

The Discreet Charm of the Bugeoisie: A First Look at Bug Reports Created by Researchers

Ji YOUNG KIM, University of Illinois Urbana-Champaign, USA
JANA DRAGOVIC, University of Illinois Urbana-Champaign, USA
ALESSANDRO BOTTA, University of Texas at Dallas, USA
T. M. RITHWANUL ISLAM, Jahangirnagar University, Bangladesh
ALAA MOHAMAD, American University of Beirut, Lebanon
KARIM SHARAF, Egyptian E-Learning University, Egypt and Alexandria University, Egypt
SEJUTI SHARMIN SIDDIQUI, University of Dhaka, Bangladesh
DIVYANSHI JOSHI, Maharaja Agrasen Institute of Technology, India
HARINI ANAND, PES University, India
NURJEMAL SARYYEVA, National University of Singapore, Singapore
SHUBHAM CHAPAGAIN, Tribhuvan University, Nepal
SAAD NASIR, American International University-Bangladesh, Bangladesh
DARKO MARINOV, University of Illinois Urbana-Champaign, USA
BOGDAN ALEXANDRU STOICA, University of Illinois Urbana-Champaign, USA

Many ISSTA papers present automated tools to find software bugs. Some researchers create bug reports for the bugs they deem worth reporting, e.g., new bugs found by the tools. We study *bug reports created by researchers* and coin the term *Bugeoisie* to refer to such bug reports. We collect the Bugeoisie for ISSTA'25 by reading the papers, examining the artifacts, and contacting the authors, trying to “chase” bug report URLs. This process turns out to be surprisingly challenging because only a few authors make such URLs readily available. We then analyze this Bugeoisie to understand potential bad practices. For most papers that claim some bugs reported or confirmed, independently checking those claims is again surprisingly challenging. We also start analyzing Bugeoisie for FSE'25 to check the generalizability of our findings. Our study points out some changes that conference organizers and reviewers may want to require for Bugeoisies. We also provide some suggestions for paper authors to prepare and describe their Bugeoisie in their papers and artifacts.

CCS Concepts: • **Software and its engineering** → **Software testing and debugging**; • **General and reference** → *Empirical studies*.

Additional Key Words and Phrases: Software Bugs, Bug Reports, Bugeoisie, Reproducibility

Authors' Contact Information: Ji young Kim, University of Illinois Urbana-Champaign, Urbana, USA, jyk14@illinois.edu; Jana Dragovic, University of Illinois Urbana-Champaign, Urbana, USA, janad2@illinois.edu; Alessandro Botta, University of Texas at Dallas, Richardson, USA, alessandro.botta@utdallas.edu; T. M. Rithwanul Islam, Jahangirnagar University, Dhaka, Bangladesh, rithwanul@gmail.com; Alaa Mohamad, American University of Beirut, Beirut, Lebanon, alaamohamma68@gmail.com; Karim Sharaf, Egyptian E-Learning University, Giza, Egypt and Alexandria University, Alexandria, Egypt, sharafkarim61@gmail.com; Sejuti Sharmin Siddiqui, University of Dhaka, Dhaka, Bangladesh, sejutiharminn@gmail.com; Divyanshi Joshi, Maharaja Agrasen Institute of Technology, New Delhi, India, divyanshi-joshi2610@gmail.com; Harini Anand, PES University, Bangalore, India, harini.anand@gmail.com; Nurjema Saryyeva, National University of Singapore, Singapore, Singapore, saryyevanurjema@gmail.com; Shubham Chapagain, Tribhuvan University, Kathmandu, Nepal, shubhamchapagain01@gmail.com; Saad Nasir, American International University-Bangladesh, Dhaka, Bangladesh, snasir2194@gmail.com; Darko Marinov, University of Illinois Urbana-Champaign, Urbana, USA, marinov@illinois.edu; Bogdan Alexandru Stoica, University of Illinois Urbana-Champaign, Urbana, USA, bastoica@illinois.edu.



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

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

ACM 2994-970X/2026/10-ARTISSTA153

<https://doi.org/10.1145/3832244>

ACM Reference Format:

Ji young Kim, Jana Dragovic, Alessandro Botta, T. M. Rithwanul Islam, Alaa Mohamad, Karim Sharaf, Sejuti Sharmin Siddiqui, Divyanshi Joshi, Harini Anand, Nurjema Saryyeva, Shubham Chapagain, Saad Nasir, Darko Marinov, and Bogdan Alexandru Stoica. 2026. The Discreet Charm of the Bugeoisie: A First Look at Bug Reports Created by Researchers. *Proc. ACM Softw. Eng.* 3, ISSTA, Article ISSTA153 (October 2026), 22 pages. <https://doi.org/10.1145/3832244>

1 Introduction

Automated bug-finding techniques and tools are among the central contributions of software testing research. The goal of our research community is—or should be—to eventually impact the practice (and train the new generations of researchers along the way) [35]. Besides the adoption of our techniques and tools by practitioners, one directly impactful external outcome of our research is when the tools lead to actionable bug reports for publicly available software. Public reports are important (1) for practice, because they help developers improve software quality and (2) for future research, when they are independently verifiable, because others can build on prior work.

To clarify some terminology early on, we use the term *bug report* (for short, just *report*) to refer to a unit of online communication that informs developers about some problem(s) in their project, e.g., a report may be created (1) on GitHub as (1.1) a GitHub issue just reporting the problem or (1.2) a pull request also providing a fix; (2) on Jira as a Jira ticket; or (3) on the Linux Kernel Mailing List as an email. We use *bug* to refer to the underlying problem in the project, what various papers may call “faults”, “defects”, “inconsistencies”, “vulnerabilities”, or “issues”.

Distinguishing between a report and a bug is important. One report may inform developers about multiple bugs, although counting bugs precisely is challenging [23, 45]. For example, for the 23 papers in our study, we collect 1,271 potentially related report URLs, determine 844 are actually related, analyze them in detail, and find they correspond to 1,460 bugs (on average, 1.73 bugs per report). Most papers state claims in terms of the number of bugs, e.g., “found *X* bugs, reported *Y*, got *Z* fixed”. The authors have an incentive to claim as high numbers as possible: more bugs found or fixed increases the chances to get a paper accepted and raises the researcher’s reputation.

While researchers create reports, our research community does not always treat these reports as first-class artifacts and does not give them the same attention as to the papers, techniques, and tools that help find the bugs described in the reports. Our study reveals that paper claims about bugs are difficult to verify because the bug-reporting evidence trail is often incomplete or costly to reconstruct. We thus evaluate this gap, identify several potential bad practices, quantify them (per the reviewers’ requests), and provide actionable suggestions to prevent evidence getting lost, thus enabling cumulative research to build on prior work. We do **not** intend to cast doubt on any specific paper or author. Due to *potential* reputation harm to the authors, we **anonymize the papers** we study: we do *not* quote from the papers or reveal titles and authors. A reader cannot easily map our claims to the authors but may still be able to de-anonymize some authors with an effort.

In this study, we focus on the reports created by the authors of the ISSTA’25 papers, presumably to inform developers about *new*, previously unknown bugs. We coin the term *Bugeoisie*¹ to refer to such reports, because they are a novel concept to be studied. Some papers also involve experiments with *old*, previously known bugs to show how well some new techniques and tools could have worked in the past. We do *not* check claims that the papers make about tools (e.g., did it really find *X* bugs); instead, we focus only on the corresponding, developer-facing Bugeoisie, attempting to collect and analyze reports. Our goal is to evaluate the research process around the Bugeoisie and suggest how it could be improved. To our knowledge, this process has never been systematically

¹The name comes from the movie titled “The Discreet Charm of the Bourgeoisie” which questions the virtues of the rich, because our study questions the verifiability, and potentially veracity, of claims made by researchers about their bug reports.

studied, although some prior work [36] pointed out concrete, anecdotal problems with reports created by researchers, and extensive prior work studied reports in general (Section 9).

Challenges. Our attempt to compile the complete ISSTA’25 Bugeoisie revealed frequent problems: papers claim that the authors reported a number of bugs, yet a reader cannot reliably locate, validate, or attribute the corresponding reports created by the ISSTA’25 paper authors, even after substantial investigative effort. We strongly believe that *almost all authors intend for openness and accurate statements about their research results*, but mapping a paper to specific reports, and reports to bug counts, often requires deep “report archaeology” while still yielding partial report set reconstruction. When we started this project, we naively expected that a dozen undergraduate students could collect most reports and bugs in just 2–3 weeks, hoping to convert fixed bugs to the SWE-Bench format [27, 47] and evaluate AI tools on these reports. Unfortunately, we ran into many obstacles.

For many papers, the bug evidence is not clearly published, forcing reconstruction by searching bug-tracking systems, inspecting tool repositories, investigating CVE reports, contacting the authors, or triangulating across authors’ activity on platforms such as GitHub. This effort is inherently manual, incomplete, and time-consuming: our initial expectation of 2–3 weeks became months and continued to run into challenges. Surprisingly, even simple summary questions like “how many new bugs this paper reported” are sometimes difficult to answer from the official proceedings or even the accompanying artifacts. Moreover, some papers have obvious inconsistencies, e.g., claiming one number of bugs in the abstract or introduction and another in a table caption or conclusions.

Across papers, counting bugs is often challenging because the terminology used is inconsistent, e.g., making it unclear how the authors handle false alarms or duplicate bugs, whether they clearly separate or mix “found”, “reported”, “confirmed”, “pending”, “rejected”, and eventually “fixed”. Moreover, some papers count bugs that are not previously unknown (e.g., false positives, previously known, or synthetic examples), so “tool found X new bugs” may not be the correct terminology to describe those, and the papers should more explicitly explain the terminology. Some papers that count together some newly discovered and previously known bugs may be making scientifically valid and justified claims (e.g., to fairly compare with prior work), but the writing often confuses readers when lumped up under phrases such as “new bugs found and reported”.

