WHITE PAPER

Guide to Automotive Software Development: Automotive Standards, Security, and Emerging Technology

Introduction

The automotive software development process can be complex and time-consuming, which is why it is important to use the right software development tools and follow best practices for automotive software development. By doing so, you are able to ensure secure, reliable, and standards-compliant automotive software.

Here, we provide an overview of the key automotive software standards that you should use, the best practices to safeguard against security threats, and touch on autonomous vehicles and smart car features.

PERFORCE

Table of Contents

Overview of Automotive Standards

All vehicles are governed by standards. These include functional safety standards and functional security standards and these in turn require the use of coding guidelines for the development of the many software components in the vehicle.

Overview of Coding Guidelines

Although no function safety or security standard specifies a particular coding standard, there are internationally recognized coding guidelines available to help meet the required security and safety standards.

MISRA

MISRA, originally written for the automotive industry, provides coding standards for developing safety-critical systems.

The initial version, published in 1998 was for C, and this was then extended to C++ in 2008.

MISRA C is the most widely used set of coding guidelines for C around the world. The most recent version of the standard is MISRA C:2012.

MISRA C++ is widely used by safety-critical developers. The current version was published in 2008 but an update is forthcoming in the near future.

MISRA coding guidelines are now widely used by industries such as aerospace and defense, telecommunications, medical devices, and rail as well as automotive

AUTOSAR C++ 14

The AUTOSAR coding guidelines are for the use of the C++14 language in critical and safety-related systems. They were developed for use in the AUTOSAR Adaptive Platform, but are applicable to any safety-critical applications written in C++.

Since MISRA C++ was published, C++ has evolved and other C++ coding guidelines are available, for example HIC++, CERT C++, and C++ Core Guidelines. AUTOSAR C++ 14 addresses these changes and incorporates the expert knowledge embedded in these other coding standards. AUTOSAR C++14 is based on MISRA C++:2008 coding guidelines but with the addition of the best features of other C++ coding standards, such as JSF and CERT C++.

The standard allows the use of some features that are not permitted by other C++ coding standards, including:

- Dynamic memory
- Exceptions
- Templates
- Inheritance
- Virtual functions

How to Achieve Coding Standard Compliance

Achieving compliance to any coding standard takes knowledge, skill, and the right tools. Here are seven recommended steps to achieve compliance:

1. **Know the Rules**
   You need to know the coding rules pertinent to which version of C or C++ you're using.
2. **Check Your Code Constantly**
   Continuously inspecting your code for violations is the best way to improve quality.
3. **Set Baselines**
   Embedded systems come with legacy codebases. By setting baselines, you can focus on making sure your new code is compliant.
4. **Prioritize Violations Based on Risk**
   You could have hundreds or even thousands of violations in your code. That's why it's important to prioritize rule violations based on risk severity. Some static code analysis tools can do this for you.
5. **Document Your Deviations**
   Sometimes there are exceptions to the rule. But when it comes to compliance, every rule deviation needs to be well-documented.
6. **Monitor Your Compliance**
   Keep an eye on how compliant your code is. Using a static code analyzer makes this easier by automatically generating a compliance report.
7. **Choose the Right Static Code Analyzer**
   Choosing the right static code analyzer makes everything else easy. It takes care of scanning your code — new and legacy — for violations. It prioritizes vulnerabilities based on risk.

ISO 26262 and ASIL: Automotive Functional Safety

ISO 26262 - "Road vehicles — functional safety", is the major functional safety standard used in the automotive industry, and ASIL is a key component to determine safety requirements for software development. It is a risk-based safety standard and applies to electric and/or electronic systems in production vehicles. This includes driver assistance, propulsion, and vehicle dynamics control systems.

It covers the functional safety aspects of the entire development process:

- Requirements specification
- Design
- Implementation
- Integration
- Verification
- Validation
- Configuration

WHY IS ISO 26262 IMPORTANT?

The goal of the standard is to ensure safety throughout the lifecycle of automotive equipment and systems.

Specific steps are required in each phase. This ensures safety from the earliest concept to the point when the vehicle is retired.

Compliance to this standard is compulsory for any road vehicle and by complying, you'll avoid or control systematic failures, detect or control random hardware failures and be able to mitigate the effects of failure.

ISO 26262 FUNCTIONAL SAFETY FOR SOFTWARE DEVELOPERS

Part 6: Product development at the software level and Part 8: Supporting processes are the sections applicable to software development. They detail the steps that must be taken to ensure the safety of each component.

