Reflections on Software Failure Analysis

Paschal 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

ACM Reference Format:
Paschal Amusuo, Aishwarya Sharma, Siddharth R. Rao, Abbey Vincent, and James C. Davis. 2022. Reflections on Software Failure Analysis. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3540250.3560879

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.

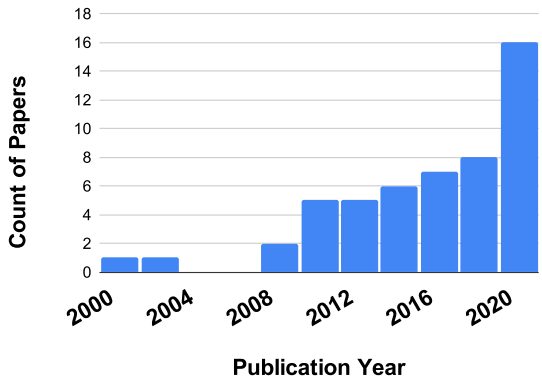

Figure 1: The distribution of failure studies by year published.

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 [24] techniques.

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.

In this section, we present an idealized model of the failure study process in software engineering. This model was derived from a review of the 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

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9413-0/22/11.
https://doi.org/10.1145/3540250.3560879

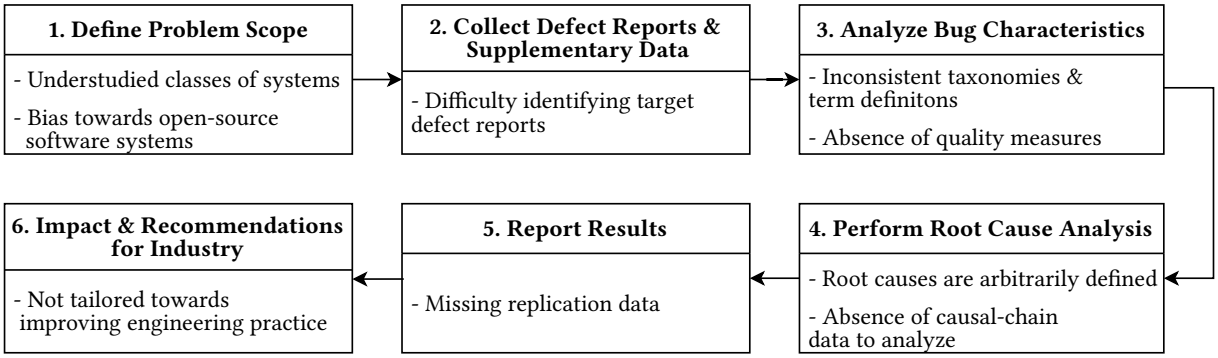

Figure 2: Idealized model of software engineering failure study that our study identified flaws in.

defined. This usually involves identifying what class of defects to study, the system to study, and how the target defects and system would be identified. Then the defect reports and other relevant data are collected and reviewed. The investigators use the information extracted to analyze the characteristics of the various defects, such as how they manifest, their impact, their life cycle, etc. In addition to this, the investigators can also perform a root cause analysis to determine the probable root cause and contributing causes of the defects. Once the study is completed, investigators report their results and discuss their implications. This report should also contain their analyzed data to aid replicability by other investigators. To ensure that practitioners learn from the results of the study, it behooves the investigators to provide recommendations to these practitioners while also working with them to validate the impact of their results and recommendations.

The figure also depicts common shortcomings of the existing studies in software engineering literature at various stages. We discuss these shortcomings in the next section.

3 FLAWS IN FAILURE STUDY METHODS

In this section, we present the flaws we identified in this research direction, as practiced in software engineering.

3.1 Methodology

We first searched the proceedings of prominent software engineering conferences (ICSE, ESEC/FSE, ASE) and journals (IEEE TSE, ESEM, JSS) and manually identified failure study papers. The results helped us define our search phrase.[1] We used this phrase to search scholarly databases (Google Scholar, IEEE Xplore, ACM Digital Library). This search yielded 92 candidate papers relevant to our study. Working in teams of 2, we manually reviewed the abstract of these papers, identified and selected 52 papers that studied and characterized defects in software, and were published in peer review venues.

We reviewed the selected papers, identified and collected data related to the various stages outlined in Figure 2. We analyzed the data extracted and identified the flaws discussed in the next subsection.

