print | close

11 Myths About Software Qualification and Certification

Electronic Design

Jay Thomas

Wed, 2016-04-20 10:47

With software taking on an ever-greater role in embedded systems, companies are realizing that "quality code" requires more than just the developer's claim. Even for systems that don't require formal certification for functional safety or security, software qualification is becoming more common. After all, who really wants to risk expensive field support, product recalls, or even legal action if software fails? Still, at least 11 myths continue to circulate about software qualification and certification.

1. Software certification only applies to avionics applications/doesn't apply to my industry/isn't possible.

This myth is often tied up in semantics. For instance, the U.S. aerospace industry requires software to be qualified to specific standards in order to certify systems. And while software isn't required to be certified in other U.S. markets yet (the situation varies by industry and geography), discussions along these lines are underway in many safety-critical industries, including medical, nuclear power, industrial, and automotive. Whether certification is required or not, software developers who can prove compliance to international quality standards can offer confidence to both OEMs and operators that the software in the device will behave as expected.

Related

Enforcing Programming Standards Protects Against Software Security Risks

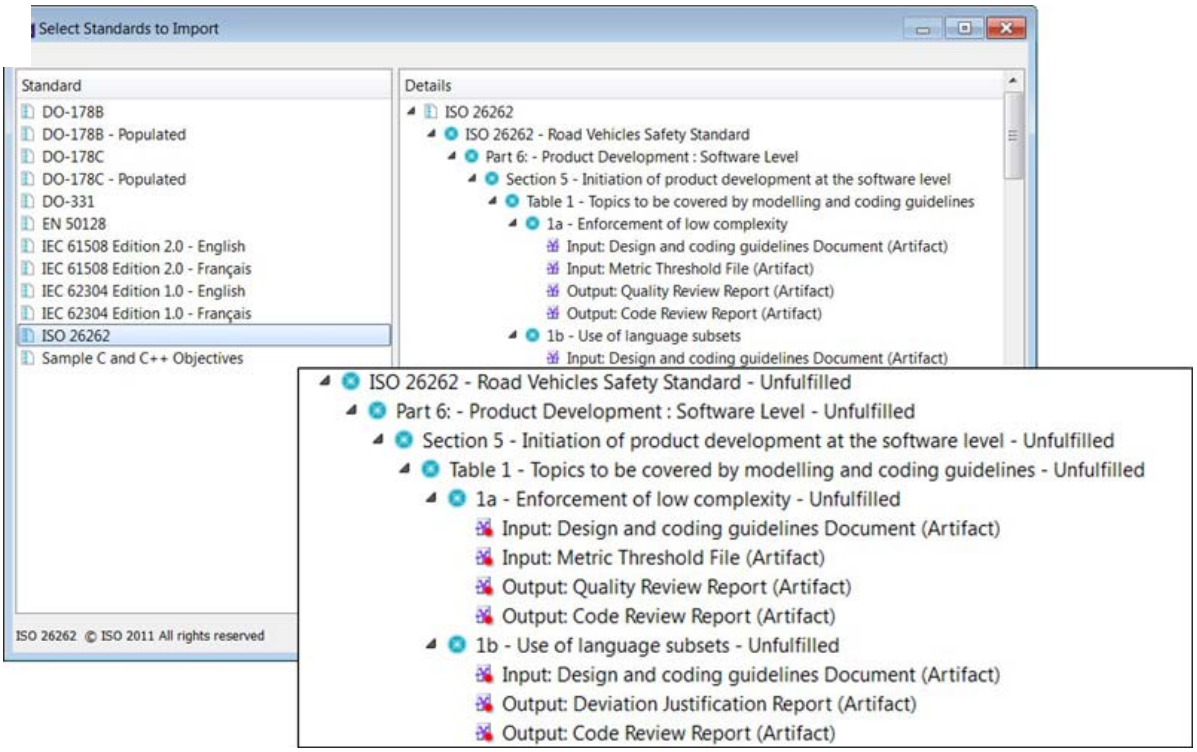
A Practitioner's Guide To Critical Software Certification

Top Five Things to Know About Software Testing

2. Software certification is only important to government regulators and industry gurus.

As the market pushes for higher-quality software, particularly in safety- and security-critical industries, software certification or qualification is becoming more mainstream. Many software development and verification organizations are being trained on compliance with functional safety and security standards, especially as compliance becomes increasingly important to the supply chain.

