# A Practical Guide to Scoping and Testing for MISRA Compliance

## Chapter 1: The Foundation of MISRA

### 1.1 Executive Summary

The Motor Industry Software Reliability Association (MISRA) provides a set of widely adopted coding guidelines for the C and C++ programming languages.1 While originally developed for the automotive industry, MISRA has evolved into the de facto standard for software development in a multitude of safety- and security-critical domains.2 The purpose of these guidelines is to enable the development of software that is robust, secure, and portable, thereby reducing risk and improving quality in complex embedded systems.2

Achieving MISRA compliance is not merely a matter of a final code check. It is a systematic process that must be integrated throughout a disciplined software development lifecycle, from initial design through to final testing and documentation.4 Effective scoping for MISRA testing requires a clear understanding of the guideline framework, a formal deviation process for permissible exceptions, and the strategic selection and deployment of automated static analysis tools.6 This report provides the essential background and a structured framework to support these scoping efforts, ensuring a pragmatic and auditable path to compliance.

### 1.2 Introduction to the Motor Industry Software Reliability Association (MISRA)

The Motor Industry Software Reliability Association is a collaborative organization formed by a consortium of vehicle manufacturers, component suppliers, and engineering consultancies.1 Its primary mission is to produce and maintain best-practice guidelines for the development of software used in electronic systems, particularly for applications where safety and security are paramount.2 The organization's guidance applies to a wide range of systems, including embedded control systems, software-intensive applications, and standalone software components.2

It is important to clarify that this organization and its guidelines are distinct from any other concepts that may share a similar name. For instance, research of the term "Mishra" reveals diverse meanings across various philosophical and cultural contexts, including in Hinduism and Jainism.8 These are entirely unrelated to the Motor Industry Software Reliability Association and its work, and any conflation of the terms is erroneous. The focus of MISRA's work is exclusively on engineering principles and software quality.

### 1.3 The Strategic Rationale for MISRA Compliance

The automotive industry's increasing reliance on software for safety-critical functions, such as anti-lock brakes and power steering, necessitated a robust coding standard to manage the growing complexity of these systems.2 Today, a single vehicle can contain over 100 million lines of code, a number projected to triple within the next decade.2 This immense scale and the severe consequences of a software malfunction underscore the need for a standardized approach to development.2

Although born in the automotive sector, MISRA's guidelines have gained broad acceptance as a benchmark for best practices in numerous other industries.2 Its adoption extends to aerospace, defense, telecommunications, medical devices, and railway systems, among others.2 The core benefit of adopting MISRA is that it defines a "safe subset" of the C and C++ languages, thereby eliminating constructs and behaviors that are prone to failure, undefined behavior, or security vulnerabilities.3 By doing so, it facilitates the creation of code that is more reliable, secure, portable, and easier to maintain throughout the software supply chain.2

The adoption of MISRA serves as a foundational step in a comprehensive strategy to reduce the cost of defects and satisfy industry process standards like ISO 26262 and IEC 61508.2 This shift-left approach to quality assurance ensures that potential issues are identified and addressed earlier in the development process, a practice that is significantly more cost-effective than delaying until later stages.6

## Chapter 2: Deciphering the MISRA Guidelines

### 2.1 Rules, Directives, and Decidability

The MISRA C:2012 standard introduced a crucial distinction between two types of guidelines: rules and directives.3 A

**rule** is a coding guideline with a complete and unambiguous description of its requirement. The source code can, in most cases, be automatically checked for compliance with a rule without needing additional information.13 A

**directive**, in contrast, is a guideline that is more open to interpretation or relates to process and procedural matters rather than concrete code syntax.6

A further classification, particularly relevant to scoping efforts, is **decidability**.6 A rule is considered

**decidable** if it can be conclusively verified by a static code analysis tool.6 For example,

MISRA C:2012 Rule 11.1 (which prohibits conversions between a function pointer and any other type) is a decidable rule.6 However, many rules, particularly those concerning complex semantic properties, are

**undecidable**.16 The inherent complexity of determining the semantic properties of code without executing it means that no algorithm can be guaranteed to definitively resolve the property check.18 This can lead to static analysis tools reporting false positives (flagging correct code as erroneous) or false negatives (missing actual issues), which can frustrate developers and compromise a project's quality assurance.18

The existence of undecidable rules has significant implications for a project's verification strategy. A traditional static analyzer may be unable to definitively prove the absence of a runtime error.19 Consequently, teams must supplement static analysis with other verification methods, such as manual code reviews and dynamic testing.18 However, advanced static analysis tools, particularly those powered by formal methods like abstract interpretation, can change this dynamic. These tools are capable of exhaustively proving the absence of certain runtime errors with no false positives. This transforms a subset of what were previously considered undecidable problems into decidable ones, thereby increasing diagnostic accuracy and reducing the burden of manual review and post-hoc testing.19 This capability is a critical factor when selecting a tool for a MISRA-compliant project.

