KDAB

Functional safety and HMI

Why should you care?

Contents

Background

You're a developer of front-end applications. You've done the courses, watched the videos, practiced the techniques. You're all over the Qt beta releases and up to speed with them by the release date. In short, you're on top of your game.

HMI development is a specialism that is in demand. Being up to speed with the latest thinking takes all of your energy. The back-end stuff is someone else's problem, isn't it? It doesn't really matter to you what happens downstream. Right?

Well, yes. And no.

Take functional safety, for example. GUIs developed to operate industrial plant, medical devices, or aircraft are portals to safety-critical systems with the potential to cause harm, injury, or worse. This paper argues that having an understanding of the implications of compliance with functional safety standards is important, even if the necessary measures to minimise potential harm are all handled by others.

Using this document

This document assumes no knowledge of functional safety, or of the standards that define how it is to be dealt with across the sectors. If you are looking to gain an understanding of functional safety and how it impacts HMI design and implementation and have the time to read it all – that's great!

If not – stick to the white section for an overview, or dip into the detailed "colour coded" sections as you please. Each offers a complete and rounded introduction to its topic.

What is functional safety?

Functional safety minimizes risk in case of system malfunction when the wellbeing of people depends on correct operation. Although functional safety is therefore a wide-ranging concept that is applicable across the safety-critical sectors, it includes no provision to ensure that the primary functionality of the application is safe (sidebar).

Functional Safety v SOTIF

Although functional safety is a wide-ranging discipline, it is not quite all embracing. Functional safety specifically excludes the Safety Of The Intended Function (or SOTIF).

Functional safety concerns the response to system failure, whereas SOTIF concerns the mitigation of safety hazards that might occur without a system failure taking place. SOTIF applies to such as advanced driver assistance systems which need to deal with safety hazards without failing themselves.

The International Electrotechnical Commission [1] is one of several organizations that publish standards for electrical, electronic, and related technologies. The safety of people and the environment is a core underlying principle in the work they do to underpin quality infrastructure.

The IEC definition of functional safety [2] is therefore a good point of reference. It states that functional safety:

/ seeks to reduce the level of risk in a device or system
/ identifies potentially dangerous conditions that could result in harm and automatically enables corrective actions to avoid or reduce the impact of an incident
/ is part of the overall safety of a system or device that depends on automatic safeguards responding to a hazardous event
/ relies on active systems that can respond to a potentially dangerous situation

Although the definition of functional safety largely precedes the evolution of our connected world, this definition does implicitly include cybersecurity issues such as the susceptibility to bad actors (or hackers). Security vulnerabilities represent a particular category of “potentially dangerous conditions that could result in harm”.

Functional safety and the HMI

Capacitive HMI systems are becoming an integral part of “safety-critical” functionality such that a malfunction could have implications for operator and bystanders alike. For example, cars released in recent years have seen the functionality traditionally associated with a plethora of physical buttons integrated into the touchscreen HMI, lending a minimalistic and clean look to the interior as a whole.

HMI interfaces are often associated with systems that are likely to have implications for functional safety, with potential implications for the development overhead associated with them. Ensuring effective segregation between the HMI and the systems it controls is not only good software engineering practice. It also has the potential to reduce the overhead of functional safety standards compliance for HMI developers, and to make the resulting system less vulnerable to cyber-attack.

If you'd like to learn more about the implications of functional safety for HMI developers and about software system segregation in safety-critical systems, read the green section, p. 6-11.

Functional safety standards

The primary role of functional safety standards is to define a process to be followed to ensure that functional safety is handled to a level that is proportionate to the potential consequences of failure. The different standards are largely similar in nature because the majority of them are adapted from the "sector-agnostic" standard IEC 61508 [3] which in turn owes a great deal to DO-178 [4] from the aviation sector. The primary difference between the more sector-specific standards is in the use of terminology appropriate to the sector, and in ensuring that pre-existing best practice is retained where appropriate.

If you'd like to learn more about functional safety standards, read the yellow section, p. 12-15.

