Developing high-quality software is tough. ECLAIR is designed to help development, QA, and safety teams reach their quality goals

Coverage of RTCA DO-178C

1 Introduction to RTCA DO-178C

DO-178C, "Software Considerations in Airborne Systems and Equipment Certification," is part of a series of international functional-safety standards published by RTCA for the aerospace industry. DO-178C covers aspects of certification related to the production of software for airborne systems and equipment used on aircraft, engines, propellers and auxiliary power units: it is the main reference used by certification authorities (such as FAA, EASA and Transport Canada) to approve all commercial software-based aerospace systems.

DO-178C prescribes that tools must be qualified when they are used to eliminate, reduce or automate processes mandated by DO-178C and the output of the tool is not verified manually or with another tool [8, Section 12.2.1]. The recommended approach is to follow the recommendations of RTCA DO-330 "Software Tool Qualification Considerations" [9].

DO-178C prescribes the allocation of *software levels* to software components. The software level establishes the rigor with which compliance with DO-178C has to be demonstrated, and it is based upon the contibution of the software to system failure conditions and their severity. There are five software levels: A, B, C, D and E [8, Section 2.3.3]. They are, respectively, for software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in

Level A: a catastrophic failure condition for the aircraft (multiple fatalities, loss of the airplane);

Level B: a hazardous failure condition for the aircraft (serious or fatal injury to a relatively small number of the occupants other than the flight crew, large reduction in safety margins or functional capabilities, flight crew physical distress or excessive workload);

Level C: a major failure condition for the aircraft (possible injuries, physical distress or discomfort to passengers, flight and cabin crew, significant reduction in safety margins);

Level D: a minor failure condition for the aircraft (physical discomfort to passengers or cabin crew, slight increase in crew workload, slight reduction in safety margins or functional capabilities);

Level E: no effect on aircraft operational capability or pilot workload.

For software components that are confirmed to be Level E according to DO-178C, DO-178C has no further prescription.

2 ECLAIR Support for DO-178C *Reviews and Analyses of Source Code*

The ECLAIR Software Verification Platform can be used to comply with several of the objectives of DO-178C Table A.5 [8, Annex A, page 100].

Table 1: Adapted from DO-178C Table A.5 — Verification of Outputs of Software Coding & Integration Processes

Objective		Software Level				ECLAIR
		A	B	C	D	
1	Source Code complies with low-level requirements	●	●	○		✓[a]
2	Source Code complies with software architecture	●	○	○		✓[b]
3	Source Code is verifiable	○	○			✓[c]
4	Source Code conforms to standards	○	○	○		✓[d]
5	Source Code is traceable to low-level requirements	○	○	○		✓[e]
6	Source Code is accurate and consistent	●	○	○		✓[f]
7	Output of software integration process is complete and correct	○	○	○		✓[g]
8	Parameter Data Item File is correct and complete	●	●	○	○	–
9	Verification of Parameter Data Item File is achieved	●	●	○		–

[a] Compliance to the MISRA C/C++ and the BARR-C:2018 guidelines greatly increases code readability and understandability, thereby facilitating verification activities by review and inspection.

[b] The *ECLAIR Independence Checker* (service `B.INDEPENDENCE`) allows the formal specification and systematic checking of software architectural constraints, e.g., to enforce constraints about layering and to prevent bypassing of software interfaces. `B.INDEPENDENCE` is instrumental in proving independence among different software components.

[c] Compliance with the MISRA C/C++ guidelines ensures that the source code complies to the applicable ISO C/C++ standards and is free from undefined and critical unspecified behavior, as well as other obstacles to verifiability of source code, such as the use of language extensions and assembly code.

[d] The MISRA C/C++ guidelines are the most autoritative coding standards for the respective languages. In addition, HIS [3] and other metrics can be used to define and enforce complexity restrictions and other code constraints such as those limiting the degree of coupling between software components. ECLAIR allows associating thresholds to each metric. The rigorously defined concept of *MISRA Compliance* [4], which is fully supported by ECLAIR, ensures that deviations to the applicable coding standards are properly justified.

[e] ECLAIR service for MISRA C:2025 Directive 3.1 allows ensuring that all code is forward and backward traceable to documented requirements, including safety requirements.

