C++ - How it Got Here, Where it's Going

With contributions from

Matthias Kalle Dalheimer, Jon Kalb, Ivan Čukić and Jens Weller

KDAB

We talked to a panel of C++ experts, the owner of a leading software expertise company, and C++ evangelists

The C++ programming language is considered by many to be the most powerful language on the planet. It is used for operating systems, web browsers, games, embedded software, autonomous cars, medical technology, and many other applications. Major companies such as Facebook, Google, Amazon, and many others rely on C++ to run their data centers.

Since its inception four decades ago, C++ has expanded many times with performance, efficiency, and flexibility of use as its main design highlights. Its most recent release, C++20, became generally available in December 2020.

"You can go as low as the hardware or as high as the clouds"

We talked to a panel of C++ experts, evangelists, and devotees from both sides of the pond to understand a little more about C++ – its current position in the development community, its history, and its future.

Overview

C++ is a language that has evolved over decades; today it gives programmers a lot of semantically powerful, yet often syntactically arcane tools. As such, it isn't particularly simple for new programmers to learn. Languages have come along (such as Java, Python, and Ruby) that have cherry picked some C++ features, while simplifying, abstracting, or dropping others. These newer, purportedly easier-to-use languages have taken some of C++'s market share and mind share – but many developers have realized that they didn't always get the functionality they needed from these newer competitors.

For Ivan Čukić, the author of *Functional Programming in C++*, the main benefit of C++ is that it gives developers enormous power. "You can go as low as the hardware or as high as the clouds," he explains. "It's probably the only language where you can write really low-level code and simultaneously develop using functional programming idioms."

The peaks and valleys of C++ popularity

C++ was originally developed by Bjarne Stroustrup at Bell Labs and AT&T for internal use. After its first commercial release in 1985, for a period of time C++ was the leading candidate for developing new software paradigms such as object-oriented

As other languages popped up, grew, and died, C++ development and standardization continued its steady pace

programming and generic programming using templates. Unlike the academic languages it emulated, C++ could be used to develop industrial-grade software. But without a universally adopted language standard through the 1980s and 90s, compiler vendors continually introduced incompatible features and platform-specific experiments. This made the language a moving target, making it hard for developers to move code and expertise between environments.

In 1998, that was helped with the release of the C++98 language standard. However, a decade of fighting compiler incompatibilities and bolted-on behaviors had taken its toll. Despite the standardization effort, C++, as Ivan says, "seemed stagnant for a long time", with a decline in C++ interest in public forums while other, newer languages got more attention. But the continual developments and improvements in C++ hadn't stopped during that time, and new developments in technology put its strengths on a path for a major rediscovery and renaissance with the 2011 release, as we'll see.

C++ seemed stagnant for a long time with a decline in C++ interest in public forums while other, newer languages got more attention.

C++ invisibility – a business owner's perspective

As other languages popped up, grew, and died during these early days, C++ development and standardization continued its steady pace, supporting language growth that was highly consistent across platforms. This stability spawned mature C++ frameworks and libraries that grew far beyond their initial developer contribution. One of these was Qt, originally a proprietary GUI framework that eventually grew to a widely supported open-source project, aiding many aspects of cross-platform application development.

Matthias Kalle Dalheimer, was one of the first programmers to take up Qt and even wrote the first Qt manuals some twenty years back. Along with many other languages and frameworks, "since around 2011, the Qt framework supports two implementation languages for application writers," he says. "It has C++ for the back-end and a choice between C++ and QML, which is essentially JavaScript-based, for the front-end." The simplicity of the JavaScript language made it easier for Qt developers to build GUIs, reducing the C++ "scare" factor. However, Qt's support for both C++ and QML now meant that developers could often opt to implement things in either language.

"This told me that there was an ecosystem of C++ developers out there, but it was essentially invisible..."

It was interesting to see the choices people made back then with Qt's two languages. Explains Kalle: "Many times, QML got implemented too far into the back end. Algorithmic bits of code for speed and energy efficiency that should have been implemented in C++ were implemented in JavaScript simply because a company had more JavaScript experience." This led to loss of efficiency and the performance that only C++ can give.

His company, KDAB, often needed to educate customers about where to draw the line – at what point to hand over between the two languages. At times they also suggested going with C++ for an entire project. The problem, at that time, was in finding C++ developers, especially in Europe. Few university curriculums used C++ as an implementation language and typically the vast majority of young developers entering the workforce were recent graduates. The only time they had a lot of C++ experience was if they had used it in open-source projects – such as through the open-source project KDE, for which Kalle was also a co-founder.

