An Empirical Investigation on the Relationship Between Bug Severity and Bug Fixing Change Complexity

Zengyang Li[1], Dengwei Li[1], Peng Liang[2,*], Ran Mo[1]
[1] School of Computer Science & Hubei Provincial Key Laboratory of Artificial Intelligence and Smart Learning, Central China Normal University, Wuhan, China
[2] School of Computer Science, Wuhan University, Wuhan, China
zengyangli@mail.ccnu.edu.cn, 762396001@qq.com, liangp@whu.edu.cn, moran@mail.ccnu.edu.cn

***Abstract*—Fixing bugs requires changing source code in most cases. The complexity of code changes for fixing bugs has an important impact on release planning. This work intends to investigate whether there are significant differences between bugs with different severity levels with respect to the complexity of code changes for fixing the bugs. We performed a case study on 13 Apache open source software (OSS) projects using commit records and bug reports. The study results show that (1) for bugs of high severity levels, there is no significant difference on the complexity of code change for fixing bugs of different severity levels for most projects, while (2) for bugs of low severity levels, fixing bugs of a higher severity level needs significantly more complex code change than fixing bugs of a lower severity level for most projects. These findings provide useful insights for effort estimation and release planning of OSS development.**

Keywords-bug severity; code change complexity; commit records

I. Introduction

Bugs of a software project are managed in an issue tracking system, in which the severity of a bug can be indicated by the development team members or external reporters. In practice, developers use severity levels, such as Blocker, Critical, Major, Minor, and Trivial in JIRA, to prioritize the urgency of bugs and to estimate influence and impact of bugs [1]. The bug severity data play an important role in release planning and task assignment [2, 3].

The complexity of required code change for fixing a bug also influences release planning in terms of effort estimation [3]. More complex code change is required for fixing a bug means more effort is needed for this bug fixing task. Effort estimation for tasks is a key aspect in release planning [3].

Both the severity of a bug and required effort for fixing the bug should be taken into consideration during release planning of a project. A natural question arises: *is bug severity in line with actual code change complexity*? The answer to this question will provide meaningful insights for effort estimation of project development.

To investigate whether bug severity is in line with complexity of actual code change, we performed a case study on 13 non-trivial Apache open source software (OSS) projects. Data on bugs were exported from JIRA – an issue tracking system deployed by Apache Software Foundation. Data on the complexity of code change for fixing bugs can be obtained by analyzing the commit records extracted from the code repositories of the OSS projects.

Our main findings are as follows: (1) There is no significant difference on the complexity of code change for fixing Blocker and Critical bugs for most projects. The situation is similar for Critical and Major bugs. (2) Code change for fixing Major bugs has a significantly higher complexity than fixing Minor bugs for most selected projects. The situation is similar for Minor and Trivial bugs.

The rest of this paper is organized as follows. Section II reports related work to this study. Section III describes the design of the case study. Section IV presents the results of the case study. Section V discusses the results of the case study and Section VI presents the threats to validity of the results. Section VI concludes this work with future directions.

II. Background and Related Work

This section presents background of this study and related work on bug severity and required code change for fixing bugs.

A. Background

In the JIRA issue tracking platform, issues are classified in multiple types, including Bug, Improvement, New Feature, Task, Sub-task, Test, and Wish. In particular, according to its severity (i.e., priority to be fixed), a bug can be labeled a level, from high to low severity, as Blocker, Critical, Major, Minor, or Trivial. These severity levels are defined as follows, according to JIRA [4].

- **Level 5 – Blocker**: a time-sensitive issue that is hindering a basic function of a project.
- **Level 4 – Critical**: a time-sensitive issue that is disrupting the project, but does not hinder basic functions.
- **Level 3 – Major**: this issue needs attention soon, but is not hindering basic functions. Most requests for new resources fall into this category.

* Corresponding author.
DOI reference number: 10.18293/SEKE2020-160

- **Level 2 – Minor**: this issue needs attention, but is not time-sensitive and does not hinder basic functions.
- **Level 1 – Trivial**: this issue is minimal and has no time constraints.

