

Survey of noexcept Policy Papers

Analysis and comparison of SD-9 noexcept proposals according to design principles

Document #:	P4403R0
Date	2026-10-20
Project	ISO/IEC JTC1/SC22/WG21 — Programming Language C++
Audience	Library Evolution Working Group
Reply-to	Inbal Levi (author) <sinbal2l@gmail.com> John Lakos <jslakos@gmail.com> Timur Doumler <papers@timur.audio> Ben Craig <ben.craig@gmail.com> Robert Leahy <rleahy@rleahy.ca> Khalil Estell <khalil.estell@sjsu.edu>

Table of Contents

0. Revision History	1
1. Introduction	2
2. Essence of Suggested Solutions	2
3. Design Principles	2
4. Comparison by Design Principle	3
5. Current-Draft Example Cases	4
6. References	4

0. Revision History

1.1. R0

- Initial paper, containing four sections: essence, design principles, comparison, representative cases.

1. Introduction

The paper "P2267R1: Library Evolution Policies" was created to define a framework for collecting and documenting widely agreed principles in the C++ standard library, defined as "policies" for the SD-9 document.

Policies aims to highlight and document widely known best practices, in order to minimize and avoid repeating discussions on topics which we can find common agreement on, and to be used to guideline for authors of future proposals to the Library Evolution Work Group.

Multiple papers published, suggesting SD-9 policies in regards to `noexcept` for the standard library. There also been work on comparing some of them, using principled design (e.g.: [P3085R3](#) (12. Appendix)).

This paper's goal is to collect and compare all proposals, and create an informative, impartial summary of the proposals. The comparison is aiming to be descriptive only; not to advocate for any preferred solution.

The paper has four parts: First, it describes the essence of the suggestion in each proposal. Next, it aims to identify canonical design principles. Next, it tries to evaluate the weight each proposal is giving to the principle. Last, to provide more concrete examples, we collect representative current-draft cases for each proposal (needless to say – these are just examples, and many more existing in the draft).

Disclaimer: This paper is an impartial survey; it does not advocate for selecting any preferred paper or policy. The paper was revised in advance by authors of the papers surveyed, to make sure representation is true to the source. The deduction and conclusions are left for the readers.

2. Essence of Suggested Solutions

The definitions were taken from the SD-9 wording in the papers and reviewed by authors of the proposals.

Paper	Definition of proposed option
P3155R1	If a library function has a wide contract (i.e., does not specify undefined behavior due to a precondition), it should be marked as unconditionally <code>noexcept</code> if it cannot throw; if it has a narrow contract, it should be specified as <code>Throws: Nothing</code> if it cannot throw when called in-contract.
P3085R3	Each library function that will not throw when called with all preconditions satisfied should be marked as unconditionally <code>noexcept</code> .
P4254R0	Apply <code>noexcept</code> to non-throwing library operations, including the <code>std::execution</code> interface, where non-throwing behavior is part of the specified operation.
P4257R0	Each library function that LEWG agree cannot throw, should be marked as unconditionally <code>noexcept</code> .

3. Design Principles

The following canonical "Design Principles" were identified from the proposals. However, these are not describing the proposals in full and are not covering the full rational and benefits advocated by the papers but aim to capture the differences/decision points. For full content of each proposal, please read the original papers (in references).

Principle	Definition	Principle	Definition
Unified rule	Similar interfaces follow a predictable library-wide rule (familiarity with other library areas allows deducing behavior)	Interface usability	The exception specification makes the interface easier to understand and use.
Information propagation in generic code	Generic code can observe and propagate non-throwing properties.	Performance / optimization opportunity	Known non-throwing behavior can support implementation or generic-code optimization.
Implementation flexibility	Retain freedom in diagnostics, hardening, testing, and implementation technics.	Precondition-diagnostic flexibility	Retain flexibility in diagnosing checked violated preconditions.
Composability	Interface composes well with user operations, CP's, and other library operations.	Specification clarity	Valid vs. exceptional vs. out-of-domain behavior remain distinct.
Evolution flexibility	Future evolution is not unnecessarily constrained by an existing exception specification.	Observable API guarantees	Users can observe/assume non-throwing behavior as an interface property.

4. Comparison by Design Principle

The following table aims to evaluate how each principle is reflected in each proposal. We see value in putting this together as a basis of understanding and as a starting point for the discussion. However, despite best effort to achieve impartiality, evaluations are not numerical, and there may be differences in perception of how strongly a principle is reflected in the proposal according to each reader. Comments and fixes are welcome.

We encourage readers to engage in critical reading and form their own estimations and conclusions.

Principle	P3155R1	P3085R3	P4254R0	P4257R0
Unified rule	Strong	Trade-offs	Trade-offs	Trade-offs
Interface usability	Trade-offs	Strong	Strong	Trade-offs
Information propagation in generic code	Lower	Strong	Strong	Trade-offs
Performance / optimization opportunity	Lower	Strong	Strong	Trade-offs
Implementation flexibility	Strong	Strong	Lower	Trade-offs
Precondition-diagnostic flexibility	Strong	Strong	Lower	Trade-offs
Composability	Trade-offs	Strong	Strong	Trade-offs
Specification clarity	Strong	Trade-offs	Strong	Strong
Evolution flexibility	Strong	Strong	Trade-offs	Strong
Observable API guarantees	Lower	Strong when selected	Strong	Strong

Legend:

Strong = strongly prioritized; Trade-offs = supported subject to classification or competing goals; Lower = comparatively low emphasis or not primary.

5. Current-Draft Example Cases

Representative current-draft cases for each proposal, many more exist in the draft.

Paper	Representative case	Details on case
P3155R1	Unchecked accessor with a precondition	Valid calls need not throw; omitting <code>noexcept</code> preserves diagnostic flexibility for an invalid call.
P3085R3	Generic operation with dependent exception behavior	Conditional <code>noexcept</code> propagates the exception behavior of an underlying user-provided operation.
P4254R0	<code>std::execution</code> protocol operations	Completion and operation-state protocols use non-throwing requirements as observable protocol properties.
P4257R0	Operation with an explicit precondition	Valid execution and the handling of a contract or precondition violation can be classified separately.

6. References

- Papers surveyed (by date of first publication):
 - [P3155](#): `noexcept` policy for SD-9 (The Lakos Rule)
 - [P3085](#): `noexcept` policy for SD-9 (throws nothing)
 - [P4254](#): Throwing Violation Handlers Are Post Hoc Library Design
 - [P4257](#): Marking wide contract non-throwing functions `noexcept` is not controversial
- [P3313](#): Impacts of `noexcept` on ARM table based exception metadata (Informative)
- Current C++ Working Draft: <https://eel.is/c++draft/>