PARASOFT®

WHITEPAPER

A Practical Guide to Accelerate MISRA C 2023 Compliance With Test Automation

INTRODUCTION

The challenges of building embedded software applications continue to increase. Demands to incorporate competencies in network connectivity, artificial intelligence (AI), and other performance improvements have increased code complexity and, in turn, heightened the risk of security vulnerabilities and safety hazards. To help mitigate these challenges, many organizations look to the Motor Industry Software Reliability Association (MISRA) coding guidelines.

MISRA C 2023 guidelines for the use of the C language in critical systems is a definition of a subset of the C language designed to minimize potential errors, security vulnerabilities, and mistakes that cause program failure or serious errors. Although initially created for automotive systems, the guidelines have been adopted by many other industries for safety-critical applications. In all cases, determining compliance, which is increasingly required in vendor/client relationships, can be complex and time-consuming if it's not automated.

There are many tools on the market that report errors indicating when code is violating a guideline. However, few tools simplify the reporting and documentation process required for demonstrating compliance—never mind implementing the guidelines into your software development process!

Test verification and validation solutions like Parasoft's make reaching MISRA compliance quicker and with less cost. The solution streamlines the entire process of achieving, documenting, and maintaining MISRA compliance throughout the product life cycle. It also streamlines other automated testing methods like unit, regression, integration, system, structural code coverage, and more.

LEGACY & EXISTING CODE

Reuse of code is a reality for every project. However, reusing legacy code in a safety-critical software project and achieving full MISRA C 2023 compliance is a daunting task.

The original MISRA principles were created to be applied as code being developed. Even the standard itself provides a warning.

> ***"...a project that checks for MISRA C compliance late in its cycle is likely to spend a considerable amount of time re-coding, re-reviewing and re-testing. It's therefore expected that the software development process will require the early application of MISRA C principles."***

Because many organizations do need to reuse their legacy codebases for business reasons, the MISRA Compliance: 2020 guidance document was created in response to these challenges. In it, there's a clear distinction between the new, native code that's developed in the scope of a current project and the adopted code that's developed outside of the scope of the project. In this paper, we explain a practical approach to dealing with legacy code and MISRA C compliance.

SHADES OF GRAY

Although it seems like it should be simple to understand what type of code you're dealing with, the situation in many cases isn't black and white. For example, an initial prototype that was developed without following MISRA guidelines is productized, and then management realizes that compliance is a requirement for the intended marketplace.

Typically, the legacy codebase was never developed with any coding guidelines in mind. Therefore, a codebase cannot be automatically classified as adopted code if updates are required in the context of a new project adding to the complexity of the situation.

All too often, initial scans of a large codebase via a static analysis tool with all MISRA C 2023 rules enabled, including the latest amendments to the standard, produces tens of thousands of violations. After the initial shock, teams begin to find creative ways of addressing the violations. It's important for development teams not to be deterred. There is light at the end of the tunnel.

Over time, we've collected and identified best practices and approaches that development teams have used to make the code compliant while not interfering with the ongoing development velocity. In this paper, we share some practical, balanced approaches to make existing codebases MISRA compliant.

FOLLOW THE GUIDANCE IN THE MISRA COMPLIANCE: 2020 FRAMEWORK

MISRA C 2023 is a set of coding guidelines for the C programming language. The focus of the standard is increasing safety of software by preemptively 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. However, MISRA continues to publish amendments to its latest MISRA C 2023 coding guidelines to mitigate the growing risk of cyber security coding vulnerabilities.

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- and security- critical software, the value of applying the standard to a project depends on factors like the following.

» Risk of a system malfunction because of a software failure.

» Risk of system exploit because of a software vulnerability.

» 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.

In the document, MISRA Compliance: 2020, the MISRA Consortium provides the response that was needed by the community, with a reasonably well-defined framework of what the phrase **"MISRA compliant"** truly means.

To claim MISRA compliance, it's required to establish:

» Use of a disciplined software development process.

» Exactly which guidelines are being applied.

» The effectiveness of the enforcement methods.

» The extent of any deviations from the guidelines.

» The status of any software components developed outside of the project.

PROOF OF COMPLIANCE: THE END GOAL

A key problem that developers of safety-critical software encounter is how to demonstrate and prove compliance at the end of the project. There's a tendency to add more information into the reports than is required. It can become a contentious issue, resulting in wasted time and effort if the evaluation criteria are based on subjective opinions from the various stakeholders.

A recommended approach to improving the evaluation of compliance readiness is to use existing templates for both the final compliance and tool qualification report. If the information is not required by the standard, avoid adding it. Combining extra information is not only a waste of time, but also introduces a risk of delaying an audit process. Having the documentation auto generated, as Parasoft does, is the ultimate solution.

