	All Rules	Advisory Rules	Document Rules	Mandatory Rules	Recommended Rules	Required Rules
Understand % Coverage	95%	96%	50%	100%	100%	94%
Understand Coverage	1,678	101	1	9	3	643
Total Rules	1,759	105	2	9	3	685

Checks

Check ID	Check Name	Supported	Automation	Category	Severity
A0-1-1	A project shall not contain instances of non-volatile variables being given values that are not subsequently used	Yes	Automated	Required	
A0-1-2	The value returned by a function shall be used	Yes	Automated	Required	
A0-1-3	Every function defined in an anonymous namespace, or static function with internal linkage, or private member	Yes	Automated	Required	

	function shall be used				
A0-1-4	There shall be no unused named parameters in non-virtual functions	Yes	Automated	Required	
A0-1-5	There shall be no unused named parameters in the set of parameters for a virtual function and all the functions that override it	Yes	Automated	Required	
A0-1-6	There should be no unused type declarations	Yes	Automated	Advisory	
A0-4-1	Floating-point implementation shall comply with IEEE 754 standard	No	Non-automated	Required	
A0-4-2	Type long double shall not be used	Yes	Automated	Required	
A0-4-3	The implementations in the chosen compiler shall strictly comply with the C++14	No	Automated	Required	

	Language Standard				
A0-4-4	Range, domain and pole errors shall be checked when using math functions	No	Non-automated	Required	
A1-1-1	All code shall conform to ISO/IEC 14882:2014 - Programming Language C++ and shall not use deprecated features	No	Automated	Required	
A1-1-2	A warning level of the compilation process shall be set in compliance with project policies	No	Non-automated	Required	
A1-1-3	An optimization option that disregards strict standard compliance shall not be turned on in the chosen compiler	No	Non-automated	Required	
A1-2-1	When using a compiler toolchain, in	No	Non-automated	Required	

	safety-related software, the tool confidence level (TCL) shall be determined				
A1-4-1	Code metrics and their valid boundaries shall be defined and code shall comply with defined boundaries of code metrics	No	Non-automated	Required	
A1-4-3	All code should compile free of compiler warnings	Yes	Automated	Advisory	
A2-3-1	Only those characters specified in the C++ Language Standard basic source character set shall be used in the source code	Yes	Automated	Required	
A2-5-1	2-3-1 Trigraphs shall not be used	Yes	Automated	Required	
A2-5-2	Digraphs shall not be	Yes	Automated	Required	

	used				
A2-7-1	The character \ shall not occur as a last character of a C++ comment	Yes	Automated	Required	
A2-7-2	Sections of code shall not be "commented out"	Yes	Non-automated	Required	
A2-7-3	All declarations of "user-defined" types, static and non-static data members, functions and methods shall be preceded by documentation	Yes	Automated	Required	
A2-7-5	Comments shall not document any actions or sources (e.g. tables, figures, paragraphs, etc.) that are outside of the file	No	Non-automated	Required	
A2-8-1	A header file name should reflect the logical entity	Yes	Non-automated	Required	

	for which it provides declarations				
A2-8-2	An implementation file name should reflect the logical entity for which it provides definitions	Yes	Non-automated	Advisory	
A2-10-1	Shadowed Identifiers	Yes	Automated	Required	
A2-10-4	The identifier name of a non-member object with static storage duration or static function shall not be reused within a namespace	Yes	Automated	Required	
A2-10-5	An identifier name of a function with static storage duration or a non-member object with external or internal linkage should not be reused	Yes	Automated	Advisory	
A2-10-6	A class or enumeration name shall not be hidden by a	Yes	Automated	Required	

	variable, function or enumerator declaration in the same scope				
A2-11-1	Volatile keyword shall not be used	Yes	Automated	Required	
A2-13-1	Only those escape sequences that are defined in ISO/IEC 14882:2014 shall be used	Yes	Automated	Required	
A2-13-2	Concatenating String Literals of Different Encodings	Yes			
A2-13-3	Type wchar_t shall not be used	Yes	Automated	Required	
A2-13-4	String literals shall not be assigned to non-constant pointers	Yes	Automated	Required	
A2-13-5	Hexadecimal constants should be upper case	Yes	Automated	Advisory	
A2-13-6	Universal character names shall be used only inside character or string literals	Yes	Automated	Required	
A3-1-1	It shall be	Yes	Automated	Required	

