Axivion Bauhaus Suite – Technical Factsheet AUTOSAR

Version 6.9.1 upwards

Contents

1. C++

1. Autosar C++14 Guidelines (AUTOSAR 17.03)

AUTOSAR Rule	Severity	Description
A0.1.1	Req	A project shall not contain instances of non-volatile variables being given values that are never subsequently used.
A0.1.2	Req	The value returned by a function having a non-void return type that is not an overloaded operator shall always be used.
A0.1.3	Req	Every defined function shall be called at least once.
A0.4.2	Req	Type long double shall not be used.
A1.1.1	Req	All code shall conform to ISO/IEC 14882:2014 - Programming Language C++ and shall not use deprecated features.
A1.4.1	Req	Code metrics and their valid boundaries shall be defined.
A2.2.1	Req	Only those characters specified in the C++ Language Standard basic source character set shall be used in the source code.
A2.5.1	Req	Trigraphs shall not be used.
A2.6.1	Req	Digraphs should not be used.
A2.8.1	Req	The character shall not occur as a last character of a C++ comment.
A2.8.2	Req	Sections of code shall not be "commented out".
A2.8.3	Req	All declarations of "user-defined" types, static and non-static data members, functions and methods shall be preceded by documentation using "///" comments and "@tag" tags.
A2.8.4	Req	C-style comments shall not be used.
A2.9.1	Req	A header file name shall be identical to a type name declared in it if it declares a type.
A2.11.1	Req	Identifiers declared in an inner scope shall not hide an identifier declared in an outer scope.

A2.11.2	Req	A "using" name shall be a unique identifier within a namespace.
A2.11.3	Req	A "user-defined" type name shall be a unique identifier within a namespace.
A2.11.4	Req	The identifier name of a non-member object or function with static storage duration should not be reused.
A2.11.5	Adv	The identifier name of a non-member object or function with static storage duration should not be reused.
A2.14.1	Req	Only those escape sequences that are defined in ISO/IEC 14882:2003 shall be used.
A2.14.2	Req	Narrow and wide string literals shall not be concatenated.
A2.14.3	Req	Type wchar_t shall not be used.
A3.1.1	Req	It shall be possible to include any header file in multiple translation units without violating the One Definition Rule.
A3.1.2	Req	Header files, that are defined locally in the project, shall have a file name' extension of one of: ".h", ".hpp" or ".hxx".
A3.1.3	Adv	Implementation files, that are defined locally in the project, should have a file name extension of ".cpp".
A3.1.4	Req	When an array is declared, its size shall either be stated explicitly or defined implicitly by initialization.
A3.3.1	Req	Objects or functions with external linkage shall be declared in a header file.
A3.3.2	Req	Non-POD type objects with static storage duration shall not be used.
A3.9.1	Req	Typedefs that indicate size and signedness should be used in place of the basic numerical types.
A4.5.1	Req	Expressions with type enum shall not be used as operands to built-in operators other than the subscript operator [], the assignment operator =, the equality operators == and !=, the unary & operator, and the relational operators <, <=, >, >=.
A4.7.1	Req	An integer expression shall not lead to data loss.

A4.10.1	Req	Only nullptr literal shall be used as the null-pointer-constant.
A5.0.1	Req	The value of an expression shall be the same under any order of evaluation that the standard permits.
A5.0.2	Req	The condition of an if-statement and the condition of an iteration-statement shall have type bool.
A5.0.3	Req	The declaration of objects should contain no more than two levels of pointer indirection.
A5.1.1	Req	Literal values shall not be used apart from type initialization, otherwise symbolic names shall be used instead.
A5.1.2	Req	Variables shall not be implicitly captured in a lambda expression.
A5.1.5	Adv	If a lambda expression is used in the same scope in which it has been defined, the lambda should capture objects by reference.
A5.1.7	Req	The underlying type of lambda expression shall not be used.
A5.1.8	Adv	Lambda expressions should not be defined inside another lambda expression.
A5.2.1	Adv	dynamic_cast should not be used.
A5.2.2	Req	Traditional C-style casts shall not be used.
A5.2.3	Req	A cast shall not remove any const or volatile qualification from the type of a pointer or reference.
A5.2.4	Req	reinterpret_cast shall not be used.
A5.3.1	Req	Evaluation of the operand to the typeid operator shall not contain side effects.
A5.5.1	Req	The right hand operand of the integer division or remainder operators shall not be equal to zero.
A5.10.1	Req	A pointer to member virtual function shall only be tested for equality with null-pointer-constant.
A5.16.1	Req	The ternary conditional operator shall not be used as a sub-expression.
A6.4.1	Req	Every switch statement shall have at least one case-clause.

A6.5.1	Req	A for-loop that loops through all elements of the container and does not use its loop-counter shall not be used.
A6.5.2	Req	A for loop shall contain a single loop-counter which shall not have floating type.
A6.6.1	Req	The goto statement shall not be used.
A7.1.1	Req	Constexpr or const specifiers shall be used for immutable data declaration.
A7.1.3	Req	CV-qualifiers shall be placed on the right hand side of the type that is a typedef or a using name.
A7.1.4	Req	The register keyword shall not be used.
A7.1.5	Req	The auto specifier shall not be used apart from following cases: (1) to declare that a variable has the same type as return type of a function call, (2) to declare that a variable has the same type as initializer of non-fundamental type, (3) to declare parameters of a generic lambda expression, (4) to declare a function template using trailing return type syntax.
A7.1.6	Req	The typedef specifier shall not be used.
A7.1.7	Req	Each expression statement and identifier declaration shall be placed on a separate line.
A7.2.1	Req	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration.
A7.2.2	Req	Enumeration underlying base type shall be explicitly defined.
A7.2.3	Req	Enumerations shall be declared as scoped enum classes.
A7.2.4	Req	In an enumerator list, the = construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized.
A7.4.1	Req	The asm declaration shall not be used.
A7.5.1	Req	A function shall not return a reference or a pointer to a parameter that is passed by reference or const reference.
A7.5.2	Req	Functions shall not call themselves, either directly or indirectly.

A8.2.1	Req	When declaring function templates, the trailing return type syntax shall be used if the return type depends on the type of parameters.
A8.4.1	Req	Functions shall not be defined using the ellipsis notation.
A8.4.2	Req	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.
A8.5.1	Req	In an initialization list, the order of initialization shall be following: (1) virtual base classes in depth and left to right order of the inheritance graph, (2) direct base classes in left to right order of inheritance list, (3) non-static data members in the order they were declared in the class definition.
A8.5.2	Req	Braced-initialization {}, without equals sign, shall be used for variable initialization.
A8.5.3	Req	A variable of type auto shall not be initialized using {} or ={} braced-initialization.
A8.5.4	Adv	A constructor taking parameter of type std::initializer_list shall only be defined in classes that internally store a collection of objects.
A9.6.1	Req	Bit-fields shall be either unsigned integral, or enumeration (with underlying type of unsigned integral type).
A10.1.1	Req	Class shall not be derived from more than one base class which is not an interface class.
A10.2.1	Req	Non-virtual member functions shall not be redefined in derived classes.
A10.3.1	Req	Virtual function declaration shall contain exactly one of the three specifiers: (1) virtual, (2) override, (3) final.
A10.3.2	Req	Each overriding virtual function shall be declared with the override or final specifier.
A10.3.3	Req	Virtual functions shall not be introduced in a final class.
A10.3.5	Req	A user-defined assignment operator shall not be virtual.
A11.0.1	Adv	A non-POD type should be defined as class.

A11.0.2	Req	A type defined as struct shall: (1) provide only public data members, (2) not provide any special member functions or methods, (3) not be a base of another struct or class, (4) not inherit from another struct or class.
A11.3.1	Req	Friend declarations shall not be used.
A12.0.1	Req	If a class declares a copy or move operation, or a destructor, either via "=default", "=delete", or via a user-provided declaration, then all others of these five special member functions shall be declared as well.
A12.1.1	Req	Constructors shall explicitly initialize all virtual base classes, all direct non-virtual base classes and all non-static data members.
A12.1.2	Req	Both NSDMI and a non-static member initializer in a constructor shall not be used in the same type.
A12.1.3	Req	If all user-defined constructors of a class initialize data members with constant values that are the same across all constructors, then data members shall be initialized using NSDMI instead.
A12.1.4	Req	All constructors that are callable with a single argument of fundamental type shall be declared explicit.
A12.4.1	Req	Destructor of a base class shall be public virtual, public override or protected non-virtual.
A12.4.2	Adv	If a public destructor of a class is non-virtual, then the class should be declared final.
A12.6.1	Req	All class data members that are initialized by the constructor shall be initialized using member initializers.
A12.8.1	Req	A copy constructor shall only initialize its base classes and the non-static members of the class of which it is a member.
A12.8.2	Adv	User-defined copy and move assignment operators should use user-defined no-throw swap function.
A12.8.3	Req	Moved-from object shall not be read-accessed.
A12.8.4	Req	Move constructor shall not initialize its class members and base classes using copy semantics.

A12.8.6	Req	Copy and move constructors and copy assignment and move assignment operators shall be declared protected or defined "=delete" in base class.
A12.8.7	Adv	Assignment operators should be declared with the ref-qualifier &.
A13.1.1	Req	User-defined literals shall not be used.
A13.1.2	Req	User defined suffixes of the user defined literal operators shall start with underscore followed by one or more letters.
A13.1.3	Req	User defined literals operators shall only perform conversion of passed parameters.
A13.2.1	Req	An assignment operator shall return a reference to "this".
A13.2.2	Req	A binary arithmetic operator and a bitwise operator shall return a "prvalue".
A13.2.3	Req	A relational operator shall return a boolean value.
A13.3.1	Req	A function that contains "forwarding reference" as its argument shall not be overloaded.
A13.5.1	Req	If "operator[]" is to be overloaded with a non-const version, const version shall also be implemented.
A13.6.1	Req	Digit sequences separators ' shall only be used as follows: (1) for decimal, every 3 digits, (2) for hexadecimal, every 2 digits, (3) for binary, every 4 digits.
A15.1.1	Req	Only instances of types derived from std::exception shall be thrown.
A15.1.2	Req	An exception object should not have pointer type.
A15.1.3	Adv	All thrown exceptions should be unique.
A15.2.1	Req	Constructors that are not noexcept shall not be invoked before program startup.
A15.3.1	Adv	Unchecked exceptions should be handled only in main or thread's main functions.

