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

Coverage of EN 50657

1 Introduction to EN 50657:2017/A1:2023

EN 50657:2017/A1:2023, "Railways Applications — Rolling stock applications — Software on Board Rolling Stock," is derived from the signalling standard EN 50128:2011 [4] which, in many cases, was also applied in *rolling stock*[1] applications.

EN 50657 uses the same structure and section numbering as EN 50128:2011 and continuity in the application of the standards is ensured: "software that was developed in accordance with EN 50128 can still be re-used for new projects" [6, Subclause 1.6]. EN 50657 and EN 50128 are European standards adapting the IEC 61508 series of standards [8] to the development of software for railway applications: while the scope of EN 50128 is software for railway control and protection systems, the scope of EN 50657 is software for use in rolling stock applications, excluding software that is part of signalling equipment installed on board trains, or does not contribute to, and is segregated from rolling stock operational functions. EN 50657 has been amended in November 2023 [6].

EN 50657 approach to risk management is based on the concept of *levels of software integrity*. There are five software integrity levels: the lowest one is called *Basic Integrity* (B. I. for short); the other four are called *Safety Integrity Levels* (SILs) and are numbered 1, 2, 3 and 4, with 1 being the lowest safety integrity level and 4 being the highest. For each function assigned to each subsystem an integrity lavel must be assigned: B. I., SIL 1, SIL 2, SIL 3 or SIL 4. SIL 4 represents likely potential for severely life-threatening or fatal injury in the event of a malfunction and requires the highest level of assurance that the dependent safety goals are sufficient and have been achieved. EN 50657, based on the integrity level, specifies whether techniques and measures are *recommended*, *highly recommended*, or even *mandatory*. For instance, static analysis is highly recommended at all SILs from 1 to 4.

1.1 Role of ECLAIR in Ensuring Compliance with EN 50657

The ECLAIR Software Verification Platform can be used to comply with several of the techniques and measures required by EN 50657:2017/A1:2023 [6]. In addition, the ECLAIR Fusa Pack greatly simpli-

[1] In the rail transport industry, *rolling stock* refers to railway vehicles, including both powered and unpowered vehicles, such as locomotives, freight cars, passenger cars or coaches, as well as non-revenue cars.

fies obtaining all the confidence-building evidence that is required to make a solid argument justifying the use of ECLAIR in safety-related projects.

2 ECLAIR Coverage of EN 50657 Techniques and Measures

EN 50657 applies to all safety-related as well as non-safety-related software used in rolling stock applications, including application programming, operating systems, support tools, and firmware. For such software, it specifies requirements for lifecycle phases and activities that shall be applied during design and development. These requirements include the application of measures and techniques for the avoidance of and the control of faults and failures in the software. Techniques and measures are detailed in tables contained in Annex A, which is normative. Annex D, which is informative, contains a bibliography of techniques and measures that is referenced in the entries of the tables of Annex A.

The degree of recommendation to use each technique and measure depends on the integrity level, and is symbolically encoded as follows:

M indicates that the method is *Mandatory* for the identified SIL;

HR indicates that the method is *Highly Recommended* for the identified SIL;

R indicates that the method is *Recommended* for the identified SIL;

– indicates that the method has no recommendation for or against its usage for the identified SIL;

NR indicates that the method is positively *Not Recommended* for the identified SIL.

If a highly-recommended technique or measure is not used, then the rationale for using alternative techniques must be detailed in the *Software Quality Assurance Plan* (or in a document referenced therein), unless an approved combination of techniques given in the corresponding table is used. In any case, the selected combination of techniques and measures must be justified, and each selected technique and measure must be demonstrated to have been applied correctly [6, Clause 4].

