# PARASOFT.

# ISO 26262 Software Compliance in the Automotive Industry

## **Table of Contents**

#### **3** Overview

- 3 Automotive Industry Outlook
- 7 What Is ISO 26262?

#### **14 Requirements for Compliance in Testing**

- 14 Static Analysis
- 18 MISRA C 2023
- 20 AUTOSAR C++14
- 24 SEI CERT
- 28 CWE Common Weakness Enumeration
- 32 Unit Testing
- 38 Regression Testing
- 41 Software Integration Testing
- 48 Software System Testing
- 51 Structural Code Coverage
- 54 Requirements and the Traceability Matrix

## 59 A Unified, Fully Integated Testing Solution for C/C++ Software Development

- 59 Tool Qualification for Safety-Critical Automotive Systems
- 65 Reporting and Analytics for Automotive Software

#### 71 More Resources

71 Safety-Critical Automotive Software Development Assets for Download

## **Overview**

## **Automotive Industry Outlook**

The automotive industry continues to rapidly evolve and grow into technical areas where other industries have operated for many years. For example, NASA's Jet Propulsion Laboratory releases code fixes and new functionality currently in development for a spacecraft millions of miles away, en route to its destination. Similarly, we find the automotive industry providing software updates on cars that have been sold and are being driven by their consumers all around the world.

The future of self-driving cars also looks promising, with potential for widespread adoption in the next few decades. Several companies, including Waymo, Tesla, Uber, and traditional car manufacturers like GM and Ford, are at the forefront of developing self-driving technology. Many are conducting extensive testing, and some have deployed pilot programs in select cities. As the technology matures, it is expected to revolutionize transportation, making it safer, more efficient, and accessible.

#### Safety & Security Challenges

This type of evolution—particularly that of advanced driver-assistance systems (ADAS) comes with a new set of challenges in safety and security. Standards like ISO 26262 address functional safety of the development of electric and electronic systems (E/E), which include propulsion, dynamic control systems, and driver assistance.

Additionally, platforms like AUTOSAR provide an open standardized software layer architecture that further improves safety. They include guidelines for the use of the C++14 language in development of critical and safety-related systems. However, manufacturers have realized that due to the increased complexity and unknowns of modern technologies working together, along with changes in the internal and external environment, safety and security concerns have arisen that these standards don't address.

When addressing ISO 21434, it's important to understand that the recommended security consideration for cybersecurity should be integrated into your existing development processes. ISO 21434 references ISO 26262 in consideration of having these two disciplines take an interdisciplinary exchange of strategies, coordination and even tools used. This means that your organization should have your system engineers work with your security engineers through the requirements analysis phase for safety and security.



parallel, perform hazard analysis and risk assessment (HARA) for safety, and threat analysis and risk assessment (TARA) for security. Nonetheless, a strong collaborative environment is needed to ensure a safe and secure result.

Ensuring security at the software implementation phase starts by applying static code analysis. The MISRA coding standard incorporates security guidelines, but you can also augment and strengthen code security by adopting CERT.

Continuing up the right side of the V, perform unit testing of all your low-level security requirements. In the next phase, create test cases that incorporate additional functionality. These test cases ensure that your high-level requirements are satisfied.

Moving to system testing, create system tests to ensure that the system requirements are verified. Confirm that all the test cases trace back to your requirements. This guarantees that no requirement goes untested. However, to safeguard that each requirement is fully tested, incorporate structural code coverage as recommended by ISO 21434 and ISO 26262. Code coverage ensures that your security test cases fully cover every possible path of execution through its security functionality remediation measures.

To overcome safety and security challenges, teams can turn to solutions like Parasoft C/C++test, which has been certified for use in safety-critical applications per ISO 26262 and is TÜV SÜD certified to satisfy ISO 21434. Both of these standards recommend performing static analysis, dynamic analysis—which includes unit, integration, and system testing—code coverage, and requirements traceability. Offering exactly what ISO 26262 and ISO 21434 recommend for software verification in safety and security, Parasoft also provides the documentation required to prove compliance with both standards.

## **UNECE WP.29 Regulatory Requirements**

The United Nations Economic Commission for Europe (UNECE) released regulatory requirements on June 23, 2020, where they outlined new processes and technologies that automotive manufacturers must incorporate into both their organization and vehicles. These regulations also apply to Tier 1 and Tier 2 suppliers of software and hardware components, including mobile services.

Vehicle manufacturers are required to put into the organizational structure a risk-based management framework for discovering, analyzing, and protecting against relevant threats, vulnerabilities, and cyberattacks.

The following categories require cybersecurity testing and passing inspections.

- » Category M covers standard four wheel cars.
- » Category N is for pickup trucks and vans.
- » Categories L6 and L7 include electric cars and autonomous capabilities.

A passing grade on both organizational and vehicle WP.29 key requirements means that the manufacturer receives a certificate of compliance. New vehicles without this certificate cannot be sold in the EU after July 2024. Be aware that the United States does not participate or have its own similar regulations. However, the writing is on the wall.

#### **Automotive SPICE**

Automotive Software Process Improvement and Capability Determination (ASPICE) provides a measurement framework for independent assessors to evaluate an organization's capability for software development. Ensuring software safety and cybersecurity does not only lie within the technical engineering aspects of the development of the electronic system, but also requires the organization to incorporate processes and checks.

These processes and checks must include ways to track and monitor progress within all practices of the organization to ensure:

- 1. Safety and cybersecurity practices have been adopted.
- 2. Safety and cybersecurity requirements are being satisfied.

This is also one of the two key certification criteria for UNECE WP.29 on organizational cybersecurity capability.

#### **Unsafe Scenarios**

It's brought to fruition other outgrowths from ISO 26262, like ISO/PAS 21448 more commonly referred to as SOTIF (safety of the intended functionality). SOTIF helps you analyze and prevent the misuse of the intended functionality where it creates an unsafe scenario. For example, your vehicle inadvertently shuts down while you're driving it, due to an initiated software update.

Security vulnerabilities also pose unsafe scenarios. An attacker could use the car's Wi-Fi connection to remotely exploit an exposed port. They could somehow work their way from the advanced in-vehicle infotainment (IVI) into taking control of, or influencing, safety-critical components like braking or steering due to sharing the same communications infrastructure.

#### The Role of Standards

Standards like SAE J3061, superseded by ISO/SAE 21434, specify that an initial Threat Analysis and Risk Assessment (TARA) be completed to assess potential threats related to operation, privacy, and other factors where a road user/driver can be impacted. If the risk for any threat is sufficiently high, then a cybersecurity process is necessary. There are various approaches to flushing out security vulnerabilities and requirements that mitigate the risks. Learn more about TARA and why your development team needs TARA.

Standards like UL 4600 now exist specifically for fully autonomous vehicle operation. This means that there is no human supervision, and the autonomy assumes full responsibility. This standard focuses on building a safety case for the deployment of SAE Level 4/5 vehicles, not on how to test safety of autonomous vehicles on public roads. That would involve a different standard.

These standards and others play a crucial role in safety and security for the automotive industry. OEMs carry the liability costs for delivering unsafe and insecure vehicles to the masses. To mitigate these risks, OEMs need to adopt and adhere to these standards. However, OEMs should mandate the same quality and adherence by their suppliers. A weakness in one component can undermine the safety and security of the entire system.

#### **Building Custom Coding Standards**

Working with some of its automotive OEMs, Parasoft has built custom coding standards that incorporate MISRA, AUTOSAR C++14, CERT, CWE, and other custom rules to be used by their suppliers. This ensures that the same level of quality software exists across the entire supply chain.

Parasoft C/C++test is a unified testing solution that includes unit testing and structural code coverage as part of its functionality. This solution for C/C++ software development supports a comprehensive set of hardware targets and development ecosystems that suppliers and OEMs can use with varying development infrastructures. Parasoft C/C++test has been certified by TÜV SÜD for use on safety- and security-critical systems. For ADAS and secure connected cards, C/C++test's seamless integration with Parasoft SOAtest and Parasoft Virtualize combines API testing with runtime application coverage and simulated virtual test beds.

## What Is ISO 26262?

ISO 26262 is a functional safety standard that covers the entire automotive product development process. It includes activities such as requirements specification, design, implementation, integration, verification, validation, and configuration.

The standard provides guidance on automotive safety lifecycle activities by specifying the following requirements:

- » Functional safety management for automotive applications
- » The concept phase for automotive applications
- » Product development at the system level for automotive applications software architectural design
- » Product development at the hardware level for automotive applications software unit testing
- » Product development at the software level for automotive applications
- » Production, operation, service, and decommissioning
- Supporting processes: interfaces within distributed developments, safety management requirements, change and configuration management, verification, documentation, use of software tools, qualification of software components, qualification of hardware components, and proven-in-use argument
- » Automotive Safety Integrity Level (ASIL) oriented and safety-oriented analyses

ISO 26262 is an adaptation of IEC 61508 for the automotive industry. IEC 61508 is a basic functional industrial safety standard for electrical, electronic, and programmable electronic devices, and applicable to all kinds of industries. Other sectors like Medical IEC 62304 and Railway EN 50128/EN 50716 have also been derived from IEC 61508.

Since ISO 26262 has been extracted and expanded from IEC 61508 for the automotive industry, by inheritance it is a functional safety standard that provides guidance for regulating the entire product lifecycle process, at the software and hardware level from conceptual development through to decommissioning. It covers electrical and electronic automotive systems and their development process, including requirements specification, design, implementation, integration, verification, validation, and configuration.

The latest release, ISO 26262:2018 is subdivided into 12 parts. The standard has been evolving since its first edition, released back in 2011.

## What Are the Parts of ISO 26262



Figure 2-1: Overview of ISO 26262 **Part 1** is the vocabulary section for the standard. Terms, definitions, and abbreviations are found here.

**Part 2** is the management of functional safety, which defines an internal functional safety process for the team or company. This includes having a safety organization that oversees the planning, coordinating and documentation activities related to functional safety.

Functional safety is of the utmost importance in the development of safety-critical automotive systems because people's lives depend on it. Especially now with the introduction of driver assist and automated driving systems. The management of security could be adapted to part 2. Security is crucial in the world we live in today.

**Part 3** is the concept phase that takes in the stakeholder requirements and drives what you are going to build and ultimately deliver. In figure 2-1, notice on the right side of the concept phase box the beginning of a gray-shaded V watermark. The shaded Vs represent the interconnection among parts 3, 4, 5, 6, and 7 of the standard. These part series are based upon the V-model software development lifecycle. You have the different phases of development represented on the left and the verification and validation or testing phases on the right. If you are a systems or software engineer in the embedded industry, the V-model is well known.

**Part 4** is the beginning of product development at the system level, which includes parts 5 and 6 but looking at these from a high level of abstraction. The architecture is defined, including functional test cases that verify and validate the architecture. To dive in deeper into the detail design and implementation, part 5 and part 6 are defined.

Part 5 targets development of hardware, which is out of scope for this document.

**Part 6** targets software development. You can see a smaller lighter grey V watermark for software development and again the left-hand side of the V encapsulates the requirements decomposition, design, and implementation phases but now a much lower level of granularity. On the right-hand side of the V, sections 6.9, 6.10, and 6.11 represent the testing or verification and validation of the software. This includes unit testing, static analysis, structural code coverage, requirements traceability and more.

It also includes requirements for the software development of automotive applications. This includes obligations for initiation of product development, specification of software safety requirements, software architectural design, software unit design and implementation. On the verification and validation of the software component, you have multiple methods recommended or mandated based on the assigned safety integrity level (ASIL).

**Part 7** addresses the production and operation of the product, once it's out in the field. This means you must consider things like maintenance and decommissioning or sunsetting of your product.

**Part 8** specifies the various supporting processes and solutions needed in the development of the system that help ensure functional safety. This includes having a configuration management solution, a change management, a documentation management, and other solutions in place.

Another important aspect of Part 8 is the qualification of the software tools being used. You don't want to use an open source tool or an uncertified tool from a vendor that undermines the safety or security of your product by introducing issues. Use a tool that has been certified by the Technical Inspection Association (TÜV) and has a proven inuse history or argument. **Part 9** is a critical section to understand because it pertains to assigning a risk classification on the system under development. This means that you have to take into consideration the risk to the passengers or pedestrians if the electrical or electronic system in development were to malfunction or fail.

A hazard analysis and risk assessment need to be performed. ISO 26262 is a risk-based safety standard, where the risk of hazardous operational situations is assessed, and safety measures are defined to detect and to avoid or control failures, so mitigating actions can take effect.

**Part 10** basically provides an overview of the ISO 26262 standard with additional explanations that enhance the understanding and concepts of the other parts in the standard, so it's informative.

**Part 11** is the adaptation of functional safety guidelines to semiconductors for automotive. It offers guidance and information to semiconductor manufacturers on how to develop ISO 26262 compliant IP. It helps incorporate functional safety because users of semiconductors may not know how to use the semiconductor safely. This came about because automotive systems have become very complex and semiconductors have enabled most of the recent innovations. That includes vision-based technology, enhanced graphics processing units (GPUs), application processors, sensors, DRAM, and other components that empower advanced driver-assistance systems or ADAS.

**Part 12** is the adaptation of the standard for motorcycles, which has been intentionally left out of Figure 2-1 and this ebook.

