IDE Support for Cloud-Based Static Analyses

Linghui Luo
Paderborn University
Germany
linghui.luo@upb.de

Martin Schäf
Amazon Web Services
USA
schaef@amazon.com

Daniel Sanchez
Amazon Alexa
USA
danjsan@amazon.com

Eric Bodden
Paderborn University & Fraunhofer IEM
Germany
eric.bodden@upb.de

ABSTRACT

Integrating static analyses into continuous integration (CI) or continuous delivery (CD) has become the best practice for assuring code quality and security. Static Application Security Testing (SAST) tools fit well into CI/CD, because CI/CD allows time for deep static analyses on large code bases and prevents vulnerabilities in the early stages of the development lifecycle. In CI/CD, the SAST tools usually run in the cloud and provide findings via a web interface. Recent studies show that developers prefer seeing the findings of these tools directly in their IDEs. Most tools with IDE integration run lightweight static analyses and can give feedback at coding time, but SAST tools used in CI/CD take longer to run and usually are not able to do so. Can developers interact directly with a cloud-based SAST tool that is *typically used in CI/CD* through their IDE? We investigated if such a mechanism can integrate cloud-based SAST tools better into a developers' workflow than web-based solutions. We interviewed developers to understand their expectations from an IDE solution. Guided by these interviews, we implemented an IDE prototype for an existing cloud-based SAST tool. With a usability test using this prototype, we found that the IDE solution promoted more frequent tool interactions. In particular, developers performed code scans three times more often. This indicates better integration of the cloud-based SAST tool into developers' workflow. Furthermore, while our study did not show statistically significant improvement on developers' code-fixing performance, it did show a promising reduction in time for fixing vulnerable code.

CCS CONCEPTS

• **Theory of computation → Program analysis**; • **Human-centered computing → User studies**; **Usability testing**.

KEYWORDS

IDE integration, static analysis, cloud service, SAST tools, security testing

ESEC/FSE '21, August 23–28, 2021, Athens, Greece
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8562-6/21/08...$15.00
https://doi.org/10.1145/3468264.3468535

ACM Reference Format:
Linghui Luo, Martin Schäf, Daniel Sanchez, and Eric Bodden. 2021. IDE Support for Cloud-Based Static Analyses. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3468264.3468535

1 INTRODUCTION

Many companies are providing static analysis as a service, e.g., Coverity Scan [31], Veracode [37], Checkmarx [7] and LGTM [12]. These tools fit well into CI/CD, since CI/CD allows time for deep static analyses (e.g., inter-procedural data-flow analysis) of the complete code base without taking up resources on a user's machine. There are many benefits for performing static analysis tasks in the cloud. From the user's perspective, it can provide central storage and tracking of analysis results. Cloud-based SAST tools usually offer hooks to integrate with popular CI/CD systems, such as GitHub Actions, Jenkins, or Travis CI and offer a browser-based dashboard for developers to manage findings. From the supplier's perspective, parallelism in the cloud can improve the performance of these tools. As reported by Microsoft [17], moving static analysis for Windows drivers to the cloud significantly reduced the analysis time spent for the `svb_bugbash` suite with 22.5x speedup. In addition, the cloud environment provides a central configuration of the analysis. SAST tool suppliers can tune the analysis engine to keep the false-positive rate low and update the analysis rule set without shipping constant updates to customers.

Despite all these benefits of doing static analysis in the cloud, multiple studies have shown that the ideal reporting location for static analysis is the developers' IDE [8, 10]. So, there is a disconnect between the typical workflow, where SAST tools perform deep analysis in CI/CD, and developers' expectation of interacting with these tools much earlier in the development lifecycle, directly from the IDE. Some cloud-based SAST tools provide IDE integration to trigger an analysis manually from the IDE. E.g., Veracode Static for IDE [38] allows developers to upload binaries to the cloud, start a scan on demand, and triage the findings from the IDE. Does this style of IDE integration meet developers' expectation?

Integrating such a cloud-based SAST tool into the developers' workflow comes with a set of challenges. In CI/CD, it is acceptable if an analysis spends several minutes computing in the cloud. How would such waiting time impact user experience in the IDE? Another challenge of designing such an IDE integration is dealing with the desynchronization between the code that is analyzed in the cloud and the code in developers' IDE. While the analysis is running remotely, developers might write more code which makes the analysis results "out-of-date". How should such results be displayed in the IDE? Especially for long-running analysis, are developers aware of the time to run it?

The main goal of this research is to explore how IDE support for a purely cloud-based static analysis, that is typically used in CI/CD, should be designed to meet the expectations of developers. We identify the key design elements for such IDE support, and investigate whether it fits better into developers' workflow in comparison to a web-based solution. Specifically, does it encourage more usage of the analysis, improve developers' performance (i.e., less time to fix code) and perceived usability? To investigate whether an IDE solution can improve developers' workflow, we conducted a user study (due to COVID-19, all interviews and usability tests were done remotely). The four stages of the user study were:
(1) User Interviews (section 2): We started by interviewing developers to understand their expectations of how cloud-based analyses should be triggered from an IDE, how the findings should be displayed there, and what UX features developers would like.
(2) Prototyping (section 3): Guided by the user interviews, we developed an IDE prototype for the existing tool CodeGuru Reviewer [2], using its infrastructure for CI/CD.
(3) Second-round Interviews (section 4): We presented the IDE prototype to the same developers in (1) to evaluate whether the design met their expectations. While developers were satisfied with most features implemented in the prototype, they found existing mechanisms for CI/CD, e.g., code uploading via Git, were cumbersome in the IDE.
(4) Usability Testing (section 5): Finally, we assessed our IDE prototype with a larger group of developers. In this test, we applied both quantitative and qualitative research methods to determine if the IDE solution was an improvement. We found that using the IDE prototype developers performed code scans three times more often than using the web-based solution. Our measurements also show a promising reduction in time for fixing code. We found that bringing the findings of the tool into the IDE didn't necessarily improve developers' workflow. Specifically, they expected:

- more education on capabilities and limitations of cloud-based SAST tools,
- real-time feedback on analysis progress (e.g., progress bars),
- quick validation of each fix, which implies incremental analysis on code changes,
- seamless analysis of code (e.g., an analysis button without going through steps such as uploading it),
- more interactive ways to suggest rescan, integrated into current workflows.

Background. Our study was conducted with developers at Amazon Web Services (AWS). We focused on the cloud-based SAST tool—CodeGuru Reviewer, which is used as part of the CI/CD process inside AWS. At AWS, every commit to its code bases is required to go through a code review process first. Teams can configure different SAST tools, including CodeGuru Reviewer, in their code review process. CodeGuru Reviewer has an expected running time of under 10 minutes. Currently, CodeGuru Reviewer integrated in the code review process only gets triggered to run (along with other quality assurance tools) when developers submit code changes to a remote repository via an internal pull request tool. This internal tool pushes code to a detached branch and developers have to wait until CodeGuru Reviewer and other quality assurance tools finish running. The findings of these tools are displayed in a web application. Developers have to address the findings before merging the code changes. Usually, developers address the findings together with comments from their teammates. However, developers told us they would like to get findings from CodeGuru Reviewer before their code reviews, which is not the case in CI/CD, so our focus of this study is to explore how IDE support could be an improvement over the current flow and whether an IDE solution is better than a web solution. CodeGuru Reviewer also provides a public API and a web interface to trigger a scan of a specific commit and fetch the findings. We used this API to build an IDE prototype.

2 USER INTERVIEWS

First, we wanted to identify developers' expectations from IDE support for cloud-based SAST tools. With user interviews we aimed to answer the following two research questions:
RQ1: What do developers expect from IDE support for a cloud-based SAST tool?
RQ2: How could such IDE support fit into developers' workflow?

Research Methods. We wanted to understand how IDE support for a cloud-based SAST tool could fit better into developers' workflow in comparison to a web solution. Thus, we interviewed developers who already used a cloud-based SAST tool in practice. We conducted *semi-structured* interviews with developers using *contextual inquiry* [26]. Contextual inquiry allows us to understand how developers work with CodeGuru Reviewer on a day-to-day basis. Before the interview, we sent each participant a link to one of their code reviews on which CodeGuru Reviewer detected issues. During the interview, each participant was first asked to talk through their code review regarding CodeGuru Reviewer's findings. Participants were asked to demonstrate how they fixed those issues in their IDEs together with the vulnerable code. Afterwards, while they had their IDEs opened, they were asked about their expectations on the IDE support and also to describe the features and demonstrate them in their IDEs if possible. The detailed questions list can be found in [20]. To differentiate from common static tools that run analysis on the same machine as the IDE, we explicitly told participants that the analysis is running in the cloud and that a scan takes minutes. Each interview lasted 45 minutes to 1 hour.