The MISRA Compliance: 2020 document is also helping organization to use a common language articulating the compliance requirements by defining the following artifacts.

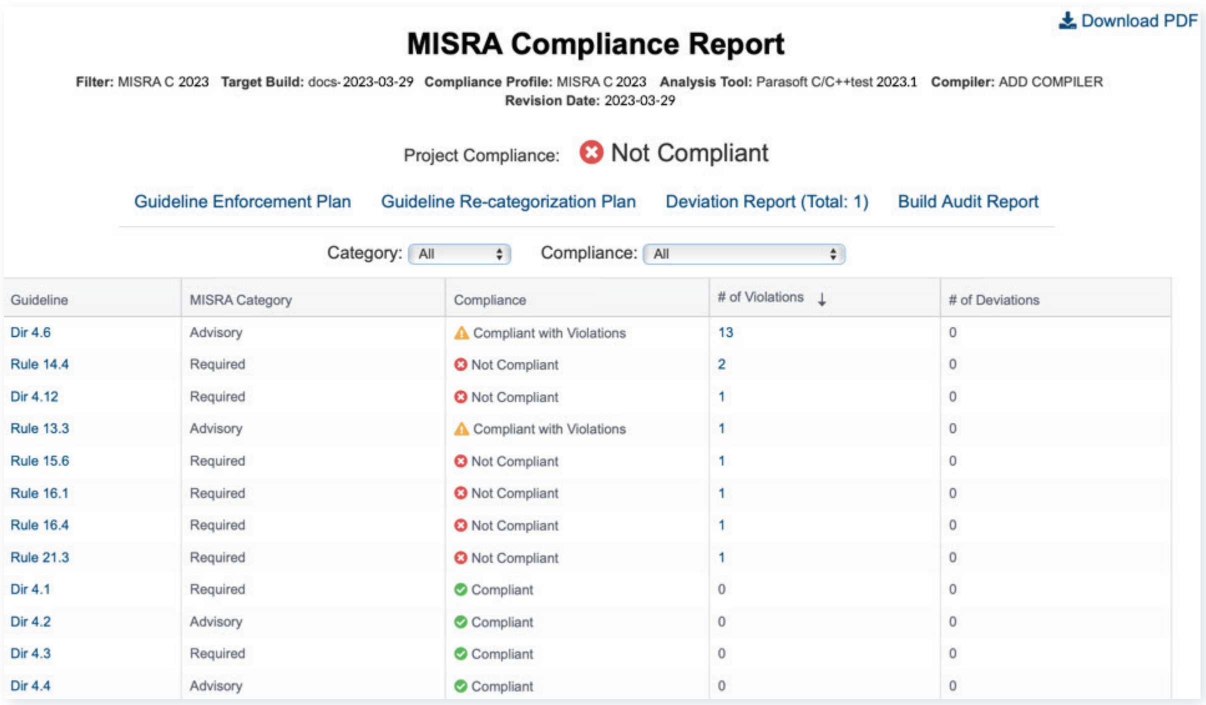
» Guidelines compliance summary

» Guideline enforcement plan

» Deviations report

» Guideline recategorization plan

The following screenshots show auto-generated reports with links to other records and/or expansion of information on the page.



Download PDF

MISRA Compliance Report

Filter: MISRA C 2023 **Target Build:** docs-2023-03-29 **Compliance Profile:** MISRA C 2023 **Analysis Tool:** Parasoft C/C++test 2023.1 **Compiler:** ADD COMPILER
Revision Date: 2023-03-29