A15.3.3	Req	There should be at least one exception handler to catch all otherwise unhandled exceptions.
A15.3.4	Req	Catch-all (ellipsis and std::exception) handlers shall be used only in (a) main, (b) task main functions, (c) in functions that are supposed to isolate independent components and (d) when calling third-party code that uses exceptions not according to AUTOSAR C++14 guidelines.
A15.3.5	Req	A class type exception shall always be caught by reference.
A15.4.1	Req	Dynamic exception-specification shall not be used.
A15.4.2	Req	If a function is declared to be noexcept, noexcept(true) or noexcept(<true condition>), then it shall not exit with an exception.
A15.4.3	Req	Function's noexcept specification shall be either identical or more restrictive across all translation units and all overriders.
A15.4.4	Req	A declaration of non-throwing function shall contain noexcept specification.
A15.4.5	Req	Checked exceptions that could be thrown from a function shall be specified together with the function declaration and they shall be identical in all function declarations and for all its overriders.
A15.4.6	Adv	Unchecked exceptions should not be specified together with a function declaration.
A15.5.1	Req	All user-provided class destructors, deallocation functions, move constructors, move assignment operators and swap functions shall not exit with an exception. A noexcept exception specification shall be added to these functions as appropriate.
A15.5.2	Req	Program shall not be abruptly terminated. In particular, an implicit or explicit invocation of std::abort(), std::quick_exit(), std::_Exit(), std::terminate() shall not be done.
A15.5.3	Req	The terminate() function shall not be called implicitly.
A16.0.1	Req	The pre-processor shall only be used for file inclusion and include guards.

A16.2.1	Req	The ', ", /*, //, characters shall not occur in a header file name or in #include directive.
A16.6.1	Req	#error directive shall not be used.
A16.7.1	Req	The #pragma directive shall not be used.
A17.0.1	Req	Reserved identifiers, macros and functions in the standard library shall not be defined, redefined or undefined.
A17.0.2	Req	All project's code including used libraries (including standard and user-defined libraries) and any third-party user code shall conform to the AUTOSAR C++14 Coding Guidelines.
A18.0.1	Req	The C library shall not be used.
A18.0.2	Req	The library functions atof, atoi and atol from library <cstdlib> shall not be used.
A18.0.3	Req	The library <clocale> (locale.h) and the setlocale function shall not be used.
A18.1.1	Adv	C-style arrays should not be used.
A18.1.2	Req	The std::vector<bool> specialization shall not be used.
A18.1.3	Req	The std::auto_ptr type shall not be used.
A18.1.4	Req	A pointer pointing to an element of an array of objects shall not be passed to a smart pointer of single object type.
A18.1.5	Req	The std::unique_ptr shall not be passed to a function by const reference.
A18.5.1	Req	Functions malloc, calloc, realloc and free shall not be used.
A18.5.2	Req	Operators new and delete shall not be called explicitly.
A18.5.3	Req	The form of delete operator shall match the form of new operator used to allocate the memory.
A18.5.4	Req	If a project has sized or unsized version of operator "delete" globally defined, then both sized and unsized versions shall be defined.
A18.9.1	Req	The std::bind shall not be used.

A18.9.2	Req	Forwarding values to other functions shall be done via: (1) std::move if the value is an rvalue reference, (2) std::forward if the value is forwarding reference.
A18.9.3	Req	The std::move shall not be used on objects declared const or const&.
A23.0.1	Req	An iterator shall not be implicitly converted to const_iterator.
M0.1.1	Req	There shall be no unreachable code.
M0.1.2	Req	A project shall not contain infeasible paths.
M0.1.3	Req	A project shall not contain unused variables.
M0.1.4	Req	A project shall not contain non-volatile POD variables having only one use.
M0.1.5	Req	A project shall not contain unused type declarations.
M0.1.8	Req	All functions with void return type shall have external side effect(s).
M0.1.9	Req	There shall be no dead code.
M0.1.10	Adv	Every defined function shall be called at least once.
M0.2.1	Req	An object shall not be assigned to an overlapping object.
M0.3.1	Req	Minimization of run-time failures shall be ensured by the use of at least one of: (a) static analysis tools/techniques; (b) dynamic analysis tools/techniques; (c) explicit coding of checks to handle run-time faults.
M0.3.2	Req	If a function generates error information, then that error information shall be tested.
M0.4.2	Req	Use of floating-point arithmetic shall be documented.
M2.10.1	Req	Different identifiers shall be typographically unambiguous.
M2.10.3	Req	A typedef name (including qualification, if any) shall be a unique identifier.
M2.10.6	Req	If an identifier refers to a type, it shall not also refer to an object or a function in the same scope.

M2.13.2	Req	Octal constants (other than zero) and octal escape sequences (other than "0") shall not be used.
M2.13.3	Req	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.
M2.13.4	Req	Literal suffixes shall be upper case.
M3.1.2	Req	Functions shall not be declared at block scope.
M3.2.1	Req	All declarations of an object or function shall have compatible types.
M3.2.2	Req	The One Definition Rule shall not be violated.
M3.2.3	Req	A type, object or function that is used in multiple translation units shall be declared in one and only one file.
M3.2.4	Req	An identifier with external linkage shall have exactly one definition.
M3.3.2	Req	If a function has internal linkage then all re-declarations shall include the static storage class specifier.
M3.4.1	Req	An identifier declared to be an object or type shall be defined in a block that minimizes its visibility.
M3.9.1	Req	The types used for an object, a function return type, or a function parameter shall be token-for-token identical in all declarations and re-declarations.
M3.9.3	Req	The underlying bit representations of floating-point values shall not be used.
M4.5.1	Req	Expressions with type bool shall not be used as operands to built-in operators other than the assignment operator =, the logical operators &&, ||, !, the equality operators == and !=, the unary & operator, and the conditional operator.
M4.5.3	Req	Expressions with type (plain) char and wchar_t shall not be used as operands to built-in operators other than the assignment operator =, the equality operators == and !=, and the unary & operator.
M4.10.1	Req	NULL shall not be used as an integer value.

M4.10.2	Req	Literal zero (0) shall not be used as the null-pointer-constant.
M5.0.2	Adv	Limited dependence should be placed on C++ operator precedence rules in expressions.
M5.0.3	Req	A cvalue expression shall not be implicitly converted to a different underlying type.
M5.0.4	Req	An implicit integral conversion shall not change the signedness of the underlying type.
M5.0.5	Req	There shall be no implicit floating-integral conversions.
M5.0.6	Req	An implicit integral or floating-point conversion shall not reduce the size of the underlying type.
M5.0.7	Req	There shall be no explicit floating-integral conversions of a cvalue expression.
M5.0.8	Req	An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue expression.
M5.0.9	Req	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression.
M5.0.10	Req	If the bitwise operators ~ and << are applied to an operand with an underlying type of unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand.
M5.0.11	Req	The plain char type shall only be used for the storage and use of character values.
M5.0.12	Req	signed char and unsigned char type shall only be used for the storage and use of numeric values.
M5.0.14	Req	The first operand of a conditional-operator shall have type bool.
M5.0.15	Req	Array indexing shall be the only form of pointer arithmetic.
M5.0.16	Req	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array.
M5.0.17	Req	Subtraction between pointers shall only be applied to pointers that address elements of the same array.

M5.0.18	Req	>, >=, <, <= shall not be applied to objects of pointer type, except where they point to the same array.
M5.0.20	Req	Non-constant operands to a binary bitwise operator shall have the same underlying type.
M5.0.21	Req	Bitwise operators shall only be applied to operands of unsigned underlying type.
M5.2.1	Req	Each operand of a logical && or || shall be a postfix-expression.
M5.2.2	Req	A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast.
M5.2.3	Adv	Casts from a base class to a derived class should not be performed on polymorphic types.
M5.2.6	Req	A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type.
M5.2.8	Req	An object with integer type or pointer to void type shall not be converted to an object with pointer type.
M5.2.9	Req	A cast should not convert a pointer type to an integral type.
M5.2.10	Req	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.
M5.2.11	Req	The comma operator, && operator and the || operator shall not be overloaded.
M5.2.12	Req	An identifier with array type passed as a function argument shall not decay to a pointer.
M5.3.1	Req	Each operand of the ! operator, the logical && or the logical || operators shall have type bool.
M5.3.2	Req	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.
M5.3.3	Req	The unary & operator shall not be overloaded.
M5.3.4	Req	Evaluation of the operand to the sizeof operator shall not contain side effects.

M5.8.1	Req	The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.
M5.14.1	Req	The right-hand operand of a logical && or || operator shall not contain side effects.
M5.17.1	Req	The semantic equivalence between a binary operator and its assignment operator form shall be preserved.
M5.18.1	Req	The comma operator shall not be used.
M5.19.1	Req	Evaluation of constant unsigned integer expressions should not lead to wrap-around.
M6.2.1	Req	Assignment operators shall not be used in sub-expressions.
M6.2.2	Req	Floating-point expressions shall not be directly or indirectly tested for equality or inequality.
M6.2.3	Req	Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment, provided that the first character following the null statement is a white-space character.
M6.3.1	Req	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.
M6.4.1	Req	An if (condition) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement.
M6.4.2	Req	All if ... else if constructs shall be terminated with an else clause.
M6.4.3	Req	A switch statement shall be a well-formed switch statement.
M6.4.4	Req	A switch label shall only be used when the most closely-enclosing compound-statement is the body of a switch-statement.
M6.4.5	Req	An unconditional throw or break statement shall terminate every non-empty switch-clause.
M6.4.6	Req	The final clause of a switch statement shall be the default clause.

