Issue 077: C++

De Programmatica Ipsum

2025-02-03

Contents

Issue 077: C++ **1**

Twin Leaks: C++ Walk With Me **3**

Herb Sutter **15**

Scott Meyers **23**

Issue 077: C++

By Adrian Kosmaczewski, February 3rd, 2025

Welcome to the 77th issue of *De Programmatica Ipsum*, about *The C++ Programming Language*.

In this edition:

- We take the risqué choice of comparing the work of Bjarne Stroustrup[1] to that of the late film director David Lynch.

[1] https://deprogrammaticaipsum.com/twin-leaks-cpp-walk-with-me/

- In the Library section[2], we review "Effective C++" by Scott Meyers[3].
- In our Vidéothèque section[4], we learn about programming language safety from Herb Sutter[5].

Download this issue in PDF[6] or EPUB[7] format, and read it on your preferred device.

We would like to thank our patrons who generously contribute every month (or have contributed in the past) to our work and help us run this magazine. Thank you so much! In alphabetical order: Adam Guest, Adrian Tineo Cabello, Benjamin Sheldon, Christopher Nascone, Colin Powell, Franz Lucien Moersdorf, Guillermo Ramos Álvarez, Jean-Paul de Vooght, Dr. Juande Santander-Vela, Patryk Matuszewski, Paul Hudson, Quico Moya, Roger Turner, Szymon Licau, and countless more leaving anonymous tips every month.

Enjoy this issue! Please subscribe to our free newsletter[8] to stay updated about new releases, share the articles on social media, or contribute[9] if you would like to support our work with a donation via Liberapay[10].

Cover photo by Danielle Cerullo[11] on Unsplash[12].

---

[2] https://deprogrammaticaipsum.com/category/library/

[3] https://deprogrammaticaipsum.com/scott-meyers/

[4] https://deprogrammaticaipsum.com/category/videotheque/

[5] https://deprogrammaticaipsum.com/herb-sutter/

[6] https://deprogrammaticaipsum.com/pdf/issue-077-c++.pdf

[7] https://deprogrammaticaipsum.com/epub/issue-077-c++.epub

[8] https://deprogrammaticaipsum.com/newsletter/

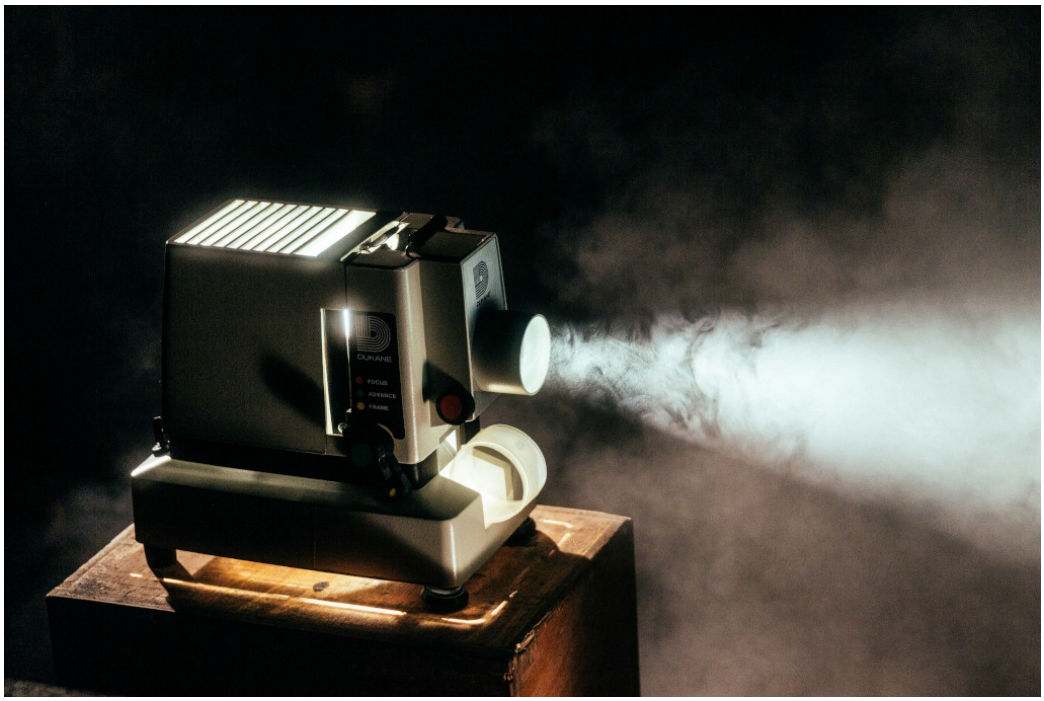
[9] https://deprogrammaticaipsum.com/contribute/

[10] https://liberapay.com/akosma/donate

[11] https://unsplash.com/@dncerullo?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

[12] https://unsplash.com/photos/white-and-gray-office-rolling-chairs-bIZJRVBLfOM?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

Twin Leaks: C++ Walk With Me



By Adrian Kosmaczewski, February 3rd, 2025

The work of Bjarne Stroustrup[13], one of the most important programming language designers of all time, has happened in parallel with that of one of the greatest filmmakers of all time, David Lynch, who passed away during the preparation of this edition. Regular readers of this magazine know that we are always eager to find analogies and synchronicities between our soulless computers and the arts[14], and if you think that

[13] https://en.wikipedia.org/wiki/Bjarne_Stroustrup

[14] https://deprogrammaticaipsum.com/you-are-doing-it-wrong/

this time it will be more of the same, follow your intuition.

The Elephant Man (1980)

C++ started its life as a language literally called "C with Classes"[15]. To this day, this name still conveys a sad inheritance; for most people, that is all that C++ brings to the table. History proves that this is a short-sighted perspective.

Bjarne Stroustrup needed to build large and performant software projects, and thought that object-oriented programming[16] was a fine paradigm to start with. He started working on a set of object-oriented extensions for the C programming language. Why C, you ask? Well, it was a little programming language that happened to be developed a few doors away from his office in Bell Labs, or so tells the legend[17]:

> Pascal was considered a toy language, so it seemed easier and safer to add type checking to C than to add the features considered necessary for systems programming to Pascal. (...)
> At the time, I considered Modula−2, Ada, Smalltalk, Mesa, and Clu as alternatives to C and as sources for ideas for C++ so there was no shortage of inspiration. However, only C, Simula, Algol68, and in one case BCPL left noticeable traces in C++ as released in 1985.

Bundling OOP features on top of C was significant in various ways; object-oriented programming was not (yet) a mainstream thing in the early 1980s, mostly because implementations such as Smalltalk were unfortunately labeled as slow and clunky; hardware in the early eighties was not particularly powerful, and dynamic runtime binding of messages (a *conditio sine qua non* for one of the tenets of OOP, polymorphism) was considered inefficient.

