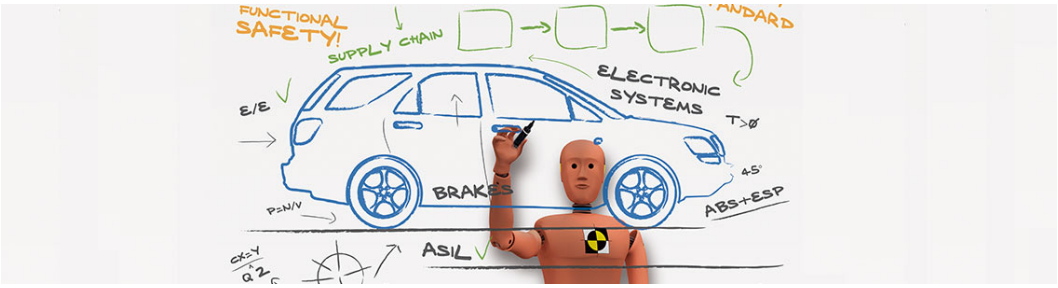
ACCELERATE THE PATH TO ISO 26262 CERTIFICATION

SHAWN PRESTRIDGE, IAR SYSTEMS

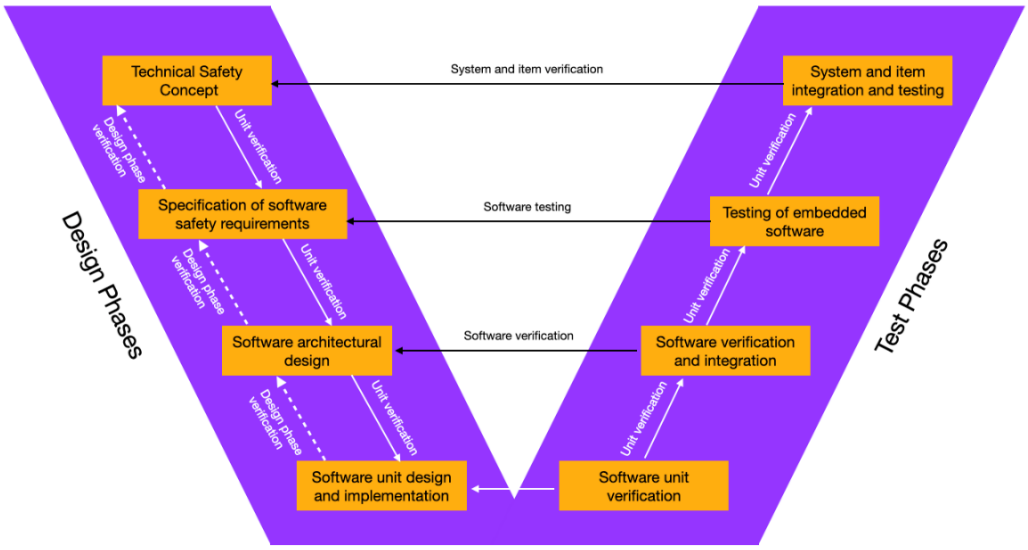
Functional safety needs to be an integral part of any automotive design. Functional safety can be defined as the part of the overall automobile safety feature set that depends on predictable automated protection, which operates in response to its inputs or failures. In other words, it's an automated way to protect the vehicle from faults, which could be human errors, or hardware or software failures. The key component here is that the process must be automated.



The ISO 26262 certification path can be complex, and the automobile's increasing number of ECUs is not making it any easier.

To ensure proper functional safety, the ISO 26262 standard was developed. It dates back to 2011, but has been updated over the years, as you would expect. As is often the case with standards, they can put some complex requirements on the software developer. The standard applies to mainstream automobiles, with exceptions made for specialty vehicles. The spec covers all aspects of product development, from the specification, to design, implementation, integration, verification, and validation, all the way through to production.

According to the specification, "ISO 26262 is a risk-based safety standard, where the risk of hazardous operational situations is qualitatively assessed and safety measures are defined to avoid or control systematic failures and to detect or control random hardware failures, or mitigate their effects." The specification acknowledges that it's impossible to reduce the risk to zero. Hence, it requires that risks be qualitatively assessed and that measures be taken to reduce them to as low as is reasonably practicable.