Participants. We interviewed 9 participants who were all *software development engineers* from different teams and countries within AWS. To ensure that participants were already familiar with SAST tools and willing to use them, we started by finding developers who were involved with code reviews on which CodeGuru Reviewer had found issues (n=328) and then invited developers (n=252) who replied to the CodeGuru Reviewer findings. All the interviewees had experience using static analysis tools (CodeGuru Reviewer (9), FindBugs (7), CheckStyle (6), ESLint (1), SonarQube (1), Coverlay (1), IntelliJ built-in static tools (1)). In the following, we denote the 9 developers with P1-9.

Data Collection. All interviews were recorded and transcribed. They were carried out over video conferencing and all participants shared their screens during the interview so that their IDE activity could be captured.

Data Analysis. The responses were analyzed using *thematic analysis*. We used both deductive (codes derived from the questions we prepared for the interviews) and inductive (codes derived from the responses) coding [11]. The codebook contains 21 codes that were discussed and agreed upon by two researchers. The list of codes and their definitions can be found in [19]. The coding itself was first done by the researcher who conducted the interviews. To ensure reliability in the coding, a second researcher checked and discussed all coded data together with the first researcher. Adjustments were made where disagreement occurred. We applied an inductive approach to extract emerging themes which could be used to answer our research questions. We hit saturation [13, 14] after the 7th participant, whereby no new information was obtained.

2.1 Result of the User Interviews

The analysis produced five themes: Analysis Triggering Mechanism, Result Retrieval Mechanism, Result Display Mechanism, UX Features, and Workflow Integration. In the following, we will talk about how the first four themes of the IDE solution could fit into developers' workflow (the fifth theme).

2.1.1 *Analysis Triggering Mechanism.*

In this section, we introduce how developers expect cloud-based SAST tools to be triggered from their IDEs, how code in their IDEs could be uploaded to the cloud and other expectations on this topic.

Ways of Triggering: The participants mentioned four ways the analysis should be triggered from the IDE: manual (n=8/9)[1], build (n=6/9), fully-automated (n=4/9), and semi-automated (n=1/9). The most mentioned way was *manual*. 8 participants said the analysis should be manually triggered by clicking a button in the IDE or by pressing a key shortcut. Participants would like control over the timing when their code is analyzed as P7 told us: "*I would want to control it by myself. If I would have a simple button to do the analysis, in preparation I will do the testing, before sending the code review I would upload the code to get the review by the machine.*"

Most participants (n=6/9) also would like the analysis to be triggered in the project *build* process. Participants expected it to work this way, since they used other lightweight static analysis tools like FindBugs that can be configured as a build target.

Some participants (n=4/9) mentioned that the analysis should be triggered in a *fully-automated* way. The developers don't want to do anything else to trigger the analysis except writing the code. Real-time feedback from the analysis was expected. P9 explained us the reason: "*I don't want to introduce new behavior [...] If there is a button, during my normal flow, I'm very likely to forget that button.*" P7 mentioned a *semi-automated* way; he expected that the analysis can be configured to run when he presses Control + S to save a file.

Code Uploading: Since the analysis is running remotely, we interviewed the participants to understand how they expected the code to be uploaded to the cloud. The participants mentioned two ways: uploading with analysis triggering (n=6/9) and continuous uploading (n=4/9). The majority of participants (n=6/9) expected the code, especially the changes, to be uploaded when the user triggers the analysis. Their responses indicate that they expected the IDE support will do it for them. Some participants expected the code changes or diffs to be continuously uploaded in the background.

> Developers have two mental models for how cloud-based static analyses should be triggered from the IDE—via active triggering (manual and build) or passive triggering (fully-automated and semi-automated). Developers with the first mental model would like to control the timing when they want feedback from the analysis. They actively search and fix issues once they are done with their coding task. The others prefer not thinking of the timing when they want feedback, they expect the IDE solution to provide feedback right after they make mistakes. Developers want to interact with the analysis as seamlessly as possible in a way that matches their individual workflows.

2.1.2 *Result Retrieval Mechanism.*

All participants expected the IDE support to retrieve analysis results automatically from the cloud. They did not want to download an analysis report from the cloud and import it into their IDEs.

Timing: All participants expected the result to be retrieved to their IDEs directly after the analysis is completed. This can be in the build phase, if the build triggers the analysis; after the user manually triggered the analysis; or while coding if real-time analysis is possible. Three participants mentioned that it would be sufficient if the result could be retrieved before they published code reviews. Although we told participants that the analysis is as time-consuming as CodeGuru Reviewer, their responses indicated that they were not aware of CodeGuru Reviewer's capabilities in terms of performance. They used phrases like "*after several seconds*" and "*at most 10 seconds*". Some developers told us that they usually go on working on other tasks after submitting a code review and get notification emails when the analysis result is ready. They only check the result (of multiple tools) after their teammates review their code. This probably explains why some developers don't have a sense of the analysis time of a specific tool.

> Despite usage of cloud-based SAST tools in the CI/CD process, some developers were not aware of the capabilities and limitations of these tools, e.g., they were unaware how long CodeGuru Reviewer takes to run.

Project Scope: The participants mentioned four project scopes: entire project (n=7/9), diffs (n=6/9), selection (n=4/9) and real-time changes (n=2/9). Scope has a twofold meaning: either they only want the code in respective project scope to be analyzed or they only want to see the result in the scope. Seven of them expected to see the result of the entire project they were working on. Five of the seven also wanted partial code to be analyzed or to only see the result in partial code. Partial code can be diffs or selection (e.g., selected packages, files or methods). We also noticed that the scope often comes with developer's primary goal as P9 explained: "*If I just added some code, I am really interested in modifications I made. [...] If I am working on making the code better, I would want to see all the issues.*" Six participants mentioned that they would like to see the analysis result in their code changes (diffs) if they

[1] We denote the numbers in fractions with the denominator being the sample size.

knew previously they passed all the analysis checks. Only two participants expressed that they would like to see analysis result in real-time changes, e.g.,"*P6: If I write something, the plugin would tell me immediately: are you sure if you want to do this?*"

Which part of the analysis result to be displayed in the IDE depends on what developers' primary goals are. If they are interested in improving overall code quality, showing findings in the entire project is preferred. If their primary goal is to implement a feature, they would like to see only findings that are context-close to the code they are working on (e.g., diffs).

2.1.3 Result Display Mechanism. When talking about how the analysis result should look in the IDE, many participants demonstrated their expectations in their IDEs with compiler errors. All participants suggested to *visually highlight* or *underline* the problematic code and display a *warning message* which explains the issue when the user *hovers* over the line of code. In addition, all participants believed the *severity* of the issue should be included in the warning, because it helps them to prioritize tasks. Some developers expected only critical issues to be shown and they must fail or block the build as P8 told us: "*If there is a failure [...] you have to fix it. However, if there is a warning [...], it is basically ignored. It is useless.*" Many participants (n=4/9) suggested to have *a list view of all issues* which allows *direct navigation to the line of code* when clicking on it. One of them expected to see issues grouped in packages. Three participants would have liked to have quick fixes attached to the warnings. P8 would like to "*have code snippet (vulnerable code)*" attached to the warnings such that he "*can easily see what the problem was*".

Display of Invalid Result: Since the analysis is running in the cloud, by the time the analysis result is back to the user's IDE, the user might have made more changes to the code. Thus, we interviewed developers to understand how they expected these invalid or old results to be handled.