What Stroustrup wanted was apparently what a lot of people wanted: a language that provided OOP features, such as those offered by Smalltalk, but compiled into binaries that would execute as fast as an executable written in C.

The solution Stroustrup adopted was *not* to adopt the Smalltalk model, but instead to take inspiration from fellow Scandinavians and to bolt ideas borrowed from Simula[18] into C. The key innovation was to ensure compile-time binding of methods whenever

[15] https://en.wikipedia.org/wiki/Cfront
[16] https://deprogrammaticaipsum.com/the-hype-cycle-of-oop/
[17] https://dl.acm.org/doi/10.1145/154766.155375
[18] https://en.wikipedia.org/wiki/Simula

possible, and in all the other cases, to use a vtable[19] to dispatch messages as runtime, in the fastest possible way.

The resulting language, created by Stroustrup with the close collaboration of Andrew Koenig[20] almost exactly 40 years ago at the time of this writing, was without any doubt, and against all odds, one of the most influential tools ever released in the history of programming.

However, just like Joseph Merrick[21], C++ was a freak, a curiosity; but one that enchanted programming elites with its capabilities and possibilities. And yes, sadly just like Merrick, C++ was often beaten by angry and uneducated mobs, who tried once and again to dismiss its towering status and to reduce it to a mere caricature.

Blue Velvet (1986)

Thanks to the extraordinary and sustained work of evangelization by Stroustrup himself, which included the release of the first edition of the eponymous book[22], C++ started making rounds on the press around this time.

Dr. Dobbs' mentioned C++ for the first time in its October 1985 issue (to be exact, on page 96[23]) in a review of a competing OOP system called Neon 1.0. But it was still too early; we had to wait until July 1987[24] for a review of that first edition of Stroustrup's book, and to January 1989[25] for a full-blown comparison between C++ and Modula-2[26] (arguably, two major contenders for the OOP crown back in the day).

What about BYTE magazine? We find a first mention of C++ in an editorial in page 6 of the April 1986[27] edition, where the author talks about an upcoming OOP confer-

[19]https://en.wikipedia.org/wiki/Virtual_method_table
[20]https://en.wikipedia.org/wiki/Andrew_Koenig_(programmer)
[21]https://en.wikipedia.org/wiki/Joseph_Merrick
[22]https://www.stroustrup.com/1st.html
[23]https://archive.org/details/dr_dobbs_journal_vol_10/page/784/mode/2up
[24]https://archive.org/details/dr_dobbs_journal_vol_12/page/572/mode/2up
[25]https://archive.org/details/dr_dobbs_journal_vol_14/page/36/mode/2up
[26]https://en.wikipedia.org/wiki/Modula-2
[27]https://archive.org/details/BYTE-MAGAZINE-COMPLETE/198604_Byte_Magazine_Vol_11-04_Number_Crunching.pdf

ence featuring names such as Brad Cox[28], Daniel Ingalls[29], Larry Tesler[30], and Bjarne Stroustrup. Quite the lineup, if you ask me.

A few months later, the August 1986[31] edition was entirely dedicated to the subject of OOP, with a review of Stroustrup's book on page 63, and an article by Larry Tesler about the developer experience of using OOP languages, using C++ as one example.

But just like Lynch's "Blue Velvet"[32], C++ still received a mixed response in those early days. Developers and their organizations were still too much entrenched in the structured[33] ways of Pascal[34], the predominant language in both Macs and PCs, and did not yet see the symbolism behind OOP. We could all feel that something great was brewing there, but we needed to ~~rewatch the film~~ rewrite our apps quite a few times more, before we understood the symbolism, and stopped focusing on that severed ear on the floor.

Wild at Heart (1990)

According to Stroustrup himself[35], around 150'000 developers were writing C++ for a living in 1990.

> In other words, the C++ user population doubled every 7.5 months or so.

Major and minor vendors, ranging from behemoths such as Intel, Microsoft, or Borland, to smaller ones like Watcom, Zortech, and Comeau, to free software communities like the GNU project, were all delivering more and more powerful compilers. Again according[36] to Stroustrup,

> Borland, the largest single C++ compiler supplier, publicly stated that it had shipped 500,000 compilers by October 1991.

---

[28] https://deprogrammaticaipsum.com/brad-cox/

[29] https://deprogrammaticaipsum.com/the-absolute-no-frills-quite-ignorant-very-incomplete-and-certainly-flawed-beginners-guide-to-smalltalk/

[30] https://deprogrammaticaipsum.com/bret-victor/

[31] https://archive.org/details/BYTE-MAGAZINE-COMPLETE/198608_Byte_Magazine_Vol_11-08_Object-Oriented_Languages.pdf

[32] https://en.wikipedia.org/wiki/Blue_Velvet_(film)

[33] https://deprogrammaticaipsum.com/edward-nash-yourdon/

[34] https://deprogrammaticaipsum.com/lazarus-come-forth/

[35] https://dl.acm.org/doi/10.1145/154766.155375

[36] https://dl.acm.org/doi/10.1145/154766.155375

The ecosystem around C++ was already strong enough for the Microsoft Systems Journal[37] to start including a "C/C++ Q&A" section in the March/April 1992 issue (volume 7, number 2). Slightly earlier than that, Apple included articles about C++ in their "develop" journal[38], like for example in issue 2 of April 1990, where we find a style guide and a guide to use C++ with MPW[39] handles under the standard Macintosh memory manager.

As Scott Meyers[40] and Stroustrup repeatedly observed, C++ was already a *federation of four languages into one*, not just a single language: C, Object-Oriented C++, Template Metaprogramming, and The Standard Template Library.

The 1990s was the time of the discovery of the curiously recurring template pattern[41], among others by James Coplien[42]. The time of Alex Stepanov[43] giving us the Standard Template Library[44]. The time of Metrowerks CodeWarrior[45] on the Mac. The time of Marshall Cline's C++ FAQ[46], nowadays merged with Stroustrup's own. The time of every framework bundling its own, incompatible, home-made `string` class. The time of growth of the C++ committee, which by the way has always been open to all and anyone, as explained[47] by (you guessed it) Stroustrup:

> The members of the [The American National Standards Institute committee for C++, or ANSI J16] committee are volunteers who have to pay (about $800 a year) for the privilege of doing all the work. Consequently, most members represent companies that are willing to pay fees and travel expenses, but there is always a small number of people who pay their own way.

