Using AUTOSAR C++ Coding Guidelines to Streamline ISO 26262 Compliance

New advanced functionalities being added to modern cars, such as automated driving or advanced safety systems, have forced a paradigm shift in automotive software development environments.

The C language, which dominated the automotive space for years, is no longer sufficient to address the growing complexity of automotive software architectures. With the requirement of object-oriented design, C++ is now a natural choice for many automotive teams.

But C++ is a complicated language and requires a lot of effort to assure predictability, safety, and security. The automotive functional safety standard, ISO 26262, provides some guidance on the software development and V&V processes, but it does not go in depth at the level of language constructs. To get such guidance, organizations turn to coding standards like MISRA C/C++ or AUTOSAR C++. In this paper, we explain how to comply with ISO 26262 by using a static analysis tool that's configured with AUTOSAR C++ 14 compliance checkers.

AUTOSAR C++ 14 CODING GUIDELINES

The AUTOSAR C++ 14 coding standard is a part of Adaptive AUTOSAR, which is a platform for developing automotive control units. The platform is developed by a consortium of automotive companies (including all main players) and provides the specification of interfaces for services and APIs. There are commercial implementations of the AUTOSAR platform provided by a few different vendors that can be used for developing automotive systems.

The Adaptive AUTOSAR specification was designed with the C++ 14 language, and includes a coding standard that restricts the usage of C++ 14 to the constructs that are predictable and that don't lead to unnecessary risk. The coding standard contains more than 300 coding guidelines grouped into different categories. Static analysis tools can help to enforce compliance with the coding guidelines. Parasoft has embraced the evolving standard and is leading the market with its C/C++test product, implementing more rules and simplifying the process of creating necessary compliance documents.

AUTOSAR C++14 was created as an update to the MISRA C++ 2008 coding standard, which is was outdated. In January 2019, the MISRA and AUTOSAR consortiums announced a merger of their two most popular coding standards for safety critical C++ and declared development and maintenance of the standard.

ISO 26262 ROAD VEHICLES FUNCTIONAL SAFETY

ISO 26262 is a functional safety standard for road vehicles. The standard focuses on electrical and/or electronic systems in production cars. ISO 26262 addresses the functional-safety related aspects of development activities and work products, defining functional-safety as "the absence of unreasonable risk due to hazards caused by malfunctioning behavior of electrical or electronic systems."

ISO 26262 consists of ten parts that address different aspects of the product development process, including requirements specification, design, implementation, integration, verification, validation, and configuration. Part 6 of the standard focuses on product development at the software level, including methods and requirements needed for the software development and testing processes, that must be followed to achieve compliance.

PARASOFT®
Automated Software Testing

HOW TO ACHIEVE COMPLIANCE WITH ISO 26262

Compliance with a functional safety standard like ISO 26262 requires significant effort and needs to be an integrated part of the project from the very beginning. Even in the case of the software components, compliance requires specific activities during requirements gathering, planning, and implementation and it is definitely not something that can be "added later."

ISO 26262 specifies a collection of methods that are required to achieve compliance with the standard. To claim compliance, users must provide evidence that all applicable requirements and methods have been implemented. For example, in Part 6, you can find recommendations that refer to the software development process. The methods are grouped in the tables, see example below:

Table 9 — Design principles for software unit design and implementation

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

Not all methods apply to everyone. Applicability of the method depends on the Automotive Safety Integrity Level (ASIL), which is a risk classification defined in the standard (ASIL A represents the lowest degree and ASIL D represents the highest degree of automotive hazard). The method can be highly recommended (++), recommended (+), or neutral (o).

The challenge that teams are typically facing when trying to comply with the standard is how to implement the methods that are recommended for their processes. The decision on how to comply with the specific method or requirement is frequently based on the team experience. In some simple situations, manual procedures and reviews can be an answer, but in most cases, teams are trying to find tools that can automate required methods.

