DeepSoft, Inc. Engineering • Programming • Training
6259 Deep River Canyon, Columbia, MD 21045-2572
Phone: 410-312-2982 Email: tkf@deepsoftinc.com
Fax: 410-312-3067 Web: www.DeepSoftInc.com

C, C++, Java, & Visual Basic: a Comparison

Introduction

Frequently students or clients ask which computer languages I use and why. This paper is an attempt at answering those two questions. It covers what I sometimes refer to as the "big four" of current programming languages or code generation tools. And they are C, C++, Java, and Visual Basic. For a variety of reasons these are probably the most popular languages and tools available today. Each of them is a general-purpose language that can be used for a wide variety of applications. This paper will discuss the basic differences between these languages, their more important pros and cons, and list some typical applications that each handles well or not well at all. It is a testament to the speed, power, quality, flexibility, and efficiency of C that three of the four are either C itself or an Object Oriented (OO) derivative of it.

I started programming in Fortran on an IBM mainframe during my undergraduate engineering curriculum at the Pennsylvania State University. After graduation I used Fortran for several years as an aid in solving mechanical engineering (ME) problems. During that time I also learned Basic and used it as well for ME problems. In graduate school at the University of California at Berkeley, I picked up 8080/Z80 assembly language programming for embedded and real-time applications as well as continuing with Fortran and Basic. After graduation again, I was doing numerical simulations with Fortran, and some PL/1 for aerospace thermal and structural analysis applications. At this time I bought my first microcomputer, which was a CP/M, based Altos running a 4 Mhz Z-80 with 64K of RAM, dual 8-inch floppy disks and a CRT terminal. From this point I was totally hooked on microcomputers and their direct descendents, PC's. At this point I was "fishing around" for a new language trying both Pascal and PL/1 and liking them before I tried C and fell in love. I've been programming in C since 1984 and C++ since 1991, and providing training classes in both since 1988 and 1991 respectively.

There seems to be a considerable amount of confusion, even among programmers, let alone administrators and marketing staff about the relative merits of these languages. A very wise man once said, "A little knowledge can be a dangerous thing". You need to pick a language that makes sense not only for your application but also your skill and experience level. You should make an intelligent decision based on the facts not just on marketing hype or which language you think is the "coolest". It is important to understand why a particular language is a good or bad choice for a particular type of application. Picking a programming language is like picking a tool. You want to match the tool to the job. Picking the wrong language for the job could be a disastrous decision.

There is a huge universe of different types of software applications – each type usually having its own unique requirements and frequently a specific language developed to meet them. All

computer languages have their own pros and cons and normally a good reason for existing. What I have described are some of the most important languages available today and why I find them to be good or bad for specific types of applications. These of course are my opinions and not everyone will agree with them. These opinions are why these languages have worked very well for my types of applications or why they have not or would not. Other people's needs may be quite different from mine and would be better served by a different language. You need to find a match that works for you, your applications, and your skill and experience level.

The types of applications that I typically work on include: engineering or scientific analysis, custom Computer Aided Design (CAD) or Computer Aided Engineering (CAE) applications, Windows OO GUI applications for technical or business applications, and embedded systems programming for small, battery powered, ROM based intelligent instruments. Many people have a rather narrow view of software systems that is either GUI or Web centric – they simply are not aware of these other embedded types of applications. In 2003 the number of processors sold annually worldwide approaches 8 billion. Of these 8 billion Central Processing Units (CPU), Microcontroller Units (MCU), and Digital Signal Processors (DSP) only 2% go into PC's, Unix workstations, and Apples – the rest go into embedded applications ("Securing Cyberspace", J. Ganssle, Embedded Systems Programming, Embedded.com, 2/27/03).

Language Specific Information

C

Brian Kernighan and Dennis Ritchie developed C at Bell Labs in the mid 70's so that they could write UNIX in something other than assembler. UNIX today is typically written 90% in C and 10% in assembler for the kernel which must be extremely fast. Many other applications as well are written primarily in C with small amounts of assembly code if speed is of the essence. When they developed C they wanted an extremely fast, powerful and efficient language, and they went to great lengths to achieve it. In my opinion they succeeded admirably. It is an ideal language for controlling digital electronic hardware. It literally has all of the power of assembly language with all of the power of a high level language. The only drawback being that it is not as fast as assembly for most applications. For embedded applications it's perfect: you can work in a high level language virtually all of the time, its very fast, very small, very efficient, and gives you total access to any digital integrated circuits that you have to control, it can do any type of high level engineering or scientific computations, plus you can link in assembly routines if needed for faster execution of specific time critical functions. Some people feel it is too terse, I think that it's elegant. You will get used to the terseness very quickly and will come to appreciate it. In 2003, C is still the Ferrari of high level computer languages.