Five of the nine participants expected to see only issues where the code is still in place, otherwise, "*it is misleading*" as P1 told us. Also they did not want to spend time on investigating issues which might not be there anymore due to code changes. P9 suggested: "*the plugin can see this suggestion was for this particular line or file, if this line or file changed, the suggestion will not be displayed.*"

Also, two participants wanted to be informed about the code changes and a rescan (rerun the analysis) to be suggested by the IDE support, as P3 told us: "*developers should be informed if they make changes to the code after they trigger the analysis, they would have to redo the analysis for the changes. They should be informed that the changes after triggering the analysis would't be considered. If we show out-of-date recommendations in the IDE, the developers should be informed that these recommendations are for the past and they might be not valid now.*"

Developers expect to be warned in their code just like the way their IDEs usually show compiler errors. They do not want to spend time on issues which are out-of-date and expect the IDE solution to remind them to rescan.

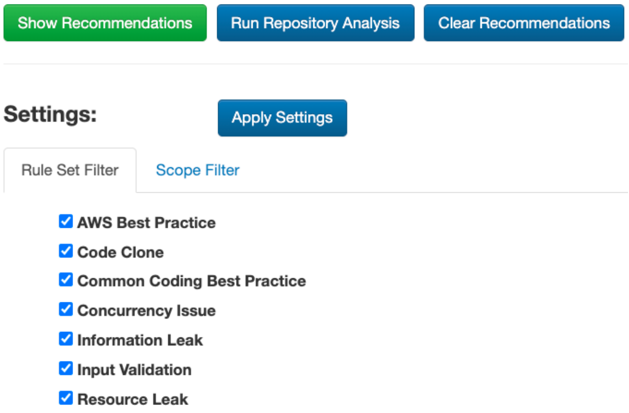
2.1.4 UX Features. The most mentioned feature by participants was quick fix (n=5/9), as P7 describes how he expected it to work: "*you type Alt+Enter, it will offer you some fixes*". Four participants would like to suppress warnings, either false positives or issues which are less severe. P1 expected it to be "*a list of previously suppressed warnings to re-enable them or something like checkboxes*". P3 suggested to import a configuration file containing suppressed warnings as CheckStyle does. P9 would like to "*add a line of comment such as 'disable CodeGuru Reviewer' to the code to suppress.*"

Participants also expected to customize the rule set of the analysis (n=3/9) and even the warning severities (n=3/9) to decide which warnings should be displayed. Both warning suppression and customization of rule set were mentioned as developers talked about features that would be beneficial for their teams.

Developers expect the IDE solution to not only pinpoint issues in their code, but also to fix them. They do not fully trust static analyses based on previous experience. They expect to suppress or prioritize warnings based on their own judgment.

3 PROTOTYPING

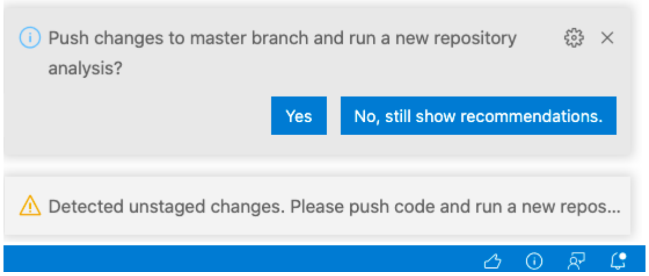
Based on what we learned from the user interviews and the public API of CodeGuru Reviewer [28], we developed an IDE prototype as a Visual Studio Code (VS Code) extension for CodeGuru Reviewer. In the following, we introduce this prototype with respect to the themes derived from the interviews.



Settings:

☑ AWS Best Practice
☑ Code Clone
☑ Common Coding Best Practice
☑ Concurrency Issue
☑ Information Leak
☑ Input Validation
☑ Resource Leak

Figure 1: Control panel of our IDE prototype



Push changes to master branch and run a new repository analysis?

Detected unstaged changes. Please push code and run a new repos...

Figure 2: Reminder notifications asking users to rescan

Analysis triggering, result retrieval and display. The prototype provides a control panel for users to interact with CodeGuru Reviewer in VS Code as shown in Figure 1. The "Show Recommendations" button allows users to view the recommendations (findings) provided by CodeGuru Reviewer directly in the IDE. The prototype automatically compares the local code version to the remote code version and fetches the result to the IDE if a matched analysis is found. If there are local code changes which haven't been uploaded to the remote repository, the prototype displays pop-up notifications to remind the user for a rescan as shown in Figure 2. The user can choose to display the result of the most recent analysis on the current branch with the "No, still show recommendations" button. Only recommendations in unchanged files will be displayed in the IDE, since developers told us they would not want to spend time on issues which might be invalid (see subsubsection 2.1.3). If the user chooses to push code and rescan, a pop-up window will ask for a commit message and code changes will be pushed to the remote repository. After that, the prototype triggers a new analysis in the cloud. A notification will then be shown to tell the user about the estimated analysis time (5 to 10 minutes according to the official documentation) and the result will be automatically retrieved once the analysis is completed. The "Run Repository Analysis" button allows the user to run a new analysis on the remote repository. Similarly, it also reminds the user to push code if there are uncommitted code changes.

Figure 3 shows a typical workflow using our IDE prototype: 1. Developer modifies code and pushes changes to a remote Git repository. 2. Developer clicks the "Run Repository Analysis" button to request CodeGuru Reviewer to run a new analysis on the Git repository. 3. CodeGuru Reviewer receives the request and clones the Git repository. 4. CodeGuru Reviewer analyzes the cloned repository and generates recommendations. 5. The IDE prototype automatically fetches the recommendations once CodeGuru Reviewer finishes the analysis or the developer clicks the "Show Recommendations" button to fetch the recommendations to the IDE. At step 4, while CodeGuru Reviewer is running, the developer can continue working on the code or switch to other tasks.

Recommendations are displayed in a list view at the bottom of the IDE as shown in Figure 4. They are organized in groups according to the source files. From the interviews, we learned that some developers expect issues with fix suggestions to be prioritized. For recommendations with fix suggestions, we used the red marker ⊗ as an attentional cue that the warning was actionable to help developers quickly get to the code. It also indicates these issues are more severe and must be fixed. Although CodeGuru Reviewer itself does not report the severity of an issue, our consultation with the engineers of CodeGuru Reviewer revealed that when the tool provides fix suggestions then it's typically for more severe issues. Yellow markers were used for all other findings. These two markers are the default markers provided by VS Code. We also included weakness types and code snippets in the recommendations, which were not provided by CodeGuru Reviewer before. Clicking on a recommendation in the list navigates to the line of code. The code is highlighted and underlined as shown in Figure 5. Recommendations are also displayed in hover messages when the user hovers over the code. The hover message supports markdown, thus, the URLs to best practices in the recommendations are also clickable. Except quick fix, we addressed all expectations on result display from developers as introduced in subsubsection 2.1.3. Although we would have liked to provide quick fixes, this feature was not supported by CodeGuru Reviewer at the time and most issues cannot be easily fixed by adding/removing/replacing a code string.

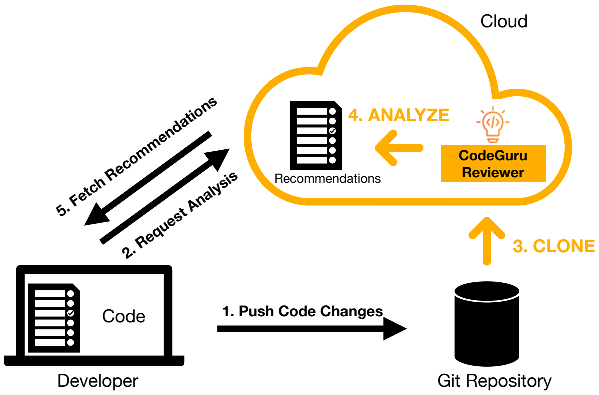
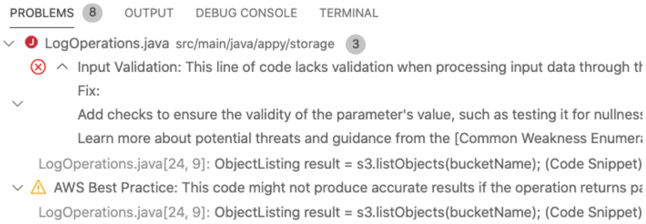
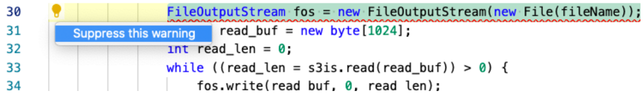


Figure 3: A typical workflow using our IDE prototype



```
LogOperations.java src/main/java/appy/storage 3
  ⊗ Input Validation: This line of code lacks validation when processing input data through tl
    Fix:
    Add checks to ensure the validity of the parameter's value, such as testing it for nullness
    Learn more about potential threats and guidance from the [Common Weakness Enumer
  LogOperations.java[24, 9]: ObjectListing result = s3.listObjects(bucketName); (Code Snippet)
  ⚠ AWS Best Practice: This code might not produce accurate results if the operation returns pa
  LogOperations.java[24, 9]: ObjectListing result = s3.listObjects(bucketName); (Code Snippet)
```

Figure 4: Recommendations are displayed in a list view.



```
            FileOutputStream fos = new FileOutputStream(new File(fileName));
                             read_buf = new byte[1024];
            int read_len = 0;
            while ((read_len = s3is.read(read_buf)) > 0) {
                fos.write(read buf, 0, read len);
```

Figure 5: Warning suppression is shown as a code action.

