2013 13th International Conference on Quality Software

An Empirical Study of Bugs in Software Build Systems

Xin Xia*‡, Xiaozhen Zhou*, David Lo†, and Xiaoqiong Zhao*
*College of Computer Science and Technology, Zhejiang University
†School of Information Systems, Singapore Management University
{xxkidd, zxztc}@zju.edu.cn, davidlo@smu.edu.sg, zhaoxiaoqiong@zju.edu.cn

***Abstract*—Build system converts source code, libraries and other data into executable programs by orchestrating the execution of compilers and other tools. The whole building process is managed by a *software build system*, such as Make, Ant, CMake, Maven, Scons, and QMake. The reliability of software build systems would affect the reliability of the build process. In this paper, we perform an empirical study on bugs in software build systems. We analyze four software build systems, Ant, Maven, CMake and QMake, which are four typical and widely-used software build systems, and can be used to build Java, C, C++ systems. We investigate their bug database and code repositories, randomly sample a set of bug reports and their fixes (800 bugs reports totally, and 199, 250, 200, and 151 bug reports for Ant, Maven, CMake and QMake, respectively), and manually assign them into various categories. We find that 21.35% of the bugs belong to the external interface category, 18.23% of the bugs belong to the logic category, and 12.86% of the bugs belong to the configuration category. We also investigate the relationship between bug categories and bug severities.**

Keywords*—*Software Build System, Bug Category, Empirical Study

I. Introduction

The most common goal of a build system is to convert source code, libraries and other data into executable programs by orchestrating the execution of compilers and other tools. In addition, build systems also support the packaging of web-based application, the generation of software product documentation, the automatic static analysis of source code, and other related activities [1]. A prior research study has shown that build system maintenance could add 12%-36% more costs on software development [2].

The whole build process use various systems and tools: *version-control tools*, which store the source code and ensure concurrent development for developers; *compilation tools*, which convert input source code into object code or executable programs; *software build systems*, which collect sufficient information about the relationship between source files and object files, and use necessary compilation tools to produce the final build output (e.g., executable programs, software package, documentation, static analysis results). *Software build systems* play the most important roles in building systems, since they orchestrate the entire build process, and control the final build output. There are various software build systems, such as Make, Ant, CMake, Maven, Scons, and QMake.

‡The work was done while the author was visiting Singapore Management University.

A number of studies have investigated bugs and their fixes in various systems [3], [4], [5], these studies provide guide for triaging bug reports, detecting duplicated bug reports, designing bug location tools, reduce testing and maintenance costs, and helping to improve development efficiency. However, to our best knowledge, none of these studies focus on bugs in software build systems. One significant feature of software build systems is that they should work on various platforms, i.e., various operating systems (e.g., Windows, Linux), various development environments (e.g., Eclipse, Visual Studio), and various programming languages (e.g., C, C++, Java, C#). Thus, analyzing bugs and their fixes in software build systems deserves a special consideration.

In this study, we analyze four software build systems:

1) Apache Ant[1], one of the most popular build systems for Java-based projects.
2) Apache Maven[2], a software build automation and comprehension tool used primarily for Java projects.
3) CMake[3], a software build automation which translates a high-level build description into a lower-level description which can be used by other build system, such as GNU Make.
4) QMake[4], a part of the QT development environment, which is similar to CMake.

To investigate bugs that appear in the four systems, we analyze their bugs and code repositories. We collect bug reports with status “closed”, and manually check their fixes. For Apache Ant, it uses Bugzilla to track all the bugs; we manually check the commit logs in CVS to retrieve the fixes related to a bug. For Apache Maven and QMake, they use JIRA to track bugs, and JIRA contains links from bug reports to a list of changes in the source control repositories that fix those bugs. For CMake, it uses MantisBT to track bugs, and uses Github[5] to manage source code, and we notice developers post links of source code changes related to the corresponding bugs in Github.