Even if software has yet to be formally regulated, large OEMs may require suppliers to adhere to industry standards such as IEC 61508 (industrial controls), IEC 62034 (medical), ISO 26262 (automotive), IEC 60880 (nuclear energy), and EN 50128 (rail transportation). For instance, GM is now training its software suppliers on formal development processes so they can provide an audit trail back to GM. Trained developers can gain a competitive advantage by showing that they already follow industry standards and best practices, and developers who understand these processes can gain a career advantage as well *(Fig.1)*.



3. Software qualification is too expensive.

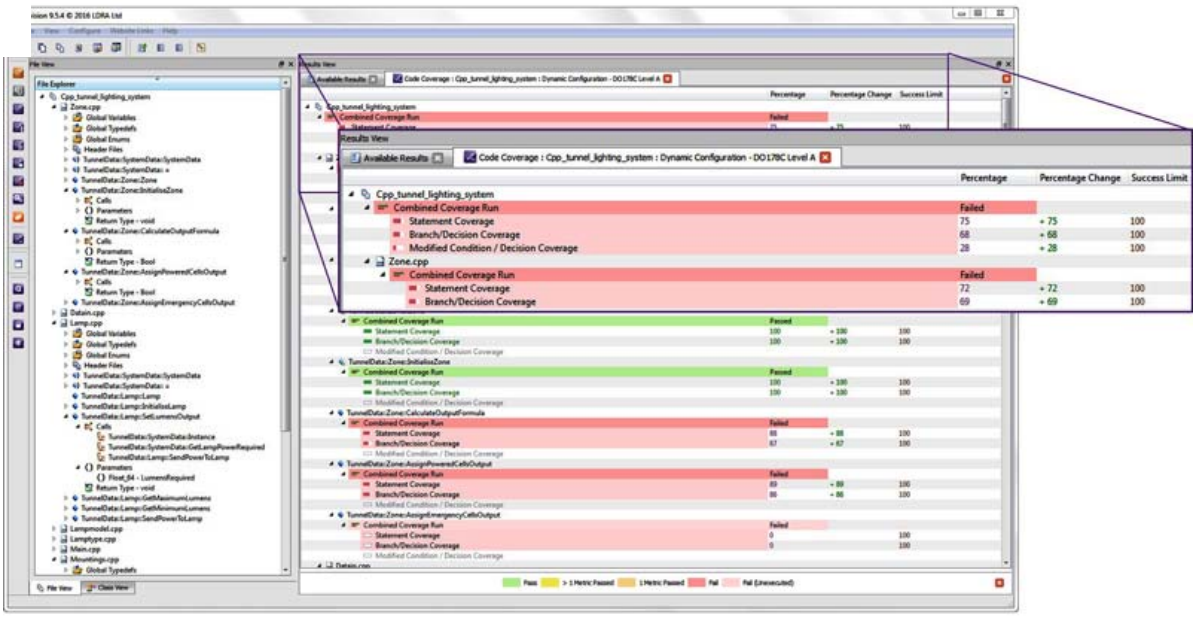
This myth is typically based on software qualification that's performed using traditional manual methods. Companies that choose to use non-automated tools, such as Word or Excel, to track processes can become hopelessly bogged down. The process of developing and verifying software needs to be formalized, repeatable, and measurable.

By leveraging automation technologies, organizations can greatly reduce the effort and cost of producing high-quality software and—depending on the market and industry standards—certified software. For this to be done cost-effectively, software organizations need to rely on automation technology to perform the menial tasks that often consume large amounts of time and human resource energy during the software development lifecycle.

4. I can do software certification cheaply without tools and automation.

Software certification is simply not possible in a cost-effective manner without software tools and automation.Without automated tools, developing and providing the evidence required for software certification can be a resource-intensive, time-consuming, and arduous task.

Luckily, many of these manually intensive development and verification tasks can be automated. Some of the manual tasks to consider for automation include requirements traceability and requirement impact analysis, static code analysis, manual code reviews, structural coverage analysis, test-harness generation, test-case generation and execution, regression testing, and documentation generation for compliance and audit trail evidence.



If automating these processes could save your development team three months, for example, (potentially 20% to 40% of the overall schedule) it could result in several hundred thousand dollars in reduced man-hour costs. Investing a comparatively small amount into the tools that automatically link requirements to written code, as well as the subsequent tests, and auto-generate documentation can save many times that amount in labor costs and provide first-to-market advantages*(Fig. 2)*.

5. Software qualification isn't worth the cost.

Depending on the market, producing software that's not high quality, certified, or qualified is simply not a business option. Companies need to perform some level of risk analysis to determine whether the costs involved in addressing market requirements of security and safety are worth it.