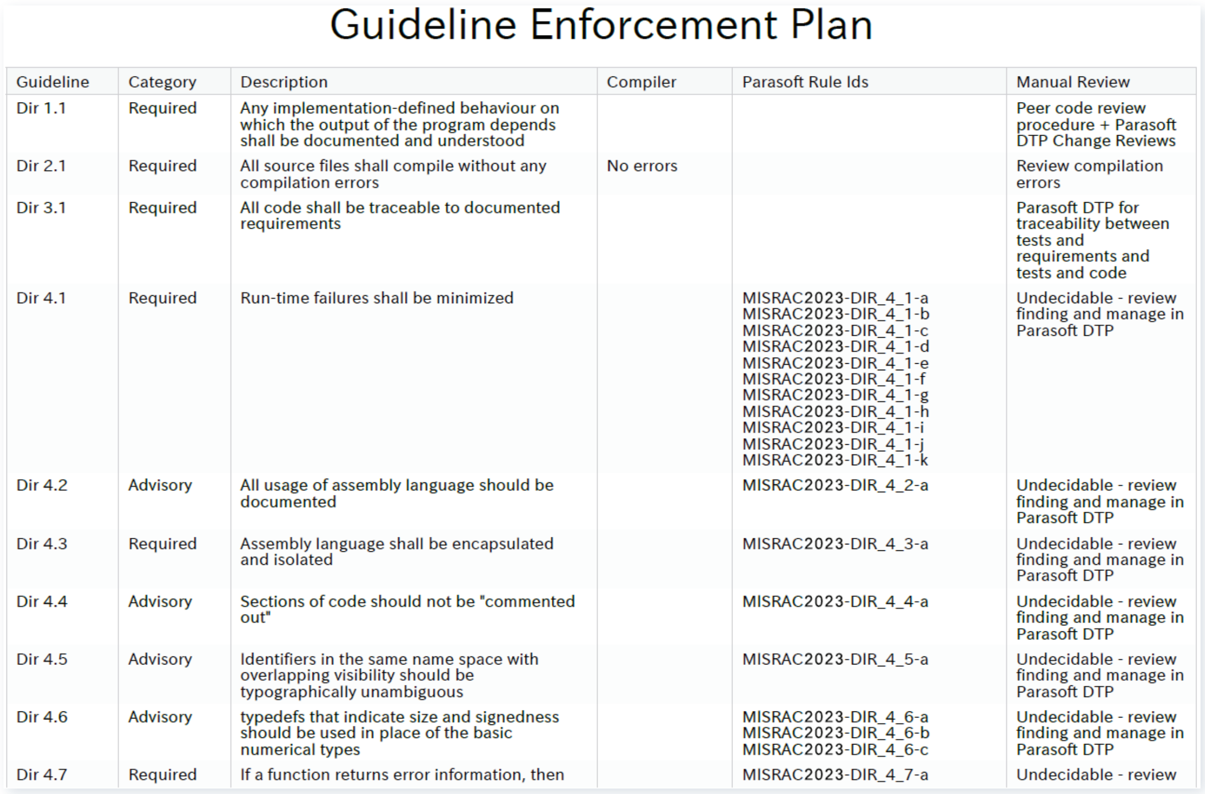
Project Compliance: ✗ Not Compliant

Guideline Enforcement Plan | Guideline Re-categorization Plan | Deviation Report (Total: 1) | Build Audit Report

Category: All Compliance: All

Guideline	MISRA Category	Compliance	# of Violations ↓	# of Deviations
Dir 4.6	Advisory	⚠ Compliant with Violations	13	0
Rule 14.4	Required	✗ Not Compliant	2	0
Dir 4.12	Required	✗ Not Compliant	1	0
Rule 13.3	Advisory	⚠ Compliant with Violations	1	0
Rule 15.6	Required	✗ Not Compliant	1	0
Rule 16.1	Required	✗ Not Compliant	1	0
Rule 16.4	Required	✗ Not Compliant	1	0
Rule 21.3	Required	✗ 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

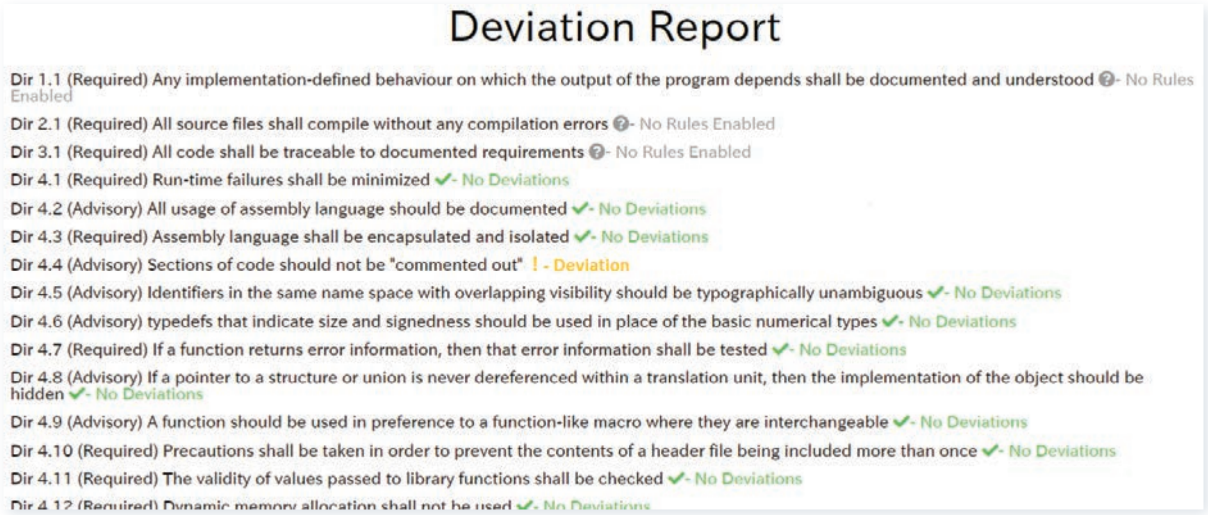
Figure 1: Parasoft's Guidelines Compliance Report summary is the primary record of overall project compliance.