[f] Compliance with the MISRA C/C++ guidelines is instrumental in determining correctness and consistency of the source code, covering, e.g., limits to recursion to make stack usage predictable, proper access to resources, exception handling, use of uninitialized variables, and unused variables.

[g] Compliance with the MISRA C/C++ guidelines that are checked at system scope ensure early detection of several errors that can be committed in the software integration process.

2.1 MISRA C:2025

MISRA C:2025 [6] is the software development C subset developed by MISRA that is a de facto standard for safety-, life-, security-, and mission-critical embedded applications in many industries, including aerospace, railway, medical, telecommunications and others. MISRA C:2025, which allows coding MISRA-compliant applications in subsets of C90, C99, C11 and C18, is supported, along with all previous versions of MISRA C, by the ECLAIR package called "MC".

2.2 MISRA C++:2023

MISRA C++:2023 [5] is the software development C++ subset developed by MISRA, which is a de facto standard for safety-, life-, and mission-critical embedded applications in many industries including aerospace, railway, medical, telecommunications and others. MISRA C++:2023 completely supersedes MISRA C++:2008 [7], the previous edition of the coding standard, which is still used by many legacy projects. MISRA C++:2023 and MISRA C++:2008 are supported by the ECLAIR package called "MP".

2.3 BARR-C:2018

The *Barr Group's Embedded C Coding Standard*, BARR-C:2018 [2], is, for coding standards used by the embedded system industry, second only in popularity to MISRA C. BARR-C:2018 guidelines include 64 guidelines dealing with language subsetting and project management as well as 79 guidelines concerning programming style. For projects in which a MISRA compliance requirement is not (yet) present, the adoption of BARR-C:2018 is a major improvement with respect to the situation where no coding standards and no static analysis is used. The adoption of the stylistic subset of BARR-C:2018 (79 out of 143 rules) can be part of complying with the MISRA requirement that a consistent programming style is adopted and systematically used as part of the software development process. Moreover, complying with BARR-C:2018, besides avoiding many dangerous bugs, entails compliance with a non-negligible subset of MISRA C:2012 [1]. ECLAIR support for BARR-C:2018 has no equals on the market: it is included in all ECLAIR packages, including the affordable package "B".

2.4 HIS and Other Source Code Metrics

Source code metrics are recognized by many software process standards (and from MISRA) as providing an objective foundation to efficient project and quality management. One well known set of metrics has been defined by HIS (Herstellerinitiative Software, an interest group set up by Audi, BMW, Daimler, Porsche and Volkswagen).

The *HIS source code metrics* [3], while well established, include some metrics that are obsolete and miss others that are required or recommended by software process standards, such as those that allow estimating function coupling. For this reason, ECLAIR supplements HIS source code metrics with numerous other metrics that allow software quality to be assessed in terms of complexity, testability, readability, maintainability and so forth. Keeping track of these metrics also provides an effective and objective method to assess the quality of the software development process. The full set of metrics is available in all ECLAIR packages.

3 ECLAIR Support for DO-178C *Independence* and *Partitioning*

Section 2.4, "Architectural Considerations," of DO-178C [8] requires the system-safety assessment process to establish that "sufficient independence exists between software components." It then makes clear what the consequences are of being unable to prove independence:

> If partitioning and independence between software components cannot be demonstrated, the software components should be viewed as a single software component when assigning

software levels (that is, all components are assigned the software level associated with the most severe failure condition to which the software can contribute).

Partitioning is defined in Section 2.4.1 of the same standard [8]:

> Partitioning is a technique for providing isolation between software components to contain and/or isolate faults and potentially reduce the effort of the software verification process. Partitioning between software components may be achieved by allocating unique hardware resources to each component (that is, only one software component is executed on each hardware platform in a system). Alternatively, partitioning provisions may be made to allow multiple software components to run on the same hardware platform. Regardless of the method, the following should be ensured for partitioned software components:
>
> a) A partitioned software component should not be allowed to contaminate another partitioned software component's code, input/output (I/O), or data storage areas.
>
> b) A partitioned software component should be allowed to consume shared processor resources only during its scheduled period of execution.
>
> c) [...] Failures of hardware unique to a partitioned software component should not cause adverse effects on other partitioned software components.
>
> d) Any software providing partitioning should have the same or higher software level as the highest level assigned to any of the partitioned software components.
>
> e) Any hardware providing partitioning should be assessed by the system safety assessment process to ensure that it does not adversely affect safety.
>
> The software life-cycle processes should address the partitioning design considerations. These include the extent and scope of interactions permitted between the partitioned components and whether the protection is implemented by hardware or by a combination of hardware and software.