### 2.2 The Guideline Categorization Framework

All MISRA guidelines are assigned to one of three categories, which define the strictness of compliance required.3

* **Mandatory:** These guidelines must be fully complied with at all times. Any violation of a mandatory rule renders the code non-compliant with the standard.2
* **Required:** Compliance with a required guideline is mandatory unless a formal, documented deviation is provided.2 Deviations are only permitted if they can be justified in writing, with proof that they do not compromise the system's safety or security and that no acceptable workaround exists.2
* **Advisory:** These guidelines represent good practice, but compliance is less formal.3 While it is recommended to follow them where practical, they can be "disapplied" without a formal deviation, though this should still be recorded in project documentation.3

The following table, based on the MISRA Compliance:2020 document, provides a clear overview of the acceptable compliance statuses for each guideline category.14

| Guideline Category | Acceptable Compliance Statuses | Prohibited Statuses |
| --- | --- | --- |
| Mandatory | Compliant | Deviations, Violations, Disapplied, Not Supported |
| Required | Compliant, Deviations | Violations, Disapplied, Not Supported |
| Advisory | Compliant, Deviations, Violations, Disapplied, Not Supported | N/A |

### \2.3 The Unified Standard: The MISRA-AUTOSAR C++ Merge

The landscape of C++ coding standards has evolved considerably. The original MISRA C++:2008 standard, which was written for C++03, became outdated with the introduction of new language features in C++11 and later versions.20 To address this gap, the AUTOSAR consortium released its own set of C++ guidelines in 2017, which were based on C++14.20 This created a challenge for the supply chain, as different vendors might have been adhering to separate standards, leading to fragmentation and interoperability issues.2

In January 2019, MISRA and AUTOSAR announced a pivotal collaboration to merge their guidelines into a single, unified standard.2 The outcome is MISRA C++:2023, a standard that incorporates the best practices of both organizations and is aligned with modern language features in C++17 and its successors.13 This integrated standard provides a "single point of reference" for all developers across the supply chain, simplifying the process of defining and enforcing coding standards.2

This unification is a significant development for project scoping and management. It provides a stable, authoritative target for both acquirers and suppliers, eliminating the need to reconcile multiple, competing rule sets.20 It also provides a clear and stable target for tool vendors, allowing them to focus on a single, well-defined rule set, which benefits the entire ecosystem by improving tool accuracy and reliability.21

## Chapter 3: The Formal MISRA Compliance Process

### 3.1 The MISRA Compliance:2020 Framework

A claim of MISRA compliance is not simply a statement that code has been checked against the guidelines.5 The

MISRA Compliance:2020 document provides a definitive framework for what a credible claim of compliance entails, superseding previous ad-hoc approaches.4 The document emphasizes that compliance is a continuous process that must be integrated into a disciplined and documented software development process.4 This framework ensures that the application of MISRA is consistent and auditable, thereby increasing the credibility and trustworthiness of the final product.4

### 3.2 The Four Key Compliance Artifacts

To demonstrate formal compliance with the MISRA Compliance:2020 framework, a project must generate and maintain four key artifacts.13 These documents are essential for any internal or external audit and are therefore a critical part of the initial scoping effort.

* **The Guideline Enforcement Plan (GEP):** This document outlines the method used to verify compliance for each specific guideline.6 This includes specifying which rules will be checked by an automated static analysis tool, which will be subject to manual code review, and which will be enforced by a compiler.24
* **The Guideline Re-categorization Plan (GRP):** While not required, this document is crucial for projects with legacy code or unique requirements.7 It formally documents any optional changes to the default categories of rules. For example, a required rule might be re-categorized as advisory, or an advisory rule might be re-categorized as mandatory, based on the project's specific needs.14
* **The Deviations Report:** This document serves as a permanent record of all formal deviations from required guidelines.3 It is essential for an auditor to be able to review each deviation, its rationale, and its approval history.7
* **The Compliance Summary:** This is the primary and mandatory record of a project's overall compliance status.7 It provides a high-level overview of which guidelines are compliant, which have documented deviations, and which are in violation.22

### 3.3 The Formal Deviation Procedure

MISRA acknowledges that adhering to every guideline may be impractical or impossible in certain scenarios, such as when integrating third-party code that cannot be altered.2 In such cases, software engineers may formally deviate from a

