SNEAK PREVIEW: From the Experts

A Conversation with Bjarne Stroustrup

Every C++ developer knows Bjarne Stroustrup is the creator of the C++ programming language. Rogue Wave Software is excited that Bjarne will be presenting the keynote address at the Rogue Wave User Conference 2002 in Vail, Colorado, July 15-18, 2002. Below are several questions from Rogue Wave developers that just couldn't wait for the conference.

RW: Is C++ relevant in the Internet Age?

Bjarne: Certainly. C++ code is not suitable for downloading into an unsafe computer, but most computing is not of that kind.

C++ is the best language for many applications that involve some systems programming and have some resource constraints and/or some serious performance requirements. Google is one example. Embedded systems for handheld gadgets are another.

In addition, there are lots of programs that don't directly interact with the Internet. For such programs, the world hasn't significantly changed.

RW: Given the language independence the .NET platform brings to the table, what do you think the impact on C++ will be?

Bjarne: There are several related C++ communities. Some programmers work on programs intimately tied to an execution environment (such as .NET, a particular embedded system, or a particular UNIX variant). For such people, smooth integration with their platform is a major part of their concern.

There is enough emphasis on C++ in Microsoft's .NET world that C++ will remain a major language there, but for many the emphasis will inevitably be on the integration into the platform and interoperability with code written in other languages. In the longer term, people will get a degree of language independence at the cost of serious platform dependence. That could stifle experimentation with newer parts of C++, but so many C++ programmers use absurdly limited subsets that in the short term .NET could actually be a stimulus to better C++ use.

That said, my heart is with the programmers who fight for platform independence and portability. For those programmers thin interfaces and platform-independent libraries are going to be key. The ISO standard is the tie that binds the C++ sub-communities together and stops the language breaking up into a mess of proprietary dialects.

RW: The ISO/ANSI C++ standards committee is starting to discuss changes and extensions to the C++ language and libraries. Do you have a particular extension that you would very much like to see, or, one to which you are strongly opposed?

Bjarne: My general idea is simple: we should be cautious and deliberate about language extensions, but aggressive and opportunistic about standard library extensions. My reasons are almost equally simple: we want to increase portability and stability, which we can't do with a changing language. Libraries are different. If we get a "dud" library, users can ignore it in favor of better alternatives, or simply build or buy a better implementation than the one they got from their compiler vendor.

People expect more from a programming language these days—giveaways from the proprietary languages led the way here—and those expectations are most easily and safest met by increasing the size of the standard library. This is especially so if we can organize much of the standard library as extensible frameworks as was done with iostreams and the STL.

A successful extension of the language requires a direction, as does a successful direction of what I think of as the core of the standard library. In the language area, I'd like to focus on minor facilities that make the language more uniform and therefore easier to learn and easier to use generically. So how might this concern programmers who are more interested in getting their work done than in language details? The most obvious impact will be the new items offered by the standard library. A larger standard library both saves work and teaches technique and style. A problem with early C++ was that even though it provided good support for object-oriented programming, it did not supply a good library that

demonstrated OOP to users. That led to much confusion and many myths. The introduction of generic programming was done better, largely because the STL provided a concrete example for use and study. I just wish we had an equally good and useful example of the use of exceptions.

I'm working on a library called XTI (eXtended Type Information), providing an interface to general C++ type information for use in introspection and program transformation. I'd like to see something like that in the standard library. In general, I'd like to see better support for distributed programming and believe that to be primarily a library issue.

RW: What important trends do you see in C++ programming?

Bjarne: The C++ world is so large that it is hard to know if something you see is a trend or not. I think that there is a serious increase of C++ use in the embedded community, but I don't know how I would go about being sure. I know that there is an increase in the interest in "template metaprogramming," "generic programming," and "generative programming," but I'm not sure how broad-based that is. My guess is that these are "hot topics" among pioneers and academics and that some—but not all—of the new techniques will enter the mainstream over the next few years. There seems to be more C++ open source projects now, but I can't be sure. The C++ world is too large for any individual to completely understand it.

New techniques like generic programming and new language tools such as template exceptions are slow to make it into the open source community—too slowly for my taste, of course. I'd like to see some of the open source projects adopt more modern programming styles. The reason for this conservatism is that the open source community cannot "just send their contributors on a course" to ensure that all have a similar and up-to-date view of tools and techniques.

RW: Has C++ been so successful that programmers who need a language with the design criteria of C++ automatically turn to C++?

