

# **Achieving the ISO/SAE 21434 software objectives**

---

## **Cost effective software certification for automotive cybersecurity**

[www.ldra.com](http://www.ldra.com)

© LDRA Ltd. This document is property of LDRA Ltd. Its contents cannot be reproduced, disclosed or utilised without company approval.

Background .....	3
Superseding SAE J3061 .....	3
A missed opportunity? .....	3
Beyond functional safety .....	4
ISO 26262, HARA, and ASILs .....	4
ISO/SAE 21434, TARA, and – what? .....	5
Seeking an ASIL equivalent .....	5
ISO/SAE 21434 cybersecurity in tandem with ISO 26262 .....	6
ISO/SAE 21434 compliant software development .....	6
ISO/SAE 21434 §8 Continual cybersecurity activities & §13 Operations and maintenance .....	7
ISO/SAE 21434 §8.5 Vulnerability analysis .....	8
ISO/SAE 21434 §9 Concept .....	8
Requirements traceability and ISO/SAE 21434 .....	8
ISO/SAE 21434 §10.4.1 Product development: design .....	9
Coding standards .....	9
ISO/SAE 21434 §10.4.2 Product development: integration and verification .....	11
Requirements-based test .....	11
Interface test .....	12
Resource usage evaluation .....	13
Verification of the control flow and data flow .....	15
Dynamic analysis .....	15
Static analysis .....	15
Conclusions .....	16
Works cited .....	17

## Background

Over the past decade 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” [1] was first published in 2012 in response to this explosion in automotive E/E/PE system complexity and the associated risks to public safety, 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 was always intended to be a stopgap, allowing time for the development of a more formal standard to address the issue more satisfactorily. SAE J3061 was superseded by ISO/SAE 21434:2021 [3] on August 31, 2021.

ISO/SAE 21434 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.

Additionally, the new UNECE WP.29 [4] regulation R155 for CSMS (Cyber Security Management System) [5] has been adopted by UNECE’s World Forum for Harmonization of Vehicle Regulations, making compliance obligatory for vehicle type approval from June 2022. ISO/SAE 21434 is cited in R155 as an appropriate reference for appropriate cybersecurity skills.

## Superseding SAE J3061

ISO/SAE 21434 supersedes SAE International’s 2016 publication SAE J3061™ Cybersecurity Guidebook for Cyber-Physical Vehicle Systems. The two documents differ a little in style in that SAE J3061 related the security and safety processes to each other, and ISO/SAE 21434 decouples them. Despite that distinction, ISO 26262 remains closely linked to the new standard, and is referenced repeatedly by it.

### A missed opportunity?

ISO/SAE 21434 is seen by many as a missed opportunity. Simply comparing the number of pages with those in ISO 26262 suggests why. ISO 26262 runs to 12 parts, many of which have a direct impact on how compliant application software is developed. Part 6 alone, entitled “Product development at the software level”, runs to 66 pages [6]. In contrast, the whole of ISO/SAE 21434 is 81 pages long, and its scope stretches across all aspects of electrical and electronic (E/E) systems within road vehicles, throughout the supply chain.

Developers can expect to find details of what needs to be achieved in ISO 26262 from the perspective of functional safety, and ISO/SAE 21434 from the perspective of cybersecurity. But whereas ISO 26262 also presents details of exactly how to achieve its aims, ISO/SAE 21434 does not.

The failure of ISO/SAE 21434 to give detailed guidance on how to achieve its objectives means that from a software perspective, the standard does little more than ratify the document it replaces. However, ISO/SAE 21434 (and SAE J3061 before it) presents a worthy set of goals for software developers to achieve. From an optimistic perspective, the lack of detail affords flexibility on how they are achieved.

## Beyond functional safety

Despite the clear synergy between the ISO/SAE 21434 and ISO 26262 it is important to note that ISO/SAE 21434 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.

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. ISO/SAE 21434 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 potentially accessed via a connect car, including personal contact details, credit card and other financial information, and browse histories.

### ISO 26262, HARA, and ASILs

The Hazard Analysis and Risk Assessment (HARA) required by ISO 26262:3 is used to identify malfunctions that could lead to hazards, to rate the relevant risks of hazards, and to formulate safety goals.

The resulting derivation of ASILs (Automotive Safety Integrity Levels) is a key concept in the development process defined by ISO 26262. ASILs are designed to allow the level of effort invested in ensuring the prevention of hazardous events to be proportionate.

Each hazardous event is assigned a severity classification (So-S<sub>3</sub>), an exposure classification (Eo-E<sub>4</sub>), and a controllability classification (Co-C<sub>3</sub>). The higher numerical values representing the least desirable characteristic in each case. The likelihood of resulting harm is naturally a combination of these factors, and that is reflected in the assigned ASIL.

ISO 26262 requires the level of effort to be proportionate to ASIL, and not just to severity. Even if a hazardous event is (say) potentially life threatening, there is no need to invest heavily in its prevention if it is incredibly unlikely to happen.

Methods		ASIL			
		A	B	C	D
<b>1a</b>	Requirement-based test	++	++	++	++
<b>1b</b>	Interface test	++	++	++	++
<b>1c</b>	Fault injection test	+	+	++	++
<b>1d</b>	Resource usage test	++	++	++	++
<b>1e</b>	Back-to-back test between code and model, if applicable	+	+	++	++
<b>1f</b>	Verification of the control flow and data flow	+	+	++	++
<b>1g</b>	Static code analysis	+	++	++	++
<b>1h</b>	Static code analysis based on abstract interpretation	++	++	++	++
"++" 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.					

Figure 1: "Methods for verification of software integration" specified by Table 10 in ISO 26262-6:2018

## ISO/SAE 21434, TARA, and – what?

The Threat Agent Risk Assessment (TARA) suggested by ISO/SAE 21434 is analogous to ISO 26262 HARA. TARA is a threat-based methodology to help identify, assess, prioritize, and control cybersecurity risks. It is a practical method to determine the most critical exposures while taking into consideration mitigation controls and accepted levels of risk.

The calculation of a “risk value” is like the calculation of an ASIL in that it accounts for the severity and likelihood of a successful attack, dependent on several factors:

- threat scenario identification
- impact
- attack path
- attack feasibility for that path

The “impact ratings” for safety damage are taken from the definitions in ISO 26262. They use the same impact metric as that used to ascertain ISO 26262 ASIL ratings. That principle is extended in ISO/SAE 21434 to address threats with the potential to cause financial damage, operational damage, and privacy damage (Figure 2).

Impact rating Damage category	Severe	Major	Moderate	Negligible
Safety criteria used by ISO/SAE 21434 are taken from ISO 26262-3:2018	S3: Life threatening injuries, fatal injuries	S2: Severe and life-threatening injuries (survival probable)	S1: Light and moderate injuries	So: No injuries
Financial impact	Catastrophic consequences which might not be overcome	Inconvenient consequences which can be overcome	Inconvenient consequences, overcome with limited resources	No effect, negligible consequences or is irrelevant
Operational	Loss or impairment of a core vehicle function	Loss or impairment of an important vehicle function	Partial degradation of a vehicle function	No perceivable impairment of a vehicle function
Privacy	Significant or even irreversible impact to the road user	Serious impact to the road user	Inconvenient consequences to the road user	Negligible consequences to the road user

*Figure 2: Abbreviated impact rating descriptions taken from ISO/SAE 21434 tables F.1 to F.4 inclusive*

Not only does ISO/SAE 21434 bring formal development to less safety-critical domains, but 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.