While formal software certification can be expensive in terms of resources, time, and money, so are product recalls, lawsuits, and brand damage, especially if a software failure results in injury or death. The question to ask is whether you can afford NOT to qualify your software to safety or security standards. We only need to look at the news to see examples such as Baxter's multimillion-dollar infusion pump recall or the Toyota recall of 625,000 hybrid cars—both as a result of software flaws.

6. You can rubber-stamp software certification after the fact.

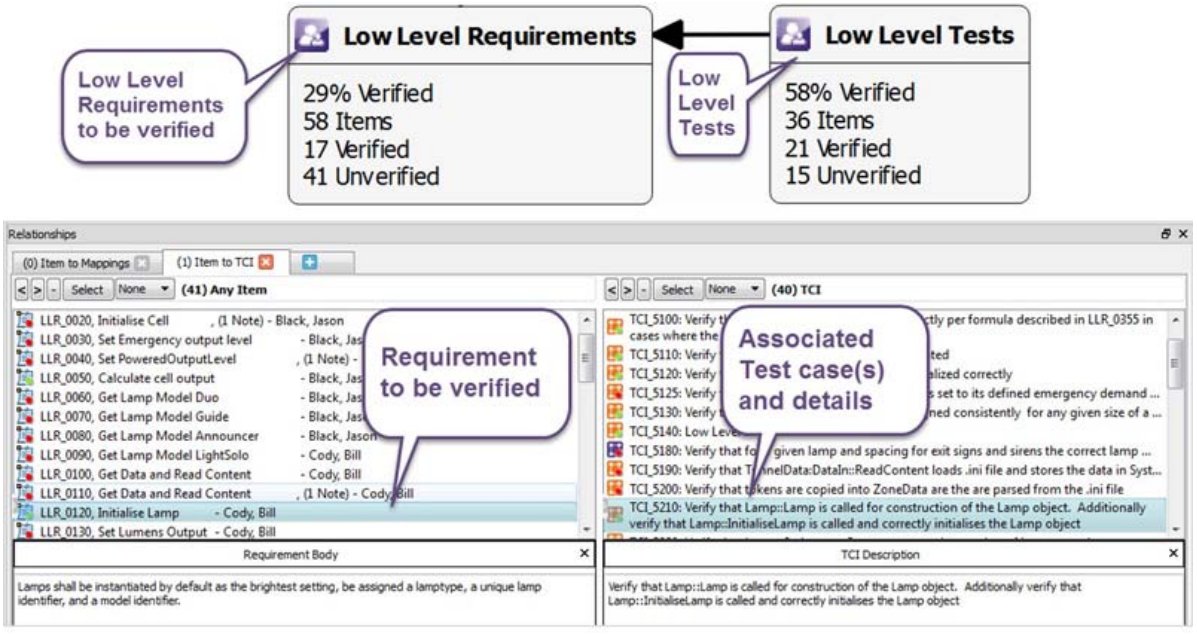
Quality is an ongoing process of design, creation, and testing. If you don't follow the process, you don't get the quality, and there's no post facto rubber stamp that will fix it. Of course, companies continue to try it—we typically read about them in the news (see comments above about Baxter and Toyota). To be performed effectively, software functional safety and security simply must be built into the product from the ground up.

7. Using static analysis is enough to ensure quality code.

Static analysis can lead to higher quality code by helping ensure that the code is clear, concise, and maintainable, and that it adheres to industry best-practice coding standards. However, it's performed without executing code, so it only addresses one part of the overall software-development lifecycle. Even if code is perfectly written, it's only correct if it meets project requirements. The bottom line? Static analysis is a great enabler, but no panacea on its own.

Software certification can be accomplished without a formal development process.

Software certification is about meeting and testing to established software requirements. So while it's possible to accomplish without a prescribed formal development process, doing so is likely cost-prohibitive.Without artifacts, you will essentially have to reverse-engineer your software after the fact in order to document development and test processes.



This is bound to be error-prone, will take additional resources and developer time, and will delay time-to-market. And if engineers have to be pulled back into the project because they are the only ones familiar enough with the code, then this approach could also impact schedules for other products. A formal development process in which certification is planned from the beginning and managed throughout the development workflow streamlines and shortens verification, making certification much more reasonable in terms of the money and resources required *(Fig. 3)*.

