Which languages are used for safety-critical software? [closed]

I'm researching the development of safety-critical software, and in particular what effects the choice of programming language has on such development.

Please explain, in detail, which languages are commonly used, and why.

software-engineering safety-critical

edited Jul 7 '14 at 10:41
Jk1
6,471 9 29 49

asked Oct 28 '08 at 13:50
marcumka
1,041 3 9 14

closed as primarily opinion-based by Danubian Sailor, Robert Longson, ApproachingDarknessFish, Flexo ♦ Nov 25 '14 at 9:42

Many good questions generate some degree of opinion based on expert experience, but answers to this question will tend to be almost entirely based on opinions, rather than facts, references, or specific expertise.

If this question can be reworded to fit the rules in the help center, please edit the question.

11 Can you elaborate on what you had in mind? The control software for an x-ray machine might have slightly different requirements to the control software for a fly-by-wire system. – ConcernedOfTunbridgeWells Oct 28 '08 at 22:07

17 Answers

Ada and SPARK (which is an Ada dialect with some hooks for static verification) are used in aerospace circles for building high reliability software such as avionics systems. There is something of an ecosystem of code verification tooling for these languages, although this technology also exists for more mainstream languages as well.

Erlang was designed from the ground up for writing high-reliability telecommunications code. It is designed to facilitate separation of concerns for error recovery (i.e. the subsystem generating the error is different from the subsystem that handles the error). It can also be subjected to formal proofs although this capability hasn't really moved far out of research circles.

Functional languages such as Haskell can be subjected to formal proofs by automated systems due to the declarative nature of the language. This allows code with side effects to be contained in monadic functions. For a formal correctness proof the rest of the code can be assumed to do nothing but what is specified.

However, these languages are garbage collected and the garbage collection is transparent to the code, so it cannot be reasoned about in this manner. Garbage collected languages are not normally predictable enough for hard realtime applications, although there is a body of ongoing research in time bounded incremental garbage collectors.

Eiffel and its descendants have built-in support for a technique called Design By Contract which provides a robust runtime mechanism for incorporating pre- and post- checks for invariants. While Eiffel never really caught on, developing high-reliability software tends to consist of writing checks and handlers for failure modes up-front before actually writing the functionality.

Although C and C++ were not specifically designed for this type of application, they are widely used for embedded and safety-critical software for several reasons. The main properties of note are control over memory management (which allows you to avoid having to garbage collect, for example), simple, well debugged core run-time libraries and mature tool support. A lot of the embedded development tool chains in use today were first developed in the 1980s and 1990s when this was current technology and come from the Unix culture that was prevalent at that time, so these tools remain popular for this sort of work.

While manual memory management code must be carefully checked to avoid errors, it allows a degree of control over application response times that is not available with languages that depend on garbage collection. The core run time libraries of C and C++ languages are relatively simple, mature and well understood, so they are amongst the most stable platforms available. Most if not all of the static analysis tools used for Ada also support C and C++, and there are many other such tools available for C. There are also several widely used C/C++ based tool chains; most tool chains used for Ada also come in versions that support C and/or C++.

Formal Methods such as Axiomatic Semantics (PDF), Z Notation or Communicating

Sequential Processes allow program logic to be mathematically verified, and are often used in the design of safety critical software where the application is simple enough to apply them (typically embedded control systems). For example, one of my former lecturers did a formal correctness proof of a signaling system for the German railway network.

The main shortcoming of formal methods is their tendency to grow exponentially in complexity with respect to the underlying system being proved. This means that there is significant risk of errors in the proof, so they are practically limited to fairly simple applications. Formal methods are quite widely used for verifying hardware correctness as hardware bugs are very expensive to fix, particularly on mass-market products. Since the Pentium FDIV bug, formal methods have gained quite a lot of attention, and have been used to verify the correctness of the FPU on all Intel processors since the Pentium Pro.

Many other languages have been used to develop highly reliable software. A lot of research has been done on the subject. One can reasonably argue that methodology is more important than the platform although there are principles like simplicity and selection and control of dependencies that might preclude the use of certain platforms.

As various of the others have noted, certain O/S platforms have features to promote reliability and predictable behaviour, such as watchdog timers and guaranteed interrupt response times. Simplicity is also a strong driver of reliability, and many RT systems are deliberately kept very simple and compact. QNX (the only such O/S that I am familiar with, having once worked with a concrete batching system based on it) is very small, and will fit on a single floppy. For similar reasons, the people who make OpenBSD - which is known for its robust security and thorough code auditing - also go out of their way keep the system simple.

