A Reflection on "An Exploratory Study on Exception Handling Bugs in Java Programs"

Felipe Ebert*, Fernando Castor*, Alexander Serebrenik†
*Federal University of Pernambuco, Brazil, {fe,castor}@cin.ufpe.br
†Eindhoven University of Technology, The Netherlands, a.serebrenik@tue.nl

***Abstract*—Exception handling is a feature provided by most mainstream programming languages, and typically involves constructs to throw and handle error signals. On the one hand, early work has argued extensively about the benefits of exception handling, such as promoting modularity by defining how exception handlers can be implemented and maintained independently of the normal behavior of the system and easing but localization. On the other hand, some studies argue that exception handling can make the programming languages unnecessarily complex and promote the introduction of subtle bugs in programs. In 2015 we published a paper describing a study investigating the prevalence and nature of exception handling bugs in two large, widely adopted Java systems. This study also confronted its findings about real exception handling bugs with the perceptions of developers about those bugs, also accounting for bugs not related to exception handling. The goal of this reflection paper is to investigate the state of the art in exception handling research, with a particular emphasis on exception handling bugs, and how our paper has influenced other studies in the area. We found that our paper was cited by 33 articles, and all themes for future work we raised in our paper have been tackled by other studies in the short span of five years.**

***Index Terms*—Exception handling, bugs, reflection study**

I. Introduction

Most modern programming languages include specific constructs to signal the occurrence of an error and, when this occurs, transfer control to another part of a program, responsible for handling it. These errors are expected to be rare and are thusly called **exceptions**, since they represent deviations to the normal behavior of the system. In a similar vein, the parts of the code responsible for capturing errors and bringing the system back to a consistent state are called **exception handlers**. The combination of these constructs in a programming language, together with the rules by which they interact, is called **exception handling mechanism**.

Early work [1], [2], [3] advocates multiple benefits of exception handling. According to these papers, exception handling promotes modularity because it makes it possible to define how errors can be handled independently of the normal behavior of the system. This is reinforced by the lexical separation created by exception handling mechanisms. This separation should also make it easier to locate the causes of bugs. In addition, by leveraging exception handling, it should be possible to incrementally implement the error recovery logic of a system.

As a counterpoint, many criticisms have been made against exception handling throughout the years. Black [4] already in 1982 argues that exception handling mechanisms make programming languages unnecessarily complex. Cargill [5] shows by means of a simple example how a combination of manual memory allocation and exception handling can lead to the introduction of subtle bugs in a program. Robillard and Murphy [6] provide evidence that bugs stemming from the use of exception handling can also happen as a consequence of complex and unintended exception propagation paths. Adding insult to injury, a number of papers show that developers do not prioritize exception handling and that this part of the code usually exhibits low quality [7], [8].

A couple of studies published around the turn of the decade attempted to provide a better understanding of the impacts [9], prevalence, and nature [10] of exception handling-related bugs. Notwithstanding, they exhibit limitations in the methodology they employ to identify exception handling bugs [10] and do not study the characteristics of these bugs [9]. Additionally, we were not aware at the time of any paper confronting the beliefs of developers with actual problems reported during system development. Hence, around that time we started working to fill this gap, which culminated with the publication in 2015 of the paper "An Exploratory Study on Exception Handling Bugs" [11].[1] Hereafter, we will refer to it as "our paper".

In this reflection paper, we examine the state of the art in exception handling research and how our paper has influenced work in the area. In particular, our investigation shows that all the five themes for future work we raised in our paper have actually been tackled by other researchers in subsequent years.

II. Summary and Main Contributions of Our Paper

In this section, we discuss the summary of our paper, its main contributions, and the future work presented at that time. Our paper [11] aimed to study exception handling bugs from two different perspectives: (i) exception handling bugs that occur in real software systems; and (ii) developers' perceptions of these bugs. To realize those goals we surveyed 154 developers, asking them questions about organizational policies pertaining to exception handling and their perceptions on exception handling bugs. At the time, this was the largest study about exception handling directly involving software developers. We also manually analyzed 220 exception handling bugs from Eclipse and Tomcat. Less than 1/3 of the respondents (27%) of the survey indicated that their organizations have policies and standards for the implementation of exception

[1]This is an extended version of another paper [12].

