Summary of C++17 features

Is this release good, great, amazing or meh?

Bartłomiej Filipek
http://www.bfilipek.com/

C++ User Group Krakow
20th November 2017

About me

www.bfilipek.com

10y+ experience

Currently @Xara.com

- Text related features

Somehow addicted to Cpp :D

The plan

- C++17 timeline
- What we get
  - Fixes and deprecation
  - Language clarification
  - Templates
  - Attributes
  - Simplification
  - Filesystem
  - Parallel STL
  - Utils
- Summary

Timeline

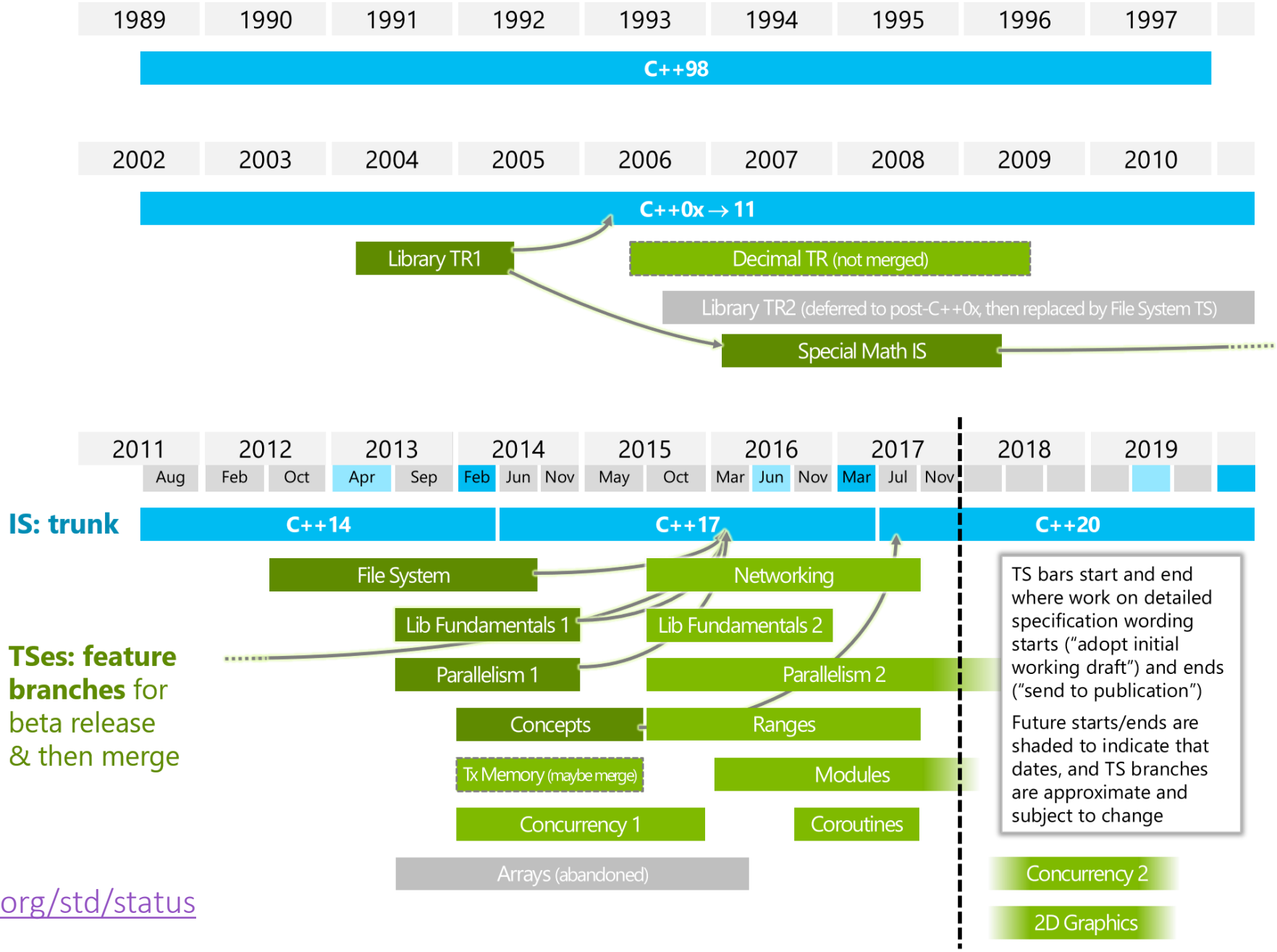
C++17 is formally approved
herbsutter.com/2017/09/06/c17 ...
#cpp
#cplusplus

