2014 IEEE International Conference on Software Maintenance and Evolution

A Pilot Study of Diversity in High Impact Bugs

Yutaro Kashiwa Hayato Yoshiyuki
Graduate School of Systems Engineering,
Wakayama University
Email: {s141015, s151054}@sys.wakayama-u.ac.jp

Yusuke Kukita Masao Ohira
Faculty of Systems Engineering,
Wakayama University
Email: {s161018, masao} @sys.wakayama-u.ac.jp

***Abstract*—Since increasing complexity and scale of modern software products imposes tight scheduling and resource allocations on software development projects, a project manager must carefully triage bugs to determine which bug should be necessarily fixed before shipping. Although in the field of Mining Software Repositories (MSR) there are many promising approaches to predicting, localizing, and triaging bugs, most of them do not consider impacts of each bug on users and developers but rather treat all bugs with equal weighting, excepting a few studies on high impact bugs including security, performance, blocking, and so forth. To make MSR techniques more actionable and effective in practice, we need deeper understandings of high impact bugs. In this paper we report our pilot study on high impact bugs, which classifies bugs reported to four open source projects into six types of high impact bugs.**

I. Introduction

Since increasing complexity and scale of modern software products imposes tight scheduling and resource allocations on software development projects, a project manager must carefully triage bugs to determine which bug should be necessarily fixed before shipping. There are a number of studies relate to triaging reported bugs correctly and/or automatically. Prior work proposed several promising approaches, for example, automating bug triaging [1]–[3], detecting duplicate bugs [4]–[6], and understanding the rationale for the reassigning and re-opening of bugs [7]–[9]. In these studies, however, each bug are equally treated without considering its impacts on bug management process and software products. Although some important attributes included in a bug report (e.g., priority and severity) are used in the previous studies, they dose not answer *why they are important*. In other words, the previous studies do not carefully consider each bug in terms of *why it is important to whom?*. To make MSR techniques more actionable and effective in practice, we need deeper understandings of high impact bugs.

In this paper we report our pilot study on high impact bugs, which classifies bugs reported to four open source projects into six types of high impact bugs. In the case study, one hundred bug reports are manually inspected for each project and are classified into six types of high impact bugs based on previous studies [10]–[15] which focus on high impact bugs. Although the previous studies [10]–[15] only focused on one or two aspects of high impact bugs, our case study aims to reveal distributions of high impact bugs in reported bugs and overlapped relationships among high impact bugs.

In what follows, Section I provides a literature review on six types of high impact bugs and we roughly classify them into process and product bugs. Section III describes our case study which analyzes high impact bugs and relationships among them in four open source software projects (Accumulo, Derby, Qpid, and Thrift). Section IV discusses our findings and limitations of our case study. Section V concludes the paper and describes our future work.

II. Literature review on high impact bugs

Since a bug can highly impact on a variety of activities in the bug management process, products and end-users, several recent studies [10]–[15] have started with exploring high impact bugs such as performance, security, blocking bugs and so forth. In this paper, we first classify studies on high impact bugs into two types: process and product. A bug can especially impact on a bug management process in a project. For instance, a security bug must be faster fixed than other bugs. Developers would need to change the order to fix bugs when a security bug is reported to the project. A bug can also affect software products as well. For instance, a performance bug can directly decrease the satisfaction of products' users. In other words, a bug impacted on a bug management process is a matter for developers and a bug impacted on products is a matter of users. In what follows, we review studies on high impact bugs in terms of process and products.

A. Process

Bugs impacted on a bug management process include surprise bugs [10], dormant bugs [11], and blocking bugs [12].

1) Surprise bugs: A surprise bug is a new concept on software bugs, which was introduced by Shihab et al. in [10]. It can disturb the workflow and/or task scheduling of developers, since it appears in unexpected timing (e.g., bugs detected in post-release) and location (e.g., bugs found in files that rarely change in pre-release). A case study of a proprietary, telephony system which has been developed for 30 years [10] showed that the number of surprise bugs are very small (found in 2 % of all files) and the co-changed files and the amount of time between the latest pre-release change and the release date can be good indicators of predicting surprise bugs.