### Seeking an ASIL equivalent

ISO/SAE 21434 is much less prescriptive of the TARA approach to be taken, as compared with ISO 26262 HARA. But perhaps more significantly, it stops short of defining an ASIL equivalent. Unlike ISO 26262, ISO/SAE 21434 does not map the level of validation and verification effort to the criticality of the software under development.

However, these ratings do lend themselves to mapping to the ASIL categories presented in ISO 26262.

Figure 3 shows a reproduction of the example table superimposed with risk values, the numeric values of which will be dependent on the calculation approach selected. If this represents best practice where safety is critical, it seems logical that the same approach would be equally appropriate when the application is critical in other ways.

Methods		ASIL			
		A	B	C	D
		ISO/SAE 21434 risk value			
		Low	Moderate	High	Very high
<b>1a</b>	Requirement-based test	++	++	++	++
<b>1b</b>	Interface test	++	++	++	++
<b>1c</b>	Fault injection test	+	+	++	++
<b>1d</b>	Resource usage test	++	++	++	++
<b>1e</b>	Back-to-back test between code and model, if applicable	+	+	++	++
<b>1f</b>	Verification of the control flow and data flow	+	+	++	++
<b>1g</b>	Static code analysis	+	++	++	++
<b>1h</b>	Static code analysis based on abstract interpretation	++	++	++	++
”++” 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.					

Figure 3: Superimposing ISO/SAE 21434 criticality groupings on to the “Methods for verification of software integration” specified by Table 10 in ISO 26262-6:2018

## ISO/SAE 21434 cybersecurity in tandem with ISO 26262

SAE J3061 explicitly tied its development process to that of ISO 26262. Although ISO/SAE 21434 is less tightly bound it does repeatedly reference ISO 26262 and there will be many cases where both standards apply. Indeed, 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.

Even where there is no safety consideration, adopting and adapting proven ISO 26262 best practice to address the high-level demands of ISO/SAE 21434 is a pragmatic approach. It provides the opportunity to apply tools and techniques that are known to development teams, and in many cases already available to them.

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 ISO/SAE 21434 compliant applications, whether ISO 26262 is also applicable or not.

## ISO/SAE 21434 compliant software development

Figure 4 shows a modified V-model illustrating the relationships between ISO/SAE 21434 sections that have the most impact on software development. Each element is expanded in the following sections.

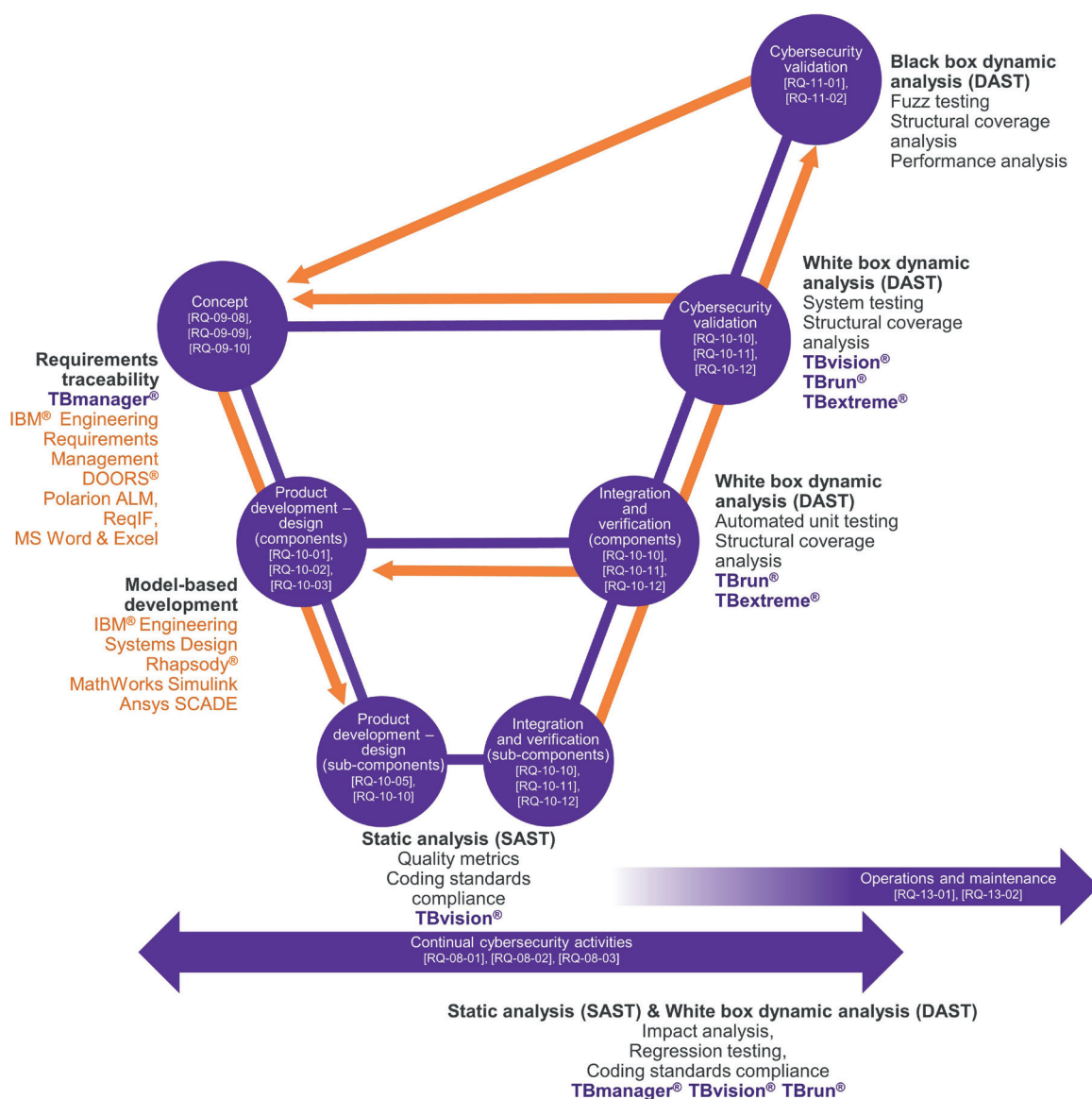


Figure 4: Representation of the ISO/SAE 21434 life cycle as a modified V-model

Despite the lack of detailed guidance for software developers in ISO/SAE 21434, the advice that is given is sound. The following sections explain in more detail how the principles outlined by the standard might be applied.

## ISO/SAE 21434 §8 Continual cybersecurity activities & §13 Operations and maintenance

Developers familiar with functional safety development are used to defending the world against their system. Once that system is developed in a functionally safe manner, the world remains protected for as long as the system is in service.

Secure software development involves defending a system against an aggressive world that is forever evolving new and more sophisticated methods of attack, as addressed by ISO/SAE 21434 §8.8 “Cybersecurity monitoring”, with [RQ-08-03] explicitly requiring that “Cybersecurity information shall be collected and triaged to determine if the cybersecurity information becomes one or more cybersecurity events”

Unlike the world of functional safety, the development lifecycle for a secure system continues after a product is released. This is reflected in ISO/SAE 21434 §13.3 “Cybersecurity incident response”, and by the [RQ-13-01] and [RQ-13-02] requirements for a “Cybersecurity incident response plan” to be defined and implemented.