Proportionate safety measures

How safe is safe enough? Ideally, the most thorough safety precautions would be applied by all software system developers but that is unlikely to be commercially justifiable. Each of the standards concerned with functional safety acknowledges this and presents a scheme so that precautions are proportionate for the resulting application to be "safe enough".

This is achieved through an analysis of risk, taking into account the likelihood of each fault condition happening, and the seriousness of the consequences should it do so. The application software and subsections of it are then classified according to this

HMI interfaces impact functional safety, requiring segregation to reduce compliance overhead and cyber vulnerability. Functional safety standards ensure proportional safety measures for critical systems.

analysis, and the effort invested into ensuring its safety varies according to that classification. The names assigned to these classifications are not consistent across the standards and they include Safety Integrity Level (SIL), Automotive SIL (ASIL), and Class.

If you'd like to learn more about how the standards require functional safety measures that are proportionate to risk, read the blue section, p. 16-19.

Summary and conclusions

Capacitive HMI systems are becoming an integral part of safety-critical functionality across the sectors. Malfunction in such systems could have safety implications for operators, patients, drivers, passengers, or the general public.

"Functional safety minimizes risk in case of system malfunction when the wellbeing of people depends on correct operation"

Functional safety minimizes risk in case of system malfunction when the wellbeing of people depends on correct operation. It follows that HMI systems in the safety-critical sectors are under increasing scrutiny from the perspective of the functional safety standards.

Functional safety decomposition offers an opportunity to minimise the overhead demanded by these standards on HMI development. However, decomposition also demands communication between segregated domains, and care is required to ensure that these communications path cannot compromise the integrity of the most critical software items. Vulnerability to such compromise is significantly increased in connected systems.

Image 1: Functionality is being integrated into the touchscreen HMI

Functional safety and the HMI

Automotive systems present an obvious example of where capacitive HMI systems are becoming an integral part of safety-critical functionality such that a malfunction could have implications for driver, passengers, or other road users. Most cars released in recent years have seen the functionality traditionally associated with a plethora of physical buttons integrated into the touchscreen HMI, lending a minimalistic and clean look to the interior as a whole.

Photo by: ün LIU

As discussed in the blue section, p. 16-19, the functional safety standard IEO 26262 applies to the automotive sector. Automotive HMI interfaces are unlikely to be directly involved with the dynamic behaviour of the vehicle and so few will be ASIL D – the most demanding level. However, there are many examples of ASIL B or even ASIL C tasks integrated into capacitive HMI interfaces.

These include:

- Engine start/stop button
  - Failure mode: Unintended touch detected for starting the engine.
- ABS activation/cancellation
  - Failure mode: Unintended touch to deactivate ABS.
- Cruise control activation/cancellation
  - Failure mode: Unintended touch to activate cruise control.
- Driver drowsiness detection activation/cancellation
  - Failure mode: Unintended touch to deactivate driver drowsiness detection.

These examples are specific to the automotive sector, but the challenges are not limited to car development. The same migration of critical function interfaces to touch screens can be found across the sectors including rail, medical devices, and industrial plant. Nor are the challenges restricted to function activation - even monitoring devices can have safety implications if data is reported wrongly, and the operator acts upon that incorrect information.

Functional safety decomposition

Just as safety implications of a failed railway passenger lighting system has fewer safety implications than a failed braking system, it is also true that different parts of an individual system are likely to be less critical than others. Functional decomposition is a systematic process designed to ensure that no software component in a system is assigned a more demanding Class, ASIL, or DAL than necessary. The yellow section, p. 12-15, presents further information on the role of Classes, ASILs and DALs in ensuring that functional safety related overhead is proportionate.

IEC 62304, for example, defines three classes for medical devices as follows:

A

"Class A
The software system cannot contribute to a hazardous situation or the software system can contribute to a hazardous situation which does not result in unacceptable risk after consideration of risk control measures external to the software system.

B

Class B
The software system can contribute to a hazardous situation which results in unacceptable risk after consideration of risk control measures external to the software system but the resulting possible harm is non-serious injury.

C

Class C
The software system can contribute to a hazardous situation which results in unacceptable risk after consideration of risk control measures external to the software system, and the resulting possible harm is death or serious injury."

