International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

A Comparative Study of Error Handling Techniques in Different Programming Languages

Adewole A.P, Qazim M. Onisesi, Sodiq O. Jimoh, Isaiah O. Aimiton, Funmilola J. Ayinde
Department of Computer Science
University of Lagos, Nigeria

ABSTRACT:- Exception handling is a core feature of modern programming languages, yet its implementation and typical usage vary across ecosystems. This paper compares exception handling in Java, Python, and C#, focusing on how exceptions are raised, propagated, and handled, and how these models influence maintainability, debugging, testing, and robustness in real systems. Using verified peer-reviewed studies (2019–2024), high-quality arXiv preprints, and official language documentation, the paper synthesises evidence on exception-handling anti-patterns, their evolution, exceptional-path testing, and developer-support-seeking behaviour. The reviewed evidence indicates that anti-patterns are widespread, exceptional behaviour is frequently under-tested, and exception-related debugging imposes substantial developer effort. The research shows no single exception model fully handles these problems; effective practice requires consistent project-level policies, careful handler design, and automated tooling to detect high-risk patterns and preserve diagnostic information.

KEYWORDS: Exception handling, Java, Python, C#, anti-patterns, maintainability, debugging, testing, robustness, and empirical software engineering.

INTRODUCTION

Exception handling separates “normal” programme logic from “exceptional” control flow. When an exception is raised/thrown, the runtime searches for a matching handler; if none is found, execution terminates with an error report. Python’s tutorial specifies this propagation model and notes that exceptions not handled by an except clause are re-raised after the finally completes. [12] C# documentation likewise defines throw/try/catch/finally for raising and dealing with exceptions within the CLR. [13]

Java differs from Python and C# in that it has compile-time checking for checked exceptions. The Java Language Specification states that handlers are required for checked exceptions. This compile-time checking is designed to reduce the number of unhandled exceptions.

Oracle’s tutorial explains the “catch or specify” requirement and its motivations. This design moves part of error management into API design, through signatures and contracts. In contrast, Python and C# shift more responsibility to documentation, conventions, and tools.
Recent empirical studies show that exception handling remains a persistent source of quality risk. Multi-language mining reports that exception-handling anti-patterns appear across all evaluated repositories and that Java exhibits more anti-pattern occurrences than Python in the analysed multi-language dataset. [3] Longitudinal analysis of a long-lived Java system indicates that the absence of an explicit exception-handling policy and guidance contributes to the replication and spread of anti-patterns over time. [4] Testing studies additionally suggest exceptional behaviour is systematically under-tested relative to normal behaviour, even in mature ecosystems. [16]

This paper discusses the following: how each language raises, propagates, and handles exceptions; what misuse patterns arise in practice; and how these mechanisms affect maintainability, debugging, testing and dependability in real systems.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

LITERATURE REVIEW

Empirical work since 2019 displays consistent evidence of exception-handling misuse, evolving technical debt, and under-tested exceptional behaviour.

A key theme is the prevalence and evolution of anti-patterns. In a longitudinal case study of a large, long-lived Java web system, de Sousa et al. argue that exception-handling quality is negatively affected by the absence (or low awareness) of explicit exception-handling policy and guidelines, and by a "silent rising" of anti-patterns across releases. [4] This supports an organisational mechanism: when guidance is weak, developers replicate existing patterns, and anti-patterns spread through new features rather than being corrected. [4]

Cross-language relevance grows with Exception Miner (SBES 2024). It provides unified anti-pattern detection across Java, Python, and TypeScript. The authors report that all multi-language projects in their evaluation contain anti-patterns in at least one language; they also report that Java exhibits more anti-patterns than Python and TypeScript in the sample. Exception Swallowing and Destructive Wrapping are among the most frequent. This matters because it applies anti-pattern definitions consistently, reducing single-language study bias.
A second theme is maintainability and design impact. Oliveira et al. study robustness changes within catch blocks, reporting that poor robustness changes (e.g., addition of empty catch blocks, catching overly general types) correlate with maintainability smells and with evidence of design problems (e.g., Concern Overload and Misplaced Concern). [5] Their results imply that exceptional code should be treated as a first-class signal of maintainability: ignoring catch-block quality can mask deeper design degradation. [18]

