LDRA
Software Technology

Technical White Paper

Applying SAE J3061 to ISO 26262 Processes with the LDRA tool suite®

Cost Effective Software Certification from ASIL A to ASIL D

www.ldra.com
* Registration required to download the document

Contents

Background

Over the past few years, there has been a proliferation of automotive electrical and/or electronic (E/E/PE) systems such as adaptive driver assistance systems, anti-lock braking systems, steering and airbags. ISO 26262 "Road vehicles – Functional safety" was first published in 2012 in response this explosion in automotive E/E/PE system complexity and the associated risks to public safety[1], bringing with it an opportunity for automotive manufacturers to embrace best functional safety practices throughout the development lifecycle.

More recently, the increasing levels of integration and connectivity associated with such systems have provided almost as many challenges as their proliferation, with non-critical systems such as entertainment systems sharing the same communications infrastructure as steering, braking and control systems. The potential for hazards and economic losses resulting from cyberattacks has consequently given increasing cause for concern over recent years. ISO 26262 requires any threats to functional safety to be adequately addressed, implicitly including those relating to security threats, but it gives no explicit guidance relating to cybersecurity.

At the time of ISO 26262's publication, that was perhaps to be expected. Automotive embedded applications have traditionally been isolated, static, fixed-function, device-specific implementations, and practices and processes have relied on that status. But the rate of change in the industry has been such that by the time of its publication in 2016, SAE International's Surface Vehicle Recommended Practice SAE J3061™ in January 2016 was much anticipated[2].

SAE J3061 can be considered complementary to ISO 26262 in that it provides guidance on best development practices from a cybersecurity perspective, just as ISO 26262 provides guidance on practices to address functional safety.

SAE J3061 defines a Cyber-Physical Vehicle System (CPVS) as being *"composed of tightly-integrated computation, communication and physical elements"*, and in that context it seeks to define *"... a complete lifecycle process framework that can be tailored and utilized within each organization's development processes to incorporate cybersecurity into Cyber-Physical Vehicle Systems from concept phase through [to] production, operation, service, and decommissioning."*

The definition of that framework includes the provision of such as basic guiding principles, information relating to existing tools and methods, and the foundation for further standards development activities.

Beyond Functional Safety

Despite the clear synergy between the two standards, it is important to note that SAE J3061 does more than simply formalize the need to include security considerations in functional safety requirements. Much of this white paper focuses on an appropriate process once functional, safety and security requirements are established but the significance of malicious intent in the definition of those requirements should not be underestimated.

SAE J3061 emphasizes this point throughout. It argues that cybersecurity is likely to be even more challenging than functional safety, stating that "*Since potential threats involve intentional, malicious, and planned actions, they are more difficult to address than potential hazards. Addressing potential threats fully, requires the analysts to think like the attackers, but it can be difficult to anticipate the exact moves an attacker may make*."

Perhaps less obviously, the introduction of cybersecurity into an ISO 26262-like formal development implies the use of similarly rigorous techniques into applications that are NOT safety critical – and perhaps into organizations with no previous obligation to apply them. SAE J3061 discusses privacy in general and Personally Identifiable Information (PII) in particular, and highlights both as key targets for a bad actor of no less significance than the potential compromise of safety systems. In practical terms, it therefore demands that ISO 26262-like rigour is required in the defence of a whole manner of personal details

[1] https://www.iso.org/news/2012/01/Ref1499.html

[2] SAE International Surface Vehicle Recommended Practice – J3061™ -Cybersecurity Guidebook for Cyber-Physical Vehicle Systems. Issued 2016-01.

potentially accessed via a connect car, including personal contact details, credit card and other financial information, and browse histories. It could be argued that this is an extreme example of the general case cited by the SAE J3061 standard, which states that "*...there is no direct correspondence between an ASIL rating and the potential risk associated with a safety-related threat*."

Not only does SAE J3061 bring formal development to less safety-critical domains, it also extends the scope of that development far beyond the traditional project development lifecycle. The need to establish an incident response process to address vulnerabilities that become apparent when the product is in the field, consideration for over-the-air (OTA) updates, and cybersecurity considerations when a vehicle changes ownership are all examples of that.

Application of the Guidelines

SAE J3061 calls for a similar sound development process to that of ISO 26262, and so the standards are not contradictory. It specifies that an initial Threat Analysis and Risk Assessment (TARA) is completed to assess potential threats related to operation, privacy, finances, and reputation, and to quantify the risk associated with each of those threats. If the risk for any particular threat is sufficiently high, then a cybersecurity process is deemed necessary.

There is some flexibility in this initial risk assessment in that it may be based on experience or expert judgment rather than on a formal assessment process. Examples of issues that might be considered in determining the risk include an estimate of the magnitude of the impact, from a financial, safety, privacy, or operational perspective, and whether an attack may involve a fleet of vehicles or a single vehicle.

System Safety and System Cybersecurity

A system can be said to be in a functionally safe state when it causes no harm to life, property, or the environment. Similarly, a system can be said to be in a state that is secure against cyberattack when it does not allow exploitation of vulnerabilities to lead to losses, such as financial, operational, privacy, or safety.

All safety critical systems are also cybersecurity critical because a direct or indirect cyberattack on a safety critical system could lead to potential losses of safety. But not all cybersecurity critical systems are safety critical because cyberattacks on cybersecurity critical systems can result in other losses such as privacy, operational, or financial.

Hazard analyses are performed to assess risks associated with safety, whereas threat analyses identify risks associated with security. The parallels between the disciplines do not extend to the hazards and threats themselves, in that hazards can usually be accurately identified by an expert team, but threats are mostly unknown and can only be predicted by such expertise supported by statistical analysis.

Hazard analysis and threat analysis techniques are again analogous, but the methods and goals used in each are unique. For example, detailed hazard analysis may involve Fault Tree Analysis[3] (FTA) whereas threat analysis may apply Attack Tree Analysis[4] (ATA).

These additional considerations for security requirements for a system can be incorporated in the process described by ISO 26262, part 8, section 6: "Specification and management of safety requirements."[5]

Another example of differing goals from analogous methods can be found in the use of static analysis, which is used in safety critical system development to identify construct, errors, and faults that could directly affect primary functionality. In cybersecurity critical system development, static code analysis is used instead to identify potential vulnerabilities in the code. Code that has been verified correct from the perspective of primary functionality may retain cybersecurity vulnerabilities.

[3] Using Fault Tree Analysis in Developing Reliable Software, E.O.Ovstedal, IFAC Proceedings Volume 24, Issue 13, October–November 1991, Pages 77-82

[4] Saini, Vineet & Duan, Qiang & Paruchuri, Vamsi. (2008). Threat Modeling Using Attack Trees. Journal of Computing Sciences in Colleges. 23.

[5] ISO 26262 - 8: 2011 (E) Road Vehicles - Functional Safety - Part 8: supporting processes

An Overview of the Cybersecurity Processes

The cybersecurity lifecycle begins with the development of the Cybersecurity Program Plan to describe the activities to be carried out. Threat analysis and risk assessments (TARA) are used to identify and assess the potential threats to the system and to determine the risk associated with each identified threat. The TARA results then drive subsequent activities by focusing analyses on the highest risk cybersecurity threats.

The product development phase of the lifecycle is subdivided into system, hardware and software levels.

Figure 1 outlines the principles of an ISO 26262 compliant cybersecurity process. It shows the relationship between the concept and product development phases, and the system, hardware and software product development phases.

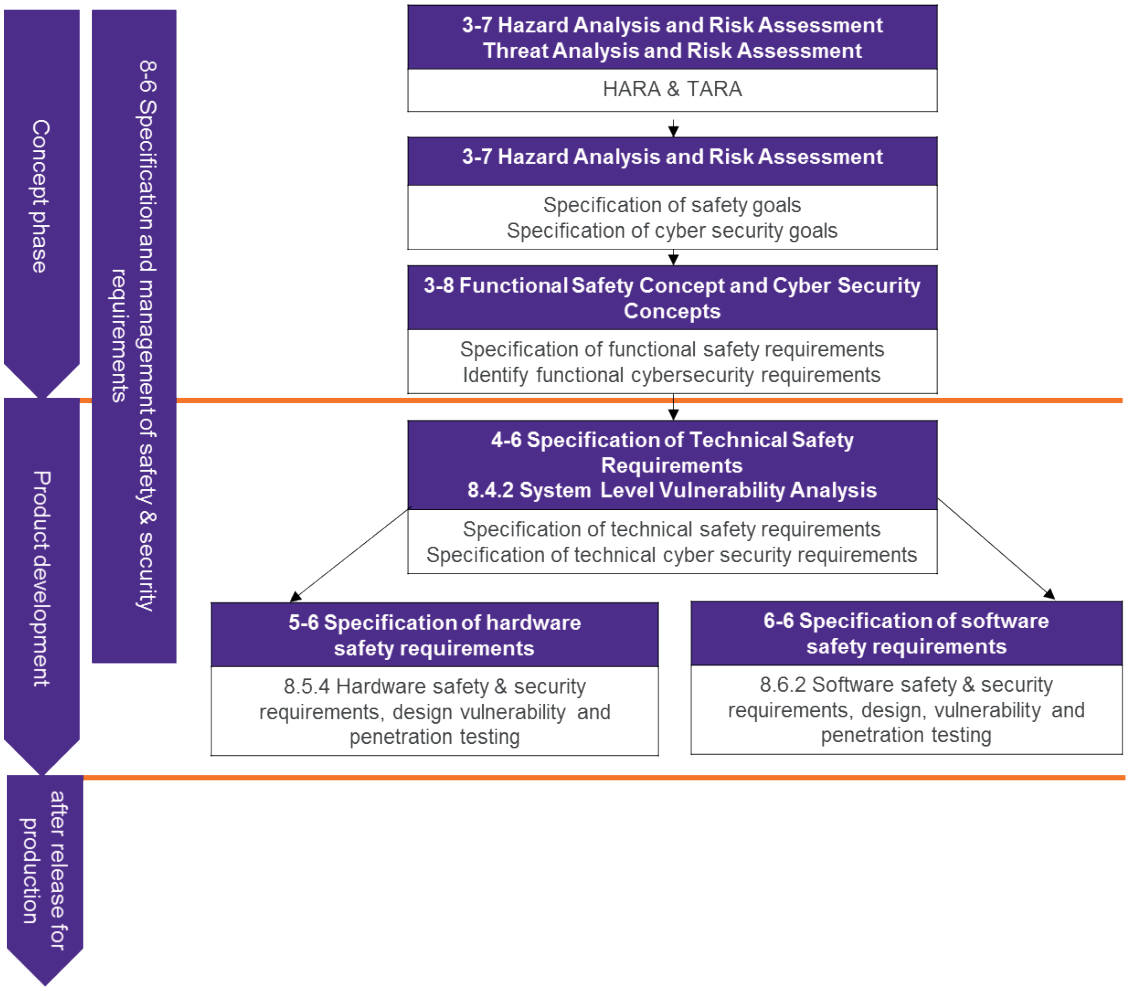

Figure 1: An ISO 26262 compliant cybersecurity process, showing the relationship between the concept and product development phases, and the system, hardware and software product development phases.