"Algorithmic bits of code for speed and energy efficiency that should have been implemented in C++ were implemented in JavaScript simply because a company had more JavaScript experience"

Kalle knew C++ was being used as an implementation language in projects as widely varied as browser engines and high-performance finance computing. "This told me that there was an ecosystem of developers out there," he says. "But it was essentially invisible because at the time there were hardly any C++ conferences, few books published on the language, podcasts were rare, and there was little on YouTube." This is what led Kalle to jump at the opportunity to support the initiative from Jens Weller when he approached him to sponsor a new conference Jens was creating in Europe called Meeting C++.

Giving conferences the spark

Jens had been inspired by what was going on in the US, where the situation was similar; there had been very little available on the conference circuit for C++ developers before 2011. One large software development conference that primarily addressed C++ programmers had recently closed down. Another conference, BoostCon, was primarily focused on Boost, a popular C++ library, but this was the closest thing to a general C++ conference at the time.

Jon Kalb had approached the BoostCon organizers with the idea to refocus the conference on C++, rename and grow it, and they had agreed. The result was C++Now, which had an immediate *the C++ community went from almost nothing in 2011 to having a conference in almost every major country today*

success in its first year. It was at this event that Jens decided, "There needs to be a conference like this in Europe and I'm going to be the one to do it." To everyone's astonishment, Jens pulled it off at first try. The conference he created in 2012, which KDAB sponsored, pulled in even more attendees than the US had.

Back in the USA, Jon began to reconsider what he was doing with C++Now. His original vision had been to grow the Boost conference into a large mainstream C++ conference. But he realized that if his efforts were unsuccessful, BoostCon could be destroyed. So, he decided to start from scratch, and that is how *CppCon* was born. Luckily for the C++ community, both conferences now co-exist.

Both Jon and Jens understood that a new generation would be needed that would be able to step into the footsteps of existing C++ experts who would, one day, retire.

Invigorating the C++ community

Meanwhile, along with the success of *Meeting C++* in Europe, the focus became more than just about conferences. As Jens explains, "When I talked myself into doing something in Europe, I realized we needed a long-term perspective for C++." He was looking for a way to teach the new standard to people who wanted to learn C++. "Not only proper conferences were needed, but a whole community to support talent and let it grow." So that's what he set out to build. Both Jon and Jens understood that a new generation would be needed that would be able to step into the footsteps of existing C++ experts who would, one day, retire.

Largely thanks to their efforts, but also because people were really hungry for this kind of community, the C++ community went from almost nothing in 2011 to having a conference in almost every major country today, with many regional conferences and countless smaller Meetups around the world.

The C++ revival

The C++ language experienced a huge revival after the C++11 standard came out in 2011. The resulting growth of these community events since then reflects this injection of excitement, but it doesn't explain all of it.

As Ivan pointed out, for 13 years, between 1998 to 2011, it had looked like C++ was essentially stagnant. "But the C++ Standards Committee hadn't been sitting idle", says Jon. Rather, they

Once people started carrying around "these little devices called mobile phones", the importance of battery life was realized

were absorbing all of the things going on in modern software development and adding them to the language, ready for the next release. "In 2011 when the new C++ came out," Jon explains, "It was like a brand-new language that just happened to be completely backwards compatible with what we've been using for decades." What's more, between the C++98 and the C++11 release, the number of pages in the standard had doubled to include a huge number of new features like Lambda expressions, range-based for loops, multithreading, and much more. This clearly helped bring interest back to C++ and showed the community that C++ had reestablished itself as a modern and thoughtful language, yet it hadn't lost its performance edge.

They began to realize that there is no language that can compare to C++ as far as performance and abstractions goes – not C, not Java, not any other language

Technology drivers

In the book he co-authored, *C++ Today: The Beast is Back*, Jon talks about how, once people started carrying around "these little devices called mobile phones", the importance of battery life was realized. At the same time, big server farms were discovering that power efficiency was critical since their biggest expense was electricity. As he says, "Developers were also finding that a lot of programming languages wasted CPU cycles and power to do the same thing that C++ could do in half the time."

Ivan adds: "As people started caring about the performance of their software, they began to realize that there is no language that can compare to C++ as far as performance and abstractions goes – not C, not Java, not any other language." An awareness tipping point came when the electricity cost for low performance software exceeded the cost of hiring developers to improve performance.

Performance isn't just speed

There was a feeling in the 2000s that desktop computers were so powerful you didn't really need to worry about performance. Jon reminisces about an old desktop joke, "How fast does a cursor need to blink when you're sitting there with your word processor?"