EDIT: This posting has some links to good articles about safety critical software, in particular Here and Here. Props to S.Lott and Adam Davis for the source. The story of the THERAC-25 is a bit of a classic work in this field.

edited Jan 26 at 18:48

community wiki
26 revs, 5 users 97%
ConcernedOfTunbridgeWells

8 +1 Good, comprehensive answer. :) Plus we need more ADA love around here. – lemnisca Dec 8 '08 at 23:39

73 Careful. When I worked in aerospace, Ada was used just because it was aerospace, not because of any specific Ada feature. You can write highly reliable software in any language. Aerospace software is reliable because of their rather extreme spec/coding processes, not because Ada is magic pixie dust. – Ken Feb 20 '09 at 0:12

8 Sorry for the edit, but the "ADA" thing really bugs me. It's a person's last name, not an acronym. Otherwise correctly accepted post. Voted up. – T.E.D. Mar 17 '09 at 20:38

10 One of Ada's strengths actually is that it actively supports the mindset and methodologies required to develop safety-critical software, of course you could program safety-critical software in any programming language (heck, even in BASIC or assembly), but Ada was specifically designed and developed for this purpose. And the SPARK extension even more so. – none Jun 9 '09 at 23:48

7 "It is a functional language, which means that code has no side effects". That term is generally used to mean that a programming language has first-class lexical closures. In fact, Erlang relies heavily upon side-effects for all IO including message passing. – Jon Harrop May 8 '12 at 15:32

|

For C++, the Joint Strike Fighter (F-35) C++ Coding Standard is a good read:

http://www.stroustrup.com/JSF-AV-rules.pdf

edited Nov 19 '14 at 15:26

answered Oct 28 '08 at 14:33
Andrew Medico
16.2k 8 37 65

I think C++ is OK if you can discipline your team not to get carried away with the wonderfulness it provides - virtual stuff, dynamic memory, overloading, etc. etc. – Mike Dunlavey Nov 28 '08 at 16:32

1 IIRC the C++ bits are the signal processing code for the radar; the avionics code was written in Ada. – ConcernedOfTunbridgeWells Dec 8 '08 at 23:14

1 I wonder if there are any tools which check these standards statically, before compiling (if possible) the code. – Nils May 8 '12 at 14:22

Parasoft has a tool which allows checking against various coding standards. – zeroc8 May 9 '12 at 5:05

@Nils: Google for "static analysis JSF++" – Richard Corden May 9 '12 at 10:45

|

I believe Ada is still in use in some government projects that are safety and/or mission critical. I've never used the language, but it's on my list of "to learn", along with Eiffel. Eiffel offers

Design By Contract, which is supposed to improve reliability and safety.

answered Oct 28 '08 at 13:52
Thomas Owens
60.6k 82 258 403

You can use DBC in C and C++ through asserts and variations of similar macros. – Shane MacLaughlin Oct 28 '08 at 13:58

Yes, you can, but it's a language construct in Eiffel, and well-documented as how to enforce DBC in the language. – Thomas Owens Oct 28 '08 at 14:00

Firstly, safety critical software adheres to the same principals that you would see in the classic mechanical and electrical engineering fields. Redundancy, fault tolerance and fail-safety.

As an aside, and as the previous poster alluded to (and was for some reason down-voted), the single most important factor in being able to achieve this is for your team to have a rock solid understanding of everything that is going on. It goes without saying that good software design on your part is key. But it also implies a language that is accessible, mature, well supported, for which there is a lot of communal knowledge and experienced developers available.

Many posters have already commented that the OS is a key element in this respect which is very true most because it must be deterministic (see QNX or VxWorks). This rules out most interpreted languages that do things behind the scenes for you.

ADA is a possibility but there is less tools and support out there, and more importantly the stellar people aren't as readily available.

C++ is a possibility but only if you strictly enforce a subset. In this respect it is devil's tool, promising to make our life easier, but often doing too much,

C is ideal. It is very mature, fast, has a diverse set of tools and support, many experienced developers out there, cross-platform, and extremely flexible, can work close to the hardware.

I've developed in everything from smalltalk to ruby and appreciate and enjoy everything that higher languages have to offer. But when I'm doing critical systems development I bite the bullet and stick with C. In my experience (defence and many class II and III medical devices) less is more.

edited Dec 8 '08 at 23:09
ConcernedOfTunbridge Wells
46k 12 113 176

answered Oct 30 '08 at 21:07
Shaun
141 2

1 "ADA is a possibility but there is less tools and support out there, and more importantly the stellar people aren't as readily available." Ada (never ADA - it isn't an acronym) does require some of those extra tools, as the language offers what the tools offer by default. Ada would be en excellent choice and is very easy to learn. Google "Ada McCormick train modeling". He has since updates his results, I believe, showing that Ada is still a very easy to use language. – YermoungDer May 28 '09 at 8:45

I'd pick up haskell if it's safety over everything else. I propose haskell because it has very rigid static type checking and it promotes programming where you build parts in a such manner that they are very easy to test.