Guideline Enforcement Plan

Guideline	Category	Description	Compiler	Parasoft Rule Ids	Manual Review
Dir 1.1	Required	Any implementation-defined behaviour on which the output of the program depends shall be documented and understood			Peer code review procedure + Parasoft DTP Change Reviews
Dir 2.1	Required	All source files shall compile without any compilation errors	No errors		Review compilation errors
Dir 3.1	Required	All code shall be traceable to documented requirements			Parasoft DTP for traceability between tests and requirements and tests and code
Dir 4.1	Required	Run-time failures shall be minimized		MISRAC2023-DIR_4_1-a MISRAC2023-DIR_4_1-b MISRAC2023-DIR_4_1-c MISRAC2023-DIR_4_1-d MISRAC2023-DIR_4_1-e MISRAC2023-DIR_4_1-f MISRAC2023-DIR_4_1-g MISRAC2023-DIR_4_1-h MISRAC2023-DIR_4_1-i MISRAC2023-DIR_4_1-j MISRAC2023-DIR_4_1-k	Undecidable - review finding and manage in Parasoft DTP
Dir 4.2	Advisory	All usage of assembly language should be documented		MISRAC2023-DIR_4_2-a	Undecidable - review finding and manage in Parasoft DTP
Dir 4.3	Required	Assembly language shall be encapsulated and isolated		MISRAC2023-DIR_4_3-a	Undecidable - review finding and manage in Parasoft DTP
Dir 4.4	Advisory	Sections of code should not be "commented out"		MISRAC2023-DIR_4_4-a	Undecidable - review finding and manage in Parasoft DTP
Dir 4.5	Advisory	Identifiers in the same name space with overlapping visibility should be typographically unambiguous		MISRAC2023-DIR_4_5-a	Undecidable - review finding and manage in Parasoft DTP
Dir 4.6	Advisory	typedefs that indicate size and signedness should be used in place of the basic numerical types		MISRAC2023-DIR_4_6-a MISRAC2023-DIR_4_6-b MISRAC2023-DIR_4_6-c	Undecidable - review finding and manage in Parasoft DTP
Dir 4.7	Required	If a function returns error information, then		MISRAC2023-DIR_4_7-a	Undecidable - review

Figure 2: Parasoft's Guideline Enforcement Plan demonstrates how each MISRA guideline is verified.



Deviation Report

Dir 1.1 (Required) Any implementation-defined behaviour on which the output of the program depends shall be documented and understood ? - No Rules Enabled

Dir 2.1 (Required) All source files shall compile without any compilation errors ? - No Rules Enabled

Dir 3.1 (Required) All code shall be traceable to documented requirements ? - No Rules Enabled

Dir 4.1 (Required) Run-time failures shall be minimized ✓- No Deviations

Dir 4.2 (Advisory) All usage of assembly language should be documented ✓- No Deviations

Dir 4.3 (Required) Assembly language shall be encapsulated and isolated ✓- No Deviations

Dir 4.4 (Advisory) Sections of code should not be "commented out" ! - Deviation

Dir 4.5 (Advisory) Identifiers in the same name space with overlapping visibility should be typographically unambiguous ✓- No Deviations

Dir 4.6 (Advisory) typedefs that indicate size and signedness should be used in place of the basic numerical types ✓- No Deviations

Dir 4.7 (Required) If a function returns error information, then that error information shall be tested ✓- No Deviations

Dir 4.8 (Advisory) If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden ✓- No Deviations