During the 1990s, developers sing "Love Me Tender" to C++, and it becomes the object-oriented programming language by antonomasia[48]. Books like the first and

[37] https://www.pcjs.org/documents/magazines/msj/
[38] https://en.wikipedia.org/wiki/Develop_(Apple_magazine)
[39] https://en.wikipedia.org/wiki/Macintosh_Programmer's_Workshop
[40] https://deprogrammaticaipsum.com/scott-meyers/
[41] https://en.wikipedia.org/wiki/Curiously_recurring_template_pattern
[42] https://deprogrammaticaipsum.com/james-coplien/
[43] https://www.stepanovpapers.com/
[44] https://en.wikipedia.org/wiki/Standard_Template_Library
[45] https://en.wikipedia.org/wiki/CodeWarrior
[46] https://isocpp.org/faq
[47] https://dl.acm.org/doi/10.1145/1238844.1238848
[48] https://deprogrammaticaipsum.com/antonomasia/

second edition of Grady Booch[49]'s "Object-Oriented Analysis and Design with Applications", and the Gang of Four's[50] "Design Patterns" book, all featured code samples in C++.

But this short honeymoon moment, just like Lula's and Sailor's, would be over soon.

Mulholland Drive (2002)

The almost simultaneous releases of Java[51] and the World Wide Web[52] sent shockwaves throughout the industry. Software developers stopped focusing on desktop operating systems, much to the dismay and urgency of Microsoft, who promptly drowned the market with a bug-ridden web browser for almost a decade.

As explained[53] by Stroustrup,

> In October 1991, the estimated number of C++ users was 400,000. The corresponding number in October 2004 was 3,270,000. Somewhere in the early '90s C++ left its initial decade of exponential growth and settled into a decade of steady growth.

Java came bundled with the killer feature that stopped the meteoric ascension of C++: a garbage collector. Forget about those `new` / `delete` cycles: just allocate objects and leave them there, somebody will take care of cleaning up the heap for you.

It is not like you could not use garbage collectors with C++; there have been a few proposals[54] for such a thing. But the tight control of resources provided by manual memory management proved too much of a burden for some applications, and C++ never got a standard garbage collection system.

Under this pressure, the work on C++ became slower (at least, as seen from the outside), and seemingly more chaotic. After the release of the 1999 edition of the standard, work seemed to stop, while the world moved towards networking applications built on top of HTML and HTTP, applications that were better served by garbage-collected runtimes

[49] https://deprogrammaticaipsum.com/the-three-amigos-among-others/
[50] https://deprogrammaticaipsum.com/the-gang-of-four/
[51] https://deprogrammaticaipsum.com/java-the-programmer-environment-that-has-it-all/
[52] https://deprogrammaticaipsum.com/from-hypertext-to-spas-to-hypertext/
[53] https://dl.acm.org/doi/10.1145/1238844.1238848
[54] https://web.archive.org/web/20040207233813/http://www.hpl.hp.com/personal/Hans_Boehm/gc/

and technologies collectively referred to as LAMP[55].

The C/C++ community became obsessed, just like Betty Elms (or was it Diane Selwyn?), with Java the seductress, akin to Rita (or was it Camilla Rhodes?) As a result, the development of the language was apparently brought almost to a halt:

> "Silencio, no hay banda".

We just got a short update of the standard in 2003, and we would have to wait 8 more years for (finally!) a radical upgrade to the language.

This was a period during which developers starting exploring other options, like the functional paradigm, scripting languages such as JavaScript[56] or Ruby, or jumping on the concurrency wagon (the "Free Lunch is Over"[57], remember?) with languages like Erlang, Go[58], or Scala.

The Story of a Small Bug (2020)

Thankfully, C++11 arrived, even if when it did, many developers had walked away in dismay.

Stroustrup starts the preface of the 2014 massive 4th edition of his *magnus opus*[59] with the following observation:

> C++ feels like a new language. That is, I can express my ideas more clearly, more simply, and more directly in C++11 than I could in C++98. Furthermore, the resulting programs are better checked by the compiler and run faster.

The list of changes between C++98 and C++11 is, indeed huge, including move semantics, lambdas, `auto` and `decltype`, the curly-bracket initialization syntax, defaulted functions, `nullptr`, and so much more. Those who suffered (like the person writing these lines) the shenanigans of C++98 should definitely give a shot to C++ once again. According to Deitel & Deitel[60], the 4 big things of C++20 are: ranges, concepts, modules and coroutines. Can you recognize your C++ now?

---

[55] https://en.wikipedia.org/wiki/LAMP_(software_bundle)
[56] https://deprogrammaticaipsum.com/douglas-crockford/
[57] https://deprogrammaticaipsum.com/herb-sutter/
[58] https://deprogrammaticaipsum.com/rob-pike/
[59] https://www.stroustrup.com/4th.html
[60] https://deitel.com/c-plus-plus-20-for-programmers/

Of course, my opinion, as usual, is not the same as the rest of the industry. 2022 ended up being "the year of the C++ successor languages", according[61] to the ACCU. And successors we got: Rust[62], Zig[63], Carbon[64], Circle[65], Hylo[66], Vale[67]; how many more hype cycles of rewriting the world into yet another language do we need to go through? We are still stuck in the same tar pits[68]. For the anecdote, let us see what happened to @mcc[69] just a few weeks ago:

> So I switch from C++ to Rust because debugging C++ packaging and linking errors is the most painful thing in the world and then my Rust crates wind up building C++ packages and then the C++ packages fail to build because debugging C++ packaging and linking errors is the most painful thing in the world

But wait, there is more: as mentioned by Fireship[70] in a recent video[71], the Defense Advanced Research Projects Agency[72] is proposing to use a program called TRACTOR[73] to translate C and C++ code into Rust using AI and LLMs. Excellent idea. Let us trust our defense to an LLM. What could possibly go wrong?

Stroustrup designed C++ to help us build huge projects. How huge are they? Taking a look at some estimations gathered from various sources online, and even if "lines of code" is not always a suitable metric, we can be genuinely impressed.

Project	Estimated Lines of Code
Microsoft Windows	~50 million
Chromium	~45 million
Mozilla Firefox	~21 million
LibreOffice	~12 million

[61] https://accu.org/journals/overload/30/172/teodorescu/
[62] https://www.rust-lang.org/
[63] https://ziglang.org/
[64] https://en.wikipedia.org/wiki/Carbon_(programming_language)
[65] https://www.circle-lang.org/site/index.html
[66] https://www.hylo-lang.org/
[67] https://vale.dev/
[68] https://deprogrammaticaipsum.com/goodness-gracious-great-balls-of-mud/
[69] https://mastodon.social/@mcc/113805217959811803
[70] https://deprogrammaticaipsum.com/fireship/
[71] https://www.youtube.com/watch?v=v4H2fTgHGuc
[72] https://en.wikipedia.org/wiki/DARPA
[73] https://www.darpa.mil/research/programs/translating-all-c-to-rust

Project	Estimated Lines of Code
MySQL	~9 million
Blender	~6 million
Qt Framework	~5 million
Unreal Engine	~3 million

Yes, I think the task is fulfilled, Bjarne.