TABLE I
COMPREHENSIVE CLASSIFICATION OF EXCEPTION HANDLING BUGS (REPRODUCED FROM [11]).

Lack of a handler that should exist
Exception not thrown
Error in the handler
Error in the clean-up action
Exception caught at the wrong level
General `catch` block
Wrong exception thrown
Exception that should not have been thrown
Wrong encapsulation of exception cause
Lack of a `finally` block that should exist
Error in the exception assertion
Inconsistency between source code and API documentation
Empty `catch` block
Error in the definition of exception class
`catch` block where only a `finally` would be appropriate

handling. Similarly, 70% of the respondents claimed that in their organizations there are no specific tests for exception handling code. Moreover, most of the the respondents (61%) mentioned that *no* to *little* importance is given to the documentation of exception handling in the design phase of the project. Additionally, 40% of the respondents considered the quality of exception handling code *good* or *very good*, while only very few of them (14%) considered it *bad* or *very bad*.

Somewhat in contrast to the perceptions of the surveyed developers, our study of exception handling bugs showed, with statistical significance, that bugs in exception handling are ignored by developers less often than other bugs. Another interesting finding is that bugs stemming from overly *general catch blocks* are rare, even though they are considered a bad smell related to exceptions [6], [7]. In addition, we identified few bug reports caused by *empty catch blocks*, although developers often mention them as causes of bugs they have fixed in the past. Furthermore, we observed that *empty catch blocks* are used as part of the bug fixes, including exception handling bug fixes. Finally, based on all the findings, we presented a classification of exception handling bugs and their causes. This classification is reproduced in Table I.

We discussed five different threads for future work, in no particular order. Firstly, to extend the study by involving richer data including interviews with developers and larger-scale surveys. Secondly, to analyze *empty catch blocks* in more depth, since their use is pervasive in the systems we analyzed and they are used in unforeseen ways, *e.g.*, to deal with exception handling bugs. Thirdly, to develop recommendation approaches to assist developers in deciding how to handle exceptions, since we identified multiple cases where developers explicitly stated they did not know what to do with an exception. Fourthly, to enrich analysis tools by tracking the typical causes of exception handling bugs and adherence to exception handling policies. This is motivated in part by the observation that *empty catch blocks* are recognized by developers as being useful if the implementation of the system guarantees that the exception will not be thrown, but such guarantee might be violated when the system is evolving, which may lead to unexpected behavior. Lastly, to study how the exception signaling and handling behavior of a system evolves over time, specially looking at information from version control systems.

III. IMPACTS

In this section, we position our paper in consideration of the recent state of the art and practice. We do this by surveying work that has cited it between 2015 (it was published online in April 2015) and December 2019. According to GOOGLE SCHOLAR[2], our paper [11] has been cited 33 times, once in a book, six times in journals, 25 times in conferences and workshops, and once in a thesis.[3] We could observe that our article more directly influenced 28 of the 33 works, *i.e.*, in five of them our article was just quickly mentioned. Among these 28, only one is a self-citation. Two of the citing articles explicitly mentioned our article as an inspiration for their study [13], [14]. The work of Coelho *et al.* [14] is a comprehensive catalogue of exception handling bad smells and associated refactorings for Java. Below we provide a brief overview of the citing papers, followed by a more detailed discussion of the most relevant ones.

Three papers and the thesis of Pádua *et al.* [15], [16], [17], [18] cite our article when discussing the importance of exception handling code. Similarly to our article, 19 different papers focus on bugs in the exception handling code [19], [20], [21], [22], [23], [24], [15], [25], [26], [27], [17], [14], [28], [18], [29], [30], [31], [32], [13]. Our categorization of exception handling bugs and their causes has been (partially) used in five different studies [20], [15], [26], [33], [18] The survey we conducted in our article also served as a basis for surveys by de Sousa [13], [34]. Finally, we found 15 papers addressing the five themes for future work that our paper discussed [23], [25], [26], [27], [35], [17], [36], [33], [29], [37], [30], [31], [32], [13], [38].

