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

Coverage of ISO 19014

1 Introduction to ISO 19014:2018

ISO 19014, "Earth-moving machinery — Functional safety," is a series of international functional-safety standards applicable to the design and development of all safety-related parts of machine-control systems (MCS) in earth-moving machinery and its equipment. It is a sector-specific implementation of the IEC 61508 series of standards and its first edition was published in 2018 [5].

ISO 19014 approach to risk management is based on the determination of the *Machine Performance Level* (MPL) for each safety-related part of control systems; the MPL specifies the ability of such parts to perform a safety-related function under reasonably foreseeable conditions [5]. There are six MPL: *QM*, *a*, *b*, *c*, *d*, and *e*. The MPLs *a–e* correspond to the *Performance Levels* (PL) *a–e* of ISO 13849-1:2015 "Safety of machinery — Safety-related parts of control systems." Thus, MPL *e* corresponds to the most critical safety-related functions, i.e., those that have potential for severely life-threatening or fatal injury in the event of a malfunction and that, consequently, require a *Probability of Dangerous Failure per Hour* (PFHd) less than 10^{-7}.

The software aspects of ISO 19014 are treated in its Part 4 "Design and evaluation of software and data transmission for safety-related parts of the control system" [6].

1.1 Role of ECLAIR in Ensuring Compliance with ISO 19014:2020

The ECLAIR Software Verification Platform can be used to comply with several of the methods and measures required by ISO 19014 Part 4 "Design and evaluation of software and data transmission for safety-related parts of the control system" [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 ISO 19014-4:2020 Methods and Measures

Part 4 of ISO 19014:2020 specifies the requirements for product development at the software level [6]. It features several tables defining methods and measures that must be considered in order to comply

with the standard. The different methods and measures listed in each table contribute to the level of confidence in achieving compliance with the corresponding requirement. Methods and measures are listed in each table either as *consecutive entries*, numbered with 1, 2, 3, ... in the leftmost table column, or as *alternative entries*, labeled with 1.a, 1.b, 1.c, ... in the same column.

The degree of recommendation to use each method or measure depends on the *Machine Performance Level required*, MPLr for short, and is symbolically encoded as follows:

+ indicates that the method or measure shall be used for the MPLr, unless there is reason not to, in which case that reason shall be documented during the safety planning phase;

o indicates that the method or measure may be used for the MPLr;

- indicates that the method or measure is not suitable for meeting the MPLr.

For consecutive entries, all methods/measures listed with **+**, in accordance with the MPLr, apply. For alternative entries, at least one of the methods/measures listed with **+** shall be applied in accordance with the MPLr.

The following tables have been obtained by extending the corresponding tables in ISO 19014:2020 Part 4 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.

2.1 MISRA C:2025

MISRA C:2025 [8] 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 [7] 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 [9], 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 [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".

Table 3 — Software safety requirements specification

Method/Measure		MPLr				ECLAIR
		a	b, c	d	e	
1	Requirements specification in natural language	+	+	+	+	✓[a]
2	Computer-aided specification tools	o	o	o	+	✓[a,b]
3.a	Informal methods	+	+	+	-	–
3.b	Semi-formal methods	+	+	+	+	–
3.c	Formal methods	+	+	+	+	✓[b]
4	Forward traceability between the system safety requirements and the software safety requirements	o	o	o	+	✓[a]
5	Backward traceability between the software safety requirements and the system safety requirements	o	o	o	+	✓[a]
6.a	Walk-through of software safety requirements	+	+	+	-	✓[a]
6.b	Inspection of software safety requirements	+	+	+	+	✓[a]

[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.

Table 4 — Software architecture design

Method/Measure		MPLr				ECLAIR
		a	b, c	d	e	
1.a	Informal methods	+	+	+	-	–
1.b	Semi-formal methods	+	+	+	+	–
1.c	Formal methods	+	+	+	+	✓[a]
2	Computer-aided design tools	o	o	o	+	✓[a,b]
3.a	Cyclic behaviour, with guaranteed maximum cycle time	o	o	+	+	–
3.b	Time-triggered architecture	o	o	+	+	–
3.c	Event-driven, with guaranteed maximum response time	o	o	+	+	–
4	Forward traceability between the software safety requirements specification and the software architecture	o	o	o	+	✓[a,b]
5	Backward traceability between the software architecture and the software safety requirements specification	o	o	o	+	✓[a,b]
6.a	Walk-through of software architecture	+	+	+	-	✓[a]
6.b	Inspection of software architecture	+	+	+	+	✓[a]

[a] 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] 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.