What many programmers do not realize is the following: modern C++ can be as safe as Rust. It can; it just requires humans to think[74], something that, in the age of the LLM, is becoming the greatest challenge of our time. If you do not believe me, ask Herb Sutter[75] for details.

The C++ galaxy is more vibrant than ever, with a new release of the programming language every 3 years (2011, 2014, 2017, 2020, 2023, and 2026 upcoming), solid support from an incredibly large array of vendors, extraordinary open-source compilers, a never-ending list of available tools... yet, we insist in sticking with old perceptions and misconceptions. Insert sad emoji face here.

Let us quickly review the state of the ecosystem. Need a package manager? vcpkg[76] by Microsoft and Conan[77] seem to lead the pack, but we also have Spack[78] for supercomputers[79], build2[80] inspired by Rust's `cargo`, with packages at cppget[81], Buckaroo[82], Cabin[83], Hunter[84], xrepo[85], and cpm[86]... made by and for CMake[87].

Speaking about CMake, what about build tools? Well, we have the veteran GNU Au-

[74]https://deprogrammaticaipsum.com/think/
[75]https://deprogrammaticaipsum.com/herb-sutter/
[76]https://vcpkg.io/en/
[77]https://conan.io/
[78]https://spack.io/
[79]https://en.wikipedia.org/wiki/Supercomputer
[80]https://www.build2.org/
[81]https://cppget.org/
[82]https://buckaroo.pm/
[83]https://github.com/cabinpkg/cabin
[84]https://github.com/cpp-pm/hunter
[85]https://xrepo.xmake.io/#/
[86]https://github.com/cpm-cmake/CPM.cmake
[87]https://cmake.org/

totools[88], Meson[89], Bazel[90], SCons[91], tipi.build[92], Soup[93], Pixi[94], Mamba[95], Conda[96], and BitBake[97].

And what other tools and libraries should we mention? cdecl[98]. Valgrind[99]. Boost[100]. Doxygen[101]. Catch2[102]. POCO[103]. A myriad of awesome header-only libraries[104]. Desktop GUI toolkits like Qt[105], wxWidgets[106], and JUCE[107]. Web frameworks like Cutelyst[108] (made with Qt), Drogon[109], libhttpserver[110], Lithium[111], or userver[112]. A lot of awesome[113] stuff.

And... yes. Lots and lots of string implementations, sadly. I agree with you, Bubba[114], initialization options abound. There are also plenty of cast operators: `dynamic_cast`, `static_cast`, `reinterpret_cast`, and `const_cast`, by the way. Bonus points if you know the differences among them. *Le sigh*. The most interesting things in life demand some effort, clearly.

Let us not drown from the incompetent perspective of those influencers disliking C++

[88] https://en.wikipedia.org/wiki/GNU_Autotools
[89] https://mesonbuild.com/
[90] https://bazel.build/
[91] https://scons.org/
[92] https://tipi.build/
[93] https://www.soupbuild.com/
[94] https://pixi.sh/latest/
[95] https://github.com/mamba-org/mamba
[96] https://docs.conda.io/en/latest/
[97] https://docs.yoctoproject.org/bitbake/
[98] https://cdecl.org/
[99] https://valgrind.org/
[100] https://www.boost.org/
[101] https://www.doxygen.nl/
[102] https://github.com/catchorg/Catch2
[103] https://pocoproject.org/
[104] https://github.com/p-ranav/awesome-hpp
[105] https://en.wikipedia.org/wiki/Qt_(software)
[106] https://en.wikipedia.org/wiki/WxWidgets
[107] https://en.wikipedia.org/wiki/JUCE
[108] https://github.com/cutelyst/cutelyst
[109] https://github.com/drogonframework/drogon
[110] https://github.com/etr/libhttpserver
[111] https://matt-42.github.io/lithium/
[112] https://userver.tech/
[113] https://awesomecpp.com/
[114] https://www.reddit.com/r/ProgrammerHumor/comments/8nn4fw/forrest_gump_learns_c/

on Reddit. We do not need yet another programming language claiming to be the next C++; we can learn to use the one we have right now, applying the best practices[115] we can find.

Maybe it is time to revisit C++ with renewed eyes and also, by the way, to find out who killed Laura Palmer[116]. Because, seriously.

Cover photo by Jeremy Yap[117] on Unsplash[118].

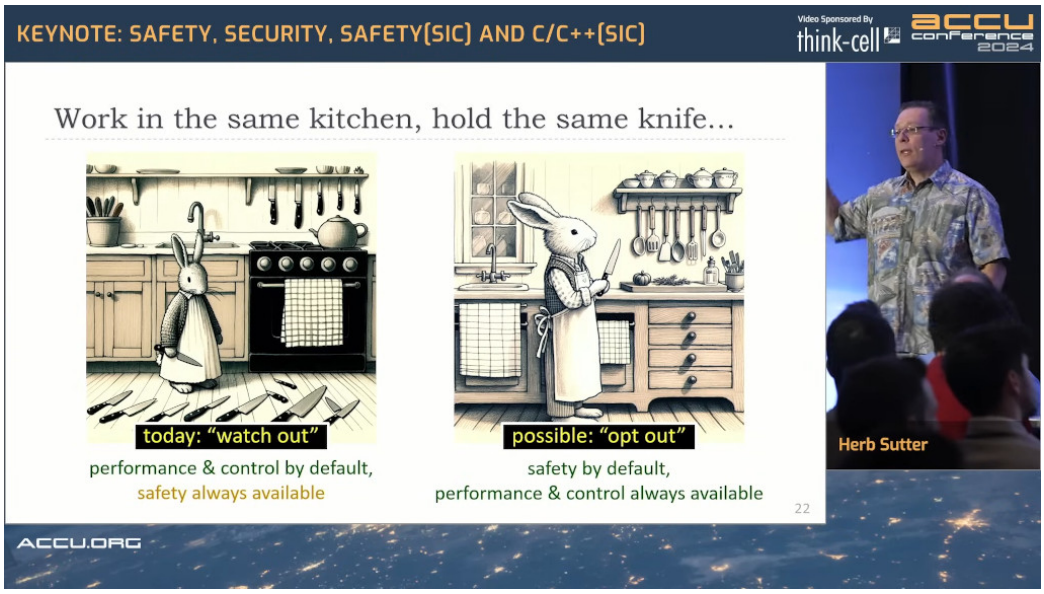
[115]https://isocpp.org/wiki/faq/coding-standards

[116]https://en.wikipedia.org/wiki/Laura_Palmer

[117]https://unsplash.com/@jeremyyappy?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

[118]https://unsplash.com/photos/turned-on-projector-J39X2xX_8CQ?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

Herb Sutter



By Adrian Kosmaczewski, February 3rd, 2025

C++ is under attack. Some argue that it is the language's fault, with those pointers and rules and complexity and undefined behavior, and try once and again to develop a "C++ killer" language, with various degrees of success. Others (rightfully so) defend the language (and its community) by acknowledging its history, its flaws, and proposing ways forward. The former group makes headlines in Reddit and Medium. Instead, the Vidéothèque entry of this month tells the story of a prominent member of the latter group, a certain Herb Sutter[119].

