ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/331792299

Challenges with Responding to Static Analysis Tool Alerts

Conference Paper · March 2019

DOI: 10.1109/MSR.2019.00049

CITATION
1

READS
59

4 authors, including:

Nasif Imtiaz
North Carolina State University

6 PUBLICATIONS **9** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project

Static Analysis Tool Usage View project

All content following this page was uploaded by Nasif Imtiaz on 15 March 2019.

The user has requested enhancement of the downloaded file.

Challenges with Responding to Static Analysis Tool Alerts

Nasif Imtiaz, Akond Rahman, Effat Farhana, and Laurie Williams
North Carolina State University, Raleigh, North Carolina
Email: simtiaz@ncsu.edu, aarahman@ncsu.edu, efarhan@ncsu.edu, lawilli3@ncsu.edu

***Abstract*—Static analysis tool alerts can help developers detect potential defects in the code early in the development cycle. However, developers are not always able to respond to the alerts with their preferred action and may turn away from using the tool. In this paper, we qualitatively analyze 280 Stack Overflow (SO) questions regarding static analysis tool alerts to identify the challenges developers face in understanding and responding to these alerts. We find that the most prevalent question on SO is how to ignore and filter alerts, followed by validation of false positives. Our findings confirm prior researchers' findings related to notification communication theory as 44.6% of the SO questions that we analyzed indicate developers face communication challenges.**

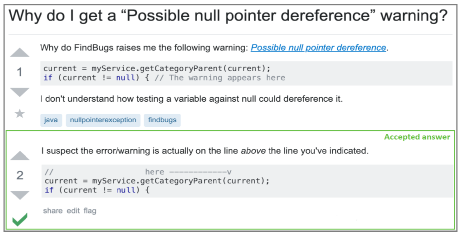
***Index Terms*—Static analysis tool, alerts, warnings, barriers, Stack Overflow**

I. Introduction

Static analysis tools (SATs) can be used to detect potential code defects (*alerts*) that could lead to field failure early in the software development process without having to execute the code [7], [10]. For example, the "goto fail" defect in a widely-used SSL implementation caused acceptance of invalid SSL certificates [19], and a date-formatting defect that caused large-scale Twitter outage [14], could have been avoided through static analysis [18]. Heartbleed [1], an infamous security defect, is now also detectable by a static analysis tool [20].

However, SATs also come with shortcomings that make developers turn away from using these tools [10]. Unactionable (e.g. false positives) alerts are a dominant problem that these tools suffer from [13], [16]. Through interviews with developers, Johnson et al. [10] found that false positives; the way in which the alerts are presented; and a lack of customizability are barriers to the use of static analysis tools. An article from Google also emphasizes that unactionable, incomprehensible, and untrustworthy alerts, among other things, are reasons that developers do not always use these tools [18]. The above-mentioned prior research [18] [10] suggest that the more effective alert messages are in helping developers acting on the alert, the more acceptance the SATs will have.

The goal of this paper is to aid static analysis tool makers to design effective alert presentation techniques by identifying the challenges developers face in responding to alert messages. We analyze questions related to SAT alerts on Stack Overflow (SO) to categorize the challenges that cause developers seek help on SO. We chose SO as it is a popular question and answer website for developers. To motivate this goal, consider Figure I as an example SO post. The user gets a "null pointer dereference" alert from the FindBugs static analysis tool that appears on a specific line of Java code. But the user is unable to understand how that line of code can dereference a null pointer and, therefore, asks a question on SO. The accepted answer tells us that while the alert is indeed true, it points to the wrong line number of code. An accurate problem description from the alert message in this case, could have saved the developer's time in fixing the alert more quickly.



Why do I get a "Possible null pointer dereference" warning?

1

Why do FindBugs raises me the following warning: *Possible null pointer dereference*.

```
current = myService.getCategoryParent(current);
if (current != null) { // The warning appears here
```

I don't understand how testing a variable against null could dereference it.