Applying SAE J3061 Cybersecurity Processes in Accordance with ISO 26262

According to SAE J3061, *"Since the Cybersecurity process described in this recommended practice is based on the ISO 26262 process framework, tightly integrating the two processes would simply mean to include the Cybersecurity activities described in this document for each product lifecycle phase, with the corresponding activities for each product lifecycle phase described in the safety process."*

It would, then, be possible to develop a safe and secure application by keeping the cybersecurity related activities described in SAE J3061 separate from the functional safety related activities described in ISO 26262. However, the standards lend themselves to the integration of the two at each stage of the product lifecycle – even to the extent that the same test team could be deployed to fulfil both roles.

For example, it is possible to develop a technique to perform a hazard analysis, safety risk assessment, threat analysis, and security risk assessment concurrently using a single integrated template and method.

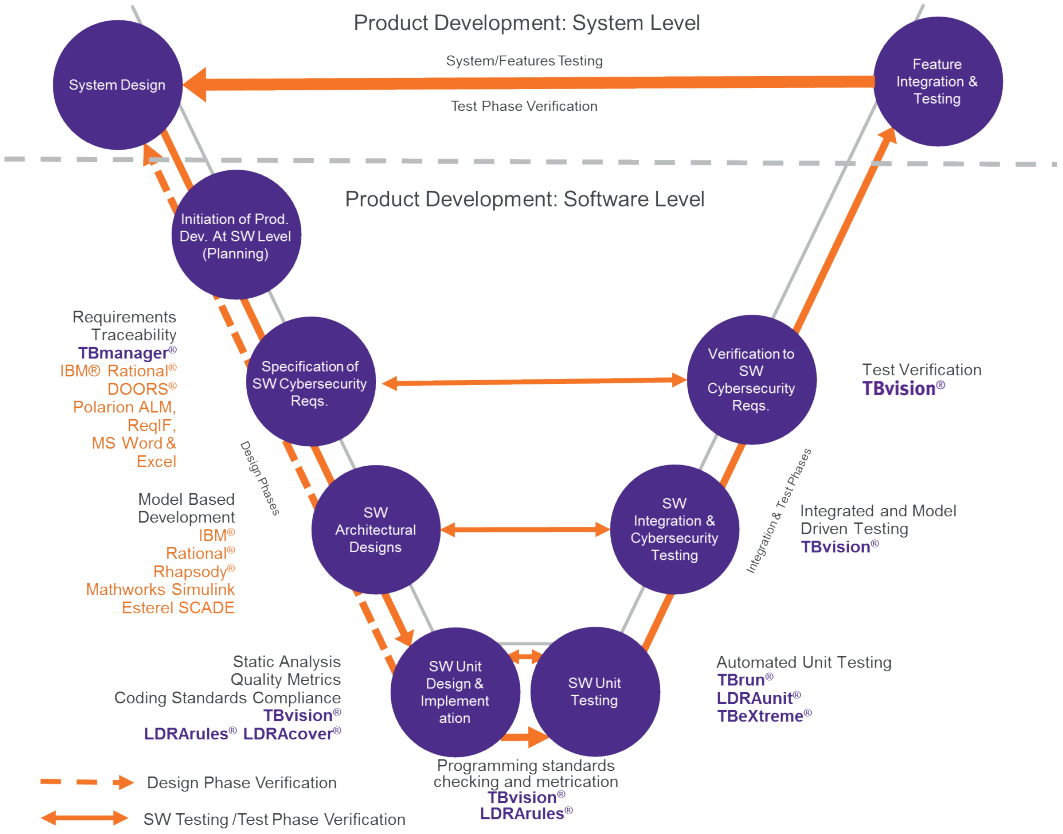

Figure 2: Mapping the capabilities of an automated tool chain to SAE J3061 process guidelines

Software tools vendors such as LDRA have long experience in easing the path to certification both in the automotive sector and elsewhere, and that expertise can be leveraged in the development of SAE J3061 and ISO 26262 compliant applications. Each element of the V diagram is expanded in the following sections, and relevant tables from ISO 26262:2011 are supplemented by reference to pertinent details from SAE J3061. Note that each of the tables has a clear association with a particular element of the V-model shown in Figure 2, but may reference techniques which cannot be applied until later in the development lifecycle.

For example, *"Table 6: Methods for the verification of the software architectural design"* is primarily focused on software architectural design, but the techniques of *"Control flow analysis"* and *"Data flow analysis"* can only be performed later in the lifecycle when the design has been interpreted and implemented in the form of source code.

SAE J3061 Section 8.6.2: "*Specification of Software Cybersecurity Requirements*"

Functional safety and security requirements are refined and allocated to hardware and software during the system design phase described in ISO 26262-4:2011, Section 7.

The primary objective of this phase from the perspective of SAE J3061 is to review and update the cybersecurity requirements previously identified in the context of the system as a whole. This activity will reference such as hardware and software interfaces, data flow, data storage, data processing, and any subsystems supporting cybersecurity functionality.

According to SAE J3061, *"The next step involves understanding how the software supports the overall system purpose or mission including the Cybersecurity functions it should perform such as preventing unauthorized access or detecting tampering. Required Cybersecurity functions may be defined in terms of parameters such as performance, effectiveness, or timeliness."* The standard gives an example of how a software module might respond to the detection of tampering, with due consideration as to what the desired outcome would be for the attacker.

This phase is essentially analogous to the interface between the product-wide system design of ISO 26262-4:2011, and the software specific ISO 26262-6:2011. It details the process of evolution of lower level requirements for safety and security as they relate specifically to the software system. That implies a continued leveraging of the requirements management tools selected for the project.

Figure 3 references table 18 from the SAE J3061 standard as a basis to illustrate how security goals would be specific to particular software modules (or *"Assets"*), and how they might be cross referenced to appropriate safety goals and requirements.

No	Asset	Threat	Security Attribute	Security Level	Security Requirement
1	Cryptographic Key	Elevation of Privilage	Authorization	Critical	
2	ECU Software	Tampering	Integrity	Medium	
3	CAN Signal X on Bus A	Spoofing	Authenticity	QM	

Safety Goal ID	Safety Concept ID	Functional Safety Requirements (FSR)	Technical Safety Requirement	Safety Requirement (Hardware)	Safety Requirement (Software)	Security Requirement	ASIL
SG_001	SC_001	The ECU shall be monitored by a watchdog		›The watchdog module shall be programmed from WDOGLOAD Register ›The watchdog module shall have a 32-bit down counter ›The counter shall decrement by one on each positive clock edge of WDOGCLK when WDOGCLKEN (the clock enables) is HIGH ›The watchdog module shall assert reset WDOGRES, when the counter reaches 0 and If RESEN bit in WDOGCONTROL register is set to 1 ›The watchdog module shall be reloaded from the WDOGLOAD Register on the next enabled WDOGCLK clock edge ›Watchdog module shall assert WDOGINT If INTEN bit in WDOGCONTROL register is set to 1	The watchdog module shall apply a reset to a system in the event of a software failure	The control pins (Clock pins, enable pins) of watchdog module shall not exposed to external interface level	B

Figure 3: Cross referencing security goals and safety goals to software modules

SAE J3061 Section 8.6.3: "*Software Architectural Design*"

The objective of this phase is to develop a software architectural design that realizes the security requirements, leveraging the analysis of the *"data types being used, how the data will flow, how the software will detect errors; and how the software will recover from errors"*. The aim is to have the data maintain confidentiality (non-disclosure of information), integrity (unauthorized modification or destruction of data), and availability (timely and reliable access) in accordance with the definition of those terms in FIPS Pub 199[6]. These properties are considered to be those of an ideal secure system.

SAE J3061 also recommends verifying the design for safety and security with reference to Table 6 in ISO 26262 :2011. For example, this recommendation might be implemented by incorporating a threat model into the design, and then verifying the anticipated behaviour of the software design in response to that threat during the design verification stage.

Figure 4 is a modified version of Table 6 reproduced from ISO 26262-6:2011, showing how the software architectural design is to be verified both at design time and as the design is implemented as development progresses.

[6] FIPS Pub 199. "Standards for Security Categorization of Federal Information and Information Systems", February 2004

Topics		ASIL			
		A	B	C	D
1a	Informal verification by walkthrough of the design	++	+	o	o
1b	Informal verification by inspection of the design	+	++	++	++
1c	Semi-formal verification by simulating dynamic parts of the design	+	+	+	+
1d	Semi-formal verification by prototype generation/animation	o	o	+	+
1e	Formal verification	o	o	+	+
1f	Control flow analysis	+✓	+✓	++✓	++✓
1g	Data flow analysis	+✓	+✓	++✓	++✓

”++” The method is highly recommended for this ASIL.
“+“ The method is recommended for this ASIL.
“o“ The method has no recommendation for or against its usage for this ASIL.
✓ Satisfied by the LDRA tool suite

Figure 4: Mapping the capabilities of the LDRA tool suite to ISO 26262-6:2011 “Table 6: Methods for Verification of Architectural design”[7]

Static analysis tools have a part to play in the verification of the design in the form of control and data flow analysis of the code generated in accordance with it. The tools derive the relationship between some or all the code components and represent it graphically such that it can then be compared with the intended design and the cybersecurity requirements as specified in SAE J3061 section 8.6.2.