Other UX features. The prototype was built to support warning suppression and rule set customization, because these were the most wanted features by developers. Warning suppression is provided as a code action (automatic refactoring source code) attached to the recommendation as shown in Figure 5. When the user chooses to suppress a warning, the line of code will not be marked as an issue anymore and a special line code comment "SUPPRESS CodeGuru Reviewer" is automatically added. Users can also manually add the suppression comment to code. No warning will be shown at lines with that comment.

Because we could not change CodeGuru Reviewer to allow its rule set to be customized, we implemented a rule set filter to allow users to select/unselect the weakness types as shown under the settings section in Figure 1. Only recommendations with the selected weakness types would be displayed in the IDE. Developers also mentioned they would like to limit the display of findings to specific packages or classes, so the prototype provided a scope filter to select files or packages they were interested in. VS Code also has a built-in filter for the warning markers such that users can filter the recommendations based on the severities in the issue list. The configuration in the rule set filter and scope filter were locally stored by the prototype.

4 SECOND-ROUND INTERVIEWS

After implementing the prototype, we re-invited the 9 developers we interviewed for a second-round interview. Five (P1, P4, P7, P8 and P9) of them accepted our invitations. These second-round interviews allowed us to fix minor issues prior to the usability test with a larger group of developers to ensure the usability feedback was focused on core issues rather than surface-level design concerns. The interviews were structured by demonstrating the features of the prototype addressing the topics from previous interviews. Each interview was about 30 minutes. After demonstrating a feature, we reminded the participant what she/he told us in the previous interview and asked how the feature differs from what she/he expected. We transcribed and coded the interviews to assess user sentiment (negative and positive) across the themes extracted from the first round of interviews.

Participants were all very positive about how analysis results were automatically retrieved and displayed in the IDE. They also liked the warning suppression and filters feature. Four participants didn't expect the code needs to be pushed to the remote repository to trigger the analysis. P7 explained: "*because for me it was like making dirty commits and I don't like it.*". P8 gave us his reason: "*I am not using test branch at all, I am only using the mainline.*" He felt it was not helpful if he needs to setup a remote branch for his changes to run the analysis before sending a code review.

Although participants were critical about pushing code to the remote Git repository, we could not change the public API of CodeGuru Reviewer to support other channels. Regarding old findings in changed files, P8 said he would expect "*to see something even I change the file, unless I change exactly that line.*" After we explained that there might be case that an issue is fixed when new lines are added to the file, he responded with "*I know the system doesn't know if it is fixed, but I would like to keep track of what was the issue.*" However, the prototype reminds the user to rescan if there are local changes and the findings displayed in the IDE will not be removed unless the user clears them intentionally or requests for an new scan. Before we started the usability test with a larger group of developers, we tested the prototype with six developers and fixed bugs discovered in the interviews and during the test.

> While code uploading mechanism via Git push is widely accepted in CI/CD integration, some developers found it cumbersome in the IDE due to different working habits, e.g., they only commit once per feature or do not use the Gitflow [1] workflow.

5 USABILITY TESTING

Study Design. To test if the IDE solution was an improvement over the web-based solution, we conducted a within-subjects usability test with developers. In comparison to between-subjects studies, it eliminates problems concerning individual differences [6]. We wanted to compare the condition with the IDE prototype to the web-based solution of CodeGuru Reviewer in AWS Console, where users can request analyses for their Git repositories and view recommendations in a web browser. For simplicity, we use *IDE* to represent our IDE prototype and *Web* to represent AWS Console in the following. We prepared two tasks, X and Y. In each task, participants were asked to fix issues in a prepared Java application either with help of the IDE prototype or AWS Console. All issues can be detected by the analysis engine of CodeGuru Reviewer. The prepared applications use AWS services with the AWS SDK for Java [27] and each of them contains 8 issues with different weakness types. The test applications and issue list can be found in [22]. Although the official documentation of CodeGuru Reviewer gives 5-10 minutes as the average analysis time, for the two applications used in our study, the analysis time was just 3 minutes for each.

We applied 4 different treatments to participants as listed in Table 1. T1, T2 are the treatments in which participants first test *IDE*, while in C1, C2 participants started with *Web*. From the study in [10], we learned that the typical length of a working session with a SAST tool of developers is 10-30 minutes (see Table 2 in [10]. The authors refer to SAST tools with "dedicated tools"). Thus, we chose 30 minutes as the session length in our study. In each session, the participants were given maximally 30 minutes to solve the task. After each session, participants were asked to fill out an exit-survey (see the survey in [21]) and take an interview with us to examine how participants used the tools and how the tools affected their behaviors.

Table 1: Four treatments

Treatment	Session 1		Session 2	
	System	Task	System	Task
T1 (n=8/32)	IDE	X	Web	Y
T2 (n=8/32)	IDE	Y	Web	X
C1 (n=8/32)	Web	X	IDE	Y
C2 (n=8/32)	Web	Y	IDE	X

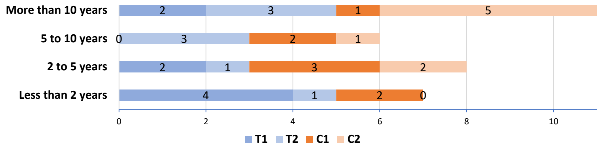


Figure 6: Years of professional coding experience in Java

Participants. We sent 1323 invitation emails to different mailing lists at AWS. In the email, we asked people to fill out a demographic survey if they accepted our invitations. We received 49 survey responses and, based on the responses, we removed participants who do not write code in Java. We chose 32 of them for our study. The participants were located in 9 different countries. 75% (n=24/32) of them never used CodeGuru Reviewer before. Half (n=16/32) of them write code in VS Code and 78% (n=25/32) had written applications before with the AWS SDK. Figure 6 shows their professional coding experience in Java. More than half of them (n=17/32) have at least 5 years experience. We refer to these 32 participants with G1-32.

Study Setup. The participants were assigned round-robin to one of the four treatments. In all treatments, participants were asked to perform the tasks in VS Code. After a brief introduction to the study, the participants were given the tasks in written form. We explicitly told the participants that CodeGuru Reviewer is a cloud-based SAST tool and the expected analysis time to be a few minutes. We ran CodeGuru Reviewer on the test application before each session and made sure that participants saw the CodeGuru Reviewer's findings displayed in either VS Code or in AWS Console before they started doing the task. We also provided participants user guides of the tested tool, i.e., IDE prototype or AWS Console. We told participants they could read them if they had questions. Participants were asked to solve the tasks independently without any help from us. They were also asked to give us clear signals as they started and finished the tasks to record the time.

After each session, participants were asked to fill out an exit-survey containing the 10 System Usability Scale (SUS) questions [5]. In the survey, we also asked participants to evaluate the difficulty of the task using a Likert scale, select features of the tool they thought were helpful, estimate the number of issues they fixed, and answer some open-ended questions.

In each treatment group, we randomly chose 3 participants for monitoring. We asked these 12 participants to share their screen with us and think aloud as they were performing the tasks. The others were not monitored. Because not all participants could take interviews with us after their sessions due to scheduling constraints, we only interviewed 24 of them after they completed the exit-surveys. We asked them about the experience using the tool for the given task and whether the tool worked as they expected.

