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

Coverage of IEC 61508

1 Introduction to IEC 61508:2010

IEC 61508:2010, "Functional safety of electrical/electronic/programmable electronic safety-related systems," is a series of standards issued by IEC [5]. IEC 61508:2010 defines a generic approach for all safety lifecycle activities regarding systems comprised of electrical and/or electronic and/or programmable electronic (E/E/PE) elements that are used to implement safety functions. IEC 61508:2010 is applicable to all industries. Several product and application sector international standards based on the IEC 61508 series have been developed, but the general framework set out by IEC 61508:2010 is applied as is whenever a more specific standard is not available.

IEC 61508:2010 considers all software safety lifecycle phases (e.g., initial concept, design, implementation, operation, maintenance decommissioning) of E/E/PE systems that are used to perform safety functions. In particular, it provides a method for the development of the safety requirements specification necessary to achieve the required functional safety for E/E/PE safety-related systems. This is based on a risk-based approach based on the notion of *Safety Integrity Level* (SIL) for specifying the target level of safety integrity for the safety functions to be implemented by E/E/PE safety-related systems.

There are four SILs: 1, 2, 3 and 4, with 1 being the lowest safety integrity level and 4 being the highest. Each SIL correspond to a different range for the average probability of a dangerous failure; the ranges are different depending on the mode of operation of the system, (low demand, high demand or continuous). On the low demand mode of operation, SIL 4 corresponds to an average probability of a dangerous failure on demand of the safety function in the range $[10^{-5}, 10^{-4}]$. On the high demand and continuous modes of operation, SIL 4 corresponds to an average probability of a dangerous failure per hour of operation of the safety function in the range $[10^{-9}, 10^{-8}]$.

1.1 Role of ECLAIR in Ensuring Compliance with IEC 61508:2010

The ECLAIR Software Verification Platform can be used to comply with several of the techniques and measures required by IEC 61508:2010 Part 3 "Software Requirements" [7]. 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 IEC 61508:2010 Techniques and Measures

IEC 61508:2010 Part 3 applies to all software forming part of a safety-related system or used to develop a safety-related system within the scope of IEC 61508:2010 [7]. For such software, it specifies requirements for safety 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 B, which is informative, contains tables that expand upon some of the entries of the tables of Annex A.

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

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.

An appropriate combination of techniques and measures shall be selected according to the identified SIL. Techniques 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 1a, 1b, 1c, ... in the same column.

For consecutive entries, all listed as highly recommended and recommended techniques and measures, in accordance with the SIL, apply. For alternative entries, an appropriate combination of topics and methods shall be applied in accordance with the indicated SIL. If methods are listed with different degrees of recommendation for a SIL, the methods with the higher recommendation should be preferred. Guidance on the selection of an appropriate combination of techniques and measures is given in Annex C of [7]. Techniques and measures are described in IEC 61508:2010 Part 7 [9].

The following tables have been obtained by extending the corresponding tables in IEC 61508 Part 3 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 [10] as amended by [11, 12, 13], whereas MISRA C++ is [14]. 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 [10], with Amendments 2 [11] and 3 [12], and Technical Corrigendum 2 [13], 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".

Table A.1 — Software safety requirements specification

Technique/Measure		SIL 1	SIL 2	SIL 3	SIL 4	ECLAIR
1a	Semi-formal methods	R	R	HR	HR	–
1b	Formal methods	—	R	R	HR	✓[a]
2	Forward traceability between the system safety requirements and the software safety requirements	R	R	HR	HR	✓[b]
3	Backward traceability between the safety requirements and the perceived safety needs	R	R	HR	HR	✓[b]
4	Computer-aided specification tools to support appropriate techniques/measures above	R	R	HR	HR	✓[a,b]

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

2.2 MISRA C++:2008

MISRA C++:2008 [14] 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".

Table A.2 — Software design and development — software architecture design

Technique/Measure		SIL 1	SIL 2	SIL 3	SIL 4	ECLAIR
1	Fault detection	—	R	HR	HR	✓[a]
2	Error detecting codes	R	R	R	HR	–
3a	Failure assertion programming	R	R	R	HR	–
3b	Diverse monitor techniques (with independence between the monitor and the monitored function in the same computer)	—	R	R	—	✓[b]

