Does Great Power Come with Great Explainability? Comparing Explanation Strategies for Automated Program Diagnosis

Christopher Klaus Lazik
HU Berlin
Berlin, Berlin, Germany
lazikchr@informatik.hu-berlin.de

Martin Eberlein
HU Berlin
Berlin, Berlin, Germany
martin.eberlein@hu-berlin.de

Aaron Ziglowski
HU Berlin
Berlin, Berlin, Germany
aaron.ziglowski@student.hu-berlin.de

Christopher Katins
HU Berlin
Berlin, Berlin, Germany
christopher.katins@hu-berlin.de

Thomas Kosch
HU Berlin
Berlin, Berlin, Germany
thomas.kosch@hu-berlin.de


```
(forall <number> elem in start:
   (<= (str.to.int elem) (str.to.int "-1")) and
exists <function> elem_0 in start:
  (= elem_0 "sqrt"))
```

AVICENNA

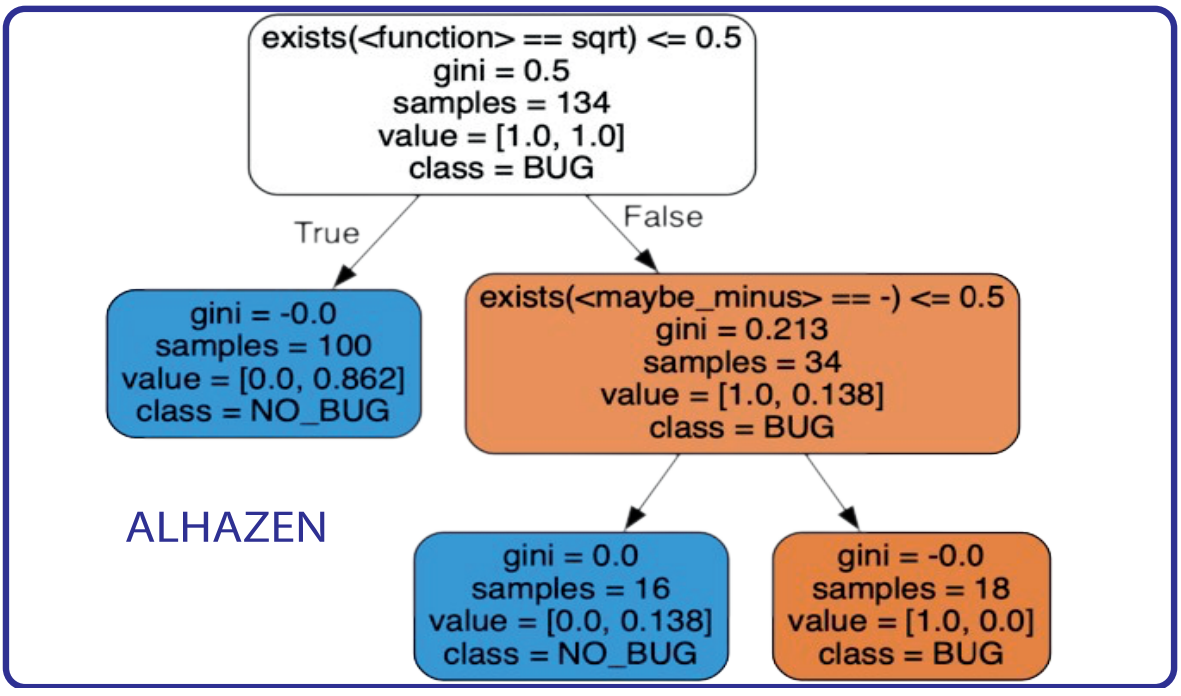


Figure 1: We investigated how users perceive the explanations of the two debugging tools ALHAZEN and AVICENNA. ALHAZEN employs binary decision trees to show failure-inducing conditions, while AVICENNA uses a specification language to model complex input dependencies. By conducting a mixed within-between-subjects lab study, we explored how participants performed using diagnoses of both tools. Our participants either used no tool, AVICENNA (top), or ALHAZEN (bottom) to investigate the cause of four bugs. We recorded feedback from the participants to evaluate the user experience. Our results suggest that participants potentially gained trust in the rather complex logical specification explanations by AVICENNA even though they took time and mental demand to understand them. The figure shows the two explanations for a bug in a simple calculator program that fails for negative numbers used in the square root function.

