GZoltarAction: A Fault Localization Bot for GitHub Repositories

Hugo Paiva[1], José Campos[1,2], Rui Abreu[1,3]
[1]Faculty of Engineering, University of Porto, Porto, Portugal
[2]LASIGE, Faculdade de Ciências, Universidade de Lisboa, Lisboa, Portugal
[3]INESC-ID Lisboa, Portugal
up202103394@edu.fe.up.pt, jcmc@fe.up.pt, rui@computer.org

Abstract—**Despite the number of automatic fault localization tools available for different languages, finding the code responsible for a software failure is still performed manually. This paper presents GZOLTARACTION, an automated fault localization bot designed for GitHub repositories to assist developers in identifying software faults within a Continuous Integration (CI) environment. Leveraging Spectrum-based Fault Localization (SBFL) via the GZOLTAR tool, GZOLTARACTION embeds tailored fault localization reports directly into pull requests or commits, enhancing developers' ability to debug efficiently. Our solution, available on the GitHub Marketplace, offers a practical and accessible tool for the open-source community, hopefully helping streamline automatic fault localization and reducing manual fault-finding efforts.**

***Index Terms*—Bots, Software debugging, Fault localization, GitHub**

I. INTRODUCTION

The continued use of traditional debugging (in particular, *fault localization*) techniques has contributed to the high cost of software failures [1]. To mitigate this, researchers have proposed automatic techniques to help software developers identify the location of faults in software programs and ultimately repair them [2].

The most successful technique to date is Spectrum-based Fault Localization (SBFL) [3]. SBFL relies on the execution behavior ("spectrum") of the program's test cases and their outcome (i.e., pass or fail). Given a spectrum, SBFL uses statistical methods [4, 5, 6] to compute the likelihood of each code component (e.g., statement) of being the truly faulty one. Components are then ranked by their likelihood, i.e., suspiciousness score, and the ones with this highest value are more likely to bee the ones responsible for the software failure.

For Java, several tools and plug-ins that use SBFL have been proposed, e.g., Vida [7], Falcon [8], GZOLTAR [9], Jaguar [10], iFL4Eclipse [11], and recently FLACOCO [12]. These tools have been successfully used in several research studies (e.g., [12, 13, 14, 15, 16]), and some have been integrated into other tools, for example, both GZOLTAR and FLACOCO have been integrated into the automatic program repair tool ASTOR [17]. Despite their success, others have pointed out [18, 19, 20] that **there is a need for a tool that could perform fault localization in a Continuous Integration (CI) environment**.

Thus, in this paper, **we propose GZOLTARACTION, a fault localization bot for GitHub repositories** that runs on GitHub CI. GZOLTARACTION leverages the most popular fault localization tool, GZOLTAR, to create tailored and embedded fault localization reports directly in commits and pull requests on GitHub. In summary, GZOLTARACTION aims to reduce the effort of debugging software.

GZOLTARACTION's source code is available at

https://github.com/GZoltar/gzoltar-github-action

and it is also available in the GitHub Actions Marketplace at

https://github.com/marketplace/actions/gzoltar

for others to integrate with their GitHub repositories.

II. GZOLTARACTION

GitHub is the most widely used code hosting service, with 94 million users and 263 million automated jobs running on GitHub Actions every month as of 2022. GitHub supports *Actions*, which allows one to configure a workflow (e.g., one that compiles the project, runs its test suite, and deploys it) based on events such as a push or pull request. There are more than 10,000 Actions on the GitHub Marketplace[1] anyone can use free of charge at the time of writing. GitHub offers 2,000 workflow minutes (per month) for any repository and it does not require the acquisition or configuration of a server, unlike other CI tools, e.g., Jenkins. That said, GitHub Actions might be the most suitable environment to run and integrate an approach that aims to automate the debugging process, just like the one we propose in this paper.