M6.4.7	Req	The condition of a switch statement shall not have bool type.
M6.5.2	Req	If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to <=, <, > or >=.
M6.5.3	Req	The loop-counter shall not be modified within condition or statement.
M6.5.4	Req	The loop-counter shall be modified by one of: --, ++, -=n, or +=n; where n remains constant for the duration of the loop.
M6.5.5	Req	A loop-control-variable other than the loop-counter shall not be modified within condition or expression.
M6.5.6	Req	A loop-control-variable other than the loop-counter which is modified in statement shall have type bool.
M6.6.1	Req	Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement.
M6.6.2	Req	The goto statement shall jump to a label declared later in the same function body.
M6.6.3	Req	The continue statement shall only be used within a well-formed for loop.
M7.1.2	Req	A pointer or reference parameter in a function shall be declared as pointer to const or reference to const if the corresponding object is not modified.
M7.3.1	Req	The global namespace shall only contain main, namespace declarations and extern "C" declarations.
M7.3.2	Req	The identifier main shall not be used for a function other than the global function main.
M7.3.3	Req	There shall be no unnamed namespaces in header files.
M7.3.4	Req	Using-directives shall not be used.
M7.3.5	Req	Multiple declarations for an identifier in the same namespace shall not straddle a using-declaration for that identifier.

M7.3.6	Req	using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.
M7.4.1	Req	All usage of assembler shall be documented.
M7.4.2	Req	Assembler instructions shall only be introduced using the asm declaration.
M7.4.3	Req	Assembly language shall be encapsulated and isolated.
M7.5.1	Req	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.
M7.5.2	Req	The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.
M8.0.1	Req	An init-declarator-list or a member-declarator-list shall consist of a single init-declarator or member-declarator respectively.
M8.3.1	Req	Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or else shall not specify any default arguments.
M8.4.2	Req	The identifiers used for the parameters in a re-declaration of a function shall be identical to those in the declaration.
M8.4.4	Req	A function identifier shall either be used to call the function or it shall be preceded by &.
M8.5.1	Req	All variables shall have a defined value before they are used.
M8.5.2	Req	Braces shall be used to indicate and match the structure in the non-zero initialization of arrays and structures.
M9.3.1	Req	const member functions shall not return non-const pointers or references to class-data.
M9.3.3	Req	If a member function can be made static then it shall be made static, otherwise if it can be made const then it shall be made const.

M9.6.1	Req	When the absolute positioning of bits representing a bit-field is required, then the behaviour and packing of bit-fields shall be documented.
M10.1.1	Adv	Classes should not be derived from virtual bases.
M10.1.2	Req	A base class shall only be declared virtual if it is used in a diamond hierarchy.
M10.1.3	Req	An accessible base class shall not be both virtual and non-virtual in the same hierarchy.
M10.2.1	Adv	All accessible entity names within a multiple inheritance hierarchy should be unique.
M10.3.3	Req	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual.
M11.0.1	Req	Member data in non-POD class types shall be private.
M12.1.1	Req	An object's dynamic type shall not be used from the body of its constructor or destructor.
M14.5.2	Req	A copy constructor shall be declared when there is a template constructor with a single parameter that is a generic parameter.
M14.5.3	Req	A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter.
M14.6.1	Req	In a class template with a dependent base, any name that may be found in that dependent base shall be referred to using a qualified-id or this->.
M14.7.3	Req	All partial and explicit specializations for a template shall be declared in the same file as the declaration of their primary template.
M14.8.1	Req	Overloaded function templates shall not be explicitly specialized.
M15.0.3	Req	Control shall not be transferred into a try or catch block using a goto or a switch statement.
M15.1.1	Req	The assignment-expression of a throw statement shall not itself cause an exception to be thrown.

M15.1.2	Req	NULL shall not be thrown explicitly.
M15.1.3	Req	An empty throw (throw;) shall only be used in the compound-statement of a catch handler.
M15.3.1	Req	Exceptions shall be raised only after start-up and before termination of the program.
M15.3.3	Req	Handlers of a function-try-block implementation of a class constructor or destructor shall not reference non-static members from this class or its bases.
M15.3.4	Req	Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point.
M15.3.6	Req	Where multiple handlers are provided in a single try-catch statement or function-try-block for a derived class and some or all of its bases, the handlers shall be ordered most-derived to base class.
M15.3.7	Req	Where multiple handlers are provided in a single try-catch statement or function-try-block, any ellipsis (catch-all) handler shall occur last.
M16.0.1	Req	#include directives in a file shall only be preceded by other preprocessor directives or comments.
M16.0.2	Req	Macros shall only be #define'd or #undef'd in the global namespace.
M16.0.5	Req	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.
M16.0.6	Req	In the definition of a function-like macro, each instance of a parameter shall be enclosed in parentheses, unless it is used as the operand of # or ##.
M16.0.7	Req	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator.
M16.0.8	Req	If the # token appears as the first token on a line, then it shall be immediately followed by a preprocessing token.

M16.1.1	Req	The defined preprocessor operator shall only be used in one of the two standard forms.
M16.1.2	Req	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related.
M16.2.3	Req	Include guards shall be provided.
M16.3.1	Req	There shall be at most one occurrence of the # or ## operators in a single macro definition.
M16.3.2	Adv	The # and ## operators should not be used.
M17.0.2	Req	The names of standard library macros and objects shall not be reused.
M17.0.3	Req	The names of standard library functions shall not be overridden.
M17.0.5	Req	The setjmp macro and the longjmp function shall not be used.
M18.0.3	Req	The library functions abort, exit, getenv and system from library <cstdlib> shall not be used.
M18.0.4	Req	The time handling functions of library <ctime> shall not be used.
M18.0.5	Req	The unbounded functions of library <cstring> shall not be used.
M18.2.1	Req	The macro offsetof shall not be used.
M18.7.1	Req	The signal handling facilities of <csignal> shall not be used.
M19.3.1	Req	The error indicator errno shall not be used.
M27.0.1	Req	The stream input/output library <cstdio> shall not be used.

2. Autosar C++14 Guidelines (AUTOSAR 17.10)

AUTOSAR Rule	Severity	Description
A0.1.1	Req	A project shall not contain instances of non-volatile variables being given values that are never subsequently used.

A0.1.2	Req	The value returned by a function having a non-void return type that is not an overloaded operator shall always be used.
A0.1.3	Req	Every defined function shall be called at least once.
A0.1.4	Req	There shall be no unused named parameters in non-virtual functions.
A0.1.5	Req	There shall be no unused named parameters in the set of parameters for a virtual function and all the functions that override it.
A0.4.2	Req	Type long double shall not be used.
A1.1.1	Req	All code shall conform to ISO/IEC 14882:2014 - Programming Language C++ and shall not use deprecated features.
A1.4.1	Req	Code metrics and their valid boundaries shall be defined.
A2.2.1	Req	Only those characters specified in the C++ Language Standard basic source character set shall be used in the source code.
A2.5.1	Req	Trigraphs shall not be used.
A2.6.1	Req	Digraphs should not be used.
A2.8.1	Req	The character shall not occur as a last character of a C++ comment.
A2.8.2	Req	Sections of code shall not be "commented out".
A2.8.3	Req	All declarations of "user-defined" types, static and non-static data members, functions and methods shall be preceded by documentation using "///" comments and "@tag" tags.
A2.8.4	Req	C-style comments shall not be used.
A2.9.1	Req	A header file name shall be identical to a type name declared in it if it declares a type.
A2.11.1	Req	Identifiers declared in an inner scope shall not hide an identifier declared in an outer scope.
A2.11.2	Req	A "using" name shall be a unique identifier within a namespace.

A2.11.3	Req	A "user-defined" type name shall be a unique identifier within a namespace.
A2.11.4	Req	The identifier name of a non-member object or function with static storage duration should not be reused.
A2.11.5	Adv	The identifier name of a non-member object or function with static storage duration should not be reused.
A2.14.1	Req	Only those escape sequences that are defined in ISO/IEC 14882:2003 shall be used.
A2.14.2	Req	Narrow and wide string literals shall not be concatenated.
A2.14.3	Req	Type wchar_t shall not be used.
A3.1.1	Req	It shall be possible to include any header file in multiple translation units without violating the One Definition Rule.
A3.1.2	Req	Header files, that are defined locally in the project, shall have a file name' extension of one of: ".h", ".hpp" or ".hxx".
A3.1.3	Adv	Implementation files, that are defined locally in the project, should have a file name extension of ".cpp".
A3.1.4	Req	When an array is declared, its size shall either be stated explicitly or defined implicitly by initialization.
A3.3.1	Req	Objects or functions with external linkage shall be declared in a header file.
A3.3.2	Req	Non-POD type objects with static storage duration shall not be used.
A3.9.1	Req	Typedefs that indicate size and signedness should be used in place of the basic numerical types.
A4.5.1	Req	Expressions with type enum shall not be used as operands to built-in operators other than the subscript operator [], the assignment operator =, the equality operators == and !=, the unary & operator, and the relational operators <, <=, >, >=.
A4.7.1	Req	An integer expression shall not lead to data loss.
A4.10.1	Req	Only nullptr literal shall be used as the null-pointer-constant.

A5.0.1	Req	The value of an expression shall be the same under any order of evaluation that the standard permits.
A5.0.2	Req	The condition of an if-statement and the condition of an iteration-statement shall have type bool.
A5.0.3	Req	The declaration of objects should contain no more than two levels of pointer indirection.
A5.1.1	Req	Literal values shall not be used apart from type initialization, otherwise symbolic names shall be used instead.
A5.1.2	Req	Variables shall not be implicitly captured in a lambda expression.
A5.1.5	Adv	If a lambda expression is used in the same scope in which it has been defined, the lambda should capture objects by reference.
A5.1.7	Req	The underlying type of lambda expression shall not be used.
A5.1.8	Adv	Lambda expressions should not be defined inside another lambda expression.
A5.2.1	Adv	dynamic_cast should not be used.
A5.2.2	Req	Traditional C-style casts shall not be used.
A5.2.3	Req	A cast shall not remove any const or volatile qualification from the type of a pointer or reference.
A5.2.4	Req	reinterpret_cast shall not be used.
A5.3.1	Req	Evaluation of the operand to the typeid operator shall not contain side effects.
A5.5.1	Req	The right hand operand of the integer division or remainder operators shall not be equal to zero.
A5.10.1	Req	A pointer to member virtual function shall only be tested for equality with null-pointer-constant.
A5.16.1	Req	The ternary conditional operator shall not be used as a sub-expression.
A6.4.1	Req	Every switch statement shall have at least one case-clause.