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* [4], 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 5 — Software module design and coding

Method/Measure		MPLr				ECLAIR
		a	b, c	d	e	
1.a	Informal methods	+	+	-	-	–
1.b	Semi-formal methods	+	+	+	+	–
1.c	Formal methods	+	+	+	+	✓[a]
2	Computer-aided design tools	o	o	o	+	✓[a,b]
3	Use of design and coding standards	o	+	+	+	✓[c]
4	No unstructured control flow in programs in higher level languages	o	o	+	+	✓[d]
5	Limited automatic type conversion	o	o	+	+	✓[e]
6	Limited use of interrupts	o	o	o	+	–
7	Limited use of pointers	o	o	o	+	✓[f]
8	Limited use of recursion	o	o	o	+	✓[g]
9.a	Dynamic variables or objects without online check	o	o	-	-	✓[h]
9.b	Dynamic variables or objects with online check	o	o	+	+	✓[h]

[a] 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] 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.

[c] ECLAIR can be used to verify the compliance of source code to coding standards, such as the MISRA coding standards and BARR-C:2018. In turn, compliance with such standards ensures that the semantics of the source code is well defined and that certain classes of run-time errors cannot occur. ECLAIR can also be used to verify that the value of software metrics are inside prescribed ranges.

[d] 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`.

[e] The MISRA C/C++ guidelines include several rules restricting the use of implicit conversions. E.g., for MISRA C:2025, Rules 10.1, 10.3, 10.4, 10.6, 10.7, 11.1, 11.2, 11.4, 11.5 and 11.9; for MISRA C++:2023, Rules 7.0.1, 7.0.2, 7.0.5, 7.0.6, 7.11.1 and 7.11.3

[f] The MISRA C/C++ guidelines include rules restricting the use of pointers. E.g., for MISRA C:2025, Rules 8.13, 11.1–11.8, and 18.1–18.5; for MISRA C++:2023, Rules 8.2.3, 8.2.4, 8.2.6–8.2.8, 8.7.1, 8.7.2, 8.9.1 and 10.1.1. The specific ECLAIR services `B.PTRDECL` and `B.PTRUSE` allow fine control of pointers' use.

[g] MISRA C:2025 Rule 17.2 and MISRA C++:2023 Rule 8.2.0 forbid recursion. A threshold on metric `HIS.ap_cg_cycle` also allows ruling out recursion.

[h] The MISRA C/C++ guidelines include prescriptions limiting the use of manual dynamic memory allocation. E.g., for MISRA C:2025, Directive 4.12 and Rules 18.7, 21.3, 22.1 and 22.2; for MISRA C++:2023, Rules 21.6.1–21.6.3.

Table 5 — Software module design and coding *(continued)*

Method/Measure		MPLr				ECLAIR
		a	b, c	d	e	
10	Software module size limit	+	+	+	+	✓[i]
11	One entry/one exit point in subroutines and functions	o	+	+	+	✓[j]
12	Fully defined interface	o	+	+	+	✓[k]
13	Information hiding/encapsulation	o	o	+	+	✓[l]
14	Software complexity control	o	o	o	+	✓[m]
15	Structured design or coding	o	+	+	+	✓[n]
16	Defensive programming	o	o	o	+	✓[o]
17	Use of trusted/verified software elements	o	o	o	o	✓[p]
18	Forward traceability between the software safety requirements specification and the software design	o	o	o	+	✓[q]
19.a	Walk-through of software design, source code or both	+	+	+	-	✓[r]
19.b	Inspection of software design, source code or both	+	+	+	+	✓[r]

[i] ECLAIR supports metrics that are strongly correlated with the size of functions, methods and translation units (e.g., the number of statements and the number of logical lines of code).

[j] 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.

[k] 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.

[l] 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; for MISRA C++:2023, Rules 6.0.3, 6.5.1 and 6.7.2. In addition, the *ECLAIR Independence Checker* can be used to enforce strict encapsulation constraints.

[m] ECLAIR supports metrics that are strongly correlated with the complexity size of functions and methods (e.g., cyclomatic complexity and the number of acyclic paths through the body).

[n] 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`.

[o] 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.

[p] The *ECLAIR Scout* product (service `B.SCOUT`) allows for the precise identification of the component to be qualified. This is particularly challenging in the case of C++, where libraries make extensive use of templates and compile-time evaluation, so that much code has not a direct manifestation in the object code.

[q] 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.

[r] 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 6 — Language and tool selection

Method/Measure		MPLr				ECLAIR
		a	b, c	d	e	
1	Suitable programming language	+	+	+	+	✓[a]
2	Language subset	o	o	o	+	✓[a]
3.a	Tools and translators with increased confidence from use or validation	o	+	+	+	✓[b]
3.b	Certified tools and certified translators	o	+	+	+	✓[c]

[a] MISRA C and MISRA C++ are subsets of C and C++, respectively that are generally recognized as suitable for all sorts of safety-related development. BARR-C:2018 defines a subset of C (significantly larger than the one defined by MISRA C) that is also widely recognized as suitable for less critical development.