2

I suspect the error/warning is actually on the line *above* the line you've indicated.

```
//                  here -----------v
current = myService.getCategoryParent(current);
if (current != null) {
```

Fig. 1. A sample SO post regarding an SAT alert

Previously, Johnson et al. [9] performed a think-aloud study with developers using program analysis tools, and identified 10 kinds of communication challenges developers face in responding to the alerts. This paper builds on their work of Johnson by identifying challenges regarding SAT alerts in the context of questions asked by developers on SO. In prior work, researchers mined questions posted on SO to identify programming concerns and challenges in specific sub-domains through qualitative analysis [12], [15]. Following similar approaches, we ask:

- **RQ1:** What categories of challenges do static analysis tool (SAT) alerts related questions fall under?
- **RQ2:** How many questions without an accepted answer do SAT alerts related question categories yield?
- **RQ3:** How many views do SAT alerts related question categories yield?

We conduct an empirical study using the MSR2019 Mining Challenge dataset [3]. By systemically applying a set of filtering criteria, we identify 280 SO questions related to SAT alerts and perform qualitative analysis to categorize the challenges developers face while responding to the alerts. Researchers in prior work have studied the reasons why developers do not always want to use SATs and how alert messages miscommunicate with them [9], [10], [18]. The research methods of this prior work involved interviews, surveys, and participant observation. In this study, we look at the problem from a different perspective — that is developers asking questions on SO; and aim to examine prior findings with empirical evidence. ***Our contribution** is a categorization of questions that developers ask on SO regarding SAT alerts.*

II. Methodology

We use the MSR 2019 Mining Challenge Dataset provided by Baltes and colleagues [3]. The following subsections summarize the four steps of our empirical study.

A. Step One: Filter SO posts

To identify the SO posts regarding questions on SAT alerts, we look for certain tags and keywords in the question title of a post. A tag is a keyword or label that categorizes an SO question with similar questions and helps the interested community to find and answer that question [2]. When developers ask a question about a specific tool, they commonly use the tool's name as a tag.

As a first step to determine if a question is about an SAT, we look for the names of popular SATs in the list of tags of an SO question. We use a list of SATs from OWASP (Open Web Application Security Project) [21] and checked if the SO platform contains any tag for these tools. We found 10 open source and 6 commercial SATs that have their respective tag on the SO platform. We also include the tag "static-analysis". The first row in Table I lists all the 17 tags that we look for in the tags of an SO question.

At the next step, we determine if each SO question is related to alert messages as we are not interested in any other aspects of SATs that developers may ask about (e.g. tool setup). To do this, we see if the question title contains any keyword that is synonymous to or related to *alerts*. The second row in Table I lists 9 keywords that we look for in an SO question title. We include keywords "positive", "negative", "right", and "wrong" as we notice that developers often ask on the validity of an SAT alert. We also had "report" and "result" as two other keywords in this list, but after primary investigation, we found that these two terms generated irrelevant results (e.g. configure, export, or format a tool report). Therefore, we discarded these two terms from our final list.

We select the SO questions that contain at least one of the selected 17 tags **and** contain at least one of the 9 keywords in their title. Similar to prior work, we discard a question in case of negative votes as it may indicate the question is ill-formed or confusing [23]. Finally, step one resulted in 468 SO posts on which we apply a manual inspection.

TABLE I
KEYWORDS USED TO QUERY SO POSTS

Tags	*fxcop, bandit, brakeman, findbug, find-sec-bug, flawfinder, pmd, prefast, sonarqube, spotbug, coverity, fortify, klocwork, parasoft, pvs-studio, veracode, static-analysis*
Keywords in Title	*warning, alert, alarm, positive, negative, notif* (to cover notify and notification), *output, right, wrong*

