Reflections on Software Failure Analysis

Paschal C. Amusuo
Purdue University, USA
pamusuo@purdue.edu

Aishwarya Sharma
Purdue University, USA
sharm234@purdue.edu

Siddharth R. Rao
Purdue University, USA
rao147@purdue.edu

Abbey Vincent
Purdue University, USA
vincen17@purdue.edu

James C. Davis
Purdue University, USA
davisjam@purdue.edu

ABSTRACT
Failure studies are important in revealing the root causes, behaviors, and life cycle of defects in software systems. These studies either focus on understanding the characteristics of defects in specific classes of systems or the characteristics of a specific type of defect in the systems it manifests in. Failure studies have influenced various software engineering research directions, especially in the area of software evolution, defect detection, and program repair.

In this paper, we reflect on the conduct of failure studies in software engineering. We reviewed a sample of 52 failure study papers. We identified several recurring problems in these studies, some of which hinder the ability of the engineering community to trust or replicate the results. Based on our findings, we suggest future research directions, including identifying and analyzing failure causal chains, standardizing the conduct of failure studies, and tool support for faster defect analysis.

CCS CONCEPTS
• Software and its engineering → Software defect analysis.

KEYWORDS
Failure analysis, software defects, empirical software engineering

1 INTRODUCTION
The study of failures is integral to the success of engineered systems [27]. In software engineering, failure studies describe the characteristics of defects in software systems. These studies, otherwise known as bug studies, are either tailored toward understanding the characteristics of defects in specific classes of systems (e.g., web systems [5], Android apps [17], or embedded systems [19]) or the characteristics of specific classes of defects (e.g., performance [15], concurrency [7], or security [20]). These studies are designed to reveal the root causes of these defects, their manifestation, impact, fix characteristics, and life-cycle.

Over the last decade, the number of failure studies has steadily increased (Figure 1). These studies have influenced research into software testing [12], defect detection [6], and repair techniques [24].

In this paper, we reflect on the conduct of software failure analysis research over the last 20 years. Using a systematic literature review, we identified several flaws and challenges that affect this research direction. Following the flaws and challenges we identified, we discussed future research directions that the software engineering community can embark on, to aid the conduct of these failure studies. Our research directions are focused on attempting to answer various questions relevant to the efficient conduct and impact of failure studies.

2 IDEALIZED FAILURE STUDY MODEL
Failure studies are research focused on understanding the characteristics and causes of failures in engineered systems [16] [39]. In software engineering, these studies commonly consider defects.

This section presents an idealized model of the failure study process in software engineering. We derived this model by reviewing steps currently taken to conduct software failure studies, complemented with failure studies conducted in other engineering disciplines [8]. We used this model to analyze and review various failure studies reported in the software engineering literature.

Figure 2 shows the various stages of this idealized model, which is applied across engineering disciplines. First, the project scope is defined. This usually involves identifying what class of defects to
We computed the Cohen kappa score on this sample as 0.763, which shows substantial agreement [13]. Subsequently, the authors continued the data extraction independently while one more experienced author reviewed the data extracted by the other authors.

Threat to validity: We sampled only 52 failure studies, which may not have included all relevant failure studies. But we believe this sample is representative, and our findings are valid and relevant. The sample was selected through a methodological process, as discussed above. We also included recent papers published in prominent venues to ensure our findings were relevant to the current peer-reviewed conduct. Also, each of the flaws we identified was prevalent in over half of the sample of papers studied. Finally, while some of the flaws identified may seem obvious, we are the first to present empirical evidence of their existence while suggesting research directions to manage them.

### 3.2 Recurring Flaws

3.2.1 Bias towards Open-source Software: Investigators conducting failure studies are biased toward studying defects in open-source software (first row of Table 1). This is usually because open-source software has publicly available code, documentation, and complete evolution history. Unfortunately, focusing on only open-source software may be inconsistent with the investigator’s goal, ultimately aiding software engineering practice beyond open-source.

Prior research has investigated and reported differences between open-source and commercial software [22] [26] [3]. Mockus et al. [22] showed that the post-release defect density for Apache was significantly different compared to 4 commercial projects. Paulson et al. [26] reported that more defects are being found and fixed in open-source software, which may have contributed to the high defect density reported in [22]. Boulanger [3] identified differences between the software development practices for open-source and commercial software projects. In open-source software, defects are usually reported by customers, unlike in commercial software. This could also affect the kinds of defects analyzed by failure studies. As a result, the results from these failure studies that studied open-source software may not generalize to commercial environments.