#### Performing Hazard Analysis and Risk Assessment

In ISO 26262, a hazard analysis and risk assessment (HARA) needs to be performed on the system under development. Upon completion of the HARA an ASIL is assigned to the software component and there are levels A through D. Level A represents the lowest hazard assignment and Level D represents the highest hazard assignment. Meaning that the failure of a system with ASIL D assignment could be catastrophic.

There is also a quality management (QM) level assignment, which means that there is no safety requirement. ASIL is assigned by taking the severity of the injury times the probability of the failure times the controllability. The following table spells out each level for severity, exposure, and controllability.

There are several tables freely made available that provide help in determining the ASIL value. The table below is an example of one that's much easier to read and shows the ASIL levels in colors based on severity, exposure, and controllability.

SEVERITY	EXPOSURE	CONTROLLABILITY
S0 No Injuries	E0 Incredibly unlikely	C0 Controllable in general
<b>S1</b> Light to moderate injuries	E1 Very low probability (injury could happen only in rare operating conditions)	C1 Simply controllable
S2 Severe to life-threatening (survival probable) injuries	E2 Low probability	C2 Normally controllable (most drivers could act to prevent injury)
<b>S3</b> Life-threatening (survival uncertain) to fatal injuries	E3 Medium probability	C3 Difficult to control or uncontrollable
	E4 High probability (injury could happen under most operating conditions)	

Figure 2-2: Hazard Analysis and Risk Assessment Severity = What would be the impact or damage if the failure occurred? Exposure = The frequency or probability that the failure would occur.

Controllability = The extent to which we can ensure that the event doesn't happen.

	EXPOSUPE		SEVE	RITY	
CONTROLLABILITY	EXPOSORE	S0	S1	S2	<b>S</b> 3
	E1	QM	QM	QM	QM
C1	E2	QM	QM	QM	QM
CI	E3	QM	QM	QM	А
	E4	QM	QM	Α	В
	E1	QM	QM	QM	QM
C2	E2	QM	QM	QM	Α
02	E3	QM	QM	А	В
	E4	QM	Α	В	С
	E1	QM	QM	QM	Α
63	E2	QM	QM	А	В
C3	E3	QM	А	В	С
	E4	ОМ	В	с	D

Figure 2-3: Simplified ASIL assesment table

#### **Active and Passive Safety**

Roadside vehicles come with lots of safety systems and some are considered active safety and others passive safety.

Active safety is used to refer to technology assisting in the prevention of a crash or accident. You have your traction control, anti-lock braking system, vision ADAS, and others.

**Passive safety** systems are to keep the passengers safe. For example, in case of a crash, you have airbags, and seatbelts. The electronic windshield wiper and instrument cluster are also passive safety systems.



# Performing Test Verification & Validation of Software Unit Design and Implementation

Since the focus of this ebook is software, it's important to cover the test verification and validation methods recommended by the standard. For example, Table 9 captures verification methods 1a through 1h to be applied during unit design and implementation. Method 1f, "Static code analysis" is recommended for ASIL level A and highly recommended for ASIL levels B through D.

The columns in Table 7 below show A to D ASIL levels. A single "+" symbol indicates recommended by the standard, a double "++" indicates highly recommended, and an "o" indicates no recommendation.

#### Table 7 - Methods for software unification

	Mathada		AS	SIL	
	Methods	Α	B	С	D
1a	Walk-through <sup>a</sup>	++	+	0	0
1b	Pair-programming <sup>a</sup>	+	+	+	+
1c	Inspectiona	+	++	++	++
1d	Semi-formal verification	+	+	++	++
1e	Formal verification	0	0	+	+
1f	Control flow analysis <sup>b, c</sup>	+	+	++	++
1g	Data flow analysis <sup>b, c</sup>	+	+	++	++
1h	Static code analysis <sup>d</sup>	++	++	++	++
1i	Static analyses based on abstract interpretation <sup>e</sup>	+	+	+	+
1j	Requirements-based test <sup>f</sup>	++	++	++	++
1k	Interface testg	++	++	++	++

Figure 2-5: ISO 26262 Part 6, 9.4.2:2018

Figure 2-4:

Safety

Active and Passive

12

Other key methods of verification are done through dynamic analysis, for requirementsbased testing and fault injection. Table 11 for example has "Analysis of boundary values". This is a method for deriving test case to flush out defects by means of proving inputs into the unit that are not just the min, mid, and max, but the boundaries outside the scope of its range, to see if the unit is robust enough to handle these outlier cases.

#### Table 8 - Methods for deriving test cases for software unit testing

	Methods		ASIL				
			AB				
1a	Analysis of requirements	++	++	++	++		
1b	Generation and analysis of equivalence classes <sup>a</sup>	+	++	++	++		
1c	Analysis of boundary values <sup>b</sup>	+	++	++	++		
1d	Error guessing based on knowledge or experience <sup>c</sup>	+	+	+	+		
a can	Equivalence classes can be identified based on the division of inputs and outputs, su be selected for each class.	ch that a re	present	ative te	st valu		
b	This method applies to interfaces, values approaching and crossing the boundaries a	nd out of ra	ange val	ues.			
с	Error guessing tests can be based on data collected through a "lessons learned" proc	ess and exi	pert jude	ment.			

Figure 2-6: ISO 26262 Part 6, 9.4.3:2018

And Table 9 lists the recommended structural code coverage metrics to ensure test coverage, flush out dead code, and hidden defects.

#### Table 9 - Structural coverage metrics at the software unit level

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

Figure 2-7: ISO 26262 Part 6, 9.4.4:2018

## **Requirements for Compliance in Testing**

## **Static Analysis**

Many of the quality tasks specified in ISO 26262, including data and control flow analysis and semantic analysis are supported by modern advanced tools like Parasoft C/C++test. In addition, static analysis tools include metrics and support peer code review with capabilities that assist unit testing and runtime error detection.

## The Role of Static Analysis in ISO 26262 Software Verification

Verification methods like static analysis provide teams with a practical way to expose, prevent, and correct errors in automotive software systems. The real power of advanced static analysis tools comes from the ability to analyze the code based on industry coding compliance standards like MISRA C/C++, CERT C/C++, and AUTOSAR C++14.

The analysis reports code rule and directive violations, along with code complexity and quality metrics. This data can be source-controlled for historical and auditing purposes. Equally important is the use of a defect tracking and managing system to provide meaningful analytical views and prioritization with the intent of solving the highest risk issues down to the lowest.

	Methods Walk-througha Pair-programminga Inspectiona Semi-formal verification Formal verification Control flow analysis <sup>b, c</sup> Data flow analysis <sup>b, c</sup> Static code analysis <sup>d</sup> Static code analysis <sup>d</sup> Static analyses based on abstract interpretation <sup>e</sup> Requirements-based test <sup>f</sup>		ASIL					
	Methods	Α	B	C	D			
1a	Walk-through <sup>a</sup>	++	+	0	0			
1b	Pair-programming <sup>a</sup>	+	+	+	+			
1c	Inspectiona	+	++	++	++			
1d	Semi-formal verification	+	+	++	++			
1e	Formal verification	0	0	+	+			
1f	Control flow analysis <sup>b, c</sup>	+	+	++	++			
1g	Data flow analysis <sup>b, c</sup>	+	+	++	++			
1h	Static code analysis <sup>d</sup>	++	++	++	++			
1i	Static analyses based on abstract interpretation <sup>e</sup>	+	+	+	+			
1j	Requirements-based test <sup>f</sup>	++	++	++	++			
1k	Interface testg	++	++	++	++			

#### Table 7 - Methods for software unit verification

Figure 3-1: ISO 26262 Part 6, 9.4.2:2018 The specific sections of ISO 26262, part 6: Product development: software level that is addressed by static analysis tools are described below.

#### Walkthroughs and Inspections

Informal methods used to verify design and implementation. Static analysis tools automate much of the tedious aspects of code inspection such as coding standards compliance while flagging errors and possible software weaknesses.

#### **Control Flow Analysis**

A static code analysis technique for determining the control flow of a program. Modern advanced static analysis tools, such Parasoft C/C++test, use sophisticated control and data flow analysis to detect complex defects and security vulnerabilities.

#### **Data Flow Analysis**

A technique for gathering information about the possible set of values calculated at various points in a computer program. Data flow analysis is a critical aspect of advanced static analysis tools that helps detect complex errors such as tainted data vulnerabilities.

#### **Static Code Analysis**

The general term used to describe the analysis of code that is performed without actual code execution. This includes the terms used above.

## The Role of Static Analysis Tools in Support of ISO 26262 Design Principles for Software Unit Design and Implementation

Coding standards embody the best practices learned from years of experience and aim to harden code by avoiding bad practices that result in inadequate quality and security while promoting good practices that create more resilient code. In the case of automotive standards, they are based on best practices plus guidance on preventing the types of software failures that have been observed over the years.

Coding standards usually define a subset of a programming language deemed safer and more secure to use. The aim of this is to prevent unpredictable behavior in the first place, limiting the risky language features that make them possible.

The only practical, objective, and sustainable way to enforce coding standards is with static code analysis tools, which can automatically analyze enormous amounts of source code at a time. These tools integrate into software builds in a CI/CD pipeline and are available directly in a developer's IDE. And they provide reports indicating the conformance of analyzed software to the standard selected.

## **Different Types of Static Code Analysis**

One of the most common types of static code analysis is SAST or static application security testing. This is also considered a best practice for application security testing but can be applied elsewhere. To identify all error classes, multiple coding standards (MISRA, AUTOSAR, CERT, CWE) may need to be used.

As such, it's best to familiarize yourself with various types of static code analysis and the errors they're meant to detect.

#### Performance

These tests identify errors that will reduce overall performance. They can also be used to ensure that developers stay up-to-date with current best practices.

#### **Security**

Certainly a critical test, security-related source code analysis finds security risks like weak cryptography, configuration problems, and framework-specific command injection errors.

#### Reliability

These tests help prevent issues with functionality. No developer wants to deal with an emergency unresponsive service message at 4 a.m. This type of static code analysis is useful for finding memory leaks or threading problems.

#### **Style**

This type of static analysis encourages teams to adopt uniform coding styles for ease of use, understanding, and bug fixing. Since it identifies style violations, developers don't have to waste time looking for them

## What Errors Can Static Code Analysis Detect?

Each static analysis rule or guideline tackles different issues. Some issues that affect reliability might be resource leaks for C or null pointer exceptions in C++. MISRA C:2023 Directive 4.12 exists to prevent the use of dynamic memory that can lead to out-of-storage run-time failures, which is undesirable.

The guideline states, "The identifiers 'calloc', 'malloc', 'realloc', 'aligned\_alloc' and 'free' shall not be used and no macro with one of these names shall be expanded."

Therefore, the following code will produce a violation.

```
int* pl = (int*)malloc(10); /* Violation */
free(pl); /* Violation */
```

The recommended solution is to pre-allocate a block of memory and manage it as needed via your defined equivalent of "malloc" and "free". Similarly in C++, the common solution is to overload the "new" and "delete" operators.

The intent of the software, the language, and the platform all affect the kinds of errors static code analysis can detect.

#### **Static Code Analysis Deviations**

Static code analysis identifies errors based on given rulesets. That means that, if any line defies a rule, it will be flagged. Of course, as in real life, there are some exceptions to these rules across different software types.

In situations like these, developers allow for deviations. The rules can adjust to the circumstances and allow for special issues. A team can decide yes or no whether or not that deviation is acceptable. This also gets documented as it violates the original rules.

#### How to Choose a Static Code Analysis Tool

Parasoft's suite of tools to automate software testing works across diverse workflows and team compositions. When it comes to static code analysis, that rings just as true. It speeds up the development cycle, reduces defect rates, and provides continuous improvement. Identifying which tool might work best for your needs starts simply with the base language of the source code. In addition to C and C++ solutions, Parasoft also provides solutions for Java testing with Jtest as well as testing C# and VB.NET languages with dotTEST.

Static code analysis can be performed either in the IDE (Eclipse, VS Code, Visual Studio) or using the command-line interface for automation and continuous integration pipelines. The results of the analysis can be accessed immediately within the IDE and from generated reports (HTML, PDF, XML), as well as aggregated for further post-processing, reporting, and analytics from the award-winning Parasoft DTP reporting and analytics dashboard.

## **MISRA C 2023**

MISRA C is a set of coding guidelines for the C programming language. The focus of the standard is increasing the safety of software by pre-emptively preventing programmers from making coding mistakes that can lead to runtime failures (and possible safety concerns) by avoiding known problem constructs in the C language.

Over the years, many developers of embedded systems were (and still are) complaining that MISRA C was too stringent of a standard and that the cost of writing fully compliant code was difficult to justify. Realistically, given that MISRA C is applied in safety-critical software, the value of applying the standard to a project depends on factors such as:

- » Risk of a system malfunction because of a software failure
- » Cost of a system failure to the business
- » Development tools and target platform
- » Level of developer's expertise

Programmers must find a practical middle ground that satisfies the spirit of the standard and still claim MISRA compliance without wasting effort on non-value added activities.