C++17 is formally approved

[revised 9/8 to reflect that there is no need to wait till the next WG21 meeting] As I mentioned in my Kona (March) trip report, WG21 (the ISO C++ committee) completed work on C++17 at o...

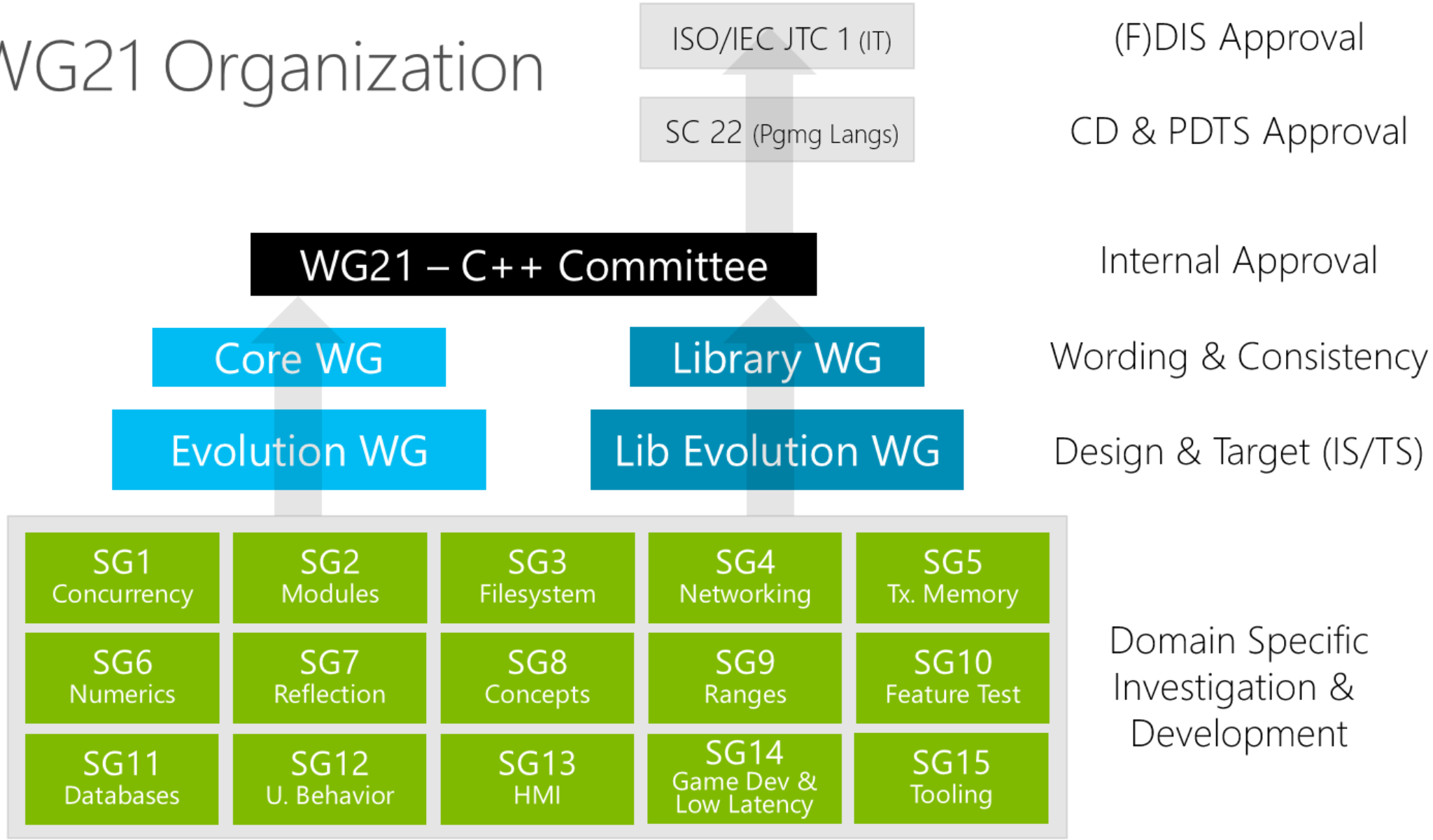