2) Dormant bugs: A dormant bug is also a new concept on software bugs and defined as "*a bug that was introduced in one version (e.g., Version 1.1) of a system, yet it is Not reported until AFTER the next immediate version (e.g., is reported against Version 1.2 or later)*" in [11]. [11] showed that 33 % of the reported bugs in Apache Software Foundation (ASF) projects were dormant bugs and were fixed faster than non-dormant bugs. It indicates that dormant bugs also affect developers' workflow in fixing assigned bugs in order to give first priority to fix the dormant bugs.

3) Blocking bugs: A blocking bug is a bug that blocks other bugs from being fixed [12]. It often happens because of a dependency relationship among software components. Since a blocking bug inhibit developers from fixing other dependent bugs, it can highly impact on developers' task scheduling since a blocking bug takes more time to be fixed [12] (i.e., a fixer needs more time to fix a blocking bug and other developers need to wait for being fixed to fix the dependent bugs).

B. Products

Bugs impacted on software products include security bugs [13], performance bugs [14], and breakage bugs [10].

1) Security bugs: A security bug can raise a serious problem which often impacts on uses of software products directly [13]. Since Internet devices (e.g., smartphones) and their users are increasing every year, security issues of software products are of interest to many people.

2) Performance bugs: A performance bug is defined as "*programming errors that cause significant performance degradation*" [14]. The "performance degradation" contains poor user experience, degrade application responsiveness, lower system throughput, and waste computational resources [16], [17]. [14] showed that a performance bug needs more time to be fixed than non-performance bug so that performance bugs affect users for a long time.

3) Breakage bugs: A breakage bug is "*a functional bug which introduced into a product because the source code is modified to add new features or to fix existing bugs*" [10]. Breakage bugs heavily impact on users who rely on the product in their daily operations.

III. Case study

This section describes our case study on high impact bugs that aims to reveal the distribution of each type of high impact bugs and overlapped relationships among the six types of high impact bugs.

A. Dataset

First we selected four open source projects (Accumulo: [1], Derby[2], Qpid[3] and Thrift[4]) from Apache Software Foundation (ASF) projects which use JIRA [5] for tracking and managing bugs and issues. An issue report tracked by JIRA has rich information for fine-grained analysis, compared to Bugzilla. It records versions affected by a bug and fixed versions so that we can identify dormant bugs.

Second we used several criteria to create a dataset for analysis, because reports in JIRA include requests for new features and enhancements and in this study we should not analyze reports not relate to bugs. We selected bugs in JIRA which meet the following conditions: (1)Type: Bug or Improvement, (2) Status: RESOLVED or CLOSED, (3) Resolution: Fixed, (4) Affects Version/s: other than None (i.e, this condition is to identify a dormant bug), (5) there is a corresponding commit to fix a bug (i.e, this condition is to identify a surprise bug).

TABLE I. Distributions of process and product bugs

Project	Subset	# of bugs	avg. days	SD
Accumulo	$\mathrm{Prc} * \overline{\mathrm{Prd}}$	6	79.0	120.2
	$\overline{\mathrm{Prc}} * \mathrm{Prd}$	39	28.5	72.2
	$\mathrm{Prc} * \mathrm{Prd}$	9	28.4	54.7
	$\overline{\mathrm{Prc}} * \overline{\mathrm{Prd}}$	46	44.7	86.4
Derby	$\mathrm{Prc} * \overline{\mathrm{Prd}}$	21	152.6	222.4
	$\overline{\mathrm{Prc}} * \mathrm{Prd}$	28	296.2	548.1
	$\mathrm{Prc} * \mathrm{Prd}$	15	23.7	25.6
	$\overline{\mathrm{Prc}} * \overline{\mathrm{Prd}}$	36	199.0	472.3
Qpid	$\mathrm{Prc} * \overline{\mathrm{Prd}}$	9	47.2	85.1
	$\overline{\mathrm{Prc}} * \mathrm{Prd}$	24	86.7	162.4
	$\mathrm{Prc} * \mathrm{Prd}$	5	113.2	152.3
	$\overline{\mathrm{Prc}} * \overline{\mathrm{Prd}}$	62	43.0	131.2
Thrift	$\mathrm{Prc} * \overline{\mathrm{Prd}}$	4	114.6	153.2
	$\overline{\mathrm{Prc}} * \mathrm{Prd}$	17	62.0	135.7
	$\mathrm{Prc} * \mathrm{Prd}$	6	188.9	278.7
	$\overline{\mathrm{Prc}} * \overline{\mathrm{Prd}}$	73	73.5	134.3