#### **MISRA C Compliance**

In the document, "MISRA Compliance:2020," the MISRA Consortium provides the response needed by the community with a well-defined framework of what the statement, "MISRA Compliant," truly means.

The document helps organizations use a common language articulating the compliance requirements by defining the following artifacts:

#### **The Guideline Enforcement Plan**

Demonstrates how each MISRA guideline is verified.

#### The Guideline Re-Categorization Plan

Communicates the agreed-upon severity of individual rules in the guidelines as part of the vendor/client relationship.

#### **The Deviations Report**

Documents the violations of guidelines with appropriate rationale.

#### The Guidelines Compliance Summary

This is the primary record of overall project compliance.

When first introducing MISRA C into a project, commonly where code already exists, the key document is the guideline re-categorization plan. This document captures all directives, rules, and identifies which categories have been re-categorized. However, it's important to have the same rational categorization for newly developed code as well. For example, the following diagram shows part of a re-categorization plan.

ARASOFT.				¢- 0-	admir
	Ν	MISRA Compliance	Report	🛓 Downie	load P
Filter	atm-for-misra-c Target Build: misra-c-2023 Compliance Pr	ofile: MISRA C 2023 Analysis Tool: Parase	oft C/C++test 2023.1 Compiler: ADI	D COMPILER Revision Date: 2023-04-03	
	(This report is compliant wit	h: *MISRA Compliance: 2020 Achieving co	empliance with MISRA Coding Gui	delines*)	
		Project Compliance: 3 Not C	ompliant		
	Cuideling Enforcement Plan	idelies Do estacoization Dies	aviation Depart (Total: 6)	Duild Aude Depend	
	Guideline Enforcement Plan Gi	udeline Re-categorization Plan	eviation Report (Total: 6)	Build Audit Report	
	Category: A	II ~ Compliance: All	×		
Guideline	MISRA Category	Compliance	# of Violations +	# of Deviations	
Dir 4.6	Advisory	A Compliant with Violations	6 -	2	
Rule 14.4	Required	O Not Compliant	2	0	
Rule 2.2	Required	O Not Compliant	1	0	
Rule 13.4	Advisory	Compliant with Violations	1	0	
Rule 14.3	Required	O Not Compliant	1	0	
Rule 15.6	Required	O Not Compliant	1	0	
Dir 4.1	Required	Compliant	0	0	
Dir 4.2	Advisory	Compliant	0	0	
Dir 4.3	Required	Compliant	0	0	
Dir 4.4	Advisory	Compliant	0	0	
Dir 4.5	Advisory	Compăant.	.0	0	
Dir 4.7	Required	Compliant	0	0	
Dir 4.8	Advisory	Compliant	0	0	
Dir 4.9	Advisory	Compliant	0	0	
Dir 4.10	Required	Compliant	0	0	

Figure 4-1: MISRA Compliance Report

The "MISRA C 2023" compliance document recommends against re-categorizations from a less stringent to a more stringent classification. In addition, it is possible to disapply advisory rules altogether after reviewing the types of violations with the team.

The requirement to document deviations is only necessary for all required rules. Any violations in adopted code should be reviewed. Deviations need to clearly state that violations do not compromise safety and security. Regardless of recategorization, if there is a finding that compromises the safety or security of the system, the issue must be fixed. Also, modifications to the existing code may introduce other issues not clearly seen by the developer.

## **AUTOSAR C++14**

AUTOSAR (AUTomotive Open System ARchitecture) is a worldwide development partnership of OEM manufacturers, Tier 1 automotive suppliers, semiconductor manufacturers, software suppliers, tool suppliers, and others that focus on establishing and standardizing automotive software architecture.

Adaptive AUTOSAR defines a platform for developing automotive control units, which provide sophisticated functionalities like advanced driving assistance systems, media streaming, or software updates via the internet. The platform contains the specification of interfaces that define services and APIs for building modern automotive systems.

A key component of the AUTOSAR Adaptive Platform is the AUTOSAR C++14 coding standard, which defines guidelines for the use of modern C++ in critical and safety-related systems. While AUTOSAR C++14 remains foundational, the automotive industry increasingly aligns with MISRA C++ 2023, the latest standard from the MISRA consortium.

MISRA C++ 2023 modernizes and replaces MISRA C++ 2008, integrating support for C++17 and C++20 while maintaining backward compatibility with AUTOSAR C++14.

Recognized as complementary standards, MISRA C++ 2023 and AUTOSAR C++14 share safety, reliability, and compliance goals. Both standards provide traceability to widely adopted C++ guidelines, including:

- » HIC++ 4.0
- » JSF AV C++
- » SEI CERT C++
- » C++ Core Guidelines
- » ISO/IEC TS 17961

The collaboration between AUTOSAR and MISRA ensures harmonization for developers working in safety-critical domains like automotive, aerospace, and industrial systems.

Parasoft C/C++test, a unified testing tool for C/C++ development, and Parasoft DTP, a reporting and analytics dashboard, deliver comprehensive support for AUTOSAR C++14 and MISRA C++ 2023. This dual coverage enables organizations to:

- » Enforce MISRA rules with industry-leading static analysis checkers.
- » Streamline compliance reporting for ISO 26262, ASPICE, and other automotive safety standards.
- » Address challenges in software quality, security, and regulatory adherence across evolving C++17/C++20 codebases.

By integrating AUTOSAR C++14's robustness with MISRA C++ 2023's modern language support, Parasoft provides a future-proof solution for automotive software teams navigating the transition to advanced C++ ecosystems.

#### AUTOSAR C++14 Compliance

AUTOSAR C++14 does not provide explicit guidance on the process of achieving compliance. However, given that AUTOSAR guidelines are based on MISRA C++ 2008 (superseded by MISRA C++ 2023), it's reasonable to refer to the MISRA Compliance:2020 document for claiming compliance.

The desired situation is to have a static analysis tool that covers as many guidelines as possible. The rules that cannot be enforced with static analysis will require manual reviews, which are expensive.

As with MISRA C or C++ compliance, a deviation handling procedure needs to be established. The deviation procedure formalizes the steps that need to be taken when development needs to deviate from a specific guideline. As MISRA prescribes, it's expected that "...the procedure will be based around obtaining a sign-off for every deviation or class of deviations."

This is a particularly important piece of the puzzle. It prevents abusing the deviation concept by developers deviating at will. Effectively, you'll need formal tickets stored in your system that document every deviation in the source code. At the end of a project, an AUTOSAR Compliance Report must be created to document the overall level of compliance achieved. This summary includes an entry for each guideline outlined in The Guidelines and specifies the compliance level attained.

The possible compliance levels for a guideline are:

- » **Compliant.** No violations of the guideline exist within the project.
- » Deviations. All violations of the guideline within the project are justified by approved deviations.
- » Violations. The project contains violations of the guideline that are not covered by deviations.
- » **Disapplied.** Compliance with the guideline has not been assessed.

Filter: a	atm-for-autosar	Target Build: atm-for-autosa	AUTOS/ ar-2018-12-03 Comp	AR Compli liance Profile: AUTOS, Revision Date: 2018-	AR C++14 Analysis	Tool: Parasoft C++test 10.4	Download
	Guidel	ine Enforcement Plan	Project Con Guideline Re-ca	npliance: 🙁 No ategorization Plan	t Compliant	port (Total: 13) Build	I Audit Report
8	Obligation:	All \$ Target:	Ali 🗘	Analysis: All	\$	Compliance: All	¢
Guideline	Obligation	Target	Analysis	Compliance	# of Violations	In-Code Suppression	# of Deviations s DTP Suppressions
0-1-1	required	implementation	automated	Compliant	0	0	0
0-1-2	required	implementation	automated	Compliant	0	0	0
0-1-3	required	implementation	automated	Compliant	0	0	0
0-1-4	required	implementation	automated	Compliant	0	0	0
0-1-5	required	implementation	automated	No rules enabled	N/A	N/A	N/A
0-4-1	required	infrastructure/toolchain	non-automated	No rules enabled	N/A	N/A	N/A
0-4-2	required	implementation	automated	No rules enabled	N/A	N/A	N/A
0-4-3	required	toolchain	automated	No rules enabled	N/A	N/A	N/A
1-1-1	required	implementation	automated	No rules enabled	N/A	N/A	N/A.
1-1-2	required	toolchain	non-automated	No rules enabled	N/A	N/A	N/A
-1-3	required	implementation	automated	No rules enabled	N/A	N/A	N/A
-2-1	required	implementation	non-automated	No rules enabled	N/A	N/A	N/A

Figure 5-1: AUTOSAR Compliance Report

A formal process for handling deviations must document enforcement methods for every applicable guideline. This document is called the Guidelines Enforcement Plan (GEP). Also, a Guidelines Recategorization Plan (GRP) is needed, which documents in a formal way any changes that are introduced to rule categories. And the Guidelines Compliance Summary (GCS) is a final artifact from the compliance process that presents the level of compliance that was achieved for every guideline.

### Support for AUTOSAR C++14 in Parasoft C/C++test

The only practical way to enforce compliance with a coding standard like AUTOSAR C++14 is with a static analysis tool, like Parasoft C/C++test, a code quality tool supporting multiple testing technologies. Parasoft C/C++test support for AUTOSAR C++14, provides a set of built-in checkers (rules) for verifying compliance with standards including MISRA C 2023, MISRA C++ 2023, JSF AV C++, SEI CERT C/C++, HIC++, CWE Top 25, CWE On the Cusp, OWASP, and more.

Parasoft compliance packs provide users with standard specific configurations, automatic generation of compliance documentation, risk assessment framework, and dynamic compliance reporting dashboards (DTP) to help stakeholders easily aggregate, correlate, and apply analytics to centralize reporting for each step along the complex software supply chain.

Obligation Level	Enforcement by Static Analysis	Supported	Unsupported	All	Percent Coverage
Required	Automated	301	0	301	100%
Required	Partially	18	2	20	90%
Required	Non-Automated	19	22	41	46%
Required	All	338	24	362	93%

Figure 5-2: Parasoft covers 100% of all required and automated AUTOSAR C++14 rules.

> With the 100% coverage for Required & Automated rules, Parasoft testing tool suite ensures AUTOSAR compliance throughout the software development life cycle, improves code quality, and reduces cost associated with resources and time to market.

Parasoft provides comprehensive support for CERT C and CERT C++ secure coding standards with complete coverage of all the CERT C/C++ guidelines including both rules and recommendations that are detectable by static analysis. Checker names, dashboards, and reports use the CERT naming convention to make conformance and auditing easier. A CERT conformance dashboard, which includes the CERT risk score, helps developers focus on the most critical violations.

## **SEI CERT**

The Software Engineering Institute (SEI) Computer Emergency Response Team (CERT) has a set of guidelines to help developers create safer, more secure, and more reliable software. Started in 2006 at a meeting of the C Standard Committee, the first CERT C standard was published in 2008, and is constantly developing and evolving.

There's a book form version published in 2016, but it doesn't include the latest updates. This standard doesn't have specific frozen releases like CWE Top 25 and OWASP Top 10. The standard arose from a large community of over 3,000 people with a focus on engineering and prevention. The CERT secure coding standards focus on prevention of the root causes of security vulnerabilities rather than treating or managing the symptoms by searching for vulnerabilities.

The CERT coding guidelines are available for C, C++, Java, Perl, and Android. They fall into two main categories: rules and recommendations.

Rules are guidelines that are detectable by static analysis tools and require strict enforcement, while recommendations are guidelines that have a lower impact and are sometimes difficult to analyze automatically.

CERT includes a risk assessment system that combines the likelihood of occurrence, severity, and relative difficulty of mitigation. This helps developers prioritize which guideline violations are the most important to investigate. The inclusion of mitigation efforts to the guideline priority is an important addition to the CERT secure coding standards, which many other standards lack.

The CERT bullseye diagram reflects the cost factor. The center of the bullseye represents the highest severity guidelines, which are more difficult to fix. The benefit of this prioritization is focusing on the most critical violations that provide the biggest bang for the buck in security improvement while helping the development team filter out less important warnings.



## SEI CERT C/C++ Conformance

According the SEI CERT C documentation, conformance "requires that the code not contain any violations of the rules specified in this standard. If an exceptional condition is claimed, the exception must correspond to a predefined exceptional condition, and the application of this exception must be documented in the source code."

Although conformance is less specific than standards such as MISRA, the principles remain similar. Rules should be followed, and deviations are rare and well documented. Recommendations should be used when possible and those that aren't needed to be documented.

Violations that persist in the source code need to be documented. However, no deviation is acceptable for performance or usability and the onus is on the developer to demonstrate that the deviation will not lead to a vulnerability.

Parasoft C/C++test provides a comprehensive CERT compliance dashboard and reports. Individual compliance reports are available on demand based on the latest build of the software or any previous build.

These reports can be sorted and navigated to investigate violations in more detail. Also, a conformance test plan is available to correlate the CERT guideline with the appropriate Parasoft static analysis checker is an important tool if conformance documentation is needed for audit purposes. In addition, all the interesting reports as specified by the team are available in a single PDF available for download for auditors.