In this paper, we aim to answer a number of research questions: How often a bug appears in software build systems? How much bugs per kLOC? What categories of bugs appear in software build systems? What are the severity distribution for each category of bugs? To answer the above questions, we

[1]http://ant.apache.org/
[2]http://maven.apache.org/
[3]http://www.cmake.org/
[4]http://qt-project.org/doc/qt-4.8/qmake-manual.html
[5]https://github.com/

TABLE I. STATISTICS INFORMATION OF CLOSED BUGS IN SOFTWARE BUILD SYSTEMS

Projects	Version	Lines of Code	No. Files	Bug Count	Duration
Ant	1.8.4	254,431	1,167	2,567	5.18 years
Maven	3.05	51,651	346	2,945	10.66 years
CMake	2.8.10	406,715	1,104	5,192	9.72 years
QMake	5.0.1	33,583	53	844	6.96 years

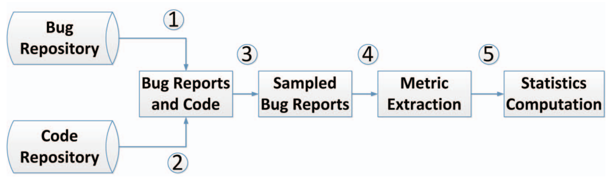

Fig. 1. The Whole Process of the Empirical Study

perform both manual and automated analysis on a randomly sampled set of "closed" bug reports and their corresponding bug fixing commits. We find that 21.35% of the bugs belong to the external interface category, 18.23% of the bugs belong to the logic category, and 12.86% of the bugs belong to the configuration category.

II. METHODOLOGY

We focus on only closed bugs from the bug repositories, as bug reports that are not closed may not be bugs or have no fixes or enough information for our analysis yet. Table I shows version, lines of source code, number of source files, and the numbers of closed bugs for Ant, Maven, CMake, and QMake, respectively. We also show the durations (in years) between the first and last bugs we collected in column Duration.

Figure 1 presents the whole process of our empirical study. We first download the bug reports from bug repositories and the source code from code repositories of the four software build systems (Steps 1 and 2). Next, we randomly select a set of bug reports from the whole bug report collection for each system (Step 3). Then, we extract some information, identity the fixes for each bug report in the four systems, and assign bug reports into different categories (Step 4). Finally, we compute some statistics based on the collected information extracted in Step 4 (Step 5). We explain more on the information extraction and statistics computation (i.e., Steps 4 and 5, respectively) in the following paragraphs.

1) Information Extraction: To get bug fixes we check both bug and code repositories. For Apache Ant, since it doesn't contain source code change information in bug reports, we check its commit log in CVS to identity the fixes (bugs whose fixes can't be identified from the commit logs are discarded). For Apache Maven, CMake, and QMake, we find the list of source code changes for each closed bug report either in the comment text (CMake) or in the source section (Maven and QMake).

We assign bug reports to several categories manually. We make use of bug description, bug comment textual information, and also bug fixes to infer the bug category. We use the set of categories proposed by Seaman et al. in [6], and we extend it by adding 1 category (i.e., configuration). Table II presents the bug categories and their description.

TABLE II. BUG CATEGORIES FOR SOFTWARE BUILD SYSTEMS

Category	Definition
algorithm/method	The implementation of an algorithm/method works in an unexperted way.
assignment/ initialization	A variable or data item is assigned a wrong value or not properly used.
checking	Missing necessary checks for potential error conditions, or an error response specified for error conditions.
data	Wrong usage of data structure, point, and type conversions.
logic	Incorrect logical expression in condition statements (e.g., if, case, and loop blocks).
non-functional defects	Failure to meet non-functional requirements, such as defines improper variable or method, implement non-compliance method with standard documentation.
timing / optimization	Error related to times, concurrency or performance issues, e.g., slow complication, memory leak, etc.
internal interface	Errors in interfaces between different component in the same system, such as incorrect operation of files or database, and errors of inheritance. For software build systems, it also refers to errors of build system dependency graph, such as generating wrong dependency, losing dependency, etc
external interface	Errors of user interface (including usability issues). For software build systems, it also refers to errors of the usage of build system tools in different platforms, such as various operation systems (e.g., Windows, Linux), various development environments (e.g., Eclipse, Visual Studio), various program languages (e.g., C, C++, Java, C#).
configuration	Error in non-code files (e.g., configuration files) that cause error in functionality.
others	Other bugs not fall into one the above categories.