B. Step Two: Prepare Coding Scheme

In order to categorize the SO questions based on what challenges developers have faced, we need a coding scheme. Previously, Johnson et al. [9] have studied the challenges in interpreting program analysis tool notifications (i.e., alerts) and proposed a **notification communication theory**. The theory comprises of 10 communication challenges developers face with alert messages. The challenges fall under two broad categories: **Knowledge Gap** and **Knowledge Mismatch**. Knowledge Gap is a gap between developer's knowledge and the information provided by the tool alert. The 5 challenges categorized under Knowledge Gap are: General Knowledge Gaps; Conceptual Knowledge Gaps; Notification Experience Gaps; Problem Importance Gaps; and Problem Resolution Gaps. Knowledge Mismatch is a mismatch between what a developer expects an alert to communicate and what the alert actually communicates. The 5 challenges categorized under Knowledge Mismatch are: General Problem Description Mismatch; Information Salience Mismatch; Visual Communication Mismatch; Consistent Communication Mismatch; and Familiar Communication Mismatch. *We start our qualitative coding based on the challenges from notification communication theory and add new challenges if a need arises.*

Three authors conducted a pilot analysis of randomly-sampled 50 SO posts. Through this pilot analysis, we see 3 recurring themes in many SO posts that are not related to communication challenges. Therefore, we decided to add 3 new categories of challenges to our coding scheme in addition to the 10 challenges from communication theory.

1) **Ignore/Filter alerts:** The questioner wants to ignore or filter some alert(s) based on a criteria but does not know how to. For example, the questioner may want to ignore (or, filter out) all alerts under a certain alert type.
2) **False Positive Validation:** The questioner suspects an alert as a false positive (FP) and wants to validate it.
3) **Handling False Positives:** The questioner identifies an alert as a false positive and wants to know how to use the tool to take desired action (e.g. mark as an FP).

We use this coding scheme that consists of 13 challenges to manually code 468 SO posts found in Step One, outlined in Section II-A. We discard a post as invalid if it does not relate to SAT alerts, and categorize as "Others" if the post is valid but does not fall under any of the 13 categories.

C. Step Three: Manual Coding

Through a pilot analysis, three authors sat together to discuss and come to a common understanding on how to code the SO posts. In many cases, an SO post could point to multiple categories. For example, a developer could ask for a solution on how to fix an alert due to not comprehending a programming concept in the alert message which indicates both conceptual knowledge and problem resolution gap. However, the authors decided to code each post to a single category — the challenge that seemed most relevant given the discussion context (question, answers, and comments). The first author

TABLE II
FINDINGS - RQ1 (Q), RQ2 (UNS), RQ3 (VQ)

Categories of Challenges	**Q**	**UNS**	**VQ**
Ignore/Filter Alerts	**23.9%**	**38.8%**	**702**
ignore an alert type	8.2%	34.8%	1041
ignore single alert	7.6%	28.6%	635
filter alerts based on code location	5.7%	62.5%	670
False Positive Validation	**22.9%**	**32.8%**	**368**
Problem Resolution Gap	**19.6%**	**43.6%**	**636**
Conceptual Knowledge Gap	**11.1%**	**22.6%**	**803**
Handling False Positives	**8.6%**	**66.7%**	**328**
General Problem Description Mismatch	**7.5%**	**38.1%**	**418**
General Knowledge Gap	2.5%	42.9%	109
Problem Importance Gap	1.4%	50.0%	420
Information Salience Mismatch	1.1%	33.3%	78
Consistent Communication Mismatch	1.1%	33.3%	1695
Visual Communication Mismatch	0.4%	100.0%	163

TABLE III
TOOL REPRESENTATION

SonarQube	72	FindBugs	61	FxCop	40
PMD	17	Brakeman	13	Coverity	12
Fortify	9	PVS-Studio	8	KlocWork	6
Visual Studio	4	Veracode	3	Others	35

then coded all the 468 posts. He discarded 188 of these posts as invalid. The rest of the 280 posts were coded according to the coding scheme discussed in Section II-B. The dataset is publicly available at: https://github.com/nasifimtiazohi/MSR_SAT_challenges

To mitigate bias introduced by a single-coder rating from the first author, the second and the third author coded a subset of the 280 valid SO posts (30 posts each) to validate and establish confidence in the first author's rating. We then measure Kohen's Kappa agreement rate between the first author and the validators. The two validators had a substantial (.64) and a moderate agrrement (.54) with the first author [11].

