print | close

Solve The Software Standards Maze

Electronic Design
Mark Pitchford
Mark Pritchford, Contributing Technical Expert
Thu, 2014-01-30 12:47

Research by the U.S. National Institute of Security Technology found that 64% of software vulnerabilities stem from programming errors. Such findings explain the increasing trend of governments, industry watchdogs, and OEMs to require suppliers to adhere to recognized safety-critical standards. With security-critical standards also increasing in prominence, the plethora of standards is confusing, especially for suppliers new to such requirements.

Related Articles

- MISRA C: 2012: Plenty Of Good Reasons To Change
- Are You Writing Safe And Secure Software?
- What Does It Mean To Say That Code Must Work All The Time?

Critical software has to work, and work properly, no matter what industry it is applied to. So, why are there endless variants of process and coding standards? Why is there a distinction between a critical medical application and one for a railway or an automobile? And why is safe code different from secure code?

Whether you're developing safety-critical software systems, two categories of standards dictate how you should go about it: process standards and coding standards. Process standards detail the process you must follow in the design and development of the system as a whole. Software coding standards outline high-level programming language rules that help you to write safe code.

Industry Process Standards And Software Certification

Most process standards prescribe a methodology for the development of a safety-critical application, and they usually deal with both hardware and software considerations. Regardless of the industry, process standards include several common threads:

• Requirements traceability

• Use of established design principles

• Application of software-coding standards

• Control and data flow analysis

• Requirements-based test

• Code coverage analysis

The need for these processes has accumulated throughout several industries. In medical, for instance, software failures were a causal factor in the federal Food and Drug Administration (FDA) requiring Cardiac Science Corp. to replace 24,000 defibrillators—a consequence that resulted in a reported net loss of $18.5 million and the company's eventual acquisition by Opto Circuits.

The standard IEC 62304 applies to such medical equipment applications and is designed specifically to provide suitable processes to minimize the likelihood of other medical device projects falling victim to similar problems. Other industries are taking a similar approach. For example, avionics uses DO-178B (first published in 1992) and DO-178C, Level E to Level A. Industrial uses IEC 61508 (first published in 1998, SIL Level 1 to 4), comprising:

• Railway: CENELEC EN50128 (first published in 2001, SIL Level 0 to 4)

• Nuclear: IEC 61513 (first published in 2001, SIL Level 1 to 4)

• Automotive: ISO/DIS 26262 (draft, ASIL A to ASIL D)

• Medical: IEC 62304 (first published in 2006, Class A to Class C)

• Process: IEC 61511 (first published in 2003, SIL Level 1 to 4)

For the most part, the standards derived from IEC 61508 are similar in that they set out the processes (including a risk management process), activities, and tasks required throughout the software lifecycle, stipulating that this cycle does not end with product release but continues through maintenance and problem resolution as long as the software is operational. Ultimately, regardless of how they specify the level of acceptable or unacceptable risk, IEC 62304, IEC 61508, and other like standards provide guides and measures that we must use to demonstrate that our system meets its safety requirements.

One example of how this commonality of purpose shows itself is in the use of classifications according to risk, often called safety integrity levels (SILs). Each standard requires a risk assessment to be completed for every software project to decide the appropriate assessment safety level of each part of the system. The more significant the risk of failure for that system, the more rigorous the process needs to be and the more thorough the testing that will be necessary.

However, these SILs also highlight one example of how each standard is tuned to existing best practice in the industry it targets. Unlike IEC 61508 and its other derivatives, IEC 62304 does not define common numerical values for acceptable failure rates. Instead, it defines safety classifications according to the level of harm a failure could cause a patient, operator, or other person. These classifications are analogous to the FDA classifications of medical devices: A (no possible injury or damage to health), B (possibility of non-serious injury or harm), and C (possibility of serious injury or harm, or death).

Requirements Management And Traceability

Requirements traceability is widely accepted through the process standards as a development best practice to ensure that all requirements are implemented and that all development artefacts can be traced back to one or more requirements. For example, the automotive industry's ISO 26262 requires bidirectional traceability and has a constant emphasis on the need for the derivation of one development tier from the one above it and for the links between these tiers to be maintained at all times.