TABLE III. BUG DENSITIES IN FOUR SOFTWARE BUILD SYSTEMS

Projects	# Bug Per kLOC	# Bug Per File	# Bug Per Year
Ant	10.09 bugs/kLOC	2.20 bugs/file	495.56 bugs/year
Maven	57.02 bugs/kLOC	8.51 bugs/file	276.27 bugs/year
CMake	12.77 bugs/kLOC	4.70 bugs/file	534.16 bugs/year
QMake	25.13 bugs/kLOC	15.92 bugs/file	121.26 bugs/year

2) Statistics Computation: We compute various statistics for each bug category and for all the bug reports we investigate in the four software build systems. These statistics are then used to answer various research questions in Section III-A. We investigate the relationships between bug categories and bug severities, bug fixing time, and number of bug comments by using these statistics.

III. EMPIRICAL STUDY

In this section, we present the research questions and their answers of our empirical study.

A. Research Questions

We are interested in the following research questions:

RQ1 How often bugs appear in software build systems?
RQ2 What are the categories of bugs appearing in software build systems?
RQ3 What are the severity distribution of the various categories of bugs?

B. RQ1: Bug Densities

We present the bug densities of Ant, Maven, CMake, and QMake in Table III. We found that Maven has the highest average number of bugs per kLOC (57.02 bugs/kLOC), followed by QMake (25.13 bugs/kLOC), CMake (12.77 bugs/kLOC),

TABLE IV. BUG CATEGORIES IN FOUR SOFTWARE BUILD SYSTEMS

Category	Number	Percentage
algorithm/method	36	4.49%
assignment/ initialization	67	8.36%
checking	61	7.62
data	100	12.48%
logic	146	18.23%
non-functional defects	12	1.50%
timing / optimization	13	1.62%
internal interface	74	9.24%
external interface	171	21.35%
configuration	103	12.86%
others	17	2.12%

and Ant (10.09 bugs/kLOC). These numbers indicate that that for every line of code, developers of Maven need to fix more bugs than the others.

We also report the average number of bugs per source file, and year in the last two columns of Table III. QMake is a component of QT toolkit, which only has 53 C/C++ source files and 844 bugs reported in QT bug tracking system, but it contains the highest number of bugs per source file (15.92 bugs/file), followed by Maven (8.51 bugs/file), CMake (4.70 bugs/file), and Ant (2.20 bugs/file). Moreover, Maven and CMake have received bug reports for a long period of time - 10.66 and 9.72 years respectively. CMake has the highest average number of bugs per year (534.16 bugs/year), followed by Ant (495.56 bugs/year), Maven (276.27 bugs/year), and QMake (121.26 bugs/year).

C. RQ2: Bug Categories

We randomly sample 800 bug reports from the four build tool systems. There bugs are then manually assigned into different categories. The distribution of bugs based on the 11 categories is presented in IV. We notice that most bugs are categorized as external interface (21.35%), followed by logic (18.23%), and then followed by configuration (12.86%). There are only 2.12% of bugs that fall into the category others. The small proportion of bugs in the category others indicates that the remaining 10 categories are sufficient to cover most of bugs for software build systems.

External interface category corresponds to bugs of user interface and those related to usability issues. Software build systems need to work for various systems, and on various platforms, i.e., various operating systems (e.g., Windows, Linux), various development environments (e.g., Eclipse, Visual Studio), and various programing languages (e.g., C, C++, Java, C#). Also as users need to use these systems often, they would pay attention to build system usability and user interface. These explain why bugs in external interface category are the most.

Logic category corresponds to bugs of incorrect expression appearing in conditional statements. For software build systems, since we need to make them platform-independent, we have to consider different conditions, such as the variable assignment in different platform conditions, which make the logic bugs appear more than many other bug categories. To identify logic bugs, we need to check the source code modification logs.