Prc: Process, Prd: Product, avg. days: average days to fix a bug

B. Study Method

For each project, we randomly chosen one hundred bug reports from the dataset and three of the authors independently reviewed them (i.e., a reviewer read four hundred bug reports in total). The reviewers tagged keywords (e.g., Surprise, Dormant, Blocking, Security, Performance, and Breakage) on each bug report. Multiple tags were allowed for a single bug report. After the review session, the three reviewers brought together their review results and discussed differences of tagged keywords between the reviewers. Although the review results were almost the same among the reviewers, some of bugs were judged differently. In particular, the reviewer had to judge performance, security and breakage bugs (i.e., product bugs) subjectively to some extent since there are no established definitions to detect performance, security and breakage bugs while surprise, dormant, blocking and breakage bugs can be semi-automatically identified by the definitions in previous studies [10]–[12]. If the reviewers found differences in the review results, they reviewed target bug reports together and discussed differences of understandings of the bugs among the reviewers until reaching a common understanding. After the discussion session, one or more keywords were tagged to four hundred bug reports in the four target projects.

C. Results

1) Process bugs vs. Product bugs: According to the classification of high impact bugs in Section I, for each project we first divided randomly selected one hundred bugs into process bugs and product bugs. We found that there were not only process bugs and product bugs but also bugs which belong to process and product bugs. Table I shows the number of process and product bugs, average days to fix a bug in each project, and the standard deviation of fixing time. Figure 1 is an example of relationships among process and product bugs. As we expected before the case study, in all the projects, the

[1]Accumulo: a high performance data storage and retrieval system with cell-level access control, http://accumulo.apache.org/

[2]Derby: a relational database management system, http://db.apache.org/derby/

[3]Qpid: an open internet protocol for reliably sending and receiving messages, http://qpid.apache.org/

[4]Thrift: an interface definition language and binary communication protocol, http://thrift.apache.org/

[5]JIRA: an issue tracking product developed by Atlassian, https://www.atlassian.com/software/jira

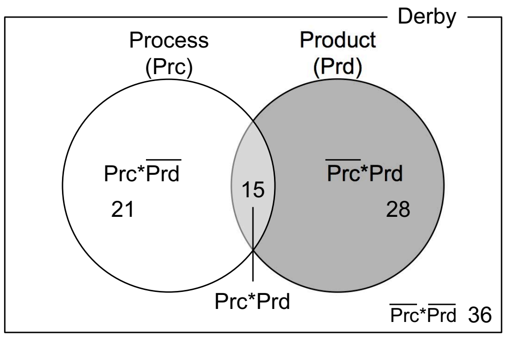
Derby
Process (Prc)
Product (Prd)
$\mathrm{Prc}*\overline{\mathrm{Prd}}$
21
15
$\overline{\mathrm{Prc}}*\mathrm{Prd}$
28
$\mathrm{Prc}*\mathrm{Prd}$
$\overline{\mathrm{Prc}}*\overline{\mathrm{Prd}}$ 36