Besides, in JIRA, the status of a bug can be one of the following: Open, In Progress, Reopen, Resolved, and Closed.

B. *Related Work*

Many studies proposed various methods to predict bug severity automatically. For instance, Roy *et al.* used text mining and machine learning techniques to improve bug severity classification [5]. Lamkanfi *et al.* applied text mining algorithms to analyze descriptions of bug reports for predicting bug severity [6]. Menzies *et al.* proposed to use standard text mining and machine learning techniques to automate severity assessment based on software defect reports [7]. Tian *et al.* used multi-factor analysis to automatically predict bug priority [8]. However, our work is not aimed to predict severity levels of bugs, but to investigate whether bugs of a higher severity level require more complex code change to fix.

A number of studies looked into the delay of bug fixing from the perspective of bug severity and required code change. Zhang *et al.* found that a larger total lines of changed code can delay bug fixing, and bugs of a high severity level were fixed earlier than bugs of a low severity level [9]. Saha *et al.* revealed that bug priority (i.e., severity) has significant impact on the delay of bug fixing [10]. However, the relationship between bug severity and complexity of changed code was not discussed in these study.

Some works on effort estimation took bug severity into consideration. For instance, Weiss *et al.* took the average time and effort of previous bugs with similar severity as an early estimation of required effort for new bugs [11].

III. STUDY DESIGN

In order to investigate the relationship between bug severity and code change complexity, we performed a case study on fifteen Apache OSS projects written in Java. In this section we describe the case study, which was designed and reported according to the guidelines proposed by Runeson and Höst [12].

A. *Objective and Research Questions*

The goal of this case study is to investigate: whether there is a significant difference on the complexity of changed source code for fixing the bugs with different bug severity levels.

In this study, the complexity of changed source code is measured in three dimensions: (i) number of modified lines of code, (ii) number of modified source files, and (iii) number of modified packages. It is convenient to extract such information on a bug by analyzing commit records and bug reports.

Based on the abovementioned goal and considering the three dimensions of the complexity of changed source code, we formulated the following three research questions (RQs):

- **RQ1**: Is there a significant difference between the numbers of **lines of modified code** for fixing bugs with different severity levels?
- **RQ2**: Is there a significant difference between the numbers of **modified source files** for fixing bugs with different severity levels?
- **RQ3**: Is there a significant difference between the numbers of **modified packages** for fixing bugs with different severity levels?

B. *Case and Unit Analysis*

According to [12], case studies can be characterized based on the way they define their cases and units of analysis. This study investigates multiple OSS projects, i.e., cases, and each bug and changed source code for fixing it is a single unit of analysis.

C. *Case Selection*

In this study, we only investigated Apache OSS projects written in Java. For selecting each case (OSS project) included in our study, we apply the following criteria:

(1) Over 70% of the source code of the project is written in Java.
(2) The age of the project is over 5 years.
(3) The number of stars of the project on GitHub is over 500.
(4) The number of revisions of code repository of the project is over 2,000.
(5) The number of bugs of the project is over 1,500.

These selection criteria were set to ensure that the selected cases are non-trivial and the resulting dataset is big enough to be statistically analyzed.

D. *Data Collection*

This section presents the data to be collected and the process for collecting required data.

1) *Data to be Collected*

To answer the RQs formulated in Section III-A, we collected the data items listed in TABLE I, which also provides the mapping between the data items and the target RQ(s).

TABLE I. DATA ITEMS TO BE COLLECTED

#	Data Item	Description	Target RQ
D1	Severity label of a bug	The priority of an issue in JIRA (the issue tracking system used by Apache), i.e., Blocker, Critical, Major, Minor, or Trivial.	RQ1, RQ2, RQ3
D2	Number of lines of modified code for fixing a bug	The number of lines of source code that is changed to fix a bug.	RQ1
D3	Number of modified source files for fixing a bug	The number of source files that is changed to fix a bug.	RQ2
D4	Number of modified packages for fixing a bug	The number of packages (for Java) that is changed to fix a bug.	RQ3