A. Implementation

GZOLTARACTION was implemented using TypeScript, a popular superset of JavaScript language that adds static typing and improves code readability, and was built with Node.js v16. It depends on:

- `@actions/artifact` (v1.1.1) to upload, as an artifact, the data generated by GZOLTAR to GitHub.
- `@actions/core` (v1.10.0) to get GZOLTARACTION's inputs, set its outputs, and set its status on GitHub Actions.
- `@actions/github` (v5.1.1) to get the diff between commits and create comments on commit or pull requests.

[1]https://github.com/marketplace?type=actions

B. Setup

GZoltarAction requires that the project under debugging is configured to run GZoltar [9] either via the project's management tool, i.e., Apache Maven plug-in or Apache Ant task, or via GZoltar's command line interface. GZoltarAction would fail and not report any result, if it does not successfully run GZoltar on the project under debugging.

As with any GitHub Action, GZoltarAction requires a YAML configuration file that specifies its permissions and steps. An example of a YAML file for GZoltarAction is presented in Listing 1. In a nutshell, it defines the command to compile the project under debugging (line 22), the command to run the project's test suite and to collect code coverage with GZoltar (line 25), and the command to make GZoltar generate the fault localization data. Lines 33 and above define GZoltarAction's parameters. The complete list of parameters and their description can be found in GZoltarAction's documentation on GitHub. Note that, given that GZoltar supports 16 different formulas [5] to compute the likelihood of each line of code, one may define 16 formulas in Line 34.

```
1  name: Run GZoltarAction
2  on:
3    push:
4      branches:
5        - main
6      paths-ignore:
7        - "**.md"
8  jobs:
9    fault-localization:
10     permissions:
11       # GZoltarAction requires these two permissions
12       # to be able to create comments on commits and
13       # pull requests
14       contents: write
15       pull-requests: write
16     runs-on: ubuntu-latest
17     steps:
18       - uses: actions/checkout@v3
19       - uses: actions/setup-java@v2
20
21       - name: Compile project's test suite
22         run: mvn clean test-compile
23
24       - name: Collect code coverage with GZoltar
25         run: mvn -P sufire gzoltar:prepare-agent test
26
27       - name: Generate fault localization data with GZoltar
28         run: mvn gzoltar:fl-report
29
30       - name: Run GZoltarAction
31         uses: GZoltar/gzoltar-github-action@v0.0.2
32         with:
33           build-path: "/target"
34           sfl-ranking: "[ochiai]"
35           sfl-threshold: "[0.5]"
36           sfl-ranking-order: "ochiai"
37           upload-artifacts: true
```

Listing 1: Example of a configuration file for GZoltarAction.

C. Modus operandi

Figure 1 shows GZoltarAction's workflow and the following subsubsections describe each step of the workflow. It all starts when a developer pushes a new commit(s) to GitHub, which triggers the execution of any configured GitHub Action, including our GZoltarAction.

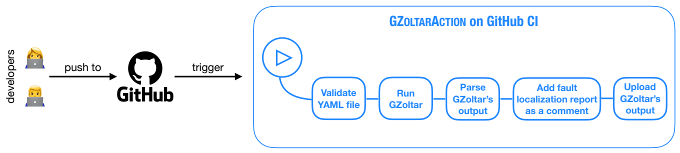


Figure 1: GZoltarAction workflow.

1) Validate YAML configuration file: GZoltarAction starts by parsing and validating whether the YAML configuration file is correct. It assesses whether GZoltarAction's parameters are correct and well instantiated, e.g., whether the number of elements in the parameter `sfl-ranking` and `sfl-threshold` are the same. GZoltarAction ends with an error message if the file is not correct.

2) Run GZoltar: GZoltarAction takes the commands (defined in the YAML file) to run the project's test suite and collect their code coverage with GZoltar, and execute them. If any command fails, GZoltarAction ends with an error.