A. Have Other Papers Followed Up?

In this section, we discuss work which, we believe, has followed up on the groundwork established by our paper. In particular, we examine work that has tackled at least one of the lines for future work mentioned in Section II.

Pádua and Shang [17] studied bugs related to exception propagation and their relationship with exception handling anti-patterns. Our work served as a rationale for three out of the 15 exception flow-related metrics used in the study. The occurrence of *generic catch blocks* was shown to have a positive correlation with the probability of post-release bugs in one of the analyzed systems; the use of *dummy handlers*, *i.e.*, handlers that only display or log information—in two.

Three studies involved interviews with developers, another line for future work we discussed. Cassee *et al.* [26] focused on the error handling mechanism of the Swift programming language. The main goal of the interviews was to investigate the adherence to guidelines and best practices about the

[2] shorturl.at/IMOS1

[3] Scholar shows 36, but two are duplicated entries and one is not in English.

development of the exception handling code. In our original survey, we had a question related to this topic. They found that exception handling guidelines usually exist, but such guidelines are usually implicit and undocumented. Melo *et al.* [37] conducted interviews with a similar goal, but focusing on Java developers. Their results show analogous trends: exception handling guidelines usually exist and they are usually implicit and undocumented. The work of deSousa *et al.* [13] used interviews to aid in the investigation of the relationship between the absence of exception handling policies and the occurrence of exception handling anti-patterns. Their findings converge to the same result: developers face development problems, such as lack of documentation, lack of exception handling policies, and the absence of tools to detect improper exception handling practices.

Nogueira *et al.* [28] conducted a study focusing specifically on gaining a better understanding about *empty catch blocks*. They investigated how *empty catch blocks* evolve within the software releases and also which types of exceptions these blocks handle. Their results show that the number of *empty catch blocks* decreased along the versions, and the most common exception handled is the most generic one: `Exception`. In our previous article, we found diverging opinions about the benefits and drawbacks of *empty catch blocks*, and few bugs related to both *empty catch blocks* and *general catch blocks*.

Two lines for future work mentioned in our paper are tool-related: tools to support developers in deciding what to do in the presence of exceptions and tools to check violations of (potentially implicitly defined) exception signaling and handling policies. We found 11 papers presenting such tools, however one of them is just about a high level idea of extending Java's exception type hierarchy to accommodate "context-dependent exceptions" that can be either checked or unchecked depending on the usage context [25].

Many papers present tools that verify conformance to exception handling policies and provide some recommendations for handling exceptions. Violations to these policies are considered exception handling bugs. Barbosa and Garcia [23] proposed a recommender heuristic strategy that recommends repairs to exception policy violations. Kistner *et al.* [35] built a tool that reveals possible sources of exceptions in the code. This tool also provides project-specific recommendations and detects common bad exception handling practices. Montenegro *et al.* [33] proposed an "exception policy expert" tool which alerts developers about policy violations and can suggest possible handlers for the exceptions. Nguyen *et al.* [29] developed a tool to support developers by recommending code to catch an exception that is likely to occur in a code snippet. This tool also recommends code to fix the occurrence of such exception, in case it stems from a bug. In a similar vein, Li *et al.* [36] devised a method that automatically recommends exception handling strategies, based on program context.

Three studies proposed a policy check for exception handling code (without promoting recommendations on the exception handling code) [27], [30], [31]. Filho *et al.* [27] implemented an architectural conformance checking solution to help avoiding exception handling design erosion. Their tool provides a declarative language for expressing design constraints regarding exception handling, and a design rule checker to verify the exception handling conformance automatically. Chen *et al.* [30] provided a static analysis tool which automatically detects inaccurate exceptions, *i.e.*, exception handling bugs or anti-patterns, by inspecting the inconsistency between an exception's class and its error message. Kechagia and Spinellis [31] designed a type system to support the identification of bugs that can lead to application crashes due to malformed inputs. One of the more concrete effect of this approach is the conversion of methods that thrown unchecked exceptions into methods that throw checked exceptions.