The video in question is a keynote Herb gave at the ACCU 2024 conference: "Safety,

[119]https://herbsutter.com/

Security, Safety (sic) and C/C++ (sic)"[120]. The ACCU conference describes itself[121] as

> ...one of the longest running developer conferences in the world. Established by the ACCU organization in 1997 it has always been open to the whole community and at the forefront of what's happening in the world of software development.
>
> Originally focused on C and C++, that community has expanded to include other language ecosystems, and you should expect to be exposed to content on C#, D, F#, Go, JavaScript, Haskell, Java, Kotlin, Lisp, Python, Ruby, Rust, Swift and more. There are always sessions on TDD, BDD, and how to do programming right.

The name of Herb Sutter has appeared several[122] times[123] in the pages of this magazine, and with reason: he is the author of the often-cited "Free Lunch is Over" March 2005 article on Dr. Dobbs' Journal[124] (of which, thankfully, the Internet Archive has taken snapshots of page 1[125], page 2[126], and page 3[127]). He is a remarkable member of the C++ committee[128], and a former Microsoft[129] employee where he worked as lead architect of (you guessed it) the C++ compiler. He is also author and co-author of various books in the field. Last but not least, Herb is currently working on a project called cppfront[130], where he explores mechanisms to make C++ a safer language, without ditching 40 years of heritage in the process.

TL;DR: Herb knows a thing or two about C++.

In his keynote, Herb starts by acknowledging a series of sins (maybe the word is too

---

[120] https://www.youtube.com/watch?v=EB7yR-1317k

[121] https://accuconference.org/

[122] https://deprogrammaticaipsum.com/the-age-of-concurrency/

[123] https://deprogrammaticaipsum.com/breaking-the-3-ghz-barrier/

[124] https://deprogrammaticaipsum.com/dr-dobbs-journal-of-computer-calisthenics-and-orthodontia/

[125] https://web.archive.org/web/20120723130911/https://drdobbs.com/web-development/a-fundamental-turn-toward-concurrency-in/184405990

[126] https://web.archive.org/web/20120827054313/http://www.drdobbs.com/web-development/a-fundamental-turn-toward-concurrency-in/184405990?pgno=2

[127] https://web.archive.org/web/20120827101613/http://www.drdobbs.com/web-development/a-fundamental-turn-toward-concurrency-in/184405990?pgno=3

[128] https://isocpp.org/std/the-committee

[129] https://deprogrammaticaipsum.com/where-does-microsoft-want-to-go-today/

[130] https://github.com/hsutter/cppfront

much?) committed by the C++... committee. First and foremost, the language was designed to put performance ahead of safety. This is a historic fact, and as such it carries into the present with a lot of consequences, including the much dreaded "C/C++" moniker, thereby providing the fertile ground for a long series of safety issues making the headlines today.

Second, yes, the community has adopted the ostrich's attitude, and has repeatedly tried to bury its head in the sand to avoid looking at the problem. The consequence is easy to see: C and C++ are both, despite their immense contributions during the past 40 years, unjustly thrown under the bus and almost completely ditched by a whole generation of software developers blinded by the word "Rust" on the title of every popular podcast episode.

Objection, your honor.

Let us be very clear: what Herb does in this talk is nothing short of uncommon, and strictly follows our wish for less evangelization, and more honestization, which we expressed in a previous article[131] on this magazine. Mature communities, just like mature individuals, are able to take ownership of their flaws, and grow.

Case in point: the week Herb gave this talk at ACCU, Rust had disclosed a very severe vulnerability; Herb does not talk about it specifically, to (very cleverly) avoid having online hordes dismiss his talk as a bashing of Rust and a glorification of C++. The truth is more subtle than that: the Rust culture treats bugs more seriously than C++! For example, CVE-2022-21658[132], is a problem similar to its buggy C++ equivalent `std::filesystem::remove_all`, but the latter (suprise!) has no CVE report.

For Herb, in this case, the key task consists in turning around the philosophical and cultural focus of C++: instead of guaranteeing performance first and making safety optional, he proposes to embrace safety profiles[133] and to *guarantee safety first, and making performance optional*.

Repeating it loudly for the people in the back: the problem is not technical, but cultural.

(For more information about the WG21 safety profiles' framework, there is an interesting paper[134] by Stroustrup & Dos Reis called "Design Alternatives for Type-and-

[131] https://deprogrammaticaipsum.com/less-evangelization-more-honestization/
[132] https://nvd.nist.gov/vuln/detail/CVE-2022-21658
[133] https://github.com/BjarneStroustrup/profiles
[134] https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p2687r0.pdf

Resource Safe C++".)

C++ was born in an era of underpowered PCs, and it largely succeeded as a remarkable language that provided a very high-level of abstraction for developers, all while generating blazingly-fast code for those same sluggish computers. The tables have turned, and in our modern world, filled with bad actors and cyber warfare scenarios, C++ will have to evolve as well: its focus has to change. Google is worried[135] about it. Apple is worried[136] about it. The White House is worried[137] about it. The NSA is worried[138] about it. Consumer reports is worried[139] about it.

You get the idea.

The goal, then, is to make C++ approach safety parity with other languages such as Go, Rust, C#, or Python, in four specific areas: *type safety, bounds safety, initialization safety, and lifetime safety.* Just like with Rust, giving C++ developers a mode that improves safety by default won't save them, but it will help them.

The situation is dire. Our modern world infrastructure is running on insecure software, and examples of breaches and vulnerabilities abound. Herb makes four major points that stand out in his keynote:

1. In the MITRE Corporation Common Weakness Enumeration (CWE) 2023 ranking[140], we see 3 vulnerabilities directly related to C++: out-of-bounds write, use after free, and out-of-bounds read. Those are the targets to eliminate at this point. However... *caveat emptor*: the most important attacks in the past few years are not related to memory safety: neither SolarWinds[141], Log4Shell[142], DarkBeam[143], nor the XZ utils backdoor[144] are.