A third theme is testing exceptional behaviour. Marcilio & Furia (MSR 2021) provide large-scale evidence from 1,157 open-source Java projects: 66% of projects with tests contain at least one exceptional test, yet exceptional tests are ~13% of all tests; exceptional tests are larger on average, and try/catch is the most common pattern for writing them. [6] Lima et al. (EMSE 2021) study 27 long-lived Java libraries and show that instructions/branches within catch blocks and throw instructions are less covered (statistically) than overall code; however, the mutation analysis shows that many test suites still kill a large fraction of injected exception-handling mutants (68% overall; >70% in most studied libraries). [7] A complementary study (EASE 2020) also reports that many projects have at least some exceptional behaviour tests, but the fraction is often small relative to total tests. [19]

A fourth theme is debugging effort and support-seeking behaviour. Hassan et al. (ESEM 2020) analyse a random sample of 5 billion web search queries and extract ~0.98 million exception-related queries; they build an ML extractor (reported F1 ≈ 0.82) and perform analyses of effort, success, and language-specific behaviour. [8] Ghadesi et al. (2023) mined 11,449 ML-related stack traces from Stack Overflow for seven Python ML libraries and reported that questions containing stack traces are more popular but less likely to have accepted answers; they also showed that exceptions and traces can span the entire software stack (applications and libraries). [9]

Finally, robustness evidence from real systems within complex environments shows that exception misuse can have a severe impact. Chen et al. (ASE 2019) studied 210 exception-related bugs (eBugs) across widely deployed cloud systems and noted that misuse of exceptions can lead to service interruptions and data loss. [20] Systems research further highlights why diagnosis is hard: ExChain (NSDI 2024) focuses on failures characterised by implicit state dependencies, silent handling without logging, and separation between root-cause exceptions and failure manifestation. [21]

The literature is still uneven: deep testing and evolution studies are strongest for Java, while comparable large-scale measurements for Python and C# remain scarcer in the verified set—supporting the need for systematic cross-language comparisons that combine official semantics with data-driven evidence. [22]

METHODOLOGY

This study follows a literature-based comparative approach. It synthesises official language documentation for semantic baselines (raise/throw, propagation, handler matching, and rethrow behaviour) and verified empirical studies (2019–2024) for real-system impacts on maintainability, debugging, testing and robustness.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Selection criteria were: (i) peer-reviewed SE venues (2019–2024) and high-quality arXiv preprints with traceable identifiers; (ii) direct relevance to at least one comparison dimension; and (iii) availability of a verifiable DOI/arXiv ID or official documentation page. A single 2018 MSR paper is included for background only and is clearly marked outside the 2019–2024 window. [23]

Comparison dimensions (applied consistently to Java, Python, C#): exception model; propagation visibility; handler design; recoverability; readability/maintainability; anti-pattern exposure; testing coverage/effectiveness; debugging/diagnostic fidelity. These dimensions are grounded in (a) official semantics and (b) recurring empirical findings on anti-patterns, exceptional testing and debugging behaviour. [24]

Code examples are selected as minimal contrasts: each shows a commonly reported anti-pattern (e.g., swallowing, destructive wrapping, stack-trace loss on rethrow) and a corrected version aligned with official guidance (e.g., Python re-raise and hierarchy rules; .NET CA2200; Java checked-exception contracts). [25]

