Tool Support for Spectrum-based Fault Localization during Continuous Integration

Erickson Lima Barbosa da Silva
Escola de Artes, Ciências e Humanidades, Universidade de São Paulo
São Paulo, SP, Brazil
ericksonlbs@usp.br

Roberto Paulo Andrioli Araujo
Escola de Artes, Ciências e Humanidades, Universidade de São Paulo
São Paulo, SP, Brazil
roberto.andrioli@gmail.com

Marcos Lordello Chaim
Escola de Artes, Ciências e Humanidades, Universidade de São Paulo
São Paulo, SP, Brazil
chaim@usp.br

ABSTRACT

Program debugging is one of the most time consuming activities carried out by developers because, in large extent, it is performed in *ad hoc* fashion. Significant research efforts have been directed to the development of new debugging techniques, in special, fault localization techniques. Spectrum-based Fault Localization (SBFL) is a debugging technique that has been experimented on programs similar to those developed in industry with promising results. To support the automated application of SBFL, tools have been developed, yet in the academic realm. Few of them, though, aim to support SBFL in the context of the Continuous Integration (CI) practice. We present the Jaguar Portal platform, designed and developed to receive, store, and provide access to SBFL information in CI environments. The platform encampsulates the Jaguar 2 tool, which is responsible for executing the SBFL technique during the build process in the CI pipeline on GitHub. Furthermore, it provides web interfaces that display SBFL analyses with code snippets and suspicious line markings. We believe Jaguar Portal is a step towards the adoption of SBFL techniques in industrial settings.
Demo video: https://www.youtube.com/@JaguarPortalSBFL.

KEYWORDS

Debugging, Fault localization, Spectra, Continuous Integration

1 Introduction

Testing and debugging are related tasks in the software development life cycle that are among the most costly. To reduce the time spent by developers on defect localization, various techniques have been studied and proposed in recent decades. Zakari et al. [39] indicate that 41% of research papers published in the field of software defect localization use the Spectrum-based Fault Localization (SBFL) technique.

SBFL utilizes code coverage from executed tests, known as spectra, to rank parts of the code (class, method, line, branch, association definition use) that are more likely to be faulty, i.e., have a higher suspicion. The rankings generated by SBFL techniques guide the developer in their search for defects. Commonly, the spectra that are exercised more frequently by failing tests and less exercised by passing tests are positioned at the top of the ranking.

In recent years, tools to support the application of SBFL have been developed. Examples of these tools include CharmFL [16], for programs written in the Python language; FLAVS [36], for programs written in the C# language; and Flacoco [33], GZoltar [4], and Jaguar [30], for programs written in the Java language. The tools that support SBFL, for the most part, are plugins for Integrated Development Environments (IDEs) or have a command-line interface. Therefore, they were not developed aimed to be used during the continous integration practice.

Continuous Integration (CI) is an agile practice that is largely adopted by the industry [25] in which developers add or modify code in a common repository, often multiple times in a single day. One of the major advantages of using CI is to anticipate the identification of defects through tests, which can be automatically triggered after new source code changes are committed. The composition of various steps to be executed through software and scripts that support this practice is called a CI pipeline.

We introduce the Jaguar Portal, a platform built to collect, store, and deliver SBFL data in continuous integration (CI) environments. It uses Jaguar 2, a tool that applies an SBFL technique during the build phase of GitHub-based CI pipelines. The platform also offers user-friendly web interfaces that visualize SBFL results, including annotated code snippets and marked suspicious lines. We believe that Jaguar Portal is a step towards automated support for fault localization in industrial settings.

The remainder of the article is organized as follows. The next section describes the concepts related to SBFL techniques and CI. Section 3 presents the related work. In Section 4, we describe the preexisting tools used in the solution designed and developed for Jaguar Portal, which is described in Section 5. An example demonstrating the use of Jaguar Portal to locate a real fault in a program similar to those developed in industry is presented in Section 6. Finally, we draw our conclusions and present future work in Section 8.

2 Background

This section presents the main concepts associated with program spectra, SBFL, and CI.

2.1 Program Spectra

Program spectra are an execution profile that indicates which parts of a program are active during a run [2]. Typically, it consists of counters and/or markers that indicate the passage through certain parts of a program. Therefore, program spectra are derived from structural testing because it requires knowledge of the code executed for the generation of spectra.

In the literature, several synonyms for spectra can be found, such as code coverage, test data, dynamic information, execution trace, execution path, path profile, and execution profile [9]. To gain an overview of the existing spectra, Table 1 considers nine types of spectra [15].