The aforementioned approaches work at development time. Additionally, we identified two papers presenting tools that check exception handling policies and provide exception handling support based on run time information [32], [38]. For example, the work of Jia and colleagues [32] uses run time information about expected behavior of a system to identify actions that should be performed when the execution of a system is inadvertently interrupted.

Finally, our article was cited in a book on *Cooperative Software Development* [39]. More specifically, in the Functional Specifications Chapter, the author used our study as evidence that developers are not good at designing errors. In particular, it highlights that overly *general catch blocks* hinder developer's understanding of the exceptions these blocks capture.

B. Discussion of Exception Handling Bugs

Since the publication of our paper, a few more papers have studied the prevalence of exception handling bugs in real-world systems. In our study, we found that exception handling bugs are rare. In the repository analysis of ECLIPSE and TOMCAT, we observed only 0.35% and 1.87% of the bugs are related to the exception handling code, respectively. Differently from these findings, Chen *et al.* [30] found about 13% of exception handling bugs in six widely-deployed cloud-based systems (CASSANDRA, HBASE, HDFS, HADOOP MAPREDUCE, YARN, and ZOOKEEPER). Sena *et al.* [19] found an even higher percentage (about 20%) of bugs related to exception handling in popular Java libraries (COMMONS-COLLECTIONS, COMMONS-LANG, LOG4J, MAVEN-FILTERING, PLEXUS-UTILS, EASYMOCK, and SLF4J-API).

We have analyzed a sample of exception handling bugs from one stand alone and one server applications. Both exhibit a minimal number of exception handling bugs (less that 2% each). The study of Chen *et al.* [30] analyzed cloud-based applications such as databases and distributed resource management systems. Sena *et al.* [19] focused on a different kind of systems: Java API libraries. New studies are necessary to identify what is the source of these differences in the prevalence of exception handling bugs.

IV. NEW EMERGING IDEAS AND CURRENT VISION

In this section, we discuss the future work we envision from the actual state of the art on exception handling studies.

While error handling practices in Java systems have been studied in great depth, there is comparatively little work targeting other languages. To the best of our knowledge, very few papers studied exception handling practices, perceptions, and bugs in popular[4] languages such as JavaScript, Python, PHP, Go, and Objective-C. Previous work targeting Swift [26] and C++ [24], [40] has shown that, although the usage of exception handling (called "error handling" in Swift) has points in common with its usage in Java, in particular the use of empty handlers, there are also significant differences, *e.g.*, generic handlers are more prevalent in Swift.

Following along the steps of the work by Pádua and Shang[17], we think it is worth conducting a more in-depth validation of existing best practices and anti-patterns. Developers have been building intensively-used, mission-critical systems that use exceptions for a long time. Some of the practices employed in the design and implementation of exception handling code in these systems can be seen as anti-patterns. However, in some circumstances developers see them as useful solutions, as in the example of developers using *empty catch blocks* as fixes for exception handling bugs (Section II). Devising new approaches to mine these solutions from source code and confronting them with the perceptions of developers about their usefulness could be a valuable contribution. This is a scenario where machine learning techniques, both supervised and unsupervised, can be helpful.

We would like to extend the analysis of the prevalence of exception handling bugs to different systems. The studies by Chen *et al.* [30] and Sena *et al.* [19], focusing on cloud systems and Java libraries, found a large number of exception handling bugs, when compared to our work. These findings suggest that the prevalence of exception handling bugs varies widely across domains, possibly with other variables such as maturity of the project and employed programming languages playing important parts. These results highlight that there is more to learn about the prevalence and maybe also about the nature of exception handling bugs. One big obstacle in this case is how to scale up the identification of exception handling bugs. Simplistic automated approaches used in previous work [10] yield unreliable results. This might be another avenue where machine learning techniques can prove useful.