**required** guideline while still being considered compliant with the standard.3 This freedom to deviate comes with a significant responsibility.4

A formal deviation must be thoroughly documented and justified.3 The deviation record should include, at a minimum: the specific guideline being violated, a clear description of the circumstances, a detailed rationale for why the deviation is necessary, and a risk analysis proving that the deviation will not have a negative impact on the system's safety or security.3 For projects with numerous deviations, tools with deviation management capabilities can simplify this process by tracking and documenting each instance with a unique identifier.26

## Chapter 4: Tools and Techniques for MISRA Testing and Scoping

### 4.1 The Central Role of Static Code Analysis

For projects of any significant size, MISRA compliance is not a task that can be accomplished through manual review.6 Reviewing hundreds of thousands or even millions of lines of code for violations is not a practical or efficient use of resources.6 Consequently, MISRA recommends the use of automated static code analysis tools to enforce the majority of its guidelines.3

A key principle of effective MISRA adoption is to "shift left" in the development lifecycle by checking for compliance as soon as code is written, and before it is submitted for code review or unit testing.6 This proactive approach is significantly less expensive and more efficient than delaying analysis until later stages of development.5 Automating the compliance reporting and documentation process is also a key factor in conserving time and resources.25

### 4.2 Evaluating and Selecting Static Analysis Tools

Choosing the right static analysis tool is a critical part of the scoping process. An effective tool should provide extensive rule coverage, produce accurate diagnostics with low false positive rates, and integrate seamlessly into existing development workflows.15 Many leading tools, such as Parasoft C/C++test and Helix QAC, can be integrated into developer IDEs and continuous integration/continuous delivery (CI/CD) pipelines, enabling constant, automated checks.13

For projects requiring formal certification, it is crucial to select a tool that has been independently certified for use in safety-critical software development.27 The following table provides a non-exhaustive overview of leading tools and their capabilities.

| Vendor | Tool | Supported MISRA Standards | Certifications & Key Features |
| --- | --- | --- | --- |
| AbsInt | Astrée | N/A | Proves absence of run-time errors with formal methods.3 |
| GrammaTech | CodeSonar | MISRA-C:2012, MISRA-C++:2008 3 | Deviation management and tracking.26 |
| LDRA | LDRA Testbed, LDRArules | MISRA C:2025 & predecessors, MISRA C++:2023 & predecessors 22 | Full software lifecycle support, members of MISRA working groups.22 |
| MathWorks | Polyspace | MISRA-C:2023, C:2012, C++:2023, C++:2008 17 | Supports static and dynamic analysis, can trace violations to a model.6 |
| Parasoft | C/C++test | MISRA C:2023, C:2012, C++:2023, C++:2008 13 | Integrates with CI/CD, provides automated compliance reports, and tracks historical data.13 |
| Perforce | Helix QAC | MISRA C:2012, C:2004, C:1998, C++:2008 3 | Independently certified for functional safety, provides risk prioritization and customizable reports.28 |
| QA Systems | QA-MISRA | MISRA C:2025, C:2023, C:2012, C++:2023, C++:2008 27 | Certified by SGS-TÜV SAAR GmbH, claims zero false negatives on syntactic rules.27 |

### 4.3 Scoping for Legacy vs. New Codebases

A significant challenge for many organizations is applying MISRA standards to large, existing codebases.7 Attempting to refactor all legacy code for full compliance can be a costly, time-consuming, and risky endeavor that may inadvertently introduce new defects.4

A pragmatic and effective scoping strategy for such projects involves an incremental approach.25 The first step is to perform an initial analysis and

**baseline** all existing violations, effectively marking them as "address later".11 Subsequently, the focus should be on ensuring that all

**new code** is fully compliant with the project's Guideline Enforcement Plan and Guideline Re-categorization Plan.7 Legacy code can then be addressed incrementally, with violations prioritized based on their risk severity.25 This methodical approach ensures that the highest-risk issues are mitigated first, providing a clear roadmap for gradual improvement without the burden of a complete rewrite.11

## Chapter 5: MISRA in a Broader Context

### 5.1 Interoperability with Other Standards

It is important to understand that MISRA is a **coding standard**, which differs from **functional safety standards** such as ISO 26262 (automotive), IEC 61508 (industrial), and IEC 62304 (medical devices).2 MISRA guidelines provide a proven and well-defined mechanism for achieving code quality, which is a required element within the broader framework of these process-based functional safety standards.7 MISRA focuses on coding security and reliability, while standards like ISO 26262 provide an overarching framework for functional safety, including risk classification (e.g., ASILs).2

MISRA is also often used in conjunction with other complementary coding standards. For example, MISRA guidelines can be combined with CERT C/C++ rules, which specifically address a broader range of security vulnerabilities.6 This layered approach allows organizations to achieve both safety and security goals simultaneously.

