2017 IEEE 41st Annual Computer Software and Applications Conference

Revisiting the Correlation Between Alerts and Software Defects

A Case Study on MyFaces, Camel, and CXF

Meng Yan[1], Xiaohong Zhang[1], Ling Xu[1], Haibo Hu[1], Song Sun[1], Xin Xia[2]
[1]School of Software Engineering, Chongqing University, Chongqing, China
[2]Department of Computer Science, University of British Columbia, Canada
Email: {meng.yan, xhongz, xuling, hbhu, SongSun }@cqu.edu.cn, xxia02@cs.ubc.ca

***Abstract*—Static analysis tools (e.g., FindBugs) are widely used to detect potential defects in software development. A recent study suggests that there is a moderate correlation between the alerts reported by static analysis tools and software defects [1]. However, despite the actionable alerts reported by static analysis tools, they may report too many meaningless unactionable alerts. Actionable alert refers to the alert which is meaningful and fixable. Unactionable alert (i.e., false positive alert) refers to the alert which is regarded as unimportant to developers, inessential to source code, or will not be fixed by developers. Are all alerts (including both actionable and unactionable alerts) suitable for indicating software defects? To address this question, we classify all the alerts into two categories, namely actionable alerts and unactionable alerts. By the following, we conduct an empirical study to evaluate the degree of correlation between defects and alerts on the evolution of three open source projects with totally 40 releases. The objective of the study is to explore two kinds of correlation analysis: one is the correlation between all the alerts reported by FindBugs and defects among the release history of a project, the other is the correlation between the actionable alerts and defects. As a result, we find that not all the alerts but the actionable alerts are suitable to be an early predictor of defects.**

***Index Terms*—software alert, actionable alert, unactionable alert, software defect**

I. Introduction

Static analysis tools (e.g., FindBugs [2]) have been widely used to detect potential bugs in software systems [3], [4]. Usually, static analysis tools work by analyzing a system without execution. They search for code violations in recommended programming practices from fixed program representations, such as source code, generated or compiled code, and abstractions or models of the system [5]. An alert is a potential code violation reported by static analysis tools, such as null pointer references, buffer overflows, and style inconsistencies [6].

Although static analysis tools are effective in some settings, it is not sure whether the reported alerts are actionable. Past studies refer to this problem as false positives, i.e., alerts on the defects which do not exist [7], [8]. Specifically, if a developer determines a alert is meaningful and fixable, we refer to it as an actionable alert. If an alert is a false positive which is regarded unimportant to developers, inessential to source code or will not be fixed by developers, we refer to it as an unactionable alert [6], [9], [10], [11].

There is a growing interest in using alerts reported by static analysis tools as an early predictor of software defects [1] (the term "defect" in this study refers to the post-release defect or field defect which is same to the work by Thung et al. [8] and Couto et al. [1]). It is reported that there is a moderate correlation between defects and alerts reported by FindBugs [1]. However, whether the unactionable alerts have a correlation with filed defects is rarely investigated. The investigation on the impact of unactionable alerts is motivated by the following two aspects. On the one hand, the unactionable alerts do not help to indicate code quality by developers and quality assurances (QAs), and they need to waste on average 5 minutes to review an unactionable alert. On the other hand, among the huge number of static alerts, 35-91% of them are unactionable alerts [6], which would hinder the practical usage of these static analysis tools [12], [13].

Intuitively, more source code static alerts indicate that more potential defects may be contained in the software, and it is useful for developers and QAs to take advantage of the quantity of alerts to estimate the number of defects. Couto et al. discovered that there is a correlation between all the FindBugs alerts and defects among 30 systems [1]. Despite the insight they provided, there are two issues which still remain: (1) they investigate relationship between all the alerts and the defects. Whether the unactionable alerts are suitable for indicating defects is not addressed; (2) the datasets they used are single releases of different projects. As a result, they reported a correlation analysis among 30 projects which contain one single release in each project. The correlation analysis among the releases history of one project is missing. However, a typical defect prediction model is often built by learning from the past data from the previous releases within a project. Thus, the correlation analysis among the releases history of one project is needed.