Abstract

Debugging is an important activity in software development, yet providing actionable and comprehensible explanations for program failures remains challenging. Automated tools such as ALHAZEN and AVICENNA address this by using distinct strategies: ALHAZEN employs binary decision trees to show failure-inducing conditions, while AVICENNA uses a specification language to model complex input dependencies. To assess their impact on usability and user efficiency, we conducted a controlled within-between-subjects user study with 18 participants tasked with resolving four software bugs using either tool or no support. Quantitative results showed that both tools improved debugging efficiency compared to manual methods, with AVICENNA offering more precise diagnostics but requiring higher cognitive effort. Qualitative feedback revealed a preference for AVICENNA's expressiveness despite its complexity. Our findings show that effective debugging tools have trade-offs between accuracy and interpretability to support developers' decision-making in increasingly complex software environments.

ACM Reference Format:

Christopher Klaus Lazik, Martin Eberlein, Aaron Ziglowski, Christopher Katins, and Thomas Kosch. 2026. Does Great Power Come with Great Explainability? Comparing Explanation Strategies for Automated Program Diagnosis. In *Proceedings of Mensch und Computer 2026 (MuC '26), August 30–September 02, 2026, Duisburg, Germany*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3820253.3831404

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 414984028 – SFB 1404 FONDA

1 Introduction and Background

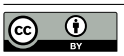
Automated debugging tools emerged to aid developers in diagnosing software issues by identifying failure-inducing inputs and explaining the causes of crashes. Tools such as ALHAZEN [5] and AVICENNA [2] use different approaches to automated debugging, each focusing on providing systematic explanations for program failures. In the literature, various approaches are employed to explain complex circumstances [6, 8, 9] with some of those still under discussion for improvement [7]. While such approaches are proven to be effective in detecting and modeling software errors, explaining the root causes in a way that is actionable and understandable remains a significant challenge, particularly as software systems grow increasingly complex. Our work compares automated debugging tools, ALHAZEN and AVICENNA, with a focus on how their explanation strategies impact general user experience, and efficiency in supporting developers' decision-making within complex software development workflows.

ALHAZEN is designed to explain failures by constructing binary decision trees, which partition the input space into regions based on their failure or non-failure outcomes. By systematically splitting inputs, ALHAZEN identifies minimal input features or characteristics that correlate with the observed failure. This approach offers a clear and structured representation of the conditions leading to crashes, making it particularly useful for pinpointing the precise combination of factors responsible for errors. However, its reliance on decision trees may abstract away nuanced relationships among inputs, limiting its ability to capture more complex input dependencies.

AVICENNA, leverages an input specification language to articulate the relationships among failure inducing inputs. This approach uses logical constraints to model input properties and interactions, enabling it to capture intricate dependencies and generate detailed explanations for failure circumstances. By framing failures in terms of input specifications, AVICENNA provides insights into why certain inputs trigger errors and how input constraints can be modified to avoid similar issues in the future. Furthermore, AVICENNA has demonstrated superior predictive capabilities compared to ALHAZEN, delivering higher accuracy in identifying and modeling the failure-inducing conditions [2]. While highly expressive, the interpretability of these explanations can vary depending on the user's familiarity with the specification language. Effectively communicating these explanations to developers is a critical aspect of both tools' success. Research suggests that user preferences for explanation strategies can differ significantly based on experience, expectations, and problem context [3, 12]. We conducted an explorative user study with 18 participants to evaluate how user approach the two diagnosis strategies. Participants were assigned to investigate four different bugs, using either no debugging tool, ALHAZEN, or AVICENNA for support. The results reveal that users are willing to invest additional time and cognitive effort to interpret AVICENNA 's outputs, potentially because they self-reported to trust its diagnoses. Our findings show the importance of designing debugging tools that not only prioritize technical accuracy but also deliver explanations that are clear, actionable, and aligned with user needs.

2 Methodology and Results

We conducted a mixed within-between-subjects lab study to evaluate the explanation strategies of ALHAZEN and AVICENNA. Participants were all randomly and equally assigned to one of three groups.



This work is licensed under a Creative Commons Attribution 4.0 International License.