C++

C++ is a superset of C and formally adds object-oriented capability to C. Bjarne Stroustrup also from Bell Labs developed it in the early 80's. He wanted to add OO capabilities to C while retaining many of its advantages relative to other programming languages. C and C++ share many common attributes: data types are the same, storage classes are the same except C++ has one new one, flow control is the same, the operators are the same except that C++ has five new

operators, function prototypes are handled the same way, and C++ can use the C run-time library if desired but also provides its own much more powerful C++ class library.

While C and C++ do have a common heritage they do things quite differently. To use C++ effectively experienced C programmers have to learn to work in an OO fashion. When C++ first came out, its developers and many others said "forget C, C is dead". Well today, not too many people are saying that anymore. Most people whether academic, commercial, or industrial, recommend learning C++ as a two-step process. That is, learning C first and then moving onto C++. C has a lot of utility in its own right and is preferred for embedded system applications because of its inherent speed and efficiency. Also most people find it somewhat easier to learn C++ as a two step process but it's not a requirement. Some people choose to go directly to C++ without any prior C experience. C is ideal for embedded applications, while C++ is ideal for logistically more complex GUI and Web based applications.

OO languages were created to provide increased power to software engineers particularly for large and logistically or computationally complex applications such as GUI applications, Web applications, transient numerical simulations, 3D graphics, 3D solid modeling, etc. Just the GUI portion of a simple program is quite complex regardless of how you implement it. Software applications are continually increasing in size, capability, and complexity while economic pressures grow to reduce development time, increase code quality, and reduce overall cost. Clearly this is not possible without changing the nature of the tools that software engineer's use. C++ has become one of several very successful OO languages that are popular today. Because of its C heritage, which is based on an almost religious adherence to efficiency, it turns out to be one of the fastest and most efficient OO languages, certainly more so than Java or Smalltalk.

Java

In a sense, we are seeing the fallout of the "Battle of the Bills". That is Bill Joy of Sun Microsystems and Bill Gates of Microsoft. Sun Microsystems is trying very hard to reduce the dominance of Microsoft in the software arena and increase their own market share. This would be extremely difficult for them to do via a standard programming language so they are attempting to do it with the quasi-proprietary language Java which was developed at Sun by James Gosling.

Changing the name from Oak to Java was a stroke of marketing genius. Java evidently has a coefficient of cool, which is much higher than Oak. Nobody remembers Oak, which was Sun's predecessor to Java.

Java combines elements of C, C++, and Smalltalk. Java seems to be well suited to distributed web applications and has certainly become very popular for e-commerce web sites. I say it "seems" to be because I've never developed any of them and am giving Java the benefit of the doubt. Because I haven't worked on distributed web applications, I will only discuss them briefly and devote most of my time to Java's applicability to real-time embedded applications; something that I have done a lot of. I've only taken one course in Java and therefore am clearly not an expert – but some of the things I saw I didn't like.

For web based applications, Java benefits enormously from the extremely slow speed of the Internet. While most people are still running at 28 to 56 Kbps with dial up access, a small percentage are currently running broadband via Comcast or DSL. The latter providing speeds up to 1 to 1.5 Mbps. Compare these speeds to typical Ethernet access of 10 Mbps, 100 Mbps, with 1 Gbps on the way. At these two latter speeds, Java wouldn't have a prayer. You would need a faster language like C++, C, or assembly language. When we finally do get ubiquitous broadband access to the Internet, are we going to have to rewrite all of these Java applets in a faster language? Gee, this sounds a little like Y2K.