	possible to include any header file in multiple translation units without violating the One Definition Rule				
A3-1-2	Header files, that are defined locally in the project, shall have a file name extension of one of: ".h", ".hpp" or ".hxx"	Yes	Automated	Required	
A3-1-3	Implementation files, that are defined locally in the project, should have a file name extension of ".cpp"	Yes	Automated	Advisory	
A3-1-4	When an array with external linkage is declared, its size shall be stated explicitly	Yes	Automated	Required	
A3-1-5	A function definition shall only be placed in a	Yes	Partially Automated	Required	

	class definition if (1) the function is intended to be inlined (2) it is a member function template (3) it is a member function of a class template				
A3-1-6	Trivial accessor and mutator functions should be inlined.	Yes	Automated	Advisory	
A3-3-1	Objects or functions with external linkage (including members of named namespaces) shall be declared in a header file	Yes	Automated	Required	
A3-3-2	Static and thread-local objects shall be constant-initialized	Yes	Automated	Required	
A3-8-1	An object shall not be accessed outside of its lifetime	No	Automated	Required	

A3-9-1	Fixed Width Integers	Yes	Automated	Required	
A4-5-1	Expressions with type enum or enum class shall not be used as operands to built-in and overloaded operators other than the subscript operator [ ], the assignment operator =, the equality operators == and ! =, the unary & operator, and the relational operators <, <=, >, >=	Yes	Automated	Required	
A4-7-1	An integer expression shall not lead to data loss.	Yes	Automated	Required	
A4-10-1	Only nullptr literal shall be used as the null-pointer-constant	Yes	Automated	Required	
A5-0-1	The value of an expression shall be the same under any order of	Yes	Automated	Required	

	evaluation that the standard permits				
A5-0-2	Condition of if statement shall be bool	Yes	Automated	Required	
A5-0-3	No more than 2 levels of pointer indirection	Yes	Automated	Required	
A5-0-4	Pointer arithmetic shall not be used with pointers to non-final classes	Yes	Automated	Required	
A5-1-1	Literal values shall not be used apart from type initialization, otherwise symbolic names shall be used instead	Yes	Automated	Required	
A5-1-2	Variables shall not be implicitly captured in a lambda expression	Yes	Automated	Required	
A5-1-3	Parameter list (possibly empty) shall be included in every lambda expression	Yes	Automated	Required	
A5-1-4	A lambda	Yes	Automated	Required	

	expression object shall not outlive any of its reference-captured objects				
A5-1-6	Specify Lambda Return Type	Yes	Automated	Advisory	
A5-1-7	A lambda shall not be an operand to decltype or typeid	Yes	Automated	Required	
A5-1-8	Lambda expressions should not be defined inside another lambda expression	Yes	Automated	Advisory	
A5-1-9	Identical unnamed lambda expressions shall be replaced with a named function or a named lambda expression	Yes	Automated	Advisory	
A5-2-1	dynamic_cast should not be used	Yes	Automated	Advisory	
A5-2-2	Traditional C-style casts shall not be used	Yes	Automated	Required	
A5-2-3	A cast shall	Yes	Automated	Required	

	not remove any const or volatile qualification from the type of a pointer or reference				
A5-2-4	reinterpret_cast shall not be used	Yes	Automated	Required	
A5-2-5A	An array or container shall not be accessed beyond its range (Part A)	Yes	Automated	Required	
A5-2-5B	An array or container shall not be accessed beyond its range Part B	Yes	Automated	Required	
A5-2-6	Operands of Logical Boolean Operators	Yes	Automated	Required	
A5-3-1	Evaluation of the operand to the typeid operator shall not contain side effects.	Yes	Non-automated	Required	
A5-3-2	Before dereferencing a pointer, compare it with NULL	Yes	Partially Automated	Required	
A5-3-3	Deleting Pointers to Incomplete Class Types	Yes	Automated	Required	