A6.5.1	Req	A for-loop that loops through all elements of the container and does not use its loop-counter shall not be used.
A6.5.2	Req	A for loop shall contain a single loop-counter which shall not have floating type.
A6.5.3	Adv	Do statements should not be used.
A6.6.1	Req	The goto statement shall not be used.
A7.1.1	Req	Constexpr or const specifiers shall be used for immutable data declaration.
A7.1.3	Req	CV-qualifiers shall be placed on the right hand side of the type that is a typedef or a using name.
A7.1.4	Req	The register keyword shall not be used.
A7.1.5	Req	The auto specifier shall not be used apart from following cases: (1) to declare that a variable has the same type as return type of a function call, (2) to declare that a variable has the same type as initializer of non-fundamental type, (3) to declare parameters of a generic lambda expression, (4) to declare a function template using trailing return type syntax.
A7.1.6	Req	The typedef specifier shall not be used.
A7.1.7	Req	Each expression statement and identifier declaration shall be placed on a separate line.
A7.2.1	Req	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration.
A7.2.2	Req	Enumeration underlying base type shall be explicitly defined.
A7.2.3	Req	Enumerations shall be declared as scoped enum classes.
A7.2.4	Req	In an enumerator list, the = construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized.
A7.4.1	Req	The asm declaration shall not be used.
A7.5.1	Req	A function shall not return a reference or a pointer to a parameter that is passed by reference or const reference.

A7.5.2	Req	Functions shall not call themselves, either directly or indirectly.
A8.2.1	Req	When declaring function templates, the trailing return type syntax shall be used if the return type depends on the type of parameters.
A8.4.1	Req	Functions shall not be defined using the ellipsis notation.
A8.4.2	Req	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.
A8.4.4	Adv	Multiple output values from a function should be returned as a struct or tuple.
A8.5.1	Req	In an initialization list, the order of initialization shall be following: (1) virtual base classes in depth and left to right order of the inheritance graph, (2) direct base classes in left to right order of inheritance list, (3) non-static data members in the order they were declared in the class definition.
A8.5.2	Req	Braced-initialization {}, without equals sign, shall be used for variable initialization.
A8.5.3	Req	A variable of type auto shall not be initialized using {} or ={} braced-initialization.
A8.5.4	Adv	A constructor taking parameter of type std::initializer_list shall only be defined in classes that internally store a collection of objects.
A9.5.1	Req	Unions shall not be used.
A9.6.1	Req	Bit-fields shall be either unsigned integral, or enumeration (with underlying type of unsigned integral type).
A10.1.1	Req	Class shall not be derived from more than one base class which is not an interface class.
A10.2.1	Req	Non-virtual member functions shall not be redefined in derived classes.
A10.3.1	Req	Virtual function declaration shall contain exactly one of the three specifiers: (1) virtual, (2) override, (3) final.

A10.3.2	Req	Each overriding virtual function shall be declared with the override or final specifier.
A10.3.3	Req	Virtual functions shall not be introduced in a final class.
A10.3.5	Req	A user-defined assignment operator shall not be virtual.
A11.0.1	Adv	A non-POD type should be defined as class.
A11.0.2	Req	A type defined as struct shall: (1) provide only public data members, (2) not provide any special member functions or methods, (3) not be a base of another struct or class, (4) not inherit from another struct or class.
A11.3.1	Req	Friend declarations shall not be used.
A12.0.1	Req	If a class declares a copy or move operation, or a destructor, either via "=default", "=delete", or via a user-provided declaration, then all others of these five special member functions shall be declared as well.
A12.1.1	Req	Constructors shall explicitly initialize all virtual base classes, all direct non-virtual base classes and all non-static data members.
A12.1.2	Req	Both NSDMI and a non-static member initializer in a constructor shall not be used in the same type.
A12.1.3	Req	If all user-defined constructors of a class initialize data members with constant values that are the same across all constructors, then data members shall be initialized using NSDMI instead.
A12.1.4	Req	All constructors that are callable with a single argument of fundamental type shall be declared explicit.
A12.1.5	Req	Common class initialization for non-constant members shall be done by a delegating constructor.
A12.1.6	Req	Derived classes that do not need further explicit initialization and require all the constructors from the base class shall use inheriting constructors.
A12.4.1	Req	Destructor of a base class shall be public virtual, public override or protected non-virtual.

A12.4.2	Adv	If a public destructor of a class is non-virtual, then the class should be declared final.
A12.6.1	Req	All class data members that are initialized by the constructor shall be initialized using member initializers.
A12.8.1	Req	A copy constructor shall only initialize its base classes and the non-static members of the class of which it is a member.
A12.8.2	Adv	User-defined copy and move assignment operators should use user-defined no-throw swap function.
A12.8.3	Req	Moved-from object shall not be read-accessed.
A12.8.4	Req	Move constructor shall not initialize its class members and base classes using copy semantics.
A12.8.6	Req	Copy and move constructors and copy assignment and move assignment operators shall be declared protected or defined "=delete" in base class.
A12.8.7	Adv	Assignment operators should be declared with the ref-qualifier &.
A13.1.1	Req	User-defined literals shall not be used.
A13.1.2	Req	User defined suffixes of the user defined literal operators shall start with underscore followed by one or more letters.
A13.1.3	Req	User defined literals operators shall only perform conversion of passed parameters.
A13.2.1	Req	An assignment operator shall return a reference to "this".
A13.2.2	Req	A binary arithmetic operator and a bitwise operator shall return a "prvalue".
A13.2.3	Req	A relational operator shall return a boolean value.
A13.3.1	Req	A function that contains "forwarding reference" as its argument shall not be overloaded.
A13.5.1	Req	If "operator[]" is to be overloaded with a non-const version, const version shall also be implemented.
A13.5.2	Req	All user-defined conversion operators shall be defined explicit.

A13.6.1	Req	Digit sequences separators ' shall only be used as follows: (1) for decimal, every 3 digits, (2) for hexadecimal, every 2 digits, (3) for binary, every 4 digits.
A15.1.1	Req	Only instances of types derived from std::exception shall be thrown.
A15.1.2	Req	An exception object should not have pointer type.
A15.1.3	Adv	All thrown exceptions should be unique.
A15.2.1	Req	Constructors that are not noexcept shall not be invoked before program startup.
A15.3.1	Adv	Unchecked exceptions should be handled only in main or thread's main functions.
A15.3.3	Req	There should be at least one exception handler to catch all otherwise unhandled exceptions.
A15.3.4	Req	Catch-all (ellipsis and std::exception) handlers shall be used only in (a) main, (b) task main functions, (c) in functions that are supposed to isolate independent components and (d) when calling third-party code that uses exceptions not according to AUTOSAR C++14 guidelines.
A15.3.5	Req	A class type exception shall always be caught by reference.
A15.4.1	Req	Dynamic exception-specification shall not be used.
A15.4.2	Req	If a function is declared to be noexcept, noexcept(true) or noexcept(<true condition>), then it shall not exit with an exception.
A15.4.3	Req	Function's noexcept specification shall be either identical or more restrictive across all translation units and all overriders.
A15.4.4	Req	A declaration of non-throwing function shall contain noexcept specification.
A15.4.5	Req	Checked exceptions that could be thrown from a function shall be specified together with the function declaration and they shall be identical in all function declarations and for all its overriders.

A15.4.6	Adv	Unchecked exceptions should not be specified together with a function declaration.
A15.5.1	Req	All user-provided class destructors, deallocation functions, move constructors, move assignment operators and swap functions shall not exit with an exception. A noexcept exception specification shall be added to these functions as appropriate.
A15.5.2	Req	Program shall not be abruptly terminated. In particular, an implicit or explicit invocation of std::abort(), std::quick_exit(), std::_Exit(), std::terminate() shall not be done.
A15.5.3	Req	The terminate() function shall not be called implicitly.
A16.0.1	Req	The pre-processor shall only be used for file inclusion and include guards.
A16.2.1	Req	The ', ", /*, //, characters shall not occur in a header file name or in #include directive.
A16.6.1	Req	#error directive shall not be used.
A16.7.1	Req	The #pragma directive shall not be used.
A17.0.1	Req	Reserved identifiers, macros and functions in the standard library shall not be defined, redefined or undefined.
A17.0.2	Req	All project's code including used libraries (including standard and user-defined libraries) and any third-party user code shall conform to the AUTOSAR C++14 Coding Guidelines.
A18.0.1	Req	The C library shall not be used.
A18.0.2	Req	The library functions atof, atoi and atol from library <cstdlib> shall not be used.
A18.0.3	Req	The library <clocale> (locale.h) and the setlocale function shall not be used.
A18.1.1	Adv	C-style arrays should not be used.
A18.1.2	Req	The std::vector<bool> specialization shall not be used.
A18.1.3	Req	The std::auto_ptr type shall not be used.

A18.1.4	Req	A pointer pointing to an element of an array of objects shall not be passed to a smart pointer of single object type.
A18.1.5	Req	The std::unique_ptr shall not be passed to a function by const reference.
A18.1.6	Req	All std::hash specializations for user-defined types shall have a noexcept function call operator.
A18.5.1	Req	Functions malloc, calloc, realloc and free shall not be used.
A18.5.2	Req	Operators new and delete shall not be called explicitly.
A18.5.3	Req	The form of delete operator shall match the form of new operator used to allocate the memory.
A18.5.4	Req	If a project has sized or unsized version of operator "delete" globally defined, then both sized and unsized versions shall be defined.
A18.5.8	Req	Objects that do not outlive a function shall have automatic storage duration.
A18.9.1	Req	The std::bind shall not be used.
A18.9.2	Req	Forwarding values to other functions shall be done via: (1) std::move if the value is an rvalue reference, (2) std::forward if the value is forwarding reference.
A18.9.3	Req	The std::move shall not be used on objects declared const or const&.
A23.0.1	Req	An iterator shall not be implicitly converted to const_iterator.
M0.1.1	Req	There shall be no unreachable code.
M0.1.2	Req	A project shall not contain infeasible paths.
M0.1.3	Req	A project shall not contain unused variables.
M0.1.4	Req	A project shall not contain non-volatile POD variables having only one use.
M0.1.5	Req	A project shall not contain unused type declarations.