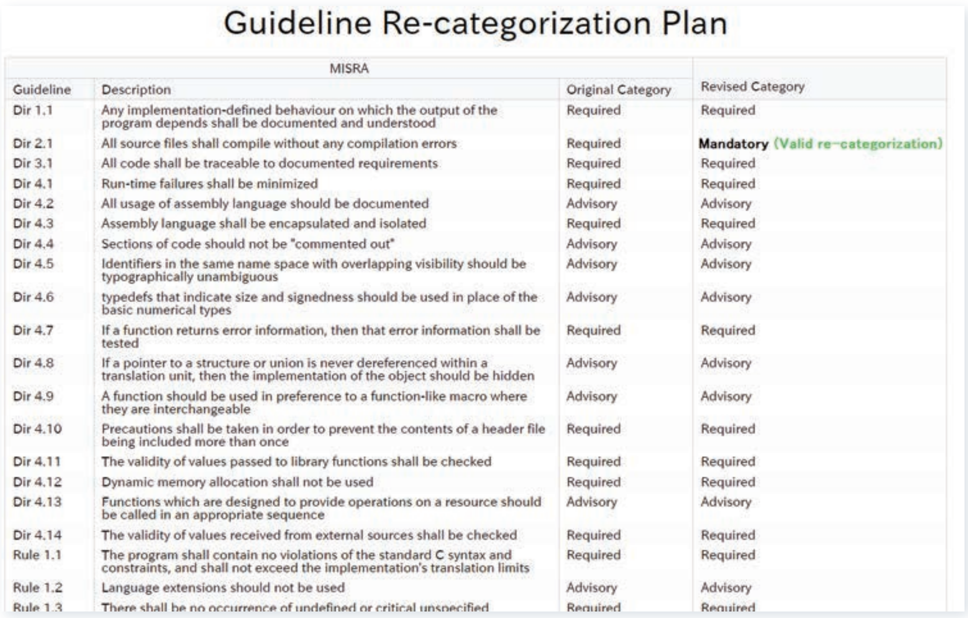
Dir 4.9 (Advisory) A function should be used in preference to a function-like macro where they are interchangeable ✓- No Deviations

Dir 4.10 (Required) Precautions shall be taken in order to prevent the contents of a header file being included more than once ✓- No Deviations

Dir 4.11 (Required) The validity of values passed to library functions shall be checked ✓- No Deviations

Dir 4.12 (Required) Dynamic memory allocation shall not be used ✓- No Deviations

Figure 3: Parasoft's Deviation Report documents all approved deviation permits.



Guideline Re-categorization Plan

MISRA			
Guideline	Description	Original Category	Revised Category
Dir 1.1	Any implementation-defined behaviour on which the output of the program depends shall be documented and understood	Required	Required
Dir 2.1	All source files shall compile without any compilation errors	Required	**Mandatory** (Valid re-categorization)
Dir 3.1	All code shall be traceable to documented requirements	Required	Required
Dir 4.1	Run-time failures shall be minimized	Required	Required
Dir 4.2	All usage of assembly language should be documented	Advisory	Advisory
Dir 4.3	Assembly language shall be encapsulated and isolated	Required	Required
Dir 4.4	Sections of code should not be "commented out"	Advisory	Advisory
Dir 4.5	Identifiers in the same name space with overlapping visibility should be typographically unambiguous	Advisory	Advisory
Dir 4.6	typedefs that indicate size and signedness should be used in place of the basic numerical types	Advisory	Advisory
Dir 4.7	If a function returns error information, then that error information shall be tested	Required	Required
Dir 4.8	If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden	Advisory	Advisory
Dir 4.9	A function should be used in preference to a function-like macro where they are interchangeable	Advisory	Advisory
Dir 4.10	Precautions shall be taken in order to prevent the contents of a header file being included more than once	Required	Required
Dir 4.11	The validity of values passed to library functions shall be checked	Required	Required
Dir 4.12	Dynamic memory allocation shall not be used	Required	Required
Dir 4.13	Functions which are designed to provide operations on a resource should be called in an appropriate sequence	Advisory	Advisory
Dir 4.14	The validity of values received from external sources shall be checked	Required	Required
Rule 1.1	The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation's translation limits	Required	Required
Rule 1.2	Language extensions should not be used	Advisory	Advisory
Rule 1.3	There shall be no occurrence of undefined or critical unspecified	Required	Required

Figure 4: Parasoft's Guideline Re-categorization Plan communicates how the guidelines are to be applied as part of the stakeholder/supplier relationship.

START BY ESTABLISHING THE END GOALS

Work with the stakeholder and establish the Guideline Re-categorization Plan (GRP) at the beginning of the project. It's possible that there could be several GRPs.

» A GRP for new code being written during the life of the project to ensure a high level of code quality is delivered.

» Another GPR for the use of legacy code that contains a different set of expectations because it may have proven itself out in the field for many years.

Also take the following into account.

1. For every incremental change, is there visibility into how many items of work are remaining to get to full compliance? This helps to plan the work accordingly and set the right expectations with the management.
2. Have the Guidelines Compliance Summary (GPS) report templates been reviewed with the acquirer at the beginning of the project? Were they acceptable and complete?

There are templates for these documents that commercial static analysis tool vendors, including Parasoft, provide to help organizations satisfy the MISRA 2020 Compliance framework.

TAKE A PHASED APPROACH: DIVIDE & CONQUER

The initial scan of the existing code by a static analysis tool tends to produce thousands of violations, particularly when using a default set of rules. It's impractical to stop new development efforts to focus on fixing all these identified violations. In fact, we've seen cases where regressions were introduced when significant changes to the codebase were made to fix the static analysis violations. Therefore, it's important to establish a workflow to fix the violations over time without disrupting the development process or degrading the quality of the software.