A5-5-1	A pointer to member shall not access non-existent class members	Yes	Automated	Required	
A5-6-1A	The right hand operand of the integer division or remainder operators shall not be equal to zero	Yes	Automated	Required	
A5-6-1B	The right hand operand of the integer division or remainder operators shall not be equal to zero	Yes	Automated	Required	
A5-10-1	A pointer to member virtual function shall only be tested for equality with null-pointer-constant	Yes	Automated	Required	
A5-16-1	The ternary conditional operator shall not be used as a sub-expression	Yes	Automated	Required	
A6-2-1	Move and copy assignment	Yes	Automated	Required	

	operators shall either move or respectively copy base classes and data members of a class, without any side effects				
A6-2-2	Explicit Calls to Constructors of Temporary Objects	Yes	Automated	Required	
A6-4-1	A switch statement shall have at least two case-clauses, distinct from the default label	Yes	Automated	Required	
A6-5-1	A for-loop that loops through all elements of the container and does not use its loop-counter shall not be used	Yes	Automated	Required	
A6-5-2	A for loop shall contain a single loop-counter which shall not have floating-point type	Yes	Automated	Required	

A6-5-3	Do statements should not be used	Yes	Automated	Advisory	
A6-5-4	For-init-statement and expression should not perform actions other than loop-counter initialization and modification	Yes	Automated	Advisory	
A6-6-1	The goto statement shall not be used.	Yes	Automated	Required	
A7-1-1	Constexpr or const specifiers shall be used for immutable data declaration	Yes	Automated	Required	
A7-1-2	The constexpr specifier shall be used for values that can be determined at compile time	Yes	Automated	Required	
A7-1-3	CV-qualifiers shall be placed on the right hand side of the	Yes	Automated	Required	

	type that is a typedef or a using name				
A7-1-4	The register keyword shall not be used	Yes	Automated	Required	
A7-1-5	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	Yes	Automated	Required	
A7-1-6	The typedef specifier shall not be	Yes	Automated	Required	

	used				
A7-1-7	Each expression statement and identifier declaration shall be placed on a separate line	Yes	Automated	Required	
A7-1-8	A non-type specifier shall be placed before a type specifier in a declaration.	Yes	Automated	Required	
A7-1-9	A class, structure, or enumeration shall not be declared in the definition of its type	Yes	Automated	Required	
A7-2-1	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration	Yes	Automated	Required	
A7-2-2	Enumeration underlying base type shall be explicitly defined	Yes	Automated	Required	
A7-2-3	Enumerations	Yes	Automated	Required	

	shall be declared as scoped enum classes				
A7-2-4	In an enumeration, either (1) none, (2) the first or (3) all enumerators shall be initialized	Yes	Automated	Required	
A7-2-5	Enumerations should be used to represent sets of related named constants	No	Non-automated	Advisory	
A7-3-1	Overloaded Function Not Visible From Where it is Called	Yes	Automated	Required	
A7-4-1	The asm declaration shall not be used.	Yes	Automated	Required	
A7-5-1	A function shall not return a reference or a pointer to a parameter that is passed by reference to const.	Yes	Automated	Required	
A7-5-2	Functions shall not call themselves,	Yes	Automated	Required	

	either directly or indirectly.				
A7-6-1	Functions declared with the [[noreturn]] attribute shall not return	Yes	Automated	Required	
A8-2-1	When declaring function templates, the trailing return type syntax shall be used if the return type depends on the type of parameters.	Yes	Automated	Required	
A8-4-1	Functions shall not be defined using the ellipsis notation.	Yes	Automated	Required	
A8-4-2	Always return a value in non-void functions	Yes	Automated	Required	
A8-4-3	Common ways of passing parameters should be used.	Yes	Automated	Required	
A8-4-4	Multiple output values from a function should be returned as a	Yes	Automated	Advisory	

	struct or tuple.				
A8-4-5	"consume" parameters declared as X && shall always be moved from.	Yes	Automated	Required	
A8-4-6	"forward" parameters declared as T && shall always be forwarded.	Yes	Automated	Required	
A8-4-7	"in" parameters for "cheap to copy" types shall be passed by value.	Yes	Automated	Required	
A8-4-8	Output parameters shall not be used.	Yes	Automated	Required	
A8-4-9	"in-out" parameters declared as T & shall be modified.	Yes	Automated	Required	
A8-4-10	A parameter shall be passed by reference if it can't be NULL	Yes	Automated	Required	
A8-4-11	A smart pointer shall only be used as a parameter type if it	Yes	Automated	Required	

