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

Coverage of EN 50716

1 Introduction to EN 50716:2023

EN 50716:2023, "Railway Applications — Requirements for software development" is a functional-safety standard for the railway industry issued by CENELEC [6]. It is a European standard adapting the IEC 61508 series of standards [8] to the development of safety-related software for railway applications. EN 50716:2023 supersedes EN 50128:2011 and EN 50657:2017 along with all their amendments and corrigenda [4], [5]. With respect to those superseded standards, EN 50716:2023 includes significant technical changes including:

- the rewriting of requirements of Clause 5 (*Software management and organization*) for improved readability;
- the update of normative Annex A (*Criteria for the selection of techniques and measures*) for better alignment with lifecycle phases;
- the addition, in informative Annex C (*Guidance on software development*), of new clauses C.1, with additional guidance on lifecycle models, and C.2, with guidance on modelling for software development;
- the provision of additional guidance software components of different software integrity levels;
- the generalization of requirements on programming languages.

EN 50716 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 level 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 50716, 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 50716

The ECLAIR Software Verification Platform can be used to comply with several of the techniques and measures required by EN 50716:2023 [6]. In addition, the ECLAIR FuSa Pack greatly simplifies 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 50716

EN 50716 applies to all safety-related as well as non-safety-related software used in railway 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.

2.1 ECLAIR Coverage of EN 50716 Architecture and Design

The requirements prescribed by EN 50716 include the development of a software architecture that achieves the requirements of the software, specified by a Software Architecture Specification. In particular, the reuse of pre-existing software components is subjected to several restrictions.

In particular, interfaces with other parts of the software shall be clearly identified and documented (7.3.4.7.a). This can be achieved via the *ECLAIR Independence Checker*, as it allows the formal specification and systematic checking of software architectural constraints. In addition pre-existing software shall be accompanied by a precise and complete description (7.3.4.7.d). Such description can be produced automatically via the *ECLAIR Scout* product (service `B.SCOUT`).

2.2 ECLAIR Coverage of EN 50716 Techniques and Measures

The requirements prescribed by EN 50716 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 50716:2023 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. 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:2025 Directive 3.1" corresponds to the ECLAIR service `MC4.D3.1`, "MISRA C++:2023 Rule 9.4.1" corresponds to the ECLAIR service `MP2.9.4.1` 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.INDEPENDENCE` 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.

2.3 MISRA C:2025

MISRA C:2025 [10] 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".

2.4 MISRA C++:2023

MISRA C++:2023 [9] 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 [11], 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".

2.5 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".

2.6 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] 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.

[c] ECLAIR can be configured to automatically produce compliance reports required to meet contractual obligations and industrial standards such as EN 50716.

[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 and EN 50657 in different ways, all of which result in a the effortless production of the required tool validation report. See Section **??** 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.	Fully Defined Interface	HR	HR	HR	M	M	✓[c]
18.	Modelling	R	R	R	HR	HR	✓[d]
19.	Structured Methodology	R	HR	HR	HR	HR	✓[d]

[a] The MISRA C/C++ guidelines promote the use of several defensive programming techniques. E.g., for MISRA C:2025, Directives 4.1, 4.7, 4.11 and 4.14, Rules 14.2, 15.7, 16.4, 17.4 and 17.7; for MISRA C++:2023, Rules 0.1.2, 9.4.1, 9.4.2, 9.5.1, 9.6.5 and 18.3.1.

[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:2025, Directive 4.7, Rules 17.7, 22.8, 22.9, and 22.10; for MISRA C++:2023, Rules 0.1.2 and 18.3.1

[c] The MISRA C/C++ guidelines promote the full definition of interfaces. E.g., for MISRA C:2025, Rules 8.2 and 8.3 prescribe the use of prototype form and the use of consistent names and qualifiers for function declarations, Rule 17.3 forbids implicit declarations, Directive 4.14 requires data verification; for MISRA C++:2023, Rules 6.2.2, 6.9.2 and 13.3.3 prescribe the use of consistent name and types for entity declarations. BARR-C:2018 Rule 2.2.h recommends commenting modules and functions with explicit specification of preconditions and post-conditions with Doxygen; such comment blocks are automatically checked by ECLAIR for consistency.

[d] 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, 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.	Modelling	R	HR	HR	HR	HR	–
2.	Structured methodology	R	HR	HR	HR	HR	–
3.	Modular Approach	HR	M	M	M	M	✓[a]
4.	Components	HR	HR	HR	HR	HR	✓[b]
5.	Design and Coding Standards	HR	HR	HR	M	M	✓[c]
6.	Analysable Programs	HR	HR	HR	HR	HR	✓[d]
7.	Structured Programming	R	HR	HR	HR	HR	✓[e]
8.	Suitable Programming Languages	R	HR	HR	HR	HR	✓[f]

continued

Table A.4 — Software Design and Implementation

	TECHNIQUE/MEASURE	B. I.	SIL				ECLAIR
			1	2	3	4	

[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.INDEPENDENCE`; 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:2025 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:2025, Rules 8.7 and 8.9; for MISRA C++:2023, Rule 6.7.2. 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] The MISRA C/C++ guidelines include limits on the use of non-structured control-flow constructs. E.g., for MISRA C:2025, Rules 14.3, 15.1–15.4, and 21.4; for MISRA C++:2023, Rules 0.0.2, 9.6.1–9.6.3 and 21.10.2. A threshold on metric `HIS.GOTO` allows limiting the use of `goto`.