TABLE II. DEMOGRAPHIC INFORMATION OF SELECTED APACHE OSS PROJECTS

#	Name	Age(year)	Java%	#(Star)	#(Revision)	#(Committer)	#(Bug in JIRA)
P1	Accumulo	9	98.6	782	10,431	113	2,250
P2	Activemq	15	95.9	1,694	10,519	97	4,771
P3	Camel	13	98.6	3,129	43,282	626	4,729
P4	CXF	12	98.9	632	15,524	151	5,114
P5	Flink	10	76.7	12,254	20,738	573	5,797
P6	Hadoop	11	92.7	10,177	23,606	291	23,373
P7	Ignite	6	72.2	3,034	26,624	241	5,575
P8	Maven	17	99.5	2,041	10,639	89	3,230
P9	Nifi	6	86.6	1,930	5,675	296	3,136
P10	Pig	13	93.1	597	3,691	28	3,099
P11	Struts	14	91.2	990	5,836	72	2,894
P12	Wicket	16	88.2	505	20,796	89	4,066
P13	Zookeeper	13	73.7	7,730	2,102	94	1,910

TABLE III. AVERAGE NUMBER OF MODIFIED LINES OF CODE, SOURCE FILES, AND PACKAGES PER BUG

Project	#(LOC)/Bug					#(File)/Bug					#(Package)/Bug				
	Blocker	Critical	Major	Minor	Trivial	Blocker	Critical	Major	Minor	Trivial	Blocker	Critical	Major	Minor	Trivial
P1	368.46	229.20	492.77	211.97	333.62	7.67	5.80	8.85	3.87	7.15	4.38	4.38	4.23	2.72	3.76
P2	398.41	178.21	199.16	109.92	49.39	6.03	3.74	4.02	2.35	1.79	2.97	2.40	2.56	1.81	1.45
P3	19.50	202.36	129.65	100.84	62.16	2.17	5.73	4.20	3.54	3.51	1.67	3.31	2.64	2.34	2.43
P4	185.07	85.05	90.98	85.20	124.47	4.37	3.39	3.48	3.59	3.97	3.15	2.39	2.47	2.53	2.67
P5	255.21	187.04	157.21	90.18	34.66	5.16	4.53	3.93	3.03	3.36	3.30	2.70	2.67	2.01	2.50
P6	145.68	122.66	106.65	49.24	23.99	4.44	3.58	3.14	2.50	1.79	3.15	2.66	2.27	1.86	1.50
P7	172.96	282.02	215.47	150.55	29.08	4.64	6.30	5.29	8.97	1.56	3.27	4.07	3.37	5.46	1.44
P8	197.69	97.94	97.08	86.76	42.75	4.14	3.06	3.03	3.48	1.83	3.00	2.48	2.25	2.43	1.67
P9	157.87	240.55	202.16	79.88	7.53	3.80	3.94	4.17	3.31	3.67	2.79	2.53	2.62	2.29	1.64
P10	78.25	53.71	152.08	79.70	30.29	2.75	2.12	3.98	2.52	2.00	2.25	1.86	2.40	1.91	1.32
P11	70.16	266.16	136.43	67.52	175.45	2.61	7.98	2.96	2.74	3.30	2.19	4.14	2.28	2.11	2.45
P12	64.36	84.70	97.46	68.39	27.10	2.27	2.88	2.87	2.59	1.39	1.91	2.10	2.14	1.85	1.24
P13	176.43	185.34	113.30	47.33	16.30	4.12	4.44	2.84	2.15	1.15	2.39	2.29	1.83	1.57	1.15

2) *Data Collection Process*

The process of collecting the data items (listed in TABLE I) for an Apache OSS project includes the following four steps.

Step 1: Export commit records. Commit records of the project were extracted from its Git repository. We developed a simple tool to extract commit records and save them in a text file.

Step 2: Export issues from JIRA. Many Apache OSS projects adopt JIRA (https://issues.apache.org/jira) as their issue tracking system. We manually exported all issues of the project and stored them in a Microsoft Access file. Please note that not all exported issues are bugs and we can get bugs by choosing the issue type ‘Bug’.

Step 3: Parse commit records. If a commit is performed to solve an issue, the committer would explicitly tell the issue ID in the message of the commit record. The changed source files and the changed lines of code can also be identified in the commit record.

