Doc. No. P4334R0
Date: 2026-08-09
Audience: EWG, SG12, SG20, SG23
Authors: Bjarne Stroustrup, J-Daniel Garcia,
Vinnie Falco, John Spicer, Ville Voutilainen
Reply to: bjarne@stroustrup.com

P2900 Contracts' fundamental flaws

C++ has *always* aimed at increasing what can be directly expressed in code and the guarantees offered. This goes all the way back to specifying function argument types in function declarations (missing in K&R C) and offering ways of guaranteeing initialization and cleanup of objects of user-defined types. The zero-overhead principle was introduced to ensure that overheads were minimal and present only when requested.

Contracts depart from that by focusing on debug support. That's not inherently bad. We have all used various forms of assertions to specify run-time tests (usually optional) and compile-time properties we wanted guaranteed. However, such systems are ad-hoc in the sense that they rely on programmers applying them correctly and consistently throughout a program.

The focus on contracts is programmer-provided run-time checks. Where this approach has been tried (Eiffel, C#, macro-based systems, etc.), the cost of that has led to a strategy of enabling contracts while testing and removing them (or at least most) in production code. This is a 1980s approach that ignores about 50 years of progress involving a combination of static analysis and improved implementation of range checks. Furthermore, some guarantees that we want, are requested by the wider community, or required by regulatory bodies are not suitable for individual checks. They require rules applied to a whole translation unit, or even a whole program.

The P2900 contracts and extensions aim to change the nature of C++ and are an existential threat to C++, adding complexity and bloat (~56 pages in the draft standard for an incomplete version), potential overheads, and novel opportunities for errors without addressing the current community and regulatory body demands for guarantees (e.g., https://docbox.etsi.org/CYBER/CYBER/Open/ETSI_TS_104_198_DRAFT.zip presents C++ as providing no memory safety guarantees).

We are not fundamentally against idea of contracts. What we are against is the bloated and incomplete P2900 contracts that seem to be endlessly extended and changed by ad-hoc patches (P3097, P3099, P3100, P3290, P3400, P3595, P3850, P4262, P4283, P4298, and more). We would welcome a contracts design that is simple, and "marketed" as just a debug aid, rather

than a safety feature. A far simpler and more powerful library-based contracts design exists. It is implemented, compatible, and tested. See https://isocpp.org/files/papers/D4324R0.html.

We are not against C++26 beyond contracts and their many promised extensions. However, we are against accepting a feature that will cause long-term damage simply to avoid delay official acceptance of good features, that in fact are already well-specified and available.

The P2900 contracts are untried, incomplete, unimplemented, and constantly proposed extended – extending what's untried with more untried material. That alone makes it completely unsuitable for an international standard. See P3573R0 where several experienced WG21 members (incl. designers, compiler implementers, library implementers, and educators) summarize the objections and were basically ignored (P3846R0). None of those objections were resolved. The current objections can be summarized. The P2900 contracts are:

- Unimplemented
- Incomplete
- Untried at scale [P3460R0, P3506R0]
- Not tried in major application domains
- Violates foundational principles of C++
- Violates fundamental principles of language design
- Hasn't been tried in major libraries (e.g., the C++ standards library [P3506R0, P3878R0])
- Isn't integrated with or appropriate for hardened libraries [P3878R0]
- Doesn't offer safety guarantees [P3573R0, P3362R0]
- Includes a completely untried inheritance model
- Offer new ways of making errors through inconsistent application in TUs
- Leads to new forms of UB, detrimental to safety and security
- Narrows the choices of error handling
- Doesn't protect against logical errors, misuses, and incoherent uses
- Hasn't been used to support static analysis
- Hasn't been demonstrated to be easily teachable [P3261R0, P3281R0]

How could such a bloated and incomplete design be voted into a draft standard? By delaying key decisions, by making many features implementation-defined, and promising future improvements, thus making many believe that they eventually will get what they desire.

See also

- John Spicer: "Contracts are inappropriate for undefined behavior checks" (https://isocpp.org/files/papers/P4332R0.html).