9. Software certification is not possible with embedded targets.

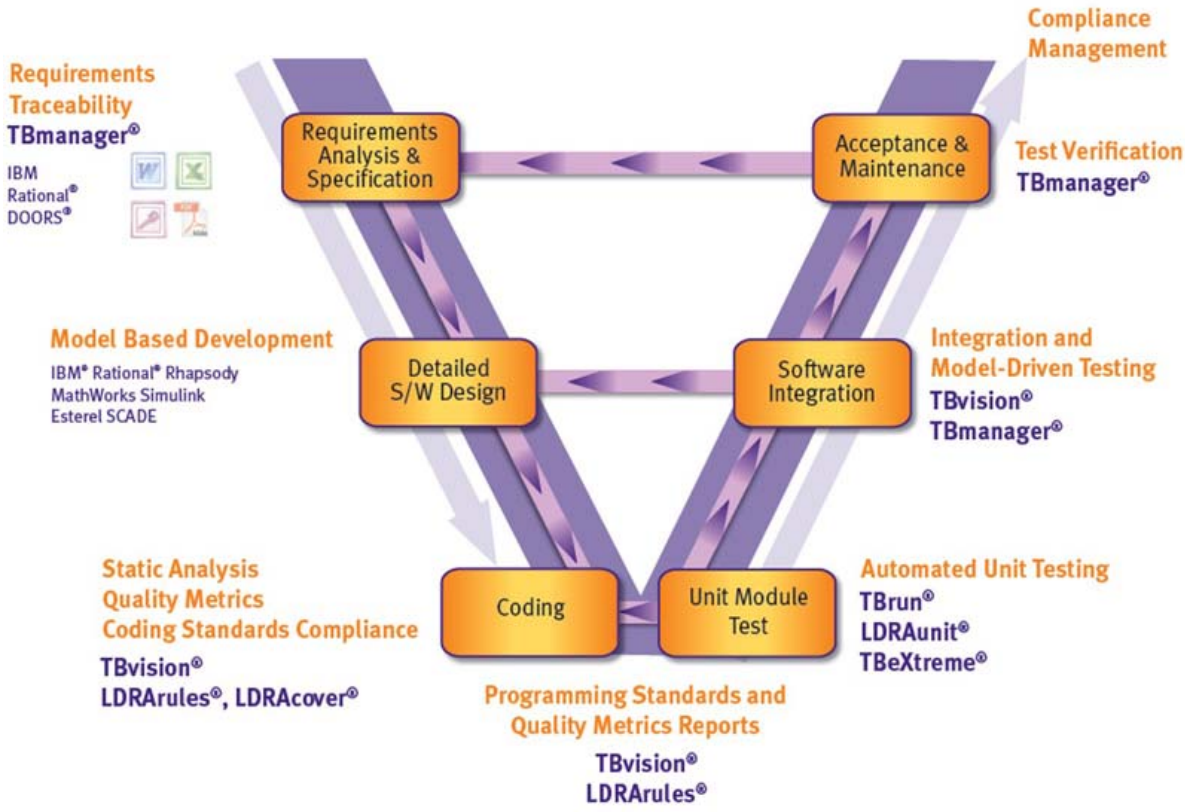
In many safety- and security-critical applications, software certification must be done on the actual hardware platform that's intended to be used for deployment. That said, software testing and verification can often be performed in virtual or simulated environments, and this simulated testing can help keep software development on track while hardware is being developed.

In those cases, it's incumbent upon the software organization to prove that the software proven in the virtual or simulated environment is the same as the software being deployed in the actual target environment. Tool suites are available, such as those from LDRA, which have probes for embedded hardware. This gives development teams the same test environment for simulated and final hardware test.

10. You can build a safe system without security.

In today's connected, Internet of Things world, an insecure system is simply not a safe system. Reports of hacked medical equipment, automobiles, and smart energy devices abound, and systems with low-quality code are simply easier targets for hackers. As a result, safety-critical software must be developed and verified with security in mind from the beginning—this isn't an either/or decision.

Software certification is not possible in an agile environment.



For some time, it was believed that only the most formal structured development approaches could produce code that is qualifiable or certifiable. That led to the well-known V model, which moves logically and sequentially from requirements to design to test *(read more at http://www.ldra.com/tool-suite)*. Today, agile development environments can be less formal on the surface; developing small pieces of code in short sprints. However, the processes used during sprints can still be automated and documented, resulting in a series of short, less error-prone, and integrated software sprints *(Fig. 4)*.

Source URL: http://electronicdesign.com/embedded/11-myths-about-software-qualification-and-certification