Java was originally created as an embedded language and is highly touted as a good language for embedded applications. I disagree totally. Three of the most important attributes for any embedded language are: execution speed, memory footprint or size requirements, and knowing exactly how long it will take a function or program to run or how deterministic it is. Java has major problems in all three areas. Any interpretive language is going to be slow by definition – in many applications this would be a fatal flaw all by itself. Its memory footprint also tends to be quite large which is always a problem for ROM based intelligent instruments such as cell phones, GPS receivers, portable medical instruments, etc. Mass produced consumer products like these are very sensitive to cost, component count, physical size, and battery life. The more memory required the larger capacity or greater number of memory chips are required, which makes the Printed Circuit Board (PCB) larger, and the housing larger which increases component, manufacturing, shipping, and warehouse costs, etc. It also requires more battery power that requires either larger or more expensive batteries, which also increases everything as listed above. If that isn't bad enough, probably the most damaging attribute is the fact that it is very hard to predict how much time it will take for a Java function or program to run because it is highly non-deterministic. Just in time compilers and Java based microprocessors are helping but these problems remain.

Java uses a technique called garbage collection to automatically deal with dynamically allocated and destroyed memory. It does all of this for you automatically rather than relying on the programmer to explicitly and properly create and destroy this dynamic data at the right time as you must do in C or C++. This is a nice feature but it comes at a very high price. Not only is it slow, but it's totally non-deterministic which is a serious problem for embedded applications. You absolutely, positively must know precisely how long a function will take to execute. Using Java - you don't have a clue. The garbage collection cure could be worse than the memory leak problem.

For real-time applications this could be a disaster. A skilled plaintiff's attorney in a product liability lawsuit might be able to show criminal negligence – they certainly will try – and you're giving them a lot of ammunition. One of my former employers, a hoist and crane manufacturer, was sued after a terrible construction accident. Although I had nothing to do with the design of this product and they were ultimately found by the jury to not be negligent in their design, the six weeks that I spent in court was a priceless education. This product did not contain any software, but many of today's products do. For better or worse, software has become just as important a component as the hardware. When things go terribly wrong, all of your design decisions will be under extreme scrutiny. If you can't do something as simple as making an intelligent choice of tools how can you possibly do an intelligent design of the software itself.

Visual Basic

Basic is the standard language that Visual Basic uses. Basic was developed primarily as a teaching language for first year programmers or more likely nonprogrammers who wanted to learn something about writing software without going through the rigors of a computer science curriculum which would have been focused on much faster and more powerful languages such as Fortran, Pascal, or C to name a few of the more popular languages at the time that Basic was developed at Dartmouth.

Basic is a good language for nonprogrammers and a terrible language for professional programmers. It's extremely slow, and somewhat limited in capability relative to other high level languages. Basic is only slightly easier to learn than other languages but you pay a significant penalty in performance and flexibility. To me this is not a good tradeoff particularly for professional programmers. Unfortunately Basic is used by many people who would be much better served by some other faster more powerful language – just about any other high level language would be a significant improvement.

Visual Basic (VB) is a variation of Basic implemented to create Graphical User Interface (GUI) programs that run under Microsoft Windows. GUI programming in any language is a difficult and complex task that requires a substantial learning curve. VB in some ways does lessen the learning curve. Again it was originally intended for nonprogrammers. At the time VB was developed, professional programmers were using C and the Software Developers Kit (SDK) to develop Windows GUI applications. VB was the only relatively easy way to develop GUI applications and it continues to serve a useful niche in this capacity in spite of all its shortcomings.

Sometime later Microsoft brought out Visual C++ (VC++) and the Microsoft Foundation Class (MFC). VC++ and the MFC were not only much easier to work with than C and the SDK they were more powerful in most respects. VC++ was also very similar to VB and today is a far better alternative, it is much faster, more powerful, and more flexible and is about the same level of difficulty to learn and use as VB. Both tools require knowledge of their respective languages; i.e., Basic or C++ prior to learning the tool itself. Both create working skeleton GUI applications that require the developer to add additional features and code. VB does create more of a working application than you would get from VC++, which generates only a working skeleton. You will have to do much more coding with VC++ than with VB.

Visual Basic being based on Basic has the same inherent limitations as the language; that is it creates very slow applications with limited flexibility. Visual C++ has no such limitations. The only thing that you can't do with Visual C++ is something that would not be possible with Windows itself. While the learning curve for VB should be shorter than VC++ there isn't a big difference. I've seen people offering as many as eight days of VB training. In eight to ten days you can learn both C++ and MFC GUI programming with VC++. For essentially the same investment of time you learn a far more powerful set of tools, which will enable you to develop much faster, more flexible, and far more powerful applications - to me that sounds like a good tradeoff.

In general the lower your skill and experience level the more VB will appeal to you. VB can be a great choice for nonprogrammers, those with little experience, or highly skilled programmers with a short development cycle for an application that can tolerate VB's slow speed and diminished capacity.