[f] C and C++ are the programming languages 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].

Table A.5 — Verification and Testing

TECHNIQUE/MEASURE		B. I.	SIL				ECLAIR
			1	2	3	4	
1.	Formal Proof	-	R	R	HR	HR	✓[a]
2.	Static Analysis	-	HR	HR	HR	HR	✓[b]
3.	Dynamic Analysis and Testing	HR	HR	HR	M	M	–
4.	Metrics	-	R	R	R	R	✓[c]
5.	Test Coverage for code	-	HR	HR	HR	HR	✓[d]
9.	Performance Testing	-	HR	HR	HR	HR	–
10.	Interface Testing	HR	HR	HR	HR	HR	–

[a] Static analysis with ECLAIR, under the condition that no language extensions were used, constitutes a formal verification of certain program properties. For example, if ECLAIR does not issue any violation report or caution report concerning MISRA C:2025 Rule 9.1 and no language extensions have been used (inline assembly in particular), this is a formal proof that uninitialized memory reads cannot take place.

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

[c] ECLAIR automatically computes numerous source code metrics.

[d] 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.10 — Software maintenance

TECHNIQUE/MEASURE		B. I.	SIL				ECLAIR
			1	2	3	4	
1.	Impact Analysis	R	HR	HR	M	M	✓[a,b]
2.	Data Recording and Analysis	HR	HR	HR	M	M	

[a] ECLAIR service `B.INDEPENDENCE` allows the formal specification and systematic checking of software architectural constraints and the precise identification of all the interactions among components; this is essential in proving independence among different software components to facilitate the impact analysis of a change or an enhancement.

[b] ECLAIR service `B.SCOUT` allows finding which part of a library is being used by a given project; this is instrumental for judging the impact of a change in the project and/or in the library.

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. Limited size and complexity of Functions, Subroutines and Methods	HR	HR	HR	HR	HR	✓[c]
4. Entry/Exit Point strategy for Functions, Subroutines and Methods	R	HR	HR	HR	HR	✓[d]
5. Defined Control of Global Variables	HR	HR	HR	M	M	✓[e]

[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:2025, Rules 7.3 and 16.5 and MISRA C++:2023, Rules 5.13.5, 5.13.6 and 6.0.1 are also stylistic.

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

[d] MISRA C:2025, 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.

[e] The MISRA C/C++ guidelines include prescriptions against the use of unnecessary global variables, e.g., for MISRA C:2025, Rules 8.7 and 8.9; for MISRA C++:2023, Rule 6.7.2. The specific ECLAIR service `B.GLOBALVAR` allows fine control of acceptable global variables. In addition, the *ECLAIR Independence Checker* can be used to enforce constraint on the access of global variable.

Table A.19 — Static Analysis

TECHNIQUE/MEASURE	B. I.	SIL				ECLAIR
		1	2	3	4	
1. Control Flow Analysis	-	HR	HR	HR	HR	✓[a]
2. Data Flow Analysis	-	HR	HR	HR	HR	✓[b]
3. Software Error Effect Analysis	-	R	R	HR	HR	–
4. 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				ECLAIR
			1	2	3	4	
1.	Information Hiding / Encapsulation [1]	R	HR	HR	HR	HR	✓[a]
2.	Parameter Number Limit	R	R	R	R	R	✓[b]
3.	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:2025, Directives 4.3 and 4.8 and Rules 8.7 and 8.9. In addition, the *ECLAIR Independence Checker* 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:2025, Rules 8.2 and 8.3 prescribe the use of prototype form and the use of consistent names and qualifiers for function declarations, Rule 17.3 forbids implicit declarations, Directive 4.14 requires data verification; for MISRA C++:2023, Rules 6.2.2, 6.9.2 and 13.3.3 prescribe the use of consistent name and types for entity declarations. 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] Requirement 1 in EN 50716 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 and EN 50657

The ECLAIR functionality described above is qualifiable in compliance with EN 50128 and EN 50657: ECLAIR is a class T2 tool and meets all the requirements set forth in EN 50128:2011/A2:2020 [4, Clause 6.7.4]. and EN 50657:2017/A1:2023 [5, Clause 6.7.4] for such tools.

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. 01, attesting that the ECLAIR Software Verification Platform is suitable to be used in safety-related development projects according to EN 50128:2011/A2:2020 and EN 50657:2017/A1:2023 for any SIL.

The ECLAIR Qualification Kits for EN 50128 and EN 50657 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

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/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.

[5] 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.

[6] CENELEC. *EN 50716:2023: Railway Applications — Requirements for software development*. CENELEC, Brussels, Belgium, November 2023.

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

[10] 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.

[11] 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.