MuC '26, Duisburg, Germany

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2611-8/26/08
https://doi.org/10.1145/3820253.3831404

One group was able to use a diagnosis of ALHAZEN, aiming to support understanding the bug circumstances. Another group used diagnoses by AVICENNA to compare the interpretability of both tools. To provide a comparison, a third control group did not consider any diagnoses and had to rely solely on reading the buggy function and executing adjustable test cases in Jupyter. This group assignment makes using a specific tool (or none of them) our primary dependent variable of interest. The goal of this study is to isolate the interaction of understanding the explanation strategies of the tools. Thus, the circumstances of the study do not reflect the complete debugging of a program, but only compare how participants understand the explanations.

Participants. We recruited eighteen participants (N = 18, three self-identified as female, 15 self-identified as male) aged 22 to 30 ($\bar{x}$= 25.78, s = 2.24) that were assigned to three groups of six. We sampled computer science researchers, students, and participants who do programming for work. Participants were informed that their involvement in the study was voluntary, not financially compensated, and that they could withdraw at any time without any repercussions. All participants gave their informed consent before starting their participation.

Procedure and Task. All groups received the bugs randomly ordered in the form of one Jupyter notebook per bug. Every notebook contained a description of the buggy tool, three positive test cases to show its intended usage, and one failing test case to show that the program failed. The test cases were open for adjustments by the participants, enabling them to try out ideas. Additionally, we printed the program's output for the participants. After introducing the failing test case, we listed the code of the buggy function of the program. Each bug in our study was produced by one function of the respective program, excluding scenarios where the interaction of multiple functions leads to failing output. Participants could not manipulate the function's code, forcing them to rely on reading and understanding it without directly debugging it. If a participant was in one of the groups that used a diagnosis, we listed the diagnosis underneath the function's code. Each diagnosis was accompanied by the input grammar on which it was expressed. We did not let the users execute the tools themselves to reduce the time needed to conduct the study. Participants were asked to explain why the presented programs were failing. We gave them a time window of a maximum of 20 minutes per bug. If a participant signaled to be ready or after 20 minutes, we recorded the explanation from the participant and noted down the number of minutes that passed. Researchers in our group evaluated the recording based on the bug descriptions of our ground truth. Whenever a participant explained circumstances correctly, we evaluated their explanation as true. Participants were not informed about their performance during the study. After explaining a bug, we asked the participants to answer a NASA-TLX questionnaire. At the end of our study, participants were debriefed. We offered insights into the ground truth of the bugs and potential fixes. After being debriefed, participants gave feedback about their experience during the study.

Bug Tasks. We included the factor *Bug* as a within factor, where we included four bugs. Participants were assigned one debugging tool (i.e., baseline, ALHAZEN, or AVICENNA) that they used to solve all four bugs. Two bugs, "Calculator" and "Middle," were toy examples introduced in the publications of ALHAZEN and AVICENNA. The other two bugs, "Cookiecutter" and "PySnooper," were real-world examples from the Tests4Py [10] benchmark.

"Calculator" is a simple program that calls functions from the Python math library. Nevertheless, it does not handle errors produced if the users try to calculate the square root of a negative number.

"Cookiecutter" helps users quickly create projects from a template (cookiecutter) from their command line. It comes with different templates, for example, for creating Python package projects. There was an issue in the past when users defined more than one "hook". Those "hooks" can be executed before or after the creation of a project. The bug led to executing only one "hook" while others were ignored.

"PySnooper" is a tool that helps developers debug by printing the state of variables. The program allows the user to specify an output path. But in the past, if the user specifies an output path, the program fails because the developers forgot to define the variable.

"Middle" is a small program that takes three numbers as input and returns the number with the value in the middle. However, because of a mistake in the code, a particular order of the input numbers leads to the wrong number being returned.