WHAT IS ASIL?

Automotive Safety Integrity Level (ASIL) is a key element of ISO 26262 and it is used to measure the risk of a specific system component. The more complex the system, the greater the risk of systematic failures and random hardware failures.

There are four Automotive Safety Integrity Level values, A–D where ASIL A is the minimum level of risk and ASIL D is the maximum. Compliance requirements become stricter as you go from A to D.

There is an additional option — QM (quality management) which is used to note that there isn't a safety requirement for that component.

HOW TO DETERMINE ASIL?

ASIL is determined by three factors — severity, exposure, and controllability.

SEVERITY

Severity measures how serious the damages are of a system failure. Damages include both people and property.

There are four classes of severity:

1. **S0:** No injuries.
2. **S1:** Light to moderate injuries.
3. **S2:** Severe to life-threatening (survival probable) injuries.
4. **S3:** Life-threatening (survival uncertain) to fatal injuries.

EXPOSURE

Exposure is the likelihood of the conditions under which a particular failure would result in a safety hazard.

The probability of each condition is ranked on five-point scale:

1. **E0:** Incredibly unlikely.
2. **E1:** Very low probability (injury could happen only in rare operating conditions).
3. **E2:** Low probability.
4. **E3:** Medium probability.
5. **E4:** High probability (injury could happen under most operating conditions).

CONTROLLABILITY

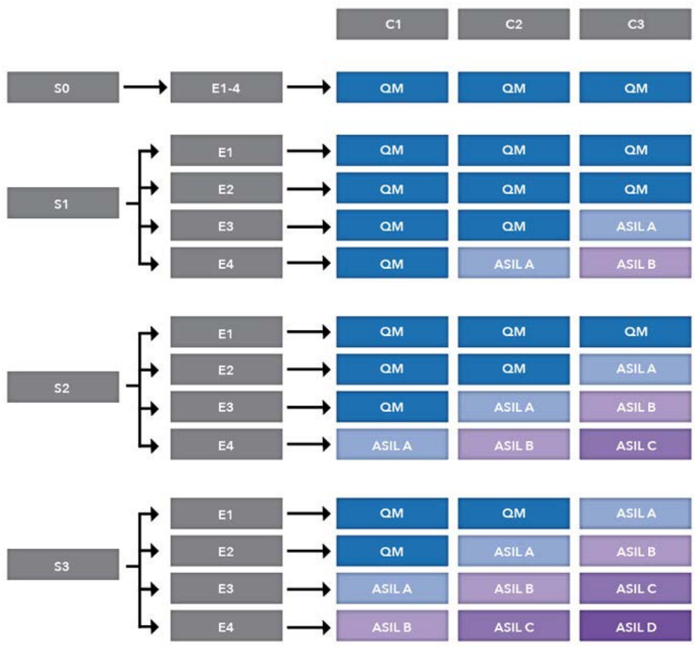
Controllability is a measure of the probability that harm can be avoided when a hazardous condition occurs. This condition might be due to actions by the driver or by external measures.

The controllability of a hazardous situation is ranked on a four-point scale:

1. **C0:** Controllable in general.
2. **C1:** Simply controllable.
3. **C2:** Normally controllable (most drivers could act to prevent injury).
4. **C3:** Difficult to control or uncontrollable.

HOW TO DETERMINE ASIL

Once you've determined severity, probability, and controllability, you can determine the Automotive Safety Integrity Level. Table 4 of Part 3 provides guidance on this.



		C1	C2	C3
S0	E1-4	QM	QM	QM
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	ASIL A
	E4	QM	ASIL A	ASIL B
S2	E1	QM	QM	QM
	E2	QM	QM	ASIL A
	E3	QM	ASIL A	ASIL B
	E4	ASIL A	ASIL B	ASIL C
S3	E1	QM	QM	ASIL A
	E2	QM	ASIL A	ASIL B
	E3	ASIL A	ASIL B	ASIL C
	E4	ASIL B	ASIL C	ASIL D

HOW TO COMPLY WITH ISO 26262

Compliance with the safety standard is important, whether you're developing traditional automotive components (e.g., integrated circuits) or virtual ones (e.g., automotive hypervisors). And it's critical to maintain compliance throughout your software development lifecycle.

But complying can be difficult for development teams. Systems and codebases grow complex. And that makes it difficult to verify and validate software.