5.1 Quantitative Analysis

Developers tend to have different working habits when it comes to fixing issues in code as we learned from previous interviews. While some developers tend to validate the fix every time they address an issue, others fix all issues at a time and check them at once. We wanted to investigate how different solutions for a cloud-based SAST tool impact developers' interaction with the tool. Our within-subjects design allows us to test the effect on individual participants. We also wanted to investigate whether our IDE prototype was sufficient to improve developers' performance in code fixing and perceived usability of the tool. With the quantitative data we collected during the test, we answer the following questions:

RQ3: Does the IDE prototype encourage developers to interact more with the cloud-based SAST tool in comparison to the AWS Console?

RQ4: Do developers fix issues more efficiently with the IDE prototype in comparison to the AWS Console?

RQ5: Do developers perceive the IDE prototype to be more usable than the AWS Console?

Behavior (RQ3): Our alternative hypothesis for RQ3 is:

H1: Using the IDE prototype developers will rescan more frequently than using the AWS Console.

To test H1, we logged how many times each participant ran repository analysis successfully. Participants ran the analysis three times more often from the IDE (117 in total) than from the AWS Console (36 in total) as Figure 7 shows. We used the Shapiro-Wilk test [29] to test whether our data is normally distributed. Since the data doesn't distribute normally, we applied a two-tailed Wilcoxon signed-rank test [40] with $\alpha = 0.05$, which is non-parametric and used for repeated measures. Although our hypothesis is one-sided, we used two-tailed testing to ensure the statistical power in both directions [24]. We present the results in Table 2 with participants grouped by their treatments.

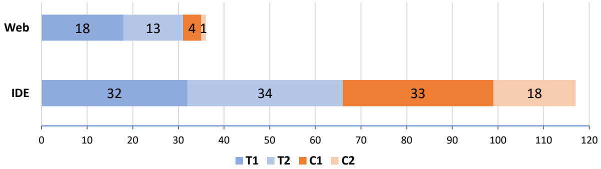


Figure 7: How often did participants rescan (run analysis)? The numbers shown in each bar are the total number of rescans performed by all participants in the associated treatment group in the condition.

Table 2: Results of two-tailed Wilcoxon signed-rank tests with $\alpha = 0.05$. N is sample size without ties. W_{crit} is the critical value for N and α. p-values < 0.05 are marked with *.

(a) Number of Scans				
Group	**N**	**W-value**	**W_{crit} at n (p<0.05)**	**p-value**
T1, T2	15	21	25	* 0.0264
C1, C2	12	7	13	* 0.0121
All	27	46	107	* 0.0006
(b) Average Time to Fix an Issue				
Group	**N**	**W-value**	**W_{crit} at n (p<0.05)**	**p-value**
T1, T2	14	46	21	0.682
C1, C2	14	50	21	0.873
All	28	194	116	0.841
(c) SUS-Score				
Group	**N**	**W-value**	**W_{crit} at n (p<0.05)**	**p-value**
T1, T2	14	27	21	0.110
C1, C2	14	46	21	0.682
All	28	160	116	0.327

The statistics in Table 2 (a) suggests that there is a significant difference ($W < W_{crit}$ and p-values are much smaller than 0.05) in number of scans. Using the IDE prototype developers performed analysis significantly more often than using the web-based solution, despite the fact that the analysis engine was the same and the tasks were similar. This indicates that the IDE solution fits better into developers' workflow. Developers wanted validation of their fixes more often when addressing static findings and the IDE prototype allowed them to run analysis easier.

Based on survey responses to the question "*How did you know that you fixed the issues?*", participants were more confident about the number of fixed issues they estimated in the *IDE* condition. While 8 participants gave the answer saying that they didn't know in the *Web* condition, only 3 participants were not sure as they used the IDE prototype. Later in subsection 5.2 we will introduce

Table 3: IDE Feature Usage

Feature	#Usage	Feature	#Usage
Show Recommendations	318	Rule Set Filter	9
Run Repository Analysis	84	Warning Suppression	2
Clear Recommendations	20	Scope Filter	0

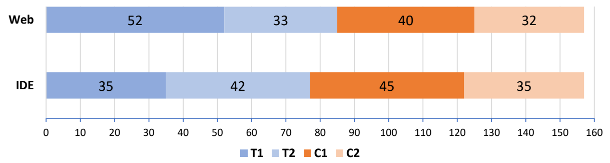


Figure 8: How many issues did participants fix?

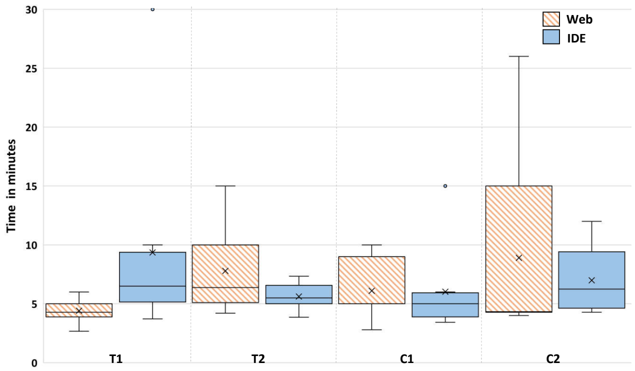


Figure 9: Boxplots for average time participants used to fix an issue. Average values are marked with ×.

the opinions of developers and how they felt their workflows were impacted in the two conditions.

The IDE prototype also logged the total number of usage by all participants for each feature as shown in Table 3. Participants clicked the "Show Recommendations" button 318 times, which is 3.8 times than they clicked the "Run Repository Analysis" button. The huge difference between the usage of these two buttons suggested that participants didn't choose to rescan but opted for displaying old findings as they were doing the tasks. While most participants actively clicked the "Run Repository Analysis" button to rescan (84 times), some participants took suggestions from the IDE prototype (33 times) and selected "Yes" in the pop-up notification to rescan as shown in Figure 2. Other features were rarely used. This is likely due to the short time planned for each session. We asked developers to select features they thought were useful in the survey, 13 of the 32 participants selected warning suppression, 8 for the rule set filter and 5 for the scope filter.

Performance (RQ4): Our alternative hypothesis is:

> **H2**: Given an application containing issues that can be detected by CodeGuru Reviewer, developers using the IDE prototype will be faster than using the AWS Console to fix an issue.

While 20 participants did not finish the task (timed out) in the session using the IDE prototype, the number with the AWS Console is 21. In three of the four groups (T2, C1 and C2), participants fixed more issues in the *IDE* than in the *Web* condition as shown in Figure 8 (An issue was considered fixed if CodeGuru Reviewer didn't report it again.). Surprisingly, participants fixed the same number of issues (157) in total when using the IDE prototype and the AWS Console. This is close to the number of issues participants estimated in the exit-survey, i.e., 165 in the *Web* and 166 in the *IDE* condition.

For each participant, we computed the average time used to fix an issue. Since one participant didn't fix any issue with AWS Console, we excluded this data from the test. Our test failed to reject the null-hypothesis as the statistics shown in Table 2 (b). However, as the boxplots in Figure 9 show, there is a promising time reduction (average and maximum values are lower) in the IDE condition among most groups of participants (T2, C1 and C2). In these groups, developers' performance was also more consistent in the *IDE* condition, since the boxes are smaller.

Usability (RQ5): Although we were comparing a research prototype to a commercial tool designed by professional UX designers, we formulated the alternative hypothesis in an optimistic way.

> **H3**: Developers will rate the IDE prototype with higher SUS-scores.

We evaluated the survey responses to the 10 SUS questions and computed the SUS-scores. The higher the score is, the better the perceived usability. Again, we applied Wilcoxon signed-rank test to the SUS-scores and the result is shown in Table 2 (c). There is no statistically significant difference between the two conditions regarding the SUS-scores.

However, we found out that participants tended to rate the tool they tested later with higher SUS-scores as the boxplots in Figure 10 show. We observed that many participants who started testing the IDE prototype at first (i.e., T1 and T2) were actually not expecting the analysis to be time-consuming. These participants were more confused as the IDE prototype did not display results immediately as they tried to run the analysis. Note that 75% (n=24/32) of the participants never used CodeGuru Reviewer before. In contrast, participants who tested the *Web* condition first had a better sense of the asynchronous nature and the analysis time as they tested the IDE prototype later. We will discuss this further with the qualitative data in subsection 5.2.