We acknowledge that some reports cannot be shared publicly for legitimate reasons such as bugs in proprietary code or security vulnerabilities. However, even for security vulnerabilities, the most widely followed coordinated vulnerability disclosure process [24] calls for keeping the reports private for a limited time (e.g., 90 days for Google Project Zero [20]) and then making them public so that the users can appropriately protect from these vulnerabilities. Some email responses we received from the authors stated that they cannot share their reports because they are for security vulnerabilities, but the emails were a year after ISSTA’25 (and potentially much longer after the initial report the authors submitted to the developers). Some papers place much more emphasis on the bugs that get assigned identifiers such as CVEs (Section 5.2). Ideally, authors should provide information for all their reports, whether assigned CVEs or not.

We believe that the Bugeoisie process is important to investigate, understand, and hopefully improve over time. The lack of reports and their clear mapping to bugs hinders follow-up research and building on top of prior results. Without reliable collections of reports, other researchers cannot easily reproduce bugs, inspect the developers’ responses, study various bug characteristics (e.g., root causes, manifestations, mitigation patterns, fix latency) to understand trends, and build novel techniques and tools to find bugs faster. Comparability and interpretability also suffer, because conflating between the numbers of “found”, “reported”, “confirmed”, or “fixed” bugs makes it difficult to evaluate claims about impact and compare with similar papers and tools.

Contributions. Our study is the first to systematically focus on the Bugeoisie—bug reports created by researchers—as a distinct class of artifacts located at the intersection of research evaluation,

reproducibility, and real-world software maintenance. Prior work in our and adjacent research areas has raised concerns about reproducibility and replicability [15], leading to changes at the ACM [2] level and efforts such as artifact evaluation [5] and reproducibility events [34].

Specifically, we construct a dataset for the ISSTA'25 Bugeoisie, including report URLs when available. Using this dataset, we identify traceability gaps and unclear counting criteria that make it difficult to independently check claims about bug counts. We also analyze the status of reports to identify how developers respond vs. how the authors count bugs. Our study identifies and quantifies several bad practices (Section 4.2). To check generalizability of our findings, we also present an initial analysis of FSE'25 papers (Section 8).

Finally, we distill these results into concrete recommendations, including conventions for documenting provenance and outcomes that improve checkability, replicability, and comparability. We summarize those recommendations for conference organizers, paper authors, and our community:

- Conference committees in general, and Artifact Evaluation committees in particular, should require auditable reporting evidence and check quantitative claims that papers make about bugs, e.g., “found X, reported Y, fixed Z”.
- Conference committees could consider establishing disclosure-aware placeholders for non-public reports, requiring authors to update them when disclosure becomes possible, and preserving independent verifiable metadata when disclosure is impossible.
- Authors should disclose *all* reports *they create*, including the subset rejected by developers, using explicit status categories (e.g., Fixed, Confirmed, Pending, Rejected).
- Authors should include, in the paper or artifact, traceable report evidence in a persistent form that remains checkable as bug trackers evolve (e.g., canonical URLs, creation dates, reporter names, statuses and, ideally, a point-in-time snapshot at a declared cutoff).
- Our community should standardize the terminology and counting rules used in bug claims, with disjoint statuses and explicit counting criteria (e.g., Section 3.7 could be a start).
- Our community may facilitate pre-registration of the evidence used for paper claims at a cutoff date, along with the measurement methodology, including the bug-counting policy.
- Our community should set reasonable expectations for bug-finding papers regarding the acceptance and rejection rates of created reports to encourage more transparent disclosure.

2 Examples Illustrating Challenges and Our Lenient Labeling

We show two examples to illustrate some concrete challenges in collecting key information on bugs and reports in a Bugeoisie. Papers often claim bug counts per status, e.g., “X confirmed, Y fixed”, but mapping report information (if found at all) to the number of bugs or status can be non-trivial. We discuss two positive examples where the authors created commendable reports.

Umka-lang PR #285. Our Bugeoisie includes a report about memory management [42] that shows why counting bugs per report is challenging. This report is a PR with four changes, one adding a NULL check after a `malloc` call, and three ensuring memory is freed along related exit paths. From a reporting perspective, grouping these in one PR is well justified. One could count this report as describing (1) one bug (with multiple hunks changed in the commit but in a related manner), (2) two bugs (one about `malloc` and one about `free`), or (3) four bugs (as there are four hunks, and each instance of `free` can be viewed as a separate bug). In general, we aim to be *lenient* to the authors whenever possible, so we counted this report as four bugs. We would have not known if our count matches the paper authors’ had they not released theirs. Fortunately, the paper comes with an artifact, showing this case counted as four bugs, so we could confirm the match.

The developer initially responds non-uniformly across the four items, (1) acknowledges the missing NULL check as symptomatic of a broader, deeper problem and points to a previous issue

“this is only the tip of the iceberg; the problem is much deeper; see #212”; and (2) argues the proposed free calls unnecessary *“it makes little sense to free anything just before returning from main”* [42].

This mixed response introduces layers of ambiguity. Should PR #285 be treated as invalid, partially valid, or duplicate? Is the acknowledged NULL check a self-contained bug that merits its own fix, or merely a manifestation of the broader problem tracked in #212? More generally, how should we count bugs when a single report contains several concerns that receive different determinations?

Furthermore, we later found a related report (not from our Bugeoisie), issue #461 [43], which also mentions, among other items, the missing NULL check. The developer again treats it as part of the broader issue #212, but the metadata differs. While our studied PR #285 had no GitHub label, the broader #212 was labeled *bugs*, whereas #461 was labeled *enhancement*. Thus, even when developers acknowledge essentially the same underlying bug, different reporters or labeling practices can yield inconsistent labels [22], further challenging analysis.

Following our main principle to be lenient, we counted this case as four Confirmed bugs. We sympathize with the authors who had to decide how to count this response. We found that they counted this case as four Pending bugs, so we could confirm that their labeling is reasonable (e.g., Fixed or Can’t reproduce would definitely not apply here).

Libplacebo issue #326. Another example illustrating the difficulty of mapping reports to the number of bugs is libplacebo issue #326 [32]. The reporter states finding *“two issues”* while fuzzing. A closer reading shows these two items are distinct: one is a build-related change to the fuzzing harness (i.e., updating deprecated header usage), while the other is an actual out-of-bounds access found by the fuzz harness. If we simply counted both items as bugs, this single report would contribute two bugs. However, under a stricter interpretation consistent with counting new bugs after manual review, only the out-of-bounds bug clearly qualifies.

In the context of our evaluation, this difference matters. We try to match the number of new bugs that the paper’s authors claim discovering after manual review of fuzzing results, while explicitly noting that not all fuzzer-triggered errors correspond to true bugs (e.g., some are spurious crashes due to API misuse, timeouts, or out-of-memory cases). Under these circumstances, a counting mismatch could occur if we simplistically treat all heterogeneous items within a single report as separate new bugs. For instance, if we counted issue #326 as two bugs because it mentions *“two issues”*, but later only one of those items is judged to be a genuine bug, then the aggregated total derived from our report counts would drift from the paper claim, yielding an apparent inconsistency (namely, us potentially finding even more bugs than claimed in the paper).

Summary. These cases illustrate some challenges in analyzing a Bugeoisie. A single report may bundle multiple bugs that could be evaluated separately and lead to different outcomes. Responses can be ambiguous (e.g., part of a larger issue, compilation fixes) and may change over time as developers make fixes elsewhere in the codebase or revisit earlier judgments. Even distinguishing one bug from another can depend on how one groups bugs and the underlying code fragments that may cause them [23]. Whether a reported behavior counts as a bug and how it is triaged reflects subjective judgment rather than an objective property of the report [22, 45, 48].

3 Methodology

Our main goal is to investigate how papers account for the *new, previously unknown* bugs that the authors claim to have reported to developers. We perform analyses at both paper level and report level. We first identify the claims that each paper makes about new bugs, recover the reports that should support those claims, and examine whether the recovered reports support a complete and consistent accounting. We collect report-level metadata to characterize individual reports and to summarize the results at the paper level.

3.1 Identifying Papers That Claim New Bugs

We started from all 107 full research papers published at ISSTA'25. We first read each paper carefully to identify how many new bugs the authors claim to have reported to developers. When doing so, we tried to distinguish reports likely newly created by the authors from reports previously created by others and only analyzed by the authors, e.g., as inspiration for developing the new technique.

Unfortunately, even such straightforward questions turned out to be much harder than expected to answer precisely. For some papers, it was difficult to understand whether the authors claim some specific reports as newly created by them or as older reports created by others. For instance, we could not determine for one paper, even after multiple readings, whether the authors created any new reports, but after we emailed the authors, they replied that they did not create any.

After collecting the data manually, we evaluated whether a large language model could have helped with this task. We found GPT-4o unreliable even for the coarse-grained task of determining whether a paper claims reporting any newly found bug, achieving under 70% F1 score, and substantially worse on the finer-grained task of determining how many new bugs a paper claims as reported. We therefore rely on mostly manual inspection for the analysis in this paper.

We identified 23 papers that claimed finding new bugs previously unreported in the repositories that they analyzed. We aimed to collect our Bugeoisie for all these 23 papers.

3.2 Recovering Reports Corresponding to Paper Claims

For each of the 23 papers, we attempted to recover the reports corresponding to the bugs that the paper claimed as reported. We followed URLs from tables and prose in both papers and artifacts, inspected associated repositories, checked reports created by likely author accounts, used the Internet Archive [4] to recover some deleted cases, and finally emailed authors when necessary.

Our initial inspection revealed some anecdotal instances of potential *bad practices*—e.g., papers claiming reports not created by their authors, papers not mentioning rejected reports that appear to be part of the same bug-finding effort, and duplicate reports potentially counted as unique bugs. Based on early feedback from several members of our Software Engineering community, we systematically checked *all* potential instances rather than reporting only anecdotal cases.