IEC 62304 permits the segregation of a software system into "software items", placing as little of the system as possible into the more critical classes. In doing so, it requires that *"The software ARCHITECTURE should promote segregation of software items that are required for safe operation and should describe the methods used to ensure effective segregation of those SOFTWARE ITEMS"*

Figure 1 shows the example of a Class C software system as illustrated in the standard.

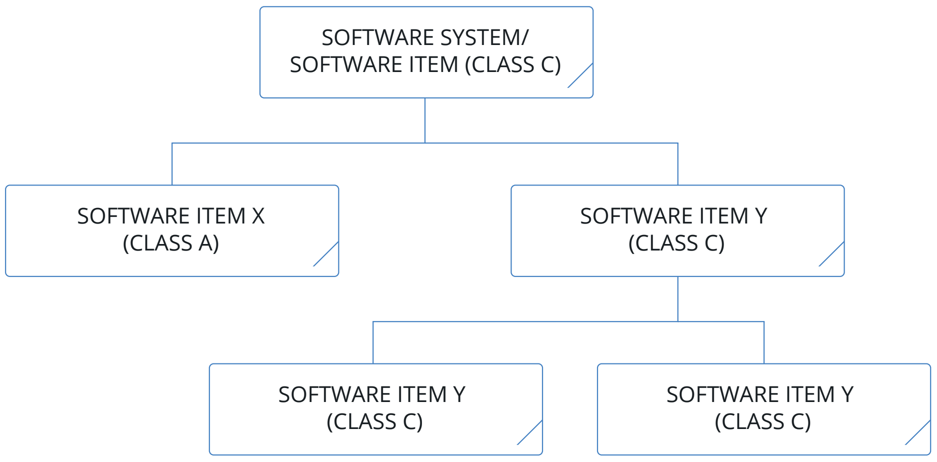

Figure 1: Example of decomposition of software items according to IEC 62304:2006 +AMD1:2015 Figure B.18

The system can be segregated into one software item to deal with functionality with fewer safety implications (software item X), and another to handle highly safety critical aspects of the system (software item Y).

That principle can be repeated in a hierarchical manner, such that software item Y can itself be segregated into software items W and Z, and so on – always on the basis that no segregated software item can negatively affect another. Software items such as X, W and Z that can be divided no further are defined as "software units".

Functional safety decomposition and the HMI

The ultimate goal of functional safety is to make a dangerous situation less severe, less probable, or more controllable. Naturally a user-friendly HMI has much to contribute to the achievement of that goal.

However, from the pragmatic perspective of HMI development, decomposition offers the opportunity to ensure that HMI associated code is segregated from any software responsible for proactive preventative and detective measures. In the medical device example shown in Figure 1, this would mean containing the HMI code within (a) software unit(s) of the lowest possible class. The effective management of system architecture and software design to this end helps to minimize the impact of functional safety overhead on HMI development.

There is, however, a sting in the tail of functional safety decomposition. None of these segregated software items exists in isolation. The more software systems are decomposed in this way, the more communication between components becomes necessary. That communication brings its own challenges.

Connectivity and functional safety

It would be easy to overlook the significance of the phrase *"effective segregation"* in the IEC 62304 description of functional safety decomposition. Effective segregation implies an assurance that issues associated with less critical software items (Class B, perhaps) cannot compromise those of higher criticality (Class C). If segregation is flawed, the additional assurance underpinning the Class C item is effectively compromised.

The security challenge complete with safety implications can be illustrated by reference to an imaginary production plant, with many demands on its systems infrastructure. The industrial pyramid in Figure 2 illustrates how hard real-time embedded applications and other Operational Technologies (OT) are integrated with IT in a typical production plant, bringing together technologies that were traditionally isolated with those that were traditionally connected.