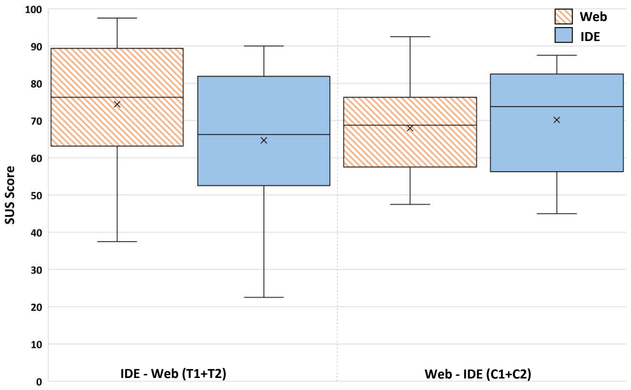


Figure 10: Boxplots for SUS-Scores

Although we pseudo-randomly sampled participants into groups to control for key factors, like experience with pre-existing tools, we had a few issues specifically affecting group T1. The data were affected by four participants in T1 who fixed more issues in the Web condition (less time per fix as shown in Figure 9) and rated extremely low SUS scores in the IDE condition. Participant G34 had more than 10 years of development experience and was very familiar with SAST tools used in CI (including CodeGuru Reviewer), so he was extremely fast in the Web condition (only used half of the time as in the IDE) and fixed more issues. In the interviews, participants G20 and G21 mentioned that they did not understand there was no local analysis and got confused in the *IDE* condition, while in the *Web* condition it was straightforward for them and they could better focus on the task. G2 was observed spending time exploring the features of the IDE prototype rather than fixing issues. He gave a much lower SUS score for *IDE* than for *Web*.

Developers' perceived usability of the IDE prototype was impacted by both their pre-existing expectations (on IDE integration of lightweight static analyses) and familiarity with the cloud-based SAST tool.

5.2 Qualitative Analysis

We coded the after-session interviews and survey responses to "*Tell us what needs to be improved*". We reused the codes from previous interviews introduced in section 2 and section 4 and also added 12 new codes (see codes in [19]). We answer the following questions:
RQ6: How does the IDE prototype differ from the expectations of developers?
RQ7: How did developers think the IDE prototype impacted their workflows?

5.2.1 ***RQ6****: How does the IDE prototype differ from the expectations of developers?* The positive things developers mentioned about the IDE prototype were similar to those we heard from previous interviews introduced in section 4. Moreover, developers were very satisfied with the quality of the analysis result. They felt the recommendations were precise and informative. This is probably the reason why warning suppression was rarely used in the test. In the following, we focus on the major issues of the IDE prototype pinpointed by developers. We also identified some issues in the AWS Console and reported them to the engineering team.

Analysis Triggering Mechanism: The biggest issue mentioned by participants (n=10/32) was uploading code via Git. Some participants felt uncomfortable to push code without a code review. Others felt less confident to push code without testing it locally. Although the test applications provide unit tests, these participants didn't run them at first, instead, they expected to get feedback before testing.

Many participants (n=8/32) were expecting fast local validation of fixes when they clicked on "Run Repository Analysis" button, as G8 said "*when I modified the code, it was strange to see the squiggly lines here and there saying there was an error.*" Although there were pop-up notifications shown in the IDE mentioning the analysis time and suggesting rescan, some participants seemed to pay little attention to those pop-ups. These participants are mostly from the groups who tested the IDE prototype at first.

Developers expected IDE support to allow usage of cloud-based SAST tools before their code goes into the next phases (test, CI/CD) in the development lifecycle. They wanted the analysis of code without going through steps such as uploading it.

Result Retrieval and Display Mechanisms: Most problems came from the analysis time and poor indication of the analysis progress in the IDE. CodeGuru Reviewer needs about 3 minutes for reanalyzing each test application, which is shorter than the official average analysis time (5-10 minutes) given by CodeGuru Reviewer. Still, it was "*painfully slow to the point I was worried the plugin was unresponsive*" as G15 told us. As mentioned above, pop-ups were not sufficient for informing participants about the analysis time. As G8 told us, although he read the pop-ups he thought it was a "*generic message*" and "*didn't consider it would be the exact time*". More than one third of the participants (n=13/32) we interviewed expected to see a progress bar or a dynamic display of analysis status in the IDE. They wanted to "*see what it (the service) is currently doing, so not just all of a sudden, a pop-up comes up saying the result is ready.*" This is also the pain point in the AWS Console, since there is no progress indication either.

Using the IDE prototype, many participants told us they kept working after started a new scan. Most confusion came from displaying old findings. Some participants didn't rescan, but chose to see them in the IDE and clicked the "Show Recommendations" button multiple times as they were fixing the issues. They thought this button performed local validation of a fix, although this button actually checks code version and only displays the findings in unchanged files. This led to "*the problem (finding) disappeared and I had trouble to get them again.*" as G12 described. This can probably be resolved by reading the user manual or a mechanism that checks local changes better. Other participants felt "*the outdated messages are annoying, I'd rather have them disappear when a line gets updated, to invalidate them.*", since they were expecting the prototype to be as reactive as lightweight static analysis tools.

Pop-ups were less effective in educating developers on mechanisms built in the IDE solution. If the analysis is not returning results instantaneously or developers' activities in the IDE are not blocked, the display of old findings and unpredictable waiting time for new findings are "deal breakers". This suggests developers need clear visual cues (e.g., progress bars) to understand when the analysis is running and if findings are outdated.

UX Features: Five participants mentioned that they were not aware that the analysis was running remotely, thus, they were expecting real-time feedback due to previous experience. As G20 told us that he didn't think about the analysis was running in the cloud. His "*initial perception is that this is going to be some sort of static analysis tool*" and he was "*expecting it to be a similar experience to other static analysis tools*" he has used. These participants suggested more obvious visual indication for the asynchronous nature of the IDE solution.

Another feature that was missed by participants for both the IDE prototype and AWS Console is a way to keep track of addressed issues. G9 told us he was using the rating buttons (thumb up and down symbol) in the AWS Console "*as almost a checklist. I know which I have done because I marked them helpful.*" Although quick fix was the most mentioned feature in our first user interviews introduced in section 2, only 3 participants mentioned it this time. Developers understood that it was harder to provide quick fixes for more complex issues detected by a SAST tool than a linter. Two participants suggested that the IDE prototype should forbid or auto-cancel multiple scans on the same commit, since "*it's a wasted action*".

Even though most developers were aware that the cloud-based SAST tool performs a deeper analysis and acknowledged that the analysis takes longer, they still complained about the waiting time and that findings and code ran out of sync. This suggests that using the same visual components for the cloud-based analysis that are also used by lightweight local static analyzers (e.g., problem-list windows, error markers on code) may create unrealistic expectations about the behavior of the tool.

5.2.2 ***RQ7****: How did developers think the IDE prototype impacted their workflows?* As we show for RQ3 the IDE prototype impacts developers' behavior in fixing code, the qualitative data also indicates the same. Although developers interacted with the same analysis engine, they approached the tasks differently in the two conditions. A participant used a metaphor to express the different feelings: "*The thing in the AWS Console felt like integration test. Having it in the IDE was like unit test.*"

In the AWS Console, because it is in a browser, participants felt a disconnect "*between running the analysis and editing*". A few participants perceived the list of findings as a task list as G9 told us: "*I saw the task list and I went to work on that code. It just didn't click for me that I can go back to the AWS Console and rerun the analysis.*" They felt that they were supposed to pick a workflow in which they would only rescan once they addressed all issues. Not seeing the result immediately was less frustrating, because it was clear that there was no synchronization. While some participants chose to address all issues at once, others felt their workflows were paused in the *Web* condition as G29 told us: "*I was somehow encouraged to wait to see what happens. I felt that if go back working, I would not be aware when the execution finishes.*"

Using the IDE prototype, without switching between the browser and the IDE, some developers rescanned more often. A participant who addressed all issues at once in the *Web* condition said: "*(in the IDE) It was like I saw the thing turned red, I fixed it, kept iterating on it until error-free, then I move on to the next one. [...] So I was expecting some feedback. I changed something, hit on 'Run Repository Analysis'[...]*" Another participant told us he felt encouraged to change code and rescan even before the previous scan is completed such that he could work more efficiently.

Using the IDE prototype, developers wanted to validate their fixes more often and felt encouraged not to wait for the analysis execution, but continue working on fixing other issues.