The ability to enforce coding standards on software modifications, re-run unit tests (regression testing), and to apply impact analysis when security is breached are all invaluable tools in ensuring an effective and efficient response to changes during development, or breaches after deployment. The LDRA tool suite [7] provides all these capabilities in a single, integrated environment which lends itself equally to the development process and to product support after launch.

## ISO/SAE 21434 §8.5 Vulnerability analysis

**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:

**S**poofing identity;  
**T**ampering with Data;  
**R**epudiation;  
**I**nformation Disclosure;  
**D**enial of Service;  
**E**levation of Privilege

And for risk ratings:

**D**amage;  
**R**eproducibility;  
**E**xploitability;  
**A**ffected users;  
**D**iscoverability

Vulnerability analysis is a key differentiator between the development lifecycles of cybersecurity critical systems, and those concerned only with functional safety. The standard uses only generalises the principles involved and so it is useful to consider how they might apply to a connected, embedded software system.

One approach involves the concept of “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 first step in a Software Vulnerability Analysis (SVA) 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 a “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 help 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 [5], such as the sanitization of data sent

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 [6], and/or DREAD [7] to identify threats based on the break-down of the system.

The Common Vulnerability Scoring System (CVSS) [8] and Common Weakness Scoring System (CWSS) [9] are associated with CVE [10] and CWE [11] respectively and can each help to categorize vulnerabilities and weaknesses in software whether at module, application, or source code level.

These threat identifications and categorisations dovetail naturally into the TARA principles discussed in the standard, helping to ensure that proportionate security measures are applied.

## ISO/SAE 21434 §9 Concept

The concept phase involves consideration of vehicle level functionality. Functions are implemented in “items” for which cybersecurity goals are defined. Cybersecurity goals are high level requirements which must be reflected in subsequent design and implementation phases. Demonstrating that traceability by traditional means can prove to be a project management headache especially when tests fail, or requirements change.

## Requirements traceability and ISO/SAE 21434

Cybersecurity requirements traceability is a key objective of ISO/SAE 21434, just like functional safety requirements traceability in ISO 26262. ISO/SAE 21434 requirements [RQ 09 08], [RQ 09 09], and [RQ 09 10] discuss the derivation of cybersecurity requirements from cybersecurity specifications, and that principle of traceability is established throughout the development lifecycle.

Other examples of that recurring them include [RQ-10-02] and [RQ-10-03] where traceability between requirements and design is addressed, and [RQ 10 08] and [RQ 10 09] which discuss traceability between component implementation and specification.

Keeping track of both traceability to the project requirements and to the objectives of the standard itself can present a project management nightmare, especially when changes occur. In many cases ISO 26262 will be applied concurrently, and standards such as AUTOSAR [16] and ASPICE [17] may also apply. The TBmanager component of the LDRA tool suite can help by automating traceability not only to the collated objectives from one or more standards, but also to project requirements [18].

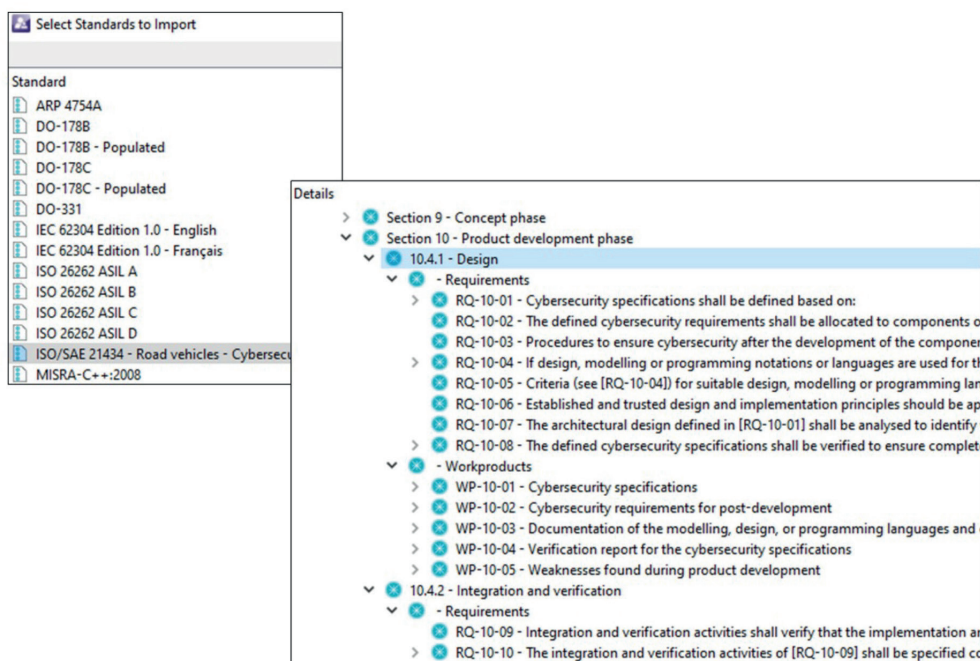


Figure 5: Selecting standards for automated traceability using the TBmanager component of the LDRA tool suite

Automated traceability ensures access to a requirements traceability matrix that is always pertinent and up to date, providing instant and current product status information and a permanently up-to-date Requirements Traceability Matrix (RTM).

## ISO/SAE 21434 §10.4.1 Product development: design

### Coding standards

ISO/SAE 21434 requirement [RQ-10-05] suggests that an appropriate language subset (often known as a coding standard) is used.

#### Language subsets

There are many language subsets (or “coding standards”) each with differing attributes but nevertheless with strong similarities, especially when referencing the same language. The most popular standards include:

<b>C</b>	<b>C++</b>
MISRA C	MISRA C++
CERT C	CERT C++
CWE	JSF++ AV
	HIC++

**Java**  
CWE  
CERT J

There are many different coding standards available (sidebar). ISO/SAE 21434 cites the MISRA [19] and SEI CERT [20] standards as examples of appropriate language subsets. ISO 26262 also recommends the MISRA language subsets from a functional safety perspective.

The TBvision component of the LDRA tool suite [21] can be used to verify adherence to the coding rules specified by the coding standard, style guide, and/or language subset. The traditional approach to enforcing adherence to such 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 (Figure 6).