COMPARISON STUDY

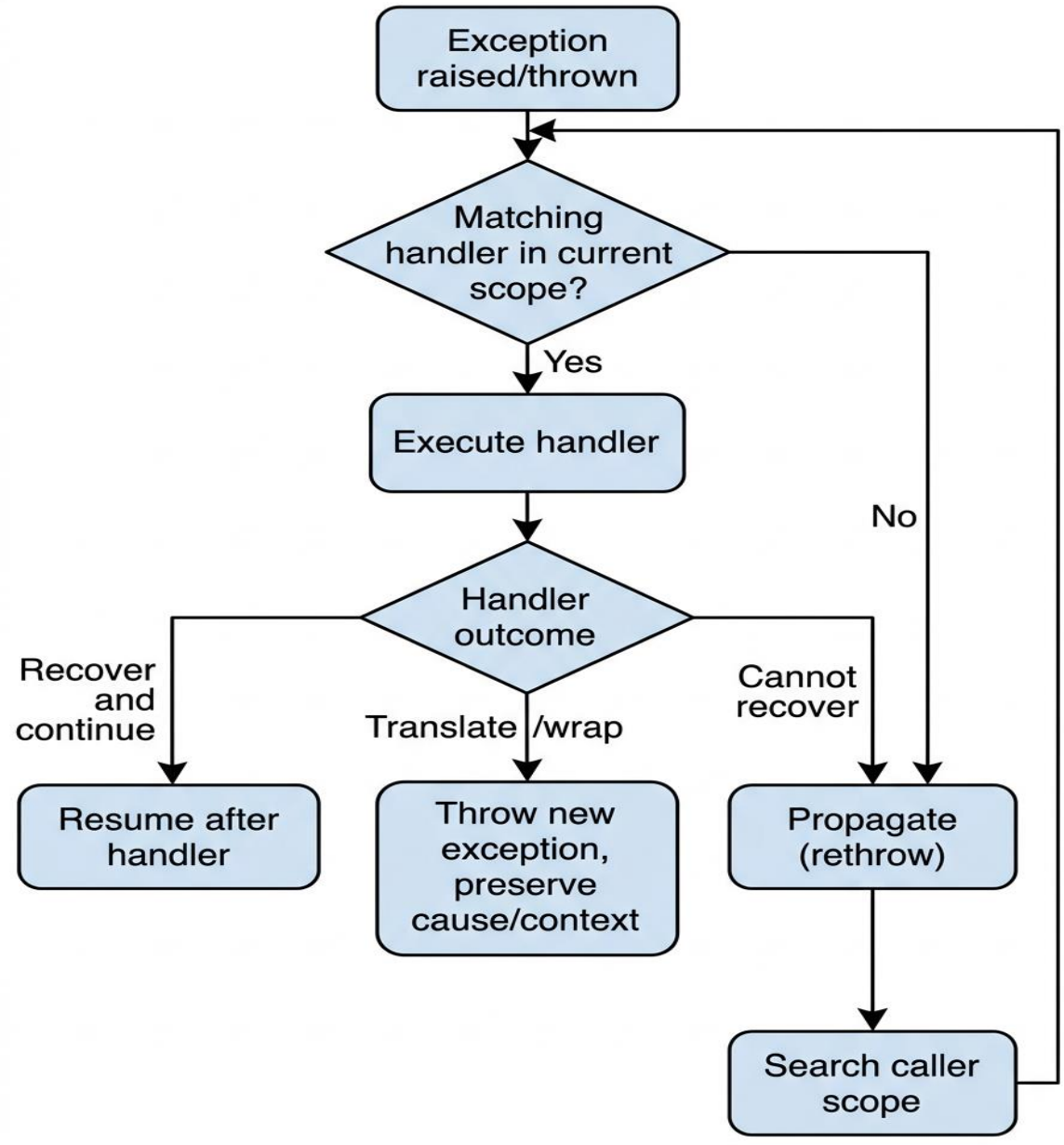

Figure 1 — Exception handling propagation flow

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

The flow above corresponds to the language-documentation descriptions of propagation and re-raising behaviour (e.g., Python's re-raise after finally, Java's handler search rules, and .NET's try/catch semantics). [35]