M0.1.8	Req	All functions with void return type shall have external side effect(s).
M0.1.9	Req	There shall be no dead code.
M0.1.10	Adv	Every defined function shall be called at least once.
M0.2.1	Req	An object shall not be assigned to an overlapping object.
M0.3.1	Req	Minimization of run-time failures shall be ensured by the use of at least one of: (a) static analysis tools/techniques; (b) dynamic analysis tools/techniques; (c) explicit coding of checks to handle run-time faults.
M0.3.2	Req	If a function generates error information, then that error information shall be tested.
M0.4.2	Req	Use of floating-point arithmetic shall be documented.
M2.10.1	Req	Different identifiers shall be typographically unambiguous.
M2.10.3	Req	A typedef name (including qualification, if any) shall be a unique identifier.
M2.10.6	Req	If an identifier refers to a type, it shall not also refer to an object or a function in the same scope.
M2.13.2	Req	Octal constants (other than zero) and octal escape sequences (other than "0") shall not be used.
M2.13.3	Req	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.
M2.13.4	Req	Literal suffixes shall be upper case.
M3.1.2	Req	Functions shall not be declared at block scope.
M3.2.1	Req	All declarations of an object or function shall have compatible types.
M3.2.2	Req	The One Definition Rule shall not be violated.
M3.2.3	Req	A type, object or function that is used in multiple translation units shall be declared in one and only one file.

M3.2.4	Req	An identifier with external linkage shall have exactly one definition.
M3.3.2	Req	If a function has internal linkage then all re-declarations shall include the static storage class specifier.
M3.4.1	Req	An identifier declared to be an object or type shall be defined in a block that minimizes its visibility.
M3.9.1	Req	The types used for an object, a function return type, or a function parameter shall be token-for-token identical in all declarations and re-declarations.
M3.9.3	Req	The underlying bit representations of floating-point values shall not be used.
M4.5.1	Req	Expressions with type bool shall not be used as operands to built-in operators other than the assignment operator =, the logical operators &&, ||, !, the equality operators == and !=, the unary & operator, and the conditional operator.
M4.5.3	Req	Expressions with type (plain) char and wchar_t shall not be used as operands to built-in operators other than the assignment operator =, the equality operators == and !=, and the unary & operator.
M4.10.1	Req	NULL shall not be used as an integer value.
M4.10.2	Req	Literal zero (0) shall not be used as the null-pointer-constant.
M5.0.2	Adv	Limited dependence should be placed on C++ operator precedence rules in expressions.
M5.0.3	Req	A cvalue expression shall not be implicitly converted to a different underlying type.
M5.0.4	Req	An implicit integral conversion shall not change the signedness of the underlying type.
M5.0.5	Req	There shall be no implicit floating-integral conversions.
M5.0.6	Req	An implicit integral or floating-point conversion shall not reduce the size of the underlying type.

M5.0.7	Req	There shall be no explicit floating-integral conversions of a cvalue expression.
M5.0.8	Req	An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue expression.
M5.0.9	Req	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression.
M5.0.10	Req	If the bitwise operators ~ and << are applied to an operand with an underlying type of unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand.
M5.0.11	Req	The plain char type shall only be used for the storage and use of character values.
M5.0.12	Req	signed char and unsigned char type shall only be used for the storage and use of numeric values.
M5.0.14	Req	The first operand of a conditional-operator shall have type bool.
M5.0.15	Req	Array indexing shall be the only form of pointer arithmetic.
M5.0.16	Req	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array.
M5.0.17	Req	Subtraction between pointers shall only be applied to pointers that address elements of the same array.
M5.0.18	Req	>, >=, <, <= shall not be applied to objects of pointer type, except where they point to the same array.
M5.0.20	Req	Non-constant operands to a binary bitwise operator shall have the same underlying type.
M5.0.21	Req	Bitwise operators shall only be applied to operands of unsigned underlying type.
M5.2.1	Req	Each operand of a logical && or || shall be a postfix-expression.
M5.2.2	Req	A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast.

M5.2.3	Adv	Casts from a base class to a derived class should not be performed on polymorphic types.
M5.2.6	Req	A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type.
M5.2.8	Req	An object with integer type or pointer to void type shall not be converted to an object with pointer type.
M5.2.9	Req	A cast should not convert a pointer type to an integral type.
M5.2.10	Req	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.
M5.2.11	Req	The comma operator, && operator and the || operator shall not be overloaded.
M5.2.12	Req	An identifier with array type passed as a function argument shall not decay to a pointer.
M5.3.1	Req	Each operand of the ! operator, the logical && or the logical || operators shall have type bool.
M5.3.2	Req	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.
M5.3.3	Req	The unary & operator shall not be overloaded.
M5.3.4	Req	Evaluation of the operand to the sizeof operator shall not contain side effects.
M5.8.1	Req	The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.
M5.14.1	Req	The right-hand operand of a logical && or || operator shall not contain side effects.
M5.17.1	Req	The semantic equivalence between a binary operator and its assignment operator form shall be preserved.
M5.18.1	Req	The comma operator shall not be used.
M5.19.1	Req	Evaluation of constant unsigned integer expressions should not lead to wrap-around.

M6.2.1	Req	Assignment operators shall not be used in sub-expressions.
M6.2.2	Req	Floating-point expressions shall not be directly or indirectly tested for equality or inequality.
M6.2.3	Req	Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment, provided that the first character following the null statement is a white-space character.
M6.3.1	Req	The statement forming the body of a switch, while, do … while or for statement shall be a compound statement.
M6.4.1	Req	An if (condition) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement.
M6.4.2	Req	All if … else if constructs shall be terminated with an else clause.
M6.4.3	Req	A switch statement shall be a well-formed switch statement.
M6.4.4	Req	A switch label shall only be used when the most closely-enclosing compound-statement is the body of a switch-statement.
M6.4.5	Req	An unconditional throw or break statement shall terminate every non-empty switch-clause.
M6.4.6	Req	The final clause of a switch statement shall be the default clause.
M6.4.7	Req	The condition of a switch statement shall not have bool type.
M6.5.2	Req	If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to <=, <, > or >=.
M6.5.3	Req	The loop-counter shall not be modified within condition or statement.
M6.5.4	Req	The loop-counter shall be modified by one of: --, ++, -=n, or +=n; where n remains constant for the duration of the loop.
M6.5.5	Req	A loop-control-variable other than the loop-counter shall not be modified within condition or expression.

M6.5.6	Req	A loop-control-variable other than the loop-counter which is modified in statement shall have type bool.
M6.6.1	Req	Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement.
M6.6.2	Req	The goto statement shall jump to a label declared later in the same function body.
M6.6.3	Req	The continue statement shall only be used within a well-formed for loop.
M7.1.2	Req	A pointer or reference parameter in a function shall be declared as pointer to const or reference to const if the corresponding object is not modified.
M7.3.1	Req	The global namespace shall only contain main, namespace declarations and extern "C" declarations.
M7.3.2	Req	The identifier main shall not be used for a function other than the global function main.
M7.3.3	Req	There shall be no unnamed namespaces in header files.
M7.3.4	Req	Using-directives shall not be used.
M7.3.5	Req	Multiple declarations for an identifier in the same namespace shall not straddle a using-declaration for that identifier.
M7.3.6	Req	using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.
M7.4.1	Req	All usage of assembler shall be documented.
M7.4.2	Req	Assembler instructions shall only be introduced using the asm declaration.
M7.4.3	Req	Assembly language shall be encapsulated and isolated.
M7.5.1	Req	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.

M7.5.2	Req	The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.
M8.0.1	Req	An init-declarator-list or a member-declarator-list shall consist of a single init-declarator or member-declarator respectively.
M8.3.1	Req	Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or else shall not specify any default arguments.
M8.4.2	Req	The identifiers used for the parameters in a re-declaration of a function shall be identical to those in the declaration.
M8.4.4	Req	A function identifier shall either be used to call the function or it shall be preceded by &.
M8.5.1	Req	All variables shall have a defined value before they are used.
M8.5.2	Req	Braces shall be used to indicate and match the structure in the non-zero initialization of arrays and structures.
M9.3.1	Req	const member functions shall not return non-const pointers or references to class-data.
M9.3.3	Req	If a member function can be made static then it shall be made static, otherwise if it can be made const then it shall be made const.
M9.6.1	Req	When the absolute positioning of bits representing a bit-field is required, then the behaviour and packing of bit-fields shall be documented.
M10.1.1	Adv	Classes should not be derived from virtual bases.
M10.1.2	Req	A base class shall only be declared virtual if it is used in a diamond hierarchy.
M10.1.3	Req	An accessible base class shall not be both virtual and non-virtual in the same hierarchy.
M10.2.1	Adv	All accessible entity names within a multiple inheritance hierarchy should be unique.

M10.3.3	Req	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual.
M11.0.1	Req	Member data in non-POD class types shall be private.
M12.1.1	Req	An object's dynamic type shall not be used from the body of its constructor or destructor.
M14.5.2	Req	A copy constructor shall be declared when there is a template constructor with a single parameter that is a generic parameter.
M14.5.3	Req	A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter.
M14.6.1	Req	In a class template with a dependent base, any name that may be found in that dependent base shall be referred to using a qualified-id or this->.
M14.7.3	Req	All partial and explicit specializations for a template shall be declared in the same file as the declaration of their primary template.
M14.8.1	Req	Overloaded function templates shall not be explicitly specialized.
M15.0.3	Req	Control shall not be transferred into a try or catch block using a goto or a switch statement.
M15.1.1	Req	The assignment-expression of a throw statement shall not itself cause an exception to be thrown.
M15.1.2	Req	NULL shall not be thrown explicitly.
M15.1.3	Req	An empty throw (throw;) shall only be used in the compound-statement of a catch handler.
M15.3.1	Req	Exceptions shall be raised only after start-up and before termination of the program.
M15.3.3	Req	Handlers of a function-try-block implementation of a class constructor or destructor shall not reference non-static members from this class or its bases.
M15.3.4	Req	Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point.