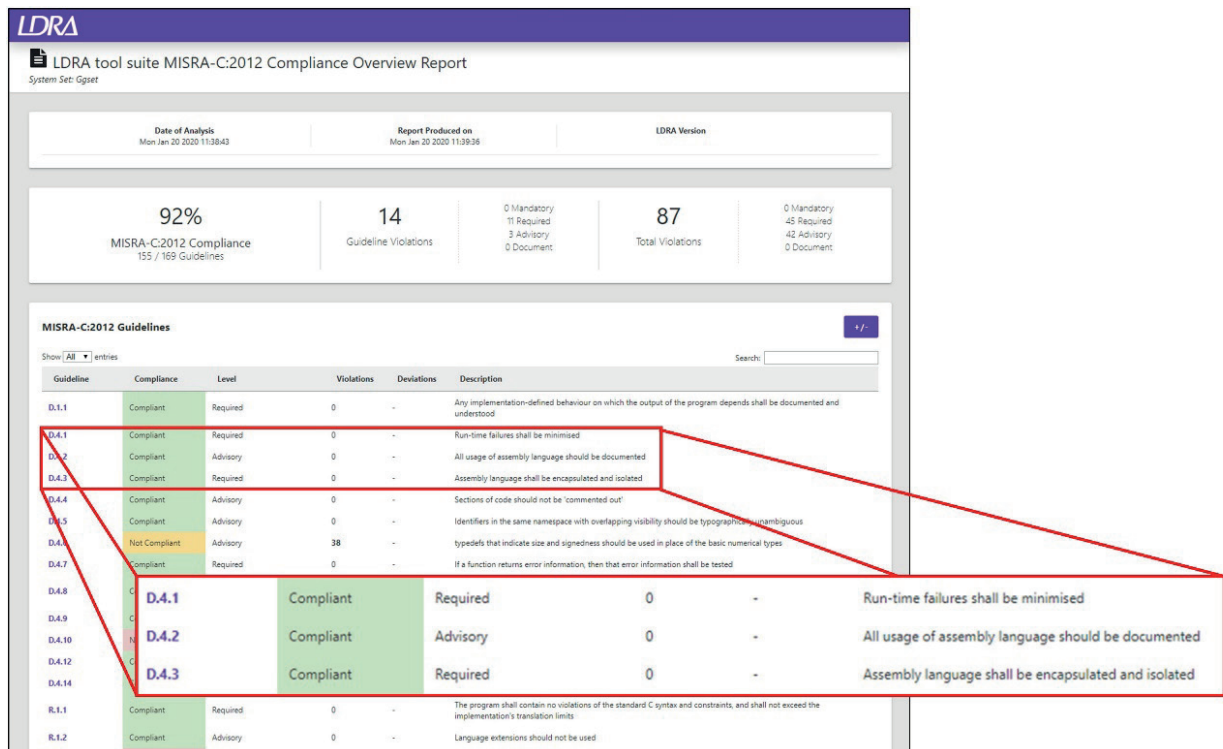


Figure 6: Reviewing coding rules and guidelines in the LDRA tool suite

ISO/SAE 21434 does not enforce the use of any particular coding standard, and it is entirely possible to devise a project specific set of coding rules. It is also possible to choose one of the established standards as a basis, and to manipulate, adjust and add to it to make it more appropriate for a particular application. Clearly, if a tool is to be useful in such circumstances, then it too must be able to accommodate these adjustments.

The TBexclude [22] module (Figure 7) offers a multi-tier coding violation exclusion capability enabling the suppression of rule violation reports at the project, team, or individual user levels – perhaps in support of a MISRA deviation process.

The phased prioritization and correction of violations simplifies the development workflow. A simple right click in the TBvision interface excludes any violation, while groups of violations can be excluded according to identifiers associated with analysis phases, or the coding standard of choice.

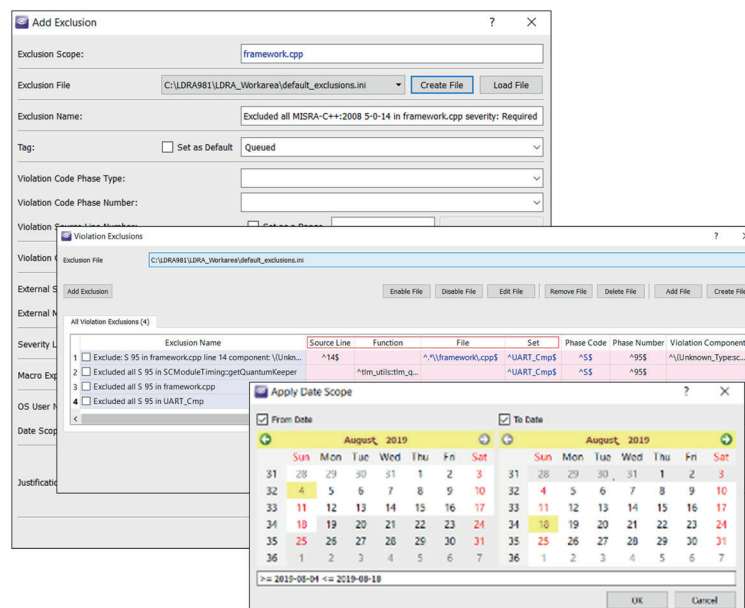


Figure 7: The TBexclude module offers a multi-tier violation exclusion capability

## ISO/SAE 21434 §10.4.2 Product development: integration and verification

ISO/SAE 21434 requirements [RQ-10-09] and [RQ-10-10] are concerned with the integration of the components into subsystems and systems, and with the verification that the result fulfils the cybersecurity specifications. It is perhaps in these sections where the lack of detailed guidance offered by the standard is most telling.

[RQ-10-10] highlights the following methods for verification but provides no details of how they should be approached. Each of them is supported by the LDRA tool suite, and each has associated methods that have been proven in countless other contexts. They offer ready made solutions to address that void for any organization looking to identify and apply best practise.

The ISO/SAE 21434 standard expands no further on the notion of each of the following verification techniques, but those described below are proven in use during the development of countless critical applications.

### Requirements-based test

Requirements-based test involve dynamic testing (execution) of code in whole or in parts to demonstrate that it fulfils the software requirements (and intermediate designs derived from those requirements) but does not include unspecified functionality or redundant code. Structural coverage analysis [23] is required to determine which code structures and component interfaces have not been exercised during execution of these requirements-based test procedures. Unexecuted portions of code require further analysis resulting in the addition or modification of test cases, changes to inadequate requirements, or the removal of dead code, deactivated code, or unspecified functionality.

The standard does not require tools to be used to complete this exercise, but it is likely to be more efficient to do so for all but the most trivial of applications. The LDRA tool suite can be used to perform structural coverage analysis during unit test, integration test, on a complete application system test (Figure 8).