Figure 6-2: CERT compliance

dashboard

PARASOFT.						🗘 - 😯 - der
			CERT Con	npliance Repo	ort	🛃 Download
	Filter: bc-cert c Tar	get Build: bc-cert c	-2018-08-09 Compliance Profile: S	EI CERT C 2018 Analysis To	ol: Parasoft C++test 10.4.1 Revision	Date: 2019-02-27
			Project Compliance: 🕴 🛚	lissing rule(s) in a	analysis	
		Confo	rmance Testing Plan Devia	ation Report (Total: 4)	Build Audit Report	
		Type:	All    Level: All	Compliance: All	•	
						# of Deviations
Buideline	Туре	Level	Compliance	# of Violations	In-Code Suppressions	DTP Suppressions
ERT-API00_C	Recommendation	L3	A Compliant with Violations	314	0	0
ERT-API01_C	Recommendation	L2	A Compliant with Violations	7	0	0
ERT-API02_C	Recommendation	L2	Compliant with Violations	15	0	0
ERT-API03_C	Recommendation	L3	No rules enabled	N/A	N/A	N/A
ERT-API04_C	Recommendation	L3	No rules enabled	N/A	N/A	N/A
ERT-API05_C	Recommendation	L1	No rules enabled	N/A	N/A	N/A
ERT-API07_C	Recommendation	L3	No rules enabled	N/A	N/A	N/A
ERT-API08_C	Recommendation	L3	No rules enabled	N/A	N/A	N/A
ERT-API09_C	Recommendation	L3	No rules enabled	N/A	N/A	N/A
ERT-API10_C	Recommendation	L1	No rules enabled	N/A	N/A	N/A
ERT-ARR00_C	Recommendation	L2	No rules enabled	N/A	N/A	N/A
ERT-ARR01_C	Recommendation	L1	A Compliant with Violations	5	0	0
ERT-ARR02_C	Recommendation	L2	O Missing rule(s) in analysis	0	0	0
ERT-ARR30_C	Rule	L2	Compliant	0	0	0
ERT-ARR32_C	Rule	L2	Compliant	0	0	0
ERT-ARR36 C	Rule	12	Compliant	0	0	0

Figure 6-3: Auto-generated CERT Compliance Report

## Support for CERT C/C++ in Parasoft C/C++test

Parasoft provides comprehensive support for CERT C and CERT C++ secure coding standards with complete coverage of all the CERT C/C++ guidelines including both rules and recommendations that are detectable by static analysis. Checker names, dashboards, and reports use the CERT naming convention to make conformance and auditing easier. A CERT conformance dashboard, which includes the CERT risk score, helps developers focus on the most critical violations.

## **CWE - Common Weakness Enumeration**

CWE is a list of discovered software weaknesses based on the analysis of reported vulnerabilities (CVEs). The collection of CVEs and CWEs is a U.S. government-funded initiative developed by the software community and managed by the MITRE organization. In its entirety, the CWE list contains over 800 items.

These 800+ items are organized in more usable lists such as the well-known CWE Top 25. The Top 25 lists the most common and dangerous security weaknesses, which are all exploits that have a high chance of occurring and the impact of exploiting the weakness is large. The software weaknesses documented by a CWE are the software implicated in a set of discovered vulnerabilities (documented as CVEs) when analysis was performed to discover the root cause. CVEs are specific observed vulnerabilities in software products that have an exact definition of how to exploit them.

The current version of CWE Top 25 is from 2024. An updated Top 25 is currently in process with improved linking to CVEs and the NVD. Ranking considers real world information so that it truly represents the Top 25 application security issues today. As soon as it is released, Parasoft will have updated support for the latest version.

Rank	ID	Name	Score	CVEs in KEV	Rank Change vs. 2023
1	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	56.92	3	+1
2	CWE-787	Out-of-bounds Write	45.20	18	-1
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	35.88	4	0
4	CWE-352	Cross-Site Request Forgery (CSRF)	19.57	0	+5
5	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	12.74	4	+3
6	CWE-125	Out-of-bounds Read	11.42	3	+1
7	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	11.30	5	-2
8	CWE-416	Use After Free	10.19	5	-4
9	CWE-862	Missing Authorization	10.11	0	+2
10	CWE-434	Unrestricted Upload of File with Dangerous Type	10.03	0	0
11	CWE-94	Improper Control of Generation of Code ('Code Injection')	7.13	7	+12
12	CWE-20	Improper Input Validation	6.78	1	-6
13	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	6.74	4	+3
14	CWE-287	Improper Authentication	5.94	4	-1
15	CWE-269	Improper Privilege Management	5.22	0	+7
16	CWE-502	Deserialization of Untrusted Data	5.07	5	-1
17	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	5.07	0	+13
18	CWE-863	Incorrect Authorization	4.05	2	+6
19	CWE-918	Server-Side Request Forgery (SSRF)	4.05	2	0
20	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	3.69	2	-3
21	CWE-476	NULL Pointer Dereference	3.58	0	-9
22	CWE-798	Use of Hard-coded Credentials	3.46	2	-4
23	CWE-190	Integer Overflow or Wraparound	3.37	3	-9
24	CWE-400	Uncontrolled Resource Consumption	3.23	0	+13
25	CWE-306	Missing Authentication for Critical Function	2.73	5	-5

The current CWE Top 25 is listed below.

Table 7-1: CWE Top 25 For software teams that have a good handle on the Top 25, there is another grouping of the next most common and impactful vulnerabilities called the CWE CUSP. Another way to think of these are the top 25 honorable mentions.

The CWE uses a risk scoring method to rank the Top 25 (and on the CUSP). This score takes into consideration the technical impact of a software weakness (how dangerous an exploit of the weakness is in the real world) as measured by the CWSS (common weakness scoring system). Examples of technical impacts from vulnerabilities may include denial of service (DoS), distributed denial of service (DDoS), read or write access to protected information, unauthorized access, and so on.

The details of these methods aren't too important, but the sorted list is useful in understanding which vulnerabilities to be concerned about the most. As an example, it's possible that your application is purely internal and DoS issues aren't critical for you. Being able to prioritize the most important weaknesses of your application can help overcome overwhelm with static analysis violations.

#### CWE Top 25 and On the Cusp Compliance

Introducing the coding standard compliance process into the team development workflow is not an easy task. As such, it's important to select a tool that will help in achieving compliance without imposing too much overhead and without the requirement for additional manual procedures. The following points are important decision-making factors when selecting the solution for static analysis.

The CWE Top 25 and its lesser known sibling, On the Cusp, are not coding standards per se, but a list of weaknesses to avoid, improving security. To be CWE compliant, a project should be able to prove that it has made reasonable efforts to detect and avoid these common weaknesses.

Parasoft's advanced static analysis tools for C, C++, Java, and .NET are officially compatible with CWE, providing automated detection of both Top 25 and On the Cusp weaknesses (and many more). CWE-centric dashboards give users quick access to standards violations and current project status. A built-in CWE Top 25 configuration is available for C, C++, .NET, and Java and has full coverage of all the 25 common weaknesses.



Figure 7-2: CWE 4.4 - .NET compliance dashboard

> The Parasoft tools include information from the CWRAF risk analysis framework, such as technical impact, so you can benefit from the same type of prioritization based on risk and technical impact and weaknesses found in your code.

The On the Cusp guidelines are also available. When enabled, they're treated the same way as the Top 25 and reports provide the same level of detail. This is useful since the UL 2900 (formerly Underwriters Laboratory) and FDA recommends the full list of guidelines (Top 25 + On the Cusp + OWASP Top 10). It's possible to integrate other guidelines from CWE lists or other standards and guidelines using Parasoft's custom checker configurations as needed.

Parasoft also supports detailed compliance reporting to streamline audit processes. The web dashboards provide the link to compliance reports that provide a complete picture of where a project stands. In addition, the CWE Weakness Detection Plan maps the CWE entry against the checkers that are used to detect the weakness. This helps illustrate how compliance was achieved by an auditor, and the audit reports are available to download as PDFs for easy reporting.

				L Download PD
	CWE C	ompliance Report		Boundard
	Filter: Quartz - CWE 4.4 Target Build: Quartz-2021-06-07 Compliance	Profile: CWE 4.5NET Analysis Tool: Parasoft	dotTEST 2021.2 Revision Date: 2021-11-11	
	Project Compliance:	🕴 Missing rule(s) in analysis		
	Weakness Detection Plan	Deviation Report (Total: 0) Build Audit	Report	
			(topon	
	Compliance:	All		
Weakness	Compliance	# of Violations	# of Deviations	
CWE-11	No rules enabled	N/A	N/A	
CWE-12	No rules enabled	N/A	N/A	
CWE-13	No rules enabled	N/A	N/A	
CWE-14	No rules enabled	N/A	N/A	
CWE-20	S Not Compliant	28	0	
CWE-22	Compliant	0	0	
CWE-59	Compliant	:0	0	
CWE-64	Compliant	0	0	
CWE-77	Compliant	0	0	
CWE-78	Compliant	0	0	
CWE-79	O Not Compliant	28	0	
CWE-80	S Not Compliant	28	0	
CWE-81	No rules enabled	N/A	N/A	
CWE-82	No rules enabled	N/A	N/A	
CWE-83	No rules enabled	N/A	N/A	
CWE-88	O Not Compliant	28	0	
CWE-89	Compliant	0	0	
CWE-90	Compliant	0	0	

Figure 7-3: CWE Compliance Report

## **Unit Testing**

Software verification and validation is an inherent part of automotive software development and testing is a key way to demonstrate correct software behavior. Unit testing is the verification of module design. It ensures that each software unit does what it's required to do.

In addition, safety and security requirements may require that software units don't behave in unexpected ways and are not susceptible to manipulation with unexpected data inputs.



The V-model of software development showing the relationship between each phase and the validation inferred at each stage of testing.

In terms of the classic V model of development, unit test execution is a validation practice to ensure module design is correct. ISO 26262 has specific guidelines for what needs to be tested for unit testing.

ISO 26262 has specific guidelines for testing in accordance with safety integrity levels where requirements-based testing, interface testing, and resource usage evaluation are highly recommended for all levels. Fault injection is recommended at lower ASIL (Automotive Safety Integrity Levels) A and B, and highly recommended at ASIL C and D. Similarly, the method of driving test cases is also specified with recommended practices.

	Methods		ASIL				
			B	С	D		
1a	Requirements-based test <sup>a</sup>	++	++	++	++		
1b	Interface test	++	++	++	++		
1c	Fault injection test <sup>b</sup>	+	+	++	++		
1d	Resource usage evaluation <sup>c, d</sup>	++	++	++	++		
1e	Back-to-back comparison test between model and code, if applicablee	+	+	++	++		
1f	Verification of the control flow and data flow	+	+	++	++		
1g	Static code analysis <sup>f</sup>	++	++	++	++		
1h	Static analyses based on abstract interpretation <sup>g</sup>	+	+	+	+		

#### Table 10 - Methods for verification of software integration

Figure 8-2: ISO 26262 Part 6, 10.4.2:2018

## Table 11 - Methods for deriving test cases for software integration testing

	Methods		ASIL						
			B	С	D				
1a	Analysis of requirements	++	++	++	++				
1b	Generation and analysis of equivalence classes <sup>a</sup>	+	++	++	++				
1c	Analysis of boundary values <sup>b</sup>	+	++	++	++				
1d	Error guessing based on knowledge or experience <sup>c</sup>	+	+	+	+				
a can	Equivalence classes can be identified based on the division of inputs and output be selected for each class.	s, such th	at a repre	sentative	test value				
b	This method applies to parameters' or variables' values approaching and crossing the boundaries and out of range values.								
с	Error guessing tests can be based on data collected through a "lessons learned" [	process a	nd expert	judgment					

Figure 8-3: ISO 26262 Part 6, 10.4.3:2018

Breaking these down individually, consider how each unit test requirement from ISO 26262 can be satisfied and accelerated with test automation tools like Parasoft C/C++test. By the way, if you're already using a unit testing framework like GoogleTest, Cppunit, Boost.Test, or a homegrown one, consider plugging in C/C++test CT. It adds code coverage, requirements traceability, and report generation to these popular open testing frameworks.

## **Unit Test Methods**

#### **Requirement-Based Test**

These tests directly test functionality as specified in each requirement. Test automation tools need to support bidirectional traceability of requirements to their tests and the requirements testing coverage reports to show compliance.

#### **Interface Test**

These tests ensure programming interfaces behave and perform as specified. Test tools need to create function stubs and data sources to emulate the behavior of external components for automatic unit test execution.

#### **Fault Injection Test**

These tests use unexpected inputs and introduce failures in the execution of code to examine failure handling or lack thereof. Test automation tools must support injection of fault conditions using function stubs and automatic unit test generation using a diverse set of preconditions, such as min, max, and heuristic values.

#### **Resource Usage Evaluation**

These tests evaluate the amount of memory, file space, CPU execution or other target hardware resources used by the application.

## **Test Case Drivers**

#### **Analysis of Requirements**

Every requirement drives at minimum a single unit test case. Although test automation tools do not generate tests directly from requirements, they must support two-way traceability from requirements to code and requirements to tests. And maintain requirements, tests, and code coverage information.

#### **Generation & Analysis of Equivalence Classes**

Test cases must ensure that units behave in the same manner for a range of inputs, not just cherry-picked inputs for each unit. Test automation tools must support test case generation using data sources to efficiently use a wide range of input values. Parasoft C/C++test uses factory functions to prepare sets of input parameter values for automated unit test generation.