	expresses lifetime semantics				
A8-4-12	Invalid Use of std::unique_ptr	Yes	Automated	Required	
A8-4-13	Invalid Use of std::shared_ptr	Yes	Automated	Required	
A8-4-14	Interfaces shall be precisely and strongly typed	No	Non-automated	Required	
A8-5-0	Uninitialized Memory Read	Yes	Automated	Required	
A8-5-1	Incorrect Order of Initialization	Yes	Automated	Required	
A8-5-2	Initializing Variables Without Using Braced-Initialization	Yes	Automated	Required	
A8-5-3	Auto Variable	Yes	Automated	Required	
A8-5-4	Class Constructor with Parameter Type std::initializer_list	Yes	Automated	Advisory	
A9-3-1	Member functions shall not return non-const raw pointers or references to private or	Yes	Automated	Required	

	protected data owned by the class				
A9-5-1	Unions Shall not be Used	Yes	Automated	Required	
A9-6-1	Data types used for interfacing	Yes	Partially Automated	Required	
A9-6-2	Bit-fields shall be used only when interfacing to hardware or conforming to communication protocols	No	Non-automated	Required	
A10-0-1	Public Inheritance not Used in a "is-a" Relationship	Yes	Non-automated	Required	
A10-0-2	Membership or non-public inheritance shall be used to implement "has-a" relationship	No	Non-automated	Required	
A10-1-1	Multiple Base Classes	Yes	Automated	Required	
A10-2-1	Non-virtual public or protected member functions shall not be redefined in derived classes	Yes	Automated	Required	
A10-3-1	Virtual function	Yes	Automated	Required	

	declaration shall contain exactly one of the three specifiers: (1) virtual, (2) override, (3) final				
A10-3-2	Use Override	Yes	Automated	Required	
A10-3-3	Virtual functions shall not be introduced in a final class	Yes	Automated	Required	
A10-3-5	User-defined assignment operator shall not be virtual	Yes	Automated	Required	
A10-4-1	Hierarchies should be based on interface classes	Yes	Non-automated	Advisory	
A11-0-1	A non-POD type should be defined as class	Yes	Automated	Advisory	
A11-0-2	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	Yes	Automated	Required	

	class, (4) not inherit from another struct or class				
A11-3-1	Friend declarations shall not be used.	Yes	Automated	Required	
A12-0-1	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.	Yes	Automated	Required	
A12-0-2	Bitwise operations and operations that assume data representation in memory shall not be performed on objects.	Yes	Automated	Required	
A12-1-1	Constructors shall	Yes	Automated	Required	

	explicitly initialize all virtual base classes, all direct non-virtual base classes and all non-static data members.				
A12-1-2	Both NSDMI and a non-static member initializer in a constructor shall not be used in the same type.	Yes	Automated	Required	
A12-1-3	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.	Yes	Automated	Required	
A12-1-4	All constructors that are callable with a single argument of	Yes	Automated	Required	

	fundamental type shall be declared explicit.				
A12-1-5	Common class initialization for non-constant members shall be done by a delegating constructor.	Yes	Partially Automated	Required	
A12-1-6	Derived classes that do not need further explicit initialization and require all the constructors from the base class shall use inheriting constructors	No	Automated	Required	
A12-4-1	Destructor of a base class shall be public virtual, public override or protected non-virtual	Yes	Automated	Required	
A12-4-2	If a public destructor of a class is non-virtual, then the class should	Yes	Automated	Advisory	

	be declared final.				
A12-6-1	All class data members that are initialized by the constructor shall be initialized using member initializers.	Yes	Automated	Required	
A12-7-1	If the behavior of a user-defined special member function is identical to implicitly defined special member function, then it shall be defined =default or be left undefined.	Yes	Automated	Required	
A12-8-1	Move and copy constructors shall move and respectively copy base classes and data members of a class, without any	Yes	Automated	Required	

	side effects				
A12-8-2	User-defined copy and move assignment operators should use user-defined no-throw swap function.	Yes	Automated	Advisory	
A12-8-3	Moved-from object shall not be read-accessed.	Yes	Partially Automated	Advisory	
A12-8-4	Move constructor shall not initialize its class members and base classes using copy semantics.	Yes	Automated	Required	
A12-8-5	A copy assignment and a move assignment operators shall handle self-assignment.	Yes	Automated	Required	
A12-8-6	Copy and move constructors and copy assignment and move assignment operators shall be declared	Yes	Automated	Required	