But these days, performance of a fully loaded desktop machine isn't really the issue. The low end of computing is now everywhere – smartphones, embedded and IoT devices, and wearables – while beyond the desktop is dominated by cloud computing and storage. Now, optimal performance isn't needed for speed as

It’s absolutely not unusual for us to be confronted with gigantic code bases – millions of lines of code tested for decades. You don’t want to throw all that away.

much as it is for power, either to enable longer life on batteries or to keep server farm electricity bills manageable.

Legacy matters

Another reason for the resurgence of interest in C++ is the recognition that legacy matters. About this, Jens says: “If you’re a startup and have the luxury to implement everything new from scratch, that’s one thing. However, there are many places where code bases have existed for ten or twenty years – you’re not going to replace and rewrite all of this legacy code.”

There are many places where code bases have existed for ten or twenty years – you’re not going to replace and rewrite all of this legacy code

Kalle agrees: “It’s absolutely not unusual for us, as a consulting company, to be confronted with gigantic code bases – millions of lines of code that have been tested for decades. You don’t want to throw all that away. We’ve seen so many long forgotten bugs reoccur when massive code bases are rewritten so we understand that this legacy can be a tremendous asset. While you certainly want to improve and modernize legacy code and get it ready for the next decennium, it’s not something to throw away and just replace with the latest language-du-jour.”

Rise of the upstarts

C++ isn’t alone in its quest for performance. There are newer languages like Rust and Dart that are vying for attention as languages that can support high-level abstractions but also offer low-level access and performance.

Jens explains: “From 2010 to 2012 we were in the sweet spot where you could no longer scale by hardware. Around 2015, you started to be able to use the cloud, but it would still cost you a lot. This is where the new languages came from. They can focus on performance not just in the embedded space, but also in mobile devices.” Nonetheless, Jens likens this to an ocean food chain with C++ at the top. Whilst there will always be some start-ups that have the luxury to implement everything in these new, easier to learn languages, he doubts they will ever really become competitive. C++ has become stronger and stronger as it standardizes and it has several implementations, unlike other languages, which are limited to one. “Haskell, for example, *C++ allows you to mix and match paradigms, gradually evolving your code base on your needs and your schedule*

has always been popular to look at, but it's never become mainstream," he says. "I'm not worried."

In order to keep abreast of where the industry is moving and ensure his business offering remains relevant, Kalle spends quite a lot of time looking at and assessing what's going on in the programming world – different languages, frameworks and environments. He has noticed a repeating pattern with the new programs that have cropped up: "The Rusts, Darts, and Haskells are solving problems that the C++ community solved a long time ago. If you look at the details of the things that are being fixed in the C++ Standards Committee right now, you'll see that fixes are in the intricate details that affect a tiny set of cases. Many C++ developers won't even notice them, because they are only relevant to library writers, or because compiler magic will make things happen anyway. But if I look at the change logs of Dart, for example, which is a very promising and interesting language that I had some fun programming in, the things getting fixed there are quite fundamental." This is of course to be expected, as these languages are still very young, but it means those environments aren't as stable and dependable as C++ and won't be for a very long time.

The Rusts, Darts, and Haskells are solving problems that the C++ community solved a long time ago.

From Ivan's perspective, the best thing about these awesome new languages is that they are teaching C++ developers how to write better code. This is because they focus on specific things which they do really well by restraining the developer to specific paradigms or specific types of work. Nonetheless he doesn't think they'll ever replace it. He offers a perfect case in point in his book. C++'s flexibility allows it to borrow Haskell programming paradigms so functional programming can be added to a broader environment of existing code and techniques. C++ allows you to mix and match paradigms, gradually evolving your code base on your needs and your schedule. That flexibility is not something that's true for most other languages.

"New languages can be very sexy since their developers have learned valuable lessons from C++ baggage," adds Jon. "But the C++ community is very well aware of what these other languages are doing, and they bring those sexy new features into C++."

The gateway to C++

Once young people find out that some of the most dazzling games are written in C++, it drives them to learn about it.

One of the things Kalle has always loved about his line of work is that programmers are so enthusiastic about what they do. “Most of them would probably program even if they weren’t paid to do it,” he says. More often than not, like he did, they start off writing code in their time off from college or a part-time job, making a name for themselves doing what they love on an open-source project. But then at some point they arrive at the reality of needing to make money to live. “What often makes a programming language desirable is whether it can get you a job or not, preferably in a subset of the industry you’re interested in”, he adds. “C++ is a very attractive language for young people to learn since there are so many jobs in interesting areas – from games, high-performance financial computing, scientific computing, to wearable devices.”