Bidirectional traceability helps determine that all source requirements have been completely addressed and that all lower-level requirements can be traced to a valid source. It also covers the relationships to other entities such as intermediate and final work products, changes in design documentation, and test plans.

Last-minute changes of requirements or code made to correct problems identified during test tend to put such ideals in disarray. Despite good intentions, many projects fall into a pattern of disjointed software development in which requirements, design, implementation, and testing artefacts are produced from isolated development phases. Without a coherent mechanism to dictate otherwise, this isolation results in tenuous and implied links between requirements, the development stages, and/or the development teams, such as that link created by a developer reading a design specification and using that to drive the implementation.

Placing a requirements traceability matrix (RTM) at the heart of a project addresses this issue and encourages the links to be physically recorded and managed *(see the figure)*. This alternative view of the development landscape illustrates the fundamental centrality of the RTM. The overhead associated with the RTM and maintenance can be daunting, but the use of appropriate tools can ensure that the RTM is automatically maintained and up-to-date at all times.

Safety Coding Standards

Coding standards are specifically concerned with how particular high-level languages should be applied to achieve a particular end. They consist of a set of rules or restrictions on the use of a specific coding language, often C or C++.

Many organisations such as MISRA, JSF AV, and the Barr Group have developed coding standards concerned with safety. Many later coding standards refer to or acknowledge MISRA and add variations to deal with such as multi-threaded applications.

Process standards usually require that use of a coding standard as well. None of the process standards insist on the use of a particular coding standard, so individual organizations are at liberty to adopt one of the existing rule sets, adapt it to suit their circumstances, or write their own from scratch.

Perhaps the best known is the MISRA family of standards. For example, MISRA C is a language subset of the C programming language that was developed and is maintained by the Motor Industry Software Reliability Association (MISRA). Originally designed to promote the use of the C language in safety-critical embedded applications within the motor industry, the original version was released to target C90 in 1998 (MISRA C:1998).

Over the years, the language subset has gained widespread acceptance for safety-critical and mission-critical applications in aerospace, telecom, medical, defense, railway, and other industries. The latest update, MISRA C:2012, supports ISO 9899:1999 (C99) while retaining support for C90. It helps mitigate software-related risks for safety-critical applications while being as unobtrusive as possible.

Security Standards

Just as there are process and software coding standards to address safety concerns, there are equivalent process and software coding standards developed for security. Process security standards are in their relative infancy, and there is considerable variation between industries on what is considered best practice. Some organizations such as Microsoft and McAfee offer consultancy services to enable clients to source a custom secure software development lifecycle (SSDLC) designed with their specific security needs in mind.

The Automation Standards Compliance Institute's recommendations are perhaps more suited to the embedded markets. They suggest the implementation of a software development security assessment, which references existing process standards such as IEC 61508 and ISO 26262 and superimposes best practice security measures on them.[1]

The International Organisation for Standards (ISO) recently published a set of security-focused coding rules, often referred to as secureC.[2] Like safety coding standards such as MISRA, SecureC can be applied across many industries. CERT C provides an alternative set of similar rules, while the Mitre Corporation's Common Weakness Enumeration "provides a unified, measurable set of software weaknesses," which essentially is a list of practices to avoid.[3]