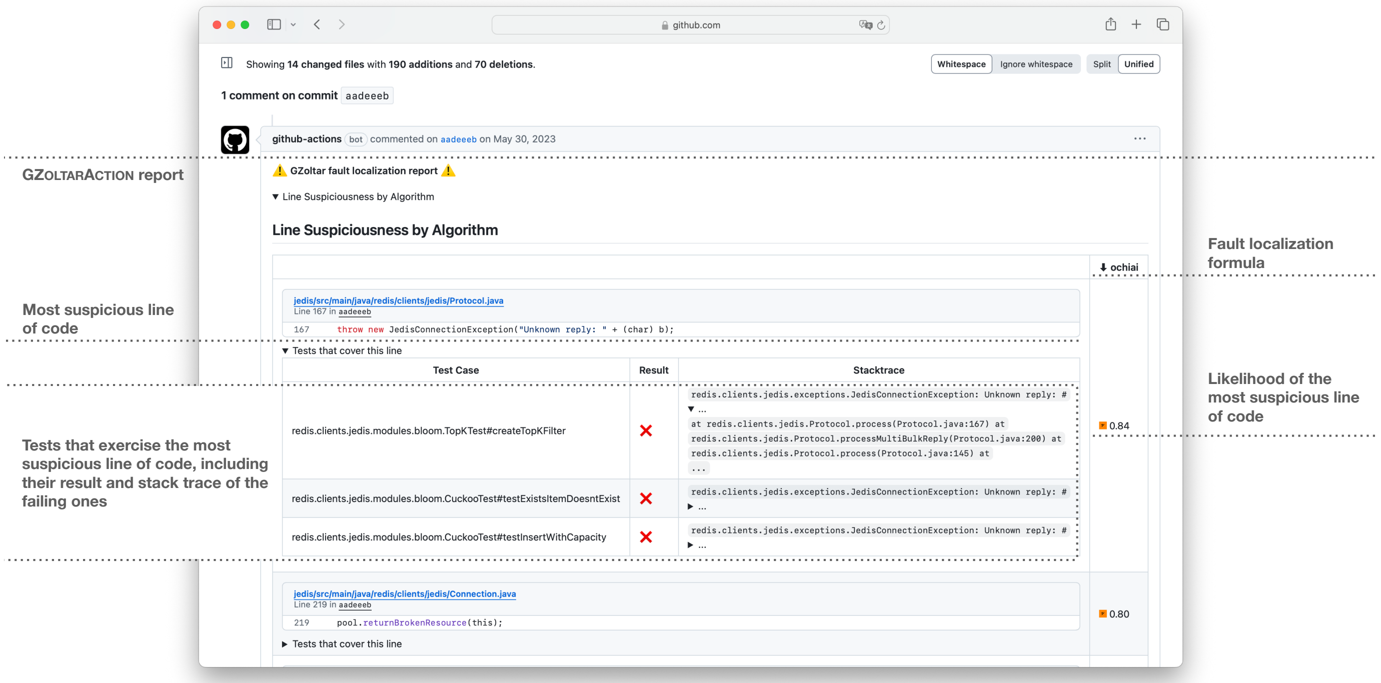
3) Parse GZoltar's output: GZoltarAction loads and parses three files generated by GZoltar:

- `tests.csv`: a CSV file with the set of test cases, one per row, executed by GZoltar. Each row contains the name of the test, the outcome of the test (i.e., pass or fail), execution time in nanoseconds, and stack trace if the test failed.
- `spectra.csv`: a CSV file with the set of lines of code, one per row, in the project under test and their corresponding suspiciousness score. Each row contains the full name of a line of code, i.e., class name, method name, and line number, and its suspiciousness score. Note that while parsing this file, GZoltarAction (a) discards suspicious scores lower than the defined `sfl-threshold` and (b) finds the file of any line of code in the `spectra.csv` so that it can later be referenced directly on a comment on GitHub.
- `matrix.txt`: a text-based coverage matrix of the tests executed by GZoltar. Rows represent test executions, and columns represent lines of code.

4) Add fault localization report as a comment to commits or pull requests on GitHub: GZoltarAction reports the fault localization data generated by GZoltar for each formula [5] defined in the YAML, in two different places on the GitHub UI: (i) as an **overall comment** of a commit or pull request and (ii) as a **comment at the line or code block level** in the commit diff. Any comment produced by GZoltarAction is written in markdown, the markup language used by GitHub.