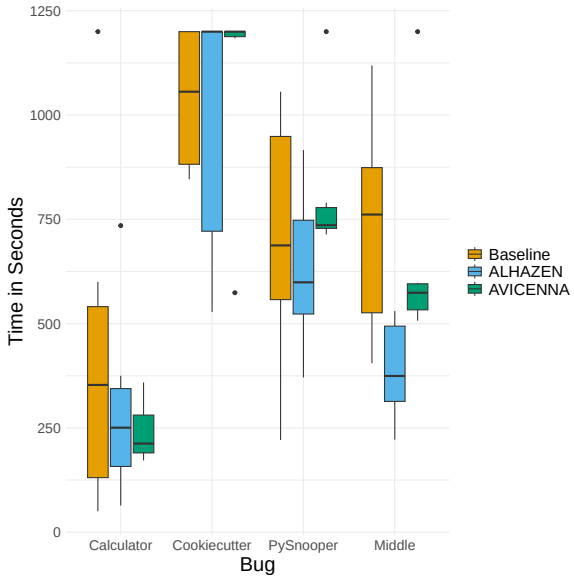
Explorative Results. Due to the lack of statistical power that comes with the limited size of our participant pool, we report the grand mean for each of our measures for each debugging tool for the task completion time and raw NASA-TLX [4]. Figure 2a illustrates the mean times for each debugging tool and bug. In average, ALHAZEN showed the lowest overall task completion time ($\bar{x}$= 575s, s = 224.22s) compared to AVICENNA ($\bar{x}$= 703.38s, s = 195.67s) and the baseline ($\bar{x}$= 727.25s, s = 297.33s). Figure 2b shows the mean raw NASA-TLX score for each debugging tool and bug. In average, ALHAZEN showed the lowest overall NASA-TLX score ($\bar{x}$= 37.00, s = 13.47) compared to AVICENNA ($\bar{x}$= 49.92, s = 18.64) and the baseline ($\bar{x}$= 52.25, s = 15.01). We calculate the overall accuracy by counting the number of errors and dividing it by the number of trials. On average, AVICENNA achieved the highest accuracy ($\bar{x}$= .79, s = .21), while ALHAZEN ($\bar{x}$= .75, s = .40) and the baseline ($\bar{x}$= .75, s = .22) have the same accuracy.

3 Discussion and Limitations

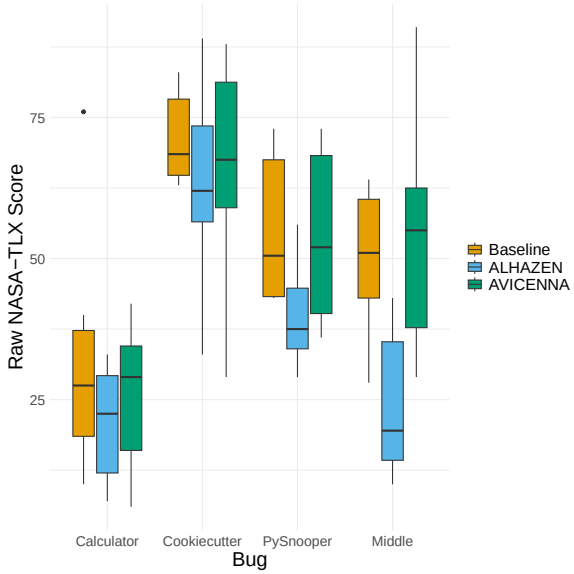
To evaluate the user experience of ALHAZEN and AVICENNA we asked the participants for feedback on their experience in the study. We recorded and transcribed their feedback. If necessary, we translated verbatim to be cited in terms of content in this work. Two researchers thematically analyzed the data individually[1] and agreed upon themes that were inductively created. The qualitative insights aim to reveal a further understanding of the quantitative results presented earlier. We structure the following based on the groups and the identified themes.

3.1 Taking Time to be sure under unsure Circumstances

Participants in the control group mentioned that they needed some time to get into the study and understand the complexity of the task and what was expected from them. Without the help of tools,



(a)



(b)

Figure 2: Results of the study. (a): The task completion time was the highest for the bug *Cookiecutter*. AVICENNA has the highest task completion time except when being used for the bug *Calculator*. (b): AVICENNA descriptively showed the highest task load across all bugs.

the group took, on average, longer than the other two groups to diagnose a bug correctly. Participants told us they tried to **make sure their diagnoses were correct** and, therefore, decided to **intentionally take longer** to be safe. While only one participant was explicitly mentioning the **well structured setup**, the other five participants were **struggling with development setup**. Participants in the first group could not use any of the tools. This baseline might not be representative of classical debugging scenarios, but it compares how people understand the code of the buggy functions in a short time. Usually, developers can play around with the code to formulate their diagnoses. Nevertheless, our study aims to investigate the capability of tools such as AVICENNA and ALHAZEN to enable a user to understand when a program produces a bug. Even though many developers debug code they are more or less familiar with, bugs are especially crucial in scenarios where the failure occurs in unknown code. Thus, our study exposed participants to a code failure to understand the code they had to understand based on the diagnoses of AVICENNA and ALHAZEN. By focusing the circumstances of our study on the above scenario, we aimed to ensure that participants could give detailed feedback about specifically searching for a bug in the code of the function.