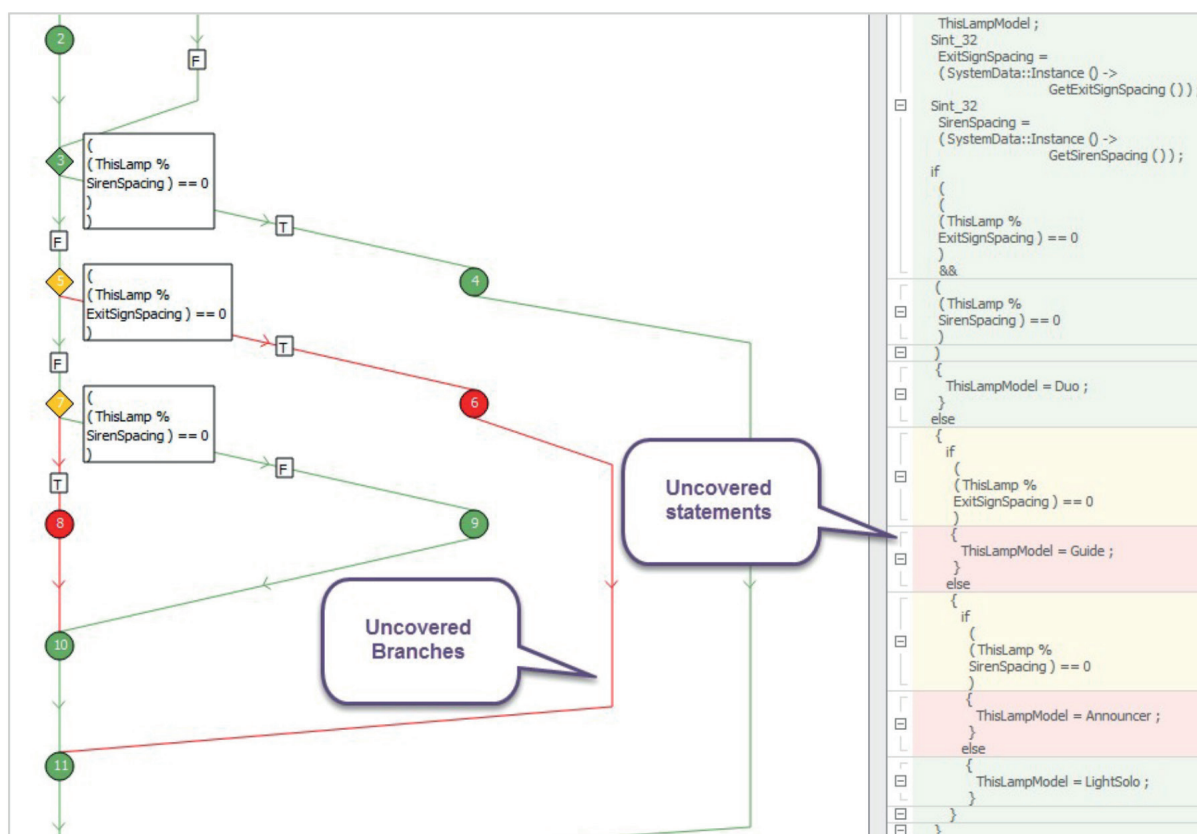


Figure 8: Graphical visualization of code coverage in a flow diagram in the LDRA tool suite

Traceability of tests to requirements is demonstrated using the TBmanager component of the LDRA tool suite (Figure 9).

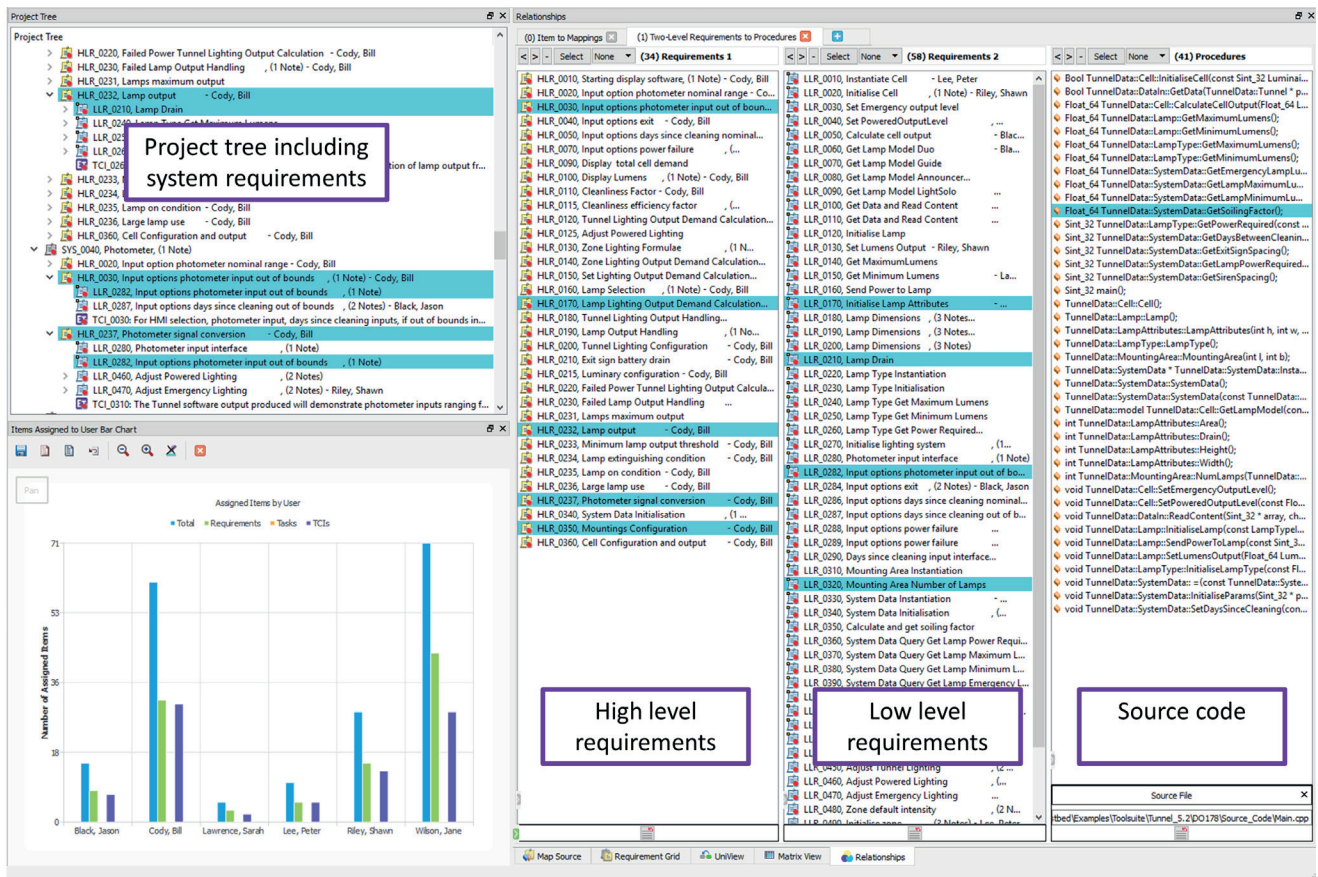


Figure 9: Automated bidirectional requirements traceability helps to ease requirements-based testing

## Interface test

Interface testing is used to verify whether the communication between software systems, subsystems and components works correctly. The TBrn [24] component of the LDRA tool suite provides a platform to demonstrate the correct functionality of those interfaces, potentially in combination with code coverage analysis. Although often described as a unit test tool, TBrn uses the same interface to perform integration testing. This allows for a progressive “bottom up” or “top down” approach to testing interfaces between units.

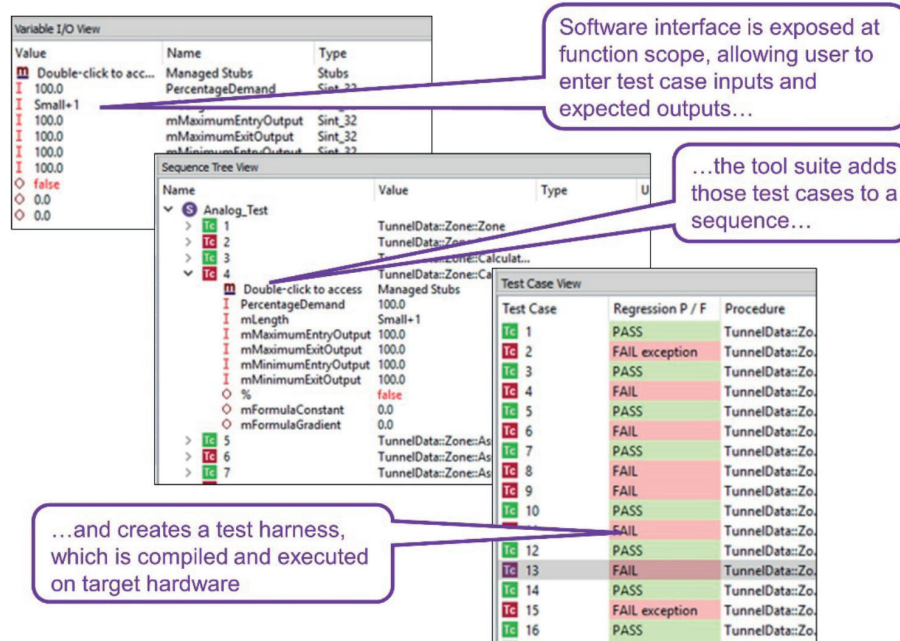


Figure 10: Unit and integration test with the TBrn component of the LDRA tool suite

The TBextreme [25] optional module helps by automating some of the test inputs – for example, to ensure that boundary conditions are handled correctly (Figure 11).