M15.3.6	Req	Where multiple handlers are provided in a single try-catch statement or function-try-block for a derived class and some or all of its bases, the handlers shall be ordered most-derived to base class.
M15.3.7	Req	Where multiple handlers are provided in a single try-catch statement or function-try-block, any ellipsis (catch-all) handler shall occur last.
M16.0.1	Req	#include directives in a file shall only be preceded by other preprocessor directives or comments.
M16.0.2	Req	Macros shall only be #define'd or #undef'd in the global namespace.
M16.0.5	Req	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.
M16.0.6	Req	In the definition of a function-like macro, each instance of a parameter shall be enclosed in parentheses, unless it is used as the operand of # or ##.
M16.0.7	Req	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator.
M16.0.8	Req	If the # token appears as the first token on a line, then it shall be immediately followed by a preprocessing token.
M16.1.1	Req	The defined preprocessor operator shall only be used in one of the two standard forms.
M16.1.2	Req	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related.
M16.2.3	Req	Include guards shall be provided.
M16.3.1	Req	There shall be at most one occurrence of the # or ## operators in a single macro definition.
M16.3.2	Adv	The # and ## operators should not be used.
M17.0.2	Req	The names of standard library macros and objects shall not be reused.

M17.0.3	Req	The names of standard library functions shall not be overridden.
M17.0.5	Req	The setjmp macro and the longjmp function shall not be used.
M18.0.3	Req	The library functions abort, exit, getenv and system from library <cstdlib> shall not be used.
M18.0.4	Req	The time handling functions of library <ctime> shall not be used.
M18.0.5	Req	The unbounded functions of library <cstring> shall not be used.
M18.2.1	Req	The macro offsetof shall not be used.
M18.7.1	Req	The signal handling facilities of <csignal> shall not be used.
M19.3.1	Req	The error indicator errno shall not be used.
M27.0.1	Req	The stream input/output library <cstdio> shall not be used.

3. Autosar C++14 Guidelines (AUTOSAR 18.03)

AUTOSAR Rule	Severity	Description
A0.1.1	Req	A project shall not contain instances of non-volatile variables being given values that are never subsequently used.
A0.1.2	Req	The value returned by a function having a non-void return type that is not an overloaded operator shall always be used.
A0.1.3	Req	Every defined function shall be called at least once.
A0.1.4	Req	There shall be no unused named parameters in non-virtual functions.
A0.1.5	Req	There shall be no unused named parameters in the set of parameters for a virtual function and all the functions that override it.
A0.1.6	Adv	A project shall not contain unused type declarations.
A0.4.2	Req	Type long double shall not be used.
A1.1.1	Req	All code shall conform to ISO/IEC 14882:2014 - Programming Language C++ and shall not use deprecated features.

A1.4.1	Req	Code metrics and their valid boundaries shall be defined.
A1.4.3	Adv	All code should compile free of compiler warnings.
A2.3.1	Req	Only those characters specified in the C++ Language Standard basic source character set shall be used in the source code.
A2.5.1	Req	Trigraphs shall not be used.
A2.5.2	Req	Digraphs should not be used.
A2.7.1	Req	The character shall not occur as a last character of a C++ comment.
A2.7.2	Req	Sections of code shall not be "commented out".
A2.7.3	Req	All declarations of "user-defined" types, static and non-static data members, functions and methods shall be preceded by documentation.
A2.8.1	Req	A header file name should reflect the logical entity for which it provides declarations.
A2.8.2	Adv	An implementation file name should reflect the logical entity for which it provides definitions.
A2.10.1	Req	Identifiers declared in an inner scope shall not hide an identifier declared in an outer scope.
A2.10.4	Req	The identifier name of a non-member object or function with static storage duration should not be reused.
A2.10.5	Adv	The identifier name of a non-member object or function with static storage duration should not be reused.
A2.10.6	Req	A class or enumeration name shall not be hidden by a variable, function or enumerator declaration in the same scope.
A2.11.1	Req	Volatile keyword shall not be used.
A2.13.1	Req	Only those escape sequences that are defined in ISO/IEC 14882:2003 shall be used.
A2.13.2	Req	Narrow and wide string literals shall not be concatenated.
A2.13.3	Req	Type wchar_t shall not be used.

A2.13.4	Req	String literals shall not be assigned to non-constant pointers.
A2.13.5	Adv	Hexadecimal constants should be upper case.
A2.13.6	Req	Universal character names shall be used only inside character or string literals.
A3.1.1	Req	It shall be possible to include any header file in multiple translation units without violating the One Definition Rule.
A3.1.2	Req	Header files, that are defined locally in the project, shall have a file name' extension of one of: ".h", ".hpp" or ".hxx".
A3.1.3	Adv	Implementation files, that are defined locally in the project, should have a file name extension of ".cpp".
A3.1.4	Req	When an array is declared, its size shall either be stated explicitly or defined implicitly by initialization.
A3.1.6	Adv	Trivial accessor and mutator functions should be inlined.
A3.3.1	Req	Objects or functions with external linkage shall be declared in a header file.
A3.3.2	Req	Static and thread-local objects shall be constant-initialized.
A3.9.1	Req	Typedefs that indicate size and signedness should be used in place of the basic numerical types.
A4.5.1	Req	Expressions with type enum shall not be used as operands to built-in operators other than the subscript operator [], the assignment operator =, the equality operators == and !=, the unary & operator, and the relational operators <, <=, >, >=.
A4.7.1	Req	An integer expression shall not lead to data loss.
A4.10.1	Req	Only nullptr literal shall be used as the null-pointer-constant.
A5.0.1	Req	The value of an expression shall be the same under any order of evaluation that the standard permits.
A5.0.2	Req	The condition of an if-statement and the condition of an iteration-statement shall have type bool.
A5.0.3	Req	The declaration of objects should contain no more than two levels of pointer indirection.

A5.0.4	Req	Pointer arithmetic shall not be used with pointers to non-final classes.
A5.1.1	Req	Literal values shall not be used apart from type initialization, otherwise symbolic names shall be used instead.
A5.1.2	Req	Variables shall not be implicitly captured in a lambda expression.
A5.1.3	Req	Parameter list (possibly empty) shall be included in every lambda expression.
A5.1.6	Adv	Return type of a non-void return type lambda expression should be explicitly specified.
A5.1.7	Req	The underlying type of lambda expression shall not be used.
A5.1.8	Adv	Lambda expressions should not be defined inside another lambda expression.
A5.1.9	Adv	Identical unnamed lambda expressions shall be replaced with a named function or a named lambda expression.
A5.2.1	Adv	dynamic_cast should not be used.
A5.2.2	Req	Traditional C-style casts shall not be used.
A5.2.3	Req	A cast shall not remove any const or volatile qualification from the type of a pointer or reference.
A5.2.4	Req	reinterpret_cast shall not be used.
A5.2.6	Req	The operands of a logical && or || shall be parenthesized if the operands contain binary operators.
A5.3.1	Req	Evaluation of the operand to the typeid operator shall not contain side effects.
A5.6.1	Req	The right hand operand of the integer division or remainder operators shall not be equal to zero.
A5.10.1	Req	A pointer to member virtual function shall only be tested for equality with null-pointer-constant.
A5.16.1	Req	The ternary conditional operator shall not be used as a sub-expression.

A6.4.1	Req	Every switch statement shall have at least one case-clause.
A6.5.1	Req	A for-loop that loops through all elements of the container and does not use its loop-counter shall not be used.
A6.5.2	Req	A for loop shall contain a single loop-counter which shall not have floating type.
A6.5.3	Adv	Do statements should not be used.
A6.5.4	Adv	For-init-statement and expression should not perform actions other than loop-counter initialization and modification.
A6.6.1	Req	The goto statement shall not be used.
A7.1.1	Req	Constexpr or const specifiers shall be used for immutable data declaration.
A7.1.3	Req	CV-qualifiers shall be placed on the right hand side of the type that is a typedef or a using name.
A7.1.4	Req	The register keyword shall not be used.
A7.1.5	Req	The auto specifier shall not be used apart from following cases: (1) to declare that a variable has the same type as return type of a function call, (2) to declare that a variable has the same type as initializer of non-fundamental type, (3) to declare parameters of a generic lambda expression, (4) to declare a function template using trailing return type syntax.
A7.1.6	Req	The typedef specifier shall not be used.
A7.1.7	Req	Each expression statement and identifier declaration shall be placed on a separate line.
A7.1.9	Req	A class, structure, or enumeration shall not be declared in the definition of its type.
A7.2.1	Req	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration.
A7.2.2	Req	Enumeration underlying base type shall be explicitly defined.
A7.2.3	Req	Enumerations shall be declared as scoped enum classes.