Typical Applications

C

Because most people are not as familiar with embedded applications I have included more information and examples about them. C is the dominant language for embedded applications that have to be small, fast, and efficient – which is most of them. C began replacing assembly language for many embedded applications in the early 80's. Embedded programmers today use C for most of the application, and only use assembly language where necessary. For GUI work it has largely been replaced by VC++, or VB.

Large embedded applications are typically handled by either stand-alone or networked general-purpose computers such as Windows or Unix workstations connected to large amounts of external hardware devices. Examples include controlling the furnaces and rolling mills in a steel manufacturing plant, controlling a nuclear power plant, or controlling robotic welders on an assembly line.

Smaller embedded applications run on individual or networked microprocessors (CPU), microcontrollers (MCU), and/or digital signal processors (DSP). These devices frequently use custom Single Board Computers (SBC) which are optimized for small size, minimum cost, and maximum battery life. These are Read Only Memory (ROM) based systems, which frequently run a Real-Time Operating System (RTOS) which is optimized to handle real-time embedded applications. Examples include: cell phones, Global Positioning System (GPS) receivers, video cameras, portable medical monitors, decompression computers for divers, etc.

Most embedded applications have to perform a variety of tasks ranging from: data input from the user interface; data acquisition from attached internal and external sensors; engineering, scientific, or medical computations based on that data; process control of internal and external hardware that has to be controlled as a function of data and algorithms; and finally displaying output data to the user, sometimes in a graphical format. And all of this must happen within many carefully controlled windows of time for each specific task. Virtually all embedded software must be able to deal successfully with many Interrupt Service Routines (ISR). ISR's are software routines that allow the hardware to communicate with the rest of the program. When a hardware peripheral or critical software module needs to do something it fires off an ISR which communicates with the rest of the software. Some specific embedded examples include such things as the following.

A Windows or Unix workstation connected to a thermal, vacuum chamber used to simulate outer space conditions for satellite testing. The workstation would be controlling vacuum pumps, heaters, chillers, and data sensors such as thermocouples, strain gages, pressure transducers, etc.

The vacuum chamber itself and the satellite inside would both be heavily instrumented. The software would be doing all of the following: real-time process control, data acquisition, numerical computation, and graphical data display on the system monitor(s).

A cell phone or a GPS receiver, which are both handheld, battery powered, ROM based intelligent instruments. These are small ROM based computers with stringent timing requirements. Devices like this need to be "sleeping" as much as possible to maximize battery life.

Controlling the wing flaps of an F-16 jet fighter. This will be a synergy of Mechanical, Electrical, and Software Engineering. Perfect execution of all three is necessary for this task to be completed properly and reliably. Improper execution could easily result in a plane crash. Specific functions may have to execute consistently and repeatedly within hundreds of microseconds. It has to be a highly deterministic function; i.e., we have to know exactly how long it takes to execute to ensure that it fits within the allowable window of time.

C++

Most mainstream commercial software products used to be written in C, today most are written in C++ or some other object oriented language. Most large commercial Microsoft Windows GUI applications are done in C++, which is an excellent choice. It's very fast, and very powerful, the only downside is that you have to be a reasonably proficient C++ programmer to do it.

Microsoft Windows GUI applications are an example of a good match with C++. It is a powerful, flexible, and relatively fast language optimized for large and logistically complex applications. The OO extensions it adds to C are perfect for dealing with these types of problems. They allow natural and relatively easy extension to existing applications via encapsulation, inheritance, and polymorphism. Resulting in a simplified programmer's user interface via class hierarchies, overloaded functions, overloaded operator functions, and virtual functions. The Standard Template Library (STL) provides a wide array of template classes and functions for many general purpose programming tasks.

Just like the extensive C function libraries that exist for a variety of specialized programming tasks, there are even more powerful and sophisticated C++ class libraries for these same types of tasks. Examples include Windows GUI programming, telecommunications, Web communications, database applications, scientific and engineering applications, graphics, CAD, matrix manipulation, etc.

Java

Java seems to be well suited to distributed web applications and has certainly become very popular for e-commerce web sites. I've never developed any of them and am giving Java the benefit of the doubt. As long as the Internet remains so slow, Java's lack of performance won't matter.