When a paper PDF, artifact, or authors' email to us disclosed some report URLs, we checked whether the account that created each report likely belongs to a paper author: the real name on the account matched a paper author, or the username matched the tool/project name introduced in the paper. When this was not clear from the account itself, we used publicly available information (e.g., personal webpages, social media) to determine whether the reporter was likely a paper author. If no public link could be established, we did not infer that the account belonged to an author.

When a paper provided no report URLs, we inspected other paper-related links, such as URLs to artifact repositories, to identify likely author accounts. For one paper that linked only to a GitHub repository for the tool, we used both contributors of that repository as candidate author accounts.

For users likely to be paper authors, we also collected the user's historical activity when it was public, e.g., on GitHub using <https://api.github.com/search/issues?q=author:{username}>. This search allows us to identify potentially relevant rejected reports omitted from the paper's bug accounting, which cannot be found by inspecting disclosed reports alone. We searched for all reports created by that user (e.g., GitHub issues and pull requests), across all repositories on the same platform up to the ISSTA'25 camera-ready deadline (CRD), 2025-04-24. Reports created after the CRD could not have been included in the camera-ready paper. However, reports created before the CRD cannot automatically be attributed to the paper being analyzed, because an author may be working on multiple unrelated research projects. We therefore carefully inspected candidate reports before the

CRD to determine whether they are relevant to the paper. If an account's activity was private or not searchable across repositories, we could not perform this additional historical search.

3.3 Email Experience

Before our paper was accepted, our extensive efforts to collect report URLs partially succeeded for only 18 papers. The authors of the other 5 papers did not share any report URLs. We emailed authors of 15 papers that claimed new bugs but lacked some public report URLs; only 10 replied. 6 shared some URLs via email, although for only 2 papers these URLs had been previously unknown to us. 3 made some of their URLs publicly available; we thank these authors because the community can benefit from having access to these reports for future research. One replied that all their reports are private. One stated that they created no report. Even authors who shared some URLs did not necessarily share all, and several replies explicitly stated they did not track all their report URLs.

Final Emails. The situation changed after our paper had been accepted. We emailed authors with the subject "*potential reputation harm from <YourTool>*", describing our study, stating how our ISSTA'26 paper would *anonymously* list their paper, and asking about specific reports (which indicated potential bad practices we found in their paper). For *all* 23 papers, at least one author replied this time! Some provided report URLs or even private information. Some argued why their cases should not count as a bad practice. Some reiterated that they have no more public information to share. We include some quotes from these emails as they do not easily reveal the authors.

3.4 Collected Reports and Scope of Inspection

Before our final emails, we collected 1,271 URLs to reports likely created by the paper authors *or* relevant to the corresponding paper, on 20 various platforms: GitHub, Jira, Gitee, etc. Of these reports, 742 were from URLs *disclosed* in PDFs, artifacts, or authors' emails to us, and 529 were *undisclosed*, recovered from authors' activity histories. 198 URLs were for feature requests not reports for new bugs found by the tools. 11 URLs were inaccessible, returning HTTP 404 Not Found.

We systematically inspected all 1,062 remaining bug reports (including 6 we got from the Internet Wayback Machine [4]). For every inspected report, we first determined whether it was likely created by one of the paper authors *and* whether it was relevant to the corresponding paper. The latter step is needed because an author may have created reports for other, unrelated research projects. We then analyzed in detail 844 reports that were relevant to the corresponding paper.

We collect two types of information: (1) per-paper information for each of 23 papers and (2) per-report metadata for 844 reports that we could link to the 18 papers for which we recovered some reports. Disclosed reports (in paper PDFs, artifacts, or authors' emails) are directly connected to the claims made by the paper. For these reports, we collected richer metadata not only to compare the paper's claims with the bug-tracker evidence (e.g., discussion, developer resolution, report status) but also to support some more open-ended analyses. Reports recovered only from authors' activity histories were inspected to determine whether they were relevant to the paper and thus revealed potential bad practices, e.g., relevant rejected reports undisclosed by the authors (RRU) or duplicate reports counted as unique bugs (CDU). We did not automatically treat all activity history reports as relevant to the papers, because such reports may belong to unrelated research projects.

3.5 Annotation Procedure and Resolution of Disagreements

We collected some report information manually and some automatically. 13 annotators contributed to our manual data collection for a period of 6+ months. Annotators were provided with detailed written guidelines describing the semantics of each field and the procedure to follow. During data collection, annotators examined the paper text, artifacts when available, associated reports,

related tracker discussions, and linked code changes. We used automated scripts to collect other information such as platform metadata exposed through APIs or report pages.

All key information related to potential bad practices was labeled independently by two *senior* annotators from our team, including whether (1) a report was relevant to the paper, (2) a reporter was likely a paper author, (3) a report was rejected or marked duplicate, (4) multiple reports appeared to be about the same underlying bug, and (5) report status was consistent with the paper's accounting. For paper relevance and author identity, the inter-rater agreement was high, Cohen's κ 0.88 and 0.92, respectively. However, for report status, inter-rater agreement was lower, Cohen's κ 0.81 at CRD and 0.79 one year later, reflecting the inherent difficulty of assigning a single status to reports with ambiguous or mixed developer responses [22].

When two labels disagreed, the two annotators first discussed the report and aimed to choose the interpretation *more lenient* to the paper authors. For example, when multiple interpretations of a report status or of the number of bugs represented by a report were plausible, we chose the interpretation matching most closely what the authors claimed in the paper. If the two annotators remained unsure, the most senior researcher in our team read the report, the notes, the related tracker discussion, and the corresponding paper text. The three annotators then discussed the case, with some lively discussions especially about status, and assigned the final consensus label.

3.6 Data Collection and Labeling

We could analyze either a historical snapshot of the report trackers or live data collected from the corresponding platforms. A traditional approach may download HTML or JSON files of reports at a particular point in time and analyze only that snapshot. Instead, we wrote scripts to collect *live data* from the platforms, e.g., using platform-specific APIs to collect the most recent information, and we archived the collected data in a snapshot from 2026-04-24 (for our private reproducibility).

This methodology provides two benefits. First, our scripts can analyze the most recent information observable at the time of collection, rather than being limited to an older historical snapshot. Second, the archived snapshot makes our analysis more reproducible. For example, one might want to study which GitHub pull request closed which GitHub issue. This information is displayed in the HTML pages that GitHub renders for issues under the tag Completed in our 2026-04-24 snapshot, but was not observable in an earlier snapshot. An analysis based only on that earlier snapshot would therefore miss some relationships between issues and their fixes.

Per-paper information. For each of the 23 papers, we carefully read the paper and extracted information about the claimed number of reported bugs, the kind of bug-finding analysis used, the use of AI, and the artifact evaluation badges received by the paper.

Bug-count metadata. Annotators inspected the paper and recorded the total number of reported bugs claimed in the paper. Annotators then attempted to find URLs for reports created by the paper authors for these bugs and determined the likely number of bugs for each recovered report. In most cases, a report is for one claimed bug. When a report addressed multiple bugs, annotators determined the number of bugs by carefully reading the paper, the report, and related artifacts.

For some papers, the sum of the number of bugs across recovered reports was *greater* than the number of bugs claimed in the paper. These cases required iteratively checking and refining the extracted numbers, the links between papers and reports, and the relationships among multiple reports that may correspond to the same underlying bug.

Overall, we counted 1,460 bugs from the inspected 844 reports (on average, 1.73 bugs per report).

Bug-finding analysis metadata. Annotators recorded the analysis type the paper used to find bugs. For simplicity, we restricted this classification to static, dynamic, or hybrid analysis. Annotators also investigated whether the paper relied on AI or LLMs at any step of its bug-finding process.

Artifact evaluation metadata. Annotators collected information about artifact evaluation for each paper by inspecting the official conference proceedings and recording the badges listed there. Our follow-up check with the co-chairs of the ISSTA'25 Artifact Evaluation Committee found that the proceedings have one false positive and one false negative for the badges.

Per-report metadata. For each of the 844 reports, we collected structured metadata that allows us to evaluate the paper's bug accounting and help us identify *potential bad practices*.

Report origin and relation to claimed bugs. Annotators recorded the URL of the original report and identified how many newly claimed bugs in the paper were covered by that report. When a report was connected to other reports, such as a pull request fixing an issue created by the authors, we inspected those related reports as well. To avoid double counting bugs, we use for further bug-level analysis only one report from each set of related reports corresponding to the same underlying bug.

Duplicate reports. We handled duplicate reports specially because they could inflate bug counts. If the authors created a report that developers marked as a duplicate of a report created by a different reporter, we still analyzed the authors' report but recorded its status as duplicate. If the authors created multiple reports and developers marked some as duplicates of another, we analyzed only one report per duplicate equivalence class and labeled the remaining reports as duplicates rather than additional unique bugs. In both cases, the duplicate report may correspond to a real bug, but it should not count as a unique new bug without further justification in the paper.

Report status metadata. For each report, annotators recorded the report's status at two dates: the CRD (2025-04-24) and one year after the CRD (2026-04-24). The former captures information observable before the final paper version, and the latter captures later developer responses.

3.7 Report Status Terminology

The papers we study use a variety of terms for reports and statuses. Each report is first *created*, which papers also called "reported", "submitted", "disclosed", "found", "identified", "discovered", or "detected". We use just four coarse-grain status categories. Some reports are (1) Confirmed (also called "accepted" or "positive response") as real bugs and can be further (2) Fixed (also called "accepted", "resolved", or "merged"); some get (3) Rejected (also called "denied" or "duplicate"); and some are (4) Pending final resolution. Some papers use more terms to indicate reports proceeding from Confirmed toward Fixed; we categorize those as Confirmed. Different papers use the same term with different meanings (Section 4.1), and some papers even use the same term with two meanings (e.g., "reported" meaning both what was created for developers and what is still pending).