A7.2.4	Req	In an enumerator list, the = construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized.
A7.3.1	Req	All overloads of a function shall be visible from where it is called.
A7.4.1	Req	The asm declaration shall not be used.
A7.5.1	Req	A function shall not return a reference or a pointer to a parameter that is passed by reference or const reference.
A7.5.2	Req	Functions shall not call themselves, either directly or indirectly.
A7.6.1	Req	Functions declared with the [[noreturn]] attribute shall not return.
A8.2.1	Req	When declaring function templates, the trailing return type syntax shall be used if the return type depends on the type of parameters.
A8.4.1	Req	Functions shall not be defined using the ellipsis notation.
A8.4.2	Req	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.
A8.4.4	Adv	Multiple output values from a function should be returned as a struct or tuple.
A8.4.5	Req	"consume" parameters declared as X && shall always be moved from.
A8.4.6	Req	"forward" parameters declared as T && shall always be forwarded.
A8.4.7	Req	"in" parameters for "cheap to copy" types shall be passed by value.
A8.4.8	Req	Output parameters shall not be used.
A8.4.9	Req	"in-out" parameters declared as T & shall be modified.
A8.5.1	Req	In an initialization list, the order of initialization shall be following: (1) virtual base classes in depth and left to right order of the inheritance graph, (2) direct base classes in left to right order of

		inheritance list, (3) non-static data members in the order they were declared in the class definition.
A8.5.2	Req	Braced-initialization {}, without equals sign, shall be used for variable initialization.
A8.5.3	Req	A variable of type auto shall not be initialized using {} or ={} braced-initialization.
A8.5.4	Adv	A constructor taking parameter of type std::initializer_list shall only be defined in classes that internally store a collection of objects.
A9.5.1	Req	Unions shall not be used.
A9.6.1	Req	Bit-fields shall be either unsigned integral, or enumeration (with underlying type of unsigned integral type).
A10.1.1	Req	Class shall not be derived from more than one base class which is not an interface class.
A10.2.1	Req	Non-virtual member functions shall not be redefined in derived classes.
A10.3.1	Req	Virtual function declaration shall contain exactly one of the three specifiers: (1) virtual, (2) override, (3) final.
A10.3.2	Req	Each overriding virtual function shall be declared with the override or final specifier.
A10.3.3	Req	Virtual functions shall not be introduced in a final class.
A10.3.5	Req	A user-defined assignment operator shall not be virtual.
A11.0.1	Adv	A non-POD type should be defined as class.
A11.0.2	Req	A type defined as struct shall: (1) provide only public data members, (2) not provide any special member functions or methods, (3) not be a base of another struct or class, (4) not inherit from another struct or class.
A11.3.1	Req	Friend declarations shall not be used.
A12.0.1	Req	If a class declares a copy or move operation, or a destructor, either via "=default", "=delete", or via a user-provided

		declaration, then all others of these five special member functions shall be declared as well.
A12.1.1	Req	Constructors shall explicitly initialize all virtual base classes, all direct non-virtual base classes and all non-static data members.
A12.1.2	Req	Both NSDMI and a non-static member initializer in a constructor shall not be used in the same type.
A12.1.3	Req	If all user-defined constructors of a class initialize data members with constant values that are the same across all constructors, then data members shall be initialized using NSDMI instead.
A12.1.4	Req	All constructors that are callable with a single argument of fundamental type shall be declared explicit.
A12.1.5	Req	Common class initialization for non-constant members shall be done by a delegating constructor.
A12.1.6	Req	Derived classes that do not need further explicit initialization and require all the constructors from the base class shall use inheriting constructors.
A12.4.1	Req	Destructor of a base class shall be public virtual, public override or protected non-virtual.
A12.4.2	Adv	If a public destructor of a class is non-virtual, then the class should be declared final.
A12.6.1	Req	All class data members that are initialized by the constructor shall be initialized using member initializers.
A12.8.1	Req	A copy constructor shall only initialize its base classes and the non-static members of the class of which it is a member.
A12.8.2	Adv	User-defined copy and move assignment operators should use user-defined no-throw swap function.
A12.8.3	Req	Moved-from object shall not be read-accessed.
A12.8.4	Req	Move constructor shall not initialize its class members and base classes using copy semantics.
A12.8.6	Req	Copy and move constructors and copy assignment and move assignment operators shall be declared protected or defined "=delete" in base class.

A12.8.7	Adv	Assignment operators should be declared with the ref-qualifier &.
A13.1.2	Req	User defined suffixes of the user defined literal operators shall start with underscore followed by one or more letters.
A13.1.3	Req	User defined literals operators shall only perform conversion of passed parameters.
A13.2.1	Req	An assignment operator shall return a reference to "this".
A13.2.2	Req	A binary arithmetic operator and a bitwise operator shall return a "prvalue".
A13.2.3	Req	A relational operator shall return a boolean value.
A13.3.1	Req	A function that contains "forwarding reference" as its argument shall not be overloaded.
A13.5.1	Req	If "operator[]" is to be overloaded with a non-const version, const version shall also be implemented.
A13.5.2	Req	All user-defined conversion operators shall be defined explicit.
A13.5.3	Adv	User-defined conversion operators should not be used.
A13.5.4	Req	If two opposite operators are defined, one shall be defined in terms of the other.
A13.6.1	Req	Digit sequences separators ' shall only be used as follows: (1) for decimal, every 3 digits, (2) for hexadecimal, every 2 digits, (3) for binary, every 4 digits.
A14.7.2	Req	All partial and explicit specializations for a template shall be declared in the same file as the declaration of their primary template.
A14.8.2	Req	Overloaded function templates shall not be explicitly specialized.
A15.1.1	Adv	Only instances of types derived from std::exception shall be thrown.
A15.1.2	Req	An exception object should not have pointer type.
A15.1.3	Adv	All thrown exceptions should be unique.

A15.2.1	Req	Constructors that are not noexcept shall not be invoked before program startup.
A15.3.3	Req	Main function and a task main function shall catch at least: base class exceptions from all third-party libraries used, std::exception and all otherwise unhandled exceptions.
A15.3.4	Req	Catch-all (ellipsis and std::exception) handlers shall be used only in (a) main, (b) task main functions, (c) in functions that are supposed to isolate independent components and (d) when calling third-party code that uses exceptions not according to AUTOSAR C++14 guidelines.
A15.3.5	Req	A class type exception shall always be caught by reference.
A15.4.1	Req	Dynamic exception-specification shall not be used.
A15.4.2	Req	If a function is declared to be noexcept, noexcept(true) or noexcept(<true condition>), then it shall not exit with an exception.
A15.4.3	Req	Function's noexcept specification shall be either identical or more restrictive across all translation units and all overriders.
A15.4.4	Req	A declaration of non-throwing function shall contain noexcept specification.
A15.4.5	Req	Checked exceptions that could be thrown from a function shall be specified together with the function declaration and they shall be identical in all function declarations and for all its overriders.
A15.5.1	Req	All user-provided class destructors, deallocation functions, move constructors, move assignment operators and swap functions shall not exit with an exception. A noexcept exception specification shall be added to these functions as appropriate.
A15.5.2	Req	Program shall not be abruptly terminated. In particular, an implicit or explicit invocation of std::abort(), std::quick_exit(), std::_Exit(), std::terminate() shall not be done.
A15.5.3	Req	The terminate() function shall not be called implicitly.
A16.0.1	Req	The pre-processor shall only be used for file inclusion and include guards.

A16.2.1	Req	The ', ", /*, //, characters shall not occur in a header file name or in #include directive.
A16.6.1	Req	#error directive shall not be used.
A16.7.1	Req	The #pragma directive shall not be used.
A17.0.1	Req	Reserved identifiers, macros and functions in the standard library shall not be defined, redefined or undefined.
A17.0.2	Req	All project's code including used libraries (including standard and user-defined libraries) and any third-party user code shall conform to the AUTOSAR C++14 Coding Guidelines.
A17.6.1	Req	Non-standard entities shall not be added to standard namespaces.
A18.0.1	Req	The C library shall not be used.
A18.0.2	Req	The error state of a conversion from string to a numeric value shall be checked.
A18.0.3	Req	The library <clocale> (locale.h) and the setlocale function shall not be used.
A18.1.1	Adv	C-style arrays should not be used.
A18.1.2	Req	The std::vector<bool> specialization shall not be used.
A18.1.3	Req	The std::auto_ptr type shall not be used.
A18.1.4	Req	A pointer pointing to an element of an array of objects shall not be passed to a smart pointer of single object type.
A18.1.6	Req	All std::hash specializations for user-defined types shall have a noexcept function call operator.
A18.5.1	Req	Functions malloc, calloc, realloc and free shall not be used.
A18.5.2	Req	Operators new and delete shall not be called explicitly.
A18.5.3	Req	The form of delete operator shall match the form of new operator used to allocate the memory.

A18.5.4	Req	If a project has sized or unsized version of operator "delete" globally defined, then both sized and unsized versions shall be defined.
A18.5.8	Req	Objects that do not outlive a function shall have automatic storage duration.
A18.9.1	Req	The std::bind shall not be used.
A18.9.2	Req	Forwarding values to other functions shall be done via: (1) std::move if the value is an rvalue reference, (2) std::forward if the value is forwarding reference.
A18.9.3	Req	The std::move shall not be used on objects declared const or const&.
A21.8.1	Req	Arguments to character-handling functions shall be representable as an unsigned char.
A23.0.1	Req	An iterator shall not be implicitly converted to const_iterator.
A26.5.1	Req	Pseudorandom numbers shall not be generated using std::rand().
A26.5.2	Req	Random number engines shall not be default-initialized.
A27.0.4	Req	C-style strings shall not be used.
M0.1.1	Req	There shall be no unreachable code.
M0.1.2	Req	A project shall not contain infeasible paths.
M0.1.3	Req	A project shall not contain unused variables.
M0.1.4	Req	A project shall not contain non-volatile POD variables having only one use.
M0.1.8	Req	All functions with void return type shall have external side effect(s).
M0.1.9	Req	There shall be no dead code.
M0.1.10	Adv	Every defined function shall be called at least once.
M0.2.1	Req	An object shall not be assigned to an overlapping object.

M0.3.1	Req	Minimization of run-time failures shall be ensured by the use of at least one of: (a) static analysis tools/techniques; (b) dynamic analysis tools/techniques; (c) explicit coding of checks to handle run-time faults.
M0.3.2	Req	If a function generates error information, then that error information shall be tested.
M0.4.2	Req	Use of floating-point arithmetic shall be documented.
M2.7.1	Req	The character sequence /* shall not be used within a C-style comment.
M2.10.1	Req	Different identifiers shall be typographically unambiguous.
M2.13.2	Req	Octal constants (other than zero) and octal escape sequences (other than "0") shall not be used.
M2.13.3	Req	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.
M2.13.4	Req	Literal suffixes shall be upper case.
M3.1.2	Req	Functions shall not be declared at block scope.
M3.2.1	Req	All declarations of an object or function shall have compatible types.
M3.2.2	Req	The One Definition Rule shall not be violated.
M3.2.3	Req	A type, object or function that is used in multiple translation units shall be declared in one and only one file.
M3.2.4	Req	An identifier with external linkage shall have exactly one definition.
M3.3.2	Req	If a function has internal linkage then all re-declarations shall include the static storage class specifier.
M3.4.1	Req	An identifier declared to be an object or type shall be defined in a block that minimizes its visibility.
M3.9.1	Req	The types used for an object, a function return type, or a function parameter shall be token-for-token identical in all declarations and re-declarations.