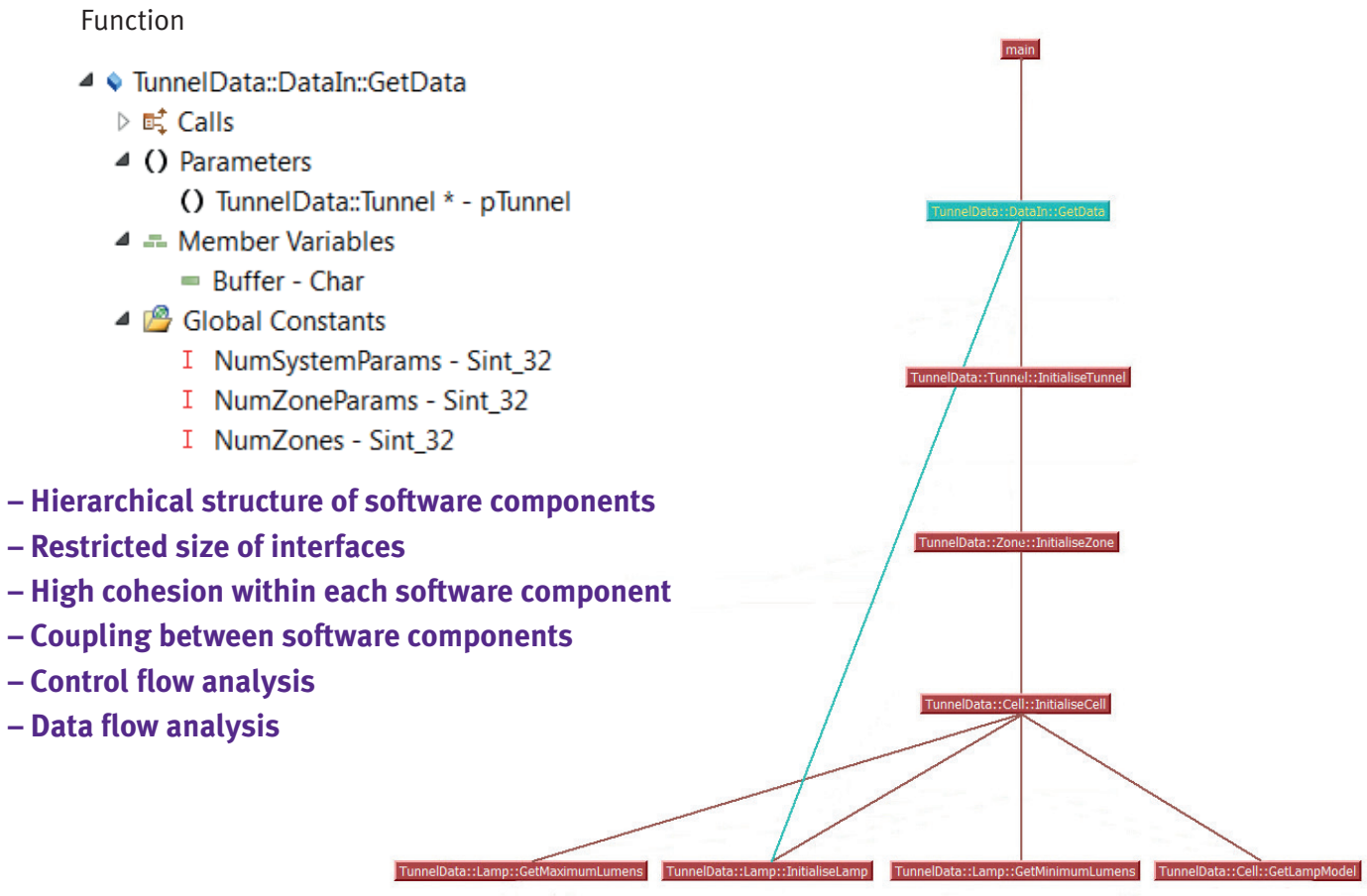

Figure 5: Diagrammatic representations of control and data flow generated from source code by the LDRA tool suite aid verification of software architectural design

[7] Based on table 6 from ISO 26262-6:2011, Copyright © 2015 IEC, Geneva, Switzerland. All rights acknowledged

Reverse Engineering

Much of the flow of the ISO 26262:2011 processes is based on an overriding presumption that a project either begins with nothing, or is *"proven in use"*[8] which implies the deployment of something that has been shown to be reliable after many runtime hours. However, it is entirely possible that there is a need to adapt a long-established application which would void that "proven in use" argument, as would the integration of third-party source code into an existing design. What if source code exists, but design documentation doesn't?

In such circumstances, a graphical representation of the code (Figure 3) can help to establish the architecture of the existing system, such that the additions to it can be designed and proven in accordance with ISO 26262 principles.

Model Based Development

Test tools such as the LDRA tool suite can be integrated with several different model based development tools, such as IBM Rational Rhapsody[9], MathWorks Simulink[10] and Esterel SCADE[11]. The development phase itself involves the creation of the model in the usual way, with the integration becoming more pertinent once source code has been auto generated from that model. The integration is primarily leveraged during Software Unit Testing, and Software Integration and Testing.

SAE J3061 Section 8.6.4: "*Software Vulnerability Analysis*"

This is a security specific activity, and so it is supplementary to the processes specified by ISO 26262: 2011. SAE J3061 refers to *"Trust Boundaries"*, which can be thought of lines drawn through a program. On one side of the line, data is untrusted. On the other side of the line, data is assumed to be trustworthy. The objective of Software Vulnerability Analysis (SVA) is to perform an analysis of the software cybersecurity requirements and the data flow from the software architectural design, to define where these trust boundaries exist.

"Threat Modelling" is a procedure for optimizing cybersecurity. It accomplishes its goal by focusing on the flow of data and control between components and through entry and exit points in the overall system for given functional use cases of the system. Any place where data and control passes any trust boundaries is given particular focus.

The first step in a software vulnerability analysis is to decompose the application, and to analyse the data and control entry and exit points. Appropriate controls are defined wherever data crosses the trust boundaries, and documented in the threat model in the form of specialized data flow diagrams which show different paths through the overall system, highlighting privilege boundaries.

This analysis and its associated documentation helps the design team determine where to place particular emphasis during cybersecurity validation – perhaps in the form of one the CERT top 10 secure coding practices[12], such as the sanitization of data send to peripherals. Later, during the implementation phase, the application of secure coding standards is an example of best practice is achieving that sanitization.

The second software vulnerability analysis step involves using threat categorization such as STRIDE[13], Application Security Frame (ASF) and/or DREAD[14] to identify threats based on the break-down of the system. In some models, severity values for process, control, and data areas are determined while in others, any entry point or trust boundary is treated as critical for the final step.

STRIDE and **DREAD** are part of a risk assessment system originally developed by Microsoft, but since used more widely.

They provide mnemonics for security threats:

Spoofing identity;
Tampering with Data;
Repudiation;
Information Disclosure;
Denial of Service;
Elevation of Privilege

And for risk ratings:

Damage;
Reproducibility;
Exploitability;
Affected users;
Discoverability

[8] ISO 26262-8:2011 Road vehicles -- Functional safety -- Part 8: Supporting processes
[9] http://www-03.ibm.com/software/products/en/ratirhapfami/
[10] https://www.mathworks.com/products/simulink.html
[11] http://www.esterel-technologies.com/products/scade-suite/
[12] https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices
[13] https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx
[14] https://msdn.microsoft.com/en-us/library/ff648644.aspx

The Common Vulnerability Scoring System (CVSS[15]) and Common Weakness Scoring System (CWSS[16]) are associated with CVE[17] and CWE[18] respectively, and can each help to categorize vulnerabilities and weaknesses in software whether at module, application or source code level. Such systems can help to ensure that proportionate security measures are applied.

Common Attack Pattern Enumeration and Classification (CAPEC™[19]) provides specific attack patterns associated with specific CWE weakness types. CAPEC is a comprehensive dictionary and classification taxonomy of known attacks that can be used by analysts, developers, testers, and educators to advance community understanding and enhance defences.

SAE J3061 Section 8.6.5: "*Software Unit Design and Implementation*"

ISO 26262-6:2011 Section 8 states that;

"The first objective of this sub-phase is to specify the software units in accordance with the software architectural design and the associated software safety requirements."

"The second objective of this sub-phase is to implement the software units as specified."

"The third objective of this sub-phase is the static verification of the design of the software units and their implementation."

All of these objectives remain valid when applying SAE J3061 in tandem with ISO 26262, with the following addition:

"The first objective of this sub-phase is to specify the software units in accordance with the software architectural design and the associated software safety [and security] *requirements."*

Coding Guidelines

There are several aspects of software unit design and implementation to be considered, starting with a definition of guidelines associated with how the selected programming language is to be used. Figure 6 is a modified version of Table 1 reproduced from ISO 26262-6:2011. Table 1 in the standard shows the coding and modelling guidelines to be enforced during implementation, and it is superimposed here with an illustration of where compliance can be confirmed by an automated tool suite.

Viewed from a cybersecurity perspective, coding guidelines are required to incorporate good coding practices (such as the CERT Top 10 secure practices) which are likely to specify the application of secure coding standards such as MISRA, CERT or CWE, to promote such as

- input validation,
- the sanitization of data,
- secure string usage,
- the avoidance of deprecated and/or unsafe API functions, and
- the avoidance of strings or arrays of unspecified length, in view of potential buffer overflows

Each rule and recommendation in the CERT secure coding standard has an assigned priority, calculated using a metric based on Failure Mode, Effects, and Criticality Analysis[20] (FMECA) and hence similar in nature to a Risk Priority Number (RPN). Three values are assigned for each rule on a scale of 1 to 3 for Severity (how serious are the consequences of the rule being ignored), Likelihood (how likely is it that a flaw introduced by ignoring the rule can lead to an exploitable vulnerability), and Remediation Cost (how expensive is it to comply with the rule). These three values are then multiplied together for each rule, and their product provides a measure that can be used in prioritizing the application of those rules.

[15] https://www.first.org/cvss/
[16] https://cwe.mitre.org/cwss/cwss_v1.0.1.html
[17] https://cve.mitre.org/
[18] https://cwe.mitre.org/index.html
[19] https://capec.mitre.org/
[20] IEC 60812:2006 Analysis techniques for system reliability – Procedure for failure mode and effects analysis

The product of the System Level Vulnerability Analysis (SLVA) specified in SAE J3061 Section 8.4.7 is the vulnerability test plan. This plan is instrumental in confirming that the implemented requirements provide effective mitigations for the identified vulnerabilities that were identified. Vulnerability testing should include:

- Vulnerability scanning methods used to detect vulnerabilities that could be exploited,
- Exploratory testing methods used to detect and probe vulnerabilities that can be present in an implementation, and
- Aggressive testing to attempt to break, bypass, or tamper with the cybersecurity controls so as to demonstrate the ability to misuse the system or feature.

Central to the vulnerability test plan from a software perspective is a definition of how the selected programming language will be deployed to minimize potential vulnerabilities, and this is analogous to the measures described to minimize functional safety risks in ISO 26262-6:2011. Figure 6 is a modified version of Table 1 reproduced from ISO 26262-6:2011. Table 1 in the standard shows the coding and modelling guidelines to be enforced during implementation, and it is superimposed here with an illustration of where compliance can be confirmed by an automated tool suite.