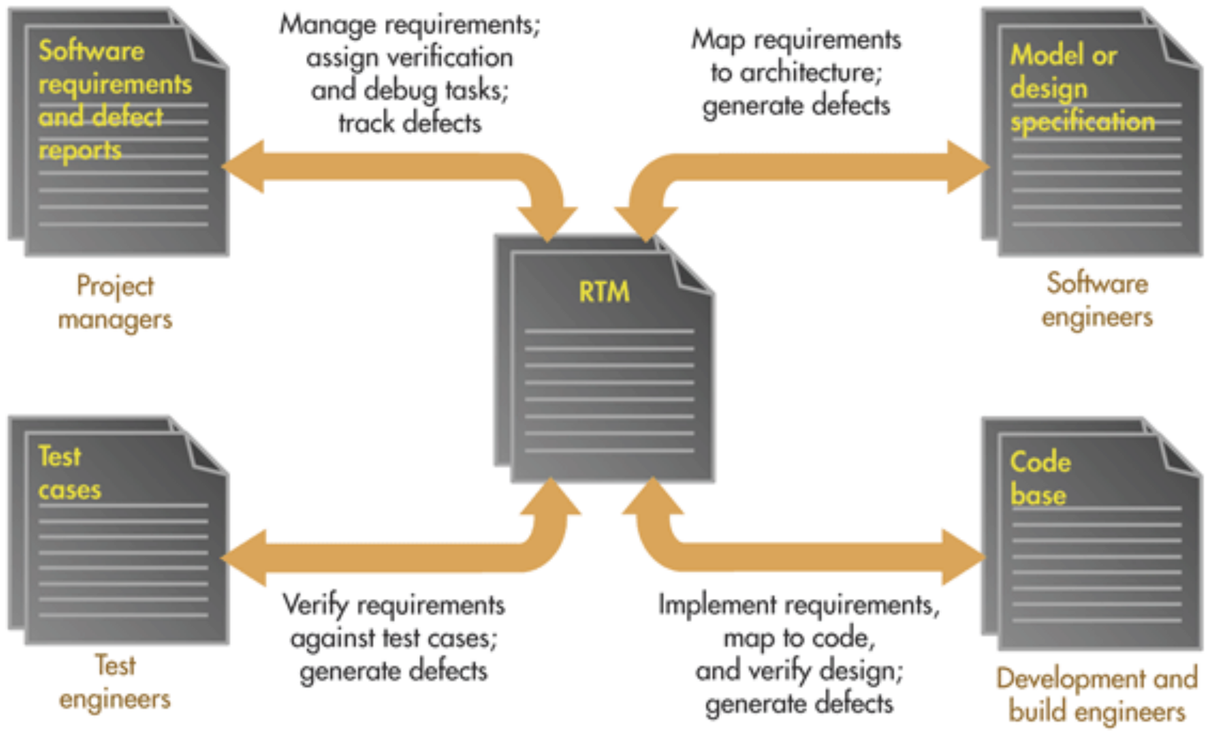
Conclusion

The standards maze is much easier to negotiate if you map out your path according to your focus: process or coding, safety, or security.

Many of the established safety process standards are industry specific. DO-178 provides process guidance for the aerospace industry, ISO 26262 for the automotive sector, and IEC 62304 for medical devices, to name a few. Complicating things, each of these process standards includes variants depending on system criticality. The more severe the consequences of system failure, the more rigorous the demands each standard makes.

Safety coding standards often originate from specific industry sectors, such as MISRA C from the automotive industry and JSF AV++ from aerospace. Despite that, in practice they are often sufficiently generic to be applied to a wide range of industrial sectors.

Security standards can similarly be subdivided into two groups, for process and coding. There are several established examples of security coding standards such as secureC, but security process standards are in their relative infancy. Perhaps the most appropriate to embedded applications are those that build upon the existing process standards, enabling safety and security to be addressed as a coherent whole.


Software requirements and defect reports
Project managers
Manage requirements; assign verification and debug tasks; track defects
Map requirements to architecture; generate defects
Model or design specification
Software engineers
RTM
Test cases
Test engineers
Verify requirements against test cases; generate defects
Implement requirements, map to code, and verify design; generate defects
Code base
Development and build engineers

References

1. ISA Security Compliance Institute Update, John Nye, Nate Kube, Andre Ristaino, http://ics-cert.us-cert.gov/sites/default/files/ICSJWG-Archive/F2010/Ristaino%20-%202010%20Oct26%20ICSJWG.pdf

2. “Information Technology—Programming languages, their environments and system software interfaces—C Secure Coding Rules,” document number ISO/IEC TS 17961:2013

3. Common Weakness Enumeration, http://cwe.mitre.org/

Mark Pitchford *has more than 25 years of experience in software development for engineering applications. He has worked on many significant industrial and commercial projects in development and management, both in the U.K. and internationally including extended periods in Canada and Australia. For the past 10 years, he has specialised in software test and works throughout Europe and beyond as a field applications engineer with LDRA. He can be reached at mark.pritchford@ldra.com.*

Source URL: http://electronicdesign.com/embedded/solve-software-standards-maze