For each report, we assigned one of our four coarse-grained categories: Fixed, Confirmed, Pending, and Rejected. For rejected reports, annotators inspected the discussion to identify the specific closure reason: Invalid, developers deemed the report a false alarm; Can't reproduce, developers could not reproduce the problem; Won't fix, developers did not intend to address the reported behavior; Inactive, the report was closed automatically or without thorough developer review; and Duplicate, developers deemed the report as corresponding to a previously reported problem.

To not overestimate potential bad practices, we apply *criteria lenient to the authors*. For example, if a developer added a label indicating a bug, e.g., a label literally named "bug" or whose description indicated bugs, we treat the report as Confirmed. If another user, even if not a project contributor, commented about the same or a similar problem, we also treat the report as Confirmed. We also treat cross-references from reports created by others as confirmation when describing the same underlying problem. Finally, if a developer stated that the bug may not exist and suggested checking a newer version, we carefully consider the report to determine whether to treat it as Confirmed as well or Rejected (because the bug had been fixed before the author's report).

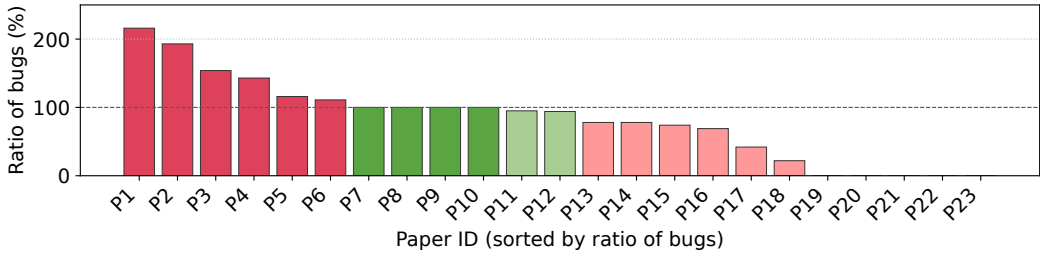


Fig. 1. Ratio of bugs collected from the reports in our Bugeoisie over bugs claimed (>100% is *not our* mistake!).

3.8 Using Report Metadata to Identify Potential Bad Practices

The report metadata enabled us to compare each paper’s claims with the bug-tracker evidence. In particular, we systematically checked whether reports claimed by papers were likely created by their authors, whether relevant rejected reports created by the authors were absent from the paper, whether duplicate reports were counted as unique new bugs, and whether report outcomes in papers or artifacts were consistent with developer resolutions.

We emphasize that we used activity history reports carefully in our analysis. The fact that an author created a report before the CRD does not by itself show that the report is relevant to the paper being analyzed. We treat such reports as relevant only after manual inspection of their content, timing, repository, relation to the technique, and surrounding activity. Similarly, a rejected or duplicate report does not by itself establish problematic reporting. Our concern is *incomplete or inconsistent accounting in papers* when such reports appear to belong to the same bug-finding effort but are not reflected in the paper’s presentation of its results.

4 Paper Analysis

4.1 Bug Counts for Recovered Reports

Figure 1 shows the number of reported bugs each paper claims versus the total number of bugs in the reports we recovered. We **anonymize the papers** using the paper ID (PID). While each paper makes claims about *new bugs*, we deliberately do not show the absolute claimed numbers of new bugs in Figure 1. Such numbers would de-anonymize papers, and the phrase “new bug” is used inconsistently across papers, making direct comparison misleading without further qualification.

Across our dataset, we observed three interpretations of a “new bug”: (1) confirmed-only, which are cases validated by developers; (2) confirmed and pending, which additionally includes reported cases still awaiting a decision from developers; and (3) confirmed, pending, and *rejected*, which appear to include even reports that had been already *rejected* by developers. These interpretations lead to different measurements and conclusions. Nevertheless, using the paper PDF, tables, figures, and artifacts as the source of claimed counts, and issue trackers as the source of recoverable reports, we observe some notable gaps between the claims and the supporting evidence we could recover.

Overall, the 23 papers claimed reporting a total of 2,135 new bugs, but from their public reports disclosed (via the paper PDFs, artifacts, or authors’ emails) whose URLs remain accessible, we recovered only 1,281 bugs (60%). We could not recover bugs from 11 disclosed yet inaccessible reports; to be lenient, we defer to the authors’ claims and *not* count the 11 towards any bad practice. We also recovered 179 bugs from the authors’ activity history, of which 63 bugs (from 62 rejected reports) were not counted among the new bugs in the papers, while the others may have counted.

For four papers (P7, P8, P9, P10), we recovered the same number of bugs as the authors claimed they reported. These papers tend to align with the confirmed-only interpretation of “new bugs”,

Table 1. Potential bad practices: ID, description, number of papers, reports, and bugs. For anonymity of the 2 papers with 90%+ reports recovered, we provide an upper bound for the MRU bugs.

identifier	potential bad practice description	#papers	#reports	#bugs
MRU	missing report URLs for some bugs	11	n/a	<822
RRU	rejected relevant reports undisclosed in the paper	6	62	63
NAR	non-author reports claimed in the paper	4	16	20
CPF	counting previously fixed bugs as new	6	10	12
CDU	counting duplicate reports as unique bugs	3	7	7
RSM	report status mismatch between claimed & our analysis	4	10	30
Total	<i>unique</i> papers, reports & bugs with a non-MRU bad practice	12	100	125

and provide enough details to make bug claims easy to validate. We would like to explicitly list the authors of these exemplary papers, but it would weaken the anonymization of the other papers.

For another two papers (P11, P12) we are able to recover over 90% of the reports. Although not complete, this high recoverability still makes much of their reporting independently verifiable.

For six papers (P1, P2, P3, P4, P5, P6) on the left-hand side of Figure 1, we recovered *more* bugs in author-created reports than the papers claim. Our inspection revealed no plausible reason for excluding many of these reports. More concernedly, papers of this type often adopt the most problematic interpretation of “new bugs” by tallying rejected reports in their counts. Even if authors disagree with a developer’s determination, counting rejected reports as new bugs without clearly describing the reasons undermines the credibility and impact of the paper results.

4.2 Potential Bad Practices

The discrepancies above are not limited to missing report URLs or minor counting errors. During our analysis we observed several recurring practices that can make paper claims difficult or even impossible to verify. We stress that we treat those as **potential bad practices**, not as evidence of ill intent. Many cases have benign explanations, e.g., authors working on related research projects in parallel, reporting bugs via private channels, or omitting details due to space limit. Still, these practices affect the readers’ ability to determine how many bugs were found by the tools, whether they were new (previously unreported) bugs, and how the developers ultimately resolved them.

Table 1 provides a summary of the six bad practices, omitting PIDs for anonymity. For MRU, we cannot analyze missing reports. For the other five, we find a total of 100 reports with some bad practice (5 reports with two bad practices each) across 12 papers. Per paper, the number of bad practices ranges from 1 to 3, and the number of reports with a bad practice ranges from 1 to 32.

MRU: Not being able to recover all report URLs. For 13 out of 23 papers, we did not find all bugs/reports; for 2 papers we could recover over 90% of bugs, so being lenient, we consider only 11 papers as having MRU. Of these 11, we could inspect some reports for six. A recurring pattern for these six is that claims about new bugs align more with *confirmed and pending* interpretation—i.e., these papers count bugs “new” even when developers have not made a determination or even commented at all. Thus, the count in “found X new bugs” can diverge over time as developer decisions accumulate, increasing or decreasing bug counts based on pending reports being confirmed or rejected. Nevertheless, not being able to retrieve report URLs for all bugs is, at minimum, a recoverability and transparency concern. It affects validating the core claims about the paper on how many bugs the authors reported and limits the ability to check most other bad practices.

Final Emails. The authors provided several explanations for why we could not recover all their report URLs. Some reports are private for security or proprietary reasons. Some authors stated

that they lost the URLs. Some URLs became inaccessible. Some papers had wrong counts in the PDFs, quoting from emails: *“a couple of statements in our paper [...] are not fully accurate”* and *“[in] ‘reported the [bugs]’, the intended meaning was that we reported a subset of the bugs, not all”*. On the positive, several authors made more of their URLs or reports public, benefiting our community at large. Still, almost 822 bugs remain unaccounted for, in an unknown number of reports.

RRU: Rejected reports relevant to, but undisclosed in, the paper. We count as RRU rejected reports that are created by the authors and undisclosed (in the paper PDF, artifact or non-final email to us) but could have likely been reflected in the paper’s bug-finding totals. To be lenient, we do not count a paper as having RRU if it has *any* relevant undisclosed report Fixed; we assume the authors just forgot to list several reports in their paper. Importantly, we do not treat every rejected author-created report likely related to a paper as an RRU case: an unintentional duplicate, a report opened by mistake and subsequently corrected, or a report concerning ancillary code outside the claimed evaluation scope need not be discussed as part of the paper’s bug-finding results.

We initially found 529 reports from the authors’ activity histories. Of these, we determined 198 as likely relevant to, but undisclosed in the papers. Among those, 108 reports were rejected by developers, while 90 were either accepted or still pending investigation. While we do not count the latter 90 as RRU, they nevertheless provide useful context: incomplete listing of author-created reports is not limited to unfavorable developer responses; papers may omit, for reasons not easily discernible, even reports whose outcomes are positive or still unresolved.

After further inspection and final email correspondence with the authors, we determine 62 reports from six papers as RRU. For four of those papers, we recovered more bugs than the authors claim, with at least one report rejected. For the remaining two papers, we recovered fewer bugs but could only determine that some of their undisclosed rejected reports are relevant to their respective papers *after* emailing the authors. Similarly, we could only determine that some of the reports are in fact *not* related to the paper only after authors provided evidence in our email correspondence, e.g., demonstrating that some were indeed from the authors’ other work.