Visual Basic

I can't think of a single major piece of commercial software that is written in VB – it's just too slow and inflexible. People use VB for small to medium sized GUI applications for internal company needs or similar scale products that are sold commercially in relatively small numbers. These are applications that can tolerate slow speed and diminished capacity, don't require a lot of time and cost to write (very important), and can be tailored to meet their needs. For those types of applications VB is an excellent choice, and this does define a significant subset of the GUI application arena.

Marketing Hype

It's not possible to discuss the relative merits of programming languages while ignoring the impact of marketing even though this has nothing to do with those merits. We live in a society that is obsessed with and saturated by marketing hype. Everybody wants the latest, hottest, coolest (now that's an oxymoron) product even if they don't have a clue what they're going to do with it. Marketing managers know this and they exploit it to the hilt. Couple this with a huge number of administrators (government, industrial, business, and academic) who only understand technology at the buzzword level and you have a very big problem.

Managers will frequently pick the overall software toolset that their engineers and programmers will use on a project. Unfortunately they frequently don't know what they are doing and saddle their high-powered technical underlings with poor quality and/or totally mismatched tools, which slows them down considerably. Or they pick tools that they understand – only because they understand them. I know - I suffered through this over and over again while working as an engineer and programmer at and for numerous organizations.

Write it Once, Run it Everywhere

This is a great concept as long as you aren't paying too high a price to get it. This concept is not new or unique to Sun or Java. Indeed the UCSD P System of the University of California at San Diego was a somewhat proprietary implementation of the standard Pascal programming language that was a popular commercial alternative at the time (circa early to mid 80's). Most people today have so little computer experience that they have never even heard of this system. It dates back in microcomputer history to the era of 8-bit CP/M, 4 Mhz Z80's, 64K of RAM, 8-inch floppy disks, and CRT terminals. The UCSD P System also preached the credo of write it once and run it everywhere. It worked a lot like Java in that it executed a universal intermediate code that was interpreted locally on a different machine. This system basically died out because it was just too slow.

C and C++ achieve their portability via a rich and varied set of compilers, tools, function libraries, and class libraries. Basically you "write it once and run it everywhere" by buying a new compiler for the desired target processor, port the source code, and recompile the source code. Recompiling the source code can be a problem if the code is not written in a portable fashion or if there are unique hardware and software features available on different platforms. The cost of

using C and C++ is dealing with these differences. The benefits include: very fast, powerful, flexible, and efficient languages that create like applications that can literally run anywhere from the smallest ROM based microcontroller to the most powerful supercomputer at very high speed with total flexibility. Even if you want to do GUI applications you can purchase a family of C++ class libraries from vendors who support a wide range of GUI platforms such as: Microsoft Windows 3.1, 95, 98, NT, 2000, Unix X Windows, IBM OS/2, and the Apple Macintosh. Basically a common GUI library interface is used across all GUI platforms so your code does not change. The guts of each library function will change to what is required for a specific GUI platform – but the GUI library vendor has already done that for you.

With the various dialects of Java, and Sun's seeming reluctance to actually submit the Java language to a national or international standards committee – it's hard to write standard Java code because there is not a standard for the language. The degree of portability in Java is not nearly as high as Sun would have you believe. Java developers are probably spending as much time porting Java code between different dialects of Java as C and C++ programmers have when porting code between different CPU's, platforms, and operating systems. The only thing you can be sure of is that Java code is not going to be very fast.

With VB you write it once and run it once.

New Alternatives

Embedded C++ (EC++)

EC++ has been under development for several years and was initiated by several Japanese manufacturing companies. Many embedded compiler vendors today offer an EC++ version of their normal C++ compiler. There is not currently a standard for EC++ and, as far as I know, there aren't any plans to implement one at the present time. Basically EC++ is a stripped down version of C++ that is more suitable for embedded applications and their frequently stringent requirements for small size, high-speed and high efficiency. It maintains most of the OO advantages of C++ while removing some of the features that are relatively inefficient with respect to execution speed, memory space, and not commonly needed for embedded applications anyway.

C#

C#, which is pronounced C-sharp, is a relatively new concept from Microsoft to address increased productivity for C and C++ programmers for developing web and GUI based applications. How far along this is or how well it will ultimately pan out is unknown at this time. To get more information about it, see Microsoft's website. C# and the .NET framework is an alternative to Java. .NET and the many languages that support it provide language neutral development tools while Java would provide platform neutral development tools.

http://msdn.microsoft.com/vstudio/nextgen/technology/csharpintro.asp

GUI Development Tools

Visual C++

Visual C++ works much like VB in that it allows the programmer to specify the application type and then automatically builds a working skeleton of the right type of application. The programmer then adds C++ code to build the rest of the application. There aren't any programming flexibility limitations, and the applications would be much faster than if generated with VB. The only drawback is that you have to be a reasonably proficient C++ programmer.