Here are key recommendations for using static analysis tools for the first time in a project.

» **Baselining.** After the initial scan of the code, mark all the initial violations as "to be addressed later" and set as a baseline. From that point forward, when updating existing code and/or developing new code, maintain a policy of "no new violations allowed."

  This policy can be enforced by the static analysis tool, applying a code review process or a continuous integration (CI) tool like GitHub, GitLab, Azure DevOps, Jenkins, or others. When developers have a few hours or days to spare, they can resolve remaining violations marked from the baseline. Organizations can prioritize this approach based on the following.

  » Current code under development

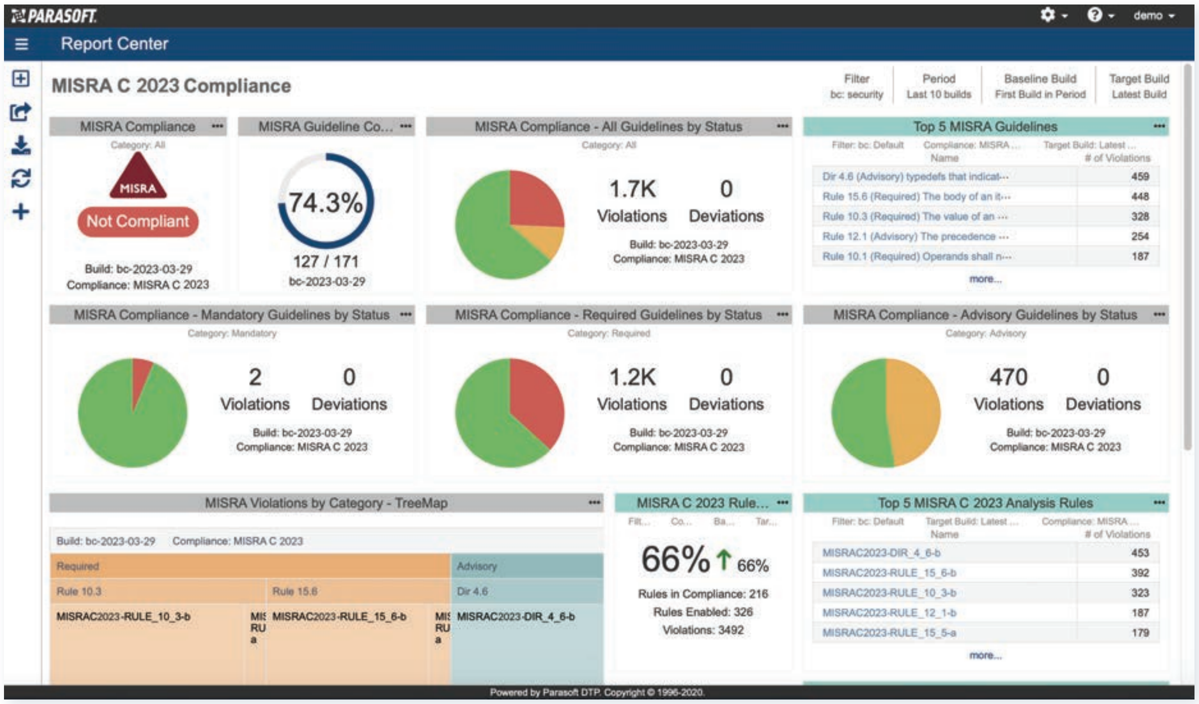
  » Code review findings

  » Relying on metrics, like severity, to suggest the next violation to resolve

» **Line in the sand.** The development sets a date—the line in the sand. After this date, every translation unit (individual source file) modified must have all violations addressed. All unmodified translation units automatically fall under the true adopted code definition from the MISRA compliance document.

» **Severity based prioritization.** The developer fixes all mandatory findings for the module assigned to them. Over time, they address all Required violations as time permits based on a priority selected by the team lead. Parasoft uses AI and machine learning to learn from this prioritization process and can take ownership of this task based on real world historical behavior.

For any of the approaches described above, it's important for technical leads and management to constantly monitor the progress and project compliance status via a centralized dashboard. For example, Parasoft's reporting hub provides the following pre-configured compliance status dashboard.



Top 5 MISRA Guidelines

Filter: bc: Default Compliance: MISRA ... Target Build: Latest ...

Name	# of Violations
Dir 4.6 (Advisory) typedefs that indicat···	459
Rule 15.6 (Required) The body of an it···	448
Rule 10.3 (Required) The value of an ···	328
Rule 12.1 (Advisory) The precedence ···	254
Rule 10.1 (Required) Operands shall n···	187

more...