(i) The **overall comment** of a commit or pull request (see Figure 2) organizes the fault localization data in two different ways:

- **Line Suspiciousness by Algorithm**, lists all lines of code with a suspiciousness score higher than the defined `sfl-threshold`. For each line of code, GZoltarAction reports (1) the source code, (2) a link to the line number in its java file on GitHub, (3) the set of test cases that exercise it (including the test result and stack trace, if any), and (4) its suspiciousness score. Lines of code are sorted from the most suspicious to the least suspicious.


GZoltarAction report
Most suspicious line of code
Tests that exercise the most suspicious line of code, including their result and stack trace of the failing ones
Fault localization formula
Likelihood of the most suspicious line of code
github.com
Whitespace
Ignore whitespace
Split
Unified

Showing **14 changed files** with **190 additions** and **70 deletions**.

1 comment on commit `aadeeeb`

github-actions bot commented on aadeeeb on May 30, 2023

⚠️ **GZoltar fault localization report** ⚠️

▼ Line Suspiciousness by Algorithm

Line Suspiciousness by Algorithm

	⬇ ochiai
jedis/src/main/java/redis/clients/jedis/Protocol.java Line 167 in aadeeeb `167 throw new JedisConnectionException("Unknown reply: " + (char) b);` ▼ Tests that cover this line	🟧 0.84
jedis/src/main/java/redis/clients/jedis/Connection.java Line 219 in aadeeeb `219 pool.returnBrokenResource(this);` ▶ Tests that cover this line	🟧 0.80

Test Case	Result	Stacktrace

redis.clients.jedis.modules.bloom.TopKTest#createTopKFilter	✗	`redis.clients.jedis.exceptions.JedisConnectionException: Unknown reply: #`
		▼ ...
		`at redis.clients.jedis.Protocol.process(Protocol.java:167) at`
		`redis.clients.jedis.Protocol.processMultiBulkReply(Protocol.java:200) at`
		`redis.clients.jedis.Protocol.process(Protocol.java:145) at`
		`...`

redis.clients.jedis.modules.bloom.CuckooTest#testExistsItemDoesntExist	✗	`redis.clients.jedis.exceptions.JedisConnectionException: Unknown reply: #` ▶ ...
redis.clients.jedis.modules.bloom.CuckooTest#testInsertWithCapacity	✗	`redis.clients.jedis.exceptions.JedisConnectionException: Unknown reply: #` ▶ ...

Figure 2: Overall comment of a commit.

- **Lines Code Block Suspiciousness by Algorithm**, also lists all lines of code with a suspiciousness score higher than the defined `sfl-threshold`, but groups the ones in **consecutive** line numbers. Given that a group could be composed of more than one line of code, the set of test cases that exercise each line is not reported.

Note that if GZOLTAR does not report any failing test case in step 2), *no fault* is detected by the project's test suite. In such cases, GZOLTARACTION generates an overall comment:

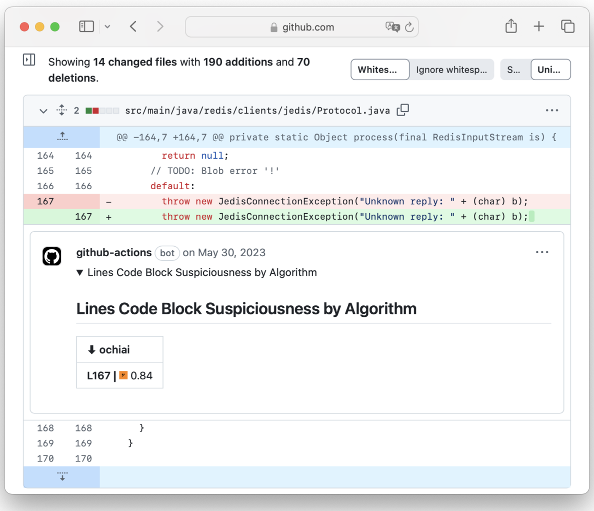
As there is no failing test, GZOLTAR *has nothing to report.*

(ii) The **comment at the line or code block level** in the commit diff (or the most recent commit in the case of a pull request, see Figure 3) is injected directly on the line of code present in the diff, and it only reports the suspiciousness score. When the `diff-comments-code-block` parameter is defined, the comment is injected directly on the last line of code of the block in the diff.