The following tables have been obtained by extending the corresponding tables in EN 50657:2017/A1:2023 Annex A with a column indicating where ECLAIR, suitably instantiated with the appropriate package, can be used to ensure compliance or to facilitate the achievement of compliance. Note that, in the sequel, every reference to MISRA C:2012 should be interpreted as referring to [9] as amended by [10, 11, 12], whereas MISRA C++ is [13]. As ECLAIR provides direct support for MISRA guidelines as well as guidelines from other coding standards, a reference for a guideline should be taken as a reference to the corresponding *ECLAIR service* as described in the *ECLAIR User's Manual*. For example, "MISRA C:2012 Directive 3.1" corresponds to the ECLAIR service `MC3R1.D3.1` and "BARR-C:2018 Rule 4.1.a" corresponds to the ECLAIR service `NC3.4.1.a`. For ECLAIR services that do not correspond to published coding standards, the service name is given in teletype font: for example, `B.PROJORG` is the name of an ECLAIR service that supports automatically enforcing software architectural constraints [1]. A complete definition of all ECLAIR services is contained in the *ECLAIR User's Manual* and, where applicable, in the corresponding coding standard documentation referenced therein.

2.1 MISRA C:2012

MISRA C:2012 Revision 1 [9], with Amendments 2 [10] and 3 [11], and Technical Corrigendum 2 [12], 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:2012, which allows coding MISRA-compliant applications in subsets of C90, C99, C11 and C18, is supported by the ECLAIR package called "MC3".

2.2 MISRA C++:2008

MISRA C++:2008 [13] is the software development C++ subset developed by MISRA for the motor industry, which is now a de facto standard for safety-, life-, and mission-critical embedded applications also in many other industries. A new set of guidelines for C++17 is currently under development: these guidelines have adapted many of the existing guidelines in MISRA C++:2008, MISRA C:2012 and AUTOSAR as well as adding many new guidelines applicable to C++17. MISRA C++:2008 is supported by the ECLAIR package called "MP1".

2.3 BARR-C:2018

The *Barr Group's Embedded C Coding Standard*, BARR-C:2018 [3], 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 [2]. 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* [7], 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.

Table A.1 — Lifecycle Issues and Documentation

DOCUMENTATION	B. I.	SIL 1	SIL 2	SIL 3	SIL 4	ECLAIR
Software requirements						
6. Software Requirements Specification	HR	HR	HR	HR	HR	✓[a]
7. Overall Software Test Specification	HR	HR	HR	HR	HR	✓[a]
8. Software Requirements Verification Report	R	HR	HR	HR	HR	–
Architecture and design						
9. Software Architecture Specification	R	HR	HR	HR	HR	✓[a,b]
10. Software Design Specification	R	HR	HR	HR	HR	✓[a]
11. Software Interface Specifications	HR	HR	HR	HR	HR	✓[a,b]
12. Software Integration Test Specification	R	HR	HR	HR	HR	✓[a]
13. Software/Hardware Integration Test Specification	R	HR	HR	HR	HR	–
14. Software Architecture and Design Verification Report	R	HR	HR	HR	HR	–
Component Design						
15. Software Component Design Specification	-	HR	HR	HR	HR	✓[a,b]
16. Software Component Test Specification	-	HR	HR	HR	HR	✓[a]
17. Software Component Design Verification Report	-	HR	HR	HR	HR	✓[c]
Component Implementation and Testing						
18. Software Source Code and supporting documentation	HR	HR	HR	HR	HR	✓[d]
19. Software Component Test Report	-	HR	HR	HR	HR	–
20. Software Source Code Verification Report	-	HR	HR	HR	HR	✓[c]
Integration						
21. Software Integration Test Report	R	HR	HR	HR	HR	–
22. Software/Hardware Integration Test Report	R	HR	HR	HR	HR	–
23. Software Integration Verification Report	R	HR	HR	HR	HR	–
Overall Software Testing / Final Validation						
24. Overall Software Test Report	HR	HR	HR	HR	HR	–
25. Software Validation Report	HR	HR	HR	HR	HR	–
26. Tools Validation Report	-	HR	HR	HR	HR	✓[e]
27. Release Note	HR	HR	HR	HR	HR	–

[a] ECLAIR service `B.REQMAN` allows ensuring that all code is forward and backward traceable to documented requirements, including safety requirements. `B.REQMAN` also allows tracing code to the tests and back. The integrated requirements management tool makes ECLAIR a cost-effective, complete solution for requirements-based development and testing.

[b] ECLAIR service `B.PROJORG` 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.

[c] ECLAIR can be configured to automatically produce compliance reports required to meet contractual obligations and industrial standards such as EN 50657.