Topics		ASIL			
		A	B	C	D
1a	Enforcement of low complexity	++✓	++✓	++✓	++✓
1b	Use of language subsets	++✓	++✓	++✓	++✓
1c	Enforcement of strong typing	++✓	++✓	++✓	++✓
1d	Use of defensive implementation techniques	o	+✓	++ ✓	++✓
1e	Use of established design principles	+✓	+✓	+✓	++✓
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+✓	++✓	++✓	++✓
1h	Use of naming conventions	++✓	++✓	++✓	++✓

”++” The method is highly recommended for this ASIL.
“+“ The method is recommended for this ASIL.
“o“ The method has no recommendation for or against its usage for this ASIL.
✓ Satisfied by the LDRA tool suite

Figure 6: Mapping the capabilities of the LDRA tool suite to “Table 1: Topics to be covered by modelling and coding guidelines” specified by ISO 26262-6:2011[21]

These guidelines can be justified on the basis that they make the resulting code more secure, reliable, less prone to error, easier to test, and/or easier to maintain. For example:

- Defensive implementation techniques (1d, Figure 6) reduce the likelihood of tainted data
- Uncomplex code is easier to read and maintain, and hence less susceptible to error. For low complexity to enforced, it needs to be quantified and “pass/fail” criteria established.
- Language subsets (1b, Figure 6) such as MISRA C or CERT C restrict the use of a coding language to those elements known to be least susceptible to causing problems.
- The use of style guides (1g, Figure 6) ensures that the code has a consistent appearance no matter who has written it. That makes it much easier to maintain or modify, which in turn makes it less prone to error.

It is entirely possible for language subsets to encompass both functional safety and cybersecurity. MISRA C:2012 in combination with Amendment 1[22] is designed to do just that. Although many subsets such as CERT C are more focused on one or the other, test tools can be configured to combined rules from more than one source.

[21] Based on table 1 from ISO 26262-6:2011, Copyright © 2015 IEC, Geneva, Switzerland. All rights acknowledged
[22] MISRA C:2012 Amendment 1 – Additional security guidelines for MISRA C:2012. April 2016.

If the coding guidelines of ISO 26262-6:2011 are analogous to the "bricks" of the software application, then its principles of software architectural design define where the "walls" should be built, and its unit implementation guidelines define how those "walls" should be built.

Establishing appropriate project guidelines for coding, architectural design and unit implementation are clearly three discrete tasks but just like a bricklayer building a wall, software developers responsible for implementing the design need to be mindful of them all concurrently.

Topics		ASIL			
		A	B	C	D
1a	Hierarchical structure of software components	++✓	++✓	++✓	++✓
1b	Restricted size of software components	++✓	++✓	++✓	++✓
1c	Restricted size of interfaces	+✓	+✓	+✓	+✓
1d	High cohesion within each software component	+✓	++✓	++✓	++✓
1e	Restricted coupling between software components	+✓	++✓	++✓	++✓
1f	Appropriate scheduling properties	++	++	++	++
1g	Restricted use of interrupts	+	+	+	++

"++" The method is highly recommended for this ASIL.
"+" The method is recommended for this ASIL.
"o" The method has no recommendation for or against its usage for this ASIL.
✓ Satisfied by the LDRA tool suite

Figure 7: Mapping the capabilities of the LDRA tool suite to "Table 3: Principles for software architectural design" specified by ISO 26262-6:2011[23]

Figure 7 and Figure 8 illustrate how the architectural design and unit implementation principles required by ISO 26262-6:2011 can usually be checked by means of static analysis, just like coding guidelines. Figure 7 is a modified version of Table 3 reproduced from ISO 26262-6:2011, and it is superimposed here with an illustration of where compliance can be confirmed by an automated tool suite.

As for the coding guidelines before them, all of these principles are founded on the notion that they make the resulting code more secure, reliable, less prone to error, easier to test and/or easier to maintain, which benefits cybersecurity as well as functional safety. For example:

- Restricted size of software components (1b, Figure 7) is important not least because large, rambling functions are difficult to read, maintain, and test – and hence more susceptible to error.
- Restricted size of interfaces (1c, Figure 7) is important for similar reasons.
- High cohesion within each software component (1d, Figure 7) refers to the level of strength and unity with which different components of a software program are inter-related with each other. High cohesion results from the close linking between the modules of a software program, which in turn impacts on how rapidly it can perform the different tasks assigned to it.
- From a cybersecurity perspective, restricted coupling between software components (1e, Figure 7) is significant with reference to trust boundaries between tasks of differing criticality levels.

Figure 8 is a modified version of Table 3 reproduced from ISO 26262-6:2011, and it is superimposed here with an illustration of where compliance can be confirmed automatically.

[23] Based on table 3 from ISO 26262-6:2011, Copyright © 2015 IEC, Geneva, Switzerland. All rights acknowledged

Topics		ASIL			
		A	B	C	D
1a	One entry and one exit point in subprograms and functions	++✓	++✓	++✓	++✓
1b	No dynamic objects or variables, or else online test during their creation	+✓	++✓	++✓	++✓
1c	Initialization of variables	++✓	++✓	++✓	++✓
1d	No multiple use of variable names	+✓	+✓	++✓	++✓
1e	Avoid global variables or else justify their usage	+✓	+✓	++✓	++✓
1f	Limited use of pointers	o	+✓	+✓	++✓
1g	No implicit type conversions	+✓	++✓	++✓	++✓
1h	No hidden data flow or control flow	+✓	++✓	++✓	++✓
1i	No unconditional jumps	++✓	++✓	++✓	++✓
1j	No recursions	+✓	+✓	++✓	++✓

”++” The method is highly recommended for this ASIL.
“+“ The method is recommended for this ASIL.
“o“ The method has no recommendation for or against its usage for this ASIL.
✓ Satisfied by the LDRA tool suite

Figure 8: Mapping the capabilities of the LDRA tool suite to “Table 8: Design principles for software unit design and implementation” specified by ISO 26262-6:2011[24]

Once again, all of these principles are founded on the notion that they make the resulting code more reliable, less prone to error, easier to test and/or easier to maintain, again to the benefit of both cybersecurity and functional safety. For example:

- The restriction to one entry and one exit point (1a, Figure 8) is a restriction of additional paths in accordance with the design requirement to minimize potential vulnerability.
- Initialization of variables (1c, Figure 8) is an example designed to deal with undefined behaviour in the C and C++ languages. Different compiler vendors can treat uninitialized variables differently. Some might initialize them to zero, but if code relies on that and it is ported to different hardware later in life then it may well behave differently.
- Avoid global variables or else justify their usage (1e, Figure 8). The problem with global variables is that since every function has access to them, it becomes increasingly hard to figure out which functions actually read and write to them making testing and maintenance difficult. Static analysis can help, but it is much cleaner to avoid them.
- No recursion (1j, Figure 8). Recursive functions are difficult to understand, and it is frequently impossible to predict their likely resource usage.

SAE J3061 Section 8.6.6: “*Software Implementation Code Reviews*” and SAE J3061 Section 8.6.10: “*Software Vulnerability Testing*”

Although the flow of the SAE J3061 document sees software implementation code reviews following logically from software integration, they are usually applied in tandem such that the code is reviewed as it is written. A fully integrated tool suite automates that process, helping to ensure that the good practices required by ISO 26262:2011 and SAE J3061 are adhered to whether they are coding rules (Figure 9), design principles, or principles for software architectural design.

[24] Based on table 8 from ISO 26262-6:2011, Copyright © 2015 IEC, Geneva, Switzerland. All rights acknowledged

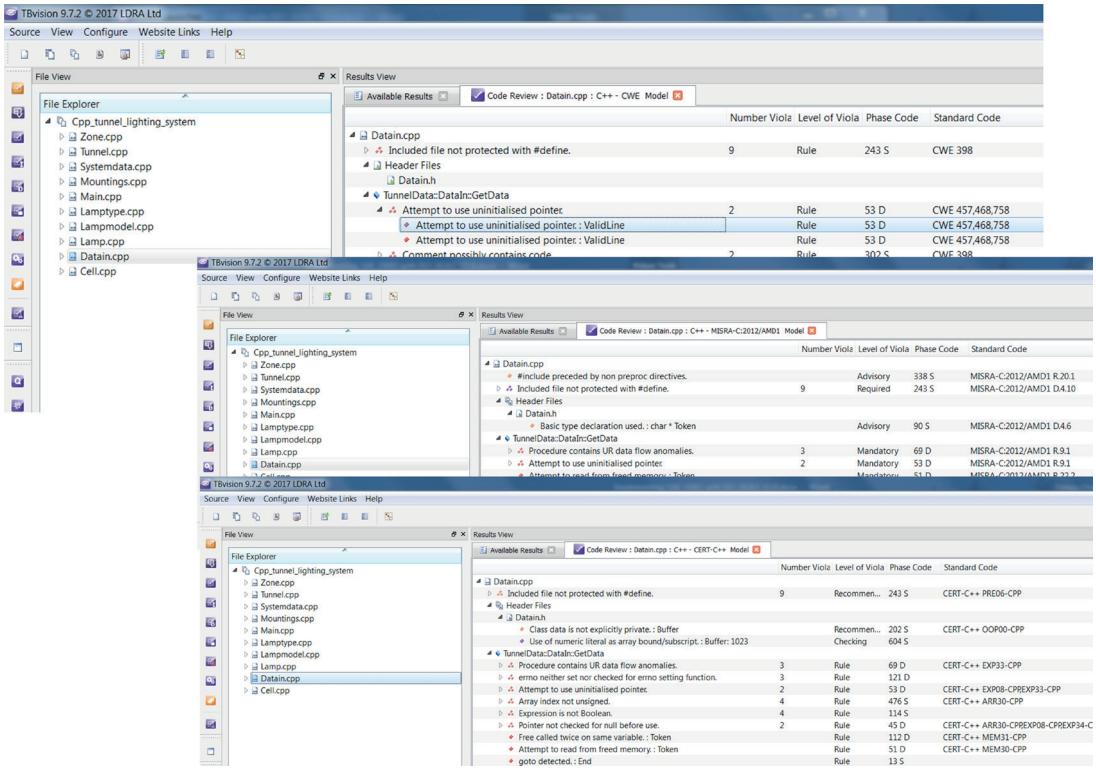

Figure 9: Using the LDRA tool suite to highlight cybersecurity related coding guideline violations with reference to CWE, MISRA C 2012 AMD1 and CERT C++

Metrics relating to such as software component size, complexity, cohesion, and coupling are more concerned with the structure of the code base rather than the use of particular constructs, and so for them to be meaningful the code needs to be a little more complete. Complexity metrics are generated as a product of interface analysis, cohesion evaluated through data object analysis, and coupling through data control coupling analysis (Figure 10).