Java

Exception model and propagation visibility. Java's compile-time requirement for checked exceptions ("catch or declare") increases visibility of certain failure modes in method signatures. The JLS explicitly frames this as a design intended to reduce unhandled exceptions, while Oracle's tutorial explains checked exceptions as conditions a well-written programme should anticipate and recover from. [14]

Maintainability and anti-pattern exposure. Longitudinal evidence indicates that policy absence and insufficient guidance lead developers to replicate and spread anti-patterns through new features, thereby increasing technical debt for exception handling over time. [4] Multi-language mining reinforces that anti-patterns are common and that Java may show higher anti-pattern counts than Python in mixed-language repositories, especially for swallowing and destructive wrapping. [3]

Anti-pattern and corrected version (Java).

```
// Anti-pattern: swallow exceptions with a broad catch
try {
   processOrder(order);
} catch (Exception e) {
   // do nothing (exception swallowing)
}

// Corrected: catch narrowly, preserve context, propagate if unrecoverable
try {
   processOrder(order);
} catch (IOException e) {
   logger.error("Order processing failed (I/O)", e);
   throw e;
}
```

Swallowing and destructive patterns are repeatedly observed in mining and evolution studies and are treated as quality risks rather than "quick fixes." [36]

Testing exceptional behaviour (Java evidence-based). Large-scale mining of tests shows that exceptional tests are common but remain a minority; they are larger on average and often use try/catch. [6] Coverage and mutation studies show that catch/throw code is less covered than overall code, yet mature test suites can still detect many injected exception-handling faults. [7]

Python

Exception model and propagation behaviour. Python documentation specifies that exceptions not handled by an except clause are re-raised after finally executes; it also defines that exception handling is based on class/type matching and subclass relationships. [33]
Maintainability and debugging in layered ecosystems. Multilingual mining confirms that Python anti-patterns exist in real repositories, even though they are less frequent than in the analysed sample. [17] In Python ML ecosystems, stack traces can span applications and multiple libraries; questions containing them attract attention but are less likely to receive accepted answers, indicating enduring diagnostic difficulty in practice. [9]

Anti-pattern and corrected version (Python).

```
# Anti-pattern: broad except with silent failure
try:
   model.fit(data)
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

```
except Exception:
    pass

# Corrected: catch narrowly and preserve cause/context (re-raise after adding context)
try:
    model.fit(data)
except ValueError as e:
    raise ValueError("Training failed: invalid data format") from e
```

The corrected pattern retains causal context, a key debugging aid when traces span multiple layers. [37]

C#

Exception model and propagation behaviour. Microsoft's C# documentation defines the throw statement for raising exceptions and the try statement variants (try-catch, try-finally, try-catch-finally) for handling and cleanup. [13] .NET guidance emphasises correct handling and rethrowing to preserve diagnostic information, conforming to a convention- and tooling-based ecosystem rather than checked exceptions. [38]

Debugging and diagnostic fidelity (stack traces). CA2200 documents a concrete pitfall: rethrowing a caught exception by specifying the exception object restarts the stack trace at the current method, losing the call chain between the first throw site and the rethrow site. [39]

Anti-pattern and corrected version (C#).