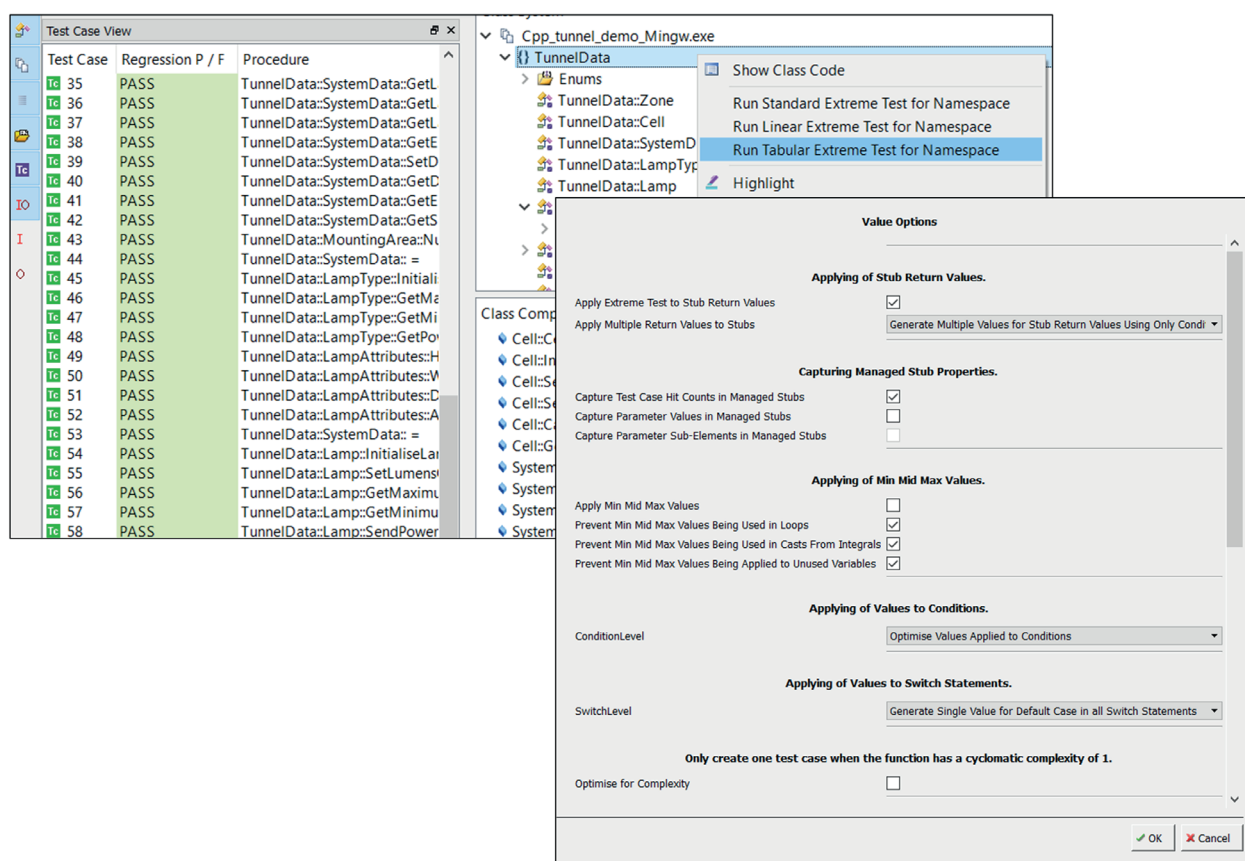


Figure 11: Automating test vector creation with the LDRA tool suite

## Resource usage evaluation

Ensuring the provision of adequate resources (memory, timing, file system ...) and the elimination of contention issues are important considerations for a connected system, particularly when under attack from bad actors. The advent of multicore processors has seen heightened importance of ensuring adequate resources and particularly in the protection of domains from each other, but it has always made ensuring those resources are available more challenging than ever.

For example, consider the challenge represented by the calculation of Worst-Case Execution Time (WCET). According to Reinhard Wilhelm et al, the calculation of a definitive value of WCET by mathematical analysis is not soluble in the general case. A purely static analysis approach will therefore require approximations to be applied which have to be correct, but not necessarily complete [26].

The result is that such tools will necessarily err “on the safe side” which is better than nothing, but in an environment where precision is everything, it cannot be ideal. However, there are long standing and proven mechanisms available to measure the properties of software code which are independent of their execution on the chosen platform. For example, Halstead’s metrics [27] reflect the implementation or expression of algorithms in different languages to evaluate the software module size, software complexity, and the data flow information – and these can be calculated precisely from the static analysis of the source code (Figure 12). Such an approach can identify which sections of code are the most demanding of processing time but cannot provide absolute values for maximum time elapsed.

Halsteads (Cashregister.c)							
File	Total Operators	Total Operands	Unique Operators	Unique Operands	Vocabulary	Length	Volume
Total for Cashregister.c	149	230	16	56	72	379	2338

Figure 12: Halstead’s Metrics calculated using the LDRA tool suite

The same static analysis also yields call diagrams associated with the code base, presenting a means of visualizing where the most demanding functions revealed by this analysis are exercised in the context of the complete code base (Figure 13).

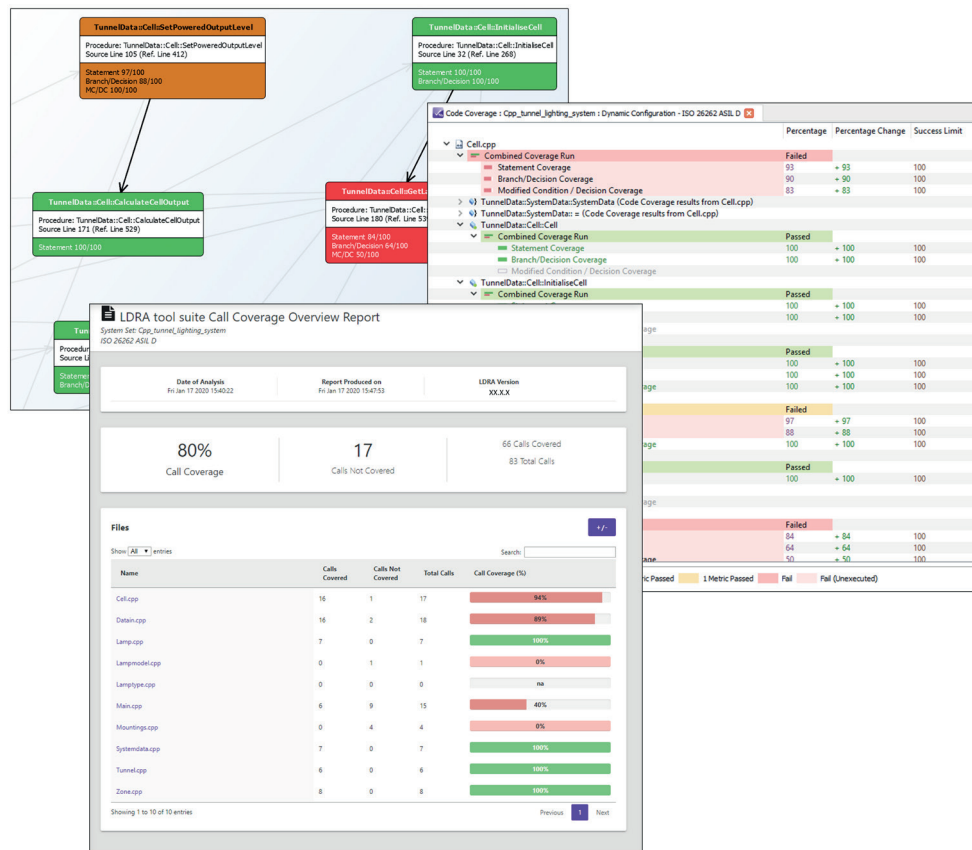


Figure 13: Call diagrams in the LDRA tool suite provide a means to visualize where the most demanding functions are called

The TBrunc component of the LDRA tool suite includes facilities to measure these critical execution times dynamically rather than relying on approximations based on static analysis.

