes, impacts, and the affected software components [8]. Pan et al. perform an empirical study on bug fixing performed in various projects written in Java [9]. Zaman et al. perform an empirical study on security and performance bugs in Firefox [10]. Chou et al. investigate the distribution of bugs in Linux and OpenBSD kernels [11]. Thung et al. perform an empirical study of bugs in 3 machine learning systems, i.e., Mahout, Lucene, and OpenNLP [12]. Xia et al. categorize bugs in Make, Ant, CMake, Maven, Scons, and QMake, and they find that most of the bugs belong to external interface and logic categories [13]. Different from the above studies, in this work, we investigate build process bugs and analyze the differences between these bugs and non-build bugs.

5. CONCLUSION AND FUTURE WORK

In this paper, we investigate build process bugs. To identify build process bugs, we employ a semi-automated heuristic method which analyzes the build files (i.e., build.xml for ant, pom.xml for maven) to detect build related components. Next, we search the bug reports for these components by using relevant keywords, and for each retrieved bug report, we read the description and comments, and check the modified source files to identify whether it is a build process bug or not. We find 121 build process bugs in 5 open-source projects (CXF, Camel, Felix, Struts, and Tuscany). Based on the 121 build process bugs, we perform an empirical study, and we compare build process bugs and other bugs across 3 different dimensions, i.e., bug severity, bug fix time, and the number of files modified to fix a bug. Our preliminary experiment results show that the fractions of build process bugs whose severity are above major is lower than that of other bugs. However, the time effort to fix a build process bug is around 2.03 times more than that of a non-build bug, and the number of source files modified to fix a build process bug is around 2.34 times more than that of a non-build bug.

In the future, we plan to investigate more build process bugs in more projects. We also plan to develop an automated method to identify build process bugs.

Acknowledgment This research is sponsored in part by NSFC Program (No.61103032) and National Key Technology R&D Program of the Ministry of Science and Technology of China (No2013BAH 01B03).

6. REFERENCES

[1] G. K. Kumfert and T. G. W. Epperly, "Software in the DOE: The Hidden Overhead of "The Build"," Tech. Rep., 2002.

[2] T. F. Bissyandé, F. Thung, S. Wang, D. Lo, L. Jiang, and L. Réveillère, "Empirical evaluation of bug linking," in *CSMR*, 2013.

[3] K. Pearson, "On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that can be reasonably supposed to have arisen from random sampling," *Philosophical Magazine*, 1900.

[4] H. Zhang, L. Gong, and S. Versteeg, "Predicting bug-fixing time: an empirical study of commercial software projects," in *ICSE*, 2013.

[5] H. Hosseini, R. Nguyen, and M. W. Godfrey, "A market-based bug allocation mechanism using predictive bug lifetimes," in *CSMR*, 2012.

[6] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The Annals of Mathematical Statistics*, 1947.

[7] C. B. Seaman, F. Shull, M. Regardie, D. Elbert, R. L. Feldmann, Y. Guo, and S. Godfrey, "Defect categorization: making use of a decade of widely varying historical data," in *ESEM*, 2008.

[8] Z. Li, L. Tan, X. Wang, S. Lu, Y. Zhou, and C. Zhai, "Have things changed now? An empirical study of bug characteristics in modern open source software," in *ASID*, 2006.

[9] K. Pan, S. Kim, and E. J. Whitehead Jr, "Toward an understanding of bug fix patterns," *Empirical Software Engineering*, 2009.

[10] S. Zaman, B. Adams, and A. E. Hassan, "Security versus performance bugs: a case study on firefox," in *MSR*, 2011.

[11] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, "An empirical study of operating systems errors," in *SOSP*, 2001.

[12] F. Thung, S. Wang, D. Lo, and L. Jiang, "An empirical study of bugs in machine learning systems," in *ISSRE*, 2012.

[13] X. Xia, X. Zhou, D. Lo, and X. Zhao, "An empirical study of bugs in software build systems," in *QSIC*, 2013, pp. 200–203.