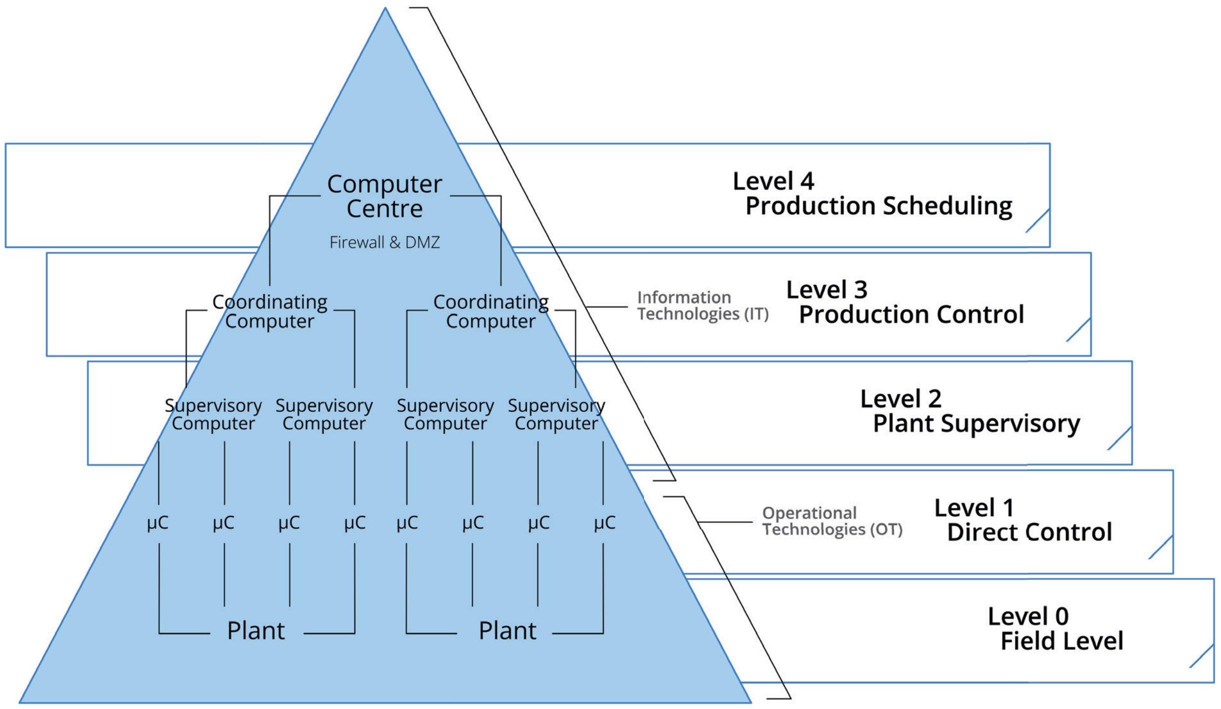

Figure 2: The industrial pyramid

Sensors provide operations owners the ability to collate data such as location, position, speed, temperature, pressure, lock status and vibration about all their assets, viewed via an HMI in near real-time.

At a higher security level still, process plant settings or even firmware can be updated remotely perhaps in response to this data, or to changing production demands. And production figures and profitability provide security headaches of a different, commercial kind; sensitive information divorced from the sensors' engineering data.

Effective domain segregation in this scenario is essential. But so is communication between those domains.

Integrating OT and IT in plants enhances data collection, remote updates, and efficiency. Ensuring secure, segregated communication between components of differing criticality is essential for safety.

Communication paths between segregated software items

There is usually a need for communication between software components that make up a system, including those of different criticality. Those communication paths deserve particular attention both from a functional safety and a security perspective.

"Communications paths from less critical components can represent attack vectors with the potential to compromise highly critical code"

The exposure of high-risk software components to data from components that have not been subjected to the same level of rigour has the potential to compromise safety. Similarly, communications paths from less critical components can represent attack vectors with the potential to compromise highly critical code.

Clearly the elements of the system considered most at risk by this, or any other measure will depend on the system itself. However, as a general rule, if domain separation or functional decomposition are to deliver their promise of safe and secure software, then it is essential that code handling any communication between domains or items of differing criticality must be scrutinised to at least the level of the most critical software component involved – and arguably to a higher standard than the remainder of the code base.

Functional safety standards originated with the DO-178 in civil aviation in the 1970s. DO-178 influenced later standards like IEC 61508, applied across various safety-critical sectors.