#### **Analysis of Boundary Values**

Automatically generated test cases, such as heuristic values, boundary values, employ data sources to use a wide range of input values in tests.

#### **Error Guessing**

This method uses the function stubs mechanism to inject fault conditions into tested code flow analysis results and can be used to write additional tests.

### **Automated Test Execution and Test Case Generation**

Test automation provides large benefits to embedded automotive software. Moving away from test suites that require a lot of manual intervention means that testing can be done quicker, easier, and more often.

Offloading this manual testing effort frees up time for better test coverage and other safety and quality objectives. An important requirement for automated test suite execution is being able to run these tests on both host and target environments.

#### **Target-Based Testing for Automotive Systems**

Automating testing for automotive software is more challenging due to the complexity of initiating and observing tests on embedded targets—not to mention the limited access to target hardware that software teams have.

Software test automation is essential to make embedded testing workable on a continuous basis from host development system to target system. Testing embedded software is particularly time consuming. Automating the regression test suite provides considerable time and cost savings. In addition, test results and code coverage data collection from the target system are essential for validation and standards compliance.

Traceability between test cases, test results, source code, and requirements must be recorded and maintained. So data collection is critical in test execution.

Parasoft C/C++test is offered with its test harness optimized to take minimal additional overhead for the binary footprint and provides it in the form of source code, where it can be customized if platform-specific modifications are required.



#### Figure 8-4:

A high-level view of deploying, executing, and observing tests from host to target. One huge benefit that the Parasoft C/C++test solution offers is its dedicated integrations with embedded IDEs and debuggers that make the process of executing test cases smooth and automated. Supported IDE environments include eclipse, VS Code, Green Hills Multi, Wind River Workbench, IAR EW, ARM MDK, ARM DS-5, TI CCS, Visual Studio, and many others.

#### **Automated Test Case Generation**

Unit test automation tools universally support some sort of test framework, which provides the harness infrastructure to execute units in isolation while satisfying dependencies via stubs. Parasoft C/C++test is no exception. Part of its unit test capability is the automated generation of test harnesses and the executable components needed for host and target-based testing.

Test data generation and management is by far the biggest challenge in unit testing. Test cases are particularly important in safety-critical software development because they must ensure functional requirements and test for unpredictable behavior, security, and safety requirements. All while satisfying test coverage criteria.

Parasoft C/C++test automatically generates test cases like the popular CppUnit format. By default, C/C++test generates one test suite per source/header file. It can also be configured to generate one test suite per function or one test suite per source file.
Safe stub definitions are automatically generated to replace "dangerous" functions, which include system I/O routines such as rmdir(), remove(), rename(), and so on. In addition, stubs can be automatically generated for missing function and variable definitions. User-defined stubs can be added as needed.



Figure 8-5:

Parasoft C/C++

per function.

37

## **Regression Testing**

As part of most C and C++ software development processes, regression testing is done after changes are made to software. These tests determine if the new changes impact the existing operation of the software.

Regression tests are necessary, but they only indicate that recent code changes have not caused tests to fail. There's no assurance that these changes will work. In addition, the nature of the changes that motivate the need to do regression testing can go beyond the current application and include changes in hardware, operating system, and operating environment.

### **Regression Testing in Safety-Critical Software**

In safety-critical C and C++ software development, validation is critical in proving correct functionality, safety, and security. Tests are needed to confirm any changes to the application to ensure functionality and to verify there are no unforeseen impacts the rest of the system.

If a test case that previously passed now fails, then a potential regression has been identified. New functionality could be the cause of the failure. If so, the test case may need to be updated with consideration to those changes in input and output values.

Regression testing of embedded systems also includes the execution of:

- » Integration test cases
- » System test cases
- » Performance test cases
- » Stress test cases and more

All previously created test cases may need to be executed to ensure that no regressions exist and that a new dependable software version release is constructed. This is critical because each new software system or subsystem release is built or developed upon. If you do not have a solid foundation the whole thing can collapse.

Parasoft DTP supports the creation of regression testing baselines as an organized collection of tests and will automatically verify all outcomes. These tests automatically run regularly to verify whether code modifications change or break the functionality captured in the regression tests.

If any changes are introduced, these test cases will fail and alert the team to the problem. During subsequent tests, DTP will report tasks if it detects changes to the behavior captured in the initial test.

### How to Decide What to Regression Test

The key challenge with regression testing is determining what parts of an application to test. It is common to default to executing all regression tests when there's doubt on what impacts recent code changes have had—the all-or-nothing approach.

For large C and C++ software projects, this becomes a huge undertaking and drags down the productivity of the team. This inability to focus testing hinders much of the benefits of iterative and continuous processes, potentially exacerbated in embedded software where test targets are a limited resource.

A couple of tasks are required here.

- » Identify which tests need to be re-executed.
- » Focus the testing efforts (unit testing, automated functional testing, and manual testing) on validating the features and related code that are impacted by the most recent changes.

## Understand the Impact of Code Changes on Testing With Test Impact Analysis

Test impact analysis uses data collected during test runs and changes in code between builds to determine which files have changed and which specific tests touched those files. Parasoft's analysis engine can analyze the delta between two builds and identify the subset of regression tests that need to be executed. It also understands the dependencies on the modified units to determine the ripple effect the changes have made on other units.

Parasoft's Jtest for Java testing and dotTEST for C# and VB.NET software testing solutions provide insight into the impact of software changes. Each solution recommends where to add tests and where further regression testing is needed. See the example change based testing report below.

ΆλΑσυγι				¥ · • ·	uamin
Change Based Filter: Parabank-v3 Baseline Build: PARABANK3-20170503 Ta	Testing -	Files 3-21170619 Coverage	<b>Tag:</b> Parabank-All		
File Name	Pass	Fail V	Incomplete	Retest	
ParaBankBeanPostProcessor.java	171	7	10		1
Transaction.java	19	6	10		5
HistoryPoint.java	13	5	0		0
Position.java	25	5	1		0
Customer.java	58	4	5		6
Account.java	34	4	9		6
BankManagerImpl.java	34	4	9		6
JdbcAdminDao.java	102	3	9		2
AbstractLoanProcessor.java	6	2	0		1
LoanRequest.java	9	2	2		1
JdbcTransactionDao.java	14	1	10		5
AdminManagerImpl.java	40	1	4		1
AvailableFundsLoanProcessor.java	4	1	0		1
LoanResponse.java	5	1	2		1
CombinedLoanProcessor.java	1	1	0		0
ConfigurableLoanProvider.java	13	1	2		1
LocalLoanProvider.java	2	1	0		0

Figure 9-1:

51 DADAGOET

An example Change Based Testing report from Parasoft DTP shows tested and not tested areas of the code.

Developers and testers can get a clear understanding of the changes in the codebase between builds using the Process Intelligence Engine (PIE) within Parasoft DTP combined with our proprietary coverage analysis engines:

- » Jtest for Java
- » dotTEST for C# and VB.NET

With this combination, teams can improve efficiency and achieve the promise of Agile.

This form of smart test execution is called test impact analysis. It's sometimes referred to as change based testing.

## **Software Integration Testing**

Integration testing follows unit testing with the goal of validating the architectural design. Integration testing can be done bottom-up and top-down with a combination of approaches likely in many software organizations.

### **Bottom-Up Integration**

This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds. The approach follows a version of the testing pyramid where unit testing forms the foundation of a thorough testing regime. Integration tests follow the integration of units into larger architectural blocks.

### **Top-Down Integration**

In this testing, the highest level modules are tested first. Progressively, testing of lowerlevel modules follows. This approach assumes significant subsystems are complete enough to include and test as a whole.

Figure 10-1: The V-model of software development showing the relationship between each phase and the validation inferred at each stage of testing. The V-model is good for illustrating the relationship between the stages of development and stages of validation. At each testing stage, more complete portions of the software are validated against the phase that defines it.

The V-model might imply a waterfall development method. However, there are ways to incorporate Agile, DevOps, and CI/CD into this type of product development while still being standards-compliant.



While the act of performing tests is considered software validation, it's supported by a parallel verification process that involves the following activities to make sure teams are building the process and the product correctly:

- » Reviews
- » Walkthroughs
- » Analysis
- » Traceability
- » Test
- » Code coverage and more

The key role of verification is to ensure building delivered artifacts from the previous stage to specification in compliance with company and industry guidelines.

## Integration and System Testing as Part of a Continuous Testing Process

Performing some level of test automation is foundational for continuous testing. Many organizations start by simply automating manual integration and system testing (top-down) or unit testing (bottom-up).

To enable continuous testing, organizations need to focus on creating a scalable test automation practice that builds on a foundation of unit tests, which are isolated and faster to execute. Once unit testing is fully automated, the next step is integration testing and eventually system testing.

Continuous testing leverages automation and data derived from testing to provide real-time, objective assessment of the risks associated with a system under development. Applied uniformly, it allows both business and technical managers to make better trade-off decisions between release scope, time, and quality.

Continuous testing isn't just more automation. It's a larger reassessment of software quality practices that are driven by an organization's cost of quality and balanced for speed and agility. Even within the V-model used in safety-critical software development, continuous testing is still a viable approach, particularly during phases of testing, for example, during unit testing and integration testing.

The diagram below illustrates how different phases of testing are part of a continuous process that relies on a feedback loop of test results and analysis.



#### Figure 10-2: A continuous testing cycle

# Parasoft Analysis and Reporting in Support of Integration and System Testing

Parasoft test automation tools support the validation (actual testing activities) in terms of test automation and continuous testing. These tools also support the verification of these activities, which means supporting the process and standards requirements. Key aspects of safety-critical automotive software development are requirements traceability and code coverage.

#### **Two Way Traceability**

Requirements in safety-critical software are the key driver for product design and development. These requirements include functional safety, application requirements, and nonfunctional requirements that fully define the product. This reliance on documented requirements is a mixed blessing because poor requirements are one of the critical causes of safety incidents in software. In other words, the implementation wasn't at fault, but poor or missing requirements were.

#### **Automating Bidirectional Traceability**

Maintaining traceability records on any sort of scale requires automation. Application lifecycle management tools include requirements management capabilities that are mature and tend to be the hub for traceability. Integrated software testing tools like Parasoft complete the verification and validation of requirements by providing an automated bidirectional traceability to the executable test case. This includes the pass or fail result and traces down to the source code that implements the requirement.

Parasoft integrates with market-leading requirements management tools or ALM systems such as IBM DOORS Next, PTC Codebeamer, Polarion from Siemens, Atlassian Jira, Jama Connect, and others. As shown in the image below, each of Parasoft's test automation solutions, C/C++test, Jtest, dotTEST, SOAtest, and Selenic, used within the development life cycle support the association of tests with work items defined in these systems, such as requirements, defects, test case/test runs. Traceability is managed through Parasoft DTP's central reporting and analytics dashboard.



Figure 10-3: Requirements traceability and reporting

Parasoft DTP correlates the unique identifiers from the management system with:

- » Static analysis findings
- » Code coverage
- » Results from unit, integration, and functional tests



Results are displayed within Parasoft DTP's traceability reports and sent back to the requirements management system. They provide full bidirectional traceability and reporting as part of the system's traceability matrix.

Figure 10-4: Parasoft provides a reporting dashboard that capture the project's testing status, correlation to requirements and progress toward completion.



The traceability reporting in Parasoft DTP is highly customizable. The following image shows a requirements traceability matrix template for requirements authored in Polarion that trace to the test cases, static analysis findings, the source code files, and the manual code reviews.

	Polarion Re	quirement Tra	ceabi	lity						
	Polarion Requirement			Tests				Files	F	leviews
iey	Summary	Success %	Total	0	0	A		棄		
ECU-524	The ECU software shall incorporate Safety, Security and Reliability	NA	NA	N/A	NA	NA	NA	N/A	NA	NA
ECU-525	The ECU shall provide the listed set of functional capabilities	NIA	NA	NA	NA	NA	N/A	NA	NA	NA
ECU-526	Development team has decided and shall comply with MISRA C:2012 and CERT C to al	NA	NA	NA	NA	NA	NA	NA	N/A	NA
ECU-527	The ECU shall be monitored during its operational state and report faults. Fault	NA	NA	NA	NA	NA	NA	NA	NA	NA
ECU-528	The system shall have a CRC checking routines to prevent compt or tampered dat	N/A.	NA	NA	NA	NA	NA	NA	NA	NA
ECU-529	The ECU must read the sensor output value and preform necessary calculation on I	NIA	NA	NA	NA	NA	NA	NA	NIA	NA
ECU-630	Memory shall be dynamically allocated at start and deallocated at its appropriat	NA	NA	NA	NA	NA	NA	NA	NA	NA
ECU-531	ECU shall loop in a continues operational state	NIA	NA	NA	NA	NIA	NA	NA	NA	N/A
ECU-632	ECU shall calculate sensor deterioration by way of value averages	NA	NA	NA	NA	NA	NA	NOA	NA	NA
ECU-533	ECU shall handle sensor input values	O 50.00%	2	1	1	0	0	0	0/0	0/0
ECU-534	ECU shall have fault prioritization levels	NA	NA	NA	NIA	NA	NA	N/A	NIA	N/A
ECU-535	ECU shall perform fault detection and reporting	NA	NA	NA	NA	NA	NA	NA	NA	NA
ECU-536	ECU software shall manage its memory allocations within its hardware physical constraints	N/A	NA	NA	NA	NA	NA	NA	N/A	N/A
ECU-537	ECU shall dynamically allocate and initialize memory	NA	NA	NA	NA	NA	NA	NA	NA	NA

Figure 10-6: Requirements traceability matrix template from Parasoft DTP integrated with Siemens Polarion.