Fig. 1. Relationship among process and product bugs in Derby

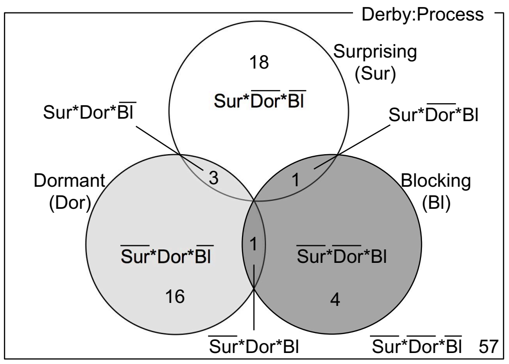
Derby:Process
18
$\mathrm{Sur}*\overline{\mathrm{Dor}}*\overline{\mathrm{Bl}}$
Surprising (Sur)
$\mathrm{Sur}*\mathrm{Dor}*\overline{\mathrm{Bl}}$
$\mathrm{Sur}*\overline{\mathrm{Dor}}*\mathrm{Bl}$
Dormant (Dor)
3
1
Blocking (Bl)
1
$\overline{\mathrm{Sur}}*\mathrm{Dor}*\overline{\mathrm{Bl}}$
16
$\overline{\mathrm{Sur}}*\overline{\mathrm{Dor}}*\mathrm{Bl}$
4
$\overline{\mathrm{Sur}}*\mathrm{Dor}*\mathrm{Bl}$
$\overline{\mathrm{Sur}}*\overline{\mathrm{Dor}}*\overline{\mathrm{Bl}}$ 57

Fig. 2. Relationship among process bugs in Derby

number of high impact bugs was smaller than other bugs which are not high impact bugs (i.e., $\overline{\mathrm{Prc}} * \overline{\mathrm{Prd}}$ in Table I). Of high impact bugs, product bugs (i.e., $\overline{\mathrm{Prc}} * \mathrm{Prd}$) were dominant in all the projects. Particularly in Accumulo, the number of product bugs was 39, much larger than the other projects. About the average days to fix a bug, we cannot observe a consistent feature among the target projects and between the subsets.

2) Process bugs: In order to deeply look at high impact bugs, next we analyze process bugs which include surprise, dormant, and blocking bugs. Table II shows distributions of the six kinds of high impact bugs in process bugs and product bugs. It also shows the average days to fix a bug by each category. Note that in the following analysis we do not consider overlapped relationships between process and product bugs as did in the previous analysis. In this analysis, one hundred bug reports are divided into process bugs or others. As described earlier, process bugs are categorized into three kinds of high impact bugs: Surprise, Dormant, and Blocking bugs. Figure 2 is an example of relationships of high impact bugs among process bugs.

From Table II, we can see that Derby and Qpid have many surprise bugs (i.e., $\mathrm{Sur} * \overline{\mathrm{Dor}} * \overline{\mathrm{Bl}}$) and we can also see that Accumulo and Derby have many dormant bugs (i.e., $\overline{\mathrm{Sur}} * \mathrm{Dor} * \overline{\mathrm{Bl}}$). Surprise and dormant bugs (i.e., $\mathrm{Sur} * \mathrm{Dor} * \overline{\mathrm{Bl}}$) were not so many but observed in all the projects.