Table 1: A catalog of program spectra [15]

Name	Description
Branch-hit	conditional branches that were executed
Branch-count	number of times each conditional branch was executed
Path-hit	path (intraprocedural, loop-free) that was executed
Path-count	number of times each path (intraprocedural, loop-free) was executed
Complete-path	complete path that was executed
Data-dependence-hit	definition-use pairs that were executed
Data-dependence-count	number of times each definition-use pair was executed
Output	output that was produced
Execution-trace	execution trace that was produced

2.2 Spectrum-Based Fault Localization

Spectrum-Based Fault Localization (SBFL) comprises a set of techniques that utilize test execution information to identify code segments suspected to be more likely to contain defects [2].

Typically, SBFL support tools perform their work in three stages. The first stage involves collecting program spectra during test execution; therefore, depending on the tool used, different spectra can be used, as mentioned in Table 1.

In the second stage, association metrics are used to calculate the *suspiciousness* of the collected spectra. The suspiciousness consists of values assigned to the spectra and the code segments associated with them. Some of the metrics include: DRT [5], Jaccard [1], Kulczynski2 [23], McCon [23], Minus [38], Ochiai [1], O^P [24], Tarantula [20], Wong3 [37], and Zoltar [18].

In the third and final stage, the code segments from the spectra are ranked in descending order based on their suspiciousness. As a result, the most suspicious code segment of being defective is placed in the first position. Figure 9 presents the mapping of the SBFL ranking of suspicious lines in the code so that the most suspicious lines are colored red and the least suspicious blue.

2.3 Continuous Integration

Continuous Integration (CI) is a software development practice in which team members frequently integrate their code, leading to multiple integrations per day. Each integration is verified by an automated build (including tests) to detect integration issues as early as possible. Many consider this approach to bring about a significant reduction in integration problems and to enable a team to keep the software consistently cohesive [14].

Continuous Integration (CI) encompasses various functionalities, with the most common ones being: Version Control, Build Integration, Testing, and Delivery. The purpose of a version control system is to manage changes to source code and other software artifacts (e.g., documentation) using controlled access to a repository. This provides a "single source of truth" where all source code is available on one central location. The version control system allows you to go back in time and retrieve different versions of source code and other files [10]. Additionally, this system detects changes in the repository and triggers a process on the CI server, called Integration Build. In this process, the server executes a pre-configured build script that can trigger various actions, such as compiling, running tests, and publishing a software version, among others.

The build script can trigger the execution of automated tests within the CI server, using testing frameworks such as xUnit [13], NUnit [12], JUnit [35], among others, depending on the programming language used by the software under test. During the execution of these tests, tools such as JaCoCo [6] and BA-DUA [31] can be invoked to capture test coverage according to a given criterion (e.g., all nodes, all branches, all uses [29]), thus providing the necessary information for an SBFL process, which can be executed subsequently in case any test fails.

Typically, CI tools have feedback mechanisms for the development team, especially regarding Integration Build, which is performed after a code change commit. It is common for this feedback to occur via email and also through web interfaces that developers can access for analysis if the build encounters errors or if any tests fail during its execution on the server.

3 Related Work

The academic community has been designing and developing support tools for the use of SBFL, such as GZoltar [4], Jaguar [30], Flacoco [33], CharmFL [16], FLAVS [36], among others. These tools can be used at various stages of the software development lifecycle. Some operate during software coding through plugins installed in IDEs such as Visual Studio Code [22], PyCharm [19] or Eclipse [11]. GZoltar, CharmFL, FLAVS, and Jaguar are tools that provide this functionality.

There are tools that offer their functionality through command-line interfaces, allowing their execution outside of an IDE. This feature allows GZoltar, Jaguar, and Flacoco to be used to automate the testing and debugging process. More recently, GZoltar and Jaguar 2 [32] were integrated as Maven plugins to support SBFL for programs written in the Java language, thus facilitating their execution in a CI environment.

Paiva et al. [26] developed the only tool that incorporates SBFL data in CI pipelines, using GitHub as the integration tool and GZoltar library as the SBFL analysis tool. Jaguar Portal differentiates by encapsulating GitHub, Jaguar 2, and Maven [28] (see Section 4) and by offering a graphical interface that visually presents SBFL data, including color-coded indicators to highlight code segments with different levels of suspiciousness, as well as allowing navigation among files that potentially host bugs.

To guide the construction of Jaguar Portal, we compared tools supporting SBFL, namely GZoltar, Flacoco, Jaguar and Jaguar 2 [34]. These tools were selected because (1) they are mature and have already been employed in academic experiments, and (2) they support the Java programming language, widely used for developing software systems in industry. We evaluated and compared the tools regarding possible integrations, SBFL characteristics, and user-centered functionalities [34]. To analyze execution times and memory usage, an experiment was conducted. The data collected indicate that the adoption of SBFL within a CI pipeline is feasible, with acceptable increases in execution time and memory consumption. Among the evaluated tools, Jaguar 2 proved to be the most promising in terms of execution time with acceptable memory usage; however, its interface still needs improvement. GZoltar has a better interface but requires performance enhancements and better integration with the CI pipeline.