continued

Table A.2 — Software design and development — software architecture design

Technique/Measure		SIL 1	SIL 2	SIL 3	SIL 4	ECLAIR
3c	Diverse monitor techniques (with separation between the monitor computer and the monitored computer)	—	R	R	HR	✓[b]
3d	Diverse redundancy, implementing the same software safety requirements specification	—	—	—	R	–
3e	Diverse redundancy, implementing the same software safety requirements specification	—	—	R	HR	–
3f	Backward recovery	R	R	—	NR	–
3g	Stateless software design (or limited state design)	—	—	R	HR	–
4a	Re-try fault recovery mechanisms	R	R	—	—	–
4b	Graceful degradation	R	R	HR	HR	–
5	Artificial intelligence - fault correction	—	NR	NR	NR	–
6	Dynamic reconfiguration	—	NR	NR	NR	–
7	Modular approach	HR	HR	HR	HR	✓[c]
8	Use of trusted/verified software elements (if available)	R	HR	HR	HR	–
9	Forward traceability between the software safety requirements specification and software architectures	R	R	HR	HR	✓[d]
10	Backward traceability between the software safety requirements specification and software architecture	R	R	HR	HR	–
11a	Structured diagrammatic methods	HR	HR	HR	HR	–
11b	Semi-formal methods	R	R	HR	HR	–
11c	Formal design and refinement methods	—	R	R	HR	–
11d	Automatic software generation	R	R	R	R	–
12	Computer-aided specification and design tools	R	R	HR	HR	✓[b]
13a	Cyclic behaviour, with guaranteed maximum cycle time	R	HR	HR	HR	–
13b	Time-triggered architecture	R	HR	HR	HR	–
13c	Event-driven, with guaranteed maximum response time	R	HR	HR	—	–
14	Static resource allocation	—	R	HR	HR	✓[e]
15	Static synchronisation of access to shared resources	—	—	R	HR	✓[f]

[a] 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.

[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. `B.PROJORG` is instrumental in proving independence among different software components.

[c] See Table B.9.

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

[e] 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.

[f] ECLAIR service for MISRA C:2012 Directive 4.13 allows specifying constraints on the sequence of actions operating on resources, including shared resources.

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 A.3 — Software design and development - support tools and programming language

	Technique/Measure	**SIL 1**	**SIL 2**	**SIL 3**	**SIL 4**	ECLAIR
1	Suitable programming language	HR	HR	HR	HR	✓[a]
2	Strongly typed programming language	HR	HR	HR	HR	✓[b]
3	Language subset	—	—	HR	HR	✓[a]
4a	Certified tools and certified translators	R	HR	HR	HR	✓[c]
4b	Tools and translators: increased confidence from use	HR	HR	HR	HR	✓[d]

[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] 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.

[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.

[d] 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.

Table A.4 — Software design and development - detailed design

	Technique/Measure	SIL 1	SIL 2	SIL 3	SIL 4	ECLAIR
1a	Structured methods	HR	HR	HR	HR	–
1b	Semi-formal methods	R	HR	HR	HR	–
1c	Formal design and refinement methods	—	R	R	HR	–
2	Computer-aided design tools	R	R	HR	HR	✓[a,f]
3	Defensive programming	—	R	HR	HR	✓[b]
4	Modular approach	HR	HR	HR	HR	✓[c]
5	Design and coding standards	R	HR	HR	HR	✓[d]
6	Structured programming	HR	HR	HR	HR	✓[e]
7	Use of trusted/verified software elements (if available)	R	HR	HR	HR	–
8	Forward traceability between the software safety requirements specification and software design	R	R	HR	HR	✓[f]

[a] 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.

[b] 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.

[c] See Table B.9.

[d] ECLAIR can be used to verify the compliance of source code to coding standards, such as the MISRA coding standards and BARR-C:2018. ECLAIR can also be used to verify that the value of software metrics are inside prescribed ranges.

[e] 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`.

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

Table A.5 — Software design and development software module testing and integration

	Technique/Measure	SIL 1	SIL 2	SIL 3	SIL 4	ECLAIR
1	Probabilistic testing	—	R	R	R	–
2	Dynamic analysis and testing	R	HR	HR	HR	–
3	Data recording and analysis	HR	HR	HR	HR	–
4	Functional and black box testing	HR	HR	HR	HR	–
5	Performance testing	R	R	HR	HR	–
6	Model based testing	R	R	HR	HR	–
7	Interface testing	R	R	HR	HR	–
8	Test management and automation tools	R	HR	HR	HR	–
9	Forward traceability between the software safety specification and the module and integration test specification	R	R	HR	HR	✓[a]
10	Formal verification	—	—	R	R	✓[b]

[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] 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:2012 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.

Table A.9 — Software verification

	Technique/Measure	SIL 1	SIL 2	SIL 3	SIL 4	ECLAIR
1	Formal proof	—	R	R	HR	–
2	Animation of specification and design	R	R	R	R	–
3	Static analysis	R	HR	HR	HR	✓[a]
4	Dynamic analysis and testing	R	HR	HR	HR	✓[b]
5	Forward traceability between the software design specification and the software verification (including data verification) plan	R	R	HR	HR	✓[c]
6	Backward traceability between the software verification (including data verification) plan and the software design specification	R	R	HR	HR	✓[c]
7	Offline numerical analysis	R	R	HR	HR	–

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

[b] 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:2012 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.