Visual Basic

VB is used as an application generator to create GUI applications. The user defines what they want and VB builds a working skeleton of their application. The programmer then adds Basic code to build the rest of the application. This is a very fast and efficient way of creating GUI applications that can tolerate VB's lack of speed and programming flexibility.

Future Applications

Workstation based GUI applications, both Windows and Unix, will continue to grow in importance, capability, and percentage with Windows based applications continuing to maintain a huge market share advantage over Unix due to their increased quality, number of features, and vastly greater numbers of applications at reasonable prices.

Web based applications are currently exploding in significance and importance. They will probably grow to eventually rival GUI applications in terms of market share. Although for this to happen we will need ubiquitous broadband Internet access of 100 Mbps or greater without the numerous JavaScript errors and telecommunications hang-ups that are all too common today. Until we get this type of high-speed access, most substantial applications will still be done as resident programs running on individual systems with the Internet providing an efficient way of handling data communications. For the time being, trying to run a feature based, parametric, 3D solid modeler remotely and interactively over the Internet is ridiculous. Right now we can't even handle email 100% reliably.

Embedded applications already totally dominate the sale of silicon real estate to the point of dwarfing everything else. They will continue to grow at a steady and unrelenting pace. The magnitude of distributed computing power that they contain will also increase.

Conclusions

Clearly I am strongly biased toward C and C++ and given my experience level and the types of applications that I work on this bias is well justified. Your applications and experience level may be quite different and Java or VB may make more sense for you. Pick a language that works for you, but don't do it in a vacuum, make a decision based on an awareness of the pros and cons of your tool choices. Early tool choices have a huge impact on downstream performance, capability,

and flexibility. Learning C and C++ is only marginally more difficult than learning VB and it is about the same as learning Java. The additional time required in learning these more powerful and more comprehensive tools would be a good investment for most programmers who aspire to become software engineers.

Windows GUI Applications

Large, powerful, flexible, and high-speed applications will best be handled with C++ using such tools as Visual C++, Borland C++ Builder, and others. Most high-level professional programmers will want to use C++ as their preferred language of choice. Visual Basic is a good choice for non-programmers, programmers with low skill levels, or high skill level programmers with very short development cycles and applications that can tolerate the slow speed and diminished capacity of Visual Basic. Anything Visual Basic can do, Visual C++ can do much better.

Web Based Applications

Java seems to be dominating in this arena and doing a fairly good job within the limitations that I mentioned previously namely that of speed, or total lack thereof.

Embedded Applications

Embedded applications will continue to be dominated by C with its inherent speed, power, and efficiency. For many applications C is an acceptable replacement for assembly language. EC++ will be an acceptable alternative for some applications. C++ is also being used for large and more complex embedded applications with time critical routines being written in C or assembly language as required. Java is not an acceptable language for the vast majority of embedded applications.

Learning C++

C++ has an undeserved reputation as being too difficult and complex to learn and use. While learning C++ is not trivial it's not much harder to learn than C which is not much harder to learn than Basic. You could teach C++ at the Junior or Senior High School level. I once had a 10 year old boy take my three day Introduction to C course and he did very well. Since then I've had numerous high school students take my Introduction to C++ and Intermediate C++ courses and they all did quite well. Much of this perceived difficulty is caused by the transition from procedural to object oriented programming concepts that will occur for any OO language. In the long run it is almost always more economical to invest the time required to learn powerful tools. For a small additional investment of time you reap a substantial increase in return on that investment.

Hardware Engineers: Mechanical, Electrical, Ocean, Civil, etc, have to study numerous subjects such as: thermodynamics, fluid dynamics, heat transfer, vibrations, elasticity, stress analysis, calculus, and differential equations; all of which are far more complex than C++. If current Software Engineers are incapable of learning C++ then we have a serious problem at the

University level when it comes to Software Engineering training. I don't really think that this is the case, just that a lot of people have swallowed the hype that says that "C++ is too hard to learn, and Java is easier to learn".

Ted K. Fryberger, PE
BSME, MSME & OE
President, DeepSoft, Inc.