In this paper, we revisit the correlations between alerts and software defects. Different from Couto et al.'s study, we classify alerts into two categiries: actionable alerts and unactionable alerts, and we investigate the correlation between actionable alerts and unactionable alerts, and defects. Moreover, we choose the continued releases of projects to investigate the correlation between the alerts and defects among the release history of a project. As Figure 1 shows, we

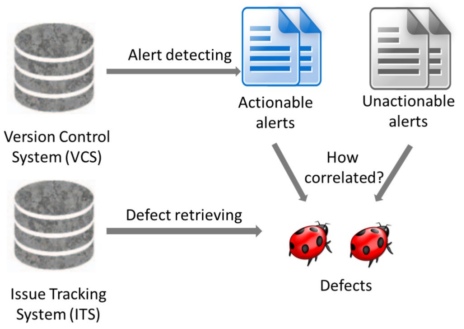

Fig. 1. Overall framework of our empirical study.

first collect a total of 40 releases from 3 open source projects (i.e., MyFaces, Camel, CXF) by mining their version control systems (e.g., SVN and Git). Second, we detect the alerts by use static analysis tool and classify them into actionable alerts and unactionable alerts. Third, we collect the defect data from their corresponding issue tracking systems (e.g., JIRA and Bugzilla). Finally, we perform the correlation analysis between alerts and defects. Our empirical study find that there is no significant correlation between the number of all (and unactionable) alerts and defects in terms of sequential project releases, and there is a statistically significant correlation between actionable alerts and defects.

The main contributions of the paper are as follows:

- We revisit the problem of the correlation degree comparison between actionable and unactionable alerts, and defects.
- We conduct an empirical study on three open source projects with totally 40 releases. The statistical analysis results show that the correlations between all and unactionable alerts, and defects are not significant, and there is a strong correlation between actionable alerts and defects.

The remainder of this paper is organized as follows: We present the empirical study data in Section 2. Section 3 describes the empirical study setup and Section 4 presents the empirical study results. In Section 5, we describe our related work. Finally, we draw a conclusion and present the future plan in Section 6.

II. Empirical Study Data

In this section, we describe the data used in this study in detail. First, we select mature open source projects with many releases. Second, we run the FindBugs to detect the alerts in each release. Third, we retrieve the defects of each release form the related issue tracking system (i.e., JIRA).

A. Selected Projects

Three projects are chosen from the Apache Foundation to collect the data needed in our research, namely MyFaces Core, Camel and CXF. We select the projects by considering the following criteria: (a) represent large-sized and mature project; (b) have a series of sequential releases; (c) open source, i.e., the version control system is accessible; (d) uses the issue tracking system to record their defects. Table I presents the summary about the target projects and the sequential chosen versions in each project.

TABLE I
Summary of the adopted projects

Project	Start	End	Date	number of releases	Description
MyFaces Core	2.1.0	2.1.15	29/05/2011-22/05/2014	16	JSF implementation
Camel	2.9.0	2.9.7	31/12/2011-20/09/2013	8	Integration framework
CXF	2.6.0	2.6.15	17/04/2012-21/10/2014	16	Web service framework

Additionally, a series of sequential releases, such as a series of 2.1.x released versions in MyFaces Core, are chosen from each project. We choose the historical releases for following reasons: (a) releases help to retrieve more accurate quantity of defects based on the search strategy we use; (b) sequential releases are required when we classify the alerts into actionable alerts and unactionable alerts.