[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 B.1 — Design and coding standards

	Technique/Measure	SIL 1	SIL 2	SIL 3	SIL 4	ECLAIR
1	Use of coding standard to reduce likelihood of errors	HR	HR	HR	HR	✓[a]
2	No dynamic objects	R	HR	HR	HR	✓[b]
3a	No dynamic variables	—	R	HR	HR	✓[b]
3b	Online checking of the installation of dynamic variables	—	R	HR	HR	–
4	Limited use of interrupts	R	R	HR	HR	–
5	Limited use of pointers	—	R	HR	HR	✓[c]
6	Limited use of recursion	—	R	HR	HR	✓[d]
7	No unstructured control flow in programs in higher level languages	R	HR	HR	HR	✓[e]
8	No automatic type conversion	R	HR	HR	HR	✓[f]

[a] 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.

[b] 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.

[c] 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.

[d] 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.

[e] 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`.

[f] The MISRA C/C++ guidelines include several rules restricting the use of implicit conversions. E.g., for MISRA C:2012, Rules 10.1, 10.3, 10.4, 10.6, 10.7, 11.1, 11.2, 11.4, 11.5, and 11.9.

Table B.8 — Static Analysis

	Technique/Measure	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	R	HR	HR	HR	✓[a]
4	Data flow analysis	R	HR	HR	HR	✓[b]
5	Error guessing	R	R	R	R	–
6a	Formal inspections, including specific criteria	R	R	HR	HR	✓[c]
6b	Walk-through (software)	R	R	R	R	✓[c]
7	Symbolic execution	—	—	R	R	–
8	Design review	HR	HR	HR	HR	–
9	Static analysis of run time error behaviour	R	R	R	HR	✓[d]
10	Worst-case execution time analysis	R	R	R	R	–

[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.

[d] ECLAIR static analysis ensures that the semantics of the source code is well defined and that certain classes of run-time errors cannot occur.

Table B.9 — Modular Approach

	Technique/Measure	SIL 1	SIL 2	SIL 3	SIL 4	ECLAIR
1	Software module size limit	HR	HR	HR	HR	✓[a]
2	Software complexity control	R	R	HR	HR	✓[b]
3	Information hiding/encapsulation	R	HR	HR	HR	✓[c]
4	Parameter number limit / fixed number of subprogram parameters	R	R	R	R	✓[d]
5	One entry/one exit point in subroutines and functions	HR	HR	HR	HR	✓[e]
6	Fully defined interface	HR	HR	HR	HR	✓[f]

[a] 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).

[b] 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).

[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] HIS metrics counting function parameters and MISRA C/C++ guidelines on reduction of variables' scope allow limiting the number of (explicit and implicit) parameters.

[e] 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.

[f] 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.

2.5 ECLAIR Support for *Independence* in IEC 61508

In IEC 61508, the strongest incentive to ensure independence can be found in Part 1 [6, 7.6.2.10]:

> **7.6.2.10** For an E/E/PE safety-related system that implements safety functions of different safety integrity levels, unless it can be shown there is sufficient independence of implementation between these particular safety functions, those parts of the safety-related hardware and software where there is insufficient independence of implementation shall be treated as belonging to the safety function with the highest safety integrity level. Therefore, the requirements applicable to the highest relevant safety integrity level shall apply to all those parts.

For software, the requirements on independence and non-interference between safety functions and non-safety functions are given in IEC 1508 Part 3 [7, 7.4.2.8, 7.4.2.9]:

> **7.4.2.8** Where the software is to implement both safety and non-safety functions, then all of the software shall be treated as safety-related, unless adequate design measures ensure that the failures of non-safety functions cannot adversely affect safety functions.

> **7.4.2.9** Where the software is to implement safety functions of different safety integrity levels, then all of the software shall be treated as belonging to the highest safety integrity level, unless adequate independence between the safety functions of the different safety integrity levels can be shown in the design. It shall be demonstrated either (1) that independence is achieved by both in the spatial and temporal domains, or (2) that any violation of independence is controlled. The justification for independence shall be documented.

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.

3 ECLAIR Qualification in Compliance with IEC 61508

The ECLAIR functionality described above is qualifiable in compliance with IEC 61508: ECLAIR is a class T2 off-line support tool [8, Clause 3.2.11] and meets all the requirements set forth in IEC 61508 Part 3 for such tools [7, Clause 7.4.4]. 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. 01, attesting that the ECLAIR Software Verification Platform is suitable to be used in safety-related development projects according to IEC 61508:2010 for any SIL.

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

[6] IEC. *IEC 61508-1:2010: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems — Part 1: General Requirements*. IEC, Geneva, Switzerland, April 2010.

[7] IEC. *IEC 61508-3:2010: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems — Part 3: Software Requirements*. IEC, Geneva, Switzerland, April 2010.

[8] IEC. *IEC 61508-4:2010: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems — Part 4: Definitions and Abbreviations*. IEC, Geneva, Switzerland, April 2010.

[9] IEC. *IEC 61508-7:2010: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems — Part 7: Overview of Techniques and Measures*. IEC, Geneva, Switzerland, April 2010.

[10] 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.

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

[12] 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.

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

[14] 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.