Figure 10-5:

codebeamer

to high-level

results.

The bidirectional correlation between test results and work items provides the basis of requirements traceability. Parasoft DTP adds test and code coverage analysis to evaluate test completeness. Maintaining this bidirectional correlation between requirements, tests, and the artifacts that implement them is an essential component of traceability.

## Code Coverage

Code coverage expresses the degree to which the application's source code is exercised by all testing practices, including unit, integration, and system testing — both automated and manual.

Collecting coverage data throughout the life cycle enables more accurate quality and coverage metrics, while exposing untested or under tested parts of the application. Depending on the safety integrity level (ASIL in ISO 26262), the depth and completeness of the code coverage will vary.

Application coverage can also help organizations focus testing efforts when time constraints limit their ability to run the full suite of manual regression tests. Capturing coverage data on the running system on its target hardware during integration and system testing completes code coverage from unit testing.

#### Benefits of Aggregate Code Coverage

Captured coverage data is leveraged as part of the continuous integration (CI) process, as well as part of the tester's workflow. Parasoft DTP performs advanced analytics on code coverage from all tests, source code changes, static analysis results, and test results. The results help identify untested and undertested code and other high risk areas in the software.

Analyzing code, executing tests, tracking coverage, and reporting the data in a dashboard or chart is a useful first step toward assessing risk, but teams must still dedicate significant time and resources to reading the tea leaves and hope that they've interpreted the data correctly.

Understanding the potential risks in the application requires advanced analytics processes that merge and correlate the data. This provides greater visibility into the true code coverage and helps identify testing gaps and overlapping tests. For example, what is the true coverage for the application under test when your tools report different coverage values for unit tests, automated functional tests, and manual tests?

The percentages cannot simply be added together because the tests overlap. This is a critical step for understanding the level of risk associated with the application under development.





# Accelerating Integration and System Testing With Parasoft Test Automation Tools

Parasoft's software test automation tools accelerate verification by automating the many tedious aspects of record keeping, documentation, reporting, analysis, and reporting.

- Two-way traceability for all artifacts ensures requirements have code and tests to prove they are being fulfilled. Metrics, test results, and static analysis results are traced to components and vice versa.
- » Code and test coverage verifies all requirements are implemented and makes sure the implementation is tested as required.
- Target and host-based test execution supports different validation techniques as required.
- Smart test execution manages change with a focus on tests for only code that changed and any impacted dependents.
- Reporting and analytics provides insight to make important decisions and keeps track of progress. Decision making needs to be based on data collected from the automated processes.
- » Automated documentation generation from analytics and test results support process and standards compliance.
- Standards compliance automation reduces the overhead and complexity by automating the most repetitive and tedious processes. The tools can keep track of the project history and relating results against requirements, software components, tests, and recorded deviations.

## Software System Testing

System testing tests the system as a whole. Once all the components are integrated, the entire system is tested rigorously to verify it meets the specified functional, safety, security, and other nonfunctional requirements.

Specialized testing teams perform this type of testing in safety-critical software. System testing falls within the scope of black box testing. As such, it shouldn't require any knowledge of the inner design of the code or logic.

An important distinction with system level testing is the system is tested in an environment that is close to the production environment where the application will be deployed. At this stage, specific safety functions are validated and system wide security testing is run.

## Automotive System Testing at the Service Level

the ECU is a critical leaf node in a business decision making process.

Individual systems within an automobile may not be considered part of a service. However, connectivity into larger systems means they should be. For example, in an automobile, the role of the engine control unit (ECU) alone is to ensure proper combustion and emissions in the engine, but the car is tracking fuel economy, using the ECU, and reporting it to a central server over a wireless connection.

This mileage data is then used to plan routes and estimate operating costs. Suddenly,

Figure 11-1: Engine Control Unit (ECU) and user services all connected to the cloud



Instead of viewing system quality in terms of meeting individual device requirements, the scope is broadened to consider the quality of the services provided. Testing at the service level ensures nonfunctional requirements are met. For example, performance and reliability are difficult to assess at the device level or during software unit testing. Service based testing can simulate the operational environment of a device to provide realistic loads. In the HVAC example, the new temperature sensor can be tested with varying request rates to see if it meets performance requirements.

Security is a significant concern in automotive systems. Cyber attacks most likely originate from the network itself by attacking the exposed APIs. Service based testing can create simulated environments for robust security testing, either through fuzzing (random and erroneous data inputs) or denial-of-service attacks. A new temperature sensor in the HVAC example might operate correctly with expected requests, but crash when overloaded. An attacker might be able to exploit this to overload the system and cause an outage.

### Virtual Test Environment and Service Level Testing

A real test lab requires the closest physical manifestation of the environment in which an automobile is planned to work. Even in the most sophisticated lab, it's difficult to scale to a realistic environment. A virtual lab fixes this problem.

Virtual labs evolve past the need for hard-to-find (or non-existent) hardware dependencies. They use sophisticated service virtualization with other key test automation tools.

#### **Service Virtualization**

Simulates all the dependencies needed by the device under test to perform full system testing. This includes all connections and protocols used by the device with realistic responses to communication. For example, service virtualization can simulate an enterprise server backend that an automobile communicates with. Similarly, virtualization can simulate a dependent system, like traffic or weather data, in a realistic manner.

#### Service and API Testing

Provide a way to drive the system under test in a manner that ensures the services it provides (and APIs provided) are performing flawlessly. These tests can be manipulated via the automation platform to perform performance and security tests as needed.

#### **Runtime Monitoring**

Detects errors in real time on the system under test and captures important trace information.

#### **Test Lab Management and Analytics**

Provide the overarching control of the virtual labs. Once virtualized, the entire lab setup can be replicated as needed and test runs can be automated and repeated. Analytics provide the necessary summary of activities and outcomes.

# Parasoft SOAtest and Virtualize for Service Level Testing of Automotive Software

Developers can build integrations earlier, stabilize dependencies, and gain full control of their test data with Parasoft Virtualize. Teams can move forward quickly without waiting for access to dependent services that are either incomplete or unavailable. Companies can enable partners to test against their applications with a dedicated sandbox environment. These Parasoft solutions are particularly valuable in the development and testing of software-defined vehicles (SDVs).

Parasoft SOAtest delivers fully integrated API and web service testing tools that automate end-to-end functional API testing. Teams can streamline automated testing with advanced functional test creation capabilities for applications with multiple interfaces and protocols.

SOAtest and Virtualize are well suited for network based system-level testing of various types, including the following:

- » Comprehensive protocol stack that supports HTTP, MQTT, RabbitMQ, JMS, XML, JSON, REST, SOAP, and more.
- » Security and performance testing during integration and system testing with integration into the existing CI/CD process.
- » End-to-end testing that combines API, web, mobile, and database interactions into virtual test environments.

## **Structural Code Coverage**

Figure 12-1: ISO 26262 Part 6, 9.4.4:2018 Collecting and analyzing code coverage metrics is an important aspect of safety-critical automotive software development. Code coverage measures the completion of test cases and executed tests. It provides evidence that validation is complete, at least as specified by the software design.

It also identifies dead code. This is code that can logically never be reached. It demonstrates the absence of unintended behavior. Code that isn't covered by any test is a liability because its behavior and functionality are unknown.

The amount and extent of code coverage depend on the safety integrity level. The higher the integrity level, the higher the rigor, and, inevitably, the number and complexity of test cases.

The following table shows the recommendations for types of code coverage at each ISO 26262 ASIL.

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

#### Table 9 - Structural coverage metrics at the software level

**Statement coverage** requires that each program statement be executed at least once and is recommended at the lower ASIL levels. Branch and MC/DC coverage encompass statement coverage.

**Branch coverage** ensures that each decision branch (if-then-else constructs) is executed.

**Modified condition/decision coverage (MC/DC)** requires the most complete code coverage to ensure test cases execute each decision branch and all the possible combinations of inputs that affect the outcome of decision logic. For complex logic, the number of test cases can explode, so the modified condition restrictions are used to limit test cases to those that result in standalone logical expressions changing. See this tutorial from NASA.

Advanced unit test automation tools such as Parasoft C/++test provide all these code coverage metrics. C/C++test automates this data collection on host and target testing and accumulates test coverage history over time. This code coverage history can span unit, integration, and system testing to ensure coverage is complete and traceable at all levels of testing.

## Increasing Code Coverage With Automated Unit Test Case Creation

The creation of productive unit tests has always been a challenge. Functional safety standards compliance demands high-quality software, which drives a need for test suites that affect and produce high code coverage statistics. Teams require unit test cases that help them achieve 100% code coverage.

This is easier said than done. Analyzing branches in the code and trying to find reasons why certain code sections are not covered continues to steal cycles from development teams.

#### Parasoft Coverage Advisor

Parasoft C/C++test resolves the coverage gaps in test suites. Parasoft discovered how to use advanced static code analysis (data and control flow analysis) to find values for the input parameters required to execute specific lines of uncovered code.

In complex code, there are always those elusive code statements of which it is exceedingly difficult to obtain coverage. It's likely there are multiple input values with various permutations and possible paths that make it mind twisting and time consuming to decipher. But only one combination can get you the coverage you need. Parasoft makes it easy to obtain coverage of those difficult to reach lines of code.

Select the line of code you want to cover, and the Coverage Advisor will tell you what input values, global variables, and external calls you need to stimulate the code and obtain coverage.

<pre>22 23 int addSignals(int mainSensorSignal, int coSensorSignal) 74 { 75     int res = 0; 76     if (sensorStatus() I= OK) { 77         local_error("Temp sensor failure"); 78         res = -1; 79     } 84     if (lvalidator(mainSensorSignal)    Ivalidator(coSensorSignal)) { 78         local_error("Signals could not be validated"); 79         return SAFE_DEFAULT; 79     } 84     if (mainSensorSignal &lt; UPPER_LIMIT_MAIN_SIGNAL &amp;&amp; 79         coSensorSignal &lt; UPPER_LIMIT_COSIGNAL) 79     { 70</pre>	Undo Typing Revert File     Save     Open Declaration     Open Type Hierarchy     Open Call Hierarchy     Quick Outline     Quick Type Hierarchy     Explore Macro Expansion     Toggle Source/Header     Open With     Show In     Cut     Copy     Paste	Ctri+Z Ctri+S F3 F4 Ctri+Atr+H Ctri+Ctri+W Ctri+Tac Ctri+Tac Ctri+Tac Ctri+Tac Ctri+Tac Ctri+Shift+W Ctri+Z Ctri+Z Ctri+Z Ctri+Z		
92) 93 «	Quick Fix Source Refactor	Ctrl+1 Alt+Shift+S >		
C Run Unit Tests II C Quality Tasks all Coverage @ Suppressions (deprecated) all Coverage Ad A Finished	Declarations References Search Text	>		
27/27 Tests Succeeded	Make Targets	>		41 Task
0.00.19	Parasoft	3	N Test Using "Recommended Rules"	
• Scope	Run As Debug As Profile As	3 3 3	C Test History Test Using Show Author at Line	->
Execution     Report & Publish	Clean Selected File(s) Build Selected File(s) Team	,	Show test case(s) for covered element     Show in Coverage Advisor	et .

#### Figure 12-2:

Invoking Coverage Advisor by rightclicking on the line of code.

The figure below shows an analysis report providing the user with a solution. The Pre-conditions field expresses:

- » The range and input values for mainSensorSignal and coSensorSignal
- » The expected outputs from the external calls

Upon creating the unit test case with these set parameter values and stubs for external calls, the user will obtain coverage of the line selected, plus the additional lines expressed in the Expected Coverage field.



Figure 12-3: Two test case solutions provided by Coverage Advisor.

## **Requirements and the Traceability Matrix**

In ISO 26262, requirements management is a mandatory part of the software development process and the traceability of those requirements to implementation—and subsequently, proof of correct implementation need to be ensured.

Requirements traceability is defined as "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases)."<sup>1</sup>

In the simplest sense, requirements traceability is needed to keep track of exactly what you're building when writing software. It is used to verify requirements, which means checking documentation and design specifications against requirements to ensure the software does what it's supposed to and that you're only building what is needed.

Traceability works both to prove you satisfied the requirements and to identify what doesn't. If there are architectural elements or source code that can't be traced to a requirement, then it's a risk and shouldn't be there. The benefits go beyond providing proof of the implementation. Disciplined traceability is an important visibility into development progress.

It's important to realize that many requirements in safety-critical software are derived from safety analysis and risk management. The system must perform its intended functions, of course, but it must also mitigate risks to greatly reduce the possibility of injury. Moreover, in order to document and prove that these safety functions are implemented and tested fully and correctly, traceability is critical.

