“What will C++17 be?” B. Stroustrup, April 21, 2015

- Improve support for large-scale dependable software
- Provide support for higher-level concurrency models
- Simplify core language use, especially as it relates to the STL and concurrency, and address major sources of errors.
- Preserve Fundamental strengths
- A direct map to hardware (initially from C)
- Zero-overhead abstraction (initially from Simula)

The devil is in the Details

- Improve support for large-scale dependable software
- Modules (to improve locality and improve compile time)
- contracts (for improved specification)
- a type-safe union (probably pattern matching)
- Provide support for higher-level concurrency models
- asio for basic networking
- a SIMD vector
- improved futures
- co-routines (finally, again for the first time since 1990)
- transactional memory
- parallel algorithms (incl. parallel versions of some of the STL)
- Simplify core language use, especially as it relates to the STL and concurrency, and address major sources of errors.

Even More details

- Concepts
- concepts in the standard library
- ranges (simplifies STL use, among other things)
- default comparisons
- uniform call syntax (among other things: it helps concepts and STL style library use)
- operator dot (to finally get proxies and smart references)
- array_view and string_view (better type checking, DMR wanted those: "fat pointers")
- arrays on the stack ("stack_array" anyone? But we need to find a safe way of dealing with stack overflow)
- optional (unless it is subsumed by pattern matching, and I think not)

“What I do not want to try do”

- Turn C++ into a radically different language
- Turn parts of C++ into a much higher-level language by providing a segregated sub-language
- Have C++ compete with every other language by adding as many as possible of their features
- Incrementally modify C++ to support a whole new "paradigm“
- Hamper C++'s use for the most demanding systems programming tasks
- Increase the complexity of C++ use for the 99% for the benefit of the 1% (us and our best friends)

Bad Committee habits to avoid

- make something a library because that's easier to get accepted in the committee than a language feature (even if there is good argument that what is provided is fundamental)
- provide an isolated feature because integration with existing features would cause work on compatibility issues. This just postpones integration until later
- if given a choice between two alternatives, the committee chooses both, adds a third, and modifies the first two "to please everybody who could affect the vote" (this is pure design-by- committee)
- oppose proposals seen as competing with your favorite proposal for time/resources
- push hard for the immediately useful (only)
- oppose proposals not relevant to your current job, stalling an improvement that would benefit others
- focus on the WP text and choose among technical alternatives based on what fits best with the current text, rather than giving precedence to user needs
- think that more syntax equate to safety and ease of use for the majority of programmers
- serve the library writers and other experts while ignoring the majority of current and potential C++ programmers
- present “principles” as non-negotiable absolutes
- try to do “everything”