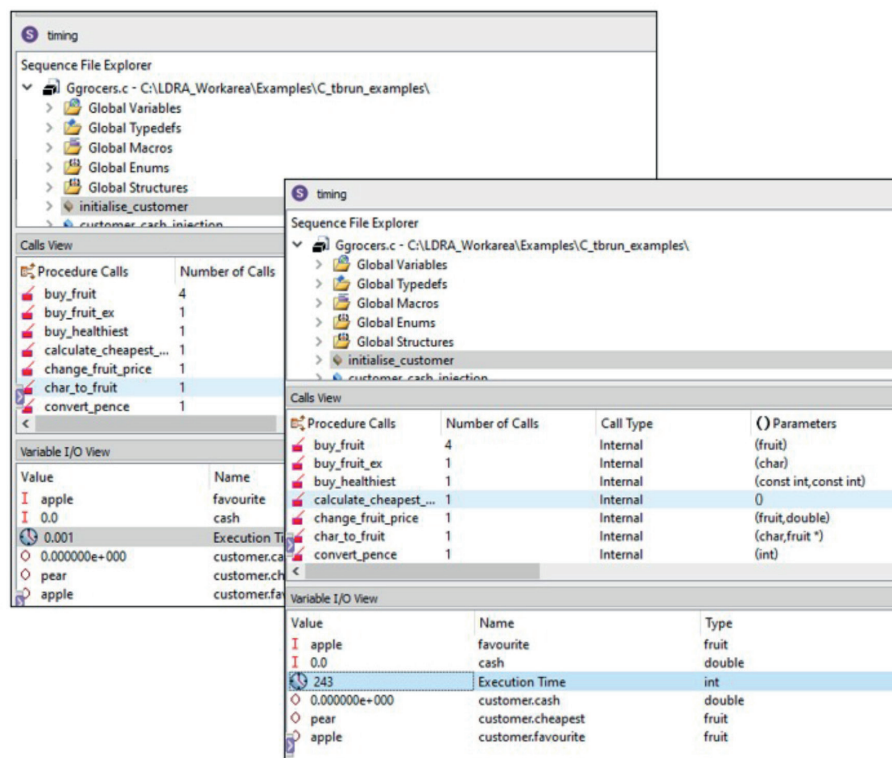


Figure 14: Timing analysis using the TBrunc component of the LDRA tool suite

## Verification of the control flow and data flow

Flawed data or control flow can lead to vulnerabilities in code. Control and data flow analysis provides a means to confirm that both are in accordance with the system design. The LDRA tool suite provides a graphical static and dynamic analysis facilities for both host and embedded software analysis. Control flow is exposed and illustrated in the form of flow diagrams. These can be superimposed with execution paths as shown in Figure 8 earlier.

The Dynamic Data Flow Coverage (DDFC) module [28] supplements that functionality by analysing and reporting on the variables used during run-time.

## Dynamic analysis

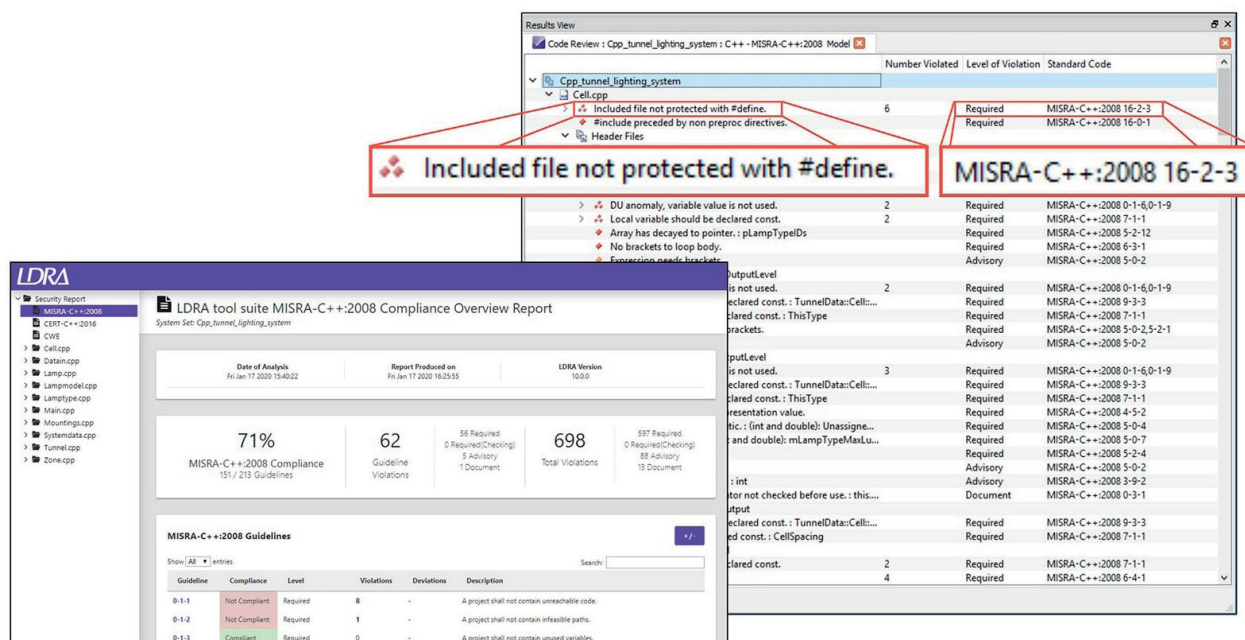
Dynamic analysis is a generic term for software test methods that allow for the validation and verification of software in whole or part by analysing its behaviour at run time, many of which are described in the previous sections. LDRA tool suite components concerned with code (structural) coverage analysis and unit test (TBrn) are examples of dynamic analysis tools.

Dynamic analysis (or DAST) in the context of code during development and integration usually implies “white box” analysis, such that the source code is available and mapped to the execution of the object code.

More traditional “black box” security techniques such as fuzz testing and penetration testing still have a place in the ISO/SAE 21434 development lifecycle, but these are best used after the system is complete to confirm the success of the precautions taken during development.

## Static analysis

Static analysis is a generic term for software test methods that allow for the validation and verification of software by analysing the source code. The LDRA tool suite provides the ability to generate quality metrics (including those relating to code complexity) and to check adherence to the coding standards specified in accordance with ISO/SAE 21434 §10.4.1 (above).



Static analysis tools vary in terms of their ability to identify the more subtle nuances of standard violations, but the more sophisticated implementations can seem slower because of the additional processing required to achieve that. A sensible approach is to choose tools with the option to run in “lightweight” mode initially, and to apply more complete analysis as development progresses.