	protected or defined "=delete" in base class.				
A12-8-7	Assignment operators should be declared with the ref-qualifier &.	Yes	Automated	Advisory	
A13-1-2	User defined suffixes of the user defined literal operators shall start with underscore followed by one or more letters	Yes	Automated	Required	
A13-1-3	User defined literals operators shall only perform conversion of passed parameters	Yes	Automated	Required	
A13-2-1	An assignment operator shall return a reference to "this"	Yes	Automated	Required	
A13-2-2	A binary arithmetic operator and a bitwise operator shall return a "prvalue"	Yes	Automated	Required	

A13-2-3	A relational operator shall return a boolean value	Yes	Automated	Required	
A13-3-1	A function that contains "forwarding reference" as its argument shall not be overloaded	Yes	Automated	Required	
A13-5-1	If "operator[]" is to be overloaded with a non-const version, const version shall also be implemented	Yes	Automated	Required	
A13-5-2	All user-defined conversion operators shall be defined explicit	Yes	Automated	Required	
A13-5-3	User-defined conversion operators should not be used	Yes	Automated	Advisory	
A13-5-4	If two opposite operators are defined, one shall be defined in terms of the other	Yes	Automated	Required	

A13-5-5	Comparison operators shall be non-member functions with identical parameter types and noexcept	Yes	Automated	Required	
A13-6-1	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	Yes	Automated	Required	
A14-1-1	A template should check if a specific template argument is suitable for this template	Yes	Non-automated	Advisory	
A14-5-1	A template constructor shall not participate in overload resolution for a single argument of the enclosing class type	Yes	Automated	Required	
A14-5-2	Class members	Yes	Partially Automated	Advisory	

	that are not dependent on template class parameters should be defined in a separate base class				
A14-5-3	A non-member generic operator shall only be declared in a namespace that does not contain class (struct) type, enum type or union type declarations	Yes	Automated	Advisory	
A14-7-1	A type used as a template argument shall provide all members that are used by the template	Yes	Automated	Required	
A14-7-2	Template specialization shall be declared in the same file as the primary template	Yes	Automated	Required	
A14-8-2	Explicit specializations of function templates	Yes	Automated	Required	

	shall not be used				
A15-0-1	A function shall not exit with an exception if it is able to complete its task	No	Non-automated	Required	
A15-0-2	At least the basic guarantee for exception safety shall be provided for all operations. In addition, each function may offer either the strong guarantee or the nothrow guarantee	No	Partially Automated	Required	
A15-0-3	Exception safety guarantee of a called function shall be considered	No	Non-automated	Required	
A15-0-4	Unchecked exceptions shall be used to represent errors from which the caller cannot reasonably be expected to recover.	No	Non-automated	Required	

A15-0-5	Checked exceptions shall be used to represent errors from which the caller can reasonably be expected to recover	No	Non-automated	Required	
A15-0-6	An analysis shall be performed to analyze the failure modes of exception handling	No	Non-automated	Required	
A15-0-7	Exception handling mechanism shall guarantee a deterministic worst-case time execution time	No	Partially Automated	Required	
A15-0-8	A worst-case execution time (WCET) analysis shall be performed to determine maximum execution time constraints of the software, covering in particular the exceptions	No	Non-automated	Required	

	processing				
A15-1-1	Only instances of types derived from std::exception should be thrown	Yes	Automated	Advisory	
A15-1-2	An exception object shall not be a pointer	Yes	Automated	Required	
A15-1-3	All thrown exceptions should be unique	Yes	Automated	Advisory	
A15-1-4	If a function exits with an exception, then before a throw, the function shall place all objects/ resources that the function constructed in valid states or it shall delete them	Yes	Partially Automated	Required	
A15-1-5	Exceptions thrown across execution boundaries	Yes	Non-automated	Required	
A15-2-1	Constructors that are not noexcept shall not be invoked before	Yes	Automated	Required	

	program startup				
A15-2-2	If a constructor is not noexcept and the constructor cannot finish object initialization, then it shall deallocate the object's resources and it shall throw an exception	Yes	Partially Automated	Required	
A15-3-2	If a function throws an exception, it shall be handled when meaningful actions can be taken, otherwise it shall be propagated	No	Non-automated	Required	
A15-3-3	Unhandled Exceptions on Main Function	Yes	Partially Automated	Required	
A15-3-4	Catch-all (ellipsis and std::exception) handlers shall be used only in (a) main, (b) task main	Yes	Non-automated	Required	