In what follows, we present Jaguar Portal as well as an example of its use.

4 Pre-existing Tools

In the composition of the Jaguar Portal platform, resources from preexisting and publicly available tools were employed, namely, GitHub, Maven, and Jaguar 2. These resources are described in the following.

4.1 GitHub

GitHub [17] is an online platform that provides hosting services for source code repositories. It has become one of the largest and most influential repositories for both open-source and private software projects in the world. Developers and development teams use it extensively not only as a repository to integrate source code, but also as a tool to collaborate, manage projects, and track software development progress.

One can configure a continuous integration pipeline with GitHub, which typically includes steps such as compilation, automated testing, and packaging, among others. These pipelines are referred to as GitHub Actions. The steps configured within this pipeline are called *actions* and can be made available and used through a marketplace[1].

For configuring these actions, if there is a need to handle sensitive data such as passwords, application programming interface (API) keys, or access tokens, the platform provides a password repository called *Secrets*. This repository ensures that critical information is not accidentally exposed and allows integration with external services or deployment in secure environments without exposing credentials directly in the source code.

4.2 Maven

Maven [28] is a build automation and project management tool for Java software projects. It supports developers in the processes of building, compiling, packaging, testing, and distributing their projects. The functionalities of Maven can be extended through modular components called plugins. These plugins can be added to the project to perform specific tasks within the project's lifecycle. Plugins are configured in the `pom.xml` (Project Object Model) file, which describes the project and its dependencies, and are automatically downloaded from Maven's central repository when needed. With plugins, Maven becomes highly adaptable and flexible to meet various project construction and management needs.

4.3 Jaguar 2

Jaguar 2 (JAva coveraGe faUlt locAlization Ranking 2) [32] is an open source tool that uses control flow and data flow information for spectrum-based fault localization (SBFL) in Java applications. It consists of a *Listener* registered in a Maven project through the *maven-surefire-plugin*[2], its *Providers* for collecting test coverage data, and its *Reports* for recording results.

The *Listener* is registered in the project and, during the build process, listens to events triggered by Junit [35]. The tool has two different *Provider* implementations. The first uses JaCoCo [6] to collect control flow coverage, and the second uses BA-DUA [3] to collect data flow coverage.

For computing SBFL rankings, the tool provides ten different heuristics: DRT [5], Jaccard [1], Kulczynski2 [23], McCon [23], Minus [38], Ochiai [1], O^P [24], Tarantula [20], Wong3 [37], and Zoltar [18]. To record the SBFL rankings generated by the tool, Jaguar 2 provides *Reports* in physical file formats such as CSV, XML, and JSON, and supports integration for publishing results directly on the Jaguar Portal platform through its Web API. The source code of the Jaguar 2 application is available on https://github.com/saeg/jaguar2.

5 Jaguar Portal

The Jaguar Portal platform was designed and developed to receive, store and provide information resulting from the application of the SBFL technique in a continuous integration environment. The platform consists of a set of applications, as described below:

(1) Jaguar 2, a tool responsible for executing the SBFL technique during build time in the CI pipeline.
(2) *jaguarportal-action*, a Github Action developed in this work to configure the Jaguar Portal platform in a Github CI pipeline.
(3) *jaguarportal-submit*, a command-line tool created in this work to capture the SBFL data file generated by Jaguar 2 and submit it to *jaguarportal-web*. This command-line tool is executed internally in the GitHub CI pipeline when configuring *jaguarportal-action* in the CI pipeline.
(4) *jaguarportal-web*, a web application developed in this work, which includes a set of endpoints responsible for receiving and storing data in the database and web pages for displaying SBFL analyses, as well as user configurations, permissions, and necessary authentication information for CI pipeline integration.

Although item 2) is directly related to Github Actions, as it is a functionality of the GitHub platform, its role is merely to automate the execution of the *jaguarportal-submit* command-line tool presented in item 3). Therefore, the execution of the command-line tool can be automated in other ways in different CI tools, such as Jenkins[3].

Jaguar Portal operation in a continuous integration environment, depicted in Figure 1, follows the steps below.

(1) The developer makes changes in a source code repository.
(2) The repository triggers the CI pipeline.
(3) The CI pipeline runs the build script.
(4) The build script, in one of its stages, runs tests; if the tests do not fail, the CI pipeline concludes successfully; otherwise, the SBFL tool is triggered.
(5) The *jaguarportal-submit* action is triggered when the previous stage, "build," fails.
(6) The *jaguarportal-submit* action submits the data generated by the SBFL tool to the WebAPI endpoint of *jaguarportal-web*.
(7) Jaguar Portal stores the information in the database.
(8) Jaguar Portal returns the analysis results URL to the action.
(9) The analysis URL is logged in the CI pipeline log.
(10) The CI pipeline concludes with a failure.