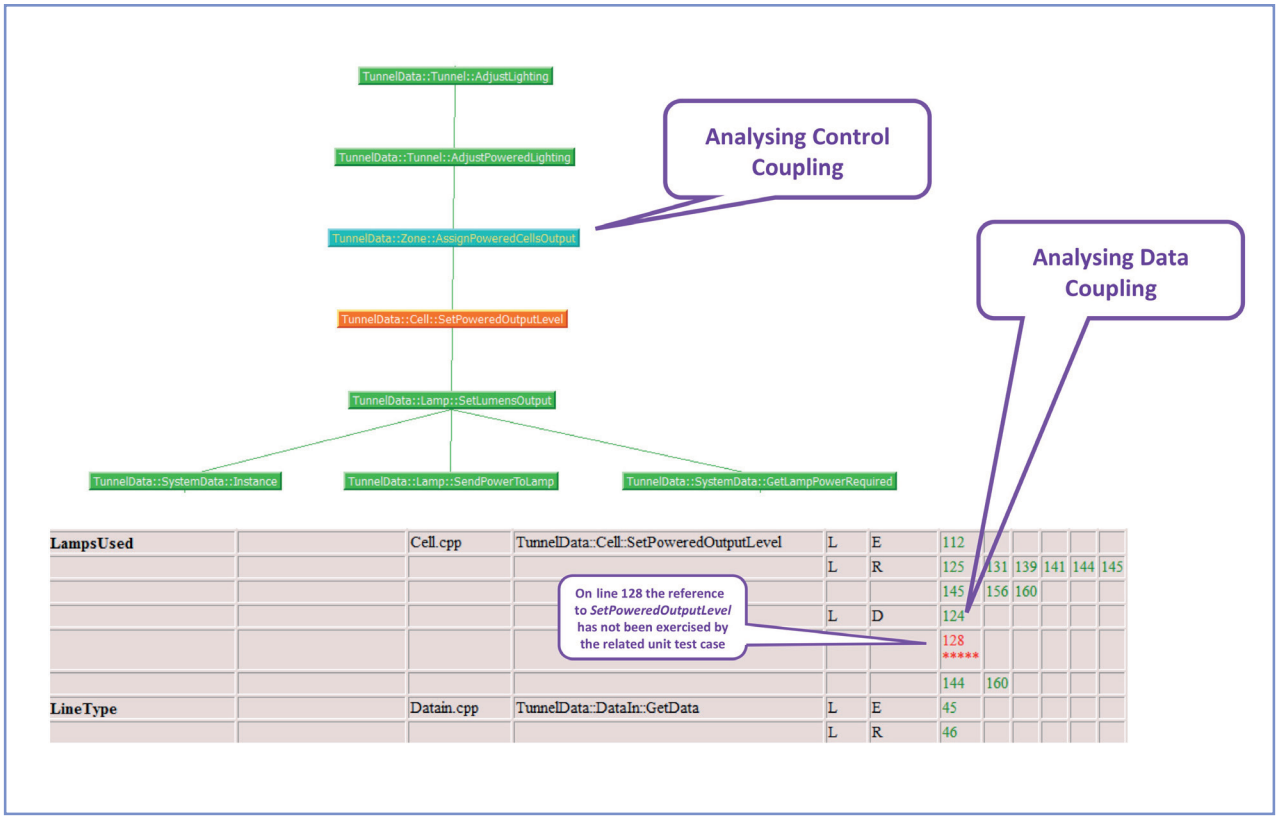

Figure 10: Output from control and data coupling analysis as represented in the LDRA tool suite

The ongoing application of static analysis throughout the code implementation phase provides support and tutelage to the development team. In practice, for developers who are newcomers to ISO 26262 and/or SAE J3061, the use of the tool often progresses from pointing out where violations have occurred, to a point where it provides confirmation that there are none.

SAE J3061 suggests that *"Code reviews should be conducted on the software throughout the software design and implementation phase, especially for new or modified software, during appropriate gate reviews. These reviews should analyse and identify coding methods and constructs that may pose or introduce risk or vulnerabilities into the system."* The traditional approach to enforcing adherence to the coding guidelines would be to use peer code reviews.

These may well still have a place in the development process – they can be very useful as an aid to learning between team members, for example – but automating the more tedious checks is far more efficient and less prone to error.

Experience shows that static analysis is a particularly useful technique in identifying software vulnerabilities in code that would otherwise successfully compile, since while the code may meet the syntactic requirements of the language it still may contain unpredictable or undefined behaviours.

SAE J3061 Section 8.6.7: "*Software Unit Testing*" and SAE J3061 Section 8.6.8: "*Software Integration and Testing*"

ISO 26262-6:2011 Sections 9 and 10 state that;

"The objective of this [Software unit testing] *sub-phase is to demonstrate that the software units fulfil the software unit design specifications and do not contain undesired functionality."*

"The first objective of this [Software integration and testing] *sub-phase is to integrate the software elements."*

"The second objective of this [Software integration and testing] *sub-phase is to demonstrate that the software architectural design is realized by the embedded software."*

All of these objectives remain valid when applying SAE J3061 in tandem with ISO 26262, with the following addition:

"*The second objective of this* [Software Integration and Testing] *sub-phase is to demonstrate that the software architectural design is realized by the embedded software and there is no malicious logic which can be sabotaged*."

The techniques discussed so far in relation to compliance with ISO 26262-6:2011 have largely focused on static analysis – that is, an automated "inspection" of the source code. Just as static analysis techniques embraced verification of adherence to the ISO 26262-6:2011 guidelines for coding, architectural design and unit implementation, the dynamic analysis techniques (which involve the execution of some or all of the code) are helpful in both software unit testing, and software integration and testing.

Table 10 and Table 13 in ISO 26262-6:2011 list techniques and metrics for performing unit and integration tests, for which a primary function is to ensure that the expected functionality and software interfaces are verified at the unit and integration levels. Software unit and integration tests need to be executed on target hardware and if the developed unit or integrated software is "safety-related", then test results should comply with safety requirements. Fault injection and resource tests help further ensure robustness and resilience. For organizations that apply model-based development back-to-back testing at the model and code level is recommended.

Figure 11 is a modified version of Table 13 reproduced from ISO 26262-6:2011, and it is superimposed here with an illustration of where compliance can be confirmed automatically.

Topics		ASIL			
		A	B	C	D
1a	Requirement-based test	++✓	++✓	++✓	++✓
1b	Interface test	++✓	++✓	++✓	++✓
1c	Fault injection test	+	+	+	++
1d	Resource usage test	+✓	+✓	+✓	++✓
1e	Back-to-back test between model and code, if applicable	+✓	+✓	++✓	++✓

”++” The method is highly recommended for this ASIL.
“+“ The method is recommended for this ASIL.
“o“ The method has no recommendation for or against its usage for this ASIL.
✓ Satisfied by the LDRA tool suite

Figure 11: Mapping the capabilities of the LDRA tool suite to “Table 13: Methods for software unit testing” specified by ISO 26262-6:2011[25]

Each developed software unit needs to be tested with reference to the software unit design specification. Test procedures then need to be authored, reviewed, and executed to ensure the software unit does not contain any undesired functionality. Unit tests can then be executed on the target hardware or simulated environment based on the verification plan and verification specification. Once the test procedures are executed actual outputs are captured and compared with the expected results. Pass/Fail results are then reported and software safety requirements are verified accordingly.

Integration testing is designed to ensure that when the units are working together in accordance with the software architectural design, they meet the related specified requirements. In practice, these integration tests typically involve the verification of safety and non-safety related software functions, including those related to cybersecurity.

It is desirable for all dynamic testing to use environments which correspond closely to the target environment and hence test dependencies between hardware and software. However, that is not always practical and one approach involves developing the tests in a simulated environment and then, once proven, re-running them on the target. ISO 26262-6:2011 provides specific guidance on simulation versus target testing:

“If the software unit testing is not carried out in the target environment, the differences in the source and object code, and the differences between the test environment and the target environment, shall be analysed in order to specify additional tests in the target environment during the subsequent test phases.”

Should changes become necessary – perhaps as a result of a failed dynamic test, or in response to a requirement change - then all impacted unit and integration tests would need to be re-run (regression tested). These regression tests can be automated and systematically re-applied, as development progresses, to ensure that new functionality does not compromise any that is established and proven. As for the static analysis of product code, ISO 26262:2011 does not require that any of these tests deploy software test tools. However, once again, such tools are capable of making the test process far more efficient, especially where the project is substantial.

[25] Based on table 10 from ISO 26262-6:2011, Copyright © 2015 IEC, Geneva, Switzerland. All rights acknowledged

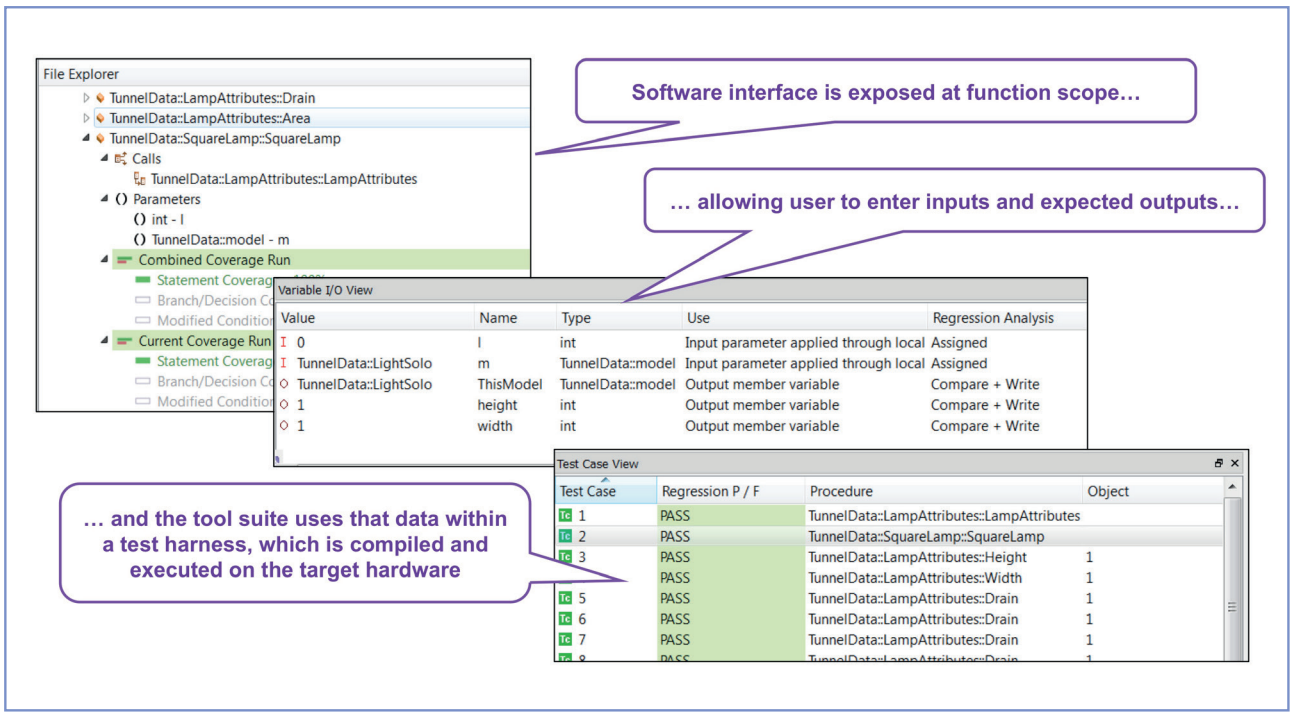

Figure 12: Performing requirement based unit-testing using the LDRA tool suite

The example in Figure 12 shows how the software interface is exposed at the function scope allowing the user to enter inputs and expected outputs. The tool suite then generates a test harness, which is compiled and executed on the target hardware. Actual outputs are captured, along with structural coverage data, and then compared with the expected outputs specified in the test cases.