But then I wouldn't care about language much. You can get much greater safety without compromising as much by having your project overall in condition and working without deadlines. Overall as in having all the basic project management in place. I'd perhaps concentrate on extensive testing to ensure everything works like it ought, tests that cover all the corner cases + more.

answered Oct 28 '08 at 14:07
Cheery
10.5k 8 44 74

2 The only problem with Haskell is that most Haskell programmers are in academia and research. Most programmers in places where they need high reliability aren't familiar with Haskell and side effect free programming. – Mark Cidade Oct 28 '08 at 21:55

9 corp.galois.com/critical-systems - a business built on haskell for safety critical systems. – Don Stewart May 8 '12 at 15:12

@MarkCidade - That's a shame, because one would imagine they're the ones who would benefit the most. – Jason Baker May 8 '12 at 17:53

The language and OS is important, but so is the design. Try to keep it bare-bones, drop-dead simple.

I would start by having the bare minimum of state information (run-time data), to minimize the chance of it getting inconsistent. Then, if you want to have redundancy for the purpose of fault-tolerance, make sure you have foolproof ways to recover from the data getting inconsistent. Redundancy without a way to recover from inconsistency is just asking for trouble.

Always have a fallback for when requested actions don't complete in a reasonable time. As they say in air traffic control, an unacknowledged clearance is no clearance.

Don't be afraid of polling methods. They are simple and reliable, even if they may waste a few cycles. Shy away from processing that relies solely on events or notifications, because they can be easily dropped or duplicated or misordered. As an adjunct to polling, they are fine.

A friend of mine on the APOLLO project once remarked that he knew they were getting serious when they decided to rely on polling, rather than events, even though the computer was horrendously slow.

P.S. I just read through the C++ Air Vehicle standards. They are OK, but they seem to assume that there will be lots of classes, data, pointers, and dynamic memory allocation. That is exactly what there should no more of than absolutely necessary. There should be a data structure czar with a big scythe.

edited Nov 28 '08 at 15:42

answered Nov 28 '08 at 14:16
Mike Dunlavey
32.9k 7 65 104

The OS is more important then the language. Use a real time kernel such as VxWorks or QNX. We looked at both for controlling industrial robots and decided to go with VxWorks. We use C for the actual robot control.

For truly critical software, such as aircraft autoland systems, you want multiple processors running independently to cross check results.

answered Oct 28 '08 at 14:19
Jim C
4,858 14 25

Real-time environments usually have "safety-critical" requirements. For that sort of thing, you could look at VxWorks, a popular real-time operating system. It's currently in use in many diverse arenas such as Boeing aircraft, BMW iDrive internals, RAID controllers, and various space craft. (Check it out.)

Development for the VxWorks platform can be done with several tools, among them Eclipse, Workbench, SCORE, and others. C, C++, Ada, and Fortran (yes, Fortran) are supported, as well as some others.

answered Oct 28 '08 at 14:27
Alan
2,620 1 20 32

Since you don't give a platform, I would have to say C/C++. On most real-time platforms, you're relatively limited in options anyway.

The drawbacks of C's tendency to let you shoot yourself in the foot is offset by the number of tools to validate the code and the stability and direct mapping of the code to the hardware capabilities of the platform. Also, for anything critical, you will be unable to rely on third-party software which has not been extensively reviewed - this include most libraries - even many of those provided by hardware vendors.

Since everything will be your responsibility, you want a stable compiler, predictable behavior and a close mapping to the hardware.

answered Oct 28 '08 at 13:53
Cade Roux
65.8k 29 131 224

1 Actually, there are several languages that offer "better" (for safety) constructs that pure C/C++ but still compile into native code. I believe Eiffel compiles into native code, and I would use that over C/C++ in a safety-critical system. – Thomas Owens Oct 28 '08 at 13:55

Eiffel's advantages there would be offset in my mind by the advantages of the wider base of experienced C engineers and more mature compilers. – Cade Roux Oct 28 '08 at 13:58

Again, I'm assuming that risk mitigation is the #1 priority - that every keypress in this system is truly life-or-death. – Cade Roux Oct 28 '08 at 13:59

I agree with the wider base of experienced engineers, however I don't think that Eiffel's compiler is that far behind a native C compiler, if it is at all. – Thomas Owens Oct 28 '08 at 13:59

1 C vs C++ flame war can't be handled in comments :) I can provide a few link for very long debates on the subject and it's always ends without resolution - it's an endless war, BTW i'm on C side :) – Ilya Oct 28 '08 at 15:05

|

Here's a few updates for some tools that I had not seen discussed yet that I've been playing with lately which are fairly good.