B. Alerts Detection

FindBugs can find potential code errors by detecting bug patterns which are code idioms with errors [2]. We chose FindBugs as our analysis tool mostly because it is applied by many big IT firms like Google [13] and alerts reported by FindBugs are more relevant than PMD [14]. Additionally, developers can configure FindBugs to report high, medium, and low priority alerts. Due to the evidence provided by a study made by Couto et al. [1], which shows that there is a stronger connection between medium-priority alerts and defects than the connection between high-priority alerts and defect among FindBugs alerts. Thus, we apply the FindBugs tool with the medium-priority setting (which also is the default setting) to our experimental releases. The results of static analysis of FindBugs for each release are presented in XML files, from which we can easily obtain the number of alerts of each release and can easily parse out alert characteristics of each alert needed in the following alert classification process.

C. Defect Retrieving

In the JIRA issue tracking system, an issue is recorded with many labels like "issuetype", "status", "assignee" etc. Same to the work of [1], we refer the issue with label "issuetype = Bug" as a defect, and to assure that a defect exists we require that it is repaired by a developer in a specific release with the label "resolution = Fixed". We access JIRA to achieve the objective of collecting the number of defects in each release using the following search strategy. The strategy uses the number of fixed bugs found in the release after the current release, as the number of defects in the current release. For example, if we want to obtain the number of defects in the MyFaces Core 2.1.0, we count the number of bugs fixed in MyFaces Core 2.1.1, the search string is "issuetype = Bug AND resolution = Fixed AND fix version = 2.1.1". Then we iterate this method through the releases chosen as the experimental samples in the selected project, to get the number of defects in each release.

III. Empirical Study Setup

In our study, we are going to revisit the problem of the correlation degree comparison between actionable and unactionable alerts, and defects in terms of the release history within a specific project. In summary, our empirical study process consists of four phases. In the first phase, we use the JIRA bug tracking system to collect defects. In the second phase, we adopt the FindBugs to detect alerts. In the third phase, we classify all the alerts into actionable alerts and unactionable alerts. In the fourth phase, we conduct two correlation analysis. The first one is to investigate the correlation between the actionable alerts and defects, the second one is to investigate the correlation between all the alerts and defects. The first phase and second phase have been shown in previous section. In this section, we describe the third and fourth phase in detail. Furthermore, we formalize our study in two research questions:

RQ1: Do release with more alerts contain more defects in terms of the release history in a project?

RQ2: Are actionable alerts more appropriate than all the alerts for indicating defects?

A. Alerts Classification

The objective of this section is to classify all the alerts into actionable alerts and unactionable alerts. To achieve this objective, we adopt the alert classification method introduced by Heckman et al. [11] and propose a new classification method based on the difference between neighbor releases. The basic thinking in their study is that it classifies the alerts which are in a prior revision but not reported in a later revision as actionable alerts, and marks the remaining alerts as unactionable. Different from their work, we classify the alerts according the difference between releases instead of revisions.

The overview of our classification method is shown in Figure 2. *Release 1, Release 2*, and *Release n* represent sequential releases of a specific project, and the stripe graph is a set of FindBugs alerts which are detected in a release. If an alert exists in the current release, but disappears in the next release, we classify the alert as an actionable alert. Otherwise, we classify it as unactionable alert which indicate the alert still exists in the next release. We use the black square represents an actionable alert and the white square represents an unactionable alert.

In summary, there are three aspects differing from the classification manner of Heckman et al [11]: (a) we use sequential releases rather than revisions of each project; (b) alerts disappearing by file deletion also regarded as actionable, since file deletion may also be an approach of fixing errors by refactoring; (c) we do not consider an alert reopening situation, a reopening alert is a new alert (gray) in our experiment. In this way, we check all of alerts in the current release to count the number of actionable alerts of the current release. Then we iterate through the releases of each project to get the data about actionable alerts in each release used in our study.