herbsutter.com

9:08 AM - 7 Sep 2017



https://isocpp.org/std/status

WG21 Organization



(F)DIS Approval

CD & PDTS Approval

Internal Approval

Wording & Consistency

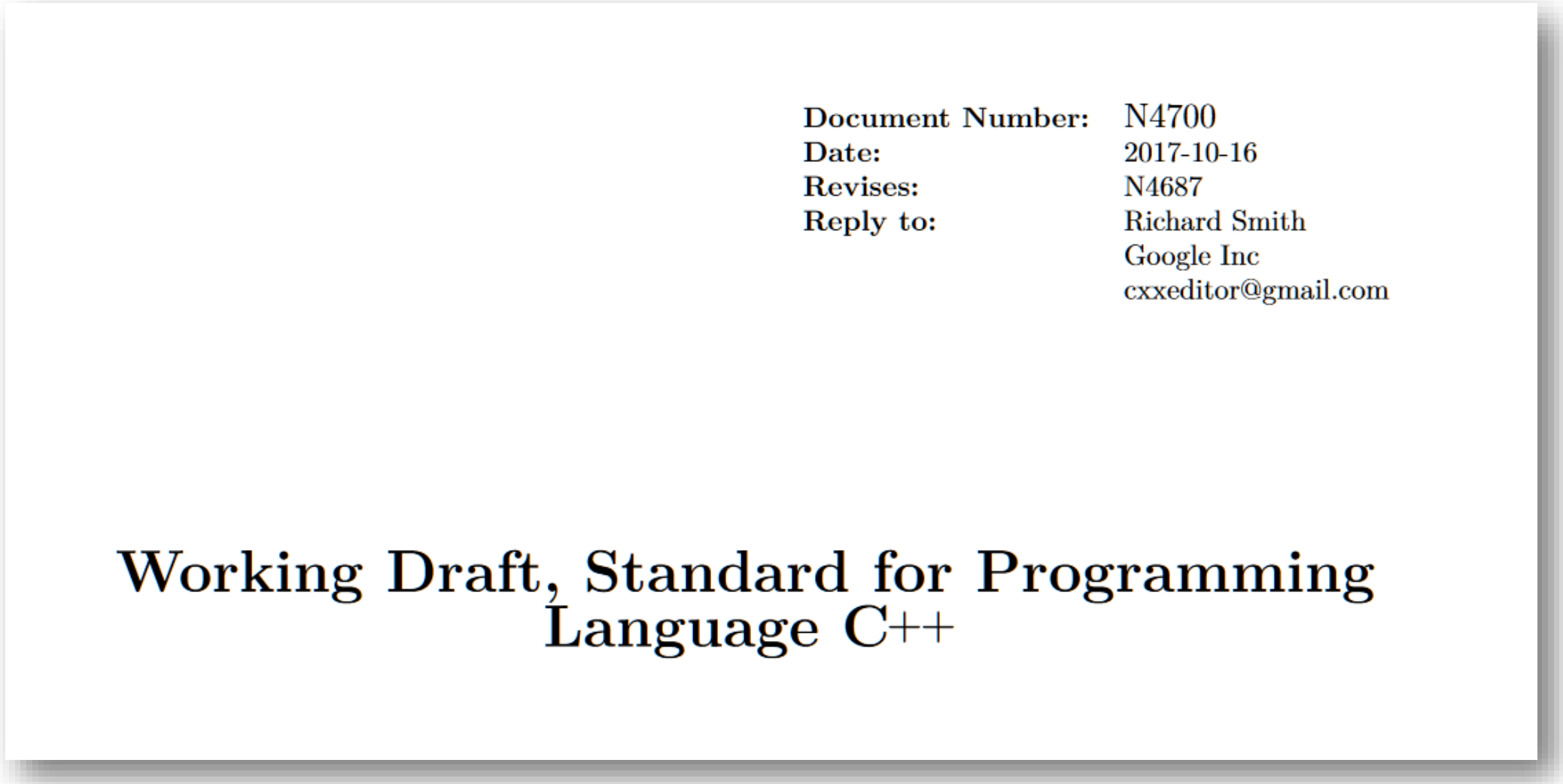
Design & Target (IS/TS)