You can make it easier by using certified software development tools.

ESTABLISH TRACEABILITY

Fulfilling compliance requirements — and proving you met them — is a tedious process. You need to document the requirements and trace them to other artifacts — including tests, issues, and source code.

Establishing requirements traceability makes your verification process easier, and it helps you manage risk in the development process.

Storing your code in a version control system securely manages revision history for all your digital assets. You'll get fine-grained access controls, high-visibility audit logs, strong password security, and secure replication. So, you can be confident in your code.

APPLY A CODING STANDARD

ISO26262 requires that a coding standard is applied which will allow fulfill specific coding and design guidelines.

Applying a coding standard, such as MISRA or AUTOSAR, is made easier by use a static analyzer.

MOTORCYCLE STANDARDS FOR FUNCTIONAL SAFETY

The first edition of ISO 26262 , published in 2011, covered series production passenger cars. While much of the guidance contained within this standard was also relevant to motorcycles, the hazard analysis and risk assessment for motorcycles required a different approach.

Therefore, the scope of the second edition of ISO 26262, published in 2018, was extended to provide guidance to motorcycle manufacturers. Part 12, "Adaption of ISO 26262 for Motorcycles" was added which places more responsibility on the motorcyclist rather than the motorcycle to mitigate risks. To better assign safety criticality to a system, the Motorcycle Safety Integrity Levels (MSIL) were developed.   They are determined by the same factors as ASIL and are assigned the same values, A-D, but include elements that are specific to motorcycle applications.

Once the MSIL has been determined, it can be mapped to an equivalent ASIL:

MSIL	ASIL
QM	QM
A	QM
B	A
C	B
C	C

This then allows motorcycle applications to be developed according to the aligned ASIL with only some minor differences.

ISO/PAS 21448 — Safety In Autonomous Driving

ISO/PAS 21448 Road Vehicles — Safety of the Intended Functionality (SOTIF) applies to functionality that requires proper situational awareness in order to be safe. The standard is concerned with guaranteeing safety of the intended functionality — SOTIF — in the absence of a fault. This is in contrast with traditional functional safety, which is concerned with mitigating risk due to system failure.

SOTIF provides guidance on design, verification, and validation measures. Applying these measures helps you achieve safety in situations without failure.

For example:

- Design measure example: requirement for sensor performance.
- Verification measure example: test cases with high coverage of scenarios.
- Validation measure example: simulations.

WHY SOTIF IS IMPORTANT

Automated systems have huge volumes of data — and that data is fed to complex algorithms. AI and machine learning are critical for developing these systems.

To avoid potential safety hazards, AI will need to make decisions. This includes scenarios that require situational awareness.

Using ISO 21448 will be key to ensure that AI is able to make decisions and avoid safety hazards.

For example:

The road is icy. An AI-based system might be unable to comprehend the situation — and respond properly. This impacts the vehicle's ability to operate safely. Without sensing the icy road condition, a self-driving vehicle might drive at a faster speed than is safe for the condition. Fulfilling ISO 21448 means taking that situation into account and making decisions based on probability.

The goal of SOTIF is to reduce potential unknown, unsafe conditions.

HOW ISO 21448 IS RELATED TO ISO 26262

Although ISO 26262 covers functional safety in the event of system failures, it doesn't cover safety hazards that don't lead to a system failure.

ISO 26262 still applies to existing, established systems — such as dynamic stability control (DSC) systems or airbags. For these systems, safety is ensured by mitigating the risk of system failure.

ISO 21448 applies to systems such as emergency intervention systems and advanced driver assistance systems. These systems could have safety hazards — without system failure.

ISO 21448 will be important for functional safety in autonomous driving. But compliance with established functional safety standards such as ISO 26262 will remain important.

ISO 21434 — Automotive Software Security

ISO 21434 "Road vehicles — cybersecurity engineering" is an automotive standard currently under development. It focuses on the cybersecurity risk in road vehicle electronic systems.

The standard will cover all stages of a vehicle's lifecycle — from design through to decommissioning by the application of cybersecurity engineering. This will apply to all electronic systems, components, and software in the vehicle, plus any external connectivity.

What's more, the standard will provide developers with a comprehensive approach to implementing security safeguards that spans the entire supplier chain. The intent behind the standard is to provide a structured process to ensure that cybersecurity considerations are incorporated into automotive products throughout their lifetime.