Figure 13 is a modified version of Table 10 reproduced from ISO 26262-6:2011, and it is superimposed here with an illustration of where compliance can be confirmed automatically.

Topics		ASIL			
		A	B	C	D
1a	Analysis of requirements	++✓	++✓	++✓	++✓
1b	Generation and analysis of equivalence classes	+✓	++✓	++✓	++✓
1c	Analysis of boundary values	+✓	++✓	++✓	++✓
1d	Error guessing	+✓	+✓	+✓	+✓

”++” The method is highly recommended for this ASIL.
“+“ The method is recommended for this ASIL.
“o“ The method has no recommendation for or against its usage for this ASIL.
✓ Satisfied by the LDRA tool suite

Figure 13: Mapping the capabilities of the LDRA tool suite to “Table 10: Methods for software unit testing” specified by ISO 26262-6:2011, promoting safety and security at the early stages of the lifecycle[26]

The inputs and expected outputs defined in each test case are typically derived from requirements (Figure 13, 1a) to ensure intended functionality is verified. Various other forms of tests including negative tests, fault injection (Figure 11, 1c), and robustness tests use the same mechanism. Unit tests become integration tests as units are tested as part of a call tree, rather than in isolation. Exactly the same test data can be used to validate the code in both cases.

[26] Based on table 13 from ISO 26262-6:2011, Copyright © 2015 IEC, Geneva, Switzerland. All rights acknowledged

The analysis of boundary values can be automated using an “extreme test” facility (Figure 13, 1c) to automatically generate a series of unit test cases. The same facility also provides a facility for the definition of equivalence boundary values such as minimum value, value below lower partition value, lower partition value, upper partition value and value above upper partition boundary. Features like automated stub management, global variable declarations, and exception handling help to complete a comprehensive unit test facility.

SAE J3061 recommends performing penetration (or “pen”) testing and fuzz testing in accordance with software security requirements. Pen testing is usually a system test techniques, performed late in the lifecycle. The types of pen tests identified by the standard are:

- *“Black Box : In Black Box testing there is very little (or no) pre-disclosed information.”*
- *“White Box : In White Box testing the tester has access to all information about the system, such as the results of the vulnerability assessment, access to source code, etc.”*
- *“Grey Box : Grey Box testing is part way between white box and black box testing, with the tester having access to system documentation and algorithms, and knowledge about the internal structure of the system.”*

Robustness testing is closely related to fuzz testing, and is complementary to it in that it can be performed at unit or integration level, and can generate structural coverage metrics (as discussed later in this document). Both robustness and fuzz testing are designed to ensure that the software does not deviate from its intended behaviour in response to unexpected inputs. Robustness testing can incorporate techniques such as boundary value analysis, conditional value analysis, error guessing, and error seeding.

It is advisable to focus robustness tests on the highest risk areas identified during threat analysis, particularly external data interfaces such as Bluetooth, USB, cellular, Wi-Fi, ODB2, and HMI.

Figure 14 shows the how robustness testing can be implemented by using a unit test tool with the capability to automatically generate test cases.

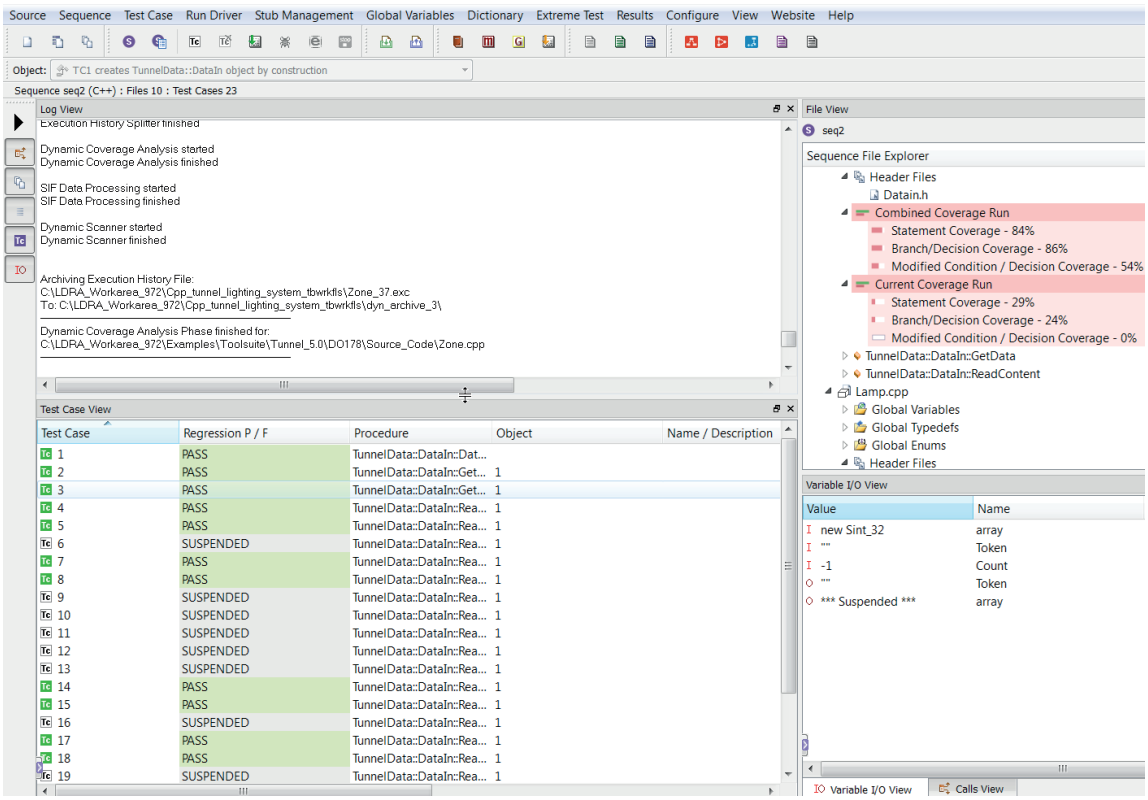

Figure 14: Using the TBeXtreme component of the LDRA tool suite to perform robustness testing

Structural Coverage Metrics

In addition to showing that the software functions correctly, dynamic analysis is also used to generate structural coverage metrics. In tandem with the coverage of requirements at the software unit level, these metrics provide the necessary data to evaluate the completeness of test cases and to demonstrate that there is no unintended functionality. Statement, branch and MC/DC coverage are provided by both the unit test and system test facilities.

Figure 15 is a modified version of Table 12 reproduced from ISO 26262-6:2011, and it is superimposed here with an illustration of where compliance can be aided by automated test tools.

Topics		ASIL			
		A	B	C	D
1a	Statement coverage	++✓	++✓	+✓	+✓
1b	Branch coverage	+✓	++✓	++✓	++✓
1c	MC/DC (Modified Condition/Decision Coverage)	+✓	+✓	+✓	++✓

”++” The method is highly recommended for this ASIL.
“+“ The method is recommended for this ASIL.
“o“ The method has no recommendation for or against its usage for this ASIL.
✓ Satisfied by the LDRA tool suite

Figure 15: Mapping the capabilities of the LDRA tool suite to “Table 12: Structural coverage metrics at the software unit level” specified by ISO 26262-6:2011[27]

These packages can also operate in tandem, so that (for instance) coverage can be generated for most of the source code through a dynamic system test and that can be complemented using unit tests to exercise constructs inaccessible during normal system operation, such as defensive constructs (Figure 16)

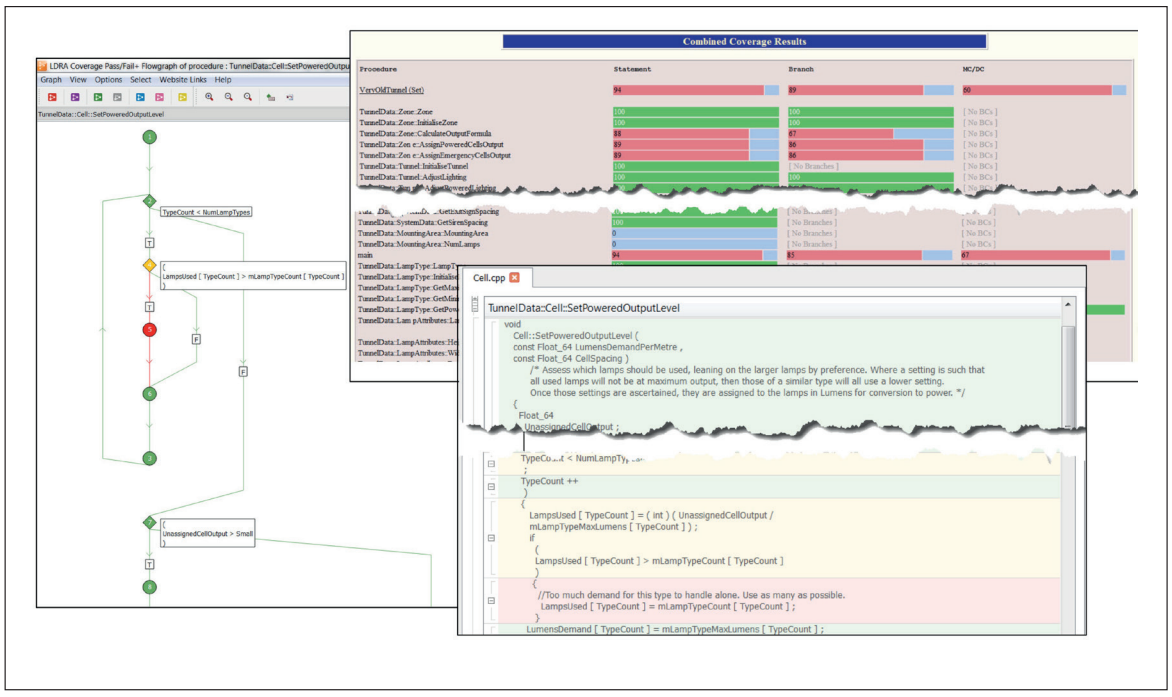

Figure 16: Examples of representations of structural coverage within the LDRA tool suite

[27] Based on table 14 from ISO 26262-6:2011, Copyright © 2015 IEC, Geneva, Switzerland. All rights acknowledged

The generation of functional and call coverage data can also be automated by executing tests on the target. Figure 17 is a modified version of Table 15 reproduced from ISO 26262-6:2011, and it is superimposed here with an illustration of where compliance can be confirmed automatically.