Bjarne: No, the world isn't that simple. Programmers, like all people, are affected by professional marketing. Programmers, like all people, tend to overestimate the (much hyped) advantages of the new while underestimating

the (rarely mentioned) complementary disadvantages, and to (wrongly) take for granted that they won't have to leave something important behind when they make a change. Note that sometimes C++ is "the new language" and people move to it for reasons that have nothing to do with its technical strengths and weaknesses. In an ideal world people objectively choose based on their needs. In the real world, we are subjective and rarely know our future needs well. Sometimes we get disappointed.

RW: Unlike Java, C# and Visual Basic, no one owns C++. In this way, it is a lot like Linux and other members of the open source movement. Yet it has not enjoyed the same appeal as the open source movement. Why not?

Bjarne: C++ is pragmatic, not messianic, and C++ does not have a political aspect—unless you want to categorize "not proprietary, controlled by an ISO standards committee" as political. Also, in some places proponents of alternatives have unfairly painted C++ as "a Microsoft language." Another problem is probably that I have concentrated too much on problems of performance and scale that have little appeal to novices and students. And then, of course, my dislike of hype may be getting in the way of effective marketing.

RW: What future role do you see for providers of proprietary libraries?

Bjarne: As the editor of Addison-Wesley's "C++ In Depth" series, I try to choose interesting and important topics, and good writers, of course. I like libraries as a way of making ideas and techniques concrete; that's why you can find books on ACE, Loki and BGL in my series. Each of these showcase ideas that are less commonly known and not as widely used as I'd like to see. However, that doesn't mean that I deem every library described in the series as perfection of its kind. I always hope for even better libraries. In particular, publishing an ACE book does not imply that I prefer open source libraries to commercial libraries. Both have important roles to play.

I'm no businessman, so I don't claim to know where the economic opportunities are, and I'm not an economist. However, my impression is that the profits are not in the really fundamental technology or the really new ideas. People prefer to spend their money on something fairly close to their specific application. Almost by definition, successful proprietary libraries and tools are where the profits are.

Solid engineering, quality, scale, interoperability, teaching, maintenance and support are areas where

software produced by unpaid programmers have a hard time competing. Note that where “free” and open source providers succeed on a larger scale, they do so through for-profit schemes. Even programmers have to eat sometimes!

I note that commercial implementations of the C++ standard library have been popular over the years, so I don't see the standard libraries as necessarily competing with commercial libraries. I hope that the C++ standard library can be a good base for more specialized libraries—many of which could be proprietary.

RW: C paved the way for C++. Do you envision C++ paving the way for a ‘next’ language? What kind of paradigm shift (procedural vs. OO, for example) should developers be expecting as a result?

Bjarne: I suspect that the world is too fractured for a single "next language." In particular, neither Java nor C# fits the bill.

Also, there is too much talk about "paradigm shifts." Object-oriented programming didn't come along as superior to procedural programming in every way. I think of "techniques" or "styles" more than of "paradigms." C++ supports several styles; it is a multi-paradigm language. This is important because what is hyped as "paradigms" tend not to be mutually exclusive. On the contrary, the styles can be mutually supportive.

So I'd rather not guess about a "next paradigm," but just predict that if and when it arrives, it will be found to complement and work well with procedural, object-oriented, and generic techniques.

RW: Was the ability to do compile-time computation using templates one of the design goals for templates, or was it a happy accident?

Bjarne: A bit of both. I worked hard at getting the primitives right, such as allowing inlining and merging of call and definition context. My focus was flexibility and efficiency of a few important examples, such as doing a sum over an array. That, plus a natural preference for generality over special-purpose features, and a dislike of unnecessary restrictions opened the way for techniques that I had not dreamed of, such as the STL. In that, templates resemble the rest of C++. I wouldn't like to build a tool that could only do what I had been able to imagine for it.

RW: How could templates have been made easier for developers to learn?

Bjarne: I see no reason to believe that templates are any more "scary" or "difficult" than classes and class hierarchies were. When I introduced those, the clamor about how complicated, unsafe, unmanageable and useless those features were was deafening.

Sadly, many programmers recoil from new constructs. I think that part of the reason is natural caution when it comes to critical tools. However, other parts are intellectual laziness, fear of new things that might disrupt comfortable old ways, and simply the habit of many in the software industry of disparaging things seen as competition.

If this reasoning is sound, we can simply wait for the clamor to die down as more people learn to use the new techniques well, as the excesses of enthusiasm cease, and our tools improve. Templates allow simpler expression of some important solutions and techniques than do alternatives, and can be used to deliver more efficient code than alternatives. In the longer run, fast code expressed concisely always wins.