The colors associated with the suspiciousness score follow the ColorADD[2] system, allowing color-blind people to distinguish colors. 🟩 reports suspiciousness scores lower than 0.50, 🟨 greater than 0.50, 🟧 greater than 0.75, and 🟥 greater than 0.90.

5) Upload GZOLTAR*'s output:* If the `upload-artifacts` is enabled in the YAML file, all files generated by GZOLTAR (e.g., serialized coverage file, or HTML fault localization reports [21]) are copied to the Action's output artifact—a zip file containing all directories

[2]ColorADD - Color is for ALL!, https://www.coloradd.net.



Showing **14 changed files** with **190 additions** and **70 deletions**.

```
2  src/main/java/redis/clients/jedis/Protocol.java
            @@ -164,7 +164,7 @@ private static Object process(final RedisInputStream is) {
164  164             return null;
165  165           // TODO: Blob error '!'
166  166           default:
167       -          throw new JedisConnectionException("Unknown reply: " + (char) b);
     167  +          throw new JedisConnectionException("Unknown reply: " + (char) b);
```

github-actions bot on May 30, 2023

▼ Lines Code Block Suspiciousness by Algorithm

Lines Code Block Suspiciousness by Algorithm

⬇ ochiai
L167 | 🟧 0.84

```
168  168         }
169  169       }
170  170
```

Figure 3: Comment at line level in the commit diff.

and files generated during the execution of the Action. The artifact is later available for download in the action menu on GitHub for developers to consult or debug.

D. Limitations

We identified two limitations not inherent to the Action itself during its development.

1) Comment string length: The comments added by GZOLTARACTION to commits or pull requests can easily get very long due to the stack trace of the fault-revealing test cases. Given that GitHub has a limit of 65,536 characters per comment on a commit or pull request, the action would fail if GZoltarAction's comments exceed this limit. To minimize this limitation, GZoltarAction truncates the stack trace of each test to 300 characters and appends "..." when the limit is exceeded. Nevertheless, the limit can still be exceeded if there are too many failing tests.

2) Comment area size: GitHub assigns a maximum width of 780px for the comments box. Although this may be sufficient for most cases, when there are long lines of code, the fault localization report generated by GZoltarAction could look unformatted, making it difficult to read and understand its content. To address this limitation, one could execute a simple JavaScript script in the browser console to increase the maximum width size. This script can be found in Listing 5.1 in [22] and its impact in Figure 5.4 in [22].

III. First attempt of running GZoltarAction in the open-source world

The open-source world represents a collaborative and transparent approach to software development, where the source code is freely accessible and modifiable. This inclusive model has had a transformative impact on technology and society. At the forefront of this movement is GitHub, one of the most trusted platforms to host open-source projects, providing an ecosystem for developers to collaborate.

Thus, we integrated GZoltarAction in the Jedis project[3], one of the most popular Java projects in the curated list of awesome frameworks, libraries, and software for the Java programming language[4] on GitHub. Jedis is the Java client for Redis, a popular in-memory database with 11,872 stars and 226 contributors. For this project, we (i) configured the execution of GZoltarAction[5], (ii) created a pull request[6] with an introduction to the world of SBFL and GZoltar, and (iii) executed the GZoltarAction in a commit with failing tests[7] as a demonstration example to show to Jedis' developers.

Although the setup process and the execution of GZoltar and GZoltarAction ran successfully, the integration of GZoltarAction was not accepted. One of the Jedis maintainers mentioned[8] that

> "**It'll be fascinating to see this project as it progresses** *- but for now I think it's too early to include within this library. For now, we'll remain on the sidelines."*

His response shows that there is interest in a solution of this type (as supported by the related literature [12]), but one might need more evidence that it works (e.g., in smaller projects) before it gets integrated into large-scale projects such as Jedis.