	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	A class type exception shall be caught by reference or const reference	Yes	Automated	Required	
A15-4-1	Dynamic exception-specification shall not be used	Yes	Automated	Required	
A15-4-2	If a function is declared to be noexcept, noexcept(true) or noexcept(<truecondition>), then it shall not exit with an exception	Yes	Automated	Required	
A15-4-3	The noexcept specification of a function	Yes	Automated	Required	

	shall either be identical across all translation units, or identical or more restrictive between a virtual member function and an overrider				
A15-4-4	A declaration of non-throwing function shall contain noexcept specification	Yes	Automated	Required	
A15-4-5	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.	Yes	Automated	Required	
A15-5-1	All user-provided class destructors, deallocation	Yes	Automated	Required	

	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	Program shall not be abruptly terminated	Yes	Automated	Required	
A15-5-3	The std::terminate() function shall not be called implicitly	Yes	Automated	Required	
A16-0-1	Incorrect Use of Pre-processor	Yes	Automated	Required	
A16-2-1	Header File Name	Yes	Automated	Required	
A16-2-2	There shall be no unused include directives (slow)	Yes	Automated	Required	
A16-2-3	An include directive shall be	Yes	Non-automated	Required	

	added explicitly for every symbol used in a file				
A16-6-1	#error directive shall not be used	Yes	Automated	Required	
A16-7-1	The #pragma directive shall not be used	Yes	Automated	Required	
A17-0-1	Reserved Builtin Macros	Yes	Automated	Required	
A17-0-2	All project's code including used libraries and any third-party user code shall conform to the AUTOSAR C++14 Coding Guidelines	No	Non-automated	Required	
A17-1-1	Use of the C Standard Library shall be encapsulated and isolated	No	Non-automated	Required	
A17-6-1	Non-standard entities shall not be added to standard namespaces	Yes	Automated	Required	
A18-0-1	The C library facilities shall only be	Yes	Automated	Required	

	accessed through C++ library headers				
A18-0-2	The error state of a conversion from string to a numeric value shall be checked	Yes	Automated	Required	
A18-0-3	Library <clocale> (locale.h)	Yes	Automated	Required	
A18-1-1	C-style Array	Yes	Automated	Required	
A18-1-2	The std::vector<bool> specialization shall not be used	Yes	Automated	Required	
A18-1-3	The std::auto_ptr type shall not be used	Yes	Automated	Required	
A18-1-4	A pointer pointing to an element of an array of objects shall not be passed to a smart pointer of single object type	Yes	Automated	Required	
A18-1-6	All std::hash specializations for user-defined types shall have a noexcept function call	Yes	Automated	Required	

	operator				
A18-5-1	Functions malloc, calloc, realloc and free shall not be used	Yes	Automated	Required	
A18-5-2	Non-placement new or delete expressions shall not be used	Yes	Partially Automated	Required	
A18-5-3	The form of the delete expression shall match the form of the new expression used to allocate the memory	Yes	Automated	Required	
A18-5-4	If a project has a sized or unsized version of operator "delete" globally defined, then both sized and unsized versions shall be defined	Yes	Automated	Required	
A18-5-5	Memory management functions shall ensure the following	No	Partially Automated	Required	
A18-5-6	An analysis shall be performed to	No	Non-automated	Required	

	analyze the failure modes of dynamic memory management				
A18-5-7	Dynamic Memory Usage on Realtime Phase	Yes	Non-automated	Required	
A18-5-8	Objects that do not outlive a function shall have automatic storage duration	Yes	Partially Automated	Required	
A18-5-9	New Method Throwing an Exception	Yes	Automated	Required	
A18-5-10	Placement new shall be used only with properly aligned pointers to sufficient storage capacity	No	Automated	Required	
A18-5-11	operator "new" and operator "delete" shall be defined together	Yes	Automated	Required	
A18