Topics		ASIL			
		A	B	C	D
1a	Function coverage	+✓	+✓	++✓	++✓
1b	Call coverage	+✓	+✓	++✓	++✓

"++" The method is highly recommended for this ASIL.
"+" The method is recommended for this ASIL.
"o" The method has no recommendation for or against its usage for this ASIL.
✓ Satisfied by the LDRA tool suite

Figure 17: Mapping the capabilities of the LDRA tool suite to the "Table 15: Structural coverage metrics at the software architectural level" specified by ISO 26262 Part 6[28]

Figure 18 illustrates some of the ways functional and call coverage can be represented.

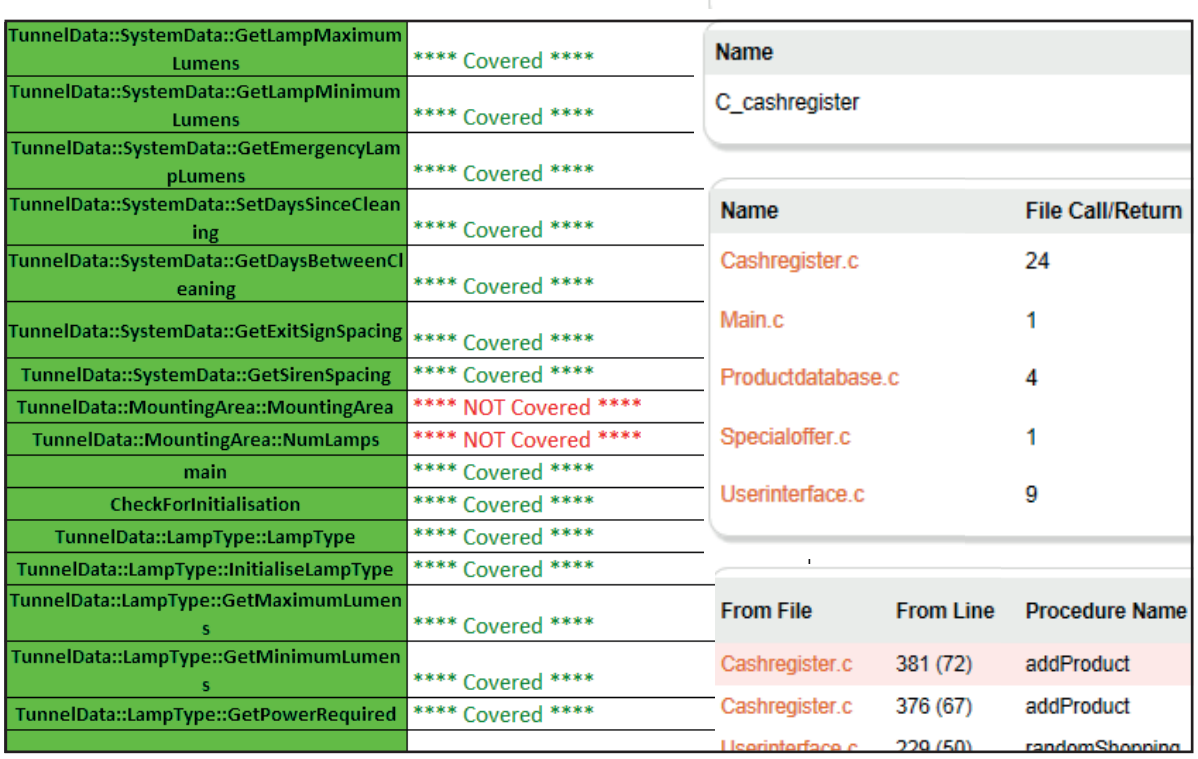

Figure 18 Examples of representations of function and call coverage within the LDRA tool suite

SAE J3061 Section 8.6.9: "*Verification/Validation to Software Cybersecurity Requirements*"

SAE J3061 states that "*During the implementation phase, software components are acquired and/or built, integrated, configured, tested, and documented. Cybersecurity tests covering all software cybersecurity requirements are conducted to verify that the actual results match the required results.*

The software cybersecurity design is traceable to, and validated, against the software cybersecurity requirements, and the implementation is traceable to and validated against the software cybersecurity design."

[28] Based on table 15 from ISO 26262-6:2011, Copyright © 2015 IEC, Geneva, Switzerland. All rights acknowledged

Bi-directional traceability is also referenced throughout ISO 26262:2011. It is explicitly mentioned in ISO 26262-6:2011 paragraph 4.7:

"This implies bi-directional traceability between the software architectural design and the software safety requirements."

More generally though, its presence is implied by the need for each development phase to accurately reflect the one before it, such as in ISO 26262-6:2011 paragraph 8.1:

"The third objective of this sub-phase is the static verification of the design of the software units and their implementation."

Tools can help in establishing the traceability from system level requirements, through high-level requirements and low level requirements, and on to source code and test cases (Figure 19 and Figure 20).

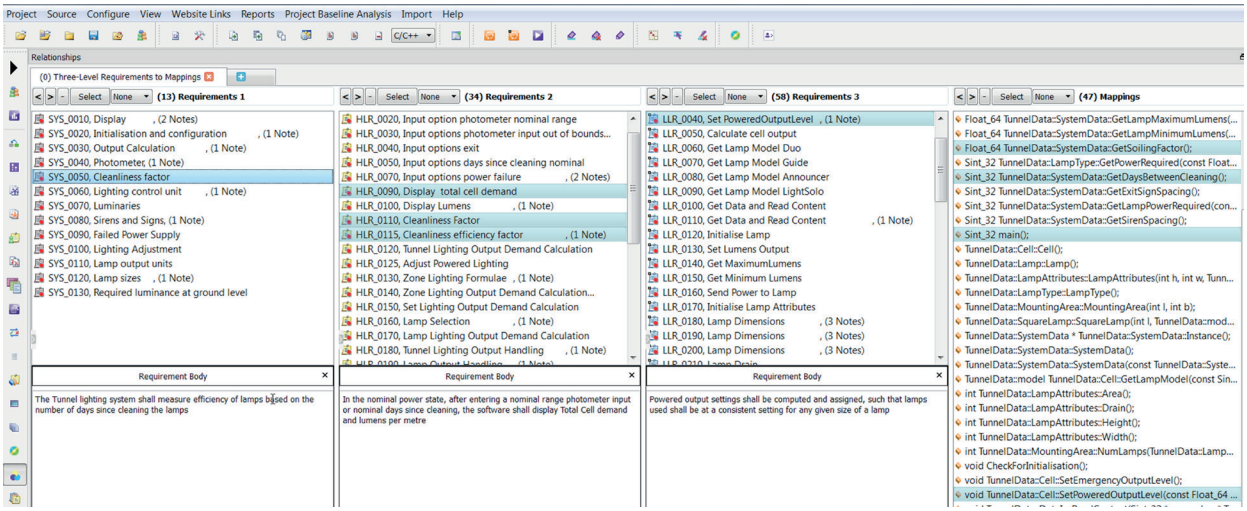

Figure 19: Traceability established for safety and cybersecurity requirements using the TBmanager component of the LDRA tool suite, which will ensure complete requirements coverage and verify that all source code is traceable to requirements

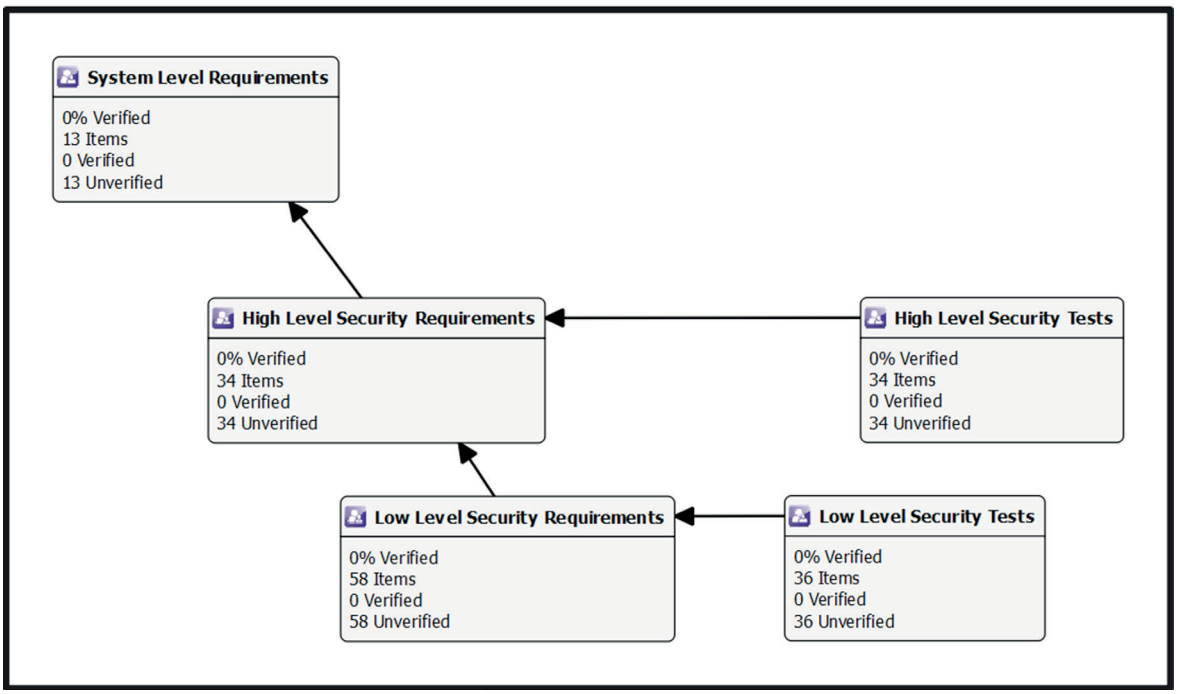

Figure 20: The Uniview graphic from the TBmanager component of the LDRA tool suite, showing the relationships between cybersecurity tests and requirements

SAE J3061 section 5.6 discusses the need for the implementation of cybersecurity in incident response. That objective has the potential to have a disproportionate impact on the nature of many automotive development teams, who are brought together to work on a particular project and go their separate ways after its conclusion.