Tracing requirements isn't simply linking a paragraph from a document to a section of code or a test. Traceability must be maintained throughout the phases of development as requirements manifest into design, architecture, and implementation. Consider the typical V diagram of software.



<sup>1</sup>Gotel O.C.Z and Finklestein A.C.W., "An analysis of the requirements traceability problem", in Proceedings of ICRE94, 1st International Conference on Requirements Engineering, Colorado Springs, Co, IEEE CS Press, 1994

Each phase drives the subsequent phase. In turn, the work items in these phases must satisfy the requirements from the previous phase. System design is driven by requirements. System design satisfies the requirements, and so on.

Requirements traceability management (RTM) proves that each phase is satisfying the requirements of each subsequent phase. However, this is only half of the picture. None of this traceability demonstrates that requirements are being met. That requires testing.



In the V diagram shown above, each testing phase verifies the satisfaction of the specifications associated with the corresponding design/implementation phase. In the example, you see:

- » Acceptance testing validates requirements.
- » System testing validates system requirements.
- » Integration testing verifies architecture design.
- » Unit testing verifies module design, and so on.

Software development on any realistic moderate to large scale will have many requirements, complex design and architecture, and possibly thousands of units and unit tests. Automation of RTM in testing is necessary, especially for safety-critical software that requires documentation of traceability for certifications and audits.

### **Requirements Traceability Matrix**

A requirement traceability matrix is a document that illustrates the satisfaction of requirements with a corresponding work item, like a unit test, module source code, architecture design element, and so on.

The matrix is often displayed as a table, which shows how each requirement is "checked off" by a corresponding part of the product. Creation and maintenance of these matrices are often automated with requirements management tools with the ability to display them visually in many forms and even hard copy, if required.

Below is a requirements traceability matrix example from Intland codebeamer. It shows system level requirements decomposed to high-level and low-level requirements, and the test cases that verify each.



Figure 13-3: Requirements traceability matrix example in PTC Codebeamer

#### **Automating Bidirectional Traceability**

Maintaining traceability records on any sort of scale requires automation. Application lifecycle management tools include requirements management capabilities that are mature and tend to be the hub for traceability. Integrated software testing tools like Parasoft complete the verification and validation of requirements by providing an automated bidirectional traceability to the executable test case, which includes the pass or fail result and traces down to the source code that implements the requirement.

Parasoft integrates with market-leading requirements management and Agile planning systems such as IBM DOORS Next, PTC Codebeamer, Polarion from Siemens, Jama Connect, Atlassian Jira, and Azure DevOps Requirements.

As shown in the image below, each of Parasoft's test automation tools, C/C++test, C/C++test CT, Jtest, dotTEST, SOAtest, and Selenic, support the association of tests with work items defined in these systems, such as requirements, stories, defects, and test case definitions. Traceability is managed through the central reporting and analytics dashboard, Parasoft DTP.



Figure 13-4: Parasoft provides bidirectional traceability from work items to test cases and test results, displaying traceability reports with Parasoft DTP and reporting results back to the requirements management system.

Parasoft DTP correlates the unique identifiers from the management system with static analysis findings, code coverage, and test results from unit, integration, and functional tests. Results are displayed within Parasoft DTP's traceability reports and sent back to the requirements management system. They provide full bidirectional traceability and reporting as part of the system's traceability matrix.

The traceability reporting in Parasoft DTP is highly customizable. The following image shows a requirements traceability matrix template with requirements authored in Polarion that trace to the test cases, static analysis findings, the source code files, and the manual code reviews.



#### Figure 13-5: Requirements traceability matrix template from Parasoft DTP integrated with Polarion ALM.

The bidirectional correlation between test results and work items provides the basis of requirements traceability. Parasoft DTP adds test and code coverage analysis to evaluate test completeness. Maintaining this bidirectional correlation between requirements, tests, and the artifacts that implement them is an essential component of traceability.

Bidirectional traceability is important so that requirement management tools and other lifecycle tools can correlate results and align them with requirements and associated work items.

The complexity of modern software projects requires automation to scale requirements traceability. Parasoft tools are built to integrate with best-of-breed requirement management tools to aid traceability into test automation results and complete the software test verification and validation of requirements.

## A Unified, Fully Integated Testing Solution for C/C++ Software Development

## **Tool Qualification for Safety-Critical Automotive** Systems

Safety-critical software development standards like ISO 26262 require that manufacturers prove that the tools they're using to develop software provide correct and predictable results. The process of providing such evidence is known as tool qualification. While it's a necessary process, tool qualification is often a tedious and time-consuming activity that many organizations fail to plan for.

The end deliverable is proof in the form of documentation, but there is more to the qualification process than just delivering a big pile of static documentation. Parasoft's <u>Qualification Kits for C/C++test</u> include a convenient tool wizard that brings automation into the picture and reduces the time and effort required for tool qualification.

## **Pre-Certified Tools**

Tool qualification needs to start with tool selection, ensuring you are using a development tool that is certified by an organization, such as TÜV SÜD. This will significantly reduce the effort when it comes to tool qualification.

Parasoft C/C++test and C/C++test CT are certified by TÜV SÜD for functional safety according to IEC 61508, IEC 62305, EN 50128/EN 50716, and ISO 26262 standards for both host based and embedded target applications. The fully integrated testing solution for

C/C++ software development paves the way for a streamlined qualification of static analysis, unit testing, and coverage requirements for the safety-critical standards.

Pre-certified tools are often enough for lower safety integrity levels such as ASIL A and B. However, for ASIL C and D, tool qualification requires further validation, usually requiring verification and validation of the tool itself on target system hardware.

## **Tool Qualification Requires More Testing**

Traditionally, tool qualification has meant significant amounts of manual labor, testing, and documenting to satisfy a certification audit. But this documentation-heavy process requires manual interpretation and completion. As a result, it's time consuming and

adds to an organization's already heavy testing schedule and budget.

Parasoft leverages its own software test automation tool qualification with Qualification Kits, which include a documented workflow to dramatically reduce the amount of effort required.

#### **Benefits of Using the Qualification Kits**

- » Automatically reduce the scope of qualification to only the parts of the tool in use.
- » Automate tests required for qualification as much as possible.
- » Manage any manual tests as eloquently as possible and integrate results alongside automated tests.
- » Automatically generate audit-ready documentation that reports on exactly what's being qualified — not more, not less!

## **Qualify Only the Tools Used**

There should be no need to do any extra work for qualifying capabilities not used during development. Reducing the scope of testing, reporting, and documentation is a key way to reduce the qualification workload.

The example below shows the use case of C/C++ static code analysis being used to check compliance to the MISRA C 2012 standard, as part of ISO 26262 qualification. The tool then selects only the parts of the qualification suite needed for this function.



Ÿ	-		×
Feature Selection			
Please select for each use case which features it uses			
Please select for each use case which features it uses         Image: Tool C++test (TCL3)         Image: Tool C++test (TCL3)	▼       ■       □	er files (t ble)	es
	<ul> <li>□ □ P Feature Reviewing unit test execution results in Quality Tasks view (testab</li> <li>□ □ Feature Reviewing unit test execution results in Test Progress dialog (testa</li> <li>&gt; □ □ Feature Reviewing unit tests execution results</li> <li>&gt; □ □ Feature Test case execution</li> <li>&gt; □ □ Feature Using Test Case Explorer view to manage tests</li> <li>&gt; □ □ Feature User interface</li> </ul>	le) able)	~
Running coding standards analysis on a project [CS-RUN-01]			^
Description:			
<ol> <li>User shall be able to perform coding standards analysis using selected using any of the following methods:         <ul> <li>using the right-click menu &gt; C++test &gt; Test Using &gt;</li> </ul> </li> </ol>	d, predefined test configuration on a project by selecting a project node in the Navigator view	and	~
	Save         < Back         Next >         Finish	Cancel	

Figure 14-2: Parasoft Qualification Kits allow users to select the options required for their project. Upon selection, only tests and documentation are used and provided from this point forward.

## Leverage Test Automation and Analytics

A unique advantage to qualifying test automation tools is that the tools can be used to automate their own testing. Automating this as much as possible is key to making it as painless as possible. Even manual tests, which are inevitable for any development tool, are handled as efficiently as possible. Step by step instructions are provided and results are entered and stored as part of the qualification record. Parasoft C/C++test and C/C++test CT collect and store all test results from each build, and tests run as they do for any type of project. These results are brought into the test status wizard in the Parasoft Qualification Kits to provide a comprehensive overview of the results like those shown below.

Ÿ			-	- 🗆	×
Test Status from Tool chain with	C++test				
Status of executed test cases. Import exist	ing test results from Tool chain with C++test	EXECUTE AND IMPORT MANUAL TESTS	BEFORE PROCEEDING		
-	-				
Export Manual Tests for Execution	Import Test Results from Run (Excel)	Import Test Results from Run (Tes	tData) 🛛 🖣 Add Test Data	🔍 An	alyze
✓ 🖑 Tool C++test (to be executed: 312	)				~
✓ ₩ UseCase JSF Compliance					
> 🚀 Feature Reviewing coding	standards results in Test Progress				
🔉 🖌 Feature Reviewing coding	standards results in Quality Tasks view				
🔉 😿 Feature Reviewing coding	standards results in Console view				
🔰 🔊 💅 Feature Reporting violation	ns of JSF-001 rule				
🔰 🖌 🚮 Feature Reporting violation	ns of JSF-003 rule				
🔉 😴 Feature Reporting violation	ns of JSF-003_b rule				
🔉 🛫 Feature Reporting violation	ns of JSF-009 rule				
> 🚀 Feature Reporting violation	ns of JSF-010 rule				
Feature Reporting violation	ns of JSF-011 rule				
> 🌠 Feature Reporting violation	ns of JSF-012 rule				
Feature Reporting violation	ns of JSF-013 rule				
Feature Reporting Violation	ns of JSF-U14 rule				
Feature Reporting violation	as of ISE 017 rule				
Feature Reporting violation	as of ISE-012 rule				
Feature Reporting violation	ns of ISE-010 rule				
Feature Reporting violation	ns of ISE-019 h rule				
Feature Reporting violation	ns of JSF-020 rule				
					*
Reviewing coding standards results in 1	est Progress [CS-RES-01]				<u>^</u>
Description:					
(1) User shall be able to review the statistic	information about the results of the coding	tandards analysis in the Test Progress di	alog.		
<ul> <li>(2) It shall contain information about:</li> <li>number of C/C++ language files in sele</li> </ul>	ection				
and					
- number of tested files					
and					
and					
- number of files that C++test failed to to	est				
and					
- number of violations found					
- number of violations suppressed					~
	(?)	Save < Back	Next > Finish	Ca	ncel
	$\odot$			cu	

#### Figure 14-3:

Leveraging centralized data collection and automating the qualification process greatly reduces manual tracking of compliance progress.

## **Managing Known Defects**

Every development tool has known bugs and any vendor selling products for safetycritical development must have these documented. There's more to dealing with known defects than just documenting them. Tool qualification requires proof that these defects are not affecting the results used for verification and validation. For each known defect, the manufacturer must provide a mitigation for each one and document it to the satisfaction of the certifying auditor.

It's incumbent on the tool vendor to automate the handling of known defects as much as possible. After all, the vendor is expecting customers to deal with third-party software bugs as part of their workload! The Parasoft C/C++test qualification kits include a wizard to automate the recording of mitigation for known defects as shown in the example below.

Figure 14-4: Known defects are managed directly in Parasoft C/C++test.

Mitigation Selection	
Please select a mitigation for each critical element (error or bug)	
2 Togqle relevant/remaining errors display	
Copie relevant/remaining errors display     V	<ul> <li>▼ Tool C++test (TCL3)</li> <li>✓ ● Restriction Do not use this feature - there are known bugs in it that have been reported</li> </ul>
<pre> False negative on 'struct' [CPP-33630] Description: JSF-82 (CODSTA-CPP-02) false negative on 'struct' Rule JSF-82 (con to report violations on structures. For example: struct B {     void operator=(B&amp; a) // No violation     return;     }     Comment: } </pre>	×
	Save         < Back         Next >         Finish         Cancel

## **Automation of Tool Qualification Documentation**

The end result of tool qualification is documentation, and lots of it. Every test executed with results, every known defect with mitigation, manual test results, and exceptions are all recorded and reported. Qualification kits from other vendors can be just documentation alone, and without automation, documenting compliance is tedious.

Instead, using the Qualification Kits for C/C++test, the critical documents are generated automatically as part of the workflow.

- Tool Classification Report determines the qualification needed, and presents the maximum safety level classification for C/C++test based on the use cases selected by the user.
- Tool Qualification Plan describes how C/C++test is going to be qualified for use in a safety relevant development project.
- Tool Qualification Report demonstrates that C/C++test has been qualified according to the tool qualification plan.
- Tool Safety Manual describes how C/C++test should be used safely, for example compliant to safety standards, like ISO 26262 and IEC 61508, in safety-critical projects.

In each of these documents, only the documentation required for the tool features in use is generated because the scope of the qualification was narrowed down at the beginning of the project. Automation and narrowing the scope of qualification greatly reduces the documentation burden.

## **Reporting and Analytics for Automotive Software**

Parasoft's extensive reporting capabilities bring the results of Parasoft C/C++test into context. Test results can quickly be accessed within the IDE or exported into the web-based reporting system, DTP.