[135]https://research.google/pubs/secure-by-design-googles-perspective-on-memory-safety/

[136]https://www.apple.com/newsroom/pdfs/The-Continued-Threat-to-Personal-Data-Key-Factors-Behind-the-2023-Increase.pdf

[137]https://web.archive.org/web/20240227002908/https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf

[138]https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF

[139]https://advocacy.consumerreports.org/wp-content/uploads/2023/01/Memory-Safety-Convening-Report-1-1.pdf

[140]https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html#tableView

[141]https://en.wikipedia.org/wiki/2020_United_States_federal_government_data_breach

[142]https://en.wikipedia.org/wiki/Log4Shell

[143]https://cybernews.com/security/darkbeam-data-leak/

[144]https://en.wikipedia.org/wiki/XZ_Utils_backdoor

2. Speaking about which, Herb quotes an an article[145] by Bruce Schneier about the XZ Utils backdoor attack where he states a chilling conclusion:

   > I simply don't believe this was the only attempt to slip a backdoor into a critical piece of Internet software, either closed source or open source. Given how lucky we were to detect this one, I believe this kind of operation has been successful in the past. We simply have to stop building our critical national infrastructure on top of random software libraries managed by lone unpaid distracted—or worse—individuals.

3. Moving safety "to the left", closer to developers and their "inner loop", is a fundamental task, and one shown by Rust to be a major step towards overall code safety. Herb quotes David Chisnall and his January 2024 message "The Case for Rust (in the base system)"[146] on the freebsd-hackers mailing list:

   > Between modern C++ with static analysers and Rust, there was a small safety delta. The recommendation was primarily based on a human-factors decision: it's far easier to prevent people from committing code that doesn't compile than it is to prevent them from committing code that raises static analysis warnings. If a project isn't doing pre-merge static analysis, it's basically impossible. Between using modern C++ (even just smart pointers and ranges) and C, there is an enormous safety delta.

4. Herb debunks a common myth: no major programming language is formally provably type-safe! Yup, not even Java or C# (both with use-before-init and use-after-dispose problems), or Rust or Go, which come bundled with sanitizers, but people not always use them. Remember, it is not the types[147] that will help you:

   > A programmer with an expensive type system does not write better programs, any more than a novice guitarist becomes a stadium-grade shredder by buying a fancy guitar.

5. Last but not least, Herb acknowledges that Rust is the only major language with strong concurrency safety guarantees... but did you know that 30 to 50% of all Rust crates use some unsafe code at some point or another, versus just 25% of Java libs? Things are not always what they seem.

To further his defense, Herb provides a single, fundamental, modern code example that

[145] https://www.schneier.com/blog/archives/2024/04/xz-utils-backdoor.html
[146] https://lists.freebsd.org/archives/freebsd-hackers/2024-January/002876.html
[147] https://deprogrammaticaipsum.com/it-is-not-the-types-that-will-help-you/

shows the levels of evolution[148] reached by C++ today. Hang tight:

```
auto a = make_unique<Widget>();
```

Boom. The code above, compatible with C++14 and later, is **type-safe, bounds-safe, initialization-safe, and lifetime-safe by default**. Now go back and re-read the previous sentence, twice.

Even better, the code above uses type inference, a sign of the times[149]. And yes, it is compatible with the most modern C++ compilers available today. Maybe the problem is not so much C++ being unsafe, but the way C++ is taught in schools, and the way it used by most organizations. Maybe it is time to review your coding guidelines? Just saying.

Bjarne Stroustrup agrees[150] with Herb:

> That (report) specifically and explicitly excludes C and C++ as unsafe. As is far too common, it lumps C and C++ into the single category C/C++, ignoring 30+ years of progress. Unfortunately, much C++ use is also stuck in the distant past, ignoring improvements, including ways of dramatically improving safety.

Herb ends the first part of his keynote with some important calls to action, which we repeat here for the sake of sanity and correctness:

- Use language static analyzers and sanitizers (yes, even if you say "I'm using a safe language!")
- Keep your tooling updated.
- Secure your software supply chain as much as possible.
- Use modern package managers.
- Generate and store SBOMs (Software Bills of Materials) together with your build artifacts.
- Don't store secrets on code (duh! Yet...)
- Configure servers properly.
- Keep non-public data encrypted.
- Keep your threat modeling up-to-date, because attackers will find new weak spots all the time.

[148] https://deprogrammaticaipsum.com/the-great-rewriting-in-rust/
[149] https://deprogrammaticaipsum.com/the-truce-of-type-inference/
[150] https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/p2739r0.pdf

Herb ends this talk with a short of his cppfront[151] project, in particular the current efforts to get faster compile-time support for regular expressions, highlighting the contributions of Hana Dusíková[152].

(By the way, the name "cppfront" sure brings memories of "cfront"[153], the first compiler for a C++ predecessor called "C with Classes" written by Bjarne Stroustrup in 1983.)

We can laugh at C++ developers all we want, but the committee and its surrounding community are doing an admirable job, and have done it for more than 35 years. C++ is robust, cross-platform[154], safe, has a vibrant ecosystem, very stable tooling, and has a brilliant (and safe) future ahead.

For more information about the current state of the "safe programming language wars" (let us throw some fuel to the flames, shall we?) we can recommend "A New Era for C and C++? Goodbye, Rust?"[155]. If you are new to C++ and would like to write safe code, avoid these common "31 nooby C++ habits you need to ditch"[156] and change your perspective.