Step 4: Extract bugs and corresponding number of lines of modified code, number of modified source files, and number of modified packages. With issue IDs obtained in **Step 3**, we picked up bugs, i.e., issues with issue type ‘Bug’. Then, we calculated the number of lines of modified code, number of modified source files, and number of modified packages for each bug. Please note that, only resolved or closed bugs were included in our dataset. We found that some bugs were resolved in previous revisions but still with the status OPEN. Such bugs actually are in the REOPEN status, which means that these bugs had not been resolved completely and they may involve more lines of source code, source files, and packages.

In this case study, we filtered out abnormal data points. By an abnormal data point, we mean that a bug whose fixing involves either more than 500 modified source files or over 20,000 lines of modified source code. The abnormal data points can affect the validity of conclusions. For instance, in project Accumulo, we found a few bugs each involving hundreds of thousands of modified lines of source code which is generated automatically using the model-driven engineering techniques [13]; if such bugs were not excluded, the average number of modified lines of source code for bugs would increase greatly.

E. *Data Analysis*

To answer the RQs formulated in Section III-A, we need to analyze the collected data on code change history and bug severity. First, we calculated the average number of modified lines of code, modified source files, and modified packages, for each category of bugs (classified according to bug severity). Second, in order to know whether there are significant differences between categories of bugs with respect to the number of modified lines of code, modified source files, and modified packages, we performed Mann-Whitney U tests on the data for each selected OSS project. The test is significant at *p-value* < 0.05, which means that the tested groups have significant difference.

IV. RESULTS

Following the study design, we performed the case study. In this section, we first present the demographic information of the selected cases, i.e., Apache OSS projects. Then, we report the results regarding the research questions formulated in Section III-A.

TABLE IV. DISTRIBUTION OF BUGS OVER DIFFERENT SEVERITY LEVELS

Project	#(Blocker)	#(Critical)	#(Major)	#(Minor)	#(Trival)	Total
P1	136	90	587	232	124	1,169
P2	34	121	1410	290	38	1,893
P3	6	59	2198	927	49	3,239
P4	46	98	1951	426	30	2,551
P5	325	299	856	249	50	1,779
P6	605	801	4174	1115	283	6,978
P7	137	286	1193	112	25	1,753
P8	58	48	569	84	12	771
P9	102	139	835	260	55	1,391
P10	12	42	1114	127	34	1,329
P11	31	49	432	176	20	708
P12	11	50	1323	475	82	1,941
P13	102	87	232	84	20	525

A. *Selected Cases*

Thirteen Apache OSS projects were selected for this case study and their demographic information is shown in TABLE II. The age of the projects is from 6 to 17 years, and 9 out of 13 projects are 10+ years old. All these projects are mainly written in Java, and more than 90% source code of 8 (out of 13) projects are written in Java. Eight out of 13 projects are starred over 1,000 times, and project Flink with 12,254 stars is most starred. Nine out of 13 projects have 10,000+ revisions, demonstrating the vitality of these projects. Each of the selected projects has experienced 2,000+ bugs, and the project Hadoop has the most bugs (23,373).

The average number of modified lines of code, source files, and packages over bugs with different severity levels are presented in TABLE III. The distribution of bugs over different severity levels for the 13 selected OSS project is shown in TABLE IV. Please note that, the total number of bugs in TABLE IV for each project is smaller than the number of bugs in JIRA (as shown in TABLE II). This is because many bugs in JIRA are not recorded in the commit messages of the master branch of the project's code repository, which is also the reason why we selected projects with a relatively large number of bugs (i.e., over 1500 bugs, described in Section III-C).

B. *Results on Modified Lines of Code (RQ1)*

We performed Mann-Whitney U tests to understand whether there is a significant difference between bugs with distinct severity levels with respect to the number of modified lines of code. Results of the tests are shown in TABLE V, where cells with *p-value* < 0.05 are filled in gray. Specifically, a cell filled in gray and with a number in bold denotes that the average number of modified lines of code for higher level bugs is significantly smaller than low level bugs; the remaining cells filled in gray mean that the average number of modified lines of code for higher level bugs is significantly larger than low level bugs.