The LLVM Compiler Infrastructure, a short blurb on their main page (includes front-ends for C and C++. Front-ends for Java, Scheme, and other languages are in development);

> A compiler infrastructure - LLVM is also a collection of source code that implements the language and compilation strategy. The primary components of the LLVM infrastructure are a GCC-based C & C++ front-end, a link-time optimization framework with a growing set of global and interprocedural analyses and transformations, static back-ends for the X86, X86-64, PowerPC 32/64, ARM, Thumb, IA-64, Alpha, SPARC, MIPS and CellSPU architectures, a back-end which emits portable C code, and a Just-In-Time compiler for X86, X86-64, PowerPC 32/64 processors, and an emitter for MSIL.

VCC;

> VCC is a tool that proves correctness of annotated concurrent C programs or finds problems in them. VCC extends C with design by contract features, like pre- and postcondition as well as type invariants. Annotated programs are translated to logical formulas using the Boogie tool, which passes them to an automated SMT solver Z3 to check their validity.

VCC uses the recently released Common Compiler Infrastructure.

Both of these tools, LLVM or VCC are designed to support multiple languages and architectures, I do think that their is a rise in coding by contract and other formal verification practices.

WPF (not the MS framework :), is a good place to start if you're trying to evaluate some of the recent research and tools in the program validation space.

WG23 is the primary resource however for fairly current and specific *critical systems development language details*. They cover everything from Ada, C, C++, Java, C#, Scripting, etc... and have at the very least a decent set of reference and guidance for direction to update information on language specific flaws and vulnerabilities.

edited Oct 20 '11 at 1:20
Iterator
12.7k 5 44 91

answered Jul 17 '09 at 14:21
RandomNickName42
4,867 1 25 30

Any software product can pass the DO-178b certification process using any tool but the questions is how difficult would it be. If the compiler isn't certified you may need to demonstrate your code is traceable at the assembly level. So it is helpful that your compiler is certified. We used C on our projects but had to verify at the assembly level and use a code standard that included turning off the optimizer, limited stack usage, limited interrupt usage, transparent certifiable libraries, etc. ADA isn't pixie dust but it makes the PSAC plan look more achievable.

As applicatons get larger, assembly code becomes less viable choice. The ARM processor just invites C++, but if you ask companies like Kiel it their tool is certified, they will return with a "huh?" And don't forget that verificaton tools also need to be certified. Try verifying a LabView test program.

answered Oct 11 '12 at 21:39
Randy Hoffman
41 1

A language that imposes careful patterns may help, but you can impose careful patterns using any language, even assembler. Every assumption about every value needs code that tests the assumption. For example, always test divisor for zero before dividing.

The more you can trust reusable components, the easier the task, but reusable components are seldom certified for critical use and will not get you through regulatory safety processes. You should use a tiny OS kernel and then build tiny modules that are unit tested with random input. A language like Eiffel might help, but there is no silver bullet.

answered Oct 28 '08 at 16:01
dongilmore
544 2 7

I agree, but I'm upvoting because of the random-data-testing statement. – Mike Dunlavey Nov 28 '08 at 16:19

There are a lot of good references at http://www.dwheeler.com ("high-assurance software").

For automotive stuff, see the MISRA C standard. C but you can't use more than two levels of pointers, and some other stuff like that.

adahome.com has good info on Ada. I liked this C++ to Ada tutorial: http://adahome.com/Ammo/cpp2ada.html

For hard real-time, Tom Hawkins has done some interesting Haskell stuff. See: ImProve (language incorporates an SMT solver to check verification conditions) and Atom (EDSL for hard realtime concurrent programming without using actual threads or tasks).

answered May 8 '12 at 19:48
solrize
41 1

There's a MISRA C++ now too. Also, although those standards are developed by and for the motor industry, they are probably suitable for any safety related development. – Richard Corden May 9 '12 at 10:44

HAL/S is used for the Space Shuttle.

answered Dec 14 '08 at 6:30
TraumaPony
7,284 11 46 67

4 There are a number of old Aerospace languages (e.g. JOVIAL) that do the rounds. Ada was supposed to unify these and has to some extent, but there is still quite a lot of code written in these languages that is still in production. – ConcernedOfTunbridgeWells Feb 26 '09 at 22:17

A new **safety-critical standard for Java** is currently in development - JSR 302: Safety Critical Java Technology.