TABLE II. DISTRIBUTIONS OF SIX KINDS OF HIGH IMPACT BUGS

		Subset	Accu mulo	Derby	Qpid	Thrift
# of bugs	Proc ess	$\mathrm{Sur} * \overline{\mathrm{Dor}} * \overline{\mathrm{Bl}}$	5	18	18	12
		$\overline{\mathrm{Sur}} * \mathrm{Dor} * \overline{\mathrm{Bl}}$	29	16	7	7
		$\overline{\mathrm{Sur}} * \overline{\mathrm{Dor}} * \mathrm{Bl}$	4	4	2	1
		$\mathrm{Sur} * \mathrm{Dor} * \overline{\mathrm{Bl}}$	7	3	1	2
		$\mathrm{Sur} * \overline{\mathrm{Dor}} * \mathrm{Bl}$	0	1	1	0
		$\overline{\mathrm{Sur}} * \mathrm{Dor} * \mathrm{Bl}$	3	1	0	1
		$\mathrm{Sur} * \mathrm{Dor} * \mathrm{Bl}$	0	0	0	0
		$\overline{\mathrm{Sur}} * \overline{\mathrm{Dor}} * \overline{\mathrm{Bl}}$	52	57	71	77
	Prod uct	$\mathrm{Sec} * \overline{\mathrm{Per}} * \overline{\mathrm{Br}}$	8	6	6	4
		$\overline{\mathrm{Sec}} * \mathrm{Per} * \overline{\mathrm{Br}}$	7	7	7	5
		$\overline{\mathrm{Sec}} * \overline{\mathrm{Per}} * \mathrm{Br}$	0	19	1	1
		$\mathrm{Sec} * \mathrm{Per} * \overline{\mathrm{Br}}$	0	0	0	0
		$\mathrm{Sec} * \overline{\mathrm{Per}} * \mathrm{Br}$	0	3	0	0
		$\overline{\mathrm{Sec}} * \mathrm{Per} * \mathrm{Br}$	0	1	0	0
		$\mathrm{Sec} * \mathrm{Per} * \mathrm{Br}$	0	0	0	0
		$\overline{\mathrm{Sec}} * \overline{\mathrm{Per}} * \overline{\mathrm{Br}}$	85	64	86	90
Avg. time	Proc ess	$\mathrm{Sur} * \overline{\mathrm{Dor}} * \overline{\mathrm{Bl}}$	39.5	215.7	130.9	80.7
		$\overline{\mathrm{Sur}} * \mathrm{Dor} * \overline{\mathrm{Bl}}$	20.0	256.6	26.4	54.2
		$\overline{\mathrm{Sur}} * \overline{\mathrm{Dor}} * \mathrm{Bl}$	106.7	67.7	7.2	0.1
		$\mathrm{Sur} * \mathrm{Dor} * \overline{\mathrm{Bl}}$	20.4	117.5	63.9	416.9
		$\mathrm{Sur} * \overline{\mathrm{Dor}} * \mathrm{Bl}$	—	24.3	27.2	—
		$\overline{\mathrm{Sur}} * \mathrm{Dor} * \mathrm{Bl}$	7.0	12.4	—	5.6
		$\mathrm{Sur} * \mathrm{Dor} * \mathrm{Bl}$	—	—	—	—
		$\overline{\mathrm{Sur}} * \overline{\mathrm{Dor}} * \overline{\mathrm{Bl}}$	48.7	181.9	43.5	75.7
	Prod uct	$\mathrm{Sec} * \overline{\mathrm{Per}} * \overline{\mathrm{Br}}$	39.5	110.9	74.5	93.1
		$\overline{\mathrm{Sec}} * \mathrm{Per} * \overline{\mathrm{Br}}$	59.1	136.4	77.5	238.4
		$\overline{\mathrm{Sec}} * \overline{\mathrm{Per}} * \mathrm{Br}$	—	99.7	1.3	27.3
		$\mathrm{Sec} * \mathrm{Per} * \overline{\mathrm{Br}}$	—	—	—	—
		$\mathrm{Sec} * \overline{\mathrm{Per}} * \mathrm{Br}$	—	14.0	—	—
		$\overline{\mathrm{Sec}} * \mathrm{Per} * \mathrm{Br}$	—	1.3	—	—
		$\mathrm{Sec} * \mathrm{Per} * \mathrm{Br}$	—	—	—	—
		$\overline{\mathrm{Sec}} * \overline{\mathrm{Per}} * \overline{\mathrm{Br}}$	37.3	241.5	55.2	71.4
SD	Proc ess	$\mathrm{Sur} * \overline{\mathrm{Dor}} * \overline{\mathrm{Bl}}$	70.8	476.2	192.0	153.7
		$\overline{\mathrm{Sur}} * \mathrm{Dor} * \overline{\mathrm{Bl}}$	36.7	545.3	40.8	83.8
		$\overline{\mathrm{Sur}} * \overline{\mathrm{Dor}} * \mathrm{Bl}$	181.4	93.7	7.2	0.0
		$\mathrm{Sur} * \mathrm{Dor} * \overline{\mathrm{Bl}}$	26.8	142.9	0.0	373.7
		$\mathrm{Sur} * \overline{\mathrm{Dor}} * \mathrm{Bl}$	—	0.0	0.0	—
		$\overline{\mathrm{Sur}} * \mathrm{Dor} * \mathrm{Bl}$	8.5	0.0	—	0.0
		$\mathrm{Sur} * \mathrm{Dor} * \mathrm{Bl}$	—	—	—	—
		$\overline{\mathrm{Sur}} * \overline{\mathrm{Dor}} * \overline{\mathrm{Bl}}$	91.6	399.5	126.3	135.6
	Prod uct	$\mathrm{Sec} * \overline{\mathrm{Per}} * \overline{\mathrm{Br}}$	58.7	148.2	139.1	75.7
		$\overline{\mathrm{Sec}} * \mathrm{Per} * \overline{\mathrm{Br}}$	115.8	193.8	95.4	311.1
		$\overline{\mathrm{Sec}} * \overline{\mathrm{Per}} * \mathrm{Br}$	—	199.7	0.0	0.0
		$\mathrm{Sec} * \mathrm{Per} * \overline{\mathrm{Br}}$	—	—	—	—
		$\mathrm{Sec} * \overline{\mathrm{Per}} * \mathrm{Br}$	—	5.5	—	—
		$\overline{\mathrm{Sec}} * \mathrm{Per} * \mathrm{Br}$	—	0.0	—	—
		$\mathrm{Sec} * \mathrm{Per} * \mathrm{Br}$	—	—	—	—
		$\overline{\mathrm{Sec}} * \overline{\mathrm{Per}} * \overline{\mathrm{Br}}$	80.6	509.2	142.0	134.6