[3]https://github.com/redis/jedis
[4]https://github.com/akullpp/awesome-java
[5]https://github.com/hugofpaiva/jedis/blob/master/.github/workflows/integration-gzoltar.yml
[6]https://github.com/redis/jedis/pull/3448
[7]https://github.com/hugofpaiva/jedis/commit/aadeeeb
[8]https://github.com/redis/jedis/pull/3448#issuecomment-1576735222

IV. Related work

Bots, automated software agents, can assist developers and testers in various software engineering tasks (e.g., [23, 24, 25, 26, 27, 28, 29, 30, 31]), including fault localization. Thus, it is natural that bots that use existing SBFL tools have emerged. To the best of our knowledge, only one work has proposed a bot for fault localization.

FLACOCOBOT is a bot that uses the FLACOCO [12] fault localization tool for Java programs. Briefly, it all starts when a developer creates or updates a pull request on a platform like GitHub, which then triggers a job in a CI system (e.g., Travis or Jenkins). When FLACOCOBOT detects a failing build (note that it autonomously scans the status of the pipeline from a list of projects), it clones the project, executes FLACOCO, computes the top most suspicious lines of code in the pull request's diff, and posts a comment on the pull request with the fault localization data (suspicious score per line of code, and the set of tests that failed and covered each line).

Although the primary goals of FLACOCOBOT and GZoltarAction are the same, and they share some features (e.g., the ability to add comments to pull requests or to individual lines of code), there are few notable differences.[9] (1) FLACOCOBOT (either source code or binary) is unavailable. GZoltarAction, on the other hand, is an open-source project and it is available in the GitHub Actions Marketplace. (2) Regarding filtering out the results of the underlying fault localization tool, FLACOCOBOT allows the selection of the top-k most suspicious lines of code and GZoltarAction allows the selection of the lines of code with a suspicious score higher than a given threshold. (3) FLACOCOBOT reports fault localization data at the line level, whereas GZoltar supports at the line and code block level. (4) Finally, FLACOCOBOT does not publish or make available the data generated by the underlying fault localization tool, as opposed to GZoltarAction, which uploads the data to the action's artifact on GitHub.

V. Conclusion and Future work

In this paper, we proposed the first *bot* on automatic fault localization that is fully integrated into the most popular code hosting service, GitHub, through GitHub Actions. As for future work, we aim to explore the creation and inline of the new *GitHub Annotations* to replace our current way of adding comments on pull requests or commit diffs. Annotations are similar to comments but can be attached to multiple lines or specific parts of a line of code. They can include a title, a message, and details; and have multiple levels of severity.

Acknowledgments

This work was supported by Fundação para a Ciência e Tecnologia (FCT) through the LASIGE Research Unit, ref. UID/000408/2025; and INESC-ID, ref. UIDB/50021/2020 (https://doi.org/10.54499/UIDB/50021/2020).

[9]Example of a comment generated by FLACOCOBOT [12], https://github.com/INRIA/spoon/pull/4709#pullrequestreview-951192304.

References

[1] Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. 3rd. Wiley Publishing, 2011. ISBN: 1118031962, 9781118031964.

[2] Martin Monperrus. *The Living Review on Automated Program Repair*. Tech. rep. hal-01956501. HAL/archives-ouvertes.fr, 2018.

[3] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. "A Survey on Software Fault Localization". In: *IEEE Transactions on Software Engineering* 42.8 (2016), pp. 707–740. DOI: 10.1109/TSE.2016.2521368.

[4] James A. Jones, Mary Jean Harrold, and John Stasko. "Visualization of test information to assist fault localization". In: *Proceedings of the 24th International Conference on Software Engineering*. ICSE '02. Orlando, Florida: Association for Computing Machinery, 2002, 467–477. ISBN: 158113472X. DOI: 10.1145/581339.581397.