The ISO 26262 standard relies on the V-Model, which acts as a framework for matching requirements (left side) with corresponding tests (right side) to provide traceability.

One of the important goals of ISO 26262 is to provide a risk-based approach for determining risk classes, also referred to as automotive safety integrity levels (ASILs). The spec then uses those ASILs to help determine what constitutes "acceptable risk." It defines four ASILs, aptly named A, B, C, and D. A represents of lowest degree of hazard, with D at the opposite end of the spectrum. Components like rear lights require an ASIL-A grade, while systems like airbags, anti-lock brakes, and power steering require an ASIL-D grade, as the risks associated with their failure are the highest. In between are head lights and brake lights (ASIL-B) and cruise control (ASIL-C).

A Lengthy Spec

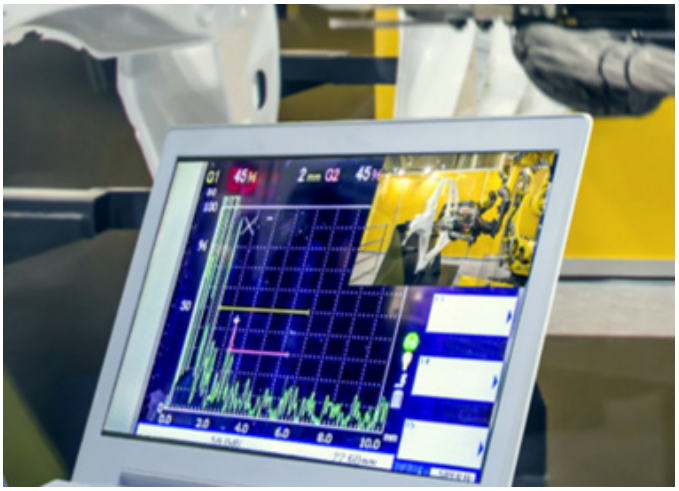
The latest version of ISO 26262 consists of 12 parts:

1. Vocabulary
2. Management of functional safety
3. Concept phase
4. Product development at the system level
5. Product development at the hardware level
6. Product development at the software level
7. Production, operation, service and decommissioning
8. Supporting processes
9. Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analysis
10. Guidelines on ISO 26262
11. Guidelines on application of ISO 26262 to semiconductors
12. Adaptation of ISO 26262 for motorcycles

We won't go through each component here, but we'll stick to the parts that will help you accelerate your path to certification, particularly the software aspects, also known as ISO-26262-6. Pertaining to the software, ISO 26262 specifies the following requirements for product development:

- the initiation of product development at the software level
- specifying the safety requirements
- the architectural design
- unit design, implementation, and testing
- integration and testing
- verification of the safety requirements

Where do you start if you want to ensure that you're writing your code in compliance with ISO 26262? The easy answer is that you focus on code quality, which goes hand-in-hand with security. For example, it's important to test for artifacts within your code. Artifacts are like maps that can be used to trace the developmental process of any piece of code. The artifacts show what's been generated in the development process, which could include mock-ups, design documents, test matrices, prototypes, notes, data models, and diagrams.



Commercial tools are available to streamline the ISO 26262 certification process, including IAR Systems' Embedded Workbench, Functional Safety Edition.

Software developers should be able to analyze and know the tools used in creating such software by looking at the artifacts. Hence, the proper toolset is required. One tool that should be on your radar is IAR Systems' Embedded Workbench, Functional Safety Edition. Available for most popular microprocessors, this special edition of Embedded Workbench is certified by TÜV SÜD according to the requirements of all relevant functional safety standards. The certification validates the entire development processes, as well as the delivered software.

The Safety Guide that's included with the Functional Safety version of Embedded Workbench has practical tips for helping you certify your application. The topics include:

- System and environment considerations
  - How to manage language standards compliance, language extensions, and subsets, potential tool failures, device-specific support files, compatibility between different versions of the same toolchain, compatibility with other toolchains, and MCU self-check strategies
- Installation, commissioning, operation, and maintenance
- Setting up the build environment
  - Debug and release modes, build configurations and options, stack depth considerations, linker configuration, and add-on analysis tools
- Implementation and coding considerations
  - Optimization modes, integral type selection, floating-point arithmetic, functions, global symbols, const and volatile, and pointers
- The C/C++ standard libraries

What differentiates the Functional Safety Edition is that IAR is guaranteeing long-term product support and providing validated service packs as the needs arise. In fact, the guarantee is not time restricted in any way, and the company is sharing renewed certificates for the product version for as long as customers need it. This differs from most vendors that typically limit support to a few years. The company also

agrees to provide regular reports of known deviations and problems. Single- or multiple-use licenses are available.