[d] ECLAIR provides services and metrics that check the presence, format, amount and language of comments in the source code.

[e] ECLAIR can be qualified in compliance with EN 50128 in different ways, all of which result in a the effortless production of the required tool validation report. See Section 3 for more details.

Table A.3 — Software Architecture

	TECHNIQUE/MEASURE	B. I.	SIL 1	SIL 2	SIL 3	SIL 4	ECLAIR
1.	Defensive Programming	-	HR	HR	HR	HR	✓[a]
2.	Fault Detection and Diagnosis	-	R	R	HR	HR	✓[b]
17.	Information Hiding	-	-	-	-	-	✓[c]
18.	Information Encapsulation	R	HR	HR	HR	HR	✓[c]
19.	Fully Defined Interface	HR	HR	HR	M	M	✓[d]
20.	Formal Methods	-	R	R	HR	HR	–
21.	Modelling	R	R	R	HR	HR	✓[e]
22.	Structured Methodology	R	HR	HR	HR	HR	✓[e]
23.	Modelling supported by computer aided design and specification tools	R	R	R	HR	HR	✓[e]

[a] The MISRA C/C++ guidelines promote the use of several defensive programming techniques. E.g., for MISRA C:2012, Directives 4.1, 4.7, 4.11 and 4.14, Rules 2.1–2.7, 14.2, 15.7, 16.4, and 17.7.

[b] The MISRA C/C++ guidelines require systematic checking of error information returned by functions. Guidance is also provided on how to perform some of these checks. E.g., for MISRA C:2012, Directive 4.7, Rules 22.8, 22.9, and 22.10.

[c] The MISRA C/C++ guidelines promote the use of information hiding and encapsulation. E.g., for MISRA C:2012, Directives 4.3 and 4.8 and Rules 8.7 and 8.9. In addition ECLAIR's `B.PROJORG` service can be used to enforce strict encapsulation constraints.

[d] The MISRA C/C++ guidelines promote the full definition of interfaces. E.g., for MISRA C:2012, Rules 8.2 and 8.3 prescribe the use of prototype form and the use of consistent names for function declarations; Rule 17.3 forbids implicit declarations; Directive 4.14 requires data verification; BARR-C:2018 Rule 2.2.h recommends commenting modules and functions with explicit specification of pre-conditions and post-conditions with Doxygen; such comment blocks are automatically checked by ECLAIR for consistency.

[e] ECLAIR service `B.PROJORG` 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.PROJORG` is instrumental in proving independence among different software components, which is essential when the software consists of components of different software safety integrity levels and treating them as belonging to the highest of these levels is inadvisable (see [8], Clause 7.3.4.9).

Table A.4 — Software Design and Implementation

	TECHNIQUE/MEASURE	B. I.	SIL 1	SIL 2	SIL 3	SIL 4	ECLAIR
1.	Formal Methods	-	R	R	HR	HR	–
2.	Modelling	R	HR	HR	HR	HR	–
3.	Structured methodology	R	HR	HR	HR	HR	–
4.	Modular Approach	HR	M	M	M	M	✓[a]
5.	Components	HR	HR	HR	HR	HR	✓[b]
6.	Design and Coding Standards	HR	HR	HR	M	M	✓[c]
7.	Analysable Programs	HR	HR	HR	HR	HR	✓[d]
8.	Strongly Typed Programming Language	R	HR	HR	HR	HR	✓[e]

continued

Table A.4 — Software Design and Implementation

	TECHNIQUE/MEASURE	B. I.	SIL				ECLAIR
			1	2	3	4	
9.	Structured Programming	R	HR	HR	HR	HR	✓[f]
10.	Programming Language	R	HR	HR	HR	HR	✓[g]
11.	Language Subset	-	-	-	HR	HR	✓[h]
12.	Object Oriented Programming	R	R	R	R	R	✓[i]
13.	Procedural programming	R	HR	HR	HR	HR	✓[j]
14.	Metaprogramming	R	R	R	R	R	–