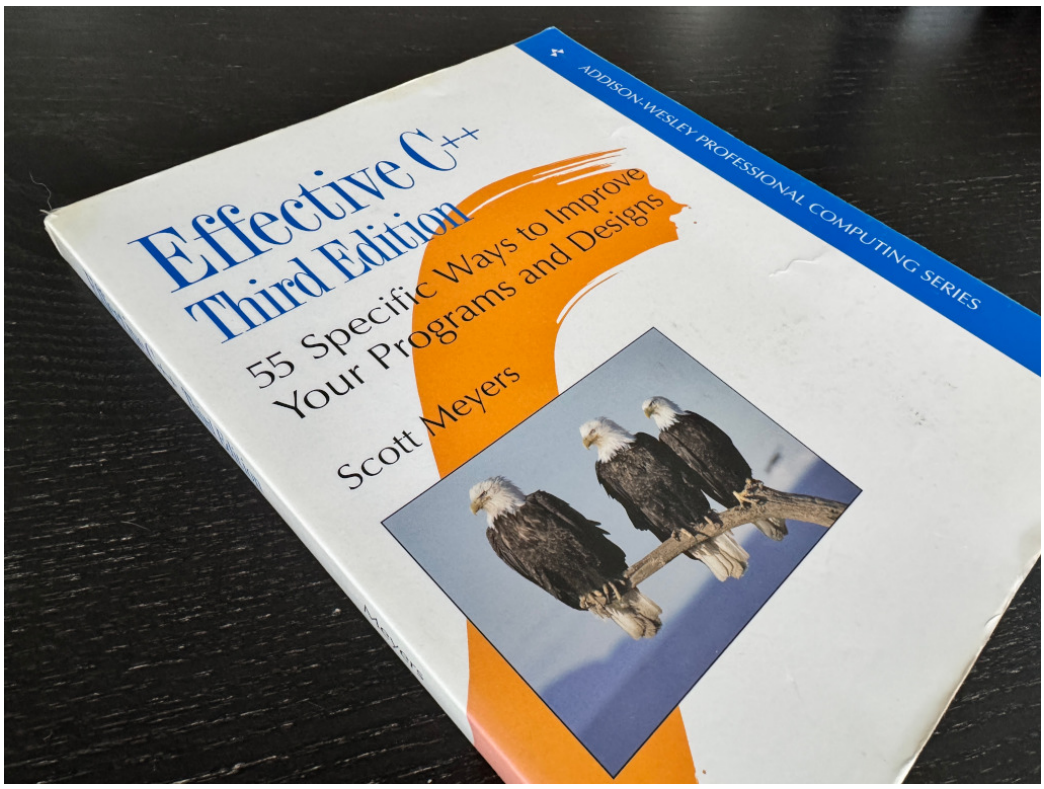
If you are interested in the history of C++, we suggest watching this lecture by Bjarne Stroustrup, "The Design of C++"[157] recorded in March 1994, providing an interesting view not only on the language, but also a candid perspective on its creator.

Watch this month's Vidéothèque article, "Safety, Security, Safety (sic) and C/C++ (sic)"[158], by Herb Sutter[159], on YouTube.

Cover snapshot chosen by the author.

[151] https://github.com/hsutter/cppfront
[152] https://compile-time.re/
[153] https://en.wikipedia.org/wiki/Cfront
[154] https://deprogrammaticaipsum.com/dr-dobbs-and-the-deathly-cross-platform-app/
[155] https://www.youtube.com/watch?v=V_QAJAhbH9A
[156] https://www.youtube.com/watch?v=i_wDa2AS_8w
[157] https://www.youtube.com/watch?v=69edOm889V4
[158] https://www.youtube.com/watch?v=EB7yR-1317k
[159] https://en.wikipedia.org/wiki/Herb_Sutter

Scott Meyers



By Adrian Kosmaczewski, February 3rd, 2025

Around 20 years ago I found a job as a C++ developer. My new employer provided me the first day with a PDF file defining the very strict and mandatory set of guidelines to be followed for the production of code in the organization. These rules can be summarized as follows: do not use the Standard Template Library; do not use templates; and do not

use multiple inheritance. If you are a C++ developer reading the previous phrase, I hope you can understand the dismay I felt while reading that. If you are not a C++ developer, suffice to say that to this day I do not understand why would anyone choose to use C++ *without* those features.

A programming language is called a "language" for a reason: it contains a set of grammar rules and a dictionary of reserved words, which can be put together in certain ways to produce a particular semantic. No French poet would consciously avoid Alexandrines[160] other than for stylistic purposes, and no Latin poets stayed away from Saturnians[161] when the time came. Those forms are tools, and even if admittedly, human languages are much less mathematical than programming ones, they are all part of a poet's arsenal, and rightfully so.

I have long meditated about the reasons behind the conscious choice of forbidding certain forms of C++ in this company's product. I have concluded that it had less to do with the explicit reasons they gave in their guidelines document (readability, maintainability, some other ability) and more with the fact that in 2006 C++ was stuck in a limbo. The standard in vogue at the time, the 2003 edition, which was a bug-correcting revision of the 1999 one, was quickly losing its prominence as an application programming language, replaced by other, more hyped, languages, almost all at a higher level in the abstraction hierarchy, and featuring a proverbial garbage collector: Java[162], C#, Scala, Erlang, Ruby, and Python[163].

(It must be said at this point, that the software product in question was a business application written for Windows, whose development had originally started in the mid-1990s, and which used object-oriented programming features extensively to model various business entities.)

We must remember that 2006 was also the year of Bruce Tate's "Beyond Java"[164]; a time of experimentation and possibilities. JavaScript had all of a sudden "good parts"[165]; Rails was taking everyone by surprise. The OOP Hype Cycle[166] was getting stuck in its trough of disillusionment. Last but not least, it was also the time of "Writing Se-

[160] https://en.wikipedia.org/wiki/Alexandrine
[161] https://en.wikipedia.org/wiki/Saturnian_(poetry)
[162] https://deprogrammaticaipsum.com/java-the-programmer-environment-that-has-it-all/
[163] https://deprogrammaticaipsum.com/the-state-of-python-in-2021/
[164] https://www.oreilly.com/library/view/beyond-java/0596100949/
[165] https://deprogrammaticaipsum.com/douglas-crockford/
[166] https://deprogrammaticaipsum.com/the-hype-cycle-of-oop/

cure Code"[167] being a mandatory reading at Microsoft, with the public opinion openly blaming C and C++ and their goddamn pointers for pretty much every software security disaster making headlines.

(To make a long story short, my employer chose to move to Java shortly after, for the next version of their flagship application. This was a rewrite that proved costly, not only in financial but also, and painfully enough, in human terms. But that is material for another story.)

Back to the coding guidelines, the explicit choices therein (no multiple inheritance, no STL, no templates) made the codebase unwieldy, heavy, prone to repeated code, and hard to evolve. In particular, avoiding multiple inheritance even prevented the team from using pure abstract classes with virtual methods (akin to Java and C# interfaces, or Objective-C and Swift protocols) which are, by all means, a fantastic abstraction and design mechanism. I could digress for many more pages; in short, and in hindsight, I do consider such choices a sad mistake.

During the year and a half I spent in that company, however, I had the immense chance of discovering the outstanding work of Scott Meyers[168]. To say that his book "Effective C++: 55 Specific Ways to Improve Your Programs and Designs" had a strong effect in me is an understatement. I still own my copy of the second edition (published in 2005), and to this day, even though C++ has evolved in ways that nobody could have predicted back in 2006 (and thankfully so!) it still remains a staple of my library, and the *go-to* book whenever I have to work on C++ code (which is not that often, alas). Last but not least, this was the book that opened the door to my university degree[169] in 2008.