At a more basic level, we think that exception handling mechanisms are treated as a second-class citizen in programming languages. Code coverage approaches often do not account for exception handling code. Complexity measures typically do not include control flows that stem from exception handling, neither implicit nor explicit, in the information they collect. Even though a number of papers discuss the complexity of implementing exception handling code and there are some approaches to support program comprehension in the presence of exceptions, to the best of our knowledge, no previous work has studied the readability of exception handling code. For example, what is more readable for exception-safe resource clean-up, Swift's `defer` or Java's `try-finally`?

[4]https://redmonk.com/sogrady/2019/07/18/language-rankings-6-19

Finally, there are many papers exploring the use of static analysis to support comprehension of and bug detection in exception handling code [41], [42], [21], [6], [43], [44]. Notwithstanding, perhaps surprisingly, there is comparatively little work studying how exception handling code is tested or debugged in practice. In a similar vein, there are few proposals to improve testing and debugging of exception handling code, with few exceptions [45], [46]. We hope to see this scenario change in the coming years, considering the many published studies targeting practical aspects of exception handling published in the last decade.

V. Concluding Remarks

Most major programming languages in widespread use, both old and new, implement an exception handling mechanism. For example, the first version of Swift, released in 2014, did not include specific features for signaling and handling errors. However, due to requests from the community, Apple ended up adding such features to the language one year later. This is clear evidence of the relevance of exception handling to the practice of software development. Researchers acknowledge this. Even though exception handling cannot be considered a popular research topic, there is a steady stream of publications on the topic in major Software Engineering research venues [40], [30], [47], [37], [48], [17]. In this paper, we re-examined a paper that investigated the nature, prevalence, and perceptions about bugs related to exception handling usage. In particular, we analyzed its impact and how it influenced other papers in the field.

We observed that a total of 33 articles cited our paper and 28 of them were directly influenced by it. Two of these studies explicitly admit to being inspired by our previous work. We have also shown that all the lines for future work raised by our paper have been addressed to some extent by subsequent research. At the same time, we presented several ideas for future research that, we hope, can guide researchers to continue further investigating exception handling.

Acknowledgments

This research was partially funded by CNPq/Brazil (304755/2014-1, 406308/2016-0, 465614/2014-0), FACEPE/Brazil (APQ-0839-1.03/14, 0388-1.03/14, 0592-1.03/15, BCT-0229-1.03/19), and CAPES/Brazil (88887.333966/2019-00).

References

[1] F. Cristian, "A recovery mechanism for modular software," in *ICSE*, 1979, pp. 42–51.

[2] P. M. Melliar-Smith and B. Randell, "Software reliability: The role of programmed exception handling," in *Proceedings of an ACM Conference on Language Design for Reliable Software (LDRS)*, Raleigh, USA, March 1977, pp. 95–100.

[3] D. L. Parnas and H. Würges, "Response to undesired events in software systems," in *ICSE*, 1976, pp. 437–446.

[4] A. P. Black, "Exception handling : The case against," Ph.D. dissertation, University of Oxford, UK, October 1982.

[5] T. Cargill, "Exception handling: A false sense of security," *C++ Report*, vol. 6, no. 9, November-December 1994.

[6] M. Robillard and G. Murphy, "Static analysis to support the evolution of exception structure in object-oriented systems," *TOSEM*, vol. 12, no. 2, pp. 191–221, April 2003.

[7] D. Reimer and H. Srinivasan, "Analysing exception usage in large Java applications," in *Proceedings of ECOOP Workshop on Exception Handling in Object-Oriented Systems*, July 2003, pp. 10–19.

[8] H. B. Shah, C. Gorg, and M. J. Harrold, "Understanding exception handling: Viewpoints of novices and experts," *TSE*, vol. 36, no. 2, pp. 150–161, 2010.