Functional safety standards

"DO-178 does have functional safety at its core and it was highly influential in the functional safety standards that followed it"

The roots of functional safety can be traced back to the late 1970s and the development of the DO178 guidance document (known colloquially as a standard and referenced as that here) for use in the civil aviation sector. Although it is generally known as a formal process standard rather than a functional safety standard, DO-178 does have functional safety at its core and it was highly influential in the functional safety standards that followed it.

DO-178 covers the complete software lifecycle – planning, development and integral processes – to ensure correctness and robustness in software systems for civil airborne applications. The integral processes include software verification, software quality assurance, configuration management assurance and certification liaison with the regulatory authorities. Increasingly, standards developed for commercial applications including the latest version of DO-178 (DO-178C [5]) have also become recognised as best practice in the defence sector.

IEC 61508 and its derivatives

The challenges addressed by DO-178C are largely common across the safety critical sectors, and the IEC 61508 standard series, first published in 1998, followed the example of DO-178. It allows for the development of a uniform technical approach that can be applied to all safety systems in electronics and related software. Its generic nature made it applicable across a wide range of sectors and it was soon established as the basis for many sector-specific derivatives. The ever-growing list of these so-called "child standards" includes the EN 5012X series [6] (railways), ISO 26262 [7] (automotive), IEC 62304 [8] (medical devices) and many more (Figure 3).

Naturally there are many similarities between the child standards, but there are variations too as the derivatives reflect not only the guidelines established in IEC 61508, but also the existing best practices in the sectors they address. For example, these variations reflect existing regulation (such as the FDA's regulatory classes [9] in IEC 62304) and the validity of a "proven in use" case in high-volume industries (as exemplified by ISO 26262).

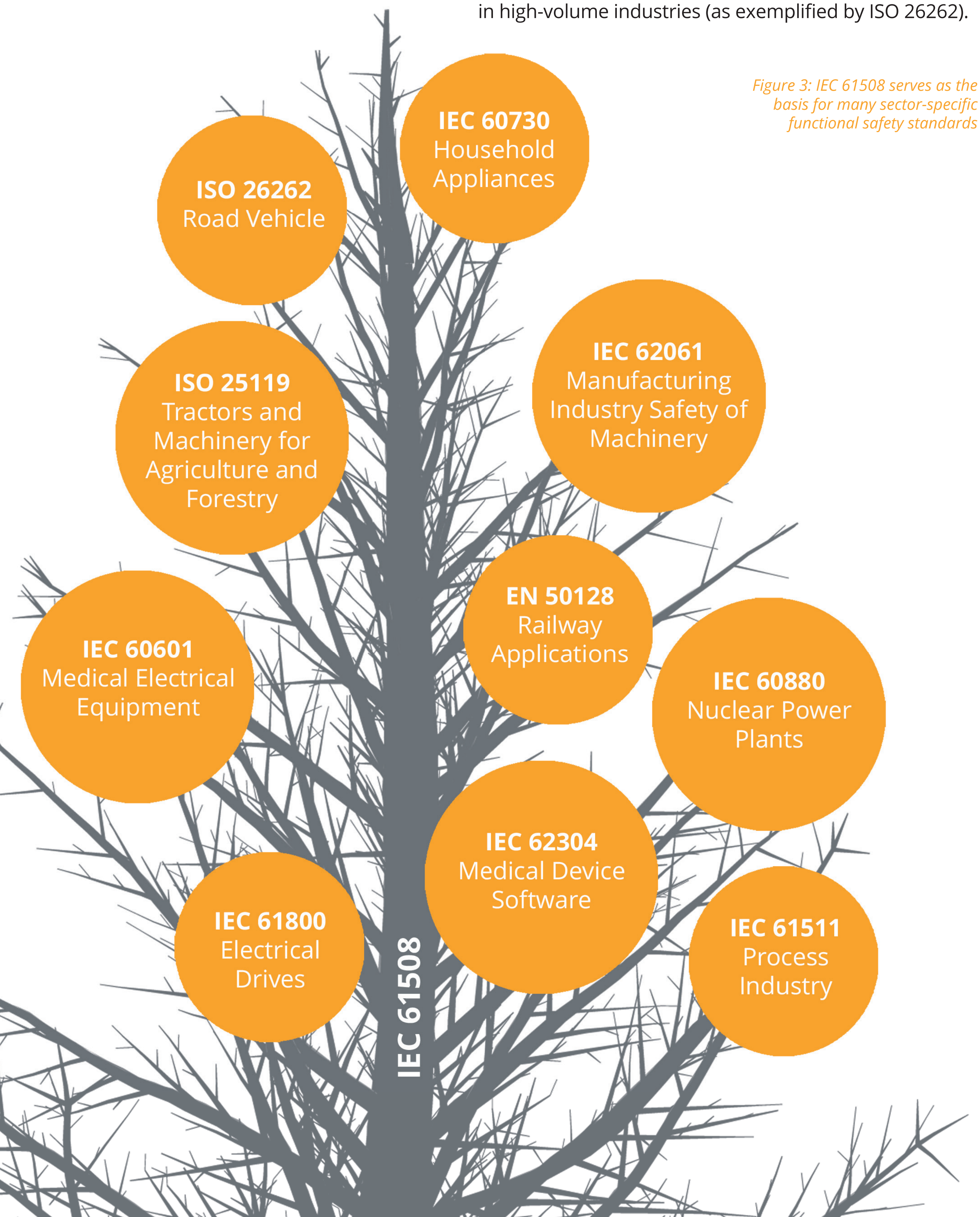
IEC 60730 Household Appliances
ISO 26262 Road Vehicle
IEC 62061 Manufacturing Industry Safety of Machinery
ISO 25119 Tractors and Machinery for Agriculture and Forestry
EN 50128 Railway Applications
IEC 60601 Medical Electrical Equipment
IEC 60880 Nuclear Power Plants
IEC 62304 Medical Device Software
IEC 61800 Electrical Drives
IEC 61511 Process Industry
IEC 61508