[1] GitHub Marketplace is a platform featuring tools from the community and partners to streamline tasks and automate processes. Available on https://github.com/marketplace
[2] Surefire Plugin: https://maven.apache.org/surefire/maven-surefire-plugin/
[3] Jenkins is an open source automation server that provides hundreds of plugins to support building, deploying, and automating any project. Available on https://www.jenkins.io/

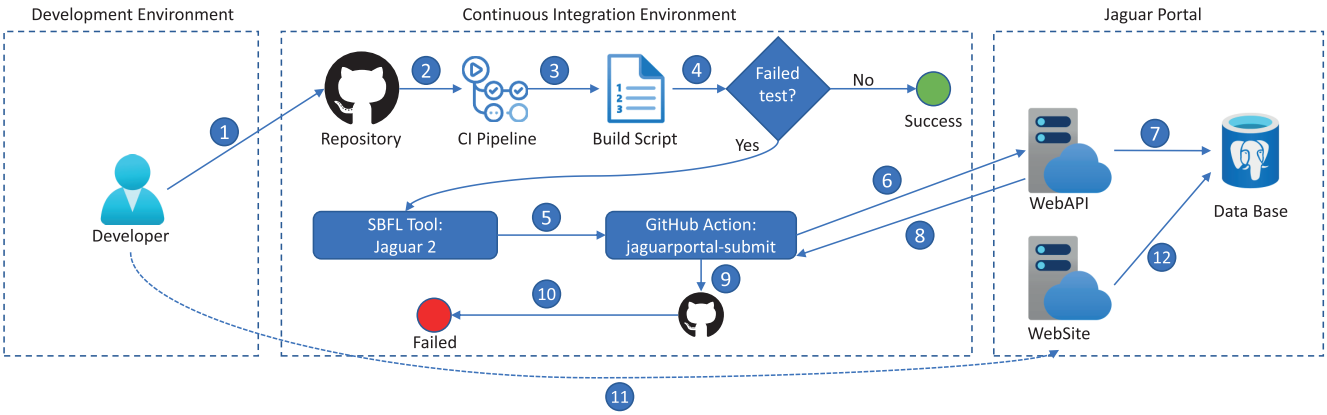

Figure 1: Proposed continuous integration environment with Jaguar Portal

(11) After data submission to Jaguar Portal, the developer can access the analysis through the link added to the executed CI pipeline comment or directly through the analysis dashboard of *jaguarportal-web*.
(12) Data stored in the database is retrieved for display on a web page.

Jaguar Portal is available on its GitHub repository[4] and is run as a container and uses the PostgreSQL data base [27], which has to be configured.

5.1 CI Pipeline Setup

The *jaguarportal-submit* action is available on the GitHub Marketplace[5], as shown in Figure 2. Note that configuring *jaguarPortalClientSecret* can utilize GitHub *secrets* for enhanced security (see Section 4).

To use it, simply configure the action in the workflow of your GitHub repository, as described in Figure 3. Note that the third line includes the `if: failure()` command, ensuring that this action is executed only if a failure occurs.

Upon a failure in the execution of the GitHub workflow, the Jaguar Portal website link that contains the SBFL technique analysis data generated by Jaguar 2 is recorded in conversation, along with the generated suspicion ranking, as shown in Figure 8.

5.2 Data Visualization

The SBFL analysis data is presented on web pages through a ranking of the most suspicious code segments in the program. Source code elements of suspicious classes are presented in different colors, ranging from most suspicious (red) to least suspicious (green) based on their suspicion level. Furthermore, the interface allows navigation of the folder structure to project class files under analysis, as shown in Figure 9.

[4]https://github.com/saeg/jaguarportal-web
[5]https://github.com/marketplace/actions/jaguarportal

