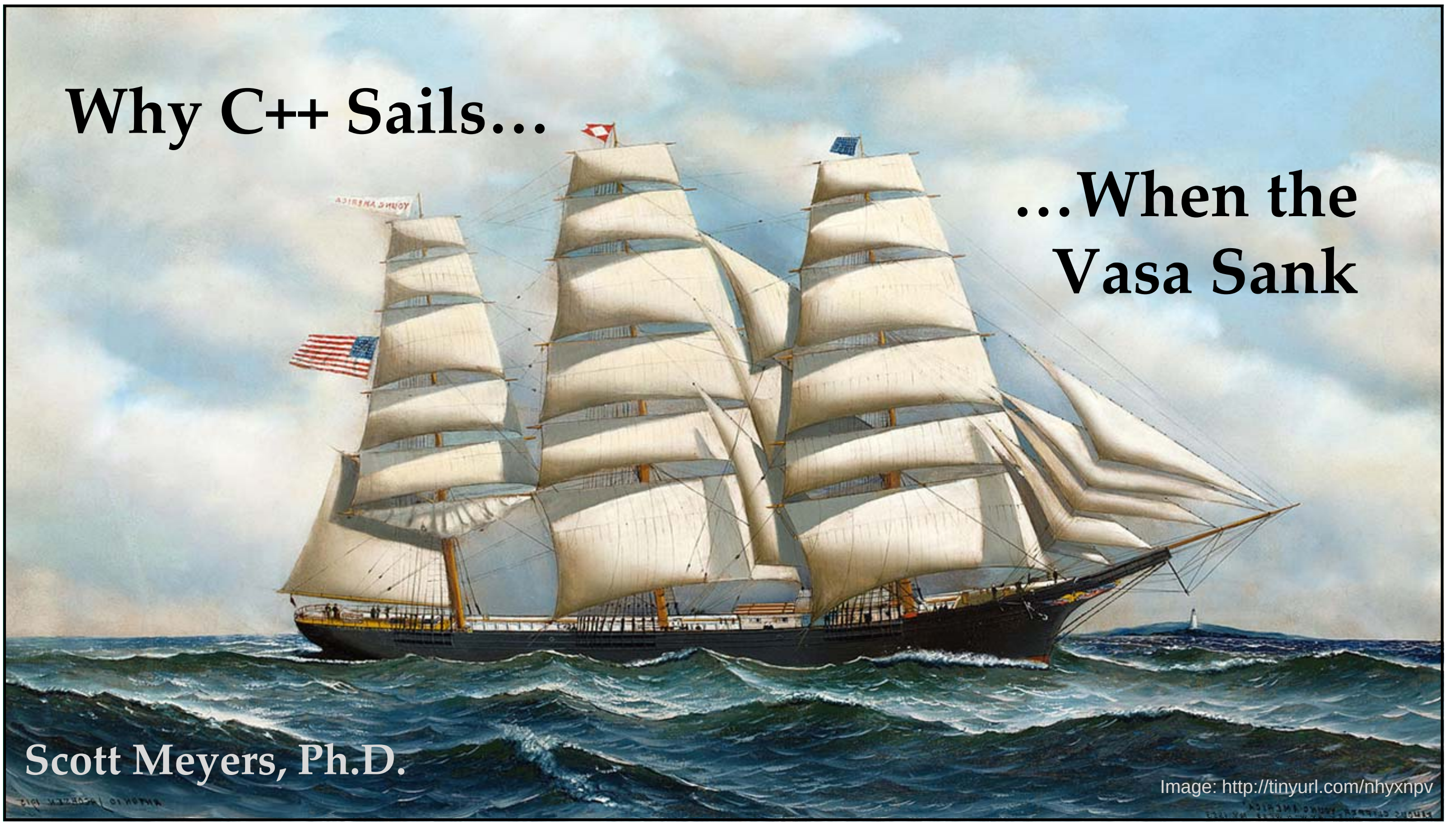


C++ in 1992

Standardization of a fledging language.

- Endless stream of extension proposals.
- Concern by standardization committee.
  - ➡ STL hadn't yet been proposed.

Every extension proposal should be required to be accompanied by a kidney. People would submit only serious proposals, and nobody would submit more than two.

— Jim Waldo

C++ is already too large and complicated for our taste...

Remember the Vasa!

— Bjarne Stroustrup


Image: http://tinyurl.com/qdam8jm

Scott Meyers, Software Development Consultant
http://www.aristeia.com/

Image: tinyurl.com/9sgwa6w

The Vasa

- Commissioned 1625 by Swedish King Gustav.
- Much redesign during construction.
  - ➡ Ultimately designed to be flagship of royal navy.
- Heavily armed and decorated.