Domain Specific Investigation & Development

https://isocpp.org/std/the-committee

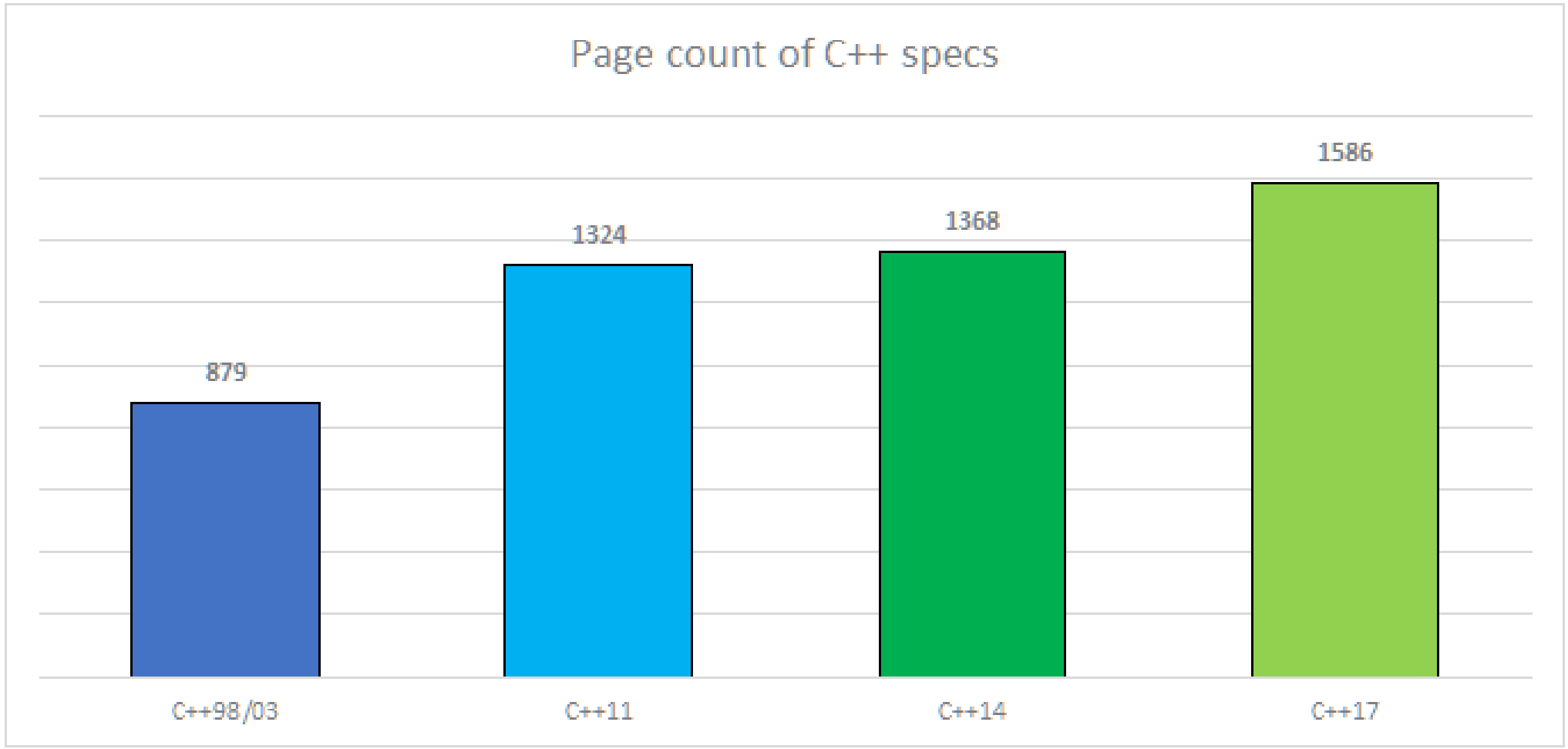
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4700.pdf