### 5.2 Cross-Industry Adoption and Case Studies

The widespread adoption of MISRA beyond its automotive origins demonstrates its effectiveness as a cross-industry benchmark for software quality.2

* **Aerospace & Defense:** MISRA C:1998 served as the foundation for the C++ Coding Standards used in the Joint Strike Fighter project, and NASA's Jet Propulsion Laboratory C Coding Standards are based on MISRA C:2004.3 In fact, MISRA-compliant code has even "left the planet in code in space vehicles," a testament to its reliability in extreme environments.33
* **Medical Devices:** The medical device industry, which is governed by standards like IEC 62304, widely utilizes MISRA guidelines to ensure the safety, security, and quality of embedded software.2 This is a critical factor for manufacturers seeking regulatory approval and ensuring patient safety.13

### 5.3 Challenges and Future Directions

Despite its benefits, the adoption of MISRA can present challenges. Developers may lack motivation or understanding of the rationale behind certain rules, which can lead to resistance.11 This can be addressed through proper training and by providing access to the detailed descriptions and reasoning behind the guidelines.4

The MISRA Consortium continues to evolve the standard to address new language features and security threats.2 Furthermore, the principles behind MISRA are being explored for application in other programming languages, such as Rust.11 The potential for new technologies, such as machine learning, to assist in compliance is also a topic of discussion. While machine learning is not expected to replace developer judgment in safety-critical contexts, it could potentially aid in prioritizing violations and suggesting fixes, thereby streamlining the process.11

## Conclusion: A Roadmap to Robust Software

Scoping a project for MISRA compliance is a critical exercise that defines a project's approach to software quality and risk management. This process requires more than a simple tool selection; it necessitates a formal, well-documented strategy.

A successful MISRA scoping effort should follow a clear roadmap:

1. **Integrate the Process:** MISRA compliance must be integrated into the software development lifecycle from the outset, not treated as a final-stage check.4
2. **Select the Right Tools:** Choose a static analysis tool that provides comprehensive coverage, integrates with your CI/CD pipeline, and can manage the formal deviation process.3 For projects requiring certification, select a tool that has been independently qualified or certified by a recognized body.27
3. **Define the Compliance Framework:** Formally define the project's approach to compliance by creating the four key artifacts: the Guideline Enforcement Plan, the Guideline Re-categorization Plan, the Deviations Report, and the Compliance Summary.7
4. **Establish a Pragmatic Strategy:** For projects with legacy code, implement a strategy of baselining existing violations and focusing on full compliance for all new code. Prioritize the remediation of legacy violations based on risk severity to ensure the most impactful issues are addressed first.11
5. **Understand the Broader Context:** Recognize that MISRA is a coding standard that supports, but is distinct from, broader functional safety and security standards.2 A comprehensive verification strategy should include a combination of static analysis, manual reviews, and dynamic testing to address all aspects of the code, including the nuances of undecidable rules.18

By following this structured approach, an organization can effectively scope a MISRA project, ensuring that the software developed is not only compliant but also demonstrably safe, secure, and reliable.