Sur: Surprise, Dor: Dormant, Bl: Blocking, Sec: Security, Per: Performance, Br: Breakage, Avg. time: Average days to fix a bug

Interestingly, which kind of bugs is fixed faster than others varies between the projects. For instance, it takes longer time to fix surprise bugs and dormant bugs in Derby while bugs relate to blocking (e.g., $\overline{\mathrm{Sur}} * \overline{\mathrm{Dor}} * \mathrm{Bl}$, $\mathrm{Sur} * \overline{\mathrm{Dor}} * \mathrm{Bl}$, and $\overline{\mathrm{Sur}} * \mathrm{Dor} * \mathrm{Bl}$) need less time to be fixed than surprise bugs and dormant bugs. In Accumulo, surprise bugs ($\mathrm{Sur} * \overline{\mathrm{Dor}} * \overline{\mathrm{Bl}}$) and dormant bugs ($\overline{\mathrm{Sur}} * \mathrm{Dor} * \overline{\mathrm{Bl}}$) are fixed faster than blocking bugs ($\overline{\mathrm{Sur}} * \overline{\mathrm{Dor}} * \mathrm{Bl}$). Currently the size of our dataset is too small to test the statistical significance, but it would be worth analyzing the reason why these differences in fixing high impact bugs occurred. In the near future, we would like to collect more data from these projects and investigate more in depth.

3) Product Bugs: From Table II, we can see that the number of product bugs are smaller than process bugs in whole. We

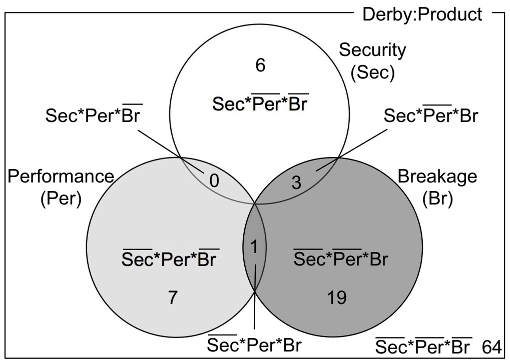
Derby:Product
Security (Sec)
6
Sec*$\overline{\text{Per}}$*$\overline{\text{Br}}$
Sec*Per*$\overline{\text{Br}}$
Sec*$\overline{\text{Per}}$*Br
Performance (Per)
Breakage (Br)
0
3
1
$\overline{\text{Sec}}$*Per*$\overline{\text{Br}}$
7
$\overline{\text{Sec}}$*$\overline{\text{Per}}$*Br
19
$\overline{\text{Sec}}$*Per*Br
$\overline{\text{Sec}}$*$\overline{\text{Per}}$*$\overline{\text{Br}}$ 64

Fig. 3. Relationship among product bugs in Derby

can also see that the time to fix security bugs ($\text{Sec} * \overline{\text{Per}} * \overline{\text{Br}}$) is shorter than performance bugs ($\overline{\text{Sec}} * \text{Per} * \overline{\text{Br}}$). This is consistent with the findings in the previous study [15]. In our dataset, only Derby has the larger number of breakage bugs ($\overline{\text{Sec}} * \overline{\text{Per}} * \text{Br}$). In the Derby project, discussions about regression test often observed as follows.