https://github.com/cplusplus/draft



Document Number: N4700
Date: 2017-10-16
Revises: N4687
Reply to: Richard Smith
Google Inc
cxxeditor@gmail.com

Working Draft, Standard for Programming Language C++


Page count of C++ specs
879
1324
1368
1586
C++98/03
C++11
C++14
C++17

Ship every 3 years

New auto rules for direct-list-initialization
static_assert with no message
typename in a template template parameter
Removing trigraphs
Nested namespace definition
Attributes for namespaces and enumerators
u8 character literals
Allow constant evaluation for all non-type template arguments
Fold Expressions
Unary fold expressions and empty parameter packs
Remove Deprecated Use of the register Keyword
Remove Deprecated operator++(bool)
Removing Deprecated Exception Specifications from C++17
Make exception specifications part of the type system
Aggregate initialization of classes with base classes
Lambda capture of *this
Using attribute namespaces without repetition
Dynamic memory allocation for over-aligned data
__has_include in preprocessor conditionals
Template argument deduction for class templates
Non-type template parameters with auto type
Guaranteed copy elision
New specification for inheriting constructors (DR1941 et al)
Direct-list-initialization of enumerations
Stricter expression evaluation order
constexpr lambda expressions
Different begin and end types in range-based for
[[fallthrough]] attribute
[[nodiscard]] attribute
[[maybe_unused]] attribute
Ignore unknown attributes
Pack expansions in using-declarations
Structured Binding Declarations
Hexadecimal floating-point literals
init-statements for if and switch
Inline variables
DR: Matching of template template-arguments excludes compatible templates
std::uncaught_exceptions()
constexpr if-statements

Merged: The Library Fundamentals 1 TS (most parts)
Removal of some deprecated types and functions, including std::auto_ptr, std::random_shuffle, and old function adaptors
Merged: The Parallelism TS, a.k.a. “Parallel STL.”,
Merged: File System TS,
Merged: The Mathematical Special Functions IS,
Improving std::pair and std::tuple
std::shared_mutex (untimed)
Variant, Optional, Any
Splicing Maps and Sets

1. Fixes and deprecation
2. Language clarification
3. Templates
4. Attributes
5. Simplification
6. Filesystem
7. Parallel STL
8. Utils

Deprecation & Removal

Removing trigraphs

Removing register keyword

Remove Deprecated operator++(bool)

Cannot inherit from std::iterator

`auto_ptr` is gone!

MSVC 2017 using /std:c++latest:

error C2039: 'auto_ptr': is not a member of 'std'

Removing Deprecated Exception Specifications from C++17

```
void fooThrowsInt(int a) throw(int)
{
        printf_s("can throw ints\n");
        if (a == 0)
                throw 1;
}
```

Use “noexcept”

Fixes

static_assert with no message

New auto rules for direct-list-initialization

```
auto x5{ 3 };                    // decltype(x5) is int, not initializer_list
```

Different begin and end types in range-based for

```
{
    auto && __range = for - range - initializer;
    for (auto __begin = begin - expr,
              __end = end - expr;
              __begin != __end;
              ++__begin)
    {
              for_range_declaration = *__begin;
              statement
    }
}
```

```
{
    auto && __range = for - range - initializer;
    auto __begin = begin - expr;
    auto __end = end - expr;
    for (; __begin != __end; ++__begin)
    {
              for_range_declaration = *__begin;
              statement
    }
}
```

Language clarifications

Guaranteed copy elision

(only for temporary objects, not for Named RVO)

Exception specifications part of the type system

```
void foo() noexcept;
void foo();
```

Dynamic memory allocation for over-aligned data

new is now aware of the alignment of the object.