Figure 3: IEC 61508 serves as the basis for many sector-specific functional safety standards

Achieving functional safety

Functional safety standards require engineers to consider different failure categories. For example, ISO 26262 defines random hardware failures *"that occur unpredictably during the lifetime of a hardware element, and that follow a probability distribution"*, whereas IEC 61508 defines systemic failure as *"... related in a deterministic way to a certain cause, which can only be eliminated by a modification of the design or of the manufacturing process, operational procedures, documentation or other relevant factors."*

The susceptibility of software to systemic failure is obvious. Software is pure "thought-stuff" by nature and there is no possibility of an apparently random failure in the same way that

Image 2: HMI interfaces in civil aviation are subject to the demands of DO-178C

there might be for a mechanical component demonstrating the vagaries of a normally distributed life expectancy.

Preventative measures aim to minimize the probability of systematic failures during the development phase of a system. Any poorly specified, designed, or implemented aspect of a system may lead to failure. The higher the quality of the development, the less likely the failure.

Random hardware failures happen once the system is operating in the field. Typical causes include issues related to temperature, pressure, vibration, radiation, pollution, or ageing. The nature of such failures can be permanent, intermittent, or transient. **Detective measures** are designed into software to detect failing or failed systems, and initiate responses to limit damage and risk.

Photo by: ThisisEngineering

Policing compliance

DO-178C is the primary document by which the certification authorities such as the FAA [10] and EASA [11] approve all commercial software-based aerospace systems. These authorities require evidence of compliance for a product before they will confirm it to be airworthy.

However, not all functional safety standards are mandatory in the sectors to which they apply. For example, compliance with ISO 26262 is not obligatory although OEMs will often mandate suppliers’ compliance by contractual means.

Developers of medical devices are not obliged to comply with IEC 62304, but they are usually required to meet the approval of authorities in the geographical area in which their products will be used. For example, this makes the standard’s alignment with the US FDA classification significant.

In all cases – whether mandated or not – compliance with standards acknowledged to represent best practice can only be a good thing, especially if things go wrong and there is a need to justify development practices later.

Proportionate safety measures

How safe is safe enough? Ideally, the most thorough safety precautions would be applied by all software system developers, however unlikely potential harm might be. That is unlikely to be commercially justifiable. A definition of “safe enough” is required to allow precautions to be proportionate.