> Comment in Bug ID: DERBY-3032 — *the test should be disabled if the user is the operating system's super-user account. However, I don't know how to detect that condition inside Java. I recommend removing this test from the regression suite. We can flag it as a test case which we run standalone from a non-root account when we vet releases. Or not. I am not terribly worried about removing* —

At any hand, we need more data to deeply analyze product bugs since overlapped relationships among security, performance and breakage bugs were rarely observed excepting Derby.

IV. Discussions

A. Overlapped Bugs: just a minority? or really high impact?

In our case study, we observed overlapped bugs such as $\text{Sur} * \text{Dor} * \overline{\text{Bl}}$ and $\text{Sur} * \overline{\text{Dor}} * \text{Bl}$ though the number of them are very small, compared with other bugs. In what follows, we introduce some bug reports which has overlapped relationships and discuss whether they are really high impact bugs or not.

1) $\text{Sur} * \text{Dor} * \overline{\text{Bl}}$*:* This type of bug is regarded as logical conjunction of surprise bugs and dormant bugs. They were relatively often observed in all the four projects, compared to other overlapped bugs. It is no wonder that there are many [$\text{Sur} * \text{Dor} * \overline{\text{Bl}}$] bugs because surprise bugs are detected after a release, which means that there were missing bugs before the release and the missing bugs also might be missed in several releases in the past.

2) $\text{Sur} * \overline{\text{Dor}} * \text{Bl}$*:* This type of bug is regarded as logical conjunction of surprise bugs and blocking bugs. Of one hundred bugs, only one [$\text{Sur} * \overline{\text{Dor}} * \text{Bl}$] bug was observed in Derby and Qpid. In Derby, one developer argued to resolve an issue which was not recognized as an issue among other developers before.

> Comment in Bug ID: DERBY-530 — *Currently the client sends all attributes specified in the the url to the server in the RDBNAM parameter of the ACC-SEC command. This includes any client attributes (which the server ignores). It does not however make proper consideration for the properties specified in the info parameter of the connect(String url, Properties info) method. Another issue with the current approach is that user and password attributes get passed to the server without encryption if specified with the url.* —

This bug (Bug ID: DERBY-530) seemed to be a blocker for other bug (Bug ID: DERBY-559) incidentally.

> Comment in Bug ID: DERBY-559 — *The sending of user/password in RDBNAM is fixed by DERBY-530. getURL still needs to be fixed.* —

3) $\overline{\text{Sur}} * \text{Dor} * \text{Bl}$*:* This type of bug is regarded as logical conjunction of dormant bugs and blocking bugs. Although we found that the following five [$\overline{\text{Sur}} * \text{Dor} * \text{Bl}$] bugs, we could not understand the reason why these bugs happened.

- https://issues.apache.org/jira/browse/ACCUMULO-1854
- https://issues.apache.org/jira/browse/ACCUMULO-806
- https://issues.apache.org/jira/browse/ACCUMULO-2540
- https://issues.apache.org/jira/browse/DERBY-3997
- https://issues.apache.org/jira/browse/THRIFT-1523

These bug reports are tagged as Blocker, but they do not explicitly link to blocked bugs except in ACCUMULO-1854. They might be just tagged as Blocker to be fixed quickly though we need to further investigate them.

4) $\text{Sec} * \text{Per} * \overline{\text{Br}}$*:* This type of bug is regarded as logical conjunction of security bugs and performance bugs, but was not observed in our case study.

5) $\text{Sec} * \overline{\text{Per}} * \text{Br}$*:* This type of bug is regarded as logical conjunction of security bugs and breakage bugs. Three bugs were observed only in Derby.

- https://issues.apache.org/jira/browse/DERBY-2874
- https://issues.apache.org/jira/browse/DERBY-3202
- https://issues.apache.org/jira/browse/DERBY-5582

Two of these bugs were found in their regression test for security.