Scott Meyers, Software Development Consultant
http://www.aristeia.com/

Slide 3

Image: tinyurl.com/bzpxqsj

The Vasa

Top-heavy and unstable, it sank on its maiden voyage.

- Sailed less than a mile.
- Dozens died.

Scott Meyers, Software Development Consultant
http://www.aristeia.com/

Slide 4

C++: Heavily Armed

Classes (concrete and abstract)	Namespaces	Virtual member functions
Constructors and Destructors	RTTI	Nested classes
Default parameters	Locales	Nonvirtual functions
Operator overloading	constexprs	Static member functions
new and delete	Bitwise operations	Pure virtual functions
Lambda expressions	Iostreams	Function overloading
Inheritance (public, private, protected, virtual, multiple)	Static data (in classes, functions, namespaces, files)	Templates (including total and partial specialization)
Nonvirtual member functions	Inline functions	Atomic data types
Exceptions/Exception Specifications	References (Arguably 3 kinds)	New Handlers
Private and protected members	Friends	Memory Models (3)
User-defined literals	Raw and smart Pointers	Promises and Futures
Type deduction (3 sets of rules)	The STL	Enums (2 kinds)

Scott Meyers, Software Development Consultant
http://www.aristeia.com/

Slide 5

C++: Ornately Decorated

What is f in this code?

```
f(x);        // "invoke f on x"
```

- Maybe a function (or function pointer or function reference).
- Maybe an instance of a class overloading operator().
- Maybe an object implicitly convertible to one of the above.
- Maybe an overloaded function name.
  - At any of multiple scopes.
- Maybe the name of one or more templates.
  - At any of multiple scopes.
- Maybe the name of one or more template specializations.
- **Maybe several of the above!**
  - E.g., overloaded function name + overloaded template name + name of template specialization(s).

Scott Meyers, Software Development Consultant
http://www.aristeia.com/

Slide 6

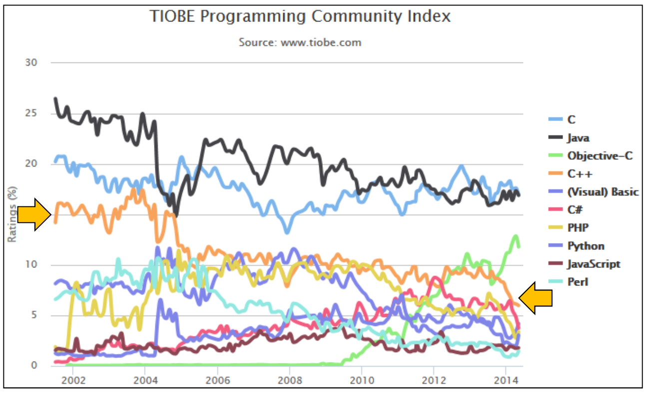
Too Complicated?

If you think C++ is not overly complicated, just what is a "protected abstract virtual base pure virtual private destructor," and when was the last time you needed one?

— Tom Cargill (1990)

```
class Base {
private:
   virtual ~Base() = 0;
};
class Derived: protected virtual Base { ... };
```


C++ Must have Done Something Right



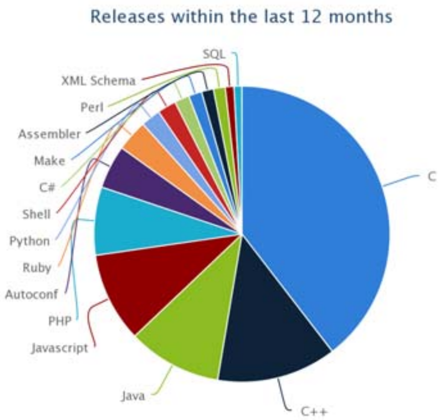
Programming Language	2014	2009	2004	1999	1994	1989
C	1	2	2	1	1	1
Java	2	1	1	15	-	-
Objective-C	3	36	45	-	-	-
C++	4	3	3	2	2	2
C#	5	7	8	26	-	-
PHP	6	5	6	-	-	-
(Visual) Basic	7	4	5	3	3	7
Python	8	6	10	29	22	-
JavaScript	9	9	9	20	-	-
Transact-SQL	10	29	-	-	-	-
Lisp	14	20	15	12	6	3

Source: http://tinyurl.com/3xutoh

C++ Must have Done Something Right

Language use in active open source projects:

Language	%
C	38.18
C++	12.62
Java	9.91
Javascript	9.46
PHP	7.19
Autoconf	4.58
Ruby	3.31
Python	2.07
Shell	1.97
C#	1.57
Make	1.39
Assembler	1.27
Perl	1.24
XML Schema	0.91
SQL	0.83



Change in past year:

Language	%
C	-14.7
C++	2
Java	2.19
Javascript	3.03
PHP	2.83
Shell	-0.52
Autoconf	2.4
Python	0.29
Assembler	-0.5
Ruby	1.72
Perl	-0.15
C#	0.55
SQL	-0.02
XML Schema	0.07
Make	0.65

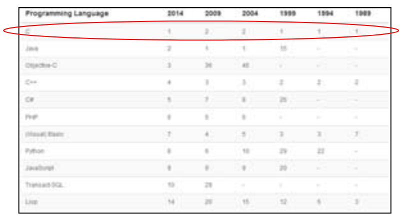
Scott Meyers, Software Development Consultant
http://www.aristeia.com/

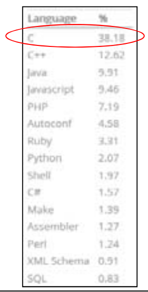
Slide 9

Compatibility with C

- Programmers:
  - Important in 1982.
  - Important now.
    - C longevity astonishing.
- Source code:
  - "As close as possible to C, but no closer."
- Object code:
  - Direct access to universal assembly language.
- Build chains:
  - Editors, compilers, linkers, debuggers, etc.

C & C++ still often lumped together: "C/C++".





Language	%
C	38.18
C++	12.62
Java	9.91
Javascript	9.46
PHP	7.19
Autoconf	4.58
Ruby	3.31
Python	2.07
Shell	1.97
C#	1.57
Make	1.39
Assembler	1.27
Perl	1.24
XML Schema	0.91
SQL	0.83

Scott Meyers, Software Development Consultant
http://www.aristeia.com/

Slide 10

Very General Features

Lead to surprising uses.

- Destructors.
  - ➡ RAII ⇒ general-purpose Undo.
    - ◆ Basis of all resource management.
    - ◆ Exception safety.
- Templates.
  - ➡ Generic programming (e.g., STL).
  - ➡ TMP.
    - ◆ Compile-time dimensional units analysis.
- Overloading.
  - ➡ Generic programming.
  - ➡ Smart pointers.
  - ➡ Function objects.
    - ◆ Basis for lambdas in C++11.

A language for library writers.

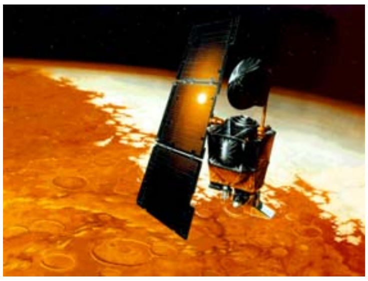


Image: tinyurl.com/awmleyr

$$\frac{1}{X_0} = 4\,\alpha\, r_e^2\, \frac{N_A}{A}\, \{Z^2\, [L_{rad} - f(Z)] + Z\, L'_{rad}\}$$

```
Energy<> finalEnergy(Element<> const & material, Density<> const dens,
                     Length<> const thick, Energy<> const initEnergy) {
  AtomicWeight<> const A = material->atomicWeight;
  AtomicNumber<> const Z = material->atomicNumber;
  Number<> const L_rad = log( 184.15 / root<3>( Z ) );
  Number<> const Lp_rad = log( 1194. / root<3>(Z*Z) );
  Length<> const X_0 = 4.0 * alpha * r_e * r_e * N_A / A *
                       ( Z * Z * L_rad + Z * Lp_rad );
  return initEnergy / exp( thick / X_0 );
}
```


Paradigm Agnosticism

Image: tinyurl.com/awhpv75

- Procedural programming.
  - ➡ Free functions:
    - ◆ Basis of STL.
    - ◆ Naturally models symmetric relationships.
    - ◆ Facilitates natural type extension.
- OOP.
  - ➡ Including MI.
- Generic programming.

  I tried to implement STL in other languages and failed.
  C++ was the only language in which I could do it.
  — Alexander Stepanov
- Functional programming.
  - ➡ Function objects (including closures).
  - ➡ TMP.

Paradigm Agnosticism

- "Unsafe" programming.
  - Pointers, casts, unchecked arrays, etc.
  - **"Trust the programmer."**

- "Safe" programming.
  - Vectors with bounds checking, checked iterators, etc.

Images: tinyurl.com/4kk6b5z and tinyurl.com/aeae4lm

Commitment to Systems Programming

Suitable for demanding systems applications.