The **Safety-Critical Java** (SCJ) is based on a subset of RTSJ. The goal is to have a framework suitable for the development and analysis of safety critical programs for safety critical certification (DO-178B, Level A and other safety-critical standards).

SCJ for example removes the heap, which is still present in RTSJ, it also defines 3 compliance levels to which both application and VM implementation may conform, the compliance levels are defined to ease certification of variously complex applications.

Resources:

- Java for Safety-Critical Applications.
- JSR 302: Safety Critical Java Technology
- Developing Safety-Critical Applications with Java

edited Dec 24 '13 at 20:41

answered Dec 24 '13 at 20:33
Ales Plsek
5,612 3 35 74

I don't know what language I'd use, but I do know what language I wouldn't:

> NOTE ON JAVA SUPPORT. THE SOFTWARE PRODUCT MAY CONTAIN SUPPORT FOR PROGRAMS WRITTEN IN JAVA. JAVA TECHNOLOGY IS NOT FAULT TOLERANT AND IS NOT DESIGNED, MANUFACTURED, OR INTENDED FOR USE OR RESALE AS ON-LINE CONTROL EQUIPMENT IN HAZARDOUS ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS IN THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION OR COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL, DIRECT LIFE SUPPORT MACHINES, OR WEAPONS SYSTEMS, IN WHICH THE FAILURE OF JAVA TECHNOLOGY COULD LEAD DIRECTLY TO DEATH, PERSONAL INJURY, OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE.

answered Oct 28 '08 at 13:53
changelog
3,199 2 27 54

3 This is just mandatory stuff from Sun to cover their ass in case they would get sued for some serious

mistake by some developer. It's clear that secure software is not so much related to the language but to the developers who write it. And in the USA you can get sued for anything. – nkr1pt Oct 28 '08 at 14:15

9 so they left out "JAVA CAN DESTROY THE EXISTENCE OF EXISTENCE" – Jimmy Oct 28 '08 at 14:38

4 @nkr1pt : 1. "collision detection" in Java ?? are we talking about online collision detection ? This is not even done in C, but electronically to be fast enough. 2. Data analysis : We use ROOT, a C++ multipurpose data analysis package. Never heard of anyone using something else for the analysis. – Barth Oct 28 '08 at 14:39

5 I see why Java's not suitable for nuclear reactor ops: "Drop the reactor control rods right- ... hmm, time for garbage collection ... -now!" – Piskvor Nov 27 '08 at 12:06

4 See JSR-1/JSR-282 ("The Real-Time Specification for Java") and the not-yet-finalized Safety-Critical Profile for Java. Much stronger guarantees about garbage collection are now available, and ways of avoiding garbage collection entirely are provided by RTSJ and envisioned in SCJ. – andersoj Apr 19 '09 at 17:12

|

Java is a nighmare language for so many reasons. It was designed by an idiot who misunderstood the Pascal and Oberon projects of Prof. Wirth.

ADA was a language designed by a large commmittee, and the resulting sprawl reminds me so much of PL/1 which was wonderful, but so complicated to write a compiler that nobody picked it up.

Modula-2 is probably the simplest language ever devised, and instead of C, i have used modula-2 with a code size half the lines (and therefore runs twice as fast). C is just one step above assembler, and just by breathing too hard you can create a nasty bug.

Pascal and basic are very reliable languages. In fact, the Visual Basic 6 compiler/toolkit is probably the best thing MS ever produced, and people still use it 15 years after it was abandoned by MS. DOn't get me started on the abomination that is .NET, the most horrendously complicated steaming pile of crap to come out of MS, which wanted to create a proprietary system that nobody could ever clone. Too bad nobody wants to clone it! they succeed too well in making something obscure.

Eiffel is intrinsically reliable, because it uses tiny sub-processes with their own stacks and heaps that get collected when the sub-task ends, so you don't fragment memory. But good luck understanding Eiffel, it was the work of a madman. The same goes for Miranda, and so many of the academic languages which were designed by math freaks instead of people who are used to accomplishing something practical.

I would say that Python is one of the best languages. Easy to read, fast, and simple. It it not particularly safe, but for scripting, it beats the pants off Bash shell scripts, or heaven forbid, the atrocious write-only language called Perl.

answered Nov 25 '14 at 8:13

ekbart
5