RRU reports exhibit three patterns: complete omission, apparently selective omission, and incomplete accounting. (1) Three papers omit all rejected reports that appear to be part of the same bug-finding effort as the reports they disclose: for one paper, nearly half of the additional reports recovered from authors’ activity history were rejected. (2) One paper appears to selectively list repos and outcomes—most of its RRU reports are from two repos not even mentioned as evaluated, and which rejected all authors’ reports, while the mentioned repos had some confirmed reports; the other undisclosed reports are from a mentioned repo. (3) Two papers acknowledge some rejected reports but omit additional ones despite being similar to those disclosed in the paper.

Final Emails. The authors provided three apparent reasons for potential RRU reports. (1) Some authors claimed that those reports are from other research projects, but only one set of authors provided the actual project information we could check; all others claimed that the other projects are not public, e.g., under submission or proprietary. (2) Some authors admitted that the reports are from the same project but should not count being from the early project stage, e.g., a prototype tool version that was calibrated to reduce false positives and not actually evaluated in the paper. We pointed all those authors to a paper that somewhat discussed rejected reports even from the early stage. (3) One set of authors stated they were not sure if the reports were real bugs so did not count in the paper: *“[These bugs] were not included [...] due to our insufficient confidence [...] We still submitted the issue report because the potential [insecurity] was critical”*.

Another reply stated *“Following the common practice adopted in other papers, these reports were therefore not included”*. Sadly, the author is correct that many papers silently omit some related reports! We are not sure how the author was aware of this potentially bad practice (without a study like ours). Our community should decide whether to require disclosure of all created reports.

NAR: Claimed reports not obviously created by authors. Four papers disclose reports (in PDF, artifact, or email) that were created by accounts not appearing to belong to any paper author. We make this determination by inspecting the reporter’s account for information that could reasonably identify the person behind it (Section 3). These reports are not necessarily “stolen”, e.g., the authors may report bugs via private channels or discuss bugs their tool could detect but did not. However, when these reports support bug-finding claims, not clearly distinguishing reports created by authors from those created by others makes the claimed number of new bugs difficult to interpret.

We found 16 such reports across four papers, corresponding to 15 non-author reporter accounts. Most papers have each NAR created by a different non-author account. One paper has a relatively large number of NAR reports and highlights why this practice is problematic even without intentionally misleading claims. The paper includes many more reports but never explicitly indicates which were created by the authors vs. others. Based on our procedure, we could establish only a subset of the reports to be created by the authors. Multiple careful readings of the paper (by at least five members of our team!) do not show that it necessarily misattributes these reports to the authors. The key challenge is lack of clear information to determine which bugs are newly reported by the authors vs. externally reported and used as supporting evidence. Similar cases may overstate the strength of a paper’s bug-finding results, or at minimum make those results harder to audit.

Final Emails. Some authors provided verifiable evidence for apparent NAR reports, e.g., using multiple accounts (up to five although commonly just two) or reporting bugs privately (e.g., sharing a Discord screenshot with us). Others claimed to have reported bugs privately but could not provide any evidence, e.g., “*I have thoroughly searched my inbox, but unfortunately, I am unable to locate the specific email trail*”. Yet others claimed independently finding previously reported bugs and admitted listing the previous reports, e.g., “*we noticed that part of the issue had been reported previously, so we [listed] the related issue as a reference*” or “*we were not experienced in [reporting bugs]*”.

CPF: Reports counted as new bugs although they had been fixed. In six papers, authors appear to count a report as a newly found bug, although the tracker evidence indicates that the same bug had been already fixed before the authors’ report. The problem is not just that the authors did not check for the fix before reporting the behavior, but that such reports should not count the same as a newly reported bug accepted by developers. As usual, we treat this case leniently and only label it as a potential bad practice if the evidence is overwhelming. We distinguish three scenarios: (1) the bug had been fixed—with its patch merged—before the authors created their report, (2) a pull request with a fix was already open but not yet merged, and (3) the bug had previously been reported but not fixed. We count only the first scenario as CPF.

For example, one paper counts as new bugs two reports for previously fixed bugs. One of those two is, in fact, also an instance of NAR. (Only 5 reports have two bad practices each.) The other is a more direct instance of CPF: the developers comment on the authors’ issue that the bug does not appear in the latest version, and the author does not follow up. Another paper has two reports where the developers suggests running on the latest version, and the author confirms that the bug no longer manifests on the latest version.

Final Emails. Determining whether papers claim that their reports are for newly found, previously unreported bugs can be difficult, so we had to seek clarification via email. Some authors state all bugs as new. Others claim only that their tools find bugs, not distinguishing new and previously known: “*We did not mean to suggest that these should be counted as new, previously unknown bugs, or as bugs fixed in response to our reports*”, “*We do not claim this bug is new or it can only be found by (OurTool)*”, or “*We did not intend to imply that every report[...] necessarily corresponded to a previously unknown [bug]*”. Yet others admit mistaken count and promise to do better: “*We will also check candidate bugs against the latest available version before reporting them going forward*”.

CDU: Duplicate reports counted as unique bugs. Counting unique bugs precisely is notoriously hard [23, 33, 45], e.g., if fuzzing produces two inputs that crash an application with similar but not same stack traces, are those two different bugs or two manifestations of the same bug [36]? However, when developers explicitly mark a report as duplicate, the determination is much clearer, although the authors could still argue in the paper why they disagree with that judgment [22], e.g., a duplicate report may provide additional reproduction steps or help developers diagnose the problem. A potential bad practice is counting all duplicate reports as unique *new* bugs, unless the paper explains why the duplicate judgment is wrong.

We found several author-created duplicate reports that were either undisclosed or counted with a status different than the developers'. For one paper, we found relatively many such reports: a few listed in the paper and even more not listed but found from the authors' activity history. The developers marked all as duplicates of other reports also created by the authors (mostly earlier but some later). Another paper lists two duplicate reports in the artifact but marks both as Confirmed; the authors' activity history also contains more duplicate reports not disclosed in the paper.

For a third paper, some duplicates are of reports that eventually get fixed: should the later duplicates inherit the fixed status of the original report or be treated as rejected because the new report itself was marked as duplicate? We leniently excluded this paper from CDU.

On the positive, one paper explicitly lists a duplicate report as rejected. This accounting is precisely the kind we would expect—the duplicate is disclosed and not counted as a unique bug!

Final Emails. Most responses for CDU first argued that the status of a report can change over time. One set of authors claimed that their reports “*were classified as confirmed when we finalized and submitted the paper [but] later marked as duplicates*”. However, after we pointed out that the public evidence shows the reports marked as duplicates very quickly after their creation, with no intermediate confirmation by developers, the authors acknowledged that “*the current bug list does not clearly distinguish between newly discovered bugs, duplicate reports that reproduce existing bugs, and the status of the corresponding underlying [bugs]*” and promised that they “*will update the [...] bug list to make these distinctions explicit and correct the statuses*”. Another set of authors initially argued “*we reported [statuses] based on [our understanding] available at that time*” but later agreed with us: “*More precise and transparent labeling of bug reports in software-testing papers is important, and we will pay closer attention to this issue in our future research*”.

RSM: Report status claimed by the authors differs from the status we determine. This category includes papers that count a more favorable report status than the evidence in the corresponding bug tracker by the CRD. We leniently consider only the *worst* mismatch: reports counted as fixed although developers explicitly rejected them. Unlike more ambiguous cases, a fixed report should have concrete developer actions, e.g., a code change. For instance, one paper counts multiple reports as Fixed, although by the CRD most were marked Won't fix, Duplicate, or Can't reproduce, with the remaining confirmed but not fixed.

We also observed a less severe pattern: several papers label confirmed-but-not-yet-fixed reports as Fixed, even though they define intermediate categories between Confirmed and Fixed for precisely such cases. These reports still appear to represent genuine bugs, but labeling them as Fixed without evidence of a completed fix obstructs independent validation of bug-finding claims.

Final Emails. The responses for these cases varied. Some authors argued that their meaning of “fixed” is rather broad and not necessarily that the developers changed code: “*We used [...] 'fixed' as a very broad umbrella term to indicate [...] validated or acknowledged by the developers*”. Others appear to label previously known rediscovered bugs as “fixed” although they admit over email that the accurate resolution should be Duplicate: “*yes, [...] best interpreted as a duplicate of a previously fixed [bug], although [it] was present in [our] analyzed version*” and “[testing] on newer versions

showed [...] failure no longer occurred[, thus status] should not be ‘fixed’ without explaining [it is due to] our retesting rather than developer confirmation or a linked fix”.

4.3 Program Analysis Kind

We evaluate whether the program analysis kind used in the paper correlates with some measures of report quality. In terms of analysis technique, five out of 18 papers for which we have some reports use static analysis, while the rest implement either dynamic or a hybrid, static-dynamic analysis. Also, ten out of 18 papers rely on AI for parts of their bug-finding analysis, and AI is used more frequently with static analysis techniques. Interestingly, we do not observe any clear association between the technique used, AI involvement, and the presence of bad practices. One might expect that bugs found by static analysis or AI techniques, and without a concrete input/PoC, may get more pushback from developers, but our Bugeoisie does not show it. Anecdotally, however, for some AI-assisted reporting efforts, or reports submitted anonymously due to the double-blind reviewing process, developers raise concerns about hindered constructive discussion.

4.4 Community Recognition

We inspect the recognition that the 23 papers received from our community; six (26%) received an artifact evaluation badge, slightly higher than the overall ISSTA’25 ratio (21% of 107 papers received a badge). Notably, four exemplary papers from our study did not receive many badges, while some papers with inconsistent bug accounting or low report recoverability did receive badges. Moreover, one paper for which we could not initially collect any report received a Distinguished Paper Award. We should *not* overgeneralize from our small sample of papers from just one conference instance, but the patterns suggest that our community’s current recognitions (i.e., badges and awards) do not reliably reflect the traceability of bug evidence needed for independent verification.