```
//Anti-pattern: stack trace is restarted
try
{
DoWork();
}
catch (Exception ex)
{
Log(ex);
throw ex;
}
// Corrected: preserves original stack trace
try
{
DoWork();
}
catch (Exception ex)
{
Log(ex);
throw;
}
```

This is a direct, ecosystem-defined example of how handler design can harm debugging outcomes. [40]

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Table 2: Cross-language comparison matrix

Dimension	Java	Python	C#
Exception model	Checked + unchecked; compile-time checking for checked exceptions [14]	Unchecked; class/type-based matching; subclass rules [41]	Unchecked; structured statements; ecosystem guidance [42]
Propagation visibility	Higher for checked exceptions via signatures and compile-time enforcement [1]	Lower (no declarations); relies on docs and conventions [33]	Lower (no checked exceptions); relies on conventions and best-practice guidance [38]
Handler design risks	Policy absence can drive replication/spread of anti-patterns; broad catches contribute to "catch generic" style issues [43]	Broad except Exception enables swallowing; chaining helps preserve context if used [37]	Broad catch (Exception) enables over-catching; incorrect rethrow can destroy trace fidelity [44]
Maintainability signals	Robustness smells in exceptional code correlate with maintainability smells / design problems [5]	Anti-patterns present in multi-language repos; layering can amplify comprehension costs [45]	Empirical maintainability coverage is thinner in verified set; tool guidance targets diagnostic correctness [40]
Testing evidence	Exceptional tests common but minority; exceptional tests larger; catch/throw often under-covered [16]	Limited large-scale exceptional testing measurement in verified set; trace mining highlights failure scenarios [46]	Limited large-scale exceptional testing measurement in verified set; best practices emphasise disciplined handling/rethrow [47]
Robustness in real systems	Exception misuse can cause severe outcomes in complex systems (e.g., cloud eBugs) [20]	Stack traces may cross the entire software stack in ML systems; some categories hard to resolve [9]	Large-scale exception debugging is evident in search-log behaviour including .NET contexts [8]

DISCUSSION

The evidence supports three practical points about exception handling across Java, Python, and C#.

First, strictness vs flexibility is a real trade-off. Java's checked exceptions increase API-level visibility, but longitudinal evidence shows that without explicit policy, developers replicate existing patterns and anti-patterns spread, undermining the intent of compile-time checking. [49] Python and C# shift responsibility to conventions and tooling; this reduces boilerplate but makes it easier to hide failures (swallowing) or degrade diagnostic fidelity (bad rethrow) if guidelines are not enforced. [50]

Second, maintainability and design quality are strongly implicated. Robustness-change analysis shows that low-quality catch-block changes correlate with maintainability smells and design-problem indicators, indicating that exception-handling decisions can signal more extensive structural issues. [5] This supports a development practice implication: treat exceptional code as part of core design, not as a low-priority patch area.

Third, testing and tooling are the most actionable levers. Large-scale Java evidence shows exceptional tests exist but are often a small fraction of the suite and are more verbose; library studies show catch/throw logic is under-covered relative to overall code, even though mutation results show many suites can still detect injected EH faults if they test exceptional paths intentionally. [51] Tooling helps in two complementary ways: cross-language anti-pattern detection (Exception Miner) and ecosystem-specific rules that protect diagnostic information (e.g., CA2200 in .NET). [52] For wide-ranging systems, diagnostic difficulty is compounded by silent exception handling and separation between root cause and failure manifestation (ExChain), motivating disciplined logging and state-aware analysis tooling. [21]

