

ELMA
Your Solution Partner

RF and Optical Open VPX
3U and 6U Backplanes



THE MAGAZINE OF RECORD FOR THE
EMBEDDED COMPUTING INDUSTRY

Read
our
latest
edition



HOME

ARCHIVES

WHITEPAPERS

ABOUT US

ADVERTISE

SUBSCRIBE

MAGAZINE SECTIONS

DEPARTMENTS

BROWSE BY TECHNOLOGY

Search Articles

SOFTWARE & DEVELOPMENT TOOLS

Safety-Critical Software

Certification Requirements for Safety–Critical Software

Safety-critical software means that failure of the software will very probably result in loss of human life. Providers of safety-critical systems exercise extreme caution against system failure. Additionally, most industries are closely regulated by government agencies with stringent certification requirements.

KELVIN NILSEN, AONIX

June 2004

Print

E-mail

A A A

Order a Reprint

Page 1 of 1

SHARE

Each industry and geo-political region is served by different safety certification standards. For example, IEC 60880 describes European standards for certification of nuclear power generating software. DO-178B and its European analog ED12B establish guidelines for certification of commercial avionics applications. And IEC 61508 describes a general-purpose hierarchy of safety-critical development methodologies that has been applied to a variety of domains ranging from medical instrumentation to electronic switching of passenger railways. Intelligent devices play increasingly critical roles, including drive-by-wire consumer vehicle subsystems, medical monitoring and diagnostic equipment. With the expansion of the safety-critical software market, there's an increased opportunity for synergy and leverage between different submarkets.

Although the specifics may differ from one safety-critical market segment to the next, the general principles are the same. And regardless of the specific industry, a commonly accepted rule of thumb is that development of safety-certified software costs roughly 10 times as much as non-certified software with equivalent functionality.

Notably, almost all safety-critical applications have hard real-time constraints. And it is well-known that development of hard real-time software is itself much more difficult than development of typical consumer or enterprise software applications. Given this, line for line, the cost of developing safety-critical software is likely to be 20 to 30 times the cost of developing typical management information software.

Using the example of DO-178B we can see how safety-critical software is certified for commercial avionics applications. The principles manifest in DO-178B certification apply in general to other application domains as well.

Isolation of Concerns

An important software engineering technique for managing complexity is to separate concerns between individual components and responsibilities. In safety-critical systems, developers identify the levels of particular functionality according to how critical they are to the survival of the system. They then separate all of the resources required for less-critical functionality from those used by the more critical functionality. Failure of a Level-A component is considered to be catastrophic, meaning that further flight and/or landing is considered impossible. Failure of a Level-B component is considered to be hazardous and severe, significantly reducing safety margins but not necessarily resulting in loss of life. Failure of a Level-C component is considered to be major, reducing the ability of the crew to cope but not necessarily decreasing the system safety margins, and so on.



RTC SUPPLEMENTS



DO-178B guidelines require much more stringent methodologies for Level-A software than for Level-B software. Furthermore, the guidelines generally prohibit coexistence of functionality from different criticality levels on the same computer. All of the software running on each computer subsystem must be certified to the same high level as the most critical functionality implemented on that computer. The standard practice has been to isolate different criticality levels on different computers.

A relatively recent improvement over past practice is enabled by the new ARINC-653 standard for time- and memory-partitioned flight computers. When using a properly certified implementation of the ARINC-653 standard, it is possible to combine Level-A and Level-C software on the same computer. The ARINC-653 partitioning protocols ensure that CPU and memory resources dedicated to each partition are not compromised by software running in other partitions.

Audit trails are very important as one of the artifacts required for Level-A certification. Developers must show that functionality requirements can be traced to system architectures, designs, source code, compiler-generated machine language and the test plan. Every artifact produced during the development process must be reviewed by independent expert peers, and each reviewer is expected to personally certify that the reviewed artifact satisfies its intended objectives. Of the many practices imposed by the DO-178B guidelines, the codification of peer review, individual accountability and audit trails is dominant. Exceptions to the testing and isolation guidelines are occasionally permitted as long as the accountability criterion is satisfied.