- C++ Alliance: "C++26 Needs To Go Back" (https://cppalliance.org/pdf/alliance-nb-petition.pdf)
- Vinnie Falco et al: "Returning C++26 for the Evaluation It Skipped" (https://isocpp.org/files/papers/P4238R0.pdf).

Appendix: Documentation

Contracts failing: The record bears this out. Eiffel's own documentation states that "when releasing the final version of a system, it is usually appropriate to turn off assertion monitoring, or bring it down to the required level"; C's assert is compiled out under NDEBUG; and Microsoft's C# Code Contracts, built on the same enable-in-test / disable-in-release model, have been abandoned. P2900R14 stands squarely in this line: a build-time "ignore" semantic removes the checks exactly where the code runs for real — keeping the life jackets on only while near the coast. The modern answer runs the other way: always-on hardened standard libraries, whose production overhead was measured across Google's server fleet at 0.3% (Dionne et al., ACM Queue, 2025), together with static analysis enforced through the C++ Core Guidelines.

Proposal churn: The scale of that churn is on the record: P2900R14 carries roughly 56 pages of normative wording inside a 119-page paper, and the set of evaluation semantics was still growing during review — quick_enforce was added in revision 7, taking three semantics to four. The pieces ordinary use needs are not in the feature but deferred to C++29-targeted extensions still being designed: contracts on virtual functions (P3097R3), capture of an object's original value for a postcondition (P3098R2), and a framework for core-language undefined behavior (P3100R7).

Serious objections ignored: Those members were not junior: P3573R0's nine authors include compiler and library implementers and language designers (Hava, García, Regev, Dos Reis, Spicer, Stroustrup, van Winkel, Vandevoorde, Voutilainen). They documented the review conditions themselves — "10+ papers suggesting changes, 1,000+ reflector messages, and dozens of changes to P2900 in 2024" — and concluded that merely keeping up was "a full-time job." P3846R1 engaged the objections, but for the contested capabilities the recorded answer was deferral to work that does not yet exist.

The P2900 contracts failings:

- Unimplemented — GCC 16.1 (released 2026-04-30) ships it only as experimental; Clang's C++ status page lists P2900R14 as "No"; Microsoft (P3506R0) and EDG have objected to standardizing it.

- Incomplete — ~56 pp of normative wording, and the pieces central to everyday use — virtual functions (P3097R3), old-value capture (P3098R2), and the UB framework (P3100R7) — are all deferred to C++29.
- Untried at scale — the one library experiment was a fork limited to contract_assert, with no new pre/post added, and was not deployment at scale (P3460R0).
- Not tried in major application domains — no shipping production codebase uses pre/post/contract_assert.
- Violates foundational principles of C++ — two language defaults cost in ordinary code: constification (P3261R0 — pre(isGood(mech)) fails with “no overload found”) and converting an escaping exception into a contract violation, which forces exception-handling scaffolding around every assertion, against zero-overhead.
- Violates fundamental principles of language design — cf. The Design and Evolution of C++: “Every feature must have a reasonably obvious implementation” and “Always provide a transition path.”
- Hasn’t been tried in major libraries (e.g., the C++ standards library) — P2900R14 proposes no change to the standard library; libc++, libstdc++, and the Microsoft STL all harden with their own assertion macros, not pre/post/contract_assert.
- Isn’t integrated with hardened libraries — the hardened libraries that do ship keep low-cost checks on in production at ~0.3% overhead; contracts are not wired into them (Dionne et al., 2025).
- Doesn’t offer safety guarantees — P3573R0: “Safety must involve guarantees over a code base or parts thereof. Contracts primarily offer checking of correctness where it is used.”
- Includes a completely untried inheritance model — the model was adopted into P2900R13 by EWG and reviewed by CWG, then removed before R14 for lack of consensus (P2899R1); the drafted mechanism (P3097R3) was described as “almost completely untried.”
- Offer new ways of making errors through inconsistent application in TUs — in mixed mode, an inline function in a header compiled with checks disabled in one TU means “the contract violation will not be detected” (P3835R0, Spicer, Voutilainen, García).
- Leads to new forms of UB, detrimental to safety and security – 3835R0 (Spicer, Voutilainen, García, “Contracts make C++ less safe — full stop!”).