(1) 3 out of 13 (23.1%) projects have a significant difference (*p-value* < 0.05) between Blocker and Critical bugs, and only in one project (i.e., P5) Blocker bugs have a higher average number of modified lines of code than Critical bugs.

(2) 5 out of 13 (38.5%) projects have a significant difference between Critical and Major bugs, and in 4 out of 13 (30.8%) projects Critical bugs have a higher average number of modified lines of code than Major bugs.

(3) **12 out of 13 (92.3%) projects have a significant difference between Major and Minor bugs, and in all the 12 projects Major bugs have a higher average number of modified lines of code than Minor bugs**.

(4) 10 out of 13 (76.9%) projects have a significant difference between Minor and Trivial bugs, and in 9 out of 13 (69.2%) projects Minor bugs have a higher average number of modified lines of code than Trivial bugs.

TABLE V. *P-VALUES* OF MANN-WHITNEY U TESTS FOR MODIFIED LINES OF CODE BETWEEN BUGS WITH DIFFERENT SEVERITY LEVELS

Project	Blocker& Critical	Critical& Major	Major& Minor	Minor& Trivial
P1	0.113	**0.006**	0.010	**0.002**
P2	0.156	0.780	<0.001	0.002
P3	**0.016**	0.005	<0.001	<0.001
P4	0.406	0.801	<0.001	0.143
P5	<0.001	0.957	<0.001	0.001
P6	0.655	<0.001	<0.001	<0.001
P7	**0.027**	<0.001	<0.001	0.003
P8	0.052	0.162	0.722	0.249
P9	0.724	0.849	<0.001	<0.001
P10	0.617	0.315	0.005	0.001
P11	0.988	0.428	0.008	0.122
P12	0.807	0.224	0.001	<0.001
P13	0.671	<0.001	0.001	0.001

C. *Results on Modified Source Files (RQ2)*

We performed Mann-Whitney U tests to understand whether there is a significant difference between bugs with distinct severity levels with respect to the number of modified source files. Results of the tests are shown in TABLE VI, where cells with *p-value* < 0.05 are filled in gray.

(1) 1 out of 13 (7.7%) project (i.e., P5) has a significant difference between Blocker and Critical bugs, and in this project Blocker bugs have a higher average number of modified files than Critical bugs.

(2) 5 out of 13 (38.5%) projects have a significant difference between Critical and Major bugs, and in 4 out of 13 (30.8%) projects Critical bugs have a higher average number of modified files than Major bugs.

(3) **11 out of 13 (84.6%) projects have a significant difference between Major and Minor bugs, and in 9 out of 13 (69.2%) projects Major bugs have a higher average number of modified files than Minor bugs**.

(4) 8 out of 13 (61.5%) projects have a significant difference between Minor and Trivial bugs, in 7 out of 13 (53.8%) projects Minor bugs have a higher average number of modified files than Trivial bugs.

TABLE VI. *P-VALUES* OF MANN-WHITNEY U TESTS FOR MODIFIED FILES BETWEEN BUGS WITH DIFFERENT SEVERITY LEVELS

Project	Blocker& Critical	Critical& Major	Major& Minor	Minor& Trivial
P1	0.263	0.220	<0.001	0.587
P2	0.096	0.782	<0.001	0.008
P3	0.097	0.007	<0.001	0.009
P4	0.330	0.701	**0.005**	0.056
P5	<0.001	0.111	<0.001	0.954
P6	0.107	<0.001	<0.001	<0.001
P7	0.125	**<0.001**	**0.001**	0.003
P8	0.117	0.982	0.944	0.241
P9	0.865	0.995	0.001	**<0.001**
P10	0.676	0.016	<0.001	0.050
P11	0.898	0.365	0.131	0.315
P12	0.424	0.181	<0.001	<0.001
P13	0.765	0.001	0.011	0.003

D. *Results on Modified Packages (RQ3)*

We performed Mann-Whitney U tests to understand whether there is a significant difference between bugs with distinct severity levels with respect to the number of modified packages. Results of the tests are shown in TABLE VII, where cells with *p-value* < 0.05 are filled in gray.