#### Works cited

1. en.wikipedia.org, accessed September 16, 2025, <https://en.wikipedia.org/wiki/Motor_Industry_Software_Reliability_Association>
2. What is MISRA (Motor Industry Software Reliability Association)? - Synopsys, accessed September 16, 2025, <https://www.synopsys.com/glossary/what-is-misra.html>
3. MISRA C - Wikipedia, accessed September 16, 2025, <https://en.wikipedia.org/wiki/MISRA_C>
4. MISRA Compliance:2020, accessed September 16, 2025, <https://misra.org.uk/app/uploads/2021/06/MISRA-Compliance-2020.pdf>
5. MISRA Compliance 2020 | PDF | Verification And Validation | Compiler - Scribd, accessed September 16, 2025, <https://www.scribd.com/document/843288312/MISRA-Compliance-2020>
6. What Is MISRA C? - MATLAB & Simulink, accessed September 16, 2025, <https://www.mathworks.com/discovery/misra-c.html>
7. MISRA: ISO 26262 Software Compliance - Parasoft, accessed September 16, 2025, <https://www.parasoft.com/learning-center/iso-26262/misra/>
8. Mishra: Significance and symbolism, accessed September 16, 2025, <https://www.wisdomlib.org/concept/mishra>
9. Vāchaspati Misra - Wikipedia, accessed September 16, 2025, <https://en.wikipedia.org/wiki/V%C4%81chaspati_Misra>
10. Whitepaper: How to Comply With the MISRA Standard - Automotive IQ, accessed September 16, 2025, <https://www.automotive-iq.com/electrics-electronics/whitepapers/whitepaper-how-to-comply-with-the-misra-standard>
11. MISRA Q&A: Frequently Ask Questions and Answers - Qt, accessed September 16, 2025, <https://www.qt.io/quality-assurance/blog/misra-qa>
12. MISRA | QA Systems, accessed September 16, 2025, <https://www.qa-systems.com/solutions/misra/>
13. MISRA C Compliance Most Extensive Coverage - Parasoft, accessed September 16, 2025, <https://www.parasoft.com/solutions/misra/>
14. MISRA Coding Standards and Compliance - PVS-Studio, accessed September 16, 2025, <https://pvs-studio.com/en/docs/manual/6966/>
15. MISRA Compliance:2020 — MISRA Rules and MISRA Guidelines - Perforce Software, accessed September 16, 2025, <https://www.perforce.com/blog/qac/misra-rules-misra-guidelines>
16. Code review MISRA 2023 rules - HCL Product Documentation, accessed September 16, 2025, <https://help.hcl-software.com/devops/test/embedded/9.0.0.1/docs/topics/r_misra_2023.html>
17. Decidable MISRA Coding Rules Supported by Polyspace Bug Finder - MATLAB & Simulink, accessed September 16, 2025, <https://www.mathworks.com/help/bugfinder/ug/statically-enforceable-misra-c-2012.html>
18. MISRA C++:2023 Guidelines: The Intricacies of Undecidability - Qt, accessed September 16, 2025, <https://www.qt.io/quality-assurance/blog/misra-guidelines-undecidability>
19. MISRA Compliance - Decidable Rules & Static Analysis - YouTube, accessed September 16, 2025, <https://www.youtube.com/shorts/NeMMlyl0_L8>
20. MISRA C++ 2023 Guide: Everything You Need to Know - Parasoft, accessed September 16, 2025, <https://www.parasoft.com/blog/misra-cpp-2023-guide/>
21. Migration from MISRA C++:2008 or AUTOSAR to MISRA C++:2023 - Qt, accessed September 16, 2025, <https://www.qt.io/quality-assurance/blog/migration-to-misra-c2023>
22. Automate MISRA guidelines compliance - LDRA, accessed September 16, 2025, <https://ldra.com/wp-content/uploads/ldra/MISRA_Data_Sheet_v2.2.pdf>
23. MISRA Compliance - Confluence Mobile - Parasoft Documentation, accessed September 16, 2025, <https://docs.parasoft.com/display/DTP20231/MISRA+Compliance>
24. MISRA. Write justification for exception - Stack Overflow, accessed September 16, 2025, <https://stackoverflow.com/questions/63157159/misra-write-justification-for-exception>
25. MISRA C/C++ Code Checking - Parasoft, accessed September 16, 2025, <https://www.parasoft.com/blog/misra-c-c-code-checking/>
26. Managing MISRA Violations - CodeSecure, accessed September 16, 2025, <https://codesecure.com/learn/managing-misra-violations/>
27. QA-MISRA: Static Code Analysis for C/C++ - QA Systems, accessed September 16, 2025, <https://www.qa-systems.com/tools/qa-misra/>
28. Best Static Code Analyzer for Functional Safety and Standards Compliance - Perforce Software, accessed September 16, 2025, <https://www.perforce.com/sites/default/files/pdfs/datasheet-helix-qac-overview.pdf>
29. Accelerating embedded software verification with Polyspace static code analysis | MATLAB EXPO, accessed September 16, 2025, <https://www.matlabexpo.com/content/dam/mathworks/mathworks-dot-com/images/events/matlabexpo/uk/2019/accelerating-embedded-software-verification-with-polyspace-static-code-analysis.pdf>
30. MISRA C & MISRA C++ | Coding Standards For Compliance - Perforce Software, accessed September 16, 2025, <https://www.perforce.com/resources/qac/misra-c-cpp>
31. Check for and Review Coding Standard Violations - MATLAB & Simulink - MathWorks, accessed September 16, 2025, <https://www.mathworks.com/help/bugfinder/ug/check-for-coding-rule-violations.html>
32. Coding standards compliance for embedded systems - LDRA, accessed September 16, 2025, <https://ldra.com/capabilities/coding-standard-compliance/>
33. MISRA C 25th Anniversary, accessed September 16, 2025, <https://misra.org.uk/misra-c-silver-anniversary/>