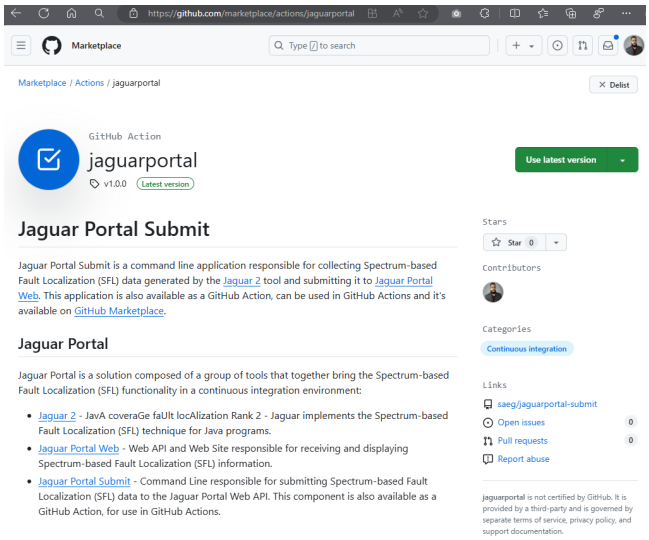

Figure 2: *jaguarportal-submit* action on GitHub Marketplace

```
- name: Submit to Jaguar Portal
  uses: saeg/jaguarportal-submit@v1.0.0
  if: failure()
  with:
    jaguarPortalProjectKey: 341e51b5-e769-4c74-af52-0ebba12ff1d0
    jaguarPortalHostUrl: https://jaguarportal.azurewebsites.net/
    jaguarPortalClientId: 2F2392AB70F21E2CFE80CC420B8CCDA3
    jaguarPortalClientSecret: ${{ secrets.jaguarPortalClientSecret }}
    jaguarPortalAnalysisPath: /target
```

Figure 3: Configuration of *jaguarportal-submit* action

6 Using Jaguar Portal

In this section, we going to use Jaguar Portal to locate a bug in JSoup. JSoup is a Java library for handling HTML (HyperText Markup Language). It parses HTML in the same DOM (Document Object Model) as modern browsers, providing an API for extracting and

manipulating data. We chose this program because it is open source and has a rich history of issues and releases on GitHub[6].

The example presented in Figure 4 uses a copy of a real version that previously existed in the Jsoup repository, which contained a defect. This same defect was documented and explored in the Defects4J repository [21].

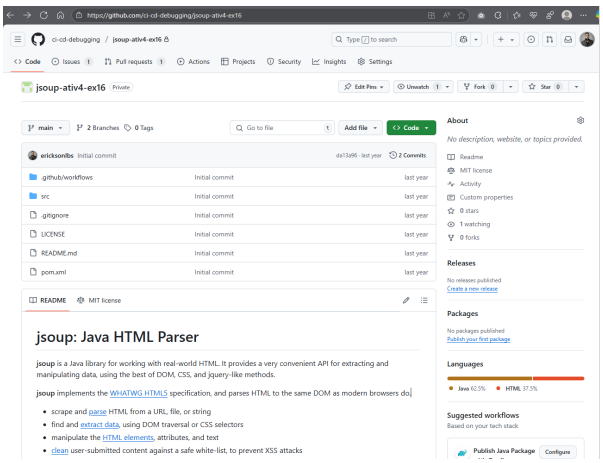

Figure 4: Cloned repository from JSoup

Figure 5 shows the issue 825 as related by the user in the Jsoup's GitHub. This issue triggered the developer to create a new test that replicates the failure caused by the bug, as shown in Figure 6. This test is incorporated into the test suite that runs in the CI pipeline under Jaguar Portal, as shown in Figure 7.