D. Step Four: Analysis

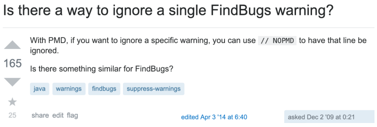
To answer RQ1, we provide the list of categories that we identified from manual coding (II-C). Then for each category x, we report the frequency of questions (Q(x)) under that category. To answer RQ2, we measure the proportion of questions that has no accepted answer (UNS(x)) under each category. The rate of questions with no accepted answer (unsatisfactory answers) indicates which types of questions are difficult to answer [17]. To answer RQ3, we report the median view count (VQ(x)) of all the questions under each category as the number of views indicate interest for a question from both registered and non-registered users of SO platform [15].

III. FINDINGS

A. RQ1

The second column (Q) in Table II shows the frequency of questions for each challenge category. Table III shows the distribution of posts under different SATs in our dataset. We find that the top six categories of challenges constitute more than 90% of the total questions. Here, we describe our findings for the top six categories:

1) Ignore/Filter alerts: Developers may want to ignore an alert irrespective of false positives or not. Developers may also want to filter the alerts based on alert type, code location (files, methods, data type etc.), or priority. We find that how to ignore/filter alerts in the SATs is the most discussed issue on SO (67 posts, 23.9%). For example, the most viewed post with the highest score and favorite votes in our dataset is a question on how to ignore a single warning in FindBugs as shown in Figure 2.


Is there a way to ignore a single FindBugs warning?

With PMD, if you want to ignore a specific warning, you can use `// NOPMD` to have that line be ignored.

Is there something similar for FindBugs?

Fig. 2. The most viewed SO post regarding SAT alerts

A further categorization tells us that most posts under this category are about how to ignore a specific alert type (23, 8.2%), while the second most dominant case is how to ignore a single alert (22, 7.6%). There are 16 posts (5.7%) where developer wants to ignore/filter alerts from certain parts of the code (e.g. files/functions/data type). In 3 of the posts, the developer wants to know how to ignore alerts of certain priorities originally set by the tool.

2) False Positive Validation: The second highest number of SO posts (64, 22.9%) comes across as false positive validation where developers want their peers to validate their suspicion of an alert as an FP. We further investigated the answers and comments of these posts to categorize if the developer's suspicion was true or not. We find that while in 42 cases the alert was indeed a false positive, in 13 cases the alert was a true positive. In 3 cases, the alert was a false positive but SO community still suggested good code practices to avoid the alert. In 6 cases, we could not find a conclusive evidence if the alert was true or not.

We noticed that in 25 cases where the alert was indeed an FP, the developer actually used the SO post as a means of documenting the FP and report a tool bug. In 7 cases, maintainers of these tools responded to these SO posts acknowledging the bug and created bug reports. This trend sheds light on how toolmakers can actively attend to developers feedback to improve their SATs.

3) Problem Resolution Gap: The third dominant challenge we find is Problem Resolution Gaps (PRG) with 55 (19.6%) SO posts where the developer wants to know *how to fix* an SAT alert as there lied a gap between what the developer knows about resolving an alert and the resolution suggested by the alert [9]. Johnson et al. identified that "most often this gap was present because the notification did not include information specific to resolution". Among the 10 challenges in the notification communication theory, we find that PRG is the most dominant one. This finding is not surprising though as it is common for developers to ask for a possible solution for their problems on SO — in this case, fixing SAT alerts.

4) Conceptual Knowledge Gap: Another challenge against effective communication of alert messages that is also a dominant issue on SO with 31 posts (11.1%) is Conceptual Knowledge Gaps (CKG). According to Johnson et el., CKG indicates a gap between developer's knowledge on a programming concept (e.g. serialization) and the information provided by the alert regarding those concepts. For posts under these category, we find that developers do not understand the concept or theory behind the code defect described in an alert, and seek help from the SO community on *why* there is a defect.

5) Handling False Positives: The next dominant challenge (24 SO posts, 8.6%) that we find is how to handle the alerts that the developer has already evaluated as a false positive. In 9 posts, the developer reports a code pattern that always raises a false positive from the tool and wants to know if the tool can be customized. In 13 posts, the developer wants to know how to configure the tool to mark some alerts as false positive and suppress them in the process.