A research gap remains: within the verified 2019–2024 set, Java has the strongest testing and evolution coverage, whereas comparable large-scale measurements for Python and C# are less common. Cross-language tools like Exception Miner help reduce this gap, but there stays a need for matched-corpus studies that measure handler structure, exceptional-path coverage, and defect/incident outcomes consistently across languages. [22]

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

CONCLUSION

Exception handling is universally available in Java, Python, and C#, but ecosystem differences strongly influence maintainability, debugging, testing and robustness. Java's checked exceptions increase compile-time visibility for some failure modes, yet empirical evidence shows that poor practices can still spread without explicit policies and guidance. [53] Python and C# reduce compile-time obligations, improving concision, but heightening reliance on developer discipline and tooling; in C#, rethrowing mistakes can directly erase diagnostic context. [44]

Across ecosystems, the evidence-backed interventions are consistent: define and disseminate exception-handling policy; avoid swallowing and destructive wrapping; preserve causal context when translating exceptions; test exceptional paths deliberately; and use automated recognition and ecosystem rules to prevent high-risk patterns from entering the codebase. [54]

REFERENCES

[1] Hassan, F., et al. "An Empirical Study of Software Exceptions in the Field Using Search Logs." ESEM 2020; arXiv:2006.00385. [26]
[2] de Sousa, D. B. C., et al. "Studying the evolution of exception handling anti-patterns in a long-lived large-scale project." Journal of the Brazilian Computer Society, 2020. doi:10.1186/s13173-019-0095-5. [4]
[3] Lima, L. P., et al. "Assessing Exception Handling Testing Practices in Open-Source Libraries." Empirical Software Engineering, 2021. doi:10.1007/s10664-021-09983-3; arXiv:2105.00500. [28]
[4] Ghadesi, A., Lamothe, M., Li, H. "What Causes Exceptions in Machine Learning Applications? Mining Machine Learning-Related Stack Traces on Stack Overflow." arXiv:2304.12857, 2023. [9]
[5] Ebert, F., Castor, F., Serebrenik, A. "A Reflection on 'An Exploratory Study on Exception Handling Bugs in Java Programs'." SANER 2020. doi:10.1109/SANER48275.2020.9054791. [29]
[6] Marcilio, D., Furia, C. A. "How Java Programmers Test Exceptional Behaviour." MSR 2021. doi:10.1109/MSR52588.2021.00033. [55]
[7] Souza, J., et al. "Exception Miner: Multi-language Static Analysis Tool to Identify Exception Handling Anti-Patterns." SBES 2024. doi:10.5753/sbes.2024.3573. [31]
[8] Oliveira, A. et al. "Don't Forget the Exception! Considering Robustness Changes to Identify Design Problems." MSR 2023. doi:10.1109/MSR59073.2023.00064. [18]
[9] Melo, H., Coelho, R., Treude, C. "Unveiling Exception Handling Guidelines Adopted by Java Developers." SANER 2019; arXiv:1901.08718. [32]
[10] Chen, H., et al. "Understanding Exception-Related Bugs in Large-Scale Cloud Systems." ASE 2019. doi:10.1109/ASE.2019.00040. [20]
[11] Dalton, F. et al. "Is Exceptional Behaviour Testing an Exception? An Empirical Assessment Using Java Automated Tests." EASE 2020. doi:10.1145/3383219.3383237. [19]
[12] Li, A., et al. "ExChain: Exception Dependency Analysis for Root Cause Diagnosis." NSDI 2024 (USENIX). [21]
[13] Oracle. "The Catch or Specify Requirement." Java Tutorials (official documentation). [56]
[14] Oracle. Java Language Specification, Chapter 11 "Exceptions." (official specification). [57]
[15] Python Software Foundation. "Errors and Exceptions." Python 3 Tutorial (official documentation). [58]
[16] Python Software Foundation. "Built-in Exceptions." Python 3 Library Reference (official documentation). [59]
[17] Microsoft. "Exception-handling statements — throw and try, catch, finally." C# language reference (official documentation). [13]
[18] Microsoft. "Best practices for exceptions." .NET documentation (official guidance). [60]
[19] Microsoft. "CA2200: Rethrow to preserve stack details." (official analyser rule). [39]
[20] de Pádua, G. B., Shang, W. "Studying the relationship between exception handling practices and post-release defects." MSR 2018. doi:10.1145/3196398.3196435. (Background; outside 2019–2024).[23]