(1) 1 out of 13 (7.7%) project (i.e., P5) has a significant difference between Blocker and Critical bugs, and in this project Blocker bugs have a higher average number of modified packages than Critical bugs.

(2) 6 out of 13 (46.2%) projects have a significant difference between Critical and Major bugs, and in 5 out of 13 (38.5%) projects Critical bugs have a higher average number of modified packages than Major bugs.

(3) **11 out of 13 (84.6%) projects have a significant difference between Major and Minor bugs, and in 10 out of 13 (76.9%) projects Major bugs have a higher average number of modified packages than Minor bugs**.

(4) 9 out of 13 (69.2%) projects have a significant difference between Minor and Trivial bugs, in 7 out of 13 (53.8%) projects Minor bugs have a higher average number of modified packages than Trivial bugs.

TABLE VII. *P-VALUES* OF MANN-WHITNEY U TESTS FOR MODIFIED PACKAGES BETWEEN BUGS WITH DIFFERENT SEVERITY LEVELS

Project	Blocker& Critical	Critical& Major	Major& Minor	Minor& Trivial
P1	0.191	0.031	<0.001	0.388
P2	0.175	0.294	<0.001	0.006
P3	0.112	0.004	0.012	**0.041**
P4	0.275	0.419	0.114	**0.024**
P5	<0.001	0.084	<0.001	0.976
P6	0.191	<0.001	<0.001	<0.001
P7	0.124	<0.001	**0.019**	0.002
P8	0.215	0.408	0.762	0.479
P9	0.692	0.550	<0.001	<0.001
P10	0.569	**0.070**	0.001	0.003
P11	0.921	0.537	0.042	0.221
P12	0.611	0.268	<0.001	<0.001
P13	0.728	<0.001	0.010	0.026

E. *Summary*

According to the results presented above, there is no significant difference on the complexity of code change for fixing Blocker and Critical bugs in terms of the average number of modified lines of code, source files, and packages for most selected projects. The situation is similar for Critical and Major bugs.

Code change for fixing Major bugs has a significantly higher complexity than fixing Minor bugs in terms of the average number of modified lines of code, source files, and packages for most selected projects. The situation is similar for Minor and Trivial bugs.

V. DISCUSSION

A. *Interpretation of Study Results*

The results have shown that there is no significant difference on the complexity of code change for fixing Blocker and Critical bugs in most selected OSS projects. As defined in Section II-A, both Blocker and Critical bugs are time-sensitive issues and of a high level of urgency, and they have a serious impact on the projects. Thus, Blocker and Critical bugs may have a similar level of change impact when fixing the bugs.

The results have also shown that code change for fixing Major bugs has a significantly higher complexity than Minor bugs in most selected OSS projects. There is a relatively clear boundary between bugs of these two severity levels. The code change impact on the software system for fixing Major bugs is significantly higher than Minor bugs. The complexity of code change for fixing Minor and Trivial bugs can be interpreted in a similar way to Major and Minor bugs.

It is consistent in general for the results of the Mann-Whitney U tests on the significant difference between code change complexity in terms of modified lines of code, source files, and packages, for fixing bugs with different severity levels.

B. Implications

There is no significant difference of average number of modified lines of code, source files, and packages between Blocker and Critical bugs for most selected OSS projects. This implies that Blocker and Critical bugs may have similar change impact and difficulty when fixing them. Hence, when estimating required effort for fixing Blocker and Critical bugs, they can be put in the same category.

Major bugs need to modify a significantly higher average number of lines of code, source files, and packages than Minor bugs for most selected OSS projects. Also, fixing Minor bugs involves code change of higher complexity than Trivial bugs. Hence, for Major, Minor, and Trivial bugs, their severity levels are in line with the complexity of code change for fixing such bugs. Therefore, when estimating needed efforts for fixing Major, Minor, and Trivial bugs, they should be placed in different categories.

VI. Threats to Validity

There are several threats to the validity of the study results. We discuss these threats according to the guidelines in [12]. Please note that internal validity is not discussed, since we do not study causal relationships.