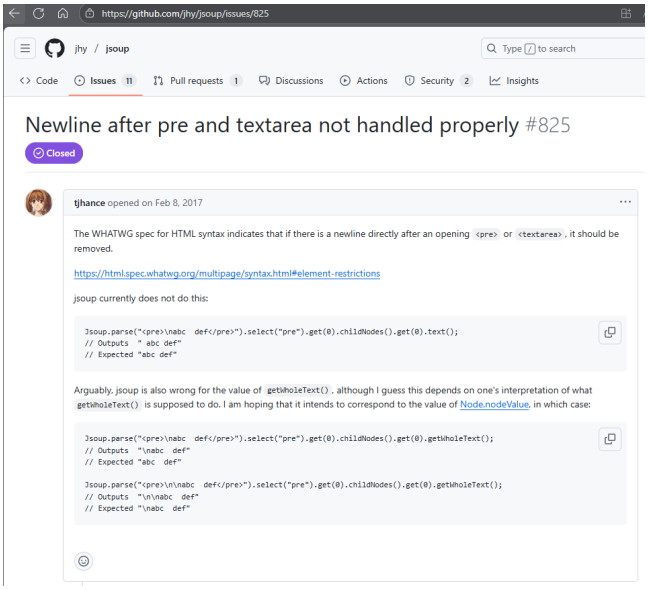

Figure 5: Issue 825 on Jsoup's GitHub

[6]JSoup: https://github.com/jhy/jsoup

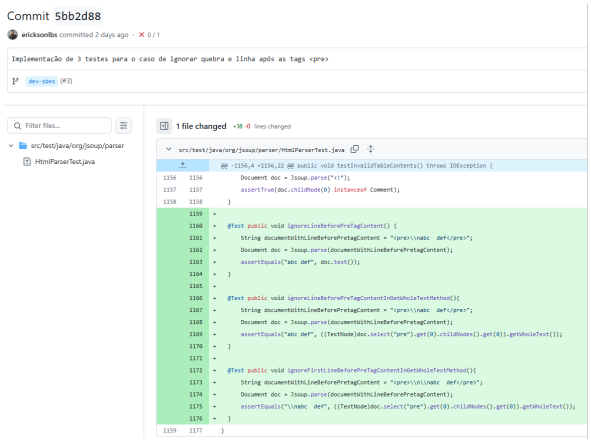

Figure 6: Unit test

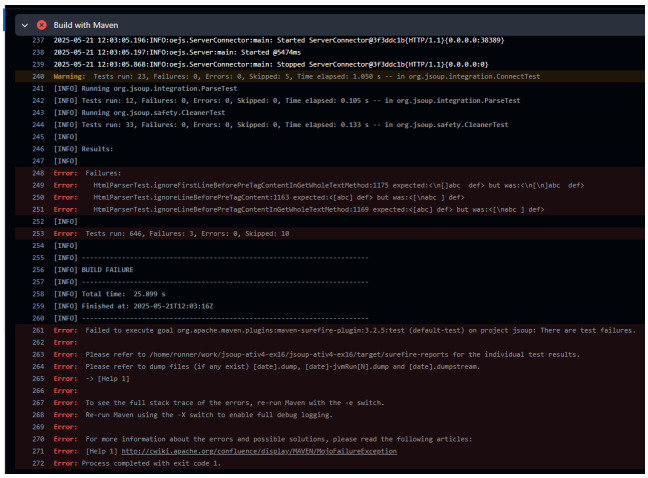

Figure 7: Test run and Build failure

Upon creation and submission of the pull request, the corresponding GitHub Action was automatically triggered. As the test failed, Jaguar Portal was then activated through the *jaguarportal-submit* GitHub Action, which executed Jaguar 2, collected the SBFL data, submitted them to Jaguar Portal's WebAPI, and finally posted a comment on the pull request with this information via the GitHub Bot.

As illustrated in Figure 8, the generated comment includes a link to the failed workflow, another link to view the data on Jaguar Portal, and a suspiciousness ranking ordered by the most to least suspicious code segments.

By accessing the Jaguar Portal link, one can view the detailed information, as shown in Figure 9. In the Jaguar Portal interface, the source code is presented along with a visual indicator of suspiciousness: the most suspicious lines are highlighted in shades of red, while the least suspicious lines appear in shades of green, as depicted in Figure 9.

By examining lines 381 to 383 in Figure 9, one can identify the region with the highest level of suspiciousness. It is precisely in this

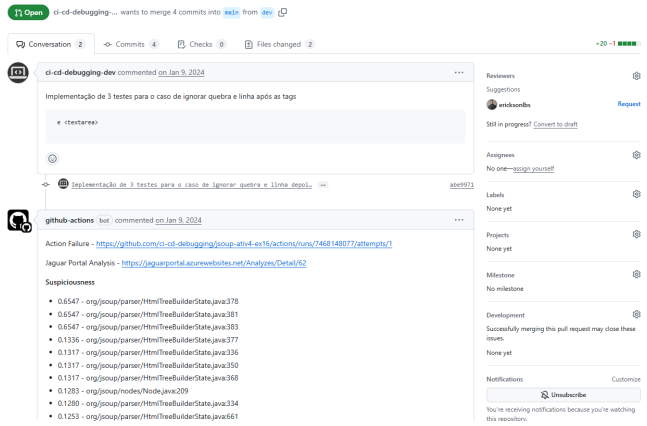

Figure 8: Pull Request in GitHub

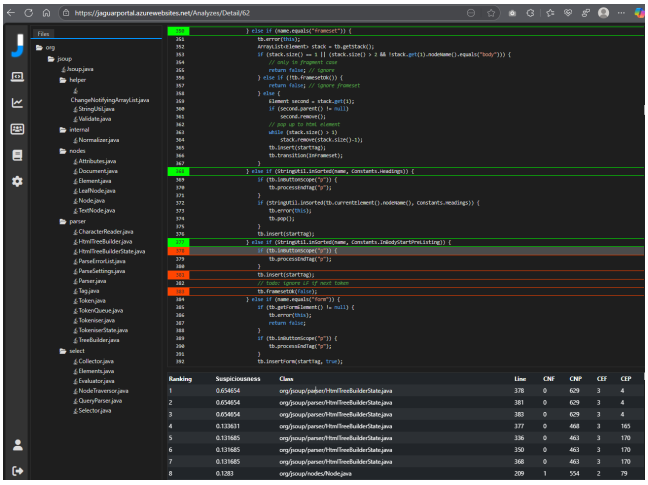

Figure 9: Jaguar Portal Web interface

section that the defect was corrected, as can be verified in the corresponding commit in the original repository: https://github.com/jhy/jsoup/commit/02668f757c59f0c1a7ad8f3169faf061b4b787c1.