- Program speed.
  - Many systems are never fast enough.
  - Some systems must conserve CPU cycles.
- Program size.
  - Static (including RTL).
  - Dynamic (i.e., image and working set size).
- Data layout.
  - Drivers.
  - Communications protocols.
  - Use with legacy code/systems.

- Efficient communication with outside entities.
  - Hardware.
  - OSes.
  - Code in other languages.

Zero overhead principle:

- **You don't pay (at runtime) for what you don't use.**

Image: tinyurl.com/yhr7e95

Commitment to Systems Programming

Designed for wide platform availability.

- Microcontrollers to supercomputers.
  - ➡ Implementation-defined sizes for e.g., `int`s and pointers.
  - ➡ No assumptions about I/O capabilities.
- Hosted and unhosted.
  - ➡ No OS required.

C++ compilers available almost everywhere.

Images: http://tinyurl.com/nokq6to and http://tinyurl.com/p5qx5ah

Dedication to Backwards-Compatibility

Legacy code keeps working.

- Circa-1989 C++ code almost always valid C++14 code, too.

Standardization safeguards investments in C++.

C++ vs. The Vasa

So far, C++ has escaped the fate of the Vasa; it has not keeled over and disappeared – despite much wishful thinking and many dire predictions.

— Bjarne Stroustrup, 1996

Why?

- Compatibility with C.
- Very general features.
- Paradigm agnosticism.
- Commitment to systems programming.
- Dedication to backwards-compatibility.

Complexity Revisited

Consider again:

```
f(x);        // "invoke f on x"
```

- f may be a function (or function pointer or function reference).
- f may be an instance of a class overloading operator().
- f may be an object implicitly convertible to one of the above.
- f may be an overloaded function name.
  - At any of multiple scopes.
- f may be the name of one or more templates.
  - At any of multiple scopes.
- f may be the name of one or more template specializations.
- **f may be several of the above!**
  - E.g., overloaded function name + overloaded template name + name of template specialization(s).

Complexity Revisited

Is this really confusing?

```
std::cout << x;                    // call operator<<(std::cout, x)
```

operator<< is overloaded, templatized, and specialized.

- In multiple scopes.

Most C++ complexity hidden from most users most of the time.

Complex for Whom?

User view often pretty simple:

```
auto y = std::move(x);                    // request that x's value be moved to y
```

Implementer view often more complex:

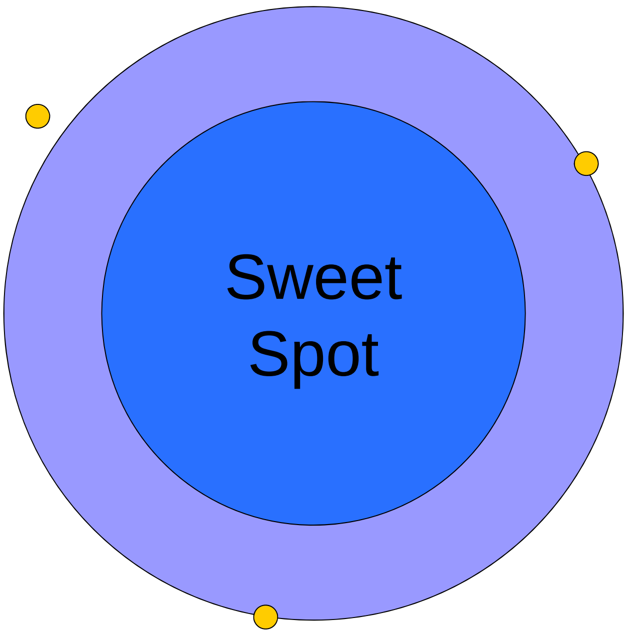
```
namespace std {
  template<typename T>
  typename remove_reference<T>::type&&
  move(T&& param) noexcept
  {
    typedef remove_reference<T>::type&& ReturnType;
    return static_cast<ReturnType>(param);
  }
}
```

Destined for Complexity?

C++ most suited for **demanding systems applications**.

- Less demanding software ⇒ other languages suffice.
- You choose C++ ⇒ the situation is already complicated.

Or a Reflection of its Users?



A Language on the Move

C++ Standard evolving increasingly rapidly:

- **ARM C++** (1990): 453 pages.
  - Includes explanatory annotations.
- **C++98**: 776 pages (71% bigger). Δt = 8 years
- **C++11**: 1353 pages (75% bigger). Δt = 13 years
- **C++14**: ~1370 pages (~1% bigger). Δt = 3 years
- **C++17**: Probably much bigger than C++14. Δt = 3 years

Bigger ≡ more complicated.