[9] C. Marinescu, "Are the classes that use exceptions defect prone?" in *Proceedings of the 12th International Workshop on Principles of Software Evolution*, September 2011, pp. 56–60.

[10] P. Sawadpong, E. B. Allen, and B. J. Williams, "Exception handling defects: An empirical study," *9th IEEE International Symposium on High-Assurance Systems Engineering*, pp. 90–97, October 2012.

[11] F. Ebert, F. Castor, and A. Serebrenik, "An exploratory study on exception handling bugs in Java programs," *Journal of Systems and Software*, vol. 106, pp. 82–101, 2015.

[12] F. Ebert and F. Castor, "A study on developers' perceptions about exception handling bugs," in *ICSM*, September 2013.

[13] D. B. C. de Sousa, P. H. Maia, L. S. Rocha, and W. Viana, "Analysing the evolution of exception handling anti-patterns in large-scale projects: A case study," in *SBCARS*. ACM, 2018, pp. 73–82.

[14] R. Coelho, J. Rocha, H. Melo, and B. SENA, "A catalogue of java exception handling bad smells and refactorings," in *Conference on Pattern Languages of Programs (PLoP)*, Portland, 2018.

[15] G. B. D. Pádua and W. Shang, "Studying the prevalence of exception handling anti-patterns," in *ICPC*, May 2017, pp. 328–331.

[16] G. B. d. Pádua and W. Shang, "Revisiting exception handling practices with exception flow analysis," in *SCAM*, Sep. 2017, pp. 11–20.

[17] G. B. de Pádua and W. Shang, "Studying the relationship between exception handling practices and post-release defects," in *MSR*, 2018, pp. 564–575.

[18] G. B. de Pádua, "Studying and assisting the practice of java and c# exception handling," Master's thesis, Concordia University, 2018.

[19] D. Sena, R. Coelho, U. Kulesza, and R. Bonifácio, "Understanding the exception handling strategies of java libraries: An empirical study," in *MSR*. ACM, 2016, pp. 212–222.

[20] E. A. Barbosa, A. Garcia, M. P. Robillard, and B. Jakobus, "Enforcing exception handling policies with a domain-specific language," *TSE*, vol. 42, no. 6, pp. 559–584, 2016.

[21] J. Oliveira, D. Borges, T. Silva, N. Cacho, and F. Castor, "Do android developers neglect error handling? a maintenance-centric study on the relationship between android abstractions and uncaught exceptions," *JSS*, vol. 136, pp. 1 – 18, 2018.

[22] J. Oliveira, N. Cacho, D. Borges, T. Silva, and F. Castor, "An exploratory study of exception handling behavior in evolving android and java applications," in *SBES*, 2016, pp. 23–32.

[23] E. A. Barbosa and A. Garcia, "Global-aware recommendations for repairing violations in exception handling," *TSE*, vol. 44, no. 9, pp. 855–873, Sep. 2018.

[24] R. Bonifácio, F. Carvalho, G. N. Ramos, U. Kulesza, and R. Coelho, "The use of c++ exception handling constructs: A comprehensive study," in *SCAM*, Sep. 2015, pp. 21–30.

[25] M. Kechagia, T. Sharma, and D. Spinellis, "Towards a context dependent Java exceptions hierarchy," in *ICSE-Companion*, May 2017, pp. 347–349.

[26] N. Cassee, G. Pinto, F. Castor, and A. Serebrenik, "How swift developers handle errors," in *MSR*, May 2018, pp. 292–302.

[27] J. L. M. Filho, L. Rocha, R. Andrade, and R. Britto, "Preventing erosion in exception handling design using static-architecture conformance checking," in *Software Architecture*, A. Lopes and R. de Lemos, Eds. Cham: Springer International Publishing, 2017, pp. 67–83.

[28] A. F. Nogueira, J. C. B. Ribeiro, and M. A. Zenha-Rela, "Trends on empty exception handlers for java open source libraries," in *SANER*, Feb 2017, pp. 412–416.