Stricter expression evaluation order

Why do we need make_unique?

```
foo(make_unique<T>(), otherFunction());
```

```
foo(unique_ptr<T>(new T), otherFunction());
```

Templates

Template argument deduction for class templates

```
std::pair<int, double> p(10, 0.0);
// same as
std::pair p(10, 0.0); // deduced automatically!
```

```
std::lock_guard<std::shared_timed_mutex,
        std::shared_lock<std::shared_timed_mutex>> lck(mut_, r1);

// Can now become :

std::lock_guard lck(mut_, r1);
```

custom class template deduction guides

Fold expressions

```
auto SumCpp11()
{
        return 0;
}

template<typename T1, typename... T>
auto SumCpp11(T1 s, T... ts)
{
        return s + SumCpp11(ts...);
}
```

```
template<typename ...Args> auto sum(Args ...args)
{
        return (args + ... + 0);
}
```

```
template<typename ...Args>
void FoldPrint(Args&&... args)
{
        (cout << ... << forward<Args>(args)) << '\n';
}
```

```cpp
// SFINAE
template <typename T, std::enable_if_t < std::is_pointer<T>{} > * = nullptr >
auto get_value(T t)
{
        return *t;
}

template <typename T, std::enable_if_t < !std::is_pointer<T>{} > * = nullptr >
auto get_value(T t)
{
        return t;
}
```

```cpp
// Tag dispatching
template <typename T>
auto get_value(T t, std::true_type)
{
        return *t;
}

template <typename T>
auto get_value(T t, std::false_type)
{
        return t;
}

template <typename T>
auto get_value(T t)
{
        return get_value(t, std::is_pointer<T>{});
}
```

https://blog.tartanllama.xyz/if-constexpr/

constexpr if

```
template <typename T>
auto get_value(T t)
{
        if constexpr (std::is_pointer_v<T>)
                return *t;
        else
                return t;
}
```

Attributes

[[fallthrough]]

[[nodiscard]]

[[maybe_unused]]

```
enum class[[nodiscard]] ErrorCode{
        OK,
        Fatal,
        System,
        FileIssue
};
ErrorCode Compute() { ... }
...
Compute(); // warning!
```

@cppreference.com
Attributes available in C++17

[[noreturn]]
[[carries_dependency]]
[[deprecated]]
[[deprecated("msg")]]
[[fallthrough]]
[[nodiscard]]
[[maybe_unused]]

How will it simplify the code?

Inline variables

- Great for header-only libs!

constexpr if

Structured Binding Declarations

```
// works with arrays:
double myArray[3] = { 1.0, 2.0, 3.0 };
auto[a, b, c] = myArray;
```

```
auto[a, b] = myPair; // binds myPair.first/second (must implement get<N>)
```

```
// non static, public members
struct S { int x1; double y1; };
S f();
const auto[x, y] = f();
```

```
std::map myMap;
for (const auto &[key, val] : myMap)
{

}
```

Init-statement for if/switch

```
{
        auto val = GetValue();
        if (condition(val))
                // on success
        else
                // on false...
}
```

```
if (auto val = GetValue(); condition(val))
        // on success
else
        // on false...
```

```
if (const auto it = myString.find("Hello"); it != std::string::npos)
        std::cout << it << " Hello\n";

if (const auto it = myString.find("World"); it != std::string::npos)
        std::cout << it << " World\n";
```

Structured bindings & init in if statements

```
// better together: structured bindings + if initializer
if (auto[iter, succeeded] = mymap.insert(value); succeeded) {
        use(iter);  // ok
// ...
} // iter and succeeded are destroyed here
```

Filesystem

```
namespace fs = std::experimental::filesystem;
void DisplayDirTree(const fs::path& pathToShow, int level)
{
        if (fs::exists(pathToShow) && fs::is_directory(pathToShow))
        {
                auto lead = std::string(level * 3, ' ');
                for (const auto& entry : fs::directory_iterator(pathToShow))
                {
                        auto filename = entry.path().filename();
                        if (fs::is_directory(entry.status()))
                        {
                                cout << lead << "[+] " << filename << "\n";
                                DisplayDirTree(entry, level + 1);
                                cout << "\n";
                        }
                        else if (fs::is_regular_file(entry.status()))
                                DisplayFileInfo(entry, lead, filename);
                        else
                                cout << lead << " [?]" << filename << "\n";
                }
        }
}
```