To ensure the quality of our results, we had multiple authors independently perform data extraction on a sample of 20 papers. 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, which is ultimately to aid 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] actually reported the use of a root cause analysis methodology to identify these root causes. According to Paradies *et al.* [25], these root causes should

[1] Our 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)"

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

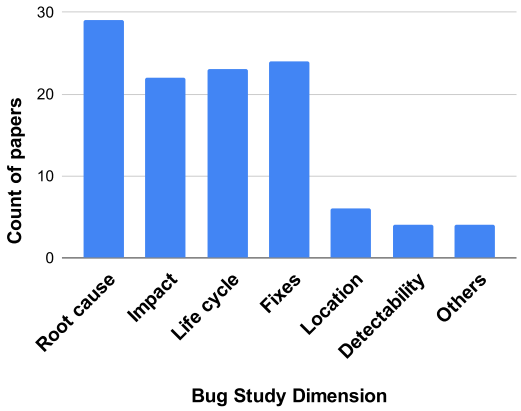

Figure 3: Research questions investigated by failure studies.

be basic causes that are within 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 any guidelines for preparing documentation. 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

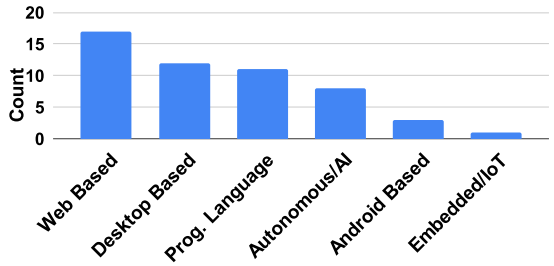

Figure 4: Distribution of failure studies by system type.

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.

Missing tool support: Failure studies are time-consuming and lack tool support. Leesatapornwongsa *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 the use of any tool in their study. These studies require investigators to manually analyze and categorize hundreds of defect reports. 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. In Mazuera-rozo *et al.* [21], 1,010 commits were identified using keyword matching, and after manual analysis by two authors, only 20% (204 commits) were true positives. In §4.4, we discuss ways tools and automation can be used to assist the conduct of failure studies.

4 A RESEARCH AGENDA

4.1 Defect Causal Chains

To effectively identify the root causes for defects, as discussed in §3.2.2, investigators need to conduct an appropriate root cause analysis. We suggest investigators also focus on other supplementary sources that can provide more information about the causal chain of the defect, in addition to the defect reports already being investigated. It is uncertain if analysis of pull request comments, meeting logs, design documents, or other artifacts will be helpful, but 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 they can be properly analyzed to identify the root causes of defects.

In addition to this, there is no standard approach to documenting design or implementation decisions or efforts by software engineers. While standards such as ISO/IEC/IEE 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 aforementioned research results can also inform engineering teams on 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 and can be extensible such that investigators conducting failure studies for a specific system or defect classes can build upon existing taxonomies with defect type categories specific to the class of system being investigated rather than inventing a new taxonomy. This map would ensure that the results of all failure studies being conducted 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 aimed at verifying if the results of failure studies conducted on open-source software also holds 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 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 also 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 and to simplify the conduct of failure analysis, we recommend the research and development of tools and automation 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 explore the use of NLP to identify target defect reports easily, characterize the defects in them and extract other relevant information about the defect (*e.g.,* consequence, manifestation behavior, component affected) from these reports. While the use of 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 the quality of software engineering.

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.

REFERENCES

[1] Kent Beck. 2001. Manifesto for Agile Software Development. https://agilemanifesto.org/

[2] Kent Beck. 2005. *Extreme Programming explained.* John Wait.

[3] A. Boulanger. 2005. Open-source versus proprietary software: Is one more reliable and secure than the other? *IBM Systems Journal* 44, 2 (2005), 239–248. https://doi.org/10.1147/sj.442.0239 Conference Name: IBM Systems Journal.

[4] Junming Cao, Bihuan Chen, Chao Sun, Longjie Hu, and Xin Peng. 2021. Characterizing Performance Bugs in Deep Learning Systems. *arXiv:2112.01771 [cs]* (Dec. 2021). http://arxiv.org/abs/2112.01771 arXiv: 2112.01771.

[5] Haicheng Chen, Wensheng Dou, Yanyan Jiang, and Feng Qin. 2019. Understanding Exception-Related Bugs in Large-Scale Cloud Systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 339–351. https://doi.org/10.1109/ASE.2019.00040 ISSN: 2643-1572.

[6] Nicolas Dilley and Julien Lange. 2020. Bounded verification of message-passing concurrency in Go using Promela and Spin. *Electronic Proceedings in Theoretical Computer Science* 314 (April 2020), 34–45. https://doi.org/10.4204/EPTCS.314.4 arXiv: 2004.01323.