Top 5 MISRA C 2023 Analysis Rules

Filter: bc: Default Target Build: Latest ... Compliance: MISRA ...

Name	# of Violations
MISRAC2023-DIR_4_6-b	453
MISRAC2023-RULE_15_6-b	392
MISRAC2023-RULE_10_3-b	323
MISRAC2023-RULE_12_1-b	187
MISRAC2023-RULE_15_5-a	179

more...

Figure 5:
Parasoft DTP – Reporting and analytics compliance dashboard

STAFF COMPETENCY & TRAINING

Today, development teams employ coding standards as a method to define, manage, and utilize a group of coding practices, with the prime objective being consistency and ensuring a baseline for code quality. The main aim of a coding rule is to limit use of the language so it prevents the developer from doing things that are "wrong" and can be potentially dangerous. Developers can avoid many defects in software by adopting sensible language-use restrictions. This results in style uniformity, which is valuable as a discipline in software projects.

The versatility aspect of the C++ and C languages allows developers to write code that can be unintentionally incorrect and possibly dangerous. It's easy for him or her to write code that adheres to the requirements of the language's standard but this, nonetheless, can result in undesirable behavior and program crashes. These include, for example, code that accesses memory beyond the parameters of an arithmetic process or of an array that results in memory or boundary violations.

It's obviously important to identify these potential problems. But the aim of MISRA is to prevent problems, not simply identify them. A compiler can detect some, but using a dedicated analysis tool is more effective.

The MISRA standard emphasizes that adhering to coding rules is only one component to developing software successfully. Developers must integrate each programming project into a disciplined engineering setting that includes methodical development workflows and apply proven in-use validation tools.

The expertise and training of the development staff are key factors often overlooked by software organizations and frequently identified by auditors as the number one issue when evaluating the readiness of a product.

According to MISRA guidelines, staff competency is an important part of the MISRA compliance. It's best to conduct training at the beginning of the project and have a training date recorded with all the developers signing off that they received the training. At the end of the project, the development team should be able to prove the following.

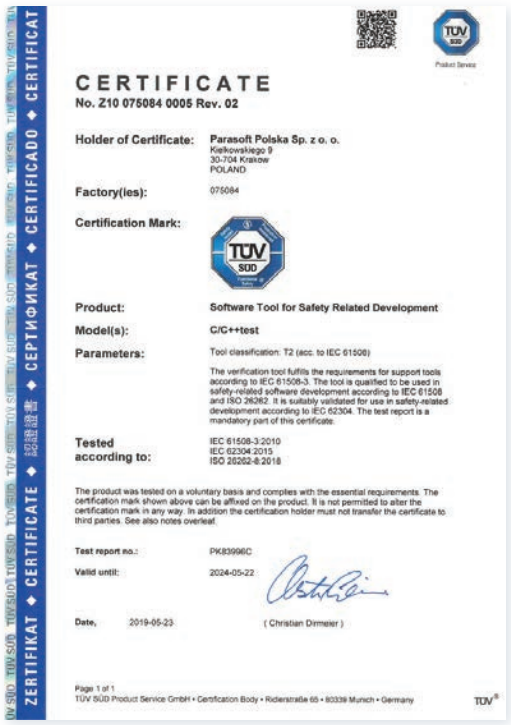
» Staff that approves deviations understands and has been trained to recognize the risks associated with the violation.

» Staff has been trained to properly configure and use the static analysis and development tools prior to their use.

In practice, training for an experienced team is relatively short. A few days invested in the beginning of the project saves weeks of rework, quality issues down the road and missed project deadlines.

TOOL QUALIFICATION

A less obvious component of MISRA compliance, often left until the end of the project, is the qualification of the development tools used in the product, and proving they're fit for purpose according to the pertinent safety standard. If a tool needs qualification, what level of validation needs to be performed?



ZERTIFIKAT ◆ CERTIFICATE ◆ 認證證書 ◆ СЕРТИФИКАТ ◆ CERTIFICADO ◆ CERTIFICAT

Product Service

CERTIFICATE
No. Z10 075084 0005 Rev. 02

Holder of Certificate:	Parasoft Polska Sp. z o. o. Kielkowskiego 9 30-704 Krakow POLAND
Factory(ies):	075084
Certification Mark:	
Product:	Software Tool for Safety Related Development
Model(s):	C/C++test
Parameters:	Tool classification: T2 (acc. to IEC 61508) The verification tool fulfills the requirements for support tools according to IEC 61508-3. The tool is qualified to be used in safety-related software development according to IEC 61508 and ISO 26262. It is suitably validated for use in safety-related development according to IEC 62304. The test report is a mandatory part of this certificate.
Tested according to:	IEC 61508-3:2010 IEC 62304:2015 ISO 26262-8:2018

The product was tested on a voluntary basis and complies with the essential requirements. The certification mark shown above can be affixed on the product. It is not permitted to alter the certification mark in any way. In addition the certification holder must not transfer the certificate to third parties. See also notes overleaf.