3.2.2 Root Causes are Subjectively Identified: Root cause analysis is the most common aspect of defects considered by failure studies (Figure 3). However, only one paper [19] reported using a root cause analysis methodology to identify these root causes. According to Paradies et al. [25], root causes should be basic causes that are within

---

1Our final search query was “(empirical OR comprehensive OR taxonomy OR characteristics) AND (bug OR bugs OR faults OR defects OR failures OR vulnerabilities) AND (study OR review)”
the ambit of management to fix. Gangidi [9] also explained that a systematic root cause analysis methodology should reveal deeper systemic causes (e.g., policies, practices, management decisions).

The root causes identified by the failure studies we reviewed mostly represent technical flaws and do not correspond with any of these definitions. Wang [40] identified root causes such as misuse of mathematical formulas, inconsistency between hardware and software, and improper handling of parameters. While these are the immediate causes of the reported defects, they are neither ‘basic’ nor systemic. Deeper investigations into defects caused by hardware/software inconsistency may reveal underlying causes such as poor documentation, which may also have been attributed to the absence of documentation guidelines. As another example, Gunawi et al. [10] identified data races as one of the root causes of data inconsistency in cloud systems, but deeper analysis might have also revealed other underlying factors that led to these data races. If papers conducted a deeper root cause analysis, their results could be more helpful to practitioners and engineering teams.

3.2.3 Inconsistent Defect Taxonomies: Failure studies attempt to characterize the defects in software systems to aid their analysis. Our results, as shown in the second row of Table 1, show that most failure studies invent the taxonomies they use for this characterization, even when they study the same class of defects. For example, Cao et al. [4] characterized performance bugs in deep learning systems using a self-generated taxonomy but could have adapted taxonomies from prior research on performance bugs [18] [21] [41]. As a result, it becomes difficult to compare the distribution of performance defects in [4] and earlier works such as [21].

We also found disagreement in the interpretation of terms in the taxonomy when investigators choose to reuse taxonomies from earlier studies. For example, Tan et al. [38] reported they reused the taxonomy defined by Sullivan et al. [36] but acknowledged that the definition of semantic bugs between the two studies may be different, accounting for the huge discrepancy between the percentage of semantic bugs found by the two papers.

3.2.4 Non-integration of Practicing Software Engineers in the Study: Our review of failure study papers shows that practitioners are not included during the conduct of these studies. Investigating the perspectives of the software engineers who create or fix these defects can be helpful in providing insights into the causes and characteristics of these defects.

Furthermore, failure study papers are focused on enabling software engineering research but fail to make contributions that are relevant to software engineers. According to the fourth row of Table 1, only 27% of reviewed papers proposed recommendations pertinent to current software engineering practices. Mantyla [23] provided guidelines for conducting code and documentation reviews. Sun [37] made recommendations for generating test cases for compilers. Others only discussed the research implications of their work. This is contrary to failure studies in other disciplines whose results recommended changes in practitioners’ practices [8] [27] [31] [32]. With an increased focus on improving engineering practice, the results and recommendations from these studies could reduce the occurrence of defects, which would significantly increase software engineers’ productivity.

3.2.5 Defects in Embedded/IoT Systems are Understudied: From our results, we observed that the software engineering community is biased towards failure studies on web-based and desktop-based systems, while embedded/IoT systems are still understudied. As shown in Figure 4, embedded/IoT systems accounted for only two papers, while web-based systems (e.g., browsers) had 16 and desktop-based systems (e.g., compilers) had 12. Embedded systems power our airplanes, vehicles, and industries and deserve additional attention.

3.2.6 Miscellaneous Flaws: In addition to the primary flaws discussed above, we summarize three more issues.

Inconsistent quality measures: Defect analysis is subjective, and single-author investigation methods are untrustworthy. Of the 52 papers reviewed, only 19 studies had multiple authors independently analyze the data. Hence, the results of most studies are untrustworthy without the use of quality control measures.

Absence of replicability data: Only 11 papers included links to their replication package; 3 of these were inaccessible.

Table 1: Table showing further failure study analysis.

| Analysis                                              | Yes | No |
|-------------------------------------------------------|-----|----|
| Papers that studied defects in proprietary software   | 3   | 49 |
| Papers that reused taxonomies from literature         | 10  | 42 |
| Papers that reported the use of any tool              | 12  | 40 |
| Papers that made practitioner-relevant contributions  | 14  | 38 |

Figure 3: Research questions investigated by failure studies.