We also relate the percentage of Fixed reports with papers that were awarded badges. We omit detailed results for anonymity but find that papers with at least one awarded badge, on average, have a larger percentage of Fixed than those that did not undergo artifact evaluation.

5 Bug Reports Analysis

Going beyond potential bad practices and aggregate paper analysis, we perform some report analysis to try to identify potential patterns in our Bugeoisie.

5.1 Timeline Analysis

We analyze the timeline of reports in our Bugeoisie: when they are created, how their status changes, and when they get closed, similar to prior work that considered how general reports, not only created by researchers, change status [29]. We focus on the reports from GitHub (both issues and PRs) because we can use APIs to automatically obtain information about *all* such reports. The average number of bugs per report is similar for reports only from GitHub and reports across all platforms. Most analyses have the qualitative conclusions consistent whether computed per report or per bug, e.g., Figure 2 shows the ratio of fixed reports is 42%, while the ratio of fixed bugs is 43%. Due to space limits, we show only report-level results.

For GitHub, we classify each report as *open* or *closed* based on its tracker status. We call a report *closed* if developers have taken an explicit closing action, either by *fixing* it (e.g., merging a PR or fixing an issue) or by *rejecting* it (e.g., closing a PR without merging or closing an issue without a fix). We call a report *open* if it has not been closed. Most papers have a mix of closed and open reports, while only one had all its reports closed.

Figure 2 shows the timing of report creation and status changes relative to the important conference dates. The left-side plot shows the cumulative distribution of created reports over time,

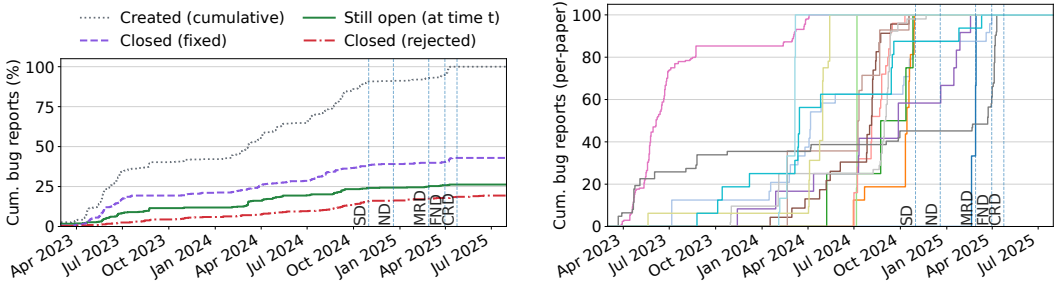


Fig. 2. Cumulative evolution of reports over time for 22 papers (one omitted for a very long timeline). Left: overall distribution aggregated over all papers, showing the cumulative fraction of reports created, still open at time t , and closed (Fixed vs. Rejected). Right: cumulative percentage of created reports per paper, where each curve corresponds to a paper (PIDs omitted). Vertical dashed lines indicate major conference deadlines: SD = submission, ND = notification, MRD = major revision, FND = final notification, CRD = camera ready.

where the x -axis represents time (with the key ISSTA'25 deadlines marked with dotted lines), and the y -axis represents the fraction of all reports created/Fixed/Rejected. The right-side plot shows the fraction per paper. Most paper authors create reports in bursts over a relatively short window (up to a few days), while three sets of authors created reports more gradually. Almost all authors create their reports before the paper submission deadline, but two set of authors created reports even after that deadline. Notably, one set of authors created a report even after the ISSTA'25 CRD (2025-04-24). We do not know if some of these papers were resubmitted from prior conferences, and if the reports may have been created at specific important dates for those conferences.

We also analyze how many report resolutions changed between the ISSTA'25 CRD and one year later (2026-04-24). We find that only 42 reports changed status between 2025 and 2026 but these come from a variety of 12 papers, with one paper having the most changes, 12. Most of these changes were to Fixed (18 reports) but also many to Inactive (12 reports) typically closed by bots.

5.2 Security-Related Bug Reports

Of the 23 papers we study, 13 include security-related bugs. 7 of 13 papers claim some vulnerability identifiers (for short IDs) such as CVE or CNVD numbers, ranging from 2 to 119 IDs per paper, for a total of 158 IDs we obtained. Of these 7 papers, 5 disclose all IDs, 1 discloses only 1 of 3 IDs, and 1 discloses no IDs. The value of CVEs is controversial [25, 37, 39, 46], and many researchers do not believe that bugs assigned CVEs are more important than others, or that the number of CVEs is a good measure of real-world impact. For example, Schloegel et al. [36] presented a case study in which six CVEs referred to a single bug, which was further never acknowledged by the developer.

Even assigned IDs do not always offer traceability to reports. Of the 158 IDs, we found 19 (12%) reserved on the NVD [10] or CNVD [14] but never fully published (lacking a description and reference links to aid traceability) and 10 more published but still lacking a report URL.

Even when IDs lead to reports, the status varies. For 129 IDs, we located 134 linked reports (3 IDs have 2 reports each, and 1 ID has 3 reports), and the corresponding reports show that developers fixed only 62 (46%), confirmed but not yet fixed 6 (4%), rejected 4 (3%), and 62 (46%) are still pending. For example, one report with an ID was marked *Invalid* by the developer.

6 Lessons Learned

I#1: Bug evidence is often poorly preserved, hindering analysis. For many bug-finding papers, the reports submitted to real projects are among their most direct external outcomes.

These reports matter for practice because they help developers improve software quality, and for research because they provide a verifiable evidence trail. Yet, reports are often treated as secondary by-products. Papers summarize top-level numbers, e.g., “X bugs found, Y reported, Z fixed”, while the underlying reports receive far less care than the paper and its artifacts, or are not preserved at all. This lack of traceability weakens bug claims and increases the cost of follow-up work.

L#2: Releasing report details makes bug-finding claims easier to verify. A recurring obstacle in our study is the lack of a stable, reader-accessible link between bug claims and their supporting tracker entries. References are often scattered across paper sections, appendices, supplemental material, or tool repositories, and sometimes omitted entirely. Thus, even a motivated reader may be unable to determine which reports correspond to which claimed bugs without substantial “report archaeology”, while such reconstruction can still be partial or uncertain.

L#3: Evidence released by the authors often under-represents rejected reports. Rejected reports are not uncommon, 16% of disclosed reports across papers and 53% of undisclosed reports. developers may reject reports, mark them as duplicates, reclassify them as enhancements, or deem them out-of-scope. When papers emphasize only fixed bugs, the rejected “failed outcomes” become obscured to the reader, although they can reveal tool limitations, reporting mismatches, and developer priorities. Omissions may reduce comparability across papers and limit what follow-up work can learn about why some researcher-reported bugs become actionable while others do not; as a result, the follow-up work may repeat the same pitfalls without knowing what was rejected.

L#4: Unclear bug counting rules make bug-finding claims difficult to interpret. Authors should preserve (in the paper or artifact), a minimal “bug accounting record” that includes each report URL or stable ID, the bugs it contains, their outcomes, and related fix commits or pull requests when applicable. They should also record why reports were closed without a fix and explain differences between their interpretation and the developer resolution. Because outcomes change, authors should state the claim cutoff date and, where feasible, preserve enough tracker metadata or archived content to reconstruct statuses at that date.

L#5: Listed report statuses are sometimes unclear and defined inconsistently. Across papers, bug claims are often difficult to interpret because the terminology is inconsistent. In particular, authors do not always clarify how they handle false positives and duplicates, whether “fixed” is in response to their report or not, or whether “fixed” even means that the underlying bug was corrected. Some papers include in their counts items that are *not* new bugs (e.g., false positives, previously discovered bugs, synthetic examples), thus phrases such as a “reported X new bugs” become ambiguous. While mixing newly discovered with previously known bugs may be sometimes scientifically justified, the writing often confuses readers when such cases are aggregated under summary statements like “new bugs found and reported” without further clarification. One final email even mentioned a criterion for new bugs being “*none of the compared tools found this bug*”.

L#6: Reconstructing report evidence via email is costly and frustrating. Overall, our email experience was rather tortuous for us and, presumably, for the authors. Most authors replied in detail to our final emails, answering (almost) all of our questions. Our key takeaway is that most such emails should be unnecessary: the same way that paper PDFs share most information about work, and that the artifacts provide a lot of code, scripts, and data, *those PDFs and artifacts should also provide traceable information for all claims made about bug counts and reports*. Only rarely should a reader need to ask authors for a clarification. We hope that our paper motivates future authors to spend time and energy in preparing artifacts rather than writing clarification emails!

7 Recommendations

R#1: Reviewers should check consistency between quantitative bug claims, report statuses, and author terminology. Conference committees in general, and Artifact Evaluation (AE)

Committees in particular, should require detailed reporting evidence whenever a paper makes quantitative claims about bugs and reports, e.g., “found X, reported Y, fixed Z”. Paper reviewers should check that the claimed totals, terminology, denominators, and supporting evidence are consistent. AE reviewers should treat the report evidence as part of the artifact and (1) check that report identifiers (IDs) are provided in a canonical format and that the evidence is included in the artifact, (2) confirm that URLs resolve or that durable tracker IDs are provided; and (3) verify that a sample of entries match the stated status categories as of a declared snapshot date and, when applicable, include sufficient details to reproduce the reported bugs. These steps would make claims reviewable with reasonable effort, similar to existing AE practices [3, 17].

R#2: Committees should encourage public evaluation whenever possible and a clear disclosure process for non-public reports. Conference committees should consider disclosure-aware placeholders for private reports, encourage authors to update them after public disclosure, and preserve verifiability as much as possible when details become public and updated later. For reports involving security or proprietary concerns, reviewers may need confidential access to supporting evidence. The community should encourage authors to also evaluate a subset of their claims on open-source repositories, providing public evidence of the technique’s utility and generalizability even when confidential bug counts and outcomes cannot be verified [26, 30].