Test report no.:	PK83996C
Valid until:	2024-05-22

[illegible]

Date, 2019-05-23 (Christian Dirmeier)

Page 1 of 1
TÜV SÜD Product Service GmbH • Certification Body • Ridlerstraße 65 • 80339 Munich • Germany

Figure 6:
Parasoft C/C++test TÜV SÜD Certificate

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

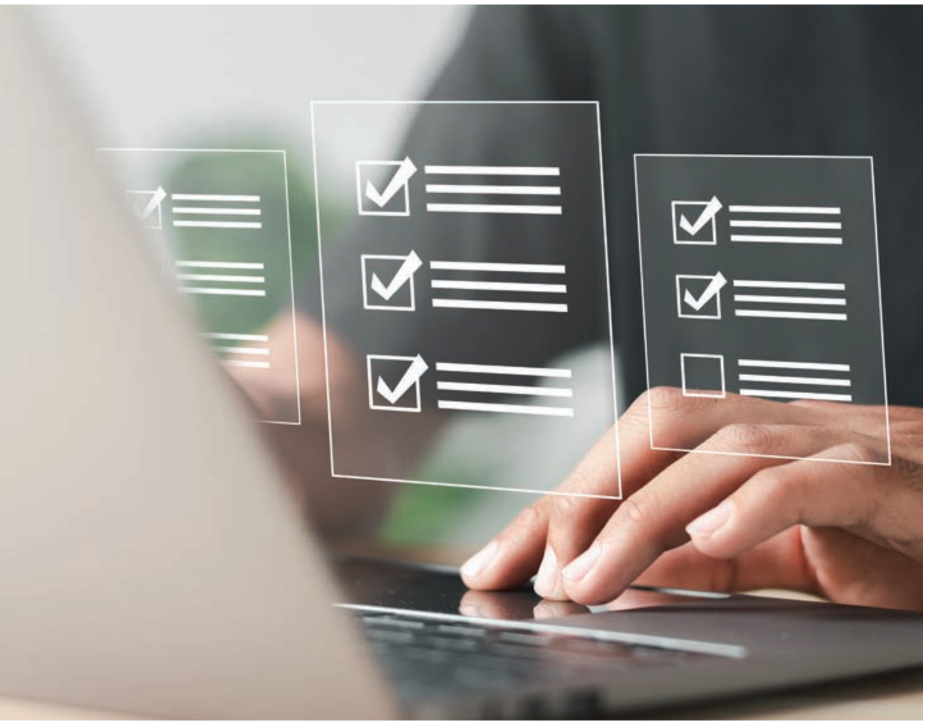
Parasoft C/C++test is certified by TÜV SÜD for functional safety according to IEC 61508, ISO 26262, and IEC 62304 standards for both host based and embedded target applications, paving the way for a streamlined qualification of static analysis, unit testing, and coverage requirements for the safety-critical standards.

While in many instances tool qualification is required, the method used to perform tool qualification varies depending on the risk associated with the tool malfunction and the software criticality level. Parasoft provides a qualification kit and certifications for specific safety standards and their requirements.

In the absence of this efficient kit, software teams must consider tool qualification costs when evaluating commercial, free, and open source tools. Some standards like DO-178C provide reasonable guidance on tool qualification requirements. Regardless of the method, the goal of the tool qualification process is to state "the tool is valid for its intended use" and provide a proof and rationale for how the team came to this conclusion.

REDUCE THE MANUAL EFFORT OF TOOL QUALIFICATION

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



A sensible way to reduce the workload is to bring automation to the process with qualification kits. Qualification kits walk the user through an intuitive workflow to dramatically reduce the amount of effort required. For example, here are the benefits of using Parasoft 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.

» Handle 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 or less!

SUMMARY

There's no silver bullet that makes achieving MISRA compliance easy in safety-critical projects. However, by introducing the MISRA Compliance 2020 framework and using a practical, phased approach with a clearly defined end point in mind, software development teams can achieve compliance without significantly disrupting their development process. The bottom line is that there's still a fair amount of work to achieve compliance, but automation goes a long way to reducing the tedious manual processes.

TAKE THE NEXT STEP

Talk to a compliance expert to learn how your development team can get the most extensive MISRA coverage.

ABOUT PARASOFT

Parasoft helps organizations continuously deliver quality software with its market-proven, integrated suite of automated software testing tools. Supporting the embedded, enterprise, and IoT markets, Parasoft's 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 delivers 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.