Section 2.4.3, "Safety Monitoring," of DO-178C [8] states:

> Safety monitoring is a means of protecting against specific failure conditions by directly monitoring a function for failures that would result in a failure condition. Monitoring functions may be implemented in hardware, software, or a combination of hardware and software. Through the use of monitoring techniques, the software level of the monitored software may be assigned a software level associated with the loss of its related system function. To allow this assignment, there are three important attributes of the monitor that should be determined:
>
> a) **Software level**: Safety monitoring software is assigned the software level associated with the most severe failure condition category for the monitored function.
>
> b) **System fault coverage**: Assessment of the system fault coverage of a monitor ensures that the monitor's design and implementation are such that the faults which it is intended to detect will be detected under all necessary conditions.
>
> c) **Independence of function and monitor**: The monitor and protective mechanism are not rendered inoperative by the same failure that causes the failure condition.

The `ECLAIR` service `B.INDEPENDENCE` allows the formal specification and systematic checking of software architectural constraints, e.g., to enforce constraints about layering and to prevent bypassing of software interfaces. `B.INDEPENDENCE` is instrumental in proving independence among different software components.

4 ECLAIR Support for DO-330

ECLAIR and the *ECLAIR Qualification Kits*, greatly simplify compliance with the prescription of RTCA DO-330 "Software Tool Qualification Considerations" [9].

4.1 ECLAIR Qualification in Compliance with DO-330

DO-178C characterizes tools according to the following criteria (Section 12.2.2, p. 84 [8]):

Criterion 1: The output of the tool is part of the airborne software, so that an error in the tool can introduce an error in the software.

Criterion 2: The tool automates some verification processes, so that an error in the tool can cause failure to detect an error in the airborne software. Moreover, use of the tool justifies the replacement or reduction of other verification or development processes.

Criterion 3: The tool automates some verification processes, so that an error in the tool can cause failure to detect an error in the airborne software. However, use of the tool does *not* replace or reduce other verification or development processes.

DO-178C defines the notion of *Tool Qualification Level* (TQL), an abstract classification of qualification rigor ranging from TQL-1 (most rigorous) to TQL-5 (least rigorous) (Section 12.2.2, p. 85 [8]). Given the software level and the applicable criteria (considered sequentially from Criterion 1 to Criterion 3), the TQL is determined by Table 2.

Table 2: Tool Qualification Level determination

Software Level	Criteria		
	1	**2**	**3**
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5

For the use of ECLAIR as a verification tool only Criteria 2 and 3 can apply, depending on how the tool is used. If Criterion 2 applies and the software level is A or B, then, to comply with DO-178C, ECLAIR must be qualified at TQL-4. In all other cases it suffices to qualify ECLAIR at TQL-5.

According to DO-330, ECLAIR is a *multi-function tool* in which the user can disable or select parts of the functionality. Qualification of commercial-off-the-shelf (COTS) tools such as ECLAIR is addressed in Section 11.3 of DO-330 (p. 61).

While qualification of ECLAIR at TQL-4 is possible, it requires restricting the considered tool functions still further and entails significantly more effort for both `BUGSENG` and the user. Consequently, qualification at TQL-4 is considered strictly on a case-by-case basis.

The ECLAIR Qualification Kits for DO-178C/DO-330 provide crucial help to safety teams in charge of qualifying ECLAIR for use in safety-related projects where the dependence on the tool operational environment has to be taken into account: the kits contain documents, test suites, procedures and automation facilities that can be used by the customer to independently obtain all the required confidence-building evidence.

5 ECLAIR Support for Compiler Qualification

ECLAIR is also instrumental in achieving *compiler qualification* by validation. This is a service offered in cooperation with our partner *Solid Sands b.v.* as part of their *Compiler Qualification Service*. In case the *SuperTest* compiler test and validation suite reveals defects in the compiler, appropriate mitigations have to be defined and be systematically enforced. Common mitigations include:

1. avoidance of the use of certain language constructs in certain contexts;
2. use of a third party tool to supplement the diagnostic messages not provided by the compiler (e.g., when exceeding translation limits or violating language constraints);
3. avoidance of the use of certain compiler option combinations.

