NB-Commenting is Not a Vehicle for Redesigning inplace_vector

Document number: P3830R0
Date: 2025-09-04
Project: Programming Language C++, Library Evolution Working Group
Reply-to: Nevin “☺” Liber, nliber@anl.gov

Table of Contents

Introduction

inplace_vector is not broken. Ever since the design of inplace_vector was approved in 2023 in Varna, there have been many attempts to redesign it. It appears there are last minute attempts to keep doing so, this time under the guise of “fixing” it via NB-comments.

Motivation and Scope

inplace_vector is not broken.

The return type of inplace_vector::try_push_back(x) is not broken.

The return type of inplace_vector::try_emplace_back(…) is not broken.

The design of inplace_vector was approved in the last LEWG session in Varna. Since then, there have been various attempts to redesign it in the small and in the large, none of which have passed LEWG. We understand that not everyone is 100% happy with the final design of inplace_vector, but that is true about almost anything that needs to be approved via consensus by committee.

Allocators

We debated allocator support multiple times.

Library Evolution Telecon 2024-01-30

LWG wanted LEWG to re-litigate the exception type thrown when adding an element to a full inplace_vector as well as allocator support. Allocator discussions were essentially delayed until a paper describing it was available from its proponents. For completeness, here is the other poll:

POLL: We want to revisit the status quo in the paper: “Inplace_vector throw a “bad_alloc” when exceeding max_size”.

SF	F	N	A	SA
3	4	2	5	4

Attendance: 18

Authors’ position: SAx2

Outcome: No consensus for a change

2024 Tokyo

P3160R0: An Allocator-aware inplace_vector

POLL: We should promise more committee time to pursuing "An Allocator-aware inplace_vector", knowing that our time is scarce and this will leave less time for other work.

SF	WF	N	WA	SA
6	6	4	5	6

Attendance: 25+9

# of Authors: 1

Author Position: SF

Outcome: No consensus to pursue

2025 Hagenberg (which, according to P1000R6, "C++26 design is feature-complete")
P3160R2 An allocator-aware inplace_vector

POLL: Knowing our time is scarce we would like to pursue allocator support for inplace_vector for C++26
(during this meeting)

SF	F	N	A	SA
3	5	1	9	11

Attendance: 32 (IP) + 10 (R)
Author's Position: SF
Outcome: Consensus against.

Let me repeat that: *Consensus against*. Still, one of the advocates for allocator support is now trying to get that redesign in under the façade of an NB-comment. But that NB-comment has *no new information*. inplace_vector may not meet their exact needs, but that isn't a sufficient reason to re-litigate this a third time, nor is it a reason to litigate this as a "bug fix". Plus, at this late date such a major design change would more likely get inplace_vector pulled from C++26 than such a drastic redesign approved in time by LEWG and LWG.

Comparisons
Library Evolution Telecon 2025-06-03 (post-Hagenberg)
P3698R0 Cross-capacity comparisons for inplace_vector

While some of the inplace_vector authors thought this might be fine, after deeper ruminations it was discovered that it violated one of the original design principles for inplace_vector: that of regularity (comparability and constructability go hand in hand). There were also concerns about the risk in making these changes for C++26 so late in the cycle. While there was weak consensus in favor of this change, no updated paper appeared in Sofia, and it is unknown if this redesign will be attempted by NB-comment, or if its authors will wait until the C++29 cycle.

In summary: initially this change was superficially acceptable, but after deeper thought brought up more reservations. It may go through for C++29, but we need the time to think it through. Last minute design changes are risky.

optional<T&>

The return types for try_push_back(x) and try_emplace_back(…) are *not* broken.

History: In P0843R6 static_vector, they were initially specified to return an optional<T> (not an optional<T&>). This ended up being undesirable. I was added as an author on P0843R7 inplace_vector. We authors internally debated whether to return a bool or a pointer, and I convinced the other authors that returning a pointer was better because

1. It supported the same use case as bool.
2. It provides more useful information than bool, by giving access to the newly constructed element.
3. It had no extra overhead (either compile time or run time) over a bool.

We then polled it in LEWG in Varna:

POLL: The signatures and semantics that D0843R7 provides for push_back, emplace_back, try_push_back,
try_emplace_back, and the unchecked versions are acceptable.

Strongly Favor	Weakly Favor	Neutral	Weakly Against	Strongly Against
9	7	0	0	0

Attendance: 23 (room) + 3 (remote)

# of Authors: 3

Author Position: 3sf

Outcome: Unamimous consensus.

P3739: Standard Library Hardening – using std::optional<T&>

This is a brand-new design, showing up under the pretense of being an NB-comment. Even ignoring the hijacking of the term "Hardening" (in the C++26 CD hardening refers to what happens with hardened preconditions under a hardened implementation), there are some novel things here, such as returning a const optional<T&>. Why const? There is no motivation. That makes it novel design, which is not something one wants to do at the last moment for a stable library.

The paper states that changing the return type of try_*_back(…) from T* to optional<T&> was discussed. While that may be true (the paper provides no information on when it was discussed, making it next to impossible to do the archeology to find the meeting notes of the discussion), it was never polled, nor mentioned in P0843 inplace_vector, nor mentioned in P2988 optional<T&>.

Because optional<T&> was adopted so late in the cycle (Sofia), we've already run into a number of issues with it: LWG4299, LWG4300 and LWG4308.

Moreso, one of my design principles behind using T* is that it is trivially copyable, which is important in my work. optional<T&> is not currently guaranteed to be trivially copyable. Did that fall through the cracks? Probably (given that a typical implementation is a wrapper around a pointer). Will it get fixed? Probably (I have filed an NB-comment on it). But how many more things will come up between now and the final ballot for C++26? It is hard to evaluate the applicability of using optional<T&> as it is still a moving target. And like comparisons, adopting this redesign requires much deeper thought.

In my experience, it is fairly rare to use to return value of try_*_back(…) in anything but a Boolean context, and we shouldn't be adding more overhead to what is likely to be an object briefly used as a temporary and thrown away.

Conclusion

inplace_vector is not broken.

The return type of inplace_vector::try_*_back(…) is not broken.

The time for redesign has passed.