Configuration category corresponds to bugs in non-code files (e.g., configuration files). The whole build process can be simply viewed as reading from a configuration file (e.g., Makefile, build.xml), and building the system according to the command in the configuration file. For some software build systems, such as CMake and QMake, they are based on low-level build systems such as make, thus they need to parse their configuration files and generate low-level configuration files. This makes configuration bugs appear more than many other bug categories.

D. RQ3: Bug Severity

Next, we investigate the relationship between bug category and bug severity. We study the same five severity levels[6] as [3], i.e., Block, Critical, Major, Minor, and Trivial. Block is the most severe category while Trivial is the least severe category. Table V presents the relationship between bug category and bug severity in Ant, Maven, CMake and QMake.

We notice that major and minor severities dominate all the bug categories. It is notable that in JIRA (Maven, QMake), the default severity level when a user creates a new bug report is major, while in Bugzilla (Ant), the default severity level is normal, and in MantisBT (CMake), the default severity level is minor. This default setting maybe the reason that major and minor severities take the majority of bugs in all of the categories; Users might not be able to distinguish the meaning of different severity levels well, and they may simply use the default severity level [7].

Following the definition of the various severity levels, block bug refers to a bug that causes system crash, data corruption, irreparable harm, etc, and critical bug refers to a bug that affects an important function and it has no reasonable workaround. Analyzing block and critical bugs can provide insight towards developing a more robust application. From Table V, we notice all the bug categories except algorithm/method and timing/optimization contain bugs of block or critical severity levels. For external interface and logic categories, they contain the most of block and critical bugs, i.e., with 50 (29.24%) and 59 (40.41%) bugs, respectively.

IV. RELATED WORK

McIntosh et al. investigate version histories of one proprietary and nine open source projects [8]. They conclude that build maintenance incurs up to a 27% overhead on source code development and a 44% overhead on test development. Adams et al. [9] analyze the changes to the Linux kernel build system from its its inception up to version 2.6 using MAKAO [10]. Suvorov et al. provide an empirical study for build system migration; they analyze two cases: KDE and Linux kernel [11]. Neitsch et al. perform an empirical study of the build systems for programs that are developed in multiple programming languages [12]. Tu and Godfrey perform case studies on build-time software architecture, and introduce the "code robot" architectural style [13].

[6]We notice in JIRA (Maven and QMake), the word priority is used instead of severity. In Bugzilla (Ant), there are 7 severity levels (blocker, critical, major, normal, minor, trivial, and enhancement). And in MantisBT (CMake), there are 8 severity levels (blocker, crash, major, minor, tweak, text, trivial, and feature). To make the severity level consistent with a previous study [3], we assign them into 5 severity levels, i.e., we assign normal to minor, enhancement to trivial for Ant, and we assign crash to critical, and tweak, text, feature to trivial for CMake.

TABLE V. RELATIONSHIP BETWEEN BUG CATEGORY AND BUG SEVERITY IN FOUR SOFTWARE BUILD SYSTEMS

Category	Severity	Number	Proportion	Category	Severity	Number	Proportion
algorithm/method	Block	0	0.00%	timing/optimization	Block	0	0.00%
	Critical	0	0.00%		Critical	0	0.00%
	Major	8	22.22%		Major	2	15.38%
	Minor	28	77.78%		Minor	10	76.92%
	Trivial	0	0.00%		Trivial	1	7.69%
assignment/initialization	Block	0	0.00%	internal interface	Block	9	12.16%
	Critical	14	20.90%		Critical	10	13.51%
	Major	16	23.88%		Major	36	48.65%
	Minor	36	53.73%		Minor	14	18.92%
	Trivial	1	1.49%		Trivial	5	6.76%
checking	Block	1	1.64%	external interface	Block	8	4.68%
	Critical	8	13.11%		Critical	42	24.56%
	Major	36	59.02%		Major	53	30.99%
	Minor	15	24.59%		Minor	57	33.33%
	Trivial	1	1.64%		Trivial	11	6.43%
data	Block	0	0.00%	configuration	Block	3	2.91%
	Critical	13	13.00%		Critical	17	16.50%
	Major	28	28.00%		Major	25	24.27%
	Minor	47	47.00%		Minor	48	46.60%
	Trivial	12	12.00%		Trivial	10	9.71%
logic	Block	3	2.05%	others	Block	1	5.88%
	Critical	56	38.36%		Critical	4	23.53%
	Major	68	46.58%		Major	7	41.18%
	Minor	14	9.59%		Minor	5	29.41%
	Trivial	5	3.42%		Trivial	0	0.00%
non-functional defects	Block	4	33.33%				
	Critical	2	16.67%				
	Major	4	33.33%				
	Minor	2	16.67%				
	Trivial	0	0.00%				