The standard will require automotive manufacturers and suppliers to demonstrate due diligence in the implementation of cybersecurity engineering and that cybersecurity management is applied throughout the supply chain to support it.

It is intended that organizations will encourage a cybersecurity culture so that everything is designed with security considerations from the start.

HOW TO COMPLY WITH ISO 21434

ISO/SAE 21434 has specific requirements for software development including analysis to check for inherent weaknesses and the overall consistency, correctness, and completeness with respect to cybersecurity requirements.

Cybersecurity should be at the forefront of all design decisions including the selection of the programming language to be used for software development.

There are several criteria to be considered when selecting a programming language, including:

- Secure design and coding techniques.
- Unambiguous syntax and semantic definitions.

However, some of these criteria may not be sufficiently addressed in the selected language. Which is why there are several ways of addressing these language deficiencies, including:

- Use of language subsets.
- Enforcement of strong typing.
- Use of defensive implementation techniques.

It is recommended to use coding guidelines to address the deficiencies of the chosen language.

C continues to be the most common language used in automotive software. MISRA C:2012 revision 1 and CERT C guidelines are particularly recommended in ISO/SAE 21434 for any projects using the C language.

Creating a language subset is the core of MISRA C:2012 and CERT C guidelines. MISRA C:2012 revision 1 states: "The MISRA C Guidelines define a subset of the C language". Both guidelines achieve this by preventing the use of functionality that may cause critical or unspecified behavior.

Strong typing ensures that there is an understanding of the language data types and thus prevents certain classes of programming errors. Using coding guidelines, such as MISRA C:2012 and CERT C, that have strong typing ensures correctness and consistency.

Defensive implementation techniques allow software to continue to function even under unforeseen circumstances. It requires thought about "what might happen". There needs to be, for example, consideration of possible tainted data and understanding of the order of evaluation of arithmetic functions. Above all the code needs to be simple to understand.

All defensive implementation techniques should start with the use of recognized coding guidelines. Both MISRA C:2012 Revision 1 and CERT C achieve this by identifying critical and unspecified language behavior and thus making the resulting code more reliable, less prone to errors, and easier to maintain.

The Essential Automotive Software Quality Metrics

In the Automotive Industry, software quality is paramount and software metrics are an important measure of that quality and are applicable to both function safety and functional security standards

However, no single metric can give a definitive measure of the quality of software and Automotive suppliers need to agree with their OEM both the metrics they require and the acceptable limits of the values of those metrics.

However, it is difficult to select the set of metrics that give the quality coverage required.

HERSTELLER INITIATIVE SOFTWARE METRICS

In the Automotive industry, the obvious starting point for the selection of metrics are those defined in Hersteller Initiative Software (HIS).

HIS defines a common set of software metrics which permits a supplier to make statements about the quality of the software product and the software development process. In addition, an acceptable range of values of the defined metrics is specified.

These metrics are separated into distinct categories:

- 15 Metrics with limits that generally measure the complexity of the code.
- 3 Metrics without limits that are simply measured values that must be documented.

METRICS WITH LIMITS

Metrics with limits indicate range of values showing the acceptable boundary limits. Violations of the boundary limits must be justified, and further action is required by the supplier.

Examples of the metrics with limits specified in HIS:

CYCLOMATIC COMPLEXITY “V(G)”

Cyclomatic Complexity is the count of the number of linearly independent paths through the source code.

It can be used in two ways:

1. To limit the complexity of code.
2. To determine the number of test cases necessary to thoroughly test it.

NUMBER OF GOTO STATEMENTS “GOTO”

This metric is very simple, but it can easily be seen that the higher the number, the more paths through the code, which means the more difficult the code is to test.

NUMBER OF RETURN POINTS WITHIN A FUNCTION „RETURN“

Good practice dictates that the ideal value of this metric should be 1 as this improves the maintainability of the function (a function with no specific return is also acceptable).

METRICS WITHOUT LIMITS

All the metrics in this section are similar: STMT (changed), STMT (new), STMT (deleted).

These measure the number of statements in a piece of software that have changed, are new, or have been deleted between the previous and the current version of the software.

These are used to calculate the stability index, which is part of Metrics with limits.

HIS is purely concerned with the coding phase of the software life cycle.

By analyzing these metrics, and ensuring that they are within the specified limits, the effort required in the following phases — particularly testing — will be reduced.