A tool that is used to comply with ISO26262 has to be approved for the intended use through the formal process called tool qualification. The objective of the qualification of software tools is to provide evidence of software tool suitability, for use when developing a safety-related item or element. This can be a time and resource consuming task. Parasoft C/C++test is supported with an automated qualification kit that streamlines the qualification process and includes a TÜV SÜD certification that in many situations is sufficient for tool qualification.

HOW DOES THE AUTOSAR C++ CODING STANDARD STREAMLINE COMPLIANCE WITH ISO 26262?

Following a coding standard like AUTOSAR C++ is a widely accepted method for satisfying some of the requirements stemming from ISO 26262. AUTOSAR C++ 14 provides the traceability tables that map ISO 26262 principles and recommendations to the appropriate coding guidelines. The mapping covers mainly section 8 of Part 6 of ISO 26262, and highly simplifies the process of achieving compliance with corresponding methods and requirements from the standard. See below for an example of the table from the AUTOSAR C++ 14 standard (expanded to show Parasoft C/C++test support):

Table B.6: The criteria that shall be considered when selecting a suitable modeling or programming language

	ISO 26262 Requirement	Relation Type	Related Rule	Comment	C/C++test Support
1a	Enforcement of low complexity	6 - Implemented	A1-4-1	Required on code metrics. Plenty of AUTOSAR C++14 Coding Guidelines rules forbid constructs that introduce unnecessary complexity and are error-prone, e.g. A9-6-2, M10-2-1, A10-2-1.	Supported
1b	Use of language subsets	6 - Implemented	3.1	Plenty of AUTOSAR C++14 Coding Guidelines rules forbid constructs that are allowed from the C++ language perspective, but (1) lead to unstructured designs, (2) are misleading for a developer, (3) are implementation defined. In case some features are to be used in a particular project nonetheless, see chapter 5.4.	Supported
1c	Enforcing of strong typing	6 - Implemented	M5-2-2, M5-2-3, A5-2-2, A5-2-3, M5-2-6, A5-2-4, M5-2-9, A8-4-14, A7-2-3	Restrictions on type casting. Recommendations on strongly typed interfaces and scoped enums	Supported
1d	Use of defensive implementation techniques	6 - Implemented	A0-4-4, A4-7-1, A5-2-5, A6-5-1, A14-1-1, A15-3-4	Error checking required for math functions, integer expressions, array access. Limitations on iteration statements. Recommendations on how to cope with external code failures	Supported
1e	Use of established design principles	6 - Implemented	6.18.5, A18-5-2, A0-1-4	Recommendation on RAII, exception in rules that facilitate correct usage of SFINAE and Concepts. Multiple rules contain references to corresponding rules from multiple standards, which confirms that the provided rule set reflects widely approved coding techniques.	Supported

1f	Use of unambiguous graphical representation	8 - Not applicable		Recommendations on graphical representation is out of scope of AUTOSAR C++14 Coding Guidelines.	N/A
1g	Use of style guides	8 - Not applicable		AUTOSAR C++14 Coding Guidelines does not introduce rules related to coding style	Supported
1h	Use of naming conventions	8 - Not applicable		AUTOSAR C++14 Coding Guidelines does not introduce rules related to naming convention.	Supported

The table above is just one example of the traceability to ISO 26262, touching the specific table from paragraph 5.4.6 (there are similar mappings for other paragraphs). ISO 26262 traceability tables can be found In the B.6 references section of the AUTOSAR C++ 14 standard that is freely available and can be downloaded here: https://www.autosar.org/fileadmin/user_upload/standards/adaptive/18-10/AUTOSAR_RS_CPP14Guidelines.pdf

The AUTOSAR C++ 14 coding guidelines alone are not sufficient to achieve compliance with ISO 26262 for the software component. Some methods in the standard can't be covered with the application of AUTOSAR guidelines, such as methods 1g and 1h from the table above. Method 1g recommends "Use of style guides" and method 1h recommends "Use of naming conventions." AUTOSAR C++ 14 does not include any style guides or naming conventions. Both methods, however, can be easily implemented with Parasoft C/C++test, which includes more than 3000 static analysis checkers, including code style checkers, and provides a module for creating a custom static analysis rules.