There are various empirical studies on bugs and fixes. Seaman et al. make use of NASA historical data by creating model to guide future software development, and propose a set of bug categories [6]. Thung et al. perform an empirical study of bugs in machine learning systems [3]. Chou et al. investigate bugs in operating systems in the Linux and OpenBSD kernels [5]. They analyze the root cause of bugs, bug distribution, bug life cycle, bug clusters, and the difference between operating system bugs and other bugs. Pan el al. investigate bug fix patterns in a number of systems, and categorize the bug fix types based on the syntax of code changes [14].

V. CONCLUSIONS AND FUTURE WORK

To better understand software build systems, we perform an empirical study on bugs in these systems. We analyze four software build systems: Apache Ant, Apache Maven, CMake, and QMake. We first download their bug repositories and code repositories. Next, we randomly pick 800 bugs (199, 250, 200, and 151 bug reports for Ant, Maven, CMake and QMake, respectively), and manually assign them into different categories. We further investigate the relationship between bug categories and bug severities, and we find that among the 800 bug reports, 21.35% of the bugs belong to the external interface category, 18.23% of the bugs belong to logic category, and 12.86% of the bugs belong to configuration category.

In the future, we plan to investigate more software build systems, and analyze more bug reports. We also plan to design an automatic bug categorization tool to assign bugs into their categories.

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 (No2013BAH01B01).

REFERENCES

[1] P. Smith, *Software Build Systems: Principles and Experience*. Addison-Wesley Professional, 2011.

[2] G. Epperly, “Software in the doe: The hidden overhead of the build,” 2002.

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

[4] S. Zaman, B. Adams, and A. E. Hassan, “Security versus performance bugs: a case study on firefox,” in *MSR*, 2011, pp. 93–102.

[5] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, *An empirical study of operating systems errors*. ACM, 2001, vol. 35, no. 5.

[6] 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*. ACM, 2008, pp. 149–157.

[7] I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, and G. Robles, “Towards a simplification of the bug report form in eclipse,” in *MSR*. ACM, 2008, pp. 145–148.

[8] S. McIntosh, B. Adams, T. Nguyen, Y. Kamei, and A. Hassan, “An empirical study of build maintenance effort,” in *ICSE*. IEEE, 2011, pp. 141–150.

[9] B. Adams, K. De Schutter, H. Tromp, and W. De Meuter, “The evolution of the linux build system,” *Electronic Communications of the EASST*, vol. 8, 2008.

[10] B. Adams, H. Tromp, K. De Schutter, and W. De Meuter, “Design recovery and maintenance of build systems,” in *ICSM*. IEEE, 2007, pp. 114–123.

[11] R. Suvorov, M. Nagappan, A. Hassan, Y. Zou, and B. Adams, “An empirical study of build system migrations in practice: Case studies on kde and the linux kernel,” in *ICSM*. IEEE, 2012.

[12] A. Neitsch, K. Wong, and M. Godfrey, “Build system issues in multilanguage software,” in *ICSM*. IEEE, 2012.

[13] Q. Tu and M. Godfrey, “The build-time software architecture view,” in *ICSM*. IEEE, 2001, pp. 398–407.

[14] K. Pan, S. Kim, and E. J. Whitehead Jr, “Toward an understanding of bug fix patterns,” *Empirical Software Engineering*, vol. 14, no. 3, pp. 286–315, 2009.