Testing

The testing required for Level-A certification is very rigorous. Much of the commercial software that we use in typical management information applications and even in many mission-critical deployments could never be tested to the satisfaction of safety-critical certification authorities. Only relatively small and simple programs can be tested to the degree required for safety certification. Developers who are writing code for safety-critical systems must carefully design testability into the application code they develop.

Consider, for example, the C source code for the max() function shown in Figure 1. This code is contrived to demonstrate the test code coverage requirements imposed by DO-178B Level-A guidelines. Even though it appears straightforward as a source code listing, this code probably would not satisfy the style guidelines imposed by the software quality team on developers of safety-critical code.

While developers of DO-178B Level-A software are encouraged to use high-level programming languages and analysis tools to reduce the likelihood of programmer error, they are required to perform all test coverage analysis in terms of the translated machine language programs. For reference in discussing the testing requirements associated with this code, an assembly-language translation of this program is provided in Figure 1.

```
/* Return the maximum of its 4 integer arguments */
int max(int a, int b, int c, int d) {
    if ((a > b) && ((a > c) & (a > d)))
        return a;
    else if ((b > a) && (b > c) && (b > d))
        return b;
    else if ((c > b) && ((c > a) & (c > d)))
        return c;
    else
        return d;
}
```

Figure 1 © source code listing for finding the greater of four integers.

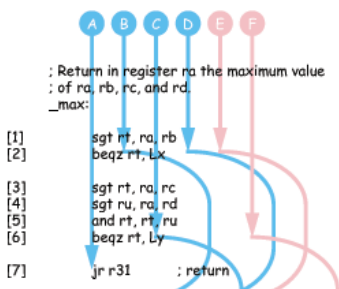
Table 1 details six parameterizations of this function that endeavor to provide all of the coverage required for Level-A certification of the code corresponding to this function. Testing of DO-178B Level-C software components must provide "statement coverage," which means that every machine instruction is executed at least once. Level-B software components also require "decision coverage," meaning that each destination of every branch instruction is exercised at least once. Level-A certification additionally requires "modified condition decision coverage" (MCDC), which is officially defined as:

	Test Vector	ra	rb	rc	rd	Code Sequence
Statement Coverage	A	5	4	3	2	1, 2, 3, 4, 5 6, 7
	B	4	5	3	2	1, 2, 8, 9, 10, 11, 12, 13
	C	5	4	6	3	1, 2, 3, 4, 5 6, 14, 15, 16, 17, 18, 19, 20, 21
	D	2	2	4	5	1, 2, 8, 9, 16, 17, 18, 19, 22, 23
Decision Coverage	E	4	5	3	6	1, 2, 8, 9, 10, 11, 22, 23
	F	5	4	3	6	1, 2, 3, 4, 5, 6, 14, 15, 22, 23

Table 1 Test data to exercise max() function.

"every condition in a decision in the program has taken all possible outcomes at least once, every decision in the program has taken all possible outcomes at least once, and each condition in a decision has been shown to independently affect that decision's outcome. A condition is shown to independently affect a decision's outcome by varying just that condition while holding fixed all other possible conditions" – Software Considerations in Airborne Systems and Equipment Certification, Document No. RTCA/DO-178B

As described in Table 1 and illustrated in Figure 1, test vectors A-D provide full statement coverage. Note that decisions (conditional branches) are made by instructions [2], [6], [9], [11], [15] and [19]. For each of these decisions except numbers [11] and [15], these first four test vectors already exercise both possible paths through the conditional branch. The purpose of test vector E is to exercise the branching path through instruction [11]. The purpose of test vector F is to exercise the branching path through instruction [15].