“There’s a certain programmer personality that wants to understand and control everything the machine does at a low level, and that’s motivation to pick a systems language like C++”

Indeed, Jon observes that many find their way in initially through games, where it indirectly comes down to performance since games are written to make the most of machine resources. He explains, “Once young people find out that some of the most dazzling games are written in C++, it drives them to learn about it.” Games have to be written with short production cycles too, which also highlights that – contrary to old impressions – C++ can be a very productive language.

There are also people who want to (or need to) control the machine, he adds. “There’s a certain programmer personality that wants to understand and control everything the machine does at a low level, and that’s motivation to pick a systems language like C++,” he believes. “But because it also had high-level abstractions, they can write beautiful and easy-to-maintain code.”

Walking a fine line

Today C++ has multiple new standards being released, large conferences, and a thriving global community. Rather than an obsolete language, it is active and growing and attracting major investments from tech companies.

By absorbing and translating new programming methodologies over the years, the C++ standard is a moving target that is very competitive to new languages, something that isn’t often well appreciated by programmers who aren’t familiar with the latest C++ developments.

"We're moving the entire community forward, but we want to do that carefully, so we don't fracture it."

As the C++ language continues to learn and absorb from other developments going on in software engineering, the C++ Standards Committee has to strike a balance between continuing to take the language in ways that offer better productivity, safety, and expressiveness, while not making compromises on performance or breaking legacy code. Jon explains that a big problem is cohesiveness. "We want to add new features that some people will use but that others won't need or can't implement," he says. "We're moving the entire community forward, but we want to do that carefully, so we don't fracture it."

But one of the unique propositions of C++ is that it has a long and huge legacy with community, libraries, tools, platforms, and training. Building these things are non-trivial; they don't happen overnight and depend on the good will of thousands of people who care about the program. Our experts are confident that C++ will continue to thrive.

C++17 will be the standard that most people will care about for a while, and it's also the version used by Qt

What's next for C++?

What are the new features released in C++20 or that are being considered for C++23 that our experts are most excited about?

Concurrency – it's something that the OS normally takes care of, but concurrency should be part of the language to get more efficiency and performance improvements. Coroutines, significantly enhance multi-threading, are in the language but with no library support so that's coming a little bit later.

GPU – Similar to concurrency is exploiting the GPU, and while non-standard solutions or proprietary libraries exist, a way to incorporate generic C++ code into current and future GPUs would be a huge benefit.

Concepts – An earlier template metaprogramming technique called SFINAE (substitution error is not an error) gave interesting performance gains. However, concepts in the latest release can now do things better, in a more straightforward, and easier to understand and explain way.

Reflection – The easy way to do it takes a huge runtime hit, which isn't acceptable for C++. We want to have it so that you only pay the price when you need it, but not if you don't use it.

However, as Jens and Ivan both noted, C++17 will be the standard that most people will care about for a while, and it's also the version used by Qt. Ivan adds: "C++20 is sort of like the preview release for C++23 because there are so many nice things that aren't fully complete. And we're going to finish them in the next release." Certainly, something to look forward to.

.

Ivan Čukić is a senior software engineer at KDAB and the author of *Functional Programming in C++*. He's one of the core developers in the KDE community, which develops the largest free open-source C++ project, and is a participant in the C++ standardization process..

Kalle Dalheimer is a software pioneer and the co-founder of KDE. Most notably, he's the founder and CEO of KDAB, the leading consulting, training, and development company for Qt, C++ and Open GL.

Jon Kalb is co-author of *C++ Today: The Beast is Back*, a speaker and trainer in C++ and is a participant in the C++ standardization process. Jon is the inspiration behind and the organizer of *CppCon*, now the largest annual C++ conference globally.

Jens Weller is a C++ evangelist and active member of the C++ community worldwide. He's also the founder and organizer of what has now become the *Meeting C++* platform in Europe.

About the KDAB Group

The KDAB Group is the world's leading software consultancy for architecture, development and design of Qt, C++ and OpenGL applications across desktop, embedded and mobile platforms. KDAB is the biggest independent contributor to Qt and is the world's first ISO 9001 certified Qt consulting and development company. Our experts build run-times, mix native and web technologies, solve hardware stack performance issues and porting problems for hundreds of customers, many among the Fortune 500. KDAB's tools and extensive experience in creating, debugging, profiling and porting complex applications help developers worldwide to deliver successful projects. KDAB's trainers, all full-time developers, provide market leading, hands-on, training for Qt, OpenGL and modern C++ in multiple languages.

www.kdab.com