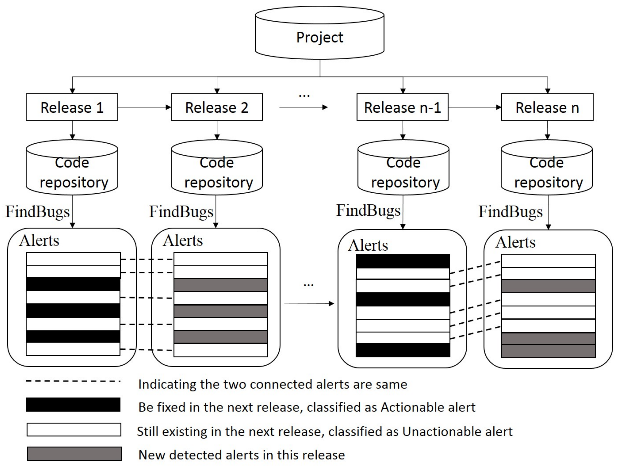
Project
Release 1
Release 2
...
Release n-1
Release n
Code repository
FindBugs
Alerts
Indicating the two connected alerts are same
Be fixed in the next release, classified as Actionable alert
Still existing in the next release, classified as Unactionable alert
New detected alerts in this release

Fig. 2. Overview of the alerts classification method in this study

B. Statistics Factors Studied

After collecting the quantity of alerts, actionable alerts, defects, and the lines of code in each release, we normalize the number of alerts and defects by dividing the KLOC (i.e., Kilo Lines of Code) in order to avoid the impact of project size. The normalized alerts and defects are named as alert density and defect density respectively. By the following, we conduct the correlation analysis between the actionable alert density and defect density in each project.

Using the data collection method presented in Section II, we get the data statistics about defects, alerts, actionable alerts, and lines of code in each of the releases of the target projects, including the detailed releases in each project, the number of defects in each release, the number of alerts reported by FindBugs (Alert), the number of actionable alerts (AAlert) found by the proposed classification method, and the number of thousands of lines of code (KLOC). By dividing with KLOC, we get the defect density (Defect Density), alert density (Alert Density), unactionable alert density (UAlert Density) and actionable alert density (AAlert Density).

IV. Empirical Study Results

A. RQ1 Results

Motivation: Previous research suggests that there is a moderate correlation between alerts and defects among various projects. However, whether the same correlation exists in the release history of a single project is not addressed. The motivation is that developers and quality managers can obtain the data about alerts and defects of the previous releases in a project. If there is a statistical correlation between alerts and defects in a specific projects release history, they can use the number of alerts as the indicator to estimate the quantity of post-release defects in projects before they are released. Otherwise, it is impractical to use the correlation to predict the number of defects in next release.

Approach: We use the Spearman's rank correlation coefficient which does not require the normality of test data. Spearman's rank correlation coefficient is a statistical measure to discover

TABLE II
DATA ANALYSIS OF THE DENSITY OF DEFECT, ALERT AND AALERT

	MyFaces Core				**Camel**				**CXF**			
	Defect Density	Alert Density	UAlert Density	AAlert Density	Defect Density	Alert Density	UAlert Density	AAlert Density	Defect Density	Alert Density	UAlert Density	AAlert Density
Max	1.0970	6.7653	6.7653	2.5738	0.6678	4.4657	4.4190	0.1336	0.3031	5.3081	5.2383	0.1174
Min	0.0556	4.9377	4.1771	0.0000	0.0000	4.3461	4.3321	0.0000	0.0048	5.1412	5.1315	0.0000
Mean	0.2487	5.4591	5.2632	0.1959	0.3024	4.4000	4.3633	0.0366	0.1542	5.2035	5.1736	0.0299
Std. Deviation	0.2716	0.5541	0.5498	0.6403	0.2106	0.0455	0.0324	0.0468	0.0937	0.0479	0.0327	0.0358
Skewness	2.482	1.693	0.810	3.878	0.225	0.517	0.562	1.550	0.093	0.697	0.624	1.533
Std.Error	0.564	0.564	0.564	0.564	0.752	0.752	0.752	0.752	0.564	0.564	0.564	0.564