[a] ECLAIR offers numerous services to enforce modularization in the design and coding phase of a software project. With reference to item D.38 (Modular Approach) in Annex D of [6]: connections between modules/components can be limited and strictly defined using `B.PROJORG`; cohesion in one module/component can be constrained to be high using ECLAIR specific metric `B.STFCO_UNIT`; modules/components and subprograms can be constrained to be small by enforcing upper bounds on HIS and other metrics related to size and complexity; MISRA C:2012 Rule 15.5 and MISRA C++:2008 Rule 6–6–5 require subprograms to have a single entry and a single exit only; the MISRA C/C++ guidelines promote the full definition of interfaces as outlined in note (d) to Table A.3; the MISRA C/C++ guidelines include prescriptions against the use of unnecessary global variables, e.g., for MISRA C:2012, Rules 8.7 and 8.9; the specific ECLAIR service `B.GLOBALVAR` allows fine control of acceptable global variables. BARR-C:2018 Rule 2.2.h recommends commenting modules and functions with explicit specification of pre-conditions and post-conditions with Doxygen; such comment blocks are automatically checked by ECLAIR for consistency; limitations on the number of parameters of a function/method can be automatically enforced by imposing upper bounds on the `HIS.PARAM` metric.

[b] See Table A.20.

[c] See Table A.12.

[d] The MISRA C/C++ and BARR-C:2018 coding standards can be used to ensure that programs are relatively easy to reason about and to analyze statically. In particular, they limit the use of non-structured programming constructs, language extensions and assembly code; they promote the reduction of the scope of identifiers, the use of simple branching and loop decision, the use of simplified loop and switch constructs. In addition, HIS and other metrics provided by ECLAIR allow imposing upper bounds on the size and complexity of components as well as on the number of possible paths through them.

[e] MISRA C/C++ enforce strong typing on the respective languages. E.g., for MISRA C:2012, Rules 10.1–10.8, 11.1–11.9, and 14.4.

[f] The MISRA C/C++ guidelines include limits on the use of non-structured control-flow constructs. E.g., for MISRA C:2012, Rules 14.3, 15.1–15.4, and 21.4. A threshold on metric `HIS.GOTO` allows limiting the use of `goto`.

[g] Excluding ADA and the obsolete programming languages in EN 50128 Table A.15, C and C++ are the ones with the longest history of application in the development of critical systems. Moreover, it can be argued that the MISRA C and MISRA C++ subsets are as safe as other languages marked as highly recommended in that table. MISRA C and MISRA C++ satisfy the requirements of D.54 (Suitable Programming languages) in Annex D of [6].

[h] MISRA C/C++ and BARR-C:2018 define language subsets where the potential of committing possibly dangerous mistakes is reduced.

[i] MISRA C++ is an object oriented programming language. Service `B.PROJORG` can support an object oriented programming style also in C, by restricting access to "private" data and functions.

[j] (All subsets of) C and C++ are procedural programming languages.

Table A.5 — Verification and Testing

TECHNIQUE/MEASURE	B. I.	SIL				ECLAIR
		1	2	3	4	
1. Formal Proof	-	R	R	HR	HR	–
2. Static Analysis	-	HR	HR	HR	HR	✓[a]
3. Dynamic Analysis and Testing	-	HR	HR	HR	HR	–
4. Metrics	-	R	R	R	R	✓[b]
5. Traceability	R	HR	HR	M	M	✓[c]
6. Software Error Effect Analysis	-	R	R	HR	HR	–
7. Test Coverage for code	-	HR	HR	HR	HR	–
8. Functional/ Black-box Testing	HR	HR	HR	M	M	–
9. Performance Testing	-	HR	HR	HR	HR	–
10. Interface Testing	HR	HR	HR	HR	HR	–

[a] ECLAIR employs state-of-the-art static analysis techniques.

[b] ECLAIR automatically computes numerous source code metrics.

[c] ECLAIR service `B.REQMAN` allows ensuring that all code is forward and backward traceable to documented requirements, including safety requirements. `B.REQMAN` also allows tracing code to the tests and back. The integrated requirements management tool makes ECLAIR a cost-effective, complete solution for requirements-based development and testing.