One approach is to use applied statistics in the form of Failure Modes and Effects Analysis (FMEA), a step-by-step approach for identifying all possible failures in a design, a manufacturing or assembly process, or a product or service. In this context, failure modes are any potential or actual errors or defects that can cause potential harm. Effects analysis consists of a systematic study of the consequences of the failure modes.

A refinement to that approach, FMECA, brings the criticality of that failure into the calculation. FMEA and FMECA are relevant to both functional safety and SOTIF.

All functional safety standards recognise that the extent to which software systems are safety-critical varies. IEC 61508 defines the notion of a “SIL”, or Safety Integrity Level. A SIL is a measure of safety system performance, in terms of probability of failure on demand (PFD). There are four discrete integrity levels associated with SIL: SIL 1, SIL 2, SIL 3, and SIL 4, with SIL 4 being the most demanding.

IEC 61508	Generic	SIL 0	SIL 1	SIL 2		SIL 3	SIL 4
ISO 26262	Automotive	QM	ASIL A	ASIL B	ASIL C	ASIL D	N/A
IEC 62304	Medical devices	Class A		Class B		Class C	
EN 5012X	Railways	SIL 0	SIL 1	SIL 2		SIL 3	SIL 4
DO-178C	Civil aerospace	DAL E	DAL D	DAL C		DAL B	DAL A

Figure 4: An approximate mapping of SILs, ASILs, DALs and Classes

This principle of differentiating between levels of criticality is common across the standards, although the naming conventions for these levels and the mechanisms for their derivation vary as illustrated in Figure 4.

Calculating appropriate classification levels considers problem likelihood and outcomes, with ISO 26262 defining ASILs based on severity, exposure, and controllability.

Calculating the appropriate level

Approaches to the calculation of the appropriate classification varies between sectors and their standards, but in general each takes into account the likelihood of a problem occurring and the likely outcome should it do so.

Consider the automotive standard ISO 26262 as an example. It defines ASILs ranging from QM, a quality management issue with nominal safety critical impact and no additional requirements, up to D, which represents likely potential for severely life-threatening or fatal injury in the event of a malfunction. ASIL D requires the highest level of assurance that the dependent safety goals are sufficient and have been achieved. These levels are assigned based on a composite score of three factors:

Severity: a measure of the gravity, or seriousness, of potential injury in the case of failure. Severity ranges from S1 (light and moderate injuries) up to S3 (life-threatening and fatal injuries).

Exposure: a measure of the relative expected frequency of exposure for each operational situation where a specific hazard may occur. Exposure ratings range from E1 (very low probability of exposure) to E4 (high probability of exposure).

Controllability: a measure of how easy or difficult it would be for the driver, or other persons involved, to control the situation, and ranges from C1 (simply controllable by 99% of drivers) to C3 (uncontrollable or difficult to control).

Figure 5 shows how these ratings are combined to derive an appropriate ASIL. 'QM' implies that no additional measures beyond normal corporate quality measures are required.

Severity class	Probability class	Controllability class		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Figure 5: The ASIL derivation matrix

Why does it matter?

Whatever the classifications – SILs, ASILs, Classes, or DALs – the criticality levels assigned are important from a commercial perspective because there is significant overhead involved in the achievement of the more demanding levels, compared to the less critical ones.

For example, the railway specific EN 50129 [10] calls its classifications SILs. The specified development process checks and safety measures avoid an unreasonable residual risk proportionate to the assigned SIL. SILs range from Basic Integrity (SIL 0) and then from 1 through to 4, where SIL 4 represents the most hazardous and hence demanding level. The overhead involved in producing a safety critical SIL 4 system (e.g., braking) is significantly greater than that required to produce a system with fewer safety implications (e.g., interior passenger lighting) as illustrated by EN 50128 Table A.5 (Figure 6).