Simplification Through Added Complexity

New features often simplify common tasks:

```
std::vector<Widget> v;

for (std::vector<Widget>::const_iterator ci = v.begin();      // C++98
     ci != v.end();
     ++ci) {
  use ci (often *ci)
}

for (auto ci = v.cbegin(); ci != v.end(); ++ci) {              // C++11
  use ci (often *ci)
}

for (const auto cw& : v) {                                     // C++11
  use cw
}
```

Simplification Through Added Complexity

E.g., evolution of function object creation:

```
std::vector<int> v;

class InValueRange {                                          // C++98
public:
  InValueRange(int bottom, int top): b(bottom), t(top) {}

  bool operator()(int val) const
  { return bottom <= val && val <= top; }

private:
  int b, t;
};

std::vector<int>::iterator i = std::find_if(v.begin(), v.end(),
                                             InValueRange(10, 20));

auto i = std::find_if(v.begin(), v.end(),                     // C++11
                      [](int val) { return 10 <= val && val <= 20; });
```


Simplification Through Added Complexity

```
struct ValidateAndFwd {                                       // C++11
  template<typename... Ts>
  auto operator()(Ts&&... params) const ->
    decltype(process(std::forward<Ts>(params)...))
  {
    validateCredentials();
    return process(std::forward<Ts>(params)...);
  }
};

auto f11 = ValidateAndFwd();


auto f14 = [](auto&&... params) -> decltype(auto) {           // C++14
  validateCredentials();
  return process(std::forward<decltype(params)>(params)...);
};
```

Status of The Vasa and C++

Image: tinyurl.com/bksdcrg

Museum piece:
Preserved for Posterity


Working Draft, Standard for Programming Language C++

Living Language:
Still Growing and Maturing

Further Information

- "Vasa (ship)," *Wikipedia*.
- "Why The Vasa Sank: 10 Problems and Some Antidotes for Software Projects," Richard E. Fairley and Mary Jane Willshire, *IEEE Software*, March/April 2003.
- "The Vasa: A Disaster Story with Sofware [sic] Analogies," Linda Rising, *The Software Practitioner*, January-February 2001.
- "How to write a C++ language extension proposal," Bjarne Stroustrup, *The C++ Report*, May 1992.
- "C++: The Making of a Standard Journey's End," Chuck Allison, *C/C++ Users Journal*, October 1996.
  - ➡ Primarily an interview with Bjarne Stroustrup.
- "TIOBE Programming Community Index," *TIOBE Software*, http://www.tiobe.com/index.php/content/paperinfo/tpci/.
- "Open Source Project Data," *Black Duck Open Source Resource Center*, http://www.blackducksoftware.com/oss/projects.

Further Information

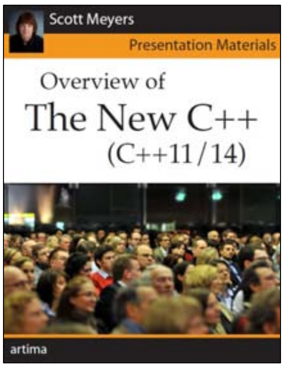
- "Software Development for Infrastructure," Bjarne Stroustrup, *IEEE Computer*, January 2012.
- "C/C++'s Enduring Popularity," *Coverity Software Integrity Blog*, 22 February 2010.
- "Why C++," Kyle Wilson, *GameArchitect.net*, 15 July 2006.

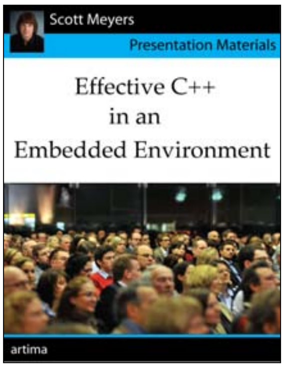
Licensing Information

Scott Meyers licenses materials for this and other training courses for commercial or personal use. Details:

- Commercial use: http://aristeia.com/Licensing/licensing.html
- Personal use: http://aristeia.com/Licensing/personalUse.html

Courses currently available for personal use include:





Scott Meyers, Software Development Consultant
http://www.aristeia.com/
© 2014 Scott Meyers, all rights reserved.

About Scott Meyers

Scott Meyers is one of the world's foremost authorities on C++. His web site,

http://aristeia.com/

provides information on:

- Technical training services
- Upcoming presentations
- Books, articles, online videos, etc.
- Professional activities blog

Scott Meyers, Software Development Consultant
http://www.aristeia.com/
© 2014 Scott Meyers, all rights reserved.