METRICS FOR ISO 26262

It is necessary for Automotive applications to certify to ISO 26262, and as a requirement to achieve this certification, a series of metrics must be gathered. The required level of metrics depends on the ASIL which determines the degree of risk. Higher ASILs require more thorough quality measures to control the risk.

Specific metrics are not required, but there are obvious well-known metrics that are applicable.

For example, **ENFORCEMENT OF LOW COMPLEXITY which is HIGHLY RECOMMENDED FOR ALL ASIL** can be measured by lines of code (LOC) and Cyclomatic Complexity (as discussed in HIS metrics).

Similarly, at an architectural level, **RESTRICT SIZE AND COMPLEXITY OF SOFTWARE COMPONENTS — HIGHLY RECOMMENDED FOR ALL ASIL can be measured by** Halstead metrics which look at the source code to identify areas that may be subject to defects by interpreting the code as a sequence of tokens.

The metrics that count the tokens are:

- STM20 — Counts ALL operands in the file
- STM21 — Counts ALL operators in the file

Other measures can be calculated regarding program length and difficulty.

For example:

- STM22 — Number of statements in a software component
- STVAR — Total number of Variables
- STTLN — Total Pre-processed Source Lines

There are, of course, other sections of ISO 26262 that require metrics, particularly methods for tests and deriving test cases.

SOFTWARE QUALITY METRICS WILL ALWAYS MATTER FOR AUTOMOTIVE SOFTWARE

Software metrics are vital for assessing and maintaining quality in the Automotive Industry.

There are metrics that are specific to the requirements of Automotive OEMs and suppliers, but the choice of metrics should not be limited by those necessary for certification purposes. The metrics selected should be applicable to the role of the viewer; the OEM's view is different to that of the supplier.

Metrics should be selected to measure the progress to achieve specific goals, and the data gathered analyzed and used by the appropriate people. When this is done, they are invaluable as a measure of progress and current software quality plus as an aid to improvement in the future.

The Future of Automotive Software Development

Future development of Autonomous vehicles relies on AI and machine learning.

One of the biggest challenges in this area is security. The starting point is a Secure Development Processes

Here are three examples of key secure development processes:

1. Good programming practices and thorough testing efforts are critical for eliminating security vulnerabilities. This can be achieved by using secure coding standards.
2. Threat modeling and risk mitigation are key to developing safe components. This can be achieved by doing a hazard and risk analysis.
3. Control over the build/release environment is key to keeping hackers out — and keeping the build secure. This can be achieved through access controls in your CI/CD environment.

Part of the secure development process should be automation. Applying automation to design, verification, and validation processes makes development teams more efficient.

Using a requirements management tool contributes to safer design of the software.

Using a test case management tool can help you ensure high coverage of different scenarios. This helps with software verification.

Using a static analysis tool can help you simulate potential run-time scenarios. This helps with software validation.

How Perforce Software Development Tools can Help Ensure Secure, Reliable, and Standards-Compliant Automotive Software

The most effective way to ensure that your automotive software is secure, reliable, and standards-compliant is to use a suite of tools, including a static code analyzer (like Helix QAC or Klocwork), a version control systems tool (like Helix Core), and an application lifecycle management tool (like Helix ALM).

A static analyzer can be used to provide automatic enforcement of automotive coding guidelines — such as MISRA and AUTOSAR.

Yet, static analysis can do so much more than this, such as:

- Automatically and consistently enforcing coding standards and detecting rule violations.
- Detecting compliance issues earlier in the SDLC.
- Accelerating code reviews.
- Reporting compliance over time and across product versions.

See for yourself how Perforce static code analyzers can help ensure that your automotive software is secure, reliable, and compliant. Request your free trial today.

TRY PERFORCE STATIC CODE ANALYZERS

perforce.com/products/sca/free-static-code-analyzer-trial

A version control systems tool, like Helix Core, supports your team and files as they grow. In addition, Helix Core supports build automation by:

- Providing a shared, centralized repository for commits.
- Maintaining a single source of truth for the build.
- Integrating with Jenkins and other build runners for better CI/CD projects.
- Automating workflows.

TRY HELIX CORE

perforce.com/products/helix-core/free-version-control

Helix ALM provides end-to-end traceability by linking your requirements, test cases, and issues all inn one platform. Its configurable workflow easily adapts to the way you already work. This helps ensure that everyone on your team is able to seamlessly work together.

TRY HELIX ALM

perforce.com/products/helix-alm/free-alm-trial

About Perforce