TABLE III
SPEARMAN'S CORRELATION AND CORRELATION LEVEL

Correlation coefficient	**Correlation level**
0.0-0.1	None
0.1-0.3	Small
0.3-0.5	Moderate
0.5-0.7	High
0.7-0.9	Very High
0.9-1	Perfect

the degree of correlation between paired data [15]. The correlation coefficient is between -1 and 1, and the paired data has a stronger monotonic relationship when the absolute value of coefficient is closer to 1. In detail, Table III describes the various correlation coefficient ranges and the corresponding correlation levels [16]. In our experiment, through the SPSS statistical tool, we calculate the correlation coefficients for the projects MyFaces Core, Camel, and CXF. We first calculate the alert density (Alert Density) and defect density (Defect Density). Then, we calculate the correlation coefficients with significance level between alert density (Alert Density) and defect density (Defect Density). In addition, since we run the test many times, we also conduct the Bonferroni correction [17] to counteract the results of the correlation analysis.

Results: We compute four variables of density in each release for MyFaces Core, Camel, and CXF, i.e., Defect density, Alert density, AAlert density and UAlert density as Table II shows. The following statistical values are included: maximum, minimum, mean, standard deviation, skewness, and standard error value. The values of skewness and standard error will be used to select the type of correlation test when we calculate the correlation coefficient between the three variables. From the results of Table II, we conclude that:

- The defect density (Defect Density) range from 0.2487 ± 0.2716, 0.3024 ± 0.2106, and 0.1542 ± 0.0937 in the experimental releases of projects MyFaces Core, Camel, and CXF respectively.
- The density of FindBugs default alert (Alert Density) is smooth relatively, and each release for MyFaces Core, Camel, and CXF has the value between 5.4591 ± 0.5541, 4.4000 ± 0.0455, and 5.2035 ± 0.0479 respectively.
- The actionable alert density (AAlert Density) in each release is found to be very small, and the values range from 0.1959 ± 0.6403, 0.0366 ± 0.0468, and 0.0299 ± 0.0358 for projects MyFaces Core, Camel and CXF, respectively.

In addition, Table IV presents the results of Spearman test, the values ρ of second column (Alert Density) are correlation coefficients and the corresponding correlation level (corr_level) between alert density and defect density. Note that we use the Bonferroni correction to counteract the results of multiple comparisons, therefore the p-value are adjusted. In detail, we consider the correlation test is statistically significant at the confidence level of 95% if the adjusted p-value is less than 0.05. Considering the coefficients between defects and alerts, the Camel project has the maximal value 0.738, while the MyFaces Core project has a negative value 0.132 which runs against the intuition of the developer. Notice that all the adjusted p-values are above 0.05 which means the results are not statistically significant. Thus, from which we can conclude that alert and defect have a weak correlation for sequential project releases. In this case, the answer to our first research question (RQ1) is negative, there is no significant correlation between the number of all alerts and defects.

This finding may seem different with the conclusion of Couto et al. [1]. This difference results from the experimental objects are different. In detail, their finding is discovered via single release of different projects, while our finding is discovered via the multiple releases of one project. We believe that our finding has its practical meaning since many defect prediction models are built by learning from the historical data from the past releases in one project.

There is no significant correlation between the number of all alerts and defects in terms of sequential project releases.

B. RQ2 Results

Motivation: Despite the actionable alerts are alerts which are going to be fixed by developers, there is no evidence that actionable alerts has a stronger connection with defects than all the alerts currently. If there is an affirmative answer to this question, it is a very meaningful guide for developers and quality managers to build a robust and better quality system. By classifying the alerts to actionable alerts and unactionable alerts, developers can fix the actionable alerts preferentially which are more likely to contain potential errors.