M3.9.3	Req	The underlying bit representations of floating-point values shall not be used.
M4.5.1	Req	Expressions with type bool shall not be used as operands to built-in operators other than the assignment operator =, the logical operators &&, ||, !, the equality operators == and !=, the unary & operator, and the conditional operator.
M4.5.3	Req	Expressions with type (plain) char and wchar_t shall not be used as operands to built-in operators other than the assignment operator =, the equality operators == and !=, and the unary & operator.
M4.10.1	Req	NULL shall not be used as an integer value.
M4.10.2	Req	Literal zero (0) shall not be used as the null-pointer-constant.
M5.0.2	Adv	Limited dependence should be placed on C++ operator precedence rules in expressions.
M5.0.3	Req	A cvalue expression shall not be implicitly converted to a different underlying type.
M5.0.4	Req	An implicit integral conversion shall not change the signedness of the underlying type.
M5.0.5	Req	There shall be no implicit floating-integral conversions.
M5.0.6	Req	An implicit integral or floating-point conversion shall not reduce the size of the underlying type.
M5.0.7	Req	There shall be no explicit floating-integral conversions of a cvalue expression.
M5.0.8	Req	An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue expression.
M5.0.9	Req	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression.
M5.0.10	Req	If the bitwise operators ~ and << are applied to an operand with an underlying type of unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand.

M5.0.11	Req	The plain char type shall only be used for the storage and use of character values.
M5.0.12	Req	signed char and unsigned char type shall only be used for the storage and use of numeric values.
M5.0.14	Req	The first operand of a conditional-operator shall have type bool.
M5.0.15	Req	Array indexing shall be the only form of pointer arithmetic.
M5.0.16	Req	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array.
M5.0.17	Req	Subtraction between pointers shall only be applied to pointers that address elements of the same array.
M5.0.18	Req	>, >=, <, <= shall not be applied to objects of pointer type, except where they point to the same array.
M5.0.20	Req	Non-constant operands to a binary bitwise operator shall have the same underlying type.
M5.0.21	Req	Bitwise operators shall only be applied to operands of unsigned underlying type.
M5.2.2	Req	A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast.
M5.2.3	Adv	Casts from a base class to a derived class should not be performed on polymorphic types.
M5.2.6	Req	A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type.
M5.2.8	Req	An object with integer type or pointer to void type shall not be converted to an object with pointer type.
M5.2.9	Req	A cast should not convert a pointer type to an integral type.
M5.2.10	Req	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.
M5.2.11	Req	The comma operator, && operator and the || operator shall not be overloaded.

M5.2.12	Req	An identifier with array type passed as a function argument shall not decay to a pointer.
M5.3.1	Req	Each operand of the ! operator, the logical && or the logical || operators shall have type bool.
M5.3.2	Req	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.
M5.3.3	Req	The unary & operator shall not be overloaded.
M5.3.4	Req	Evaluation of the operand to the sizeof operator shall not contain side effects.
M5.8.1	Req	The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.
M5.14.1	Req	The right-hand operand of a logical && or || operator shall not contain side effects.
M5.17.1	Req	The semantic equivalence between a binary operator and its assignment operator form shall be preserved.
M5.18.1	Req	The comma operator shall not be used.
M5.19.1	Req	Evaluation of constant unsigned integer expressions should not lead to wrap-around.
M6.2.1	Req	Assignment operators shall not be used in sub-expressions.
M6.2.2	Req	Floating-point expressions shall not be directly or indirectly tested for equality or inequality.
M6.2.3	Req	Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment, provided that the first character following the null statement is a white-space character.
M6.3.1	Req	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.
M6.4.1	Req	An if (condition) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement.

M6.4.2	Req	All if ... else if constructs shall be terminated with an else clause.
M6.4.3	Req	A switch statement shall be a well-formed switch statement.
M6.4.4	Req	A switch label shall only be used when the most closely-enclosing compound-statement is the body of a switch-statement.
M6.4.5	Req	An unconditional throw or break statement shall terminate every non-empty switch-clause.
M6.4.6	Req	The final clause of a switch statement shall be the default clause.
M6.4.7	Req	The condition of a switch statement shall not have bool type.
M6.5.2	Req	If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to <=, <, > or >=.
M6.5.3	Req	The loop-counter shall not be modified within condition or statement.
M6.5.4	Req	The loop-counter shall be modified by one of: --, ++, -=n, or +=n; where n remains constant for the duration of the loop.
M6.5.5	Req	A loop-control-variable other than the loop-counter shall not be modified within condition or expression.
M6.5.6	Req	A loop-control-variable other than the loop-counter which is modified in statement shall have type bool.
M6.6.1	Req	Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement.
M6.6.2	Req	The goto statement shall jump to a label declared later in the same function body.
M6.6.3	Req	The continue statement shall only be used within a well-formed for loop.
M7.1.2	Req	A pointer or reference parameter in a function shall be declared as pointer to const or reference to const if the corresponding object is not modified.

M7.3.1	Req	The global namespace shall only contain main, namespace declarations and extern "C" declarations.
M7.3.2	Req	The identifier main shall not be used for a function other than the global function main.
M7.3.3	Req	There shall be no unnamed namespaces in header files.
M7.3.4	Req	Using-directives shall not be used.
M7.3.6	Req	using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.
M7.4.1	Req	All usage of assembler shall be documented.
M7.4.2	Req	Assembler instructions shall only be introduced using the asm declaration.
M7.4.3	Req	Assembly language shall be encapsulated and isolated.
M7.5.1	Req	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.
M7.5.2	Req	The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.
M8.0.1	Req	An init-declarator-list or a member-declarator-list shall consist of a single init-declarator or member-declarator respectively.
M8.3.1	Req	Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or else shall not specify any default arguments.
M8.4.2	Req	The identifiers used for the parameters in a re-declaration of a function shall be identical to those in the declaration.
M8.4.4	Req	A function identifier shall either be used to call the function or it shall be preceded by &.
M8.5.2	Req	Braces shall be used to indicate and match the structure in the non-zero initialization of arrays and structures.

M9.3.1	Req	const member functions shall not return non-const pointers or references to class-data.
M9.3.3	Req	If a member function can be made static then it shall be made static, otherwise if it can be made const then it shall be made const.
M9.6.1	Req	When the absolute positioning of bits representing a bit-field is required, then the behaviour and packing of bit-fields shall be documented.
M10.1.1	Adv	Classes should not be derived from virtual bases.
M10.1.2	Req	A base class shall only be declared virtual if it is used in a diamond hierarchy.
M10.1.3	Req	An accessible base class shall not be both virtual and non-virtual in the same hierarchy.
M10.2.1	Adv	All accessible entity names within a multiple inheritance hierarchy should be unique.
M10.3.3	Req	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual.
M11.0.1	Req	Member data in non-POD class types shall be private.
M12.1.1	Req	An object's dynamic type shall not be used from the body of its constructor or destructor.
M14.5.3	Req	A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter.
M14.6.1	Req	In a class template with a dependent base, any name that may be found in that dependent base shall be referred to using a qualified-id or this->.
M15.0.3	Req	Control shall not be transferred into a try or catch block using a goto or a switch statement.
M15.1.1	Req	The assignment-expression of a throw statement shall not itself cause an exception to be thrown.
M15.1.2	Req	NULL shall not be thrown explicitly.

M15.1.3	Req	An empty throw (throw;) shall only be used in the compound-statement of a catch handler.
M15.3.1	Req	Exceptions shall be raised only after start-up and before termination of the program.
M15.3.3	Req	Handlers of a function-try-block implementation of a class constructor or destructor shall not reference non-static members from this class or its bases.
M15.3.4	Req	Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point.
M15.3.6	Req	Where multiple handlers are provided in a single try-catch statement or function-try-block for a derived class and some or all of its bases, the handlers shall be ordered most-derived to base class.
M15.3.7	Req	Where multiple handlers are provided in a single try-catch statement or function-try-block, any ellipsis (catch-all) handler shall occur last.
M16.0.1	Req	#include directives in a file shall only be preceded by other preprocessor directives or comments.
M16.0.2	Req	Macros shall only be #define'd or #undef'd in the global namespace.
M16.0.5	Req	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.
M16.0.6	Req	In the definition of a function-like macro, each instance of a parameter shall be enclosed in parentheses, unless it is used as the operand of # or ##.
M16.0.7	Req	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator.
M16.0.8	Req	If the # token appears as the first token on a line, then it shall be immediately followed by a preprocessing token.
M16.1.1	Req	The defined preprocessor operator shall only be used in one of the two standard forms.

M16.1.2	Req	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related.
M16.2.3	Req	Include guards shall be provided.
M16.3.1	Req	There shall be at most one occurrence of the # or ## operators in a single macro definition.
M16.3.2	Adv	The # and ## operators should not be used.
M17.0.2	Req	The names of standard library macros and objects shall not be reused.
M17.0.3	Req	The names of standard library functions shall not be overridden.
M17.0.5	Req	The setjmp macro and the longjmp function shall not be used.
M18.0.3	Req	The library functions abort, exit, getenv and system from library <cstdlib> shall not be used.
M18.0.4	Req	The time handling functions of library <ctime> shall not be used.
M18.0.5	Req	The unbounded functions of library <cstring> shall not be used.
M18.2.1	Req	The macro offsetof shall not be used.
M18.7.1	Req	The signal handling facilities of <csignal> shall not be used.
M19.3.1	Req	The error indicator errno shall not be used.
M27.0.1	Req	The stream input/output library <cstdio> shall not be used.