Doc. no.: P0977r0
Date: 2018-03-6
Programming Language C++
Audience: All
Reply to: Bjarne Stroustrup (bs@ms.com)

Remember the Vasa!

Bjarne Stroustrup

Many/most people in WG21 are working independently towards non-shared goals. Individually, many (most?) proposals make sense. Together they are insanity to the point of endangering the future of C++.

Here is a selection of papers from the pre-Jacksonville mailing. I list papers that I think has the potential for significantly change the way we write code, so that each has significant implications on teaching, maintenance, and coding guidelines. Many also have implications for implementations:

1. *Gabriel Dos Reis: Modules TS
2. Herb Sutter: Concepts in-place syntax
3. H. Carter Edwards: Polymorphic Multidimensional Array Reference
4. H. Carter Edwards: Relaxed Incomplete Multidimensional Array Type Declaration
5. H. Carter Edwards: Span - foundation for the future
6. David S. Hollman: An Ontology for Properties of mdspan
7. *Neil MacIntosh: span: bounds-safe views for sequences of objects
8. Vicente J. Botet Escribá: C++ generic overload function (Revision 3)
9. Matúš Chochlík: Static reflection
10. Matúš Chochlík: Static reflection of functions
11. Jonathan Coe: A polymorphic value-type for C++
12. Matthias Kretz: Data-Parallel Vector Types & Operations
13. Vicente Botet: std::expected
14. Tom Honermann : char8_t: A type for UTF-8 characters and strings (Revision 1)
15. *J. D. Garcia: Support for contract based programming in C++
16. Victor Zverovich: Text Formatting
17. Vicente J. Botet Escribá: C++ Monadic interface
18. Axel Naumann: Parametric Functions
19. Bryce Adelstein Lelbach: I Back to the std2::future Part I
20. Herb Sutter : Metaclasses: Generative C++

21. Jeff Snyder: Class Types in Non-Type Template Parameters
22. Vicente J. Botet Escribá: ValuedOrError and ValueOrNone types
23. John McFarlane: Elastic Integers
24. Gašper Ažman: Deducing this
25. Corentin jabot: A plea for a consistent, terse and intuitive declaration syntax
26. *Bruno Cardoso Lopes*: A proposal for modular macros
27. Barry Revzin: Chaining Comparisons
28. Lee Howes: A strawman Future API
29. Tomasz Kamiński : Symmetry for spaceship
30. *Gor Nishanov: merging Coroutines TS [N4723] into the C++20 working draft
31. Vittorio Romeo: Concept-constrained auto
32. Titus Winters: LEWG wishlist for EWG
33. Nathan Sidwell: Modules:Dependent ADL
34. Nathan Sidwell: Modules:Unqualified Using Declarations
35. James Dennett: Towards A (Lazy) Forwarding Mechanism for C++
36. *Herb Sutter: A Modest Proposal: Fixing ADL
37. Richard Smith: Towards consistency between <=> and other comparison operators
38. Richard Smith: Another take on Modules
39. Peter Dimov: Adding support for type-based metaprogramming to the standard library
40. Matúš Chochlík: constexpr reflexpr
41. Mingxin Wang: PFA: A Generic, Extendable and Efficient Solution for Polymorphic Programming
42. Ville Voutilainen: Allow initializing aggregates from a parenthesized list of values
43. Richard Smith: P0936R0 Bind Returned/Initialized Objects to Lifetime of Parameters

To avoid this list looking longer and even scarier, I have listed only first authors.

For truth in advertising, I have marked proposals I support with *. I may support parts of other proposals here, but not all or exactly as written. There are also proposals that I'm working on not present here, such as operator dot, uniform function call, and **stack_array**, see Thoughts about C++17. Since I'm not bringing those forward in Jacksonville, they don't (yet) belong on this list.

Apologies if I forgot a paper with major implications or if I listed one without a potential significant impact. This list contains less than half of the proposals for extensions, changes, and improvements currently being discussed. They are merely the most prominent. Proposals related to concurrency are underrepresented.

Hardly any paper contains extensive discussions of the proposed feature's effect in combination with other new features, existing features, and libraries in "ordinary code" written by "ordinary programmers." Few present details of experience of use or teaching. Hardly any contain a serious discussion of objections raised. Every proposal is subject to the law of unexpected consequences: There will be unexpected consequences.

We are on a path to disaster though enthusiasm and design-by-committee (or rather "design-by-committees"). During the early days of WG21 the story of the Vasa was popular as warning against overelaboration (from 1992):

> Please also understand that there are dozens of reasonable extensions and changes being proposed. If every extension that is reasonably well-defined, clean and general, and would make life easier for a couple of hundred or couple of thousand C++ programmers were accepted, the language would more than double in size. We do not think this would be an advantage to the C++ community.
>
> We often remind ourselves of the good ship Vasa. It was to be the pride of the Swedish navy and was built to be the biggest and most beautiful battleship ever. Unfortunately, to accommodate enough statues and guns it underwent major redesigns and extension during construction. The result was that it only made it half way across Stockholm harbor before a gust of wind blew it over and it sank killing about 50 people. It has been raised and you can now see it in a museum in Stockholm. It is a beauty to behold - far more beautiful at the time than its unextended first design and far more beautiful today than if it had suffered the usual fate of a 17th century battle ship -- but that is no consolation to its designer, builders, and intended users.

Remember the Vasa!

There are people who concluded from the Vasa story that all incremental improvement is a bad strategy. However, if the Vasa had been sent to sea as originally designed, it could not have served its purpose. Being under-gunned, someone would have sent it to the bottom full of holes. Being somewhat ordinary, it would have failed in its representative (image) role. Recent research has shown that a relatively modest increase of the Vasa's length and breadth (claimed technically feasible) would have made it stable, so my reading of the Vasa story is: Work hard on a solid foundation, learn from experience, and don't scrimp on the testing.

The foundation begun in C++11 is not yet complete, and C++17 did little to make our foundation more solid, regular, and complete. Instead, it added significant surface complexity and increased the number of features people need to learn. C++ could crumble under the weight of these – mostly not quite fully-baked – proposals. We should not spend most our time creating increasingly complicated facilities for experts, such as ourselves.

We need a reasonably coherent language that can be used by "ordinary programmers" whose main concern is to ship great applications on time. We now have about 150 cooks; that's not a good way to get a tasty and balanced meal.

We are on the path to something that could destroy C++. We must get off that path!

See (not a feature proposals):

- Ville Voutilainen: To boldly suggest an overall plan for C++20
- B. Dawes, H. Hinnant, B. Stroustrup, D. Vandevoorde, M. Wong: Direction for ISO C++