Despite of all the problems identified in the IDE prototype, many participants expressed that they would prefer the IDE support to interact with cloud-based SAST tools. G21 told us he "*would prefer IDE, because less time wasted having to go through other screens. I can push code, and let analysis run on branch, while continuing workflow in the IDE. Less context switching.*" G1 also preferred the prototype but wished it was more interactive:"*It runs analysis on the file you are working on and tells you if you fixed it correctly or not.*" Also G22 thought that "*the IDE integration is the better path, because it's closer to the activity being performed: writing code.*" Despite the delay of the analysis, he would "*much rather see a list of suggested problems/fixes in my editor than changing screens back-and-forth.*"

6 THREATS TO VALIDITY

External Validity: We conducted our study with developers of a single company. Among them, only a few participants were female. This may lead to limited generalizability of our findings to the whole developer community. However, the participants were located in 9 different countries, have years of professional experience, and work on different kinds of products. Furthermore, we only studied the effect of IDE integration for one cloud-based SAST tool and one IDE. As we demonstrated, the response time of the SAST tool is a major factor, so our findings cannot be generalized to tools that are significantly faster or slower. A follow-up study can determine this effect by artificially introducing delays when retrieving findings. Our prototype is based on language server protocol [18, 23] that integrates with most modern IDEs, so we expect the impact of the IDE choice to be minimal, but developers' familiarity with an IDE could affect the study.

We only compared the IDE solution to the web-based solution of CodeGuru Reviewer regarding repository analysis. We did not consider the impact on development lifecycle management with the issue board in the web-based solution. It is likely that project managers and team leaders would have a higher preference for the web-based solution. Moreover, we did not consider cost, security, and trust. Some participants were critical about pushing code for a rescan and proposed to hide the action, however, real customers of CodeGuru Reviewer might not want their code to be uploaded silently due to security concerns.

Internal Validity: The first threat is the session time. Some participants told us they felt stressed and they did not have enough time to fix all issues. While the available time limited the performance of some participants, it also simulated the pressure of software development on tight deadlines, e.g. before releases. We also observed that some participants were less motivated to fix code, but more interested in playing with the features of the tools.

Another threat was attrition. We had four developers (who did the first interviews with us) not attend the second-round interviews and two (their data are not included in the paper) did not participate in the usability test. We are aware that our findings are likely to be based on a biased sample of developers who have higher motivation to use static analysis tools or cloud services. Moreover, the tasks in the usability test were artificial. Due to unfamiliarity with the projects or the used Java libraries, some participants may have performed worse than in real development situations. However, we only selected developers who have professional experience in Java and the majority (n=25/32) of them used AWS SDK before.

Regarding the impact of developers' familiarity with the IDE, we applied Wilcoxon-singed rank test to the sample grouped by tasks and grouped by the experience with VS Code, the result indicated there was no significant difference between the groups. Regarding the issues detected by CodeGuru Reviewer, they were all true positives.

Lastly, the IDE prototype was not designed by professional UX designers, but researchers. It is likely that developers would perceive a significant improvement of the usability using a professionally designed IDE solution in comparison to the web-based solution. Although we were comparing a prototype to a web application with real customers, our result indicates that the prototype is not worse.

7 RELATED WORK

The usability of static analysis tools has been studied by many researchers. Johnson et al. interviewed experienced developers to understand why developers were not widely using static analysis tools [16]. They found out that false positives and bad warnings were the major reasons for developers' dissatisfaction. Christakis and Bird surveyed developers at Microsoft to understand what developers want and need from static analysis tools [8]. Their study shows that developers would like static analysis tools to detect more critical issues for them such as security or concurrency issues and display the findings directly in their IDEs. Beller et al. studied the usage of static analysis tools on open source projects [3]. They found out that most open source developers only use static tools sporadically and they need to be made aware of the benefits of using these tools. Vassallo et al. studied developers' behavior using static analysis tools over different development stages [36]. They found out that severity is the most important factor for developers in prioritizing issues to fix, which was confirmed in our study. Steidl et al. suggested to prioritize issues that are easy to refactor [30]. Their study indicates prioritizing by low refactoring costs matches greatly the developers' opinions. In our study, we also heard expectation of such prioritization mechanism from some developers. A more recent study from Nguyen Quang Do et al. took a user-centric approach to understand why developers use static analysis tools and which decision they make when using these tools [10]. According to their study with developers at Software AG, IDEs are still the ideal reporting locations wanted by developers. However, we observed that there exists a disconnect between the typical usage of cloud-based SAST tools in CI/CD and developers' wish to interact with them earlier in the lifecycle, in their IDEs. Our work focuses on exploring how IDE support for cloud-based SAST tools that are typically used in CI/CD should be designed. We approached the exploration from developers' perspective with a user study. We found out that developers expected more than just seeing the findings of these tools in their IDEs.

In recent years, we see an increased interest in studies that apply static analysis tools at scale [4, 15, 17, 35, 41]. Facebook's static analyzer infer has detected over 100,000 issues that have been resolved by Facebook's developers since 2014 [9]. As reported by Google [25], their static analysis platform Tricorder could analyze 50,000 code review changes per day. More than 5,000 issues per day were tagged to be fixed by developers. These studies discuss tools that are integrated in the code review process. Our work builds on the results of these papers and asks the question how we can give developers access to cloud-based SAST tools directly through their IDEs, and if this improves developers' workflows. We share challenges and lessons learned in the exploration that can be beneficial for suppliers that wish to build such IDE support.

Many researchers have studied the impact of cloud services on the user experience [33, 39]. Kaisa Väänänen-Vainio-Mattila et al. studied the user perceptions of Wow—a positive user experience when using cloud services [34]. They proposed a few design implications for achieving Wow such as pushing dynamic features to keep the user stimulated. Tang et al. interviewed users of file synchronizing and sharing services to understand the cloud-based user experience [32]. They found out that users' understanding and usage of cloud functionalities are limited by their existing practices. Similarly, we also learned that developers' expectations of IDE support for cloud-based SAST tools were affected by their awareness of the limitations of these tools, and their previous experience with lightweight analysis tools. Developers' overexpectations hugely impacted the perceived usability when interacting with our IDE prototype. Through a usability test, we identified important design elements and mechanisms required for a better tool support.

8 CONCLUSION

To investigate how IDE support for cloud-based SAST tools should be designed, we conducted a multiple-staged user study. We first interviewed developers at AWS to understand their expectations. Developers' feedback indicates that they expected the IDE support for cloud-based analyses to behave similar to the lightweight static analysis tools they already use in their daily work. Their responses also indicate that they have limited understanding of the capabilities and limitations of SAST tools. Guided by the user interviews, we developed an IDE prototype that was positively confirmed by the same group of developers. We tested this IDE prototype on 32 developers. This usability test showed that allowing developers to interact with a cloud-based SAST tool through their IDE significantly increased their interaction with the tool, i.e., they ran the analysis much more frequently than using the web-based solution. This might impact the code quality in a long time span. Moreover, we found promising reduction in fix time even in our small-size study. A larger longitudinal study on this impact should be conducted in the future. However, we also found out that reusing the same visual components for the cloud-based analysis that are also used by lightweight static analyzers (e.g., problem-list windows, error markers on code) created confusion and that developers need clear visual cues to understand the asynchronous nature of cloud-based analyses.

ACKNOWLEDGMENTS

This research was partially supported by the research training group Human Centered Systems Security (NERD.NRW) sponsored by the state of North Rhine-Westphalia in Germany.

REFERENCES

[1] Atlassian. 2021. Gitflow Workflow. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

[2] AWS. 2019. CodeGuru Reviewer. https://aws.amazon.com/codeguru

[3] Moritz Beller, Radjino Bholanath, Shane McIntosh, and Andy Zaidman. 2016. Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*. IEEE Computer Society, 470–481. https://doi.org/10.1109/SANER.2016.105

[4] Claude Bolduc. 2016. Lessons learned: Using a static analysis tool within a continuous integration system. In *2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 37–40. https://doi.org/10.1109/ISSREW.2016.48

[5] John Brooke. 1996. Sus: a "quick and dirty'usability. *Usability evaluation in industry* 189 (1996).