There may be no incident of note for years after the launch of a product or feature – or to put it another way, perhaps not until all the developers with intimate knowledge of it have moved on. The availability of a tool to provide instant impact analysis to those tasked with actioning the Impact Response Process would clearly be invaluable.

Confidence in the Use of Software Tools

ISO 26262-8:2011 section 11 defines a mechanism to provide evidence that the software tool chain can be relied upon.

The required level of confidence in a software tool depends upon the circumstances of its deployment, with particular reference to:

- The possibility that the malfunctioning software tool and its corresponding erroneous output can introduce or fail to detect errors in a safety-related item or element being developed, and
- The confidence in preventing or detecting such errors in its corresponding output.

For example the LDRA tool suite has been qualified for ISO 26262 up to ASIL D, which removes considerable user overhead in providing evidence of that confidence.

ZERTIFIKAT ◆ CERTIFICATE ◆ 認証書 ◆ СЕРТИФИКАТ ◆ CERTIFICADO ◆ CERTIFICAT

TÜV SÜD Product Service

CERTIFICATE
No. Z10 17 12 84753 005

Holder of Certificate: LDRA Ltd.
Portside, Monks Ferry
Wirral
Merseyside CH41 5LH
UNITED KINGDOM

Factory(ies): 84753

Certification Mark:

Product: Software Tool for Safety Related Development

Model(s): LDRA tool suite
LDRArules
LDRAcover
LDRAunit
LDRAlite

Parameters: The certified tools, classified T2, fulfil the requirements for support tools according to IEC 61508-3 and EN 50128. The tools are qualified to be used in safety-related software development according to IEC 61508, EN 50128 and ISO 26262. It is suitably validated for use in safety-related development according to IEC 62304.

The test report is a mandatory part of this certificate.

Tested according to: IEC 61508-3:2010
ISO 26262-8:2011
EN 50128:2011
IEC 62304:2015

The product was tested on a voluntary basis and complies with the essential requirements. The certification mark shown above can be affixed on the product. It is not permitted to alter the certification mark in any way. In addition the certification holder must not transfer the certificate to third parties. See also notes overleaf.

Test report no.: LW85043C

Valid until: 2022-12-20

Date, 2017-12-21 (Peter Weiss)

Page 1 of 1

TÜV SÜD Product Service GmbH · Zertifizierstelle · Ridlerstraße 65 · 80339 München · Germany

SGS TÜV SAAR

CERTIFICATE NO FS/71/220/17/0249 PAGE 1/1

LICENCE HOLDER: LDRA LTD, PORTSIDE, MONKS FERRY, WIRRAL, CH41 5LH, UNITED KINGDOM

MANUFACTURING PLANT: LDRA LTD, PORTSIDE, MONKS FERRY, WIRRAL, CH41 5LH, UNITED KINGDOM

PROJECT NO/-ID: M15F-AU01

LICENSED TEST MARK: SGS TÜV SAAR FUNKTIONALE SICHERHEIT GEPRÜFT / FUNCTIONAL SAFETY APPROVED

CERT. REPORT NO.: M15F0001

Tested according to: IEC 61508:2010; clause 7.4.4
ISO 26262:2011; clause 11.4.6
EN 50128:2011
IEC 60880:2006
IEC 62304:2015

Certified product(s): LDRA tool suite
LDRAunit
LDRArules
LDRAcover
LDRAlite

Model(s): Version 9.6.0
Version 9.7.0
Version 9.7.1

Technical Data and Parameter: Suitable for development of safety related software acc. to:
- ISO 26262 up to ASIL D, TCL1 can be reached
- IEC 61508 up to SIL 4, class T2 tool
- EN 50128 up to SW-SIL 4, class T2 tool
- IEC 62304 up to SW safety Class C
- IEC 60880

Specific Requirements: Any changes to the design, components or processing may require repetition of some parts of the pre-qualification to retain the certification. The certificate report is an integral part of this certificate. The safety related application guidelines shall be maintained.

Certification Body for Functional Safety SGS-TÜV Saar GmbH

Munich, 01.12.2017

Gudrun Neumann

SGS-TÜV Saar GmbH, Hofmannstr. 50, 81379 München, Germany

CERTIFICATE

Figure 21: Evidence of TUV approved qualification of the LDRA tool suite

It is classified as a TI2 (Tool Impact - 2) category tool because although a verification tool can fail to detect existing errors in source code, unlike (say) an auto code generator it cannot introduce errors in to an application.

It is classified as a TI2 (Tool Impact - 2) category tool because although a verification tool can fail to detect existing errors in source code, unlike (say) an auto code generator it cannot introduce errors in to an application.

Figure 22 illustrates how the TI2 classification dictates that an organization using the tool suite in the development of a compliant application is required to estimate the likelihood of them detecting any error introduced by the tool (Tool Error Detection), with categories ranging from TD1 (high confidence) to TD3. Evidence of the reasoning behind that assessment is required.

		Tool Error Detection		
		TD1	TD2	TD3
Tool Impact	T11	TCL1	TCL1	TCL1
	T12	TCL1	TCL2	TCL3

Figure 22: Mapping the characteristics of the LDRA tool suite to "Table 3: Determination of the tool confidence level" from ISO 26262-8:2011[29]

In turn, the Tool Confidence Level (TCL) is then derived from a look-up table as shown in Figure 23. Depending on the user's assessment of the application, the resulting TCL will therefore be either TCL1 or TCL2 for the LDRA tool suite. In all cases except where the tool suite is assigned TCL2 and the product is designated ASIL D, the existence of a TUV certificate (Figure 22) is sufficient to establish confidence in the tool.

	Methods	ASIL			
		A	B	C	D
1a	Increased confidence from use in accordance with 11.4.7	++	++	++	+
1b	Evaluation of the tool development process in accordance with 11.4.8	++	++	++	+
1c	Validation of the software tool in accordance with 11.4.9	+	+	+	++
1d	Development in accordance with a safety standard[a]	+	+	+	++

[a] No safety standard is fully applicable to the development of software tools. Instead, a relevant subset of requirements of the safety standard can be selected.

EXAMPLE Development of the software tool in accordance with ISO 26262, IEC 61508 or RTCA DO-178.

Figure 23: Assessing the need for the qualification the LDRA tool suite when it is designated TCL2 from "Table 5: Qualification of software tools classified TCL2" from ISO 26262-8:2011[30]

Otherwise, the tool is required to be subjected to a validation process (Figure 23). That involves showing that the tool is capable of analysing sample software in the target environment. For example, a Tool Qualification Support Package (TQSP) is available from LDRA to provide that sample software which includes:

- Tool Criteria Evaluation Report
- Tool Qualification Report
- Tool Operational Requirement/Use Cases
- Tool verification cases and Procedures
- Tool Verification Results
- Tool User Manual/Tool Installation Manual

[29] Based on table 3 from ISO 26262-6:2011, Copyright © 2015 IEC, Geneva, Switzerland. All rights acknowledged
[30] Based on table 5 from ISO 26262-8:2011, Copyright © 2015 IEC, Geneva, Switzerland. All rights acknowledged

Conclusions

SAE J3061 can be considered complementary to ISO 26262 in that it provides guidance on best development practices from a cybersecurity perspective, just as ISO 26262 provides guidance on practices to address functional safety.

SAE J3061 also calls for a similar sound development process to that of ISO 26262 and mirrors its development phases, and so the standards are not contradictory such that functional safety requirements are mirrored by cybersecurity requirements, designs must reflect both safety and cybersecurity, and so on.

Neither standard mandates the use of automated tools, but it is clear that their use is almost essential in all but the simplest of projects. Just as the standards complement each other, so the same tools can be used to support those standards concurrently.

Works Cited

ANSYS Esterel SCADE SUITE
http://www.esterel-technologies.com/products/scade-suite/

CAPECTM Common Attack Pattern Enumeration and Classification
https://capec.mitre.org/

Common Vulnerabilities and Exposures
https://cve.mitre.org/

Common Weakness Enumeration
https://cwe.mitre.org/index.html

Common Weakness Enumeration – Scoring CWEs
https://cwe.mitre.org/cwss/cwss_v1.0.1.html

"High-tech vehicles - High-tech ISO safety standards", Maria Lazarte, 10 January 2012
https://www.iso.org/news/2012/01/Ref1499.html

IBM Rational Rhapsody family
http://www-03.ibm.com/software/products/en/ratirhapfami/

IEC 60812:2006 Analysis techniques for system reliability – Procedure for failure mode and effects analysis

ISO 26262-6:2011(E) "Road Vehicles – Functional safety – Part 6: Product development at the software level"

ISO 26262-8:2011(E) "Road Vehicles – Functional safety – Part 8: Supporting processes"

ISO 26262-FIPS Pub 199. "Standards for Security Categorization of Federal Information and Information Systems", February 2004.

Microsoft Developer Network – Improving Web Application security: Threats and Countermeasures
https://msdn.microsoft.com/en-us/library/ff648644.aspx

Microsoft Developer Network – The STRIDE Threat Model
https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx

MISRA C:2012 Amendment 1 – Additional security guidelines for MISRA C:2012. April 2016.

SAE International Surface Vehicle Recommended Practice – J3061TM - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems. Issued 2016-01

Saini, Vineet & Duan, Qiang & Paruchuri, Vamsi. (2008). Threat Modeling Using Attack Trees. Journal of Computing Sciences in Colleges. 23

SEI CERT Coding standards Top 10 Secure Coding Practices
https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices

The FIRST organization – Common Vulnerability Scoring System SIG
https://www.first.org/cvss/

Using Fault Tree Analysis in Developing Reliable Software, E.O.Ovstedal, IFAC Proceedings Volume 24, Issue 13, October–November 1991, Pages 77-82

www.ldra.com

LDRA UK & Worldwide
Portside, Monks Ferry,
Wirral, CH41 5LH
Tel: +44 (0)151 649 9300
e-mail: info@ldra.com

LDRA Technology Inc.
2540 King Arthur Blvd, Suite 228
Lewisville, Texas 75056
United States
Tel: +1 (855) 855 5372
e-mail: info@ldra.com

LDRA Technology Pvt. Ltd.
Unit No B-3, 3rd floor Tower B,
Golden Enclave. HAL Airport Road
Bengaluru
560017
India
Tel: +91 80 4080 8707
e-mail: india@ldra.com