Table A.12 — Coding Standards

	TECHNIQUE/MEASURE	B. I.	SIL				ECLAIR
			1	2	3	4	
1.	Coding Standard	HR	HR	HR	M	M	✓[a]
2.	Coding Style Guide	HR	HR	HR	HR	HR	✓[b]
3.	No Dynamic Objects	-	R	R	HR	HR	✓[c]
4.	No Dynamic Variables	-	R	R	HR	HR	✓[c]
5.	No uncontrolled address referencing, for example pointers	-	R	R	R	R	✓[d]
6.	No uncontrolled Recursion	-	R	R	HR	HR	✓[e]
7.	No Unconditional Jumps	-	HR	HR	HR	HR	✓[f]
8.	Limited size and complexity of Functions, Subroutines and Methods	HR	HR	HR	HR	HR	✓[g]
9.	Only one Entry/Exit Point for Functions, Subroutines and Methods	R	HR	HR	HR	HR	✓[h]
10.	Limited number of subroutine parameters	R	R	R	R	R	✓[i]
11.	Defined Control of Global Variables	HR	HR	HR	M	M	✓[j]

[a] The MISRA C/C++ and BARR-C:2018 coding standards define language subsets where the potential of committing possibly dangerous mistakes is reduced.

[b] More than half of the guidelines in BARR-C:2018 [3] concern coding style [2]. MISRA C:2012 Rules 7.3 and 16.5 are also stylistic.

[c] The MISRA C/C++ guidelines include prescriptions limiting the use of dynamic memory allocation. E.g., for MISRA C:2012, Directive 4.12 and Rules 18.7, 21.3, 22.1 and 22.2.

[d] The MISRA C/C++ guidelines include rules restricting the use of pointers. E.g., for MISRA C:2012, Rules 8.13, 11.1–11.8, and 18.1–18.5. The specific ECLAIR services `B.PTRDECL` and `B.PTRUSE` allow fine control of pointers' use.

[e] MISRA C Rule 17.2 and MISRA C++ Rule 7-5-4 forbid recursion. A threshold on metric `HIS.ap_cg_cycle` also allows ruling out recursion.

[f] The MISRA C/C++ guidelines include limits on the use of non-structured control-flow constructs as well as other unconditional jumps. E.g., for MISRA C:2012, Rules 14.3, 15.1–15.4, and 21.4. A threshold on metric `HIS.GOTO` allows limiting the use of `goto`.

[g] HIS and other metrics are related to the size and complexity of software components. ECLAIR allows associating thresholds to each metric.

[h] MISRA C:2012 Rule 15.5 and MISRA C++:2008 Rule 6–6–5 require subprograms to have a single entry and a single exit only. An upper threshold on metric `HIS.RETURN` allows for a more flexible approach.

[i] Limitations on the number of parameters of a function/method can be automatically enforced by imposing upper bounds on the `HIS.PARAM` metric.

[j] The MISRA C/C++ guidelines include prescriptions against the use of unnecessary global variables, e.g., for MISRA C:2012, Rules 8.7 and 8.9; the specific ECLAIR service `B.GLOBALVAR` allows fine control of acceptable global variables. In addition ECLAIR's `B.PROJORG` service can be used to enforce constraint on the access of global variable.

Table A.19 — Static Analysis

TECHNIQUE/MEASURE	B. I.	SIL 1	SIL 2	SIL 3	SIL 4	ECLAIR
1. Boundary Value Analysis	-	R	R	HR	HR	–
2. Checklists	-	R	R	R	R	–
3. Control Flow Analysis	-	HR	HR	HR	HR	✓[a]
4. Data Flow Analysis	-	HR	HR	HR	HR	✓[b]
5. Error Guessing	-	R	R	R	R	–
6. Walkthroughs/Design Reviews	HR	HR	HR	HR	HR	✓[c]

[a] ECLAIR builds accurate control flow graphs to reason on (feasible and unfeasible) execution paths.

[b] ECLAIR performs a number of data flow analyses to reason about, e.g., pointers, values and dead stores.

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

Table A.20 — Components

TECHNIQUE/MEASURE	B. I.	SIL 1	SIL 2	SIL 3	SIL 4	ECLAIR
1. Information Hiding[1]	-	-	-	-	-	✓[a]
2. Information Encapsulation	R	HR	HR	HR	HR	✓[a]
3. Parameter Number Limit	R	R	R	R	R	✓[b]
4. Fully Defined Interface	R	HR	HR	M	M	✓[c]

[a] The MISRA C/C++ guidelines promote the use of information hiding and encapsulation. E.g., for MISRA C:2012, Directives 4.3 and 4.8 and Rules 8.7 and 8.9. In addition ECLAIR's `B.PROJORG` service can be used to enforce strict encapsulation constraints.