Approach: Similar to the approach in answering for RQ1, we adopt the spearman correlation for answering this research question. In detail, we first classify all the alerts into actionable alerts and unactionable alerts. Then we calculate the actionable alert density (AAlert Density), unactionable alert density (UAlert Density) and defect density (Defect Density). Finally,

TABLE IV
RESULTS OF SPEARMAN CORRELATION COEFFICIENT IN EACH PROJECT

	Alert Density		UAlert Density		AAlert Density	
	ρ(p-value)	**corr_level**	**ρ(p-value)**	**corr_level**	**ρ(p-value)**	**corr_level**
Defect Density(Myfaces Core)	-0.132(1.000)	Small	-0.556(0.075)	High	*0.767(0.003)*	*Very High*
Defect Density(Camel)	0.738(0.111)	Very High	-0.190(1.000)	Small	*0.814(0.042)*	*Very High*
Defect Density(CXF)	0.311(0.72)	Moderate	-0.163(1.000)	Small	*0.624(0.030)*	*High*

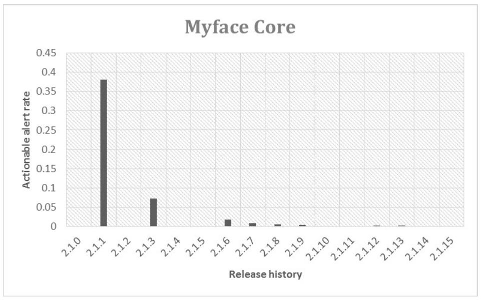

Fig. 3. The rate of actionable alerts in the release history of MyFaces Core

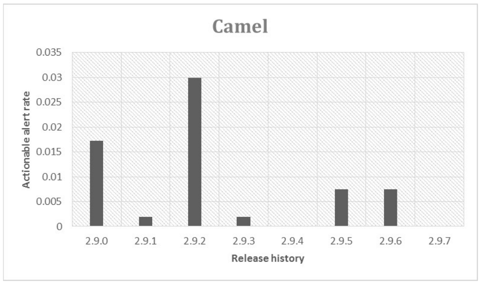

Fig. 4. The rate of actionable alerts in the release history of Camel

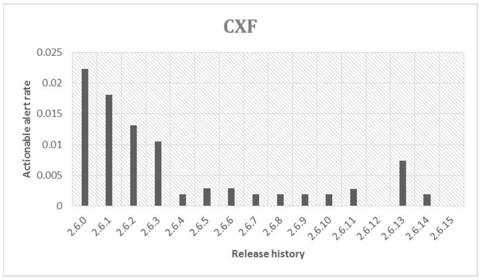

Fig. 5. The rate of actionable alerts in the release history of CXF

we calculate the correlation coefficients with significance level between AAlert density, UAlert density and Defect Density.

Results: Figure 3, Figure 4 and Figure 5 visualize the detailed numerical values about the rates of actionable alerts among FindBugs default alerts in each release of MyFaces Core, Camel and CXF, respectively. It is shown that the rate of actionable alerts is very low in our experiment. Except for the rate of release 2.1.1 in MyFaces Core that is over 0.38, other actionable alert rates are all lower than 0.1, and eleven releases among the total forty releases in three projects do not find any actionable alerts.

As Table IV shows, the values ρ in the column of UAlert Density represent the correlation coefficients between unactionable alert density and defect density, the values ρ in the column of AAlert Density are coefficients between actionable alert density and defect density with their adjusted p-values for each project.

Considering the UAlert Density, the results show that two projects possess the correlation level is "Small", the other project is "High". However, all the correlation is not statistically significant (p-value > 0.05). It indicates that there is no significant correlation between unactionable alerts and defects. Considering the AAlert Density, the maximal coefficient value is 0.814 in project Camel, and the minimum coefficient value is 0.624 in project CXF, the correlation level is high or very high and all of them have an high significance level (p-value < 0.05), which means that there is a statistically significant correlation between actionable alerts and defects. This indicates that the answer to our second research question (RQ2) is affirmative. Namely, the actionable alerts are more appropriate than all the alerts for indicating defects.