6) $\overline{\text{Sec}} * \text{Per} * \text{Br}$*:* This type of bug is regarded as logical conjunction of performance bugs and breakage bugs. One bug was observed only in Derby. This bug (Bug ID: DERBY-5412) was due to errors during regression test for performance.

From these bugs, we cannot provide any general findings and bugs introduced above might be incidentally observed in each project. Although it is also currently difficult to provide useful corpus to researchers and practitioners for classifying and/or predicting each type of high impact bugs, we would like to do so in the future by adding more data (e.g., at least one thousand bugs for each project).

B. Limitations

In this study we only used bug report data in four open source projects. As described above, we need to add more bug data to provide useful, actionable findings because we only analyzed one hundred bug reports for each project. The studied open source projects are relatively large scale, successful projects, but our findings in this study might not be applicable to any corporate projects. We need to analyze OSS projects and corporate projects to verify the generality of our findings.

V. Conclusion and Future Work

In this paper we conducted a case study on high impact bugs, which classified bugs reported to four open source projects into six types of high impact bugs. In the case study, one hundred bug reports were manually inspected for each project and are classified into six types of high impact bugs based on previous studies [10]–[15] which focus on high impact bugs. Our case study aimed to reveal distributions of high impact bugs in reported bugs and overlapped relationships among high impact bugs. In the future, we will add more bug data to provide useful corpus to make MSR techniques more actionable.

Acknowledgment

This research is conducted as part of Grant-in-Aid for Scientific Research (C) 24500041 by Japan Society for the Promotion of Science (JSPS).

References

[1] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (ESEC/FSE '09)*, 2009, pp. 111–120.

[2] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering (ICSE '06)*, 2006, pp. 361–370.

[3] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Fuzzy set and cache-based approach for bug triaging," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ser. ESEC/FSE '11. ACM, 2011, pp. 365–375.

[4] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE '10) - Volume 1*, 2010, pp. 45–54.

[5] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th international conference on Software engineering (ICSE '08)*, 2008, pp. 461–470.

[6] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Proceedings of the 29th international conference on Software Engineering (ICSE '07)*, 2007, pp. 499–510.

[7] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, ""not my bug!" and other reasons for software bug report reassignments," in *Proceedings of the ACM 2011 conference on Computer supported cooperative work (CSCW'11)*, 2011, pp. 395–404.

[8] ——, "Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10) - Volume 1*, 2010, pp. 495–504.

[9] E. Shihab, A. Ihara, Y. Kamei, W. Ibrahim, M. Ohira, B. Adams, A. Hassan, and K. Matsumoto, "Predicting re-opened bugs: A case study on the eclipse project," in *17th Working Conference on Reverse Engineering (WCRE'10)*, 2010, pp. 249–258.

[10] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan, "High-impact defects: A study of breakage and surprise defects," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*, 2011, pp. 300–310.

[11] T.-H. Chen, M. Nagappan, E. Shihab, and A. E. Hassan, "An empirical study of dormant bugs," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR '14)*, 2014, pp. 82–91.

[12] H. Valdivia Garcia and E. Shihab, "Characterizing and predicting blocking bugs in open source projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR '14)*, 2014, pp. 72–81.

[13] M. Gegick, P. Rotella, and T. Xie, "Identifying security bug reports via text mining: An industrial case study," in *Proceedings of the 7th Working Conference on Mining Software Repositories (MSR '10)*, 2010, pp. 11–20.

[14] A. Nistor, T. Jiang, and L. Tan, "Discovering, reporting, and fixing performance bugs," in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*, 2013, pp. 237–246.

[15] S. Zaman, B. Adams, and A. E. Hassan, "Security versus performance bugs: A case study on firefox," in *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11)*, 2011, pp. 93–102.

[16] I. Molyneaux, *The Art of Application Performance Testing : Help for Programmers and Quality Assurance*. O'Reilly Medea, 2009.

[17] R. E. Bryant and D. R. O'Hallaron, *Computer Systems : A Programmer's Perspective*. Addison-Wesley, 2010.