- Narrows the choices of error handling — a predicate's escaping exception can no longer be caught as it would be from the function body; it is converted to a violation (P4308R0).
- Doesn't protect against logical errors, misuses, and incoherent uses — a contract checks only what a programmer remembered to write, where they wrote it.
- Hasn't been used to support static analysis — the whole-codebase guarantees national bodies ask for are delivered by construction (Core Guidelines enforcement, hardening), not by programmer-written run-time checks.
- Hasn't been demonstrated to be easily teachable — learners must absorb four build-time semantics and the constification rule under which a predicate can mean something different from the identical expression in the body (P3261R0).

The mechanics are on public record. The design was pursued as a deliberately minimal MVP, which let hard questions — virtual functions, old-value capture — be set aside as out of scope rather than answered (P2899R1). Consensus was then assessed by counting votes: at Hagenberg the poll to remove contracts from C++26 was recorded "Consensus against" at SF:9 F:8 N:3 A:19 SA:41, and two compiler vendors were among the objectors (P4020R0). A numeric rule does not weigh the implementation responsibility behind a sustained objection, and a foundational feature is where that weight has mattered most. Finally, the contested capabilities were answered by deferral — for example guaranteed enforcement and in-source selection to P3400R1, which is "not yet ready to adopt and has not yet reached consensus, in any WG21 subgroup." An objection answered by a promise of future work is answered only when that work arrives.

The cost of getting this wrong is not symmetric. The C++26 draft has not yet had its final ballot: a default changed now changes text that no implementation relies on, while the same default changed after C++26 ships changes what a decade of code will have been written against, and the design freedoms a shipped default removes do not return on the next revision's schedule. On the evidence available today the design is not ready — and what would make it ready is no mystery: an implementation deployed at scale, use in a real standard library, a settled treatment of the parts now deferred, and defaults that match the aims C++ has always had. A simpler, fully implemented, tested alternative already exists (P4324R0). We could save time and work by adopting that path rather than ship a foundational feature that is not finished.

References

- P2900R14 — Contracts for C++ (Berne, Doumler, Krzemieński, 2025).

- P2899R1 — Contracts for C++ — Rationale (Berne, Doumler, Khlebnikov, Krzemieński, 2025).
- P3097R3 / P3098R2 / P3100R7 — deferred contracts extensions (virtual functions / postcondition captures / UB framework).
- P3261R0 — Revisiting const-ification in Contract Assertions (Berne, 2024).
- P3460R0 — C++ Contracts: Implementers Report (Fiselier, Ranns, Sandoe, 2024).
- P3506R0 — P2900 Is Still Not Ready for C++26 (Dos Reis, 2025).
- P3573R0 — Contract concerns (Hava, García, Regev, Dos Reis, Spicer, Stroustrup, van Winkel, Vandevoorde, Voutilainen, 2025).
- P3616R0 - Remove contracts on virtual functions for now (Voutilainen, 2025).
- P3835R0 — Contracts make C++ less safe — full stop! (Spicer, Voutilainen, García, 2025).
- P3846R1 — C++26 Contract Assertions, Reasserted (Doumler, Berne, et al., 2025).
- P3878R1 — C++26 Contracts are not a good fit for standard library hardening" (Voutilainen, Wakely, Spicer, Lavavej).
- P4020R0 — Concerns about contract assertions (Krzemieński, 2026).
- P4308R0 — Eight Responses to a Throwing Implicit Contract Assertion (Falco, Voutilainen, 2026).
- P4324R0 — C++ Design By Contract: Minimum Language, Maximum Library (Falco, Voutilainen, 2026).
- Dionne, Rebert, Shavrick, Varlamov — Practical Security in Production (ACM Queue, 2025).
- C++ Core Guidelines (Stroustrup and Sutter, eds.).