To demonstrate compliance with the MCDC coverage requirement, focus on the two decisions in this simple program that involve multiple conditions. The decision at instruction [6] involves the conditions represented by instructions [3] and [4]. The decision at instruction [19] involves the conditions represented by instructions [16] and [17]. First, consider conditions [3] and [4]. With test vectors A, C and F, these two conditions have values TT, FT and TF respectively. Comparing the results for vectors A and F, we see that the second condition independently affects the decision outcome. Comparing the results for vectors A and C, we see that the first condition independently affects the decision outcome.

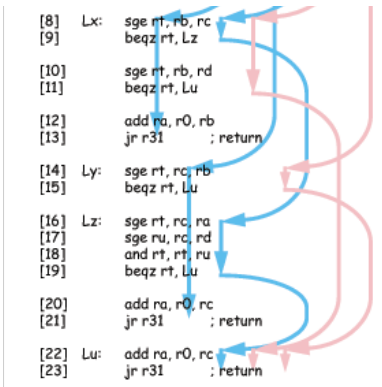


Figure 2 Assembly code listing for finding the greater of two integers showing execution traces through the code.

Next, look at conditions [16] and [17]. With test vectors C and D, these conditions have values TT and TF respectively. These two vectors demonstrate that the second condition independently affects the decision outcome. To complete the MCDC test coverage analysis, we need to find a test vector for which conditions [16] and [17] have values FT, and demonstrate that the decision outcome is different than when these two conditions have values TT.

Unfortunately, no such test vector can be found! In terms of the original C code, if c is less than a and c is greater than or equal to d, this would mean that neither the variables c nor d are the maximum value. Thus, for any set of parameters that would satisfy the desired criteria, control cannot reach the point of this decision. This function is therefore not testable according to the rigors of MCDC testing required for Level-A criticality.

System View

DO-178B guidelines do not allow individual software components to be certified. Only the complete system, which is typically integrated from many independently developed components, can be safety certified. No amount of unit testing will satisfy the FAA-authorized safety auditor. Instead, all tests must be performed on the integrated system and must be directly derived from the functional requirements of the complete system. If running these functional tests does not provide the testing coverage required for Level-A certification, this reveals a development inconsistency that must be resolved before the system can be certified. For example:

Perhaps the original system requirements are incomplete. It may be that the programmer over-generalized the original requirements and implemented a system that is truer to the original intent than was specified in the requirements. However, it is not appropriate for programmers to extrapolate on provided requirements. If the original requirements overlooked particular issues, those requirements must be revised and the impact of these changes must be traced through to the architecture, design, models, code and test plans.
Perhaps the original system requirements were complete, but the test cases that were designed to ensure compliance with the original requirements were not adequate. In this case, the team responsible for functional testing must augment its test plan.
Perhaps the original system requirements were complete, but the programmer over-engineered the solution, resulting in code that serves no useful purpose in meeting the specified requirements. In this case, the extra code is dead code with respect to the intended application and must be removed from the system.

Another caution with regards to testing of the complete system is that while MCDC testing does force a particular discipline on the structure of the source code and on the testing of the complete system, it does not guarantee that the code is correct. It is quite possible, for example, that MCDC test coverage will not detect that a particular greater-than-or-equal test should really test for greater-than (but not equal). Once again, the software architects, developers, quality assurance engineers and auditors must exercise good judgment and accountability in certifying that the software is safe. MCDC coverage analysis is just one of many means recommended to increase confidence that the complete system is indeed safe.

Development Support

Given the expense and effort required to develop and certify safety-critical software, there is considerable interest in buying components rather than building them. This also applies at the tool level. Developers of safety-critical systems need tools to help reduce the costs of developing and certifying any components that they build themselves.

Here are some of the kinds of commercially available products that can simplify the development of safety-critical systems:

Real-time executives designed to support the specific needs of safety-critical software tend to be smaller and simpler than typical real-time operating systems. Full source code and safety certification artifacts including development audit trail and test plans are generally available for an extra licensing fee.
Analytical tools determine whether existing tests perform sufficient code coverage that meets particular certification requirements. Note that this tool itself must be "qualified" correct or its analysis must be independently verified.
Tools for developing test cases fully exercise the system, providing all of the code coverage required to meet particular certification requirements.
Tools that assist with requirements capture and traceability from requirements to designs, source code, machine language and test plans ease the certification process.
Certification management tools assist with gathering and managing all of the certification artifacts that must be provided when certifying agencies audit your development.