There is a statistically significant correlation between actionable alerts and defects. The actionable alerts are more appropriate than all the alerts for indicating defects. On the other hand, there is no significant correlation between unactionable alerts and defects.

V. RELATED WORK

This section presents related work about (a) the correlation studies between alerts and defects, (b) the alert classification techniques.

A. Correlation Analysis Between Alert and Defect

There are several papers about correlation studies between alerts and defects. Boogerd and Moonen made a study on the correlation between coding standard violations and defects in three levels (across software versions, on individual files and code lines) [18]. Alerts used in Boogerd and Moonen's experiment are raised by QA-C tool, a static analysis tool used to detect C language coding standard violations. Differing from our work, they only have analyzed a single software component of the NXP TV platform, while we use three open source projects. Also they considered the pre-release defects, instead of post-release defects.

Similar to our work, Couto et al. [1] evaluated the correlation between alerts and defects. In their work, they got the number of default alerts and high-priority alerts by executing FindBugs with default and high-priority configuration respectively, and also retrieved the number of defects from the JIRA system. Using the Spearmans rank correlation test, they found that there is a stronger positive correlation between default alerts and defects than the correlation between high-priority alerts and defects. The main differences between their work and our research are as follows: (a) the experimental subjects they used are a single release from 30 different systems, while we choose a series of sequential bug fix releases from each project among three projects; (b) particularly, we evaluate the correlation between actionable alerts and defects by using alert classification techniques, which was not addressed in their work.

B. Alert Classification

Alert classification is used to divide the alerts into actionable alerts and unactionable alerts. To save time used to review the huge number of alerts, Ogasawara et al. have only selected 41 key alert types as the actionable alert from over 500 kinds of warnings reported by the QA-Cła static analysis tool for C languagełbased on their experiences [19]. Ruthruff et al. [20] built a logistic regression model to classify the actionable alerts raised by using 33 alert characteristics, which had an accuracy over 70%, and they introduced a screening process to select alert characteristics for the model, which also saved large time required to generate the model. By combining static analysis with dynamic analysis, Chen et al. [21] implemented IntFinder to detect suspect code instruction area, reported by static analysis tools, to lower the false positives.

In a recent actionable alert classification study, Hanam et al. used the contextual information location of each static analysis alert (also called alert patterns) to classify actionable and unactionable alerts [9]. In their work, they have extracted alert characteristics from source code features at or near the source of an alert, and have evaluated three machine learning techniques over three open source projects using the extracted alert characteristics. Via combining with other alert characteristics like FindBugs alert characteristics, they have identified 6% more actionable alerts than previous techniques.

VI. Conclusions and Future Work

In our work, we have evaluated whether alerts or actionable alerts can be used as an early predictor of post-release defects in the release history of a project. We have investigated a series of sequential releases of open source projects to answer these two questions: (RQ1) Do release with more alerts contain more defects in terms of the release history in a project? (RQ2) Are actionable alerts more appropriate than all the alerts for indicating defects? We obtained a negative result to RQ1, namely, there was no significant correlation between all the alerts and defects, the fluctuation of alert density stayed relatively steady especially in sequential releases. Therefore, it cannot thoughtlessly take advantage of FindBugs alerts to estimate the quantity of the potential defects of a system. On the contrary, the answer to RQ2 was positive, the result of the correlation test provided the evidence that there is a significant correlation between actionable alerts and defects. Therefore, it indicates that not all the alerts but the actionable alerts can be a better predictor of defects of a project.

In the future work, we also intent to extend our correlation study to other kinds of static analysis tools like PMD and Checkstyle. As the correlation between actionable alerts and defects, additional work we plan to do is to build quality model of source code by introducing the actionable alerts as a factor.

Acknowledgment

The work described in this paper was partially supported by the Fundamental Research Funds for the Central Universities of China (Grant No. 106112014CDJZR098801).