[7] Pedro Fonseca, Cheng Li, Vishal Singhal, and Rodrigo Rodrigues. 2010. A study of the internal and external effects of concurrency bugs. In *2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*. 221–230. https://doi.org/10.1109/DSN.2010.5544315 ISSN: 2158-3927.

[8] Matthew R. Fox. 2001. Failure analysis at the National Transportation Safety Board - Journal of Failure Analysis and Prevention. https://link.springer.com/article/10.1007/s11668-006-5004-5

[9] Prashant Gangidi. 2018. A systematic approach to root cause analysis using 3 × 5 why's technique. *International Journal of Lean Six Sigma* 10, 1 (Jan. 2018), 295–310. https://doi.org/10.1108/IJLSS-10-2017-0114 Publisher: Emerald Publishing Limited.

[10] Haryadi S. Gunawi, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, Thanh Do, Jeffry Adityatama, Kurnia J. Eliazar, Agung Laksono, Jeffrey F. Lukman, Vincentius Martin, and Anang D. Satria. 2014. What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems. In *Proceedings of the ACM Symposium on Cloud Computing (SOCC '14)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/2670979.2670986

[11] Hui Guo and Munindar P. Singh. 2020. Caspar: Extracting and Synthesizing User Stories of Problems from App Reviews. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. 628–640. ISSN: 1558-1225.

[12] Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2021. DeepCrime: mutation testing of deep learning systems based on real faults. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2021)*. Association for Computing Machinery, New York, NY, USA, 67–78. https://doi.org/10.1145/3460319.3464825

[13] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 159–174. https://doi.org/10.2307/2529310 Publisher: [Wiley, International Biometric Society].

[14] Tanakorn Leesatapornwongsa, Jeffrey F. Lukman, Shan Lu, and Haryadi S. Gunawi. 2016. TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, Atlanta Georgia USA, 517–530. https://doi.org/10.1145/2872362.2872374

[15] Penghui Li, Yinxi Liu, and Wei Meng. 2021. Understanding and Detecting Performance Bugs in Markdown Compilers. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 892–904. https://doi.org/10.1109/ASE51524.2021.9678611 ISSN: 2643-1572.

[16] Benjamin Liblit and Alexander Aiken. 2002. *Building a better backtrace: Techniques for postmortem program analysis.* Computer Science Division, University of California.

[17] Mario Linares-Vásquez, Gabriele Bavota, and Camilo Escobar-Velásquez. 2017. An Empirical Study on Android-Related Vulnerabilities. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 2–13. https://doi.org/10.1109/MSR.2017.60

[18] Yepang Liu, Chang Xu, and Shing-Chi Cheung. 2014. Characterizing and detecting performance bugs for smartphone applications. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 1013–1024. https://doi.org/10.1145/2568225.2568229

[19] Amir Makhshari and Ali Mesbah. 2021. IoT Bugs and Development Challenges. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, Madrid, ES, 460–472. https://doi.org/10.1109/ICSE43902.2021.00051

[20] Alejandro Mazuera-Rozo, Jairo Bautista-Mora, Mario Linares-Vásquez, Sandra Rueda, and Gabriele Bavota. 2019. The Android OS stack and its vulnerabilities: an empirical study. *Empirical Software Engineering* 24, 4 (Aug. 2019), 2056–2101. https://doi.org/10.1007/s10664-019-09689-7

[21] Alejandro Mazuera-Rozo, Catia Trubiani, Mario Linares-Vásquez, and Gabriele Bavota. 2020. Investigating types and survivability of performance bugs in mobile apps. *Empirical Software Engineering* 25, 3 (May 2020), 1644–1686. https://doi.org/10.1007/s10664-019-09795-6

[22] Audris Mockus, Roy T. Fielding, and James Herbsleb. 2000. A case study of open source software development: the Apache server. In *Proceedings of the 22nd international conference on Software engineering (ICSE '00)*. Association for Computing Machinery, New York, NY, USA, 263–272. https://doi.org/10.1145/337180.337209

[23] Mika V. Mäntylä and Casper Lassenius. 2009. What Types of Defects Are Really Discovered in Code Reviews? *IEEE Transactions on Software Engineering* 35, 3 (May 2009), 430–448. https://doi.org/10.1109/TSE.2008.71 Conference Name: IEEE Transactions on Software Engineering.