[b] The first commercial ECLAIR license for use in safety-critical development was granted in June 2013. Over the years, ECLAIR has been used for safety- and security-related development in the following industry sectors: automotive, aviation, energy, household appliances, industrial, medical devices, railways, space.

[c] ECLAIR has been certified by TÜV SÜD according to IEC 61508 up to SIL 4, ISO 26262 up to ASIL D, EN 50128 up to SIL 4, IEC 62304 up to Class C, ISO 25119 up to SRL 3: see the TÜV SÜD Report for details.

Table 7 — Software module testing

Method/Measure		MPLr				ECLAIR
		a	b, c	d	e	
1	Boundary value analysis	o	o	+	+	–
2	Control flow analysis	o	o	+	+	✓[a]
3	Data flow analysis	o	o	+	+	✓[b]
4	Test case execution from boundary value analysis	o	o	o	+	–
5	Functional/black box testing (including fault insertion (FI) testing)	+	+	+	+	–
6.a	Structure test coverage (entry points)	o	+	-	-	–
6.b	Structure test coverage (statements)	o	+	+	-	–
6.c	Structure test coverage (branches)	o	+	+	+	–
7	Equivalence classes and input partition testing	o	o	+	+	–
8	Test case execution from model-based test case generation	o	o	o	+	–
9	Response timings and memory constraints testing	o	+	+	+	–
10	Performance requirements testing	o	+	+	+	–
11	Avalanche/stress testing	o	o	o	+	–
12	SW module interface testing	o	o	o	+	–
13	Back-to-back comparison testing	o	o	+	+	–
14	Forward traceability between the software module design and the module test specifications	o	o	o	+	✓[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] 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 8 — Software module integration and testing

Method/Measure		MPLr				ECLAIR
		a	b, c	d	e	
1	Functional/black box testing (including fault insertion (FI) testing)	+	+	+	+	–
2	Equivalence classes and input partition testing	o	o	+	+	–
3	Response timings and memory constraints	o	+	+	+	–
4	Performance requirements testing	o	+	+	+	–
5	Avalanche/stress testing	o	o	o	+	–
6	Back-to-back comparison testing	o	o	+	+	–
7	Forward traceability between the software architecture design and the integration test specifications	o	o	o	+	✓[a,b]

[a] 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] 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 9 — Software validation

Method/Measure		MPLr				ECLAIR
		a	b, c	d	e	
1.a	Machine network test	+	+	+	+	–
1.b	Hardware-in-the-loop test	+	+	+	+	–
1.c	Machine level test	+	+	+	+	–
2	Forward traceability between the software safety requirements specification and software verification (including data verification) plan	o	o	o	+	✓[a]
3	Backward traceability between the software verification (including data verification) plan and the software safety requirements specification	o	o	o	+	✓[a]

[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.

2.5 ECLAIR Support for *Independence by software partitioning*

Part 4 of ISO 19014 recommends software partitioning as a way of ensuring independence among software components [6, Clause 7]. The effectiveness criteria of the adopted methods and measures are spelled out as follows:

- to control hazards that can occur in subsystems so that they cannot affect other subsystems;
- to have adequate independence of software components by software partitioning.

ECLAIR can be used to provide evidence ensuring independence as follows:

MISRA compliance: Compliance with the MISRA guidelines reduces the risk of execution blocking due to unexpected excessive loop iterations as well as stack overflow.

ECLAIR Independence Checker: This is a very general ECLAIR service to detect and check, by control and data flow static analyses, all interactions between user-defined software elements occurring via read or write accesses to shared memory, function calls, passing and returning of data, as well as static dependencies due to header file inclusion and macro expansion.

3 ECLAIR Qualification in Compliance with ISO 19014 Part 4

The requirements of ISO 19014 on providing evidence for correct tool behavior are not substantially different from the ones of the base norm IEC 61508. Part 4 of ISO 19014:2020 specifically recommends the use of tools: see, e.g., Annexes A.2, A.9, and A.14 of [6].

Part 4 of ISO 19014:2020 poses requirements for the use of any tool [5]: these are summarized in Table 6 and then expanded in Annexes A.20 (Tools with increased confidence from use or validation) and A.21 (Certified tools and certified translators).

The ECLAIR functionality described above is qualifiable in compliance with ISO 19014. 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 ISO 19014:2020 for any MPLr.

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] 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.

[5] ISO. *ISO 19014:2018: Earth-moving machinery — Functional safety — Part 1: Methodology to determine safety-related parts of the control system and performance requirements*. ISO, Geneva, Switzerland, August 2018.

[6] ISO. *ISO 19014:2020: Earth-moving machinery — Functional safety — Part 4: Design and evaluation of software and data transmission for safety-related parts of the control system*. ISO, Geneva, Switzerland, July 2020.

[7] 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.

[8] 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.

[9] 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.