References

[1] C. Couto, J. E. Montandon, C. Silva, and M. T. Valente, "Static correspondence and correlation between field defects and warnings reported by a bug finding tool," *Software Quality Journal*, vol. 21, no. 2, pp. 241–257, 2013.

[2] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *ACM Sigplan Notices*, vol. 39, no. 12, pp. 92–106, 2004.

[3] N. Ayewah, D. Hovemeyer, J. D. Morgenthaler, J. Penix, and W. Pugh, "Using static analysis to find bugs," *IEEE software*, vol. 25, no. 5, pp. 22–29, 2008.

[4] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler, "A few billion lines of code later: using static analysis to find bugs in the real world," *Communications of the ACM*, vol. 53, no. 2, pp. 66–75, 2010.

[5] J. R. Ruthruff, J. Penix, J. D. Morgenthaler, S. Elbaum, and G. Rothermel, "Predicting accurate and actionable static analysis warnings: an experimental approach," in *ICSE*. ACM, 2008, pp. 341–350.

[6] S. Heckman and L. Williams, "A systematic literature review of actionable alert identification techniques for automated static code analysis," *IST*, vol. 53, no. 4, pp. 363–387, 2011.

[7] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman, "Analyzing the state of static analysis: A large-scale evaluation in open source software," in *SANER*, vol. 1. IEEE, 2016, pp. 470–481.

[8] F. Thung, D. Lo, L. Jiang, F. Rahman, P. T. Devanbu *et al.*, "To what extent could we detect field defects? an empirical study of false negatives in static bug finding tools," in *ASE*. ACM, 2012, pp. 50–59.

[9] Q. Hanam, L. Tan, R. Holmes, and P. Lam, "Finding patterns in static analysis alerts: improving actionable alert ranking," in *MSR*. ACM, 2014, pp. 152–161.

[10] S. Heckman and L. Williams, "On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques," in *ESEM*. ACM, 2008, pp. 41–50.

[11] ——, "A model building process for identifying actionable static analysis alerts," in *International Conference on Software Testing Verification and Validation*. IEEE, 2009, pp. 161–170.

[12] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *ICSE*. IEEE, 2013, pp. 672–681.

[13] N. Ayewah and W. Pugh, "The google findbugs fixit," in *Proceedings of the 19th international symposium on Software testing and analysis*. ACM, 2010, pp. 241–252.

[14] J. E. M. Araujo, S. Souza, and M. T. Valente, "Study on the relevance of the warnings reported by java bug-finding tools," *IET software*, vol. 5, no. 4, pp. 366–374, 2011.

[15] P. Sedgwick, "Spearmans rank correlation coefficient," 2014.

[16] W. G. Hopkins, *A new view of statistics*. Will G. Hopkins, 1997.

[17] H. Abdi, "The bonferonni and šidák corrections for multiple comparisons," *Encyclopedia of measurement and statistics*, vol. 3, pp. 103–107, 2007.

[18] C. Boogerd and L. Moonen, "Evaluating the relation between coding standard violations and faultswithin and across software versions," in *MSR*. IEEE, 2009, pp. 41–50.

[19] H. Ogasawara, M. Aizawa, and A. Yamada, "Experiences with program static analysis," in *Proceedings of the Fifth International Software Metrics Symposium*. IEEE, 1998, pp. 109–112.

[20] J. R. Ruthruff, J. Penix, J. D. Morgenthaler, S. Elbaum, and G. Rothermel, "Predicting accurate and actionable static analysis warnings: an experimental approach," in *ICSE*. ACM, 2008, pp. 341–350.

[21] P. Chen, H. Han, Y. Wang, X. Shen, X. Yin, B. Mao, and L. Xie, "Intfinder: Automatically detecting integer bugs in x86 binary program," in *International Conference on Information and Communications Security*. Springer, 2009, pp. 336–345.