ASSUMPTIONS / UNSPECIFIED DETAILS

• de Pádua & Shang (2018) is included only as background because it remains a useful Java–C# empirical bridge, but it is outside the requested 2019–2024 window. [61]
• IJERT layout requirements (single column, Times New Roman 12pt, 1.5 line spacing) cannot be enforced in plain text; the manuscript is structured to match IJERT section conventions and can be transferred into an IJERT Word/LaTeX template without altering content.
• Image links are provided in code blocks to comply with the "URLs in code" constraint; the de Sousa et al. figure link is verified via Springer's figure page and the resolved PNG. [62]

[1] [56] The Catch or Specify Requirement (The Java™ Tutorials > …
https://docs.oracle.com/javase/tutorial/essential/exceptions/catchOrDeclare.html
[2] [12] [15] [33] [35] [37] [50] [58] 8. Errors and Exceptions
https://docs.python.org/3/tutorial/errors.html
[3] [11] [17] [22] [25] [36] [45] [52] [54] Exception Miner: Multi-language Static Analysis Tool to …
https://leopoldomt.github.io/assets/pdf/2024-sbes.pdf
[4] [27] [43] [49] Studying the evolution of exception handling anti-patterns in a long-lived large-scale project | Journal of the Brazilian Computer Society | Springer Nature Link
https://link.springer.com/article/10.1186/s13173-019-0095-5
[5] [18] Considering Robustness Changes to Identify Design …
https://research.jku.at/en/publications/dont-forget-the-exception-considering-robustness-changes-to-ident/

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

[6] [16] [30] [51] [55] How Java Programmers Test Exceptional Behavior
https://dvmarcilio.github.io/papers/msr2021.pdf
[7] Assessing Exception Handling Testing Practices in Open-Source Libraries
https://arxiv.org/abs/2105.00500
[8] [2006.00385] An Empirical Study of Software Exceptions in the Field …
https://ar5iv.labs.arxiv.org/html/2006.00385
[9] [46] What Causes Exceptions in Machine Learning Applications? Mining Machine Learning-Related Stack Traces on Stack Overflow
https://arxiv.org/abs/2304.12857
[10] [39] [40] [44] CA2200: Rethrow to preserve stack details (code analysis)
https://learn.microsoft.com/en-us/dotnet/fundamentals/code-analysis/quality-rules/ca2200
[13] [34] [42] Exception-handling statements - throw and try, catch, finally
https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/exception-handling-statements
[14] [24] [53] [57] Chapter 11. Exceptions
https://docs.oracle.com/javase/specs/jls/se21/html/jls-11.html
[19] Is Exceptional Behavior Testing an Exception?
https://dl.acm.org/doi/10.1145/3383219.3383237
[20] Understanding Exception-Related Bugs in Large-Scale Cloud …
https://dl.acm.org/doi/pdf/10.1109/ASE.2019.00040
[21] Exception Dependency Analysis for Root Cause Diagnosis
https://www.usenix.org/system/files/nsdi24-li-ao.pdf
[23] [61] Studying the relationship between exception handling …
https://dl.acm.org/doi/10.1145/3196398.3196435
[26] An Empirical Study of Software Exceptions in the Field …
https://arxiv.org/abs/2006.00385
[28] Assessing exception handling testing practices in open- …
https://dl.acm.org/doi/abs/10.1007/s10664-021-09983-3
[29] A Reflection on “An Exploratory Study on Exception Handling …
https://aserebre.win.tue.nl/SANER2020Felipe.pdf
[31] Exception Miner: Multi-language Static Analysis Tool to …
https://sol.sbc.org.br/index.php/sbes/article/view/30420
[32] Unveiling Exception Handling Guidelines Adopted by Java …
https://arxiv.org/abs/1901.08718
[38] [47] [60] Best practices for exceptions - .NET
https://learn.microsoft.com/en-us/dotnet/standard/exceptions/best-practices-for-exceptions
[41] [59] Built-in Exceptions
https://docs.python.org/3/library/exceptions.html
[48] media.springernature.com
https://media.springernature.com/full/springer-static/image/art%3A10.1186%2Fs13173-019-0095-5/MediaObjects/13173_2019_95_Fig5_HTML.png
[62] Figure 5 | Studying the evolution of exception handling anti-patterns in a long-lived large-scale project | Journal of the Brazilian Computer Society | Springer Nature Link
https://link.springer.com/article/10.1186/s13173-019-0095-5/figures/5