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Formal Proof	D.29	-	R	R	HR	HR
2. Static Analysis	Table A.19	-	HR	HR	HR	HR
3. Dynamic Analysis and Testing	Table A.13	-	HR	HR	HR	HR
4. Metrics	D.37	-	R	R	R	R
5. Traceability	D.58	R	HR	HR	M	M
6. Software Error Effect Analysis	D.25	-	R	R	HR	HR
7. Test Coverage for code	Table A.21	-	HR	HR	HR	HR
8. Functional/ Black-box Testing	Table A.14	HR	HR	HR	M	M
9. Performance Testing	Table A.18	-	HR	HR	HR	HR
10. Interface Testing	D.34	HR	HR	HR	HR	HR

Figure 6: EN 50128 Table A.5 illustrates increasing V&V overhead in proportion to criticality
R: Recommended
HR: Highly Recommended
M: Mandatory

Although the underlying software development process is the same irrespective of criticality, simply comparing the requirements for SIL 4 compared to those for SIL 0 highlights the point.

Throughout the development lifecycle across the safety critical sectors, these classifications therefore have a tremendous impact on the code development process from planning, developing, testing, and verification through to release and beyond. ■

Works Cited

[1] International Electrotechnical Commission (IEC), "What we do," 2022. [Online]. Available: https://www.iec.ch/what-we-do. [Accessed 11 November 2022].
[2] International Electrotechnical Commission (IEC), "Safety and functional safety," 2022. [Online]. Available: https://www.iec.ch/functional-safety. [Accessed 18th November 2022].
[3] International Electrotechnical Commission (IEC), IEC 61508:2010 "Functional safety of electrical/electronic/programmable electronic safety-related systems" (all parts), IEC, 2010.
[4] RTCC, DO-178C "Software Considerations in Airbourne Systems and Equipment Certification", RTCA, 2011.
[5] RTCC, DO-178C "Software Considerations in Airborne Systems and Equipment Certification", RTCA, 2011.
[6] European Committee for Electrotechnical Standardization (CENELEC), EN 50128 "Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems", CENELEC, 2020.
[7] International Organization for Standardization, ISO 26262:2018 "Road vehicles - Functional safety", International Organization for Standardization, 2018.
[8] International Electrotechnical Commission, IEC 62304:2006+AMD1 Medical device software - Software life cycle processes, IEC, 2015.
[9] U S Food & Drug Administration, "Classify your medical device," 2 July 2020. [Online]. Available: https://www.fda.gov/medical-devices/overview-device-regulation/classify-your-medical-device. [Accessed 30 December 2022].
[10] European Committee for Electrotechnical Standardization, EN 50129 "Railway applications. Communication, signalling and processing systems. Safety related electronic systems for signalling", CENELEC, 2017.
[11] RTCA, DO-178A "Software Considerations in Airborne Systems and Equipment Certification", RTCA, 1985.
[12] Federal Aviation Administation (FAA), "Federal Aviation Administation (FAA)," [Online]. Available: https://www.faa.gov/. [Accessed 3 November 2021].
[13] European Union Aviation Safety Agency (EASA), "European Union Aviation Safety Agency (EASA)," 2021. [Online]. Available: https://www.easa.europa.eu/. [Accessed 3 November 2021].

Images

[1] ün LIU, üL. (2022). "A couple of people in a car" [photograph]. Unsplash. https://unsplash.com/photos/a-couple-of-people-in-a-car-jbilhRsQ_-0
[2] ThisisEngineering. (2020). "Pilot flying flight simulator" [photograph]. Unsplash. https://unsplash.com/photos/person-holding-black-and-red-hand-tool-_PdoBI2XhkY

About the KDAB Group

The KDAB Group is the world's leading software consultancy for architecture, development and design of Qt, C++ and OpenGL applications across desktop, embedded and mobile platforms. KDAB is the biggest independent contributor to Qt and is the world's first ISO 9001 certified Qt consulting and development company. Our experts build runtimes, mix native and web technologies, solve hardware stack performance issues and porting problems for hundreds of customers, many among the Fortune 500. KDAB's tools and extensive experience in creating, debugging, profiling and porting complex applications help developers worldwide to deliver successful projects. KDAB's trainers, all full-time developers, provide market leading, hands-on, training for Qt, OpenGL and modern C++ in multiple languages.

www.kdab.com