R#3: Authors should list complete bug counts (including rejected), disclose all created reports, and clearly define the bug status terms used. Authors should disclose both the number of reports they created and the number of bugs in those reports. They should explain how they count reports containing multiple bugs or multiple reports about the same bug, and distinguish new bugs from previously known ones. Authors should also report the full range of outcomes, including rejected along with the reasons for rejection. Of the 23 papers, only four disclosed some rejected reports, ranging from 4% to 25% of the reports claimed as created. These cases show that transparent reporting of negative outcomes is compatible with publication at a premier venue.

R#4: Authors should preserve a dated report evidence trail supporting each quantitative bug claim. The paper PDF or artifact should present a minimal “bug accounting record” that makes the bug claims more easily verifiable. In the limit, this could be a table with `report_url`, `status`, `number_of_bugs` for each report, and notes on whether those are new or previously known (R#3). The record could also include other related reports when applicable, e.g., fix commits, pull requests, or patch locations. Authors should state the cutoff date for their bug claims and, where feasible, preserve tracker metadata or archived pages needed to reconstruct the corresponding record entry.

R#5: The community should use uniform status categories and counting rules so readers can interpret and compare bug claims consistently. The community should make an effort to standardize the terminology and counting rules used for bug claims. Papers should map their results to a small set of disjoint, consistently named categories, at minimum distinguishing between Fixed, Confirmed, Pending, and Rejected, as in our coarse-grained classification (Section 3.7). Rejected reports can be further divided into Duplicate, Can’t reproduce, Invalid, Inactive, and Won’t fix when greater detail is useful. Authors should also explicitly state inclusion/exclusion rules for rejected, duplicate, and previously known bugs, applying them consistently across the paper PDF and artifact so that readers can validate counts and easily interpret report outcomes.

R#6: The community may want to facilitate the preregistration of report evidence used for paper claims. The community should consider preregistration [18] of report evidence at a cutoff date or specific code version. Before evaluating their tools, authors would record the cutoff date and version, what will be counted and how (e.g., bug “units”, inclusion/exclusion rules, and treatment of different labeling by developer), the bug trackers to be used, and the report identifiers or other metadata in support of future validation. The preregistration should occur well before the paper deadline so readers can distinguish the original evidence from later tracker changes.

8 Threats to Validity

Our study, as every empirical study, has certain threats to validity. Our key claims are related to the *existence* of potential bad practices in some papers that make claims about bug reports.

The main *internal* threat is that we may have methodological flaws in determining these bad practices. We mitigate some of this threat by having multiple annotators for various items, extensive discussion in cases of disagreement, and generally lenient labeling of cases. When we claim that a paper has a bad practice for some report, we are fairly confident about *that* specific report. However, we may have collected wrong reports, or the authors did not convince us to remove some reports. One key problem is that collecting report URLs should not be as hard as it currently is.

The main *external* threat is that our results may not generalize beyond the ISSTA'25 Bugeoisie. We do not expect ISSTA papers to be particularly worse (or better) than papers in any similar conference. To reduce this threat, we have performed an initial analysis of the FSE'25 papers. We selected FSE'25 because it was colocated with ISSTA'25. FSE'25 main research track had 135 papers (vs. 107 for ISSTA'25). We checked all these papers for the key bad practice (MRU): are report URLs easy to find? We found 22 papers that claim some reports. The ratio $22/135=16.3\%$ is lower than for ISSTA'25 ($23/107=21.5\%$) but likely expected as FSE broadly covers all aspects of software engineering not just testing and analysis. We could easily find (891 of claimed 1,043) report URLs for only 13 of those 22 papers. The ratio $13/22=59.1\%$ is even lower than we had for ISSTA'25 ($15/23=65.2\%$) *before* emailing the authors. These results show that the problem of report URLs is not specific to ISSTA but more widespread in our Software Engineering community.

The main *construct* validity is that measuring quality and quantity of reports and bugs may not provide a good judgment about the scientific contributions in the paper. We do *not* claim that papers with bad practices about reports have worse techniques or tools. Indeed, simply counting bugs in not a perfect metric for evaluating contributions [37]. Some bugs are easier to fix than others, some are more important than others, and some may be of a novel kind not familiar to the developers. Authors can discuss these topics in their papers, e.g., we should expect a lower acceptance rate (and a higher rejection rate) for reports for a novel kind of bugs. It is similar how paper reviewing accepts smaller evaluations for more novel ideas but requires bigger evaluations for less novel ideas. The basic change we argue for is to treat reports as first-class artifacts.

9 Related Work

There is a large body of research studying bug reports in general, but to the best of our knowledge, our study is the first systematically focused on a set of reports created by researchers (Bugeoisie). **Report management.** Multiple research projects have studied how bug trackers are used and how reports evolve over time. Bissyandé et al. [8] characterized GitHub issue tracker adoption and usage at scale, including who reports issues and how tracker activity correlates with popularity proxies such as number of forks. Kikas et al. [29] examined issue dynamics over time, showing that issue arrival is higher early in a project's life. Issues not closed during the study period remain in the pending set and accumulate over time, increasing the open-issue inventory even when closure rates for resolved issues remain relatively stable. Tan et al. [41] studied GitHub's good first issue tag at scale and showed that its onboarding effectiveness depends on report actionability. Zhou et al. [49] analyze issues in open-source AI repos, proposing a taxonomy of issue types and highlighting how frequently issues stem from runtime errors and unclear instructions. Curtis et al. [16] studied the use of the `wontfix` label on GitHub, investigating why issues are rejected and quantifying how `wontfix` co-occurs with other labels. Outside GitHub, Wang et al. [44] used comparative case studies (Mozilla vs. Python) to connect tooling and process to peer review and triage practices.

Report quality, information need, and reproducibility. Another research direction focuses on the quality of reports and what information helps developers to fix issues. Bettenburg et al. [6, 7] studied what makes a good report and found a mismatch: developers value information like steps to reproduce, stack traces, and tests, but reporters often fail to provide it. Breu et al. [11] studied the questions developers ask during bug resolution, proposing a taxonomy of information requests that shows how much bug fixing depends on iterative clarification between developers and reporters. Guo et al. [21] investigated which reports get fixed in Microsoft Windows, showing that fix likelihood is associated not only with report attributes (e.g., severity and signals of ownership uncertainty) but also with factors such as reporter/assignee reputation and organizational proximity.

More targeted work aims to address missing report content automatically. For example, Chaparro et al. [13] showed that observed behavior is common in reports, but expected behavior and steps-to-reproduce are not. Their follow-up work [12] studied the quality of reproducibility steps for mobile apps. Johnson et al. [28] manually reproduced a large set of Android GitHub issues to characterize what reproduction actually requires.

GitHub templates are a practical attempt to request structured information at report creation time. Recent studies suggest that templates can improve understanding, reproducing, and fixing reported bugs. Sulun et al. [40] analyzed GitHub issue templates and found them associated with reduced resolution time and fewer reopenings. Li et al. [31] offer a complementary “first look” at report templates on GitHub, showing that templated reports tend to resolve faster. They also propose an automated “templativization” approach to help retrofit unstructured reports.

Structured bug reporting using AI. Recent work started using LLMs to transform unstructured reports into high-quality, more structured reports. Acharya et al. [1] use fine-tuning of open-source LLMs to convert casual, unstructured reports into templated ones. Bo et al. [9] propose ChatBR, which first uses a fine-tuned classifier to detect whether key report elements are missing, and then prompts ChatGPT to fill in missing fields. Feng et al. [19] introduce a prompting approach that extracts reproducing step information from reports to guide GUI testing for mobile apps. Song et al. [38] propose an interactive NLP reporting chatbot that guides reporters and offers immediate feedback by matching descriptions against a GUI-level execution model.

10 Conclusion

We study Bugeoisie, a set of reports created by researchers, as a first-class artifact at the boundary of empirical research evaluation and real-world software maintenance. We build a dataset from ISSTA’25 papers and measure how well claims about reported bugs can be traced to tracker evidence. We also identify and systematically study six potential bad practices. Finally, we provide practical guidance for our community to make bug-related claims easier to check and compare.

Acknowledgments

We thank Wentao Zhang for inspecting reports not in English; Pierre Francis, Mahadi Hasan, Muhammad Hassan, Anika Islam, Harsha Malaviya, and Shrihan Tummala for inspecting some FSE’25 papers; and the anonymous reviewers for nudging us to quantify potential bad practices.

Data Availability

We do not make our current data publicly available due to (1) potential reputation harm to the authors of the papers we studied and (2) receiving some reports via email without asking the authors for permission to share. We recognize the seeming irony that our paper, which reproduces some results, does not itself release the data to reproduce our own results. After more extensive checks for anonymization, we *may* release some data at <https://github.com/Bugeoisie/Bugeoisie>. Researchers interested in our data can contact us via email to discuss what we could share.