[24] Frolin S. Ocariza, Jr., Karthik Pattabiraman, and Ali Mesbah. 2014. Vejovis: suggesting fixes for JavaScript faults. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 837–847. https://doi.org/10.1145/2568225.2568257

[25] M. Paradies and D. Busch. 1988. Root cause analysis at Savannah River plant (nuclear power station). In *Conference Record for 1988 IEEE Fourth Conference on Human Factors and Power Plants,*. 479–483. https://doi.org/10.1109/HFPP.1988.27547

[26] James W Paulson, Giancarlo Succi, and Armin Eberlein. 2004. An empirical study of open-source and closed-source software products. *IEEE transactions on software engineering* 30, 4 (2004), 246–256. https://doi.org/10.1109/TSE.2004.1274044

[27] Henry Petroski. 1994. *Design Paradigms: Case Histories of Error and Judgment in Engineering*. Cambridge University Press. Google-Books-ID: C_ZroS6rY54C.

[28] Paul Ralph, Sebastian Baltes, Domenico Bianculli, Yvonne Dittrich, Michael Felderer, Robert Feldt, Antonio Filieri, Carlo Alberto Furia, Daniel Graziotin, Pinjia He, Rashina Hoda, Natalia Juristo, Barbara Kitchenham, Romain Robbes, Daniel Mendez, Jefferson Molleri, Diomidis Spinellis, Miroslaw Staron, Klaas Stol, Damian Tamburri, Marco Torchiano, Christoph Treude, Burak Turhan, and Sira Vegas. 2020. ACM SIGSOFT Empirical Standards. https://onikle.com/articles/288927

[29] Sarah Rastkar, Gail C. Murphy, and Gabriel Murray. 2010. Summarizing software artifacts: a case study of bug reports. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, Vol. 1. 505–514. https://doi.org/10.1145/1806799.1806872 ISSN: 1558-1225.

[30] Sarah Rastkar, Gail C. Murphy, and Gabriel Murray. 2014. Automatic Summarization of Bug Reports. *IEEE Transactions on Software Engineering* 40, 4 (April 2014), 366–380. https://doi.org/10.1109/TSE.2013.2297712 Conference Name: IEEE Transactions on Software Engineering.

[31] James Reason. 1990. *Human Error*. Cambridge University Press. Google-Books-ID: WJL8NZc8lZ8C.

[32] J Reason. 1997. Organizational accidents: the management of human and organizational factors in hazardous technologies. *England: Cambridge University Press, Cambridge* (1997).

[33] Benedetta Rosadini, Alessio Ferrari, Gloria Gori, Alessandro Fantechi, Stefania Gnesi, Iacopo Trotta, and Stefano Bacherini. 2017. Using NLP to Detect Requirements Defects: An Industrial Experience in the Railway Domain. In *Requirements Engineering: Foundation for Software Quality (Lecture Notes in Computer Science)*, Paul Grünbacher and Anna Perini (Eds.). Springer International Publishing, Cham, 344–360. https://doi.org/10.1007/978-3-319-54045-0_24

[34] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. 2007. Detection of Duplicate Defect Reports Using Natural Language Processing. In *29th International Conference on Software Engineering (ICSE'07)*. 499–510. https://doi.org/10.1109/ICSE.2007.32 ISSN: 1558-1225.

[35] Qingchao Shen, Haoyang Ma, Junjie Chen, Yongqiang Tian, Shing-Chi Cheung, and Xiang Chen. 2021. A comprehensive study of deep learning compiler bugs. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 968–980. https://doi.org/10.1145/3468264.3468591

[36] M. Sullivan and R. Chillarege. 1992. A comparison of software defects in database management systems and operating systems. In *[1992] Digest of Papers. FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing*. 475–484. https://doi.org/10.1109/FTCS.1992.243586

[37] Chengnian Sun, Vu Le, Qirun Zhang, and Zhendong Su. 2016. Toward understanding compiler bugs in GCC and LLVM. In *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016)*. Association for Computing Machinery, New York, NY, USA, 294–305. https://doi.org/10.1145/2931037.2931074

[38] Lin Tan, Chen Liu, Zhenmin Li, Xuanhui Wang, Yuanyuan Zhou, and Chengxiang Zhai. 2014. Bug characteristics in open source software. *Empirical Software Engineering* 19, 6 (Dec. 2014), 1665–1705. https://doi.org/10.1007/s10664-013-9258-8

[39] E. Ubani and C. Ononuju. 2013. A study of failure and abandonment of public sector-driven civil engineering projects in Nigeria: An empirical review. *American Journal of Scientific and Industrial Research* 4, 1 (Feb. 2013), 75–82. https://doi.org/10.5251/ajsir.2013.4.1.75.82

[40] Dinghua Wang, Shuqing Li, Guanping Xiao, Yepang Liu, and Yulei Sui. 2021. An exploratory study of autopilot software bugs in unmanned aerial vehicles.

In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 20–31. https://doi.org/10.1145/3468264.3468559

[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.