Construct validity. Since a bug is closed or resolved, its severity level (i.e., bug priority in JIRA) was confirmed by the development team member of the project. Thus, we believe that the severity levels of closed or resolved bugs can genuinely reflect the actual severity of the bug. In the data collection process, only the changed code written in Java were included. In some cases, the changed code for fixing a bug may involve source files in other programming languages than Java, which threatens the construct validity. To reduce this threat, we filtered out any bug whose fixing entails non-Java source code.

External validity. Since we collected bugs whose fixing requires changing Java code only, the conclusions of this case study may not be generalized to projects not written in Java. Only 13 projects were used in this case study, more projects are needed to establish more solid conclusions.

Conclusion validity. Only descriptive statistics was used in the calculation of the average number of modified lines of code, source files, and packages. The Mann-Whitney U tests were executed in SPSS, which is a widely-used and well-engineered statistical tool. Thus, we believe that the threats to conclusion validity are minimal.

VII. Conclusions

This work investigates whether there are significant differences between bugs of different severity levels with respect to the complexity of code changes for fixing the bugs. We conducted a case study on 13 Apache OSS projects. Based on the study results, we obtain the following findings:

- In most (>=10/13, 76.9%) projects Blocker (Level 5) and Critical (Level 4) bugs have no significant difference on complexity of code change.
- In most (>=7/13, 53.8%) projects Critical (Level 4) and Major (Level 3) bugs have no significant difference on complexity of code change.
- In most (>=10/13, 76.9%) projects Major (Level 3) bugs have a significantly higher complexity of code change than Minor (Level 2) bugs.
- In most (>=8/13, 61.5%) projects Minor (Level 2) bugs have a significantly higher complexity of code change than Trivial (Level 1) bugs.

Based on the findings of this work, in the next step, we plan to include more software projects (from both commercial and open source) to replicate the case study in this work, in order to establish a more solid foundation for the findings in this work.

Acknowlegments

This work is supported by the National Natural Science Foundation of China under the grant Nos. 61702377 and 61773175, the Fundamental Research Funds for the Central Universities under the grant No. CCNU19TD003, and IBO Technology (Shenzhen) Co., Ltd., China.

References

[1] Y. Tian, N. Ali, D. Lo, and A. E. Hassan, "On the unreliability of bug severity data," *Empirical Software Engineering,* vol. 21, no. 6, pp. 2298-2323, 2016.

[2] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *Proceedings of the 19th Working Conference on Reverse Engineering (WCRE'12)*, 2012, pp. 215-224.

[3] M. Svahnberg, T. Gorschek, R. Feldt, R. Torkar, S. B. Saleem, and M. U. Shafique, "A systematic review on strategic release planning models," *Information and Software Technology,* vol. 52, no. 3, pp. 237-248, 2010.

[4] Apache Software Foundation. *Guidelines for creating a Jira ticket*. Available: https://infra.apache.org/pages/jira-guidelines.html, accessed on Dec. 20, 2019.

[5] N. K. S. Roy and B. Rossi, "Towards an improvement of bug severity classification," in *Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA'14)*, 2014, pp. 269-276.

[6] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR'10)*, 2010, pp. 1-10.

[7] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM'08)*, 2008, pp. 346-355.

[8] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empirical Software Engineering,* vol. 20, no. 5, pp. 1354-1383, 2015.

[9] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan, "An empirical study on factors impacting bug fixing time," in *Proceedings of the 19th Working Conference on Reverse Engineering (WCRE'12)*, 2012, pp. 225-234.

[10] R. K. Saha, S. Khurshid, and D. E. Perry, "An empirical study of long lived bugs," in *Proceedings of the 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE'14)*, 2014, pp. 144-153.

[11] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?," in *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR'07)*, 2007, pp. 1-8.

[12] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering,* vol. 14, no. 2, pp. 131-164, 2009.

[13] X. He, P. Avgeriou, P. Liang, and Z. Li, "Technical debt in MDE: A case study on GMF/EMF-based projects," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS'16)*, 2016, pp. 162-172.