3.2 ALHAZEN, the Subtle Supporter

In their feedback, most participants explained that they see ALHAZEN as **potentially helpful**, but they told us that they did not really use it in the study. Participants described the diagnoses of ALHAZEN as **confusing**, because they did not seem to be straightforward to them. Based on their feedback, participants needed time to understand how to read the diagnoses. ALHAZEN's "Middle" and "Cookiecutter" diagnoses were confusing. For "Middle" ALHAZEN's tree is quite **complex** for the relatively simple logical problem of "Middle". In the case of "Cookiecutter" ALHAZEN does focus on input features such as "date", which will not help a user understand the problem of defining multiple hook elements of the same kind. Thus, ALHAZEN's diagnosis cannot help the participant by explaining the problem and **confuses** them. This confusion might explain the worse performance of participants from the ALHAZEN group regarding "Cookiecutter". Participants explained that they **ignored** ALHAZEN's diagnosis whenever it was too **confusing**. However, we see a potential that ALHAZEN could have subconsciously helped the participants by providing a point of orientation that the group without tools was missing. The "Calculator" subject was described as simple and easy to understand by all the participants who used ALHAZEN. On the other hand, participants without tools took longer to realize and assure themselves that the bug in the "Calculator" subject is really that simple. Since the participants explained that they ignored ALHAZEN's diagnoses in certain subjects, we assume they either understood that the diagnoses do not always explain the actual problem or did not feel confident enough about their understanding of the tool. In both cases, the lower task load scores could point to ALHAZEN being helpful but not convincing enough to rely on. This could lead to less effort for the participants to understand the diagnosis and less frustration when ignoring it if necessary. But also some categorization of whether a task could be more complex if ALHAZEN is not clear with its diagnosis.

3.3 AVICENNA comes with great Power, but also great Responsibility

In their feedback, participants all agreed on AVICENNA needing some time to be understood. The diagnoses were compared to logical formulas that seemed rather **complex** initially. This could explain

the lesser impact on the time required to diagnose a bug correctly compared to ALHAZEN. If the participants need a little bit longer to understand the diagnosis of AVICENNA, they need longer, in general, to start with their diagnosis. One participant told us they did **not understand** the diagnoses of AVICENNA during the study. AVICENNA's diagnoses using the ISLa specification language[11] appear to be logical specification formulas. It takes mental effort to understand them properly. Participants might get more frustrated if it is not understood due to its complexity, while it still seems understandable because of its structure being familiar to computer scientists. The corresponding participant was not able to give any diagnosis at all twice. However, most participants explained that they **used the diagnoses** for the bugs and found them especially **helpful for the more complex subjects**. All the participants who understood AVICENNA's diagnoses got either **confused by the diagnosis for "Cookiecutter"**, or diagnosed the problem straight up wrong, but according to AVICENNA. It seems like the participants could have gained trust in AVICENNA's diagnoses. AVICENNA's diagnosis for "Cookiecutter" expresses the length of given input parameters, even though the actual problem is the frequency of a specific hook element. We assume that the potential trust in AVICENNA's diagnosis leads to wrong diagnoses or confusion if the diagnosis is incorrect. Since the feedback points to the participants being misled by AVICENNA for the "Cookiecutter" subject, it would potentially explain their performance on that bug. The combination of trusting the diagnoses and wanting to use them to understand the problem, together with complex logical specification-based explanations, is shared across the feedback of the AVICENNA group. This may be why participants had higher task load scores without giving frustrated feedback. Since understanding AVICENNA is mentally demanding, all the subjects might become perceived as mentally demanding. Compared to ALHAZEN, participants did not tend to ignore AVICENNA's diagnoses but wanted to consider them and, therefore, had to process the explanations. Nevertheless, we did not capture the construct of "trust" explicitly. Our qualitative insights only suggest the trust in AVICENNA according to the aforementioned reasoning.