[5] Rui Abreu, Peter Zoeteweij, and Arjan J.c. Van Gemund. "An Evaluation of Similarity Coefficients for Software Fault Localization". In: *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*. 2006, pp. 39–46. DOI: 10.1109/PRDC.2006.18.

[6] Rui Abreu, and Arjan J.C. van Gemund. "A Low-Cost Approximate Minimal Hitting Set Algorithm and its Application to Model-Based Diagnosis". In: *Proceedings of the 8th Symposium Abstraction, Reformulation, Approximation (SARA)*. Lake Arrowhead, CA, USA, 2009, 1–8.

[7] Dan Hao, Lingming Zhang, Lu Zhang, Jiasu Sun, and Hong Mei. "VIDA: Visual interactive debugging". In: *2009 IEEE 31st International Conference on Software Engineering*. 2009, pp. 583–586. DOI: 10.1109/ICSE.2009.5070561.

[8] Sangmin Park, Richard W. Vuduc, and Mary Jean Harrold. "Falcon: fault localization in concurrent programs". In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*. ICSE '10. Cape Town, South Africa: Association for Computing Machinery, 2010, 245–254. ISBN: 9781605587196. DOI: 10.1145/1806799.1806838.

[9] José Campos, André Riboira, Alexandre Perez, and Rui Abreu. "GZoltar: an eclipse plug-in for testing and debugging". In: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ASE '12. Essen, Germany: Association for Computing Machinery, 2012, 378–381. ISBN: 9781450312042. DOI: 10.1145/2351676.2351752.

[10] Henrique L. Ribeiro, Roberto P. A. de Araujo, Marcos L. Chaim, Higor A. de Souza, and Fabio Kon. "Jaguar: A Spectrum-Based Fault Localization Tool for Real-World Software". In: *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. 2018, pp. 404–409. DOI: 10.1109/ICST.2018.00048.

[11] Gergo Balogh, Ferenc Horváth, and Árpád Beszédes. "Poster: Aiding Java Developers with Interactive Fault Localization in Eclipse IDE". In: *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. 2019, pp. 371–374. DOI: 10.1109/ICST.2019.00045.

[12] André Silva, Matias Martinez, Benjamin Danglot, Davide Ginelli, and Martin Monperrus. *FLACOCO: Fault Localization for Java based on Industry-grade Coverage*. 2023. arXiv: 2111.12513 [cs.SE].

[13] Spencer Pearson, José Campos, René Just, Gordon Fraser, Rui Abreu, Michael D. Ernst, Deric Pang, and Benjamin Keller. "Evaluating and Improving Fault Localization". In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 2017, pp. 609–620. DOI: 10.1109/ICSE.2017.62.

[14] David Paterson, Jose Campos, Rui Abreu, Gregory M. Kapfhammer, Gordon Fraser, and Phil McMinn. "An Empirical Study on the Use of Defect Prediction for Test Case Prioritization". In: *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. 2019, pp. 346–357. DOI: 10.1109/ICST.2019.00041.

[15] Prantik Chatterjee, Abhijit Chatterjee, José Campos, Rui Abreu, and Subhajit Roy. "Diagnosing software faults using multiverse analysis". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. IJCAI'20. Yokohama, Yokohama, Japan, 2021. ISBN: 9780999241165.

[16] Prantik Chatterjee, José Campos, Rui Abreu, and Subhajit Roy. "Augmenting automated spectrum based fault localization for multiple faults". In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. IJCAI '23. Macao, P.R.China, 2023. ISBN: 978-1-956792-03-4. DOI: 10.24963/ijcai.2023/350.

[17] Matias Martinez and Martin Monperrus. "ASTOR: a program repair library for Java (demo)". In: *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ISSTA 2016. Saarbrücken, Germany: Association for Computing Machinery, 2016, 441–444. ISBN: 9781450343909. DOI: 10.1145/2931037.2948705.