In DTP, reports can be automatically generated as part of CI builds and printed for code audits in safety-critical organizations. Results from across builds can be aggregated to give the team a detailed view without requiring access to the code within their IDE.

In the reporting dashboard, Parasoft's Process Intelligence Engine (PIE) helps managers understand the quality of a project over time. It illustrates the impact of change after each new code change. Integrating with the overall toolchain, PIE provides advanced analytics that pinpoint areas of risk.

### **Developer's View in the IDE**

Parasoft C/C++test helps teams efficiently understand results from software testing by reporting and analyzing results in multiple ways. Directly in the developer's IDE, users can view:

- » Static analysis findings: warnings and coding standard violations
- » Unit testing details: passed/failed assertions, exceptions with stack traces, info/debug messages
- » Runtime analysis failures with allocation stack traces
- » Code coverage details: percentage values, code highlights, including coverage test case correlation

The Quality Tasks view in the IDE makes it easy for developers to sort and filter the results, for example group per file, per rule, or per project. Developers can make annotations directly in the source code editors to correlate issues with the source code. This provides context and more details about reported issues and how to apply a fix. Code coverage information is presented with visual green and red highlights displayed in the code editor, together with percentage values (for project, file, and function) in a dedicated Coverage view.

Analysis results for both IDE and command-line workflows can also be exported to standard HTML and PDF reports for local reporting. For safety-critical software development, C/C++test provides an additional dedicated report format. It details unit test case configuration and includes the log of results from test execution. Users get a complete report of how the test case was constructed and what happened during runtime.



Figure 15-1: Parasoft C/C++test unified testing view

## **Team Web-Based Reporting**

For team collaboration, Parasoft C/C++test and C/C++test CT publish analysis results to DTP, a centralized server. Developers can access test results from automated runs and project managers can quickly assess the quality of the project. Reported results are stored with a build identifier for full traceability between the results and the build. Those results include:

- » Static analysis findings
- » Metric analysis details
- » Unit testing details
- » Code coverage details
- » Source code details

When integrating into CI/CD workflows, Parasoft users benefit from a centralized and flexible web-based interface for browsing results. The dynamic web-based reporting dashboard includes customizable reporting widgets, source code navigation, advanced filtering, and advanced analytics from Parasoft's Process Intelligence Engine. Users can access historical data and trends, apply baselining and test impact analysis, and integrate with external systems like those for test requirements traceability.

₩PA	RASOFT.										¢-0-	demo <del>-</del>	
=	Report Cente	er											
Ð	Reporting a	& Analytics	for Embed	ded Softwa	are				Filter Parabank_Master	Period Last 10 builds	Baseline Build First Build in Period	Target Build Latest Build	
C	Coverage ····		Coverage (Unit) ··· Coverage (Functional) ···			Coverage (	Manual)		Resource	ce Groups			
*			Image: 2: Unit Test (Parabank-UT)		Image: 3: Functiona	Test (Paraba	Image: 4: Manual Te	st (Parabank	Sort	Image: 1: All (Parabank-All)	Coverage		
C	_		1		1				parabank-mode	el_master		87.8%	
	61	20/	50 1	20/	10.00/		210	2/	parabank-contr	oller_master		87.5%	
+	04.	570	59.2	270	10.0	070	21	70	parabank-view	master		74.8%	
			-						bookstore2_ma	ister		62%	
	2.8K /	4.3K	2.5K /	4.2K	803 / 4.3K		899 / 4	4.3K	bookstore_mas	ter		59.3%	
	Parabank_SA-L	JT_3-12-2020	Parabank_SA-U	JT_3-12-2020	Parabank_SA-L	JT_3-12-2020	Parabank_SA-U	T_3-12-2020		m	ore		
	All Te	ests ····	Unit Te	ests	Function	al Tests 🚥	Tes	t		Coverage T	rend by Build		
			Analysis Type	: Unit Test	Analysis Type: F	unctional Test	Analysis Type: 1	Mahuai Test	il.	Image: 1: All	l (Parabank-Alt)		
	% Passing	83.7% 👃	% Passing	98.7%	% Passing	57.4% 🕇	% Passing	50%	66 -				
	# Toete	368 🛧	B Toote	236	# Teste	122 1	# Toots	10	64 -			/	
	# teses	000 1	a resus	200	# Jeaus	ice 1	# teaca	10	62 -				
	# Failed Tests	55 ↑	# Failed Tests	0	# Falled Tests	52 个	# Failed Tests	3	60				
	and the second s	5				0		2	58				
	# incomplete	.0	# incomplete	- 3	# Incomplete	0	# incomplete	2			Builds		
	All Te	ests	Unit Te	ests	Functiona	al Tests	Tes	t	l.	Test Trer	nd by Build		
			Analysis Type	: Unit Test	Analysis Type: F	unctional Test	Analysis Type: I	Manual Test					
	260	260 1 226		100		10		247			/		
	300 T 122		3001 122 230			122 1 60		10					
	Total 1	Tests	Total T	Tests	Total 1	fests	Total T	ests	301		$\wedge$		
		,		_		_			255 -		\		
	$\sim$	$\sim$							209	P 71 2	<u>, , , , , , , , , , , , , , , , , , , </u>		
											Builds		
		Unit Tests	s by Build			Functional Te	ests by Build			Manual Te	ests by Build		
	- A	Analysis Typ	pe: Unit Test		1	Analysis Type:	Functional Test		Analysis Type: Manual Test				
	237		_		120 -	,		12					
	233 -				80				10				
	220						•		8				
	229				40				6 -				
_	005				Po	wered by Parasoft D	TP. Copyright © 1996-20	21.					

Figure 15-2: Centralized webbased reporting and analytics dashboard.

#### **Test Impact Analysis**

Each and every test performed, including manual, system level, and UI-based, is recorded for test/fail results, along with their coverage impact on the code base. Each additional test is overlaid on this existing information, creating a complete picture of test success and coverage.

As code is changed, the impact is clearly visible on the underlying record, highlighting tests that now fail or code that is now untested. Raising this information in various degrees of detail allows developers and testers to quickly identify what needs to be altered or fixed for the next test run.

## **Risk-Based Assessment**

In addition to change impact analysis, static analysis can be used to highlight areas of the code that appear riskier than others. Risk can take a variety of forms including:

- » Highly complex code
- » Unusually high number of coding standard violations
- » Use of libraries with known vulnerabilities
- » High number of reported static analysis warnings

These are areas of code that may require additional test coverage and even refactoring.

## **Functional Safety Reporting**

Parasoft C/C++ testing solutions provides specific reporting capabilities suited to functional safety development. Here are two report examples:

- » Tests to Requirements Traceability
- » Test to Code Coverage Traceability

The ISO26262 Compliance Pack provides a dedicated, standard-driven report template to help teams comply with industry standards and provide automatically generated reports required for code audits.

## **Code Coverage Metrics**

There are various coverage metrics to consider. Knowing which specific type to apply depends on the software integrity level (ASIL) as defined in ISO 26262.

For automotive systems, the control metrics referenced are statements, branch, modified condition decision coverage (MC/DC). For the strictest requirements, there's object/assembly code. Parasoft supports gathering all these coverage metrics, including terms other industries use like block, call, function, path, decision, and more.



68

## **Custom Analytics, Reports and Dashboards**

Parasoft DTP is highly customizable and supports user-configured custom processors for project-specific analysis, custom widgets, and dashboards.

# Benefits From Centralized, Aggregated Data Analysis and Reporting

#### Manage Compliance With Efficiency, Visibility, and Ease

Instead of just providing static analysis checkers with basic reporting and trends visualization, Parasoft's solution for coding standards compliance provides a complete framework for building a stable and sustainable compliance process.

In addition to standard reporting, Parasoft provides a dedicated compliance reporting module that gives users a dynamic view into the compliance process. Users can see results grouped according to categorizations from the original coding standard, manage the deviations process, and generate compliance documents required for code audits and certification as defined by the MISRA Compliance:2020 specification.

#### **Reduce the Overhead of Testing**

With a unified reporting framework, Parasoft C/C++test efficiently provides multiple testing methodologies required by the functional safety standards including static analysis, unit testing, and code coverage.

By presenting cumulative results from the multiple testing techniques, Parasoft provides consistent reporting that reduces the overhead of testing activities. The analytics, reports, and dashboards:

- » Simplify code audits and the certification process.
- » Eliminate the need for users to manually process reporting to build documentation for the certification process.
- » Focus testing efforts where needed by eliminating extraneous testing and guesswork from test management.
- » Reduce the costs of testing while improving test outcomes with better tests, more coverage, and streamlined test execution.
- » Minimize the impact of changes by efficiently managing the change itself.

#### Pinpoint Priority and Risk Between New and Legacy Code

Parasoft's Process Intelligence Engine enables users to look at the changes between two builds to understand, for example, the level of code coverage or static analysis violations on the code that has been modified between development iterations, different releases, or an incremental development step from the baseline set on the legacy code.

Teams can converge on better quality over time by improving test coverage, reducing the potential risky code. The technical debt due to untested code, missed coding guidelines and potential bugs and security vulnerabilities can be reduced gradually build by build. Using the information provided by Parasoft tools, teams can focus in on the riskiest code for better testing and maintenance.

#### TAKE THE NEXT STEP

Learn how your embedded software development team can accelerate the delivery of high-quality and compliant software. <u>Contact one of our experts today to</u> request a demo.

#### **ABOUT PARASOFT**

<u>Parasoft</u> helps organizations continuously deliver high-quality software with its Alpowered software testing platform and automated test solutions. Supporting the embedded, enterprise, and IoT markets, Parasoft's proven technologies reduce the time, effort, and cost of delivering secure, reliable, and compliant software by integrating everything from deep code analysis and unit testing to web UI and API testing, plus service virtualization and complete code coverage, into the delivery pipeline. Bringing all this together, Parasoft's award-winning reporting and analytics dashboard provides a centralized view of quality, enabling organizations to deliver with confidence and succeed in today's most strategic ecosystems and development initiatives—security, safety-critical, Agile, DevOps, and continuous testing.

"MISRA", "MISRA C" and the triangle logo are registered trademarks of The MISRA Consortium Limited. ©The MISRA Consortium Limited, 2021. All rights reserved.

## **More Resources**

## Safety-Critical Automotive Software Development Assets for Download

## **Case Study**

» Renovo Balances Speed & Agility With Safety & Security in ADAS Development

### Website

- » Automotive Software Testing That Drives Success
- » Easily Automate the Tool Qualification Process
- » Software Compliance Testing Solutions
- » AUTOSAR Compliance With Parasoft
- » ISO 26262 Compliance With Parasoft
- » MISRA Compliance With Parasoft
- » Integrate Codebeamer and Parasoft
- » Integrate Jama Connect and Parasoft
- » Integrate Siemens and Parasoft

## Whitepapers

- » A Practical Guide to Accelerate MISRA C 2023 Compliance With Test Automation
- » DevOps Best Practices for Automotive Development
- » Embedded Cybersecurity Through Secure Coding Standards CWE and CERT
- » Guide to CI/CD for Automotive DevOps
- » Guide to ISO 26262 Software Compliance: Achieving Functional Safety in the Automotive Industry
- » How to Accelerate ISO/SAE 21434 Compliance With Automated Software Testing

- » How to Accelerate MISRA C & SEI CERT C Compliance
- » How to Address Software-Defined Vehicle Challenges With Test Automation
- » How to Satisfy ISO 26262 ASIL Requirements: Guide to Achieving Functional Safety in Automotive
- » How to Streamline Unit Testing for Embedded and Safety-Critical Systems
- » Overcoming the Challenges of Safety & Security in the Renovo Automotive Data Platform
- » Using AUTOSAR C++ Coding Guidelines to Streamline ISO 26262 Compliance

## **Blog Posts**

- » Achieve ASIL D Compliance With Automated Testing
- » Automotive CI/CD DevOps & Test Automation
- » AUTOSAR Architecture Requirements on Runtime Environments
- » Breaking Down the AUTOSAR C++14 Coding Guidelines
- » Coding Standards Compliance for Autonomous Driving Software Testing
- » MISRA C/C++ Code Checking
- » Reducing the Risk and Cost of Achieving Compliant Software
- » Software Development Process for Safety-Critical Systems
- » Turn Software Requirements Into Tested Solutions
- » Why Static Analysis Is Key to Meeting ADAS Safety Standards
- » Why Automotive Cybersecurity Is Important
- » Why Your Development Team Needs TARA

#### Webinars

- » Automotive Software Testing for SDVs
- » General Motor's Journey to Adopting Static Analysis With a Legacy Codebase
- » Get Complete MISRA C 2023 Support in New Parasoft C/C++test Release
- » Is This the Year of EVs?
- » MISRA C:2012 AMD 3: What You Need to Know
- » MISRA C++ 2023: Everything You Need to Know
- » Qualcomm Presents: How to Achieve System on Chip Functional Safety Compliance
- » Streamline ISO 26262 & ISO 21434 Compliance With Automated Testing
- » Test Verification Principles & Practices of ISO 26262, Parts 4 & 6
- » Unveiling Parasoft C/C++test CT for Continuous Testing & Compliance Excellence