7 User study with Jaguar Portal

A user experiment was conducted to identify the potential use of SBFL technology within CI pipelines, employing the Technology Acceptance Model (TAM) [7, 8] as the methodological approach.

Thirteen experimental sessions were conducted, in which each participant accessed a virtual environment pre-configured with an IDE and Java [34]. Within this environment, the participants encountered a buggy version of the Jsoup program that belongs to the Defects4J [21] repository and were challenged to resolve the defect using Jaguar Portal. At the end of each session, participants completed a questionnaire based on TAM [7, 8].

The TAM's potential usage results indicate the participants' willingness to employ Jaguar Portal within an CI environment during debugging activities, should it become available. Even participants who did not find the bug positively evaluated the tool, indicating an overall propensity to adopt debugging support techniques, even after initial unsuccessful attempts.

8 Conclusion

This paper presented Jaguar Portal, a platform designed to integrate SBFL techniques into CI environments, with a focus on real-world applications. The proposed solution leverages established tools such as Jaguar 2 and GitHub, automating the collection, submission, and visualization of SBFL data directly within the CI pipeline. Unlike previous approaches, Jaguar Portal also provides an accessible and intuitive graphical interface that facilitates the analysis of code suspiciousness through rankings and visual highlights.

We conducted a user study in which Jaguar Portal was used to located different Jsoup's bugs [34]. The results suggest that SBFL is effective when used during the CI practice. Thus, we believe this integration represents an important step toward the adoption of SBFL practices in industry, helping to accelerate the debugging process and reduce the response time to failures in the development cycle. As future work, we intend to expand the solution's compatibility with other SBFL tools, such as Flacoco [33] and GZoltar [4].

ARTIFACT AVAILABILITY

Jaguar Portal and its documents use the MIT license. The other tools utilized by Jaguar Portal have their particular licenses. The demo video and the source code for the *jaguarportal-web* and *jaguarportal-submit* applications are available on:

- Permanent location in Figshare: https://doi.org/10.6084/m9.figshare.29114858.v5
- Demo video in Youtube: https://www.youtube.com/@JaguarPortalSBFL.
- Jaguarportal-web in GitHub: https://github.com/saeg/jaguarportal-web
- Jaguarportal-submit in GitHub: https://github.com/saeg/jaguarportal-submit

ACKNOWLEDGMENTS

AI was used to assist with the translation of the article's text using ChatGPT (https://chatgpt.com/) and Writefull (https://www.writefull.com/) was used to check English grammar. For the translation of the video, we used CapCut (https://capcut.com/).

REFERENCES

[1] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan J.C. van Gemund. 2009. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software* 82, 11 (2009), 1780–1792. SI: TAIC PART 2007 and MUTATION 2007.

[2] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. 2007. On the accuracy of spectrum-based fault localization. In *Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007)*. IEEE, 89–98.

[3] Roberto Paulo Andrioli de Araujo. 2014. *Scalable data-flow testing*. Master's thesis. Escola de Artes, Ciências e Humanidades, Universidade de São Paulo.

[4] José Campos, André Riboira, Alexandre Perez, and Rui Abreu. 2012. GZoltar: an eclipse plug-in for testing and debugging. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. 378–381.

[5] Marcos L Chaim, José C Maldonado, and Mario Jino. 2004. A debugging strategy based on the requirements of testing. *Journal of Software Maintenance and Evolution: Research and Practice* 16, 4-5 (2004), 277–308.

[6] Eclipse Contributors. 2023. *JaCoCo*. https://www.jacoco.org/jacoco/

[7] Fred D Davis. 1989. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly* (1989), 319–340.

[8] Fred D Davis. 1993. User acceptance of information technology: system characteristics, user perceptions and behavioral impacts. *International journal of man-machine studies* 38, 3 (1993), 475–487.

[9] Higor Amario de Souza. 2018. *Assessment of spectrum-based fault localization for practical use.* Ph. D. Dissertation. Instituto de Matemática e Estatística (IME), Universidade de São Paulo (USP).

[10] Paul M Duvall, Steve Matyas, and Andrew Glover. 2007. *Continuous integration: improving software quality and reducing risk.* Pearson Education.

[11] Eclipse Foundation. 2023. *Eclipse.* https://www.eclipse.org/

[12] .NET Foundation. 2023. *NUnit.* https://nunit.org/

[13] .NET Foundation. 2023. *xUnit.* https://xunit.net/

[14] Martin Fowler and Matthew Foemmel. 2006. Continuous integration. https://www.martinfowler.com/articles/continuousIntegration.html

[15] Mary Jean Harrold, Gregg Rothermel, Rui Wu, and Liu Yi. 1998. An empirical investigation of program spectra. In *Proceedings of the 1998 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. 83–90.