Note that there is a difference between being certified and being compliant, and it likely will depend on who your customer is as to which path you need to follow. Obviously, you need to be compliant before you can be certified, but the certification process is one that's far more time consuming as you are dealing with outside authorities.

A vendor can choose to adopt just portions of the standard and be compliant, assuming those portions are all that's needed for that application. This is true for most safety-critical applications, but we'll stick to automotive here.

Ensuring that code is safe, secure, and reliable can be difficult. You need to fulfill specific coding and design guidelines. Applying a coding standard, such as MISRA, makes it easier to verify your code against the safety standard guidelines.

As stated, ISO 26262, Part 6 defines the software methods for achieving compliance with the standard, including the modeling and coding guidelines recommended for all ASIL levels. You can use a tool like Embedded Workbench to comply with most of these guidelines for software written in C or C++.

Start with Static Analysis

Using the right tool for the right job is one of the keys to success. For example, static-analysis tools are best used to validate that the software conforms to the coding standards set forth by the specification. While the main function of these tools is to find serious programming errors, they are capable of much more, particularly when it comes to ISO 26262 and the robustness that helps prevent common errors.

Static-analysis tools help find defects by working their way through the entire program, searching for anomalies and other defects. They can do this by running a simulation with variables. The tools have the ability to explore various execution paths. While it may not be possible to explore every path, as many as possible is recommended, especially as the more traditional methods can be less successful.

Followed by Run-Time Tools

While static analysis is a useful way to spot the pitfalls in your code, run-time analysis is just as critical. Looking for failures throughout the system may seem like an arduous task, but it's a necessary one. And it gets more difficult as OEMs attempt to combine more and more functions into components and subsystems. Code must be run on each embedded control unit (ECU) separately, with the results coming back together at some point. Emulating every possible scenario in order to find every possible bug is a lofty goal, but likely unrealistic, particularly within a given time budget.

Time determinism is a key attribute here, as it deals with necessary time constraints by synchronizing time-bases for synchronized execution and support. It also detects timing violations and prevents their propagation.

Coding with MISRA C Standards

One of the best ways to help aid in the compliance of ISO 26262 is to adhere to MISRA C standards. In a nutshell, that would support the development of a system that's safe, secure, reliable, and portable. Note that MISRA (the Motor Industry Software Reliability Association) is a consortium formed by various players in the automotive industry.

MISRA C can automatically eliminate some of the more common pitfalls that developers tend to fall into. In the end, when combined with static analysis, it lets the developer focus his testing time on the less obvious issues. The MISRA standards can also remove undefined, and potentially risky, behaviors.

Proper Testing

While testing can be used simply to improve quality and safety, it's just as important to employ it to assess quality. Just because a system passes a test doesn't mean that it's ready for market. Using feedback loops, the developer can improve traceability, coverage, and robustness through a disciplined safety process. It's important that testing not be relegated to an after-thought. Rather, it should be a tool in the developer's arsenal, like any other.

When products are rolled out to end customers, if possible, a phased rollout is recommended. While using early adopters as beta testers is not recommended, that's sometimes an unavoidable situation. Like it or not, there's a chance that the first round of customers will pick up on bugs that could not be simulated in the lab. The easiest fix is an over-the-air (OTA) update, followed by a continuous series of updates.

Is Free Worth It?

Finally, we'd be remiss if we didn't discuss some of the "free" offerings. Remember the line, "you get what you pay for," as it certainly applies here. And while the upfront cost may be free, you may end up paying severely in terms of development time, and trying to figure out how to get all these pieces to work in harmony.

Obviously that cost is harder to adjudicate, but it's there. Partly because you're taking away your developer's time from the ability to produce newer and better products, products that have a higher value.

The bottom line is, are you going to pay the money upfront where the cost is fixed and you know what it is, or are you going to pay it somewhere down the road where you really don't have any idea what the cost is? That can be troubling as you don't know how long it'll take your development team to come to grips with all the different free solutions they're trying to cobble together.