[29] T. T. Nguyen, P. M. Vu, and T. T. Nguyen, "Recommendation of exception handling code in mobile app development," 2019, arXiv, 1908.06567.

[30] H. Chen, W. Dou, Y. Jiang, and F. Qin, "Understanding exception-related bugs in large-scale cloud systems," in *ASE*, November 2019.

[31] M. Kechagia and D. Spinellis, "Type checking for reliable apis," in *2017 IEEE/ACM 1st International Workshop on API Usage and Evolution (WAPI)*, May 2017, pp. 15–18.

[32] Z. Jia, S. Li, T. Yu, X. Liao, and J. Wang, "Automatically detecting missing cleanup for ungraceful exits," in *ESEC/FSE*, 2019, pp. 751–762.

[33] T. Montenegro, H. Melo, R. Coelho, and E. Barbosa, "Improving developers awareness of the exception handling policy," in *SANER*, March 2018, pp. 413–422.

[34] D. B. C. de Sousa, W. V. de Carvalho, and L. S. Rocha, "Avaliando o tratamento de exceção em um sistema web corporativo: Um estudo de caso," in *Anais Estendidos do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web*. Porto Alegre, RS, Brasil: SBC, 2017, pp. 23–28.

[35] F. Kistner, M. Beth Kery, M. Puskas, S. Moore, and B. A. Myers, "Moonstone: Support for understanding and writing exception handling code," in *VL/HCC*, Oct 2017, pp. 63–71.

[36] Y. Li, S. Ying, X. Jia, Y. Xu, L. Zhao, G. Cheng, B. Wang, and J. Xuan, "Eh-recommender: Recommending exception handling strategies based on program context," in *2018 23rd International Conference on Engineering of Complex Computer Systems*, Dec 2018, pp. 104–114.

[37] H. Melo, R. Coelho, and C. Treude, "Unveiling exception handling guidelines adopted by java developers," in *SANER*, Feb 2019, pp. 128–139.

[38] L. Zhang and M. Monperrus, "Tripleagent: Monitoring, perturbation and failure-obliviousness for automated resilience improvement in java applications," *CoRR*, vol. abs/1812.10706, 2018.

[39] A. J. Ko and B. Xie, "Cooperative software development." [Online]. Available: https://andrewbegel.com/info461/readings/index.html

[40] K. Bradley and M. Godfrey, "A study on the effects of exception usage in open-source C++ systems," in *SCAM*, Sep. 2019.

[41] M. Bravenboer and Y. Smaragdakis, "Exception analysis and points-to analysis: better together," in *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, July 2009, pp. 1–12.

[42] C. Fu and B. G. Ryder, "Exception-chain analysis: Revealing exception handling architecture in java server applications," in *ICSE*, May 2007, pp. 230–239.

[43] C. F. Schaefer and G. N. Bundy, "Static analysis of exception handling in Ada," *Softw., Pract. Exper.*, vol. 23, no. 10, pp. 1157–1174, 1993.

[44] W. Weimer and G. C. Necula, "Exceptional situations and program reliability," *TOPLAS*, vol. 30, no. 2, pp. 1–51, 2008.

[45] P. Zhang and S. Elbaum, "Amplifying tests to validate exception handling code," in *In Proceedings of 34th International Conference on Software Engineering*, june 2012, pp. 595 –605.

[46] E. S. F. Najumudheen, R. Mall, and D. Samanta, "Modeling and coverage analysis of programs with exception handling," in *ISEC*. New York, NY, USA: ACM, 2019, pp. 15:1–15:11.

[47] L. Fan, T. Su, S. Chen, G. Meng, Y. Liu, L. Xu, G. Pu, and Z. Su, "Large-scale analysis of framework-specific exceptions in android apps," in *ICSE*, 2018, pp. 408–419.

[48] T. Nguyen, P. Vu, and T. Nguyen, "Recommending exception handling code," in *ICSME*, September/October 2019.