Methods in the standard that can't be implement with static analysis in general require other testing techniques such as fault injection testing.

FINDING THE RIGHT TOOL FOR AUTOSAR C++ 14 COMPLIANCE

Introducing the coding standard compliance process into the team development workflow is not an easy task. As such, it is very 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.

1. **Coverage of the coding guidelines from the standard**
   AUTOSAR C++ 14 defines a substantial number of the guidelines. The most up-to-date version of the AUTOSAR coding standard contains at this moment approximately 400 guidelines, with 350 of these guidelines possible to be enforced with static analysis. Supporting this number of guidelines is a challenge for static analysis tool vendors, and not all static analysis tools available on the market cover the standard sufficiently enough for compliance.

   Parasoft C/C++test is the leading solution in this case, covering the highest number of the AUTOSAR C++ guidelines, and continuing to implement more every day. (You can request the information about current coverage for the AUTOSAR C++ standard by at Parasoft's customer portal.)

2. **Support for Data and flow technology**
   Guidelines defined in the AUTOSAR C++ coding standard have different levels of complexity. Some are simple guidelines that can be enforced with relatively simple static analysis technology, like:

> **Rule A0-4-2 (required, implementation, automated)**
> **Type long double shall not be used.**

   But there are also guidelines that require sophisticated data and control flow analysis to simulate the paths in the analyzed source code and decide if a given guideline is violated or not. For example, the following guideline:

> **Rule A5-2-5 (required, implementation, automated)**
> **An array or container shall not be accessed beyond its range.**

   This guideline cannot be reliably detected without data and control flow analysis. The static analysis tool you choose has to evaluate the paths in the code to correctly determine if the index that is used for accessing the data in the container is within the correct range or not. Many commercial tools and most open-source tools on the market apply very rudimentary flow analysis to this class of problems, and in effect they either miss an issue in the code or report an enormous number of false-positives, which consume a huge amount of time to review, and kill productivity.

   When benchmarking a static analysis tool, it is recommended to put special attention on comparing results for more complex guidelines, which require flow analysis technology. (You can request an exemplary list of the guidelines that require flow analysis at the Parasoft customer portal.)

3. **Support for tool qualification**
   Although AUTOSAR C++ does not explicitly require tool qualification to approve the static analysis solution for use, ISO 26262 does. So when planning to use AUTOSAR C++ for streamlining the compliance with ISO 26262, it is recommended to pick a static analysis solution that supports end-users with appropriate certificates and a qualification kit. Parasoft C/C++test is supported with a TÜV SÜD certification, approving it for use when developing safety-critical software with ISO 26262, and with a qualification kit that can be used to approve the tool for use in the projects with highest level of risk.

SUMMARY

Following a coding standard like AUTOSAR C++ 14 can help organizations achieve compliance with ISO 26262, as there are multiple methods and requirements defined in the ISO 26262 standard that can be satisfied by conforming with the AUTOSAR coding guidelines. AUTOSAR C++ 14 provides dedicated traceability tables that demonstrate the mapping between ISO 26262 requirements and the coding guidelines, and teams wishing to streamline their ISO 26262 compliance efforts by applying AUTOSAR C++ coding guidelines need to be well informed when they select the static analysis tools for use. Of utmost importance to the success of compliance, the static analysis tool should provide: high coverage of the guidelines in the static analysis checkers, advanced flow analysis technology, and support for the end-user in the tool qualification process.

www.parasoft.com

Parasoft Headquarters:
+1-626-256-3680

Parasoft EMEA:
+31-70-3922000

Parasoft APAC:
+65-6338-3628

PARASOFT®
Automated Software Testing