[b] Limitations on the number of parameters of a function/method can be automatically enforced by imposing upper bounds on the `HIS.PARAM` metric.

[c] The MISRA C/C++ guidelines promote the full definition of interfaces. E.g., for MISRA C:2012, Rules 8.2 and 8.3 prescribe the use of prototype form and the use of consistent names for function declarations; Rule 17.3 forbids implicit declarations; Directive 4.14 requires data verification; BARR-C:2018 Rule 2.2.h recommends commenting modules and functions with explicit specification of pre-conditions and post-conditions with Doxygen; such comment blocks are automatically checked by ECLAIR for consistency.

[1] Note 1 in EN 50657 Table A.20 says that "Information Hiding and encapsulation are only highly recommended if there is no general strategy for data access."

3 ECLAIR Qualification in Compliance with EN 50128/EN 50657

The ECLAIR functionality described above is qualifiable in compliance with EN 50128: ECLAIR is a class T2 tool and meets all the requirements set forth in EN 50128:2011/A2:2020 for such tools [5, Clause 6.7.4]. TÜV SÜD audited BUGSENG software development and quality assurance processes for ECLAIR, as well as the internal validation activities performed by BUGSENG on each ECLAIR release. At the end of its assessment, TÜV SÜD awarded BUGSENG the "Software Tool for Safety Related Development" Certificate no. Z10 116151 0001 Rev. 00, attesting that the ECLAIR Software Verification Platform is suitable to be used in safety-related development projects according to EN 50128:2011/A2:2020 for any SIL.

The ECLAIR Qualification Kits for EN 50128 provide further 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. BUGSENG also offers the ECLAIR Qualification Service, whereby qualified BUGSENG personnel undertakes almost all the qualification effort.

4 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, A. Bagnara, and P. M. Hill. Formal verification of software architectural constraints. In DESIGN&ELEKTRONIK, editor, *embedded world Conference 2023 — Proceedings*, pages 271–279, Nuremberg, Germany, 2023. WEKA FACHMEDIEN, Richard-Reitzner-Allee 2, 85540 Haar, Germany.

[2] 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.

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

[4] CENELEC. *EN 50128:2011: Railway applications — Communication, signalling and processing systems — Software for railway control and protection systems*. CENELEC, Brussels, Belgium, June 2011.

[5] CENELEC. *EN 50128:2011/A2:2020: Railway applications — Communication, signalling and processing systems — Software for railway control and protection systems*. CENELEC, Brussels, Belgium, August 2020. Amendment A2 to EN 50128:2011.

[6] CENELEC. *EN 50657:2017/A1:2023: Railway applications — Rolling stock applications — Software on Board Rolling Stock*. CENELEC, Brussels, Belgium, November 2023. Amendment A1 to EN 50657:2017.

[7] 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.

[8] IEC. *IEC 61508-1:2010: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*. IEC, Geneva, Switzerland, April 2010.

[9] MISRA. *MISRA C:2012 — Guidelines for the use of the C language critical systems*. HORIBA MIRA Limited, Nuneaton, Warwickshire CV10 0TU, UK, February 2019. Third edition, first revision.

[10] MISRA. *MISRA C:2012 Amendment 2 — Updates for ISO/IEC 9899:2011 Core functionality*. HORIBA MIRA Limited, Nuneaton, Warwickshire CV10 0TU, UK, February 2020.

[11] MISRA. *MISRA C:2012 Amendment 3 — Updates for ISO/IEC 9899:2011/2018 Phase 2 — New C11/C18 features*. The MISRA Consortium Limited, Norwich, Norfolk NR3 1RU, UK, October 2022.

[12] MISRA. *MISRA C:2012 Technical Corrigendum 2 — Technical clarification of MISRA C:2012*. The MISRA Consortium Limited, Norwich, Norfolk NR3 1RU, UK, March 2022.

[13] 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.