## Conclusions

---

Automotive embedded applications have traditionally been isolated, static, fixed-function, device-specific implementations, and practices and processes have relied on that status. But the explosion in demand for connectivity in the industry was such that even the publication of the stopgap SAE J3061™ in January 2016 was very welcome.

This allowed time for the development of a more formal standard to address the issue more satisfactorily, and ISO/SAE 21434:2021 was published to replace SAE J3061 in August 2021.

ISO/SAE 21434 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.

The failure of ISO/SAE 21434 to give detailed guidance on how to achieve its objectives means that from a software perspective, the standard does little more than ratify the document it replaces. However, ISO/SAE 21434 presents a worthy set of goals for software developers to achieve. From an optimistic perspective, the lack of detail affords flexibility on how they are achieved. In particular, the analogous nature of severity categories in ISO/SAE 21434 with ASIL values in ISO 26262 suggests a similar approach to varying the level of verification and validation activity in a similar way.

When applied appropriately, established and proven tools such as the LDRA tool suite have a key role to play in supporting the principles outlined in ISO/SAE 21434. They provide a means to define and apply a pragmatic approach to achieving them, despite the failure of the standard to provide that guidance.

## Works Cited

- [1] International Organization for Standardization, ISO 26262:2018 “Road vehicles - Functional safety”, International Organization for Standardization, 2018.
- [2] SAE International, in J3061 “*SURFACE VEHICLE RECOMMENDED PRACTICE - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems*”, SAE International, 2016.
- [3] International Organization for Standardization / SAE International, ISO/SAE 21434:2021 “*Road vehicles — Cybersecurity engineering*”, International Organization for Standardization / SAE International, 2021.
- [4] The UNECE World Forum for Harmonization of Vehicle Regulations, “*WP.29 - Introduction*,” United Nations Economic Commission for Europe (UNECE), [Online]. Available: <https://unece.org/wp29-introduction>. [Accessed 23 March 2022].
- [5] United Nations, UN Regulation No. 155 - Uniform provisions concerning the approval of vehicles with regards to cyber security and cyber security management system, Geneva: United Nations, 2021.
- [6] International Organization for Standardization, ISO 26262-6:2018 “*Road vehicles - Functional Safety - Product development at the software level*”, International Organization for Standardization, 2018.
- [7] LDRA, “*LDRA tool suite*,” LDRA, [Online]. Available: <https://ldra.com/products/ldra-tool-suite/>. [Accessed 23 March 2022].
- [8] R. Seacord, “*Top 10 Secure Coding Practices*,” Carnegie Mellon University Software Engineering Institute, 2 May 2018. [Online]. Available: <https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices>. [Accessed 23 March 2022].
- [9] Microsoft, “*The STRIDE Threat Model*,” Microsoft, 15 December 2021. [Online]. Available: <https://docs.microsoft.com/en-us/windows-hardware/drivers/driversecurity/threat-modeling-for-drivers#the-stride-approach-to-threat-categorization>. [Accessed 23 March 2022].
- [10] Microsoft, “*The DREAD approach to threat assessment*,” 5 December 2021. [Online]. Available: <https://docs.microsoft.com/en-us/windows-hardware/drivers/driversecurity/threat-modeling-for-drivers#the-dread-approach-to-threat-assessment>. [Accessed 23 March 2022].
- [11] Forum of Incident Response and Security Teams, Inc, “*Common Vulnerability Scoring System*,” 2015-2021. [Online]. Available: <https://www.first.org/cvss/>. [Accessed 23 March 2022].
- [12] The Mitre Corporation, “*Common Weakness Scoring System (CWSS)*,” 5 September 2014. [Online]. Available: [https://cwe.mitre.org/cwss/cwss\\_v1.0.1.html](https://cwe.mitre.org/cwss/cwss_v1.0.1.html). [Accessed 23 March 2022].
- [13] The Mitre Organization, “*CVE Program Mission*,” 1999-2021. [Online]. Available: <https://www.cve.org/>. [Accessed 23 March 2022].
- [14] Homeland Security Systems Engineering and Development Institute, “*CWE Common Weakness Enumeration*,” The MITRE corporation, August 2021. [Online]. Available: [https://cwe.mitre.org/cwss/cwss\\_v1.0.1.html](https://cwe.mitre.org/cwss/cwss_v1.0.1.html)
- [15] The Mitre Organization, “*Common Attack Pattern Enumerations and Classifications (CAPEC)*,” The Mitre Organization, 2 March 2022. [Online]. Available: <https://capec.mitre.org/>. [Accessed 23 March 2022].
- [16] AUTOSAR, “*AUTOSAR (AUTomotive Open System ARchitecture)*,” AUTOSAR, 2022. [Online]. Available: <https://www.autosar.org/>. [Accessed 23 March 2022].
- [17] S. Neemeh, “*What is ASPICE in Automotive?*,” LHP, 9 June 2020. [Online]. Available: <https://www.lhpes.com/blog/what-is-aspice-in-automotive>. [Accessed 13 July 2021].
- [18] M. Pitchford, “*AUTOSAR, ISO 26262, SAE J3061: So many rules, so little time!*,” in *Embedded World*, Nuremberg, 2020.
- [19] MISRA, “*MISRA C:2012 Third Edition, First Revision*,” The MISRA Consortium Limited, February 2019. [Online]. Available: <https://www.misra.org.uk/product/misra-c2012-third-edition-first-revision/>. [Accessed 24 September 2021].
- [20] Carnegie Mellon University Software Engineering Institute, “*SEI CERT C Coding Standard*,” Carnegie Mellon University Software Engineering Institute, 5 December 2018. [Online]. Available:

- [21] LDRA, “LDRA Testbed® and TBvision®,” LDRA, [Online]. Available: <https://ldra.com/products/ldra-testbed-tbvision/>. [Accessed 23 March 2022].
- [22] LDRA, “TBexclude®,” LDRA, [Online]. Available: <https://ldra.com/products/tbexclude/>. [Accessed 23 March 2022].
- [23] LDRA, “Code Coverage analysis,” LDRA, [Online]. Available: <https://ldra.com/capabilities/code-coverage-analysis/>. [Accessed 23 March 2022].
- [24] LDRA, “TBrun®,” LDRA, [Online]. Available: <https://ldra.com/products/tbrun/>. [Accessed 23 March 2022].
- [25] LDRA, “TBextreme®,” [Online]. Available: <https://ldra.com/products/tbextreme/>. [Accessed 23 March 2022].
- [26] R. W. e. al., “The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools,” April 2008. [Online]. Available: [http://www.es.mdh.se/pdf\\_publications/1258.pdf](http://www.es.mdh.se/pdf_publications/1258.pdf). [Accessed 1 November 2021].
- [27] Halstead, Maurice H., Elements of Software Science., Amsterdam: Elsevier North-Holland, 1977.
- [28] LDRA, “Dynamic Data Flow Coverage (DDFC),” LDRA, [Online]. Available: <https://ldra.com/products/dynamic-data-flow-coverage-ddfc/>. [Accessed 23 March 2022].



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

**LDRA Technology Inc.**  
 2540 King Arthur Blvd, 3rd Floor, 12th Main, Lewisville, Texas 75056  
 Tel: +1 (855) 855 5372  
 e-mail: [info@ldra.com](mailto:info@ldra.com)

**LDRA Technology Pvt. Ltd.**  
 Unit B-3, Third floor Tower B, Golden Enclave  
 HAL Airport Road Bengaluru 560017  
 Tel: +91 80 4080 8707  
 e-mail: [india@ldra.com](mailto:india@ldra.com)