6) General Problem Description Mismatch: According to Johnson et al., General Problem Description Mismatch (GPDM) is a mismatch between how the participant would describe a problem and how the tool describes it. We find 21 posts (7.5%) where the developer was able to understand the alert from an accepted answer with a better explanation. Therefore, we code these posts as GPDM as our rational is that the developer would have liked a similar explanation to the accepted answer in the alert message generated by the tool.

B. RQ2, RQ3

The third (UNS) and the fourth column (VQ) in Table II shows results for RQ2 and RQ3 respectively. Overall, SAT alerts related questions have a median view count of 513 while 38.6% of the questions do not have a satisfactory (accepted) answer. Among the top 6 categories, the lowest view count for questions regarding False Positive Validation may explain the highest rate of unsatisfactory answers. Also, false positive patterns are more likely to be project specific and hence may not find an answer from SO community. Conversely, we see questions regarding Conceptual Knowledge Gap have the highest view count with the lowest rate of unsatisfactory answers. A possible reason behind this is that conceptual questions regarding programming concepts are easier to find an answer than tool-specific questions, as the former can be answered by a more generic community. We also see questions on how to filter alerts based on code location attract high views, yet a low rate of satisfactory answers.

C. Threats to Validity

A major threat of our study is that we primarily use a single-coder rating. Having multiple coder for each SO post could ensure more reliable human evaluation. Also, one SO post could point to multiple categories of challenges as discussed in II-C. However, for the sake of simplicity, we coded each post to a single category which may affect final results.

IV. Discussion

We find six challenges to constitute more than 90% of the questions developers asked on SO regarding SAT alerts. The questions come from more than 15 distinct tools which indicates that these challenges are generic to all SATs. We find that developers want to know how to ignore or filter alerts based on different criteria. A possible explanation behind this observation can be developer overload in triaging large volume of alerts [8]. As a result, developers may want to focus on the parts of the codebase and the alert types that are more critical to the project. Furthermore, prior work has shown that developers prioritize different types of warnings under different development contexts [22]. For example, developers may focus more on code logic and concurrency during continuous integration and on style conventions during code review. Also, the use of SATs can be of complex nature which may confuse the new users (e.g. code annotations to suppress alerts). Toolmakers may consider adding features to filter the output alerts in a flexible, easy manner. We also find that false positives (FP) are still a dominant issue in using SATs [5], [13]. Developers may find specific code patterns that induce FPs from the tools. Customizing the tools to specific project requirements may increase the acceptance of SATs in this regard.

We find that ineffective alert messages are also a major barrier to act upon those alerts. Our study confirms notification communication theory from prior work [9] as we find 44.6% of all the questions that we analyzed stemmed from communication challenges. Our result suggests that it is often difficult for developers to find correct fixes for the SAT alerts, and alert messages can be unable to properly explain the underlying problem. Integrated Development Environments (IDE) often provide quick fixes for compiler error messages (e.g. Eclipse) although their effectiveness is still being studied [4]. HelpMeOut is another tool that aids the debugging of error messages by suggesting past solutions that others have applied [6]. Toolmakers may consider adding similar features to their SATs so that developers are better aided by the alert messages.

V. Conclusion

We categorize SO questions regarding SAT alerts based on the challenges developers face and find that top six categories constitute more than 90% of all the questions. We find that developers want more control over the tools on filtering the alerts while false positives remain a challenge. The questions from SO also strengthen notification communication theory suggested in prior work that gaps and mismatches between developers' knowledge and the information provided in the alert messages create challenges for developers in responding to SAT alerts. While future research is required on automatically suggesting alert resolution techniques and reducing false positives, our findings suggest that static analysis tool makers should consider adding easy-to-use features for flexible alert filtration and project-specific customization to their tools.

VI. Acknowledgement

This material is based in part upon work supported by the National Science Foundation under grant number 1451172.