Scott (I hope he does not mind me calling him by his first name) is one of my all-time heroes in the world of programming. To give you an idea why, I will just quote a 2003 panel where he was asked about how to interview programmers[170], and his observations are golden:

> I hate anything that asks me to design on the spot. That's asking to demonstrate a skill rarely required on the job in a high-stress environment, where it is difficult for a candidate to accurately prove their abilities. I think it's fundamentally an unfair thing to request of a candidate.

[167] https://deprogrammaticaipsum.com/microsofts-writings-on-security/
[168] https://en.wikipedia.org/wiki/Scott_Meyers
[169] https://akos.ma/blog/master/
[170] https://www.artima.com/articles/how-to-interview-a-programmer

Yes, Scott. This magazine wholeheartedly[171] agrees[172] with you.

But back to the book. There is one precise reason why I admire (and still consult) "Effective C++" 20 years after its publication; C++ might have changed in the past two decades, yet the tenets that define the spirit of the language (most of which are described in the book) are still the same. I have grown a fondness for boring technology, proven time and again, and backed by a solid ecosystem around it: C++ ticks all of these boxes.

"Effective C++" is, in essence, a book quite similar to Robert L. Glass[173]' "Facts and Fallacies of Software Engineering": a series of very specific guidelines, stating clearly what to do (and most importantly, what *not* to do) in various situations and conundrums. In this case, regarding the essence of C++.

Let us see some examples, starting with Item 1, "View C++ as a federation of languages":

> Today's C++ is a *multiparadigm programming language*, one supporting a combination of procedural, object-oriented, functional, generic, and metaprogramming features.

Yes, functional features, too. C++ is a very complex beast, offering unprecedented levels of flexibility and power. Why restrict oneself?

The power of C++ often comes hidden, however, and as Item 5 explains, developers must "Know what functions C++ silently writes and calls".

> When is an empty class not an empty class? When C++ gets through with it. If you don't declare them yourself, compilers will declare their own versions of a copy constructor, a copy assignment operator, and a destructor.

Developers new to C++ are simply unaware of such rules[174]; they are certainly not obvious at first sight, and can lead them to write code that simply does not behave as expected. In the same vein, Item 7 reminds you of making destructors virtual in polymorphic class families, while Item 9 tells you not to call virtual functions in constructors nor destructors.

Scott dives into the famous RAII[175] principle in the third chapter, starting with Item 13,

[171] https://deprogrammaticaipsum.com/do-not-ask-me-about-how-interviewing-works/
[172] https://deprogrammaticaipsum.com/tales-of-the-interview/
[173] https://deprogrammaticaipsum.com/robert-l-glass/
[174] https://en.wikipedia.org/wiki/Rule_of_three_%28C++_programming%29
[175] https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization

"Use objects to manage resources". Then it uses chapters 4 and 6 to explain code design techniques, arguably useful not only for C++ code but in so many other languages. The quintessential example? "Item 22: Declare data members private". Duh.

Item 31, "Minimize compilation dependencies between files" has the intended effect of speeding up the compilation of C++ projects, a fact often neglected in large codebases. For the anecdote, the system of my employer consisted of half a million lines of code, and took around three hours to compile on a standard single-core PC of 2006, using the current version of the Microsoft C++ compiler. One of my colleagues literally spent a weekend applying Scott's best practices, and cut down the compilation time... in half.

Finally, Item 53 is a pet peeve of mine: "Pay attention to compiler warnings". Although to be honest, I would have not only suggested to pay attention to them, but literally to remove them altogether. I also remember applying this principle to Objective-C code[176] back when I was earning a living as an iPhone app developer. (In general, I considered[177] knowing C and C++ as a very important part of the upbringing of pre-Swift iOS developers.)

In hindsight, I have the impression that giving a copy of Scott Meyers' "Effective C++" to each one of my colleagues would have been a better choice than penning an overly restrictive *ad hoc* document. These days, the C++ Core Guidelines[178] live document by none other than Stroustrup and Herb Sutter[179] (the last update of which, at the time of this writing, happened October last year) took over the role that "Effective C++" played 20 years ago, becoming the most authoritative coding guidelines document available today.

Scott decided to retire from the C++ world in 2015[180], so let this article be a belated "thank you" from a developer who benefitted tremendously from his outstanding contributions to the field. We need more people like him.

If you want to continue your exploration of C++, I can recommend the eponymous book[181] by Stroustrup himself, or if you are in a hurry, its shorter version[182]. A decidedly more advanced reading, Andrei Alexandrescu's "Modern C++ Design: Generic

[176] https://akos.ma/blog/objective-c-compiler-warnings/
[177] https://akos.ma/blog/how-knowing-c-and-c-can-help-you-write-better-iphone-apps-part-1/
[178] https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines
[179] https://deprogrammaticaipsum.com/herb-sutter/
[180] http://scottmeyers.blogspot.com/2015/12/good-to-go.html
[181] https://www.stroustrup.com/4th.html
[182] https://www.stroustrup.com/tour3.html

Programming and Design Patterns Applied"[183] is one of the most extensive works on template metaprogramming ever written, a book highly recommended by Scott Meyers himself. The first chapter of the (often-mentioned in the pages of this magazine) book "Masterminds of Programming"[184] by Federico Biancuzzi and Shane Warden contains an extensive interview of Bjarne Stroustrup.

If you are interested in the genesis and history of the language, you will be happy to learn that C++ is the *only* programming language to have been featured in three consecutive editions of the famous "History of Programming Languages" (HOPL) conferences, created by the late Jean Sammet[185]. The three papers, all written and delivered by Stroustrup himself, are available online: "A history of C++: 1979–1991"[186] (HOPL II, 1993), "Evolving a language in and for the real world: C++ 1991-2006"[187] (HOPL III, 2007), and "Thriving in a crowded and changing world: C++ 2006–2020"[188] (HOPL IV, 2021). At the current rate, HOPL V will take place in 2035, so you have time to read them all.

C++ is a complex yet rewarding language, and it is time for software developers to understand that it has changed for good, that it has all the traits of a "modern" language, and that it definitely deserves a place in the programming arsenal of each one of us in this industry.

Cover photo by the author.

[183] https://en.wikipedia.org/wiki/Modern_C%2B%2B_Design
[184] https://www.oreilly.com/pub/pr/2277
[185] https://deprogrammaticaipsum.com/jean-sammet/
[186] https://dl.acm.org/doi/10.1145/154766.155375
[187] https://dl.acm.org/doi/10.1145/1238844.1238848
[188] https://dl.acm.org/doi/10.1145/3386320