ECLAIR supports all three kinds of mitigations:

1. compliance with the MISRA guidelines excludes the use of several language constructs in certain contexts, and the MISRA guidelines have been designed taking into account the fact that some language constructs are more likely to expose compiler defects than others;
2. ECLAIR checkers for MISRA C:2025 Rule 1.1 and for MISRA C++:2023 Rule 4.1.1 provide suitable diagnostic message for all the violations of the applicable language standard, including the exceeding of translation limits;
3. due to the way ECLAIR works (intercepting all calls to the compiler, linker, assembler and librarian), checks can be defined to ensure the unwanted compiler option combinations are not used.

In addition, ECLAIR's *CerTran* extension automates the configuration of *SuperTest* for compiler qualification by scanning the application build process and creating the exact test configuration files needed to cover all use cases. This can be completely incorporated into a *Continuous Integration system*. Not only does this save a considerable amount of time, but it also avoids configuration errors that can easily occur when scanning the build process manually.

6 The Bigger Picture

ECLAIR is very flexible and highly configurable: it supports all kinds of software development workflows and environments.

ECLAIR is fit for use in mission- and safety-critical software projects: it has been designed from the outset to exclude configuration errors that would undermine the significance of the obtained results.

ECLAIR is developed in a rigorous way and carefully checked with extensive internal test suites (tens of thousands of test cases) and industry-standard validation suites.

ECLAIR is based on solid scientific research results and on the best practices of software development.

ECLAIR's unique features and BUGSENG's strong commitment to the customer, allow for a smooth transition to ECLAIR from any other tool.

BUGSENG's quality system has been certified by TÜV Italia (TÜV SÜD Group) to comply with the requirements of UNI EN ISO 9001:2015 for the "Design, development, maintenance and support of tools for software verification and validation" (IAF 33).

BUGSENG is an Arm's Functional Safety Partner, and is thus recognized as a partner who can reliably support their customers with industry leading functional safety products and services.

For More Information

BUGSENG srl
Via Marco dell'Arpa 8/B
I-43121 Parma, Italy
Email: info@bugseng.com
Web: http://bugseng.com
Tel.: +39 0521 461640

bugseng

**no shortcuts,
no compromises,
no excuses:
software verification done right**

References

[1] R. Bagnara, M. Barr, and P. M. Hill. BARR-C:2018 and MISRA C:2012 (with Amendment 2): Synergy between the two most widely used C coding standards. In DESIGN&ELEKTRONIK, editor, *embedded world Conference 2021 DIGITAL — Proceedings*, pages 378–391, Nuremberg, Germany, 2021. WEKA FACHMEDIEN, Richard-Reitzner-Allee 2, 85540 Haar, Germany.

[2] M. Barr. *BARR-C:2018 — Embedded C Coding Standard*. Barr Group, www.barrgroup.com, 2018.

[3] H. Kuder et al. HIS source code metrics. Technical Report HIS-SC-Metriken.1.3.1-e, Herstellerinitiative Software, April 2008. Version 1.3.1.

[4] MISRA. *MISRA Compliance:2020 — Achieving compliance with MISRA Coding Guidelines*. HORIBA MIRA Limited, Nuneaton, Warwickshire CV10 0TU, UK, February 2020.

[5] MISRA. *MISRA C++:2023 — Guidelines for the use of C++17 in critical systems*. The MISRA Consortium Limited, Norwich, Norfolk, NR3 1RU, UK, October 2023.

[6] MISRA. *MISRA C:2025 — Guidelines for the use of the C language in critical systems*. The MISRA Consortium Limited, Norwich, Norfolk, NR3 1RU, UK, March 2025.

[7] Motor Industry Software Reliability Association. *MISRA C++:2008 — Guidelines for the use of the C++ language in critical systems*. MIRA Limited, Nuneaton, Warwickshire CV10 0TU, UK, June 2008.

[8] RTCA, SC-205. *DO-178C: Software Considerations in Airborne Systems and Equipment Certification*. RTCA, December 2011.

[9] RTCA, SC-205. *DO-330: Software Tool Qualification Considerations*. RTCA, December 2011.