References

[1] Heartbleed. https://en.wikipedia.org/wiki/Heartbleed. Accessed of Feb. 6, 2019.

[2] Tags. https://stackoverflow.com/tags.

[3] Sebastian Baltes, Christoph Treude, and Stephan Diehl. Sotorrent: Studying the origin, evolution, and usage of stack overflow code snippets. In *Proceedings of the 16th International Conference on Mining Software Repositories (MSR 2019)*, 2019.

[4] Titus Barik, Yoonki Song, Brittany Johnson, and Emerson Murphy-Hill. From quick fixes to slow fixes: Reimagining static analysis resolutions to enable design space exploration. In *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*, pages 211–221. IEEE, 2016.

[5] Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. A few billion lines of code later: using static analysis to find bugs in the real world. *Communications of the ACM*, 53(2):66–75, 2010.

[6] Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R Klemmer. What would other programmers do: suggesting solutions to error messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1019–1028. ACM, 2010.

[7] Sarah Heckman and Laurie Williams. A model building process for identifying actionable static analysis alerts. In *Software Testing Verification and Validation, 2009. ICST'09. International Conference on*, pages 161–170. IEEE, 2009.

[8] Sarah Heckman and Laurie Williams. A comparative evaluation of static analysis actionable alert identification techniques. In *Proceedings of the 9th International Conference on Predictive Models in Software Engineering*, page 4. ACM, 2013.

[9] Brittany Johnson, Rahul Pandita, Justin Smith, Denae Ford, Sarah Elder, Emerson Murphy-Hill, Sarah Heckman, and Caitlin Sadowski. A cross-tool communication study on program analysis tool notifications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 73–84. ACM, 2016.

[10] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *Proceedings of the 2013 International Conference on Software Engineering*, pages 672–681. IEEE Press, 2013.

[11] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[12] Na Meng, Stefan Nagy, Danfeng Yao, Wenjie Zhuang, and Gustavo Arango-Argoty. Secure coding practices in java: Challenges and vulnerabilities. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 372–383. IEEE, 2018.

[13] Muhammad Nadeem, Byron J Williams, and Edward B Allen. High false positive detection of security vulnerabilities: a case study. In *Proceedings of the 50th Annual Southeast Regional Conference*, pages 359–360. ACM, 2012.

[14] Hacker News. Twitter outage report. https://news.ycombinator.com/item?id=8810157, 2016.

[15] Akond Rahman, Asif Partho, Patrick Morrison, and Laurie Williams. What questions do programmers ask about configuration as code? In *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*, pages 16–22. ACM, 2018.

[16] Zachary P Reynolds, Abhinandan B Jayanth, Ugur Koc, Adam A Porter, Rajeev R Raje, and James H Hill. Identifying and documenting false positive patterns generated by static code analysis tools. In *Software Engineering Research and Industrial Practice (SER&IP), 2017 IEEE/ACM 4th International Workshop on*, pages 55–61. IEEE, 2017.

[17] Christoffer Rosen and Emad Shihab. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering*, 21(3):1192–1223, 2016.

[18] Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, and Ciera Jaspan. Lessons from building static analysis tools at google. *Communications of the ACM*, 61(4):58–66, 2018.

[19] Synopsys Editorial Team. Coverity report on the 'goto fail' bug. https://www.synopsys.com/blogs/software-security/apple-security-55471-aka-goto-fail/, February 2014.

[20] Synopsys Editorial Team. On detecting heartbleed with static analysis. https://www.synopsys.com/blogs/software-security/detecting-heartbleed-with-static-analysis/, April 2014.

[21] Open Web Application Security Project user group. Source code analysis tools.

[22] Carmine Vassallo, Sebastiano Panichella, Fabio Palomba, Sebastian Proksch, Andy Zaidman, and Harald C Gall. Context is king: The developer perspective on the usage of static analysis tools. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 38–49. IEEE, 2018.

[23] John Viega, Gary McGraw, Tom Mutdosch, and Edward W Felten. Statically scanning java code: Finding security vulnerabilities. *IEEE software*, 17(5):68–74, 2000.

View publication stats