[6] Gary Charness, Uri Gneezy, and Michael A Kuhn. 2012. Experimental methods: Between-subject and within-subject design. *Journal of Economic Behavior & Organization* 81, 1 (2012), 1–8. https://doi.org/10.1016/j.jebo.2011.08.009

[7] Checkmarx. 2021. Checkmarx. https://www.checkmarx.com

[8] Maria Christakis and Christian Bird. 2016. What Developers Want and Need from Program Analysis: An Empirical Study. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (Singapore, Singapore) *(ASE 2016)*. Association for Computing Machinery, New York, NY, USA, 332–343. https://doi.org/10.1145/2970276.2970347

[9] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. 2019. Scaling Static Analyses at Facebook. *Commun. ACM* 62, 8 (July 2019), 62–70. https://doi.org/10.1145/3338112

[10] Lisa Nguyen Quang Do, James Wright, and Karim Ali. 2020. Why do software developers use static analysis tools? a user-centered study of developer needs and motivations. *IEEE Transactions on Software Engineering* (2020). https://doi.org/10.1109/TSE.2020.3004525

[11] Jennifer Fereday and Eimear Muir-Cochrane. 2006. Demonstrating Rigor Using Thematic Analysis: A Hybrid Approach of Inductive and Deductive Coding and Theme Development. *International Journal of Qualitative Methods* 5, 1 (2006), 80–92. https://doi.org/10.1177/160940690600500107 arXiv:https://doi.org/10.1177/160940690600500107

[12] GitHub. 2021. LGTM. https://lgtm.com

[13] Barney G Glaser, Anselm L Strauss, and Elizabeth Strutzel. 1968. The discovery of grounded theory; strategies for qualitative research. *Nursing research* 17, 4 (1968), 364. https://doi.org/10.1177/003803856900300233

[14] Greg Guest, Arwen Bunce, and Laura Johnson. 2006. How many interviews are enough? An experiment with data saturation and variability. *Field methods* 18, 1 (2006), 59–82. https://doi.org/10.1177/1525822X05279903

[15] Nasif Imtiaz, Brendan Murphy, and Laurie Williams. 2019. How do developers act on static analysis alerts? an empirical study of coverity usage. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 323–333. https://doi.org/10.1109/ISSRE.2019.00040

[16] Brittany Johnson, Yoonki Song, Emerson R. Murphy-Hill, and Robert W. Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, David Notkin, Betty H. C. Cheng, and Klaus Pohl (Eds.). IEEE Computer Society, 672–681. https://doi.org/10.1109/ICSE.2013.6606613

[17] Rahul Kumar, Chetan Bansal, and Jakob Lichtenberg. 2016. Static Analysis Using the Cloud. *Electronic Proceedings in Theoretical Computer Science* 228 (Oct 2016), 2–15. https://doi.org/10.4204/eptcs.228.2

[18] Linghui Luo, Julian Dolby, and Eric Bodden. 2019. MagpieBridge: A General Approach to Integrating Static Analyses into IDEs and Editors (Tool Insights Paper). In *33rd European Conference on Object-Oriented Programming, ECOOP 2019, July 15-19, 2019, London, United Kingdom (LIPIcs, Vol. 134)*, Alastair F. Donaldson (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 21:1–21:25. https://doi.org/10.4230/LIPIcs.ECOOP.2019.21

[19] Linghui Luo, Martin Schäf, Daniel Sanchez, and Eric Bodden. 2021. List of codes and their definitions. https://github.com/linghuiluo/FSE21Study/blob/main/ListOfCodes.pdf

[20] Linghui Luo, Martin Schäf, Daniel Sanchez, and Eric Bodden. 2021. List of questions asked in user interviews. https://github.com/linghuiluo/FSE21Study/blob/main/ListOfInterviewQuestions.pdf

[21] Linghui Luo, Martin Schäf, Daniel Sanchez, and Eric Bodden. 2021. Questions in the survey. https://github.com/linghuiluo/FSE21Study/blob/main/SurveyQuestions.pdf

[22] Linghui Luo, Martin Schäf, Daniel Sanchez, and Eric Bodden. 2021. Test applications and issue list. https://github.com/linghuiluo/FSE21Study/tree/main/tasks

[23] Microsoft. 2021. Language Server Protocol. https://microsoft.github.io/language-server-protocol

[24] Graeme D Ruxton and Markus Neuhäuser. 2010. When should we use one-tailed hypothesis testing? *Methods in Ecology and Evolution* 1, 2 (2010), 114–117.

[25] Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, and Ciera Jaspan. 2018. Lessons from Building Static Analysis Tools at Google. *Commun. ACM* 61, 4 (March 2018), 58–66. https://doi.org/10.1145/3188720

[26] Douglas Schuler and Aki Namioka. 1993. *Participatory design: Principles and practices*. CRC Press.

[27] Amazon Web Services. 2021. AWS SDK for Java. https://aws.amazon.com/sdk-for-java

[28] Amazon Web Services. 2021. Public API of CodeGuru Reviewer. https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/services/codegurureviewer/package-summary.html

[29] S. S. SHAPIRO and M. B. WILK. 1965. An analysis of variance test for normality (complete samples). *Biometrika* 52, 3-4 (dec 1965), 591–611. https://doi.org/10.1093/biomet/52.3-4.591

[30] Daniela Steidl and Sebastian Eder. 2014. Prioritizing maintainability defects based on refactoring recommendations. In *22nd International Conference on Program Comprehension, ICPC 2014, Hyderabad, India, June 2-3, 2014*, Chanchal K. Roy, Andrew Begel, and Leon Moonen (Eds.). ACM, 168–176. https://doi.org/10.1145/2597008.2597805

[31] Synopsys. 2021. Coverity Scan. https://scan.coverity.com

[32] John C. Tang, Jed R. Brubaker, and Catherine C. Marshall. 2013. What Do You See in the Cloud? Understanding the Cloud-Based User Experience through Practices. In *Human-Computer Interaction - INTERACT 2013 - 14th IFIP TC 13 International Conference, Cape Town, South Africa, September 2-6, 2013, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 8118)*, Paula Kotzé, Gary Marsden, Gitte Lindgaard, Janet Wesson, and Marco Winckler (Eds.). Springer, 678–695. https://doi.org/10.1007/978-3-642-40480-1_47

[33] Ilkka Uusitalo, Kaarina Karppinen, Arto Juhola, and Reijo Savola. 2010. Trust and cloud services-an interview study. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. IEEE, 712–720. https://doi.org/10.1109/CloudCom.2010.41

[34] Kaisa Väänänen-Vainio-Mattila, Jarmo Palviainen, Santtu Pakarinen, Else Lagerstam, and Eeva Kangas. 2011. User perceptions of Wow experiences and design implications for Cloud services. In *Designing Pleasurable Products and Interfaces, DPPI '11, Milano, Italy, June 22-25, 2011*, Alessandro Deserti, Francesco Zurlo, and Francesca Rizzo (Eds.). ACM, 63:1–63:8. https://doi.org/10.1145/2347504.2347573

[35] Carmine Vassallo, Fabio Palomba, Alberto Bacchelli, and Harald C Gall. 2018. Continuous code quality: are we (really) doing that?. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 790–795. https://doi.org/10.1145/3238147.3240729

[36] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, A. Zaidman, and H. C. Gall. 2018. 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)*. 38–49. https://doi.org/10.1109/SANER.2018.8330195

[37] Veracode. 2021. Veracode. https://www.veracode.com/products/binary-static-analysis-sast

[38] Veracode. 2021. Veracode Static For IDE. https://help.veracode.com/r/api_eclipse

[39] Lizhe Wang, Jie Tao, Marcel Kunze, Alvaro Canales Castellanos, David Kramer, and Wolfgang Karl. 2008. Scientific Cloud Computing: Early Definition and Experience. In *10th IEEE International Conference on High Performance Computing and Communications, HPCC 2008, 25-27 Sept. 2008, Dalian, China*. IEEE Computer Society, 825–830. https://doi.org/10.1109/HPCC.2008.38

[40] R. F. Woolson. 2008. *Wilcoxon Signed-Rank Test*. American Cancer Society, 1–3. https://doi.org/10.1002/9780471462422.eoct979 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780471462422.eoct979

[41] Fiorella Zampetti, Simone Scalabrino, Rocco Oliveto, Gerardo Canfora, and Massimiliano Di Penta. 2017. How open source projects use static code analysis tools in continuous integration pipelines. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 334–344. https://doi.org/10.1109/MSR.2017.2