[18] Pavneet Singh Kochhar, Xin Xia, David Lo, and Shanping Li. "Practitioners' expectations on automated fault localization". In: *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ISSTA 2016. Saarbrücken, Germany: Association for Computing Machinery, 2016, 165–176. ISBN: 9781450343909. DOI: 10.1145/2931037.2931051.

[19] Aaron Ang, Alexandre Perez, Arie Van Deursen, and Rui Abreu. "Revisiting the Practical Use of Automated Software Fault Localization Techniques". In: *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. 2017, pp. 175–182. DOI: 10.1109/ISSREW.2017.68.

[20] Archana and Ashutosh Agarwal. "Evaluation of spectrum based fault localization tools". In: *Proceedings of the 15th Innovations in Software Engineering Conference*. ISEC '22. Gandhinagar, India: Association for Computing Machinery, 2022. ISBN: 9781450396189. DOI: 10.1145/3511430.3511470.

[21] Carlos Gouveia, José Campos, and Rui Abreu. "Using HTML5 visualizations in software fault localization". In: *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*. 2013, pp. 1–10. DOI: 10.1109/VISSOFT.2013.6650539.

[22] Hugo Paiva. "Integration of Fault Localization into your GitHub Repository". MA thesis. Porto, Portugal: Faculty of Engineering of the University of Porto, 2023.

[23] Zhendong Wang, Yi Wang, and David Redmiles. "Optimizing Workflow for Elite Developers: Perspectives on Leveraging SE Bots". In: *2023 IEEE/ACM 5th International Workshop on Bots in Software Engineering (BotSE)*. 2023, pp. 23–27. DOI: 10.1109/BotSE59190.2023.00013.

[24] Théo Zimmermann, Julien Coolen, Jason Gross, Pierre-Marie Pédrot, and Gaëtan Gilbert. "The Advantages of Maintaining a Multitask, Project-Specific Bot: An Experience Report". In: *IEEE Software* 39.5 (2022), pp. 32–37. DOI: 10.1109/MS.2022.3179773.

[25] Florian Markusse, Alexander Serebrenik, and Philipp Leitner. "Towards Continuous Performance Assessment of Java Applications With PerfBot". In: *2023 IEEE/ACM 5th International Workshop on Bots in Software Engineering (BotSE)*. 2023, pp. 6–8. DOI: 10.1109/BotSE59190.2023.00009.

[26] Florian Markusse, Philipp Leitner, and Alexander Serebrenik. "Using Benchmarking Bots for Continuous Performance Assessment". In: *IEEE Software* 39.5 (2022), pp. 50–55. DOI: 10.1109/MS.2022.3184430.

[27] Doje Park, Heetae Cho, and Seonah Lee. "Classifying issues into custom labels in GitBot". In: *Proceedings of the Fourth International Workshop on Bots in Software Engineering*. BotSE '22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, 28–32. ISBN: 9781450393331. DOI: 10.1145/3528228.3528404.

[28] Hamid Mohayeji, Felipe Ebert, Eric Arts, Eleni Constantinou, and Alexander Serebrenik. "On the adoption of a TODO bot on GitHub: a preliminary study". In: *Proceedings of the Fourth International Workshop on Bots in Software Engineering*. BotSE '22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, 23–27. ISBN: 9781450393331. DOI: 10.1145/3528228.3528408.

[29] Arkadip Basu and Kunal Banerjee. "Designing a Bot for Efficient Distribution of Service Requests". In: *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*. 2021, pp. 16–20. DOI: 10.1109/BotSE52550.2021.00011.

[30] Ilham Qasse, Shailesh Mishra, and Mohammad Hamdaqa. "iContractBot: A Chatbot for Smart Contracts' Specification and Code Generation". In: *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*. 2021, pp. 35–38. DOI: 10.1109/BotSE52550.2021.00015.

[31] Dragos Serban, Bart Golsteijn, Ralph Holdorp, and Alexander Serebrenik. "SAW-BOT: Proposing Fixes for Static Analysis Warnings with GitHub Suggestions". In: *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*. 2021, pp. 26–30. DOI: 10.1109/BotSE52550.2021.00013.