Co-Safety vs. Mission-Critical Development

The differences between mission-critical and safety-critical development are primarily economic. Clearly, owners of mission-critical systems would like those systems to be just as reliable and safe as safety-critical systems. However, they typically cannot justify the extra development effort, runtime resources and costs required to satisfy safety certification requirements.

Over the last 20 years, there's been an interesting trend in the development of mission-critical systems. Less than half of the customers who license safety-critical Ada runtime components use these components in safety-critical products. Instead, they use these components in mission-critical systems, for which they do not need to license the certification artifacts available separately. Nevertheless, they apparently take comfort in knowing that these certification artifacts are available and that the technology itself has been subjected to rigorous accountability standards required for safety-critical software.

In the Open Group's standardization of safety-critical Java technologies, the Java subset that is recommended for safety-critical development is very limited. We expect that implementations of this standard will be available from multiple suppliers, accompanied by appropriate DO-178B Level-A certification artifacts. Based on the trends we have observed in the Ada industry, we expect that certain developers of hard real-time mission-critical systems will use the safety-critical Java standard as a foundation for their development activities.

In those sorts of applications, developers would not require the certification artifacts and would be likely to incorporate certain capabilities that are absent from safety-critical Java because these features could not be certified to the satisfaction of Level-A auditors. These capabilities offer easier development, improved dynamic behavior and lower deployment costs. Examples include the ability to dynamically allocate and de-allocate memory and to

dynamically load new software libraries and device drivers, asynchronous transfer of control including timeouts and selective abortion of running tasks, and priority inheritance to supplement the immediate priority ceiling synchronization protocol.

We can all take comfort in knowing that developers of safety-critical software are required to follow very stringent development methodologies when building the products upon which our lives depend every day. These methodologies add significantly to the costs of development. As safety-critical systems expand their roles throughout modern society, we will see increasing use of commercial off-the-shelf technologies in support of the safety-critical software development effort.

Aonix
San Diego, CA.
(858) 457-2700.
[www.aonix.com].

0 Comments

RTC Magazine

1

Login

Recommend

Share

Sort by Newest




Start the discussion...

Be the first to comment.

ALSO ON RTC MAGAZINE

New System Architectures to Interface High Data Rate Sensors


1 comment • 3 years ago



James Falasco — It seems that this solution set would be ideal to play into new mission payloads for the ...

Big Data and Small Things: prpl Foundation Aims for Compatible ...


1 comment • 2 years ago



Cesare Garlati — Securing the Internet of (broken) Things: join me in Taipei Fri 9/11at 12:10pm - Imagination ...

Libérer le potentiel des "Big Data" en Basse-Power Wireless Sensor ...

1 comment • 4 years ago



Mark Gray — I found this very informative. Thank you. In your table, for Bluetooth, that does not represent ...

Extending VME Life to 2020 and Beyond

1 comment • 2 years ago



Iván García — IOxOS Technologies (www.ioxos.ch) is releasing since last February the ALTHEA 7910 solution, ...

Subscribe

Add Disqus to your siteAdd DisqusAdd

Privacy

RTC, the magazine of record for the embedded open systems industry, covers the latest in hardware, software and peripherals for board and subsystem level solutions to embedded and real-time challenges. With strong market analysis and technical content, RTC has become the magazine engineers and managers rely on for in-depth coverage of this developing and expanding marketplace.

Site Map:

- Home
- Subscribe
- Advertise
- Resources
- About Us
- Contact Us

Magazine Sections:

- Technology In Context
- Solutions Engineering
- Industry Insight
- System Integration
- Products
- Editorial

Other Sites:

- Intelligent Systems Source
- COTS Journal
- RTECC
- MEDS Magazine
- RTC Group