Parallel STL

```
std::vector<int> v = genLargeVector();

// standard sequential sort
std::sort(v.begin(), v.end());

// explicitly sequential sort
std::sort(std::seq, v.begin(), v.end());

// permitting parallel execution
std::sort(std::par, v.begin(), v.end());

// permitting vectorization as well
std::sort(std::par_unseq, v.begin(), v.end());
```

Utils

std::any

std::variant

std::optional

string_view

Searchers

```
auto a = std::any(12);
a = std::string("hello world");
a = 10.0f;
cout << std::any_cast<float>(a);
```

```
std::variant<int, float, std::string> abc;
// get<>
// get_if<>
// visit()
```

```
std::optional<std::string> ostr = GetUserResponse();
```

```
default_searcher
boyer_moore_searcher
boyer_moore_horspool_searcher
```

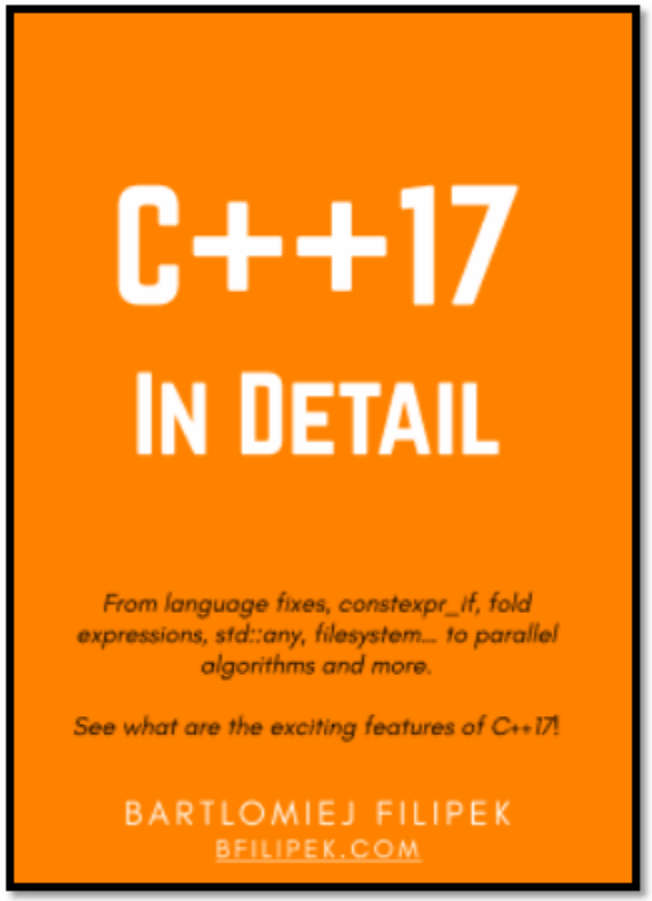
http://www.bfilipek.com/2017/09/cpp17-details-utils.html

Compiler support

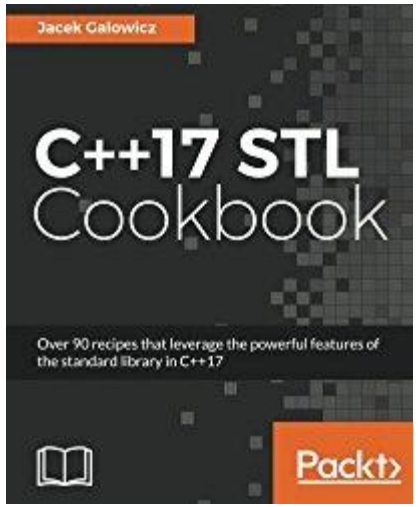
Summary

Is C++17 great? Amazing? Or just meh… ?

Bonus



http://www.bfilipek.com/2017/09/c17-in-detail-summary-bonus.html



https://isocpp.org/