References

- [1] Jagrit Acharya and Gouri Ginde. 2025. Can We Enhance Bug Report Quality Using LLMs?: An Empirical Study of LLM-Based Bug Report Generation. In *EASE*. doi:10.1145/3756681.3756995
- [2] ACM Emerging Interest Group on Reproducibility and Replicability. 2023. ACM EIG on Reproducibility and Replicability. Accessed: 2026-07-23. <https://reproducibility.acm.org/>
- [3] ACM Publications Board. 2020. Artifact Review and Badging – Current. <https://www.acm.org/publications/policies/artifact-review-and-badging-current>. Accessed: 2026-07-23.
- [4] Internet Archive. 2026. *Wayback Machine*. Accessed: 2026-07-23. <https://archive.org>
- [5] Association for Computing Machinery. 2020. Artifact Review and Badging – Definitions. Accessed: 2026-07-23. <https://www.acm.org/publications/badging-terms>
- [6] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What Makes a Good Bug Report?. In *FSE*. doi:10.1145/1453101.1453146
- [7] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. 2008. Extracting Structural Information from Bug Reports. In *MSR*. doi:10.1145/1370750.1370757
- [8] Tegawendé F. Bissyandé, David Lo, Lingxiao Jiang, Laurent Réveillère, Jacques Klein, and Yves Le Traon. 2013. Got Issues? Who Cares About It? A Large Scale Investigation of Issue Trackers from GitHub. In *ISSRE*. doi:10.1109/ISSRE.2013.6698918
- [9] Lili Bo, Wangjie Ji, Xiaobing Sun, Ting Zhang, Xiaoxue Wu, and Ying Wei. 2024. ChatBR: Automated Assessment and Improvement of Bug Report Quality Using ChatGPT. In *ASE*. doi:10.1145/3691620.3695518
- [10] Harold Booth, Doug Rike, and Gregory Witte. 2013. The National Vulnerability Database (NVD): Overview. Accessed: 2026-07-23. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=915172
- [11] Silvia Breu, Rahul Premraj, Jonathan Sillito, and Thomas Zimmermann. 2010. Information Needs in Bug Reports: Improving Cooperation Between Developers and Users. In *CSCW*. doi:10.1145/1718918.1718973
- [12] Oscar Chaparro, Carlos Bernal-Cárdenas, Jing Lu, Kevin Moran, Andrian Marcus, Massimiliano Di Penta, Denys Poshyvanyk, and Vincent Ng. 2019. Assessing the Quality of the Steps to Reproduce in Bug Reports. In *ESEC/FSE*. doi:10.1145/3338906.3338947
- [13] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. 2017. Detecting Missing Information in Bug Descriptions. In *ESEC/FSE*. doi:10.1145/3106237.3106285
- [14] China National Vulnerability Database. 2009. *CNVD Vulnerability Detail Site*. Accessed: 2026-07-23. <https://www.cnvd.org.cn>
- [15] Christian Collberg and Todd A. Proebsting. 2016. Repeatability in Computer Systems Research. *CACM* (2016). doi:10.1145/2812803
- [16] J. Alexander Curtis, Sharadha Kasiviswanathan, and Nasir Eisty. 2025. Deciphering WONTFIX: A Mixed-Method Study on Why GitHub Issues Get Rejected. <https://arxiv.org/abs/2510.01514>
- [17] Daniele Cono D'Elia, Thaleia Dimitra Doudali, Cristiano Giuffrida, Miguel Matos, Mathias Payer, Solal Pirelli, Georgios Portokalidis, Valerio Schiavoni, Salvatore Signorello, and Anjo Vahldiek-Oberwagner. 2025. Lessons Learned from Five Years of Artifact Evaluations at EuroSys. In *REP*. doi:10.1145/3736731.3746152
- [18] Neil A. Ernst and Maria Teresa Baldassarre. 2023. Registered Reports in Software Engineering. *Empirical Softw. Engg.* (2023). doi:10.1007/s10664-022-10277-5
- [19] Sidong Feng and Chunyang Chen. 2024. Prompting Is All You Need: Automated Android Bug Replay with Large Language Models. In *ICSE*. doi:10.1145/3597503.3608137
- [20] Google Project Zero. 2014. *Project Zero*. Accessed: 2026-07-23. <https://projectzero.google>
- [21] Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. 2010. Characterizing and Predicting Which Bugs Get Fixed: An Empirical Study of Microsoft Windows. In *ICSE*. doi:10.1145/1806799.1806871
- [22] Kim Herzig, Sascha Just, and Andreas Zeller. 2013. It's Not a Bug, It's a Feature: How Misclassification Impacts Bug Prediction. In *ICSE*. doi:10.1109/ICSE.2013.6606585
- [23] Michael Hicks. 2015. *What is a bug?* Accessed: 2026-07-23. <http://www.pl-enthusiast.net/2015/09/08/what-is-a-bug/>
- [24] Allen Householder, Garret Wassermann, Arthur Manion, and Christopher King. 2017. *CERT Guide to Coordinated Vulnerability Disclosure*. Technical Report. Carnegie Mellon University. doi:10.1184/R1/12367340.v1
- [25] Aram Hovsepian. 2025. *The Realities of AppSec Risk Management using CVE and CVSS*. Accessed: 2026-07-23.
- [26] Lexiang Huang, Matthew Magnusson, Abishek Bangalore Muralikrishna, Salman Estyak, Rebecca Isaacs, Abutalib Aghayev, Timothy Zhu, and Aleksey Charapko. 2022. Metastable Failures in the Wild. In *OSDI*.
- [27] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *ICLR*.
- [28] Jack Johnson, Junayed Mahmud, Tyler Wendland, Kevin Moran, Julia Rubin, and Mattia Fazzini. 2022. An Empirical Investigation into the Reproduction of Bug Reports for Android Apps. In *SANER*. doi:10.1109/saner53432.2022.00048

- [29] Riivo Kikas, Marlon Dumas, and Dietmar Pfahl. 2015. Issue Dynamics in Github Projects. In *PROFES*. doi:10.1007/978-3-319-26844-6_22
- [30] Guangpu Li, Shan Lu, Madanlal Musuvathi, Suman Nath, and Rohan Padhye. 2019. Efficient Scalable Thread-Safety-Violation Detection: Finding thousands of concurrency bugs during testing. In *SOSP*. doi:10.1145/3341301.3359638
- [31] Hongyan Li, Meng Yan, Weifeng Sun, Xiao Liu, and Yunsong Wu. 2023. A first look at bug report templates on GitHub. *JSS* (2023). doi:10.1016/j.jss.2023.111709
- [32] LibPlacebo. 2024. *Issue #326*. Accessed: 2026-07-23. <https://code.videolan.org/videolan/libplacebo/-/issues/326>
- [33] Dongliang Mu, Yuhang Wu, Yueqi Chen, Zhenpeng Lin, Chensheng Yu, Xinyu Xing, and Gang Wang. 2022. An In-Depth Analysis of Duplicated Linux Kernel Bug Reports. In *NDSS*. doi:10.14722/ndss.2022.24159
- [34] International Workshop on Robotics Software Engineering. 2025. Accessed: 2026-07-23. <https://conf.researchr.org/series/rose>
- [35] Leon J. Osterweil, Carlo Ghezzi, Jeff Kramer, and Alexander L. Wolf. 2008. Determining the Impact of Software Engineering Research on Practice. *Computer* (2008). doi:10.1109/MC.2008.85
- [36] Moritz Schloegel, Nils Bars, Nico Schiller, Lukas Bernhard, Tobias Scharnowski, Addison Crump, Arash Ale-Ebrahim, Nicolai Bissantz, Marius Muench, and Thorsten Holz. 2024. SoK: Prudent Evaluation Practices for Fuzzing. In *IEEE S&P*. doi:10.1109/SP54263.2024.00137
- [37] Moritz Schloegel, Daniel Klischies, Simon Koch, David Klein, Lukas Gerlach, Malte Wessels, Leon Trampert, Martin Johns, Mathy Vanhoef, Michael Schwarz, Thorsten Holz, and Jo Van Bulck. 2025. Confusing Value with Enumeration: Studying the Use of CVEs in Academia. In *USENIX Security*.
- [38] Yang Song, Junayed Mahmud, Nadeeshan De Silva, Ying Zhou, Oscar Chaparro, Kevin Moran, Andrian Marcus, and Denys Poshyvanyk. 2023. BURT: A Chatbot for Interactive Bug Reporting. In *ICSE Demo*. doi:10.1109/ICSE-Companion58688.2023.00048
- [39] Daniel Stenberg. 2023. *CVE-2020-19909 is everything that is wrong with CVEs*. Accessed: 2026-07-23. <https://daniel.haxx.se/blog/2023/08/26/cve-2020-19909-is-everything-that-is-wrong-with-cves/>
- [40] Emre Sülün, Metehan Saçakçı, and Eray Tüzün. 2024. An Empirical Analysis of Issue Templates Usage in Large-Scale Projects on GitHub. *TOSEM*. doi:10.1145/3643673
- [41] Xin Tan, Minghui Zhou, and Zeyu Sun. 2020. A First Look at Good First Issues on GitHub. In *ESEC/FSE*. doi:10.1145/3368089.3409746
- [42] Umka-lang. 2023. *PR #285*. Accessed: 2026-07-23. <https://github.com/vtereshkov/umka-lang/pull/285>
- [43] Umka-lang. 2024. *Issue #461*. Accessed: 2026-07-23. <https://github.com/vtereshkov/umka-lang/issues/461>
- [44] Jing Wang, Patrick C. Shih, Yu Wu, and John M. Carroll. 2015. Comparative case studies of open source software peer review practices. *Information and Software Technology* (2015). doi:10.1016/j.infsof.2015.06.002
- [45] David Gray Widder and Claire Le Goues. 2024. What Is a 'Bug'? *CACM*. doi:10.1145/3662730
- [46] Julia Wunder, Andreas Kurtz, Christian Eichenmüller, Freya Gassmann, and Zinaida Benenson. 2024. Shedding Light on CVSS Scoring Inconsistencies: A User-Centric Study on Evaluating Widespread Security Vulnerabilities. In *IEEE S&P*. doi:10.1109/SP54263.2024.00058
- [47] Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. 2025. SWE-bench Goes Live!. In *NeurIPS*.
- [48] Hao Zhong. 2025. Understanding Compiler Bugs in Real Development. In *ICSE*. doi:10.1109/ICSE55347.2025.00068
- [49] Xiyu Zhou, Peng Liang, Beiqi Zhang, Zengyang Li, Aakash Ahmad, Mojtaba Shahin, and Muhammad Waseem. 2025. Exploring the Problems, their Causes and Solutions of AI Pair Programming: A Study on GitHub and Stack Overflow. *JSS* (2025). doi:10.1016/j.jss.2024.112204

Received 2026-01-30; accepted 2026-06-25