Figure 4: Distribution of failure studies by system type.
Missing tool support: Failure studies are time-consuming and lack tool support. Leesapatpornwongsa et al. [14] and Shen et al. [35] reported that it took them 15 and 24 months to conduct their study. Yet, according to the third row of Table 1, only 23% of failure studies reported using any tool in their study. These studies require investigators to analyze and categorize hundreds of defect reports manually. When studying a specific class of defects, these investigators rely on only keyword matching to filter prospective defect reports and need to go through each filtered report to identify and remove false positives. Mazuera-rozo et al. [21] identified 1,010 commits using keyword matching, and after manual analysis by two authors, only 20% (204 commits) were true positives.

4 A RESEARCH AGENDA

4.1 Defect Causal Chains

To effectively identify the root causes of defects, as discussed in §3.2.2, we suggest investigators use additional sources that provide more information about the causal chain of the defect. It is uncertain if analysis of pull request comments, meeting logs, design documents, or other artifacts will be helpful. Still, these documents can provide more insights into the reason behind the codes written by the developers. The research community can conduct further research to determine which artifacts would be more helpful and how investigators can adequately analyze them to identify the root causes of defects.

In addition, software engineers have no standard approach to documenting design or implementation decisions or efforts. While standards such as ISO/IEC/IEEE 12207 require detailed documentation by the software engineers, Agile methodologies [1] [2] recommend less comprehensive documentation. Hence, this presents another challenge as there is no guarantee that these documents will be available for analysis. The research results can also inform engineering teams what documentation needs to be maintained if they want to learn from their failures.

4.2 Standardizing the Conduct of Failure Studies

As we discussed in §3.2.3, there are inconsistencies in the conduct of failure studies. We suggest two ways to standardize the conduct of these studies. First, add a standard for failure analysis to the SIGSOFT empirical standards [28] to note the quality measures, replication packages, and expected general guidelines for conducting a failure study. Second, we suggest the development of a defect-type taxonomy map for software defects, similar to the Common Weakness Enumeration (CWE) used for categorizing security vulnerabilities. Such a map would contain a taxonomy of common defect types. It can be extensible that investigators conducting failure studies for a specific system or defect classes can build upon existing taxonomies with defect type categories particular to the class of system being investigated rather than inventing a new taxonomy. This map would ensure that the results of all failure studies are comparable, which will improve the generalizability of research influenced by the results.

4.3 Increased Impact on Engineering Practices

Following the bias reported in §3.2.1, we propose increased research emphasis on replicability studies to verify if failure studies conducted on open-source software also hold for commercial software. We also suggest increased collaboration between investigators of failure studies and software engineering companies, which would provide these investigators access to defect reports of commercial software. This collaboration would ensure that failure studies’ results influence research, which would also be relevant to practitioners in these companies.

We also recommend that, in addition to providing research directions, software failure studies provide recommendations to engineering teams that will reduce the occurrence of defects and the time to debug and fix reported defects. This is akin to failure analysis in other engineering disciplines, such as in the NTSB, where such studies have led to various changes in engineering, management, and regulatory practices [8].

4.4 Tool Support for Faster Defect Analysis

With the challenge of missing tool support discussed in §3.2.6, we recommend the research and development of tools that would aid the conduct of these studies. Natural Language Processing (NLP) techniques have become increasingly helpful in understanding the semantic meaning of documents, summarizing, and extracting useful information from documents. They have successfully been used to identify defects in requirement documents [33], identify duplicate defect reports [34], extract tasks and user stories from app store reviews [11], and summarize defect reports [30] [29]. Hence, the research community can easily explore the use of NLP to identify target defect reports, characterize the defects in them and extract other relevant information about the defect (e.g., consequence, manifestation behavior, component affected) from these reports. While using NLP can not replace the need for expertise-based human analysis, automating the above-listed tasks would significantly reduce the time the investigators spend conducting manual analysis.

5 CONCLUSION

In this paper, we reflect on the conduct of failure studies in software engineering by surveying 52 published failure study papers. We identified eight recurring flaws that have marred the conduct of failure studies. These flaws impede the correctness, reliability, and impact of the reported results of these studies.

Motivated by these challenges, we identify various ways the research community can support the conduct of these failure studies. We encourage further research on identifying and analyzing causal chains for defects and tool support to simplify defect analysis while recommending efforts to standardize the conduct of failure studies. With these steps, software failure studies may improve software engineering quality.

DATA AVAILABILITY

Our artifact can be found at https://doi.org/10.5281/zenodo.7041931. This spreadsheet contains our analysis of the failure study papers we surveyed.
[41] Shahed Zaman, Bram Adams, and Ahmed E. Hassan. 2012. A qualitative study on performance bugs. In 2012 9th IEEE Working Conference on Mining Software Repositories (MSR). 199–208. https://doi.org/10.1109/MSR.2012.6224281 ISSN: 2160-1860.