[16] Q. Idrees Sarhan, A. Szatmari, R. Toth, and A. Beszedes. 2021. CharmFL: A Fault Localization Tool for Python. *Proceedings - IEEE 21st International Working Conference on Source Code Analysis and Manipulation, SCAM 2021* (2021), 114–119.

[17] GitHub Inc. 2023. GitHub. https://github.com

[18] Tom Janssen, Rui Abreu, and Arjan JC Van Gemund. 2009. Zoltar: a spectrum-based fault localization tool. In *Proceedings of the 2009 ESEC/FSE workshop on Software integration and evolution@ runtime*. 23–30.

[19] JetBrains. 2023. *PyCharm.* https://www.jetbrains.com/pycharm/

[20] James A Jones, Mary Jean Harrold, and John Stasko. 2002. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002.* IEEE, 467–477.

[21] René Just, Darioush Jalali, and Michael D. Ernst. 2014. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis* (San Jose, CA, USA) *(ISSTA 2014).* Association for Computing Machinery, New York, NY, USA, 437–440.

[22] Microsoft. 2023. *Visual Studio Code.* https://code.visualstudio.com/

[23] Lee Naish, Hua Jie Lee, and Kotagiri Ramamohanarao. 2009. Spectral Debugging with Weights and Incremental Ranking. In *2009 16th Asia-Pacific Software Engineering Conference*. 168–175.

[24] Lee Naish, Hua Jie Lee, and Kotagiri Ramamohanarao. 2011. A model for spectra-based software diagnosis. *ACM Transactions on software engineering and methodology (TOSEM)* 20, 3 (2011), 1–32.

[25] Ali Ouni, Islem Saidani, Eman Alomar, and Mohamed Wiem Mkaouer. 2023. An Empirical Study on Continuous Integration Trends, Topics and Challenges in Stack Overflow. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering* (Oulu, Finland) *(EASE '23).* Association for Computing Machinery, New York, NY, USA, 141–151.

[26] Hugo Paiva, Jose Campos, and Rui Abreu. 2025. GZoltarAction: A Fault Localization Bot for GitHub Repositories. In *2025 IEEE/ACM International Workshop on Bots in Software Engineering (BotSE).* IEEE Computer Society, 38–42.

[27] PostgresSQL. 2023. *PostgresSQL.* https://www.postgresql.org/

[28] Apache Maven Project. 2023. *Apache Maven is a software project management and comprehension tool.* https://maven.apache.org

[29] Sandra Rapps and Elaine J. Weyuker. 1985. Selecting Software Test Data Using Data Flow Information. *IEEE Trans. Softw. Eng.* 11, 4 (April 1985), 367–375. doi:10.1109/TSE.1985.232226

[30] Henrique L. Ribeiro, Roberto P. A. de Araujo, Marcos L. Chaim, Higor A. de Souza, and Fabio Kon. 2018. Jaguar: A Spectrum-Based Fault Localization Tool for Real-World Software. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST).* 404–409.

[31] Software Analysis & Experimentation Group SAEG. 2023. *BA-DUA.* https://github.com/saeg/ba-dua

[32] Software Analysis & Experimentation Group SAEG. 2023. *Jaguar 2.* https://github.com/saeg/jaguar2

[33] André Silva, Matias Martinez, Benjamin Danglot, Davide Ginelli, and Martin Monperrus. 2021. FLACOCO: Fault Localization for Java based on Industry-grade Coverage. *arXiv preprint arXiv:2111.12513* (2021).

[34] Erickson Lima Barbosa Silva. 2024. *Localização de Defeitos Baseada em Espectro e a Prática de Integração Contínua.* Master's thesis. Escola de Artes, Ciências e Humanidades, Universidade de São Paulo.

[35] The JUnit Team. 2023. *JUnit.* https://junit.org/

[36] Nan Wang, Zheng Zheng, Zhenyu Zhang, and Cheng Chen. 2015. FLAVS: A Fault Localization Add-In for Visual Studio. In *2015 IEEE/ACM 1st International Workshop on Complex Faults and Failures in Large Software Systems (COUFLESS).* 1–6.

[37] W Eric Wong, Vidroha Debroy, and Byoungju Choi. 2010. A family of code coverage-based heuristics for effective fault localization. *Journal of Systems and Software* 83, 2 (2010), 188–208.

[38] Jian Xu, Zhenyu Zhang, Wing Kwong Chan, TH Tse, and Shanping Li. 2013. A general noise-reduction framework for fault localization of Java programs. *Information and Software Technology* 55, 5 (2013), 880–896.

[39] Abubakar Zakari, Sai Peck Lee, Khubaib Amjad Alam, and Rodina Ahmad. 2019. Software fault localisation: a systematic mapping study. *IET Software* 13, 1 (2019), 60–74.