4 Conclusion and Future Work

This study compared the explanation approaches of AVICENNA and ALHAZEN for automated bug diagnosis. While the expressiveness of AVICENNA was already proven to be more potent than ALHAZEN [2], we show insights into the direct perception of users of both tools. Since the expressiveness of AVICENNA comes with a rather complicated output, users must invest time and effort to understand the diagnosis. Nevertheless, users are willing to invest in AVICENNA likely because they trust its diagnoses. We observed that users recognized AVICENNA's precision for the bugs in our study and seem to have gained trust, which led to their willingness to try to understand the diagnosis and rely on it. AVICENNA's output is a logical specification language that is, although complex to understand for human users, structured and parsable by machines. Thus, we expect future research to achieve an explanation pipeline that produces easy-to-understand explanations based on precise and reliable formalisms.

References

[1] Ann Blandford, Dominic Furniss, and Stephann Makri. 2016. *Qualitative HCI Research: Going Behind the Scenes.* Springer International Publishing. doi:10.1007/978-3-031-02217-3

[2] Martin Eberlein, Marius Smytzek, Dominic Steinhöfel, Lars Grunske, and Andreas Zeller. 2023. Semantic Debugging. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (New York, NY, USA, 2023-11-30) *(ESEC/FSE 2023)*. Association for Computing Machinery, 438–449. doi:10.1145/3611643.3616296

[3] Fatih Gedikli, Dietmar Jannach, and Mouzhi Ge. 2014. How should I explain? A comparison of different explanation types for recommender systems. 72, 4 (2014), 367–382. doi:10.1016/j.ijhcs.2013.12.007

[4] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Human Mental Workload*, Peter A. Hancock and Najmedin Meshkati (Eds.). Advances in Psychology, Vol. 52. North-Holland, 139–183. doi:10.1016/S0166-4115(08)62386-9

[5] Alexander Kampmann, Nikolas Havrikov, Ezekiel O. Soremekun, and Andreas Zeller. 2020. When does my program do this? learning circumstances of software behavior. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (New York, NY, USA, 2020-11-08) *(ESEC/FSE 2020)*. Association for Computing Machinery, 1228–1239. doi:10.1145/3368089.3409687

[6] Scott M. Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2017-12-04) *(NIPS'17)*. Curran Associates Inc., 4768–4777. https://dl.acm.org/doi/10.5555/3295222.3295230

[7] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-Precision Model-Agnostic Explanations. 32, 1 (2018). doi:10.1609/aaai.v32i1.11491

[8] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2016-08-13) *(KDD '16)*. Association for Computing Machinery, 1135–1144. doi:10.1145/2939672.2939778

[9] Sharath M. Shankaranarayana and Davor Runje. 2019. ALIME: Autoencoder Based Approach for Local Interpretability. In *Intelligent Data Engineering and Automated Learning – IDEAL 2019* (Cham, 2019), Hujun Yin, David Camacho, Peter Tino, Antonio J. Tallón-Ballesteros, Ronaldo Menezes, and Richard Allmendinger (Eds.). Springer International Publishing, 454–463. doi:10.1007/978-3-030-33607-3_49

[10] Marius Smytzek, Martin Eberlein, Batuhan Serçe, Lars Grunske, and Andreas Zeller. 2024. Tests4Py: A Benchmark for System Testing. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering* (New York, NY, USA, 2024-07-10) *(FSE 2024)*. Association for Computing Machinery, 557–561. doi:10.1145/3663529.3663798

[11] Dominic Steinhöfel and Andreas Zeller. 2022. Input invariants. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (New York, NY, USA, 2022-11-09) *(ESEC/FSE 2022)*. Association for Computing Machinery, 583–594. doi:10.1145/3540250.3549139

[12] Simone Stumpf, Vidya Rajaram, Lida Li, Margaret Burnett, Thomas Dietterich, Erin Sullivan, Russell Drummond, and Jonathan Herlocker. 2007. Toward harnessing user feedback for machine learning. In *Proceedings of the 12th international conference on Intelligent user interfaces* (New York, NY, USA, 2007-01-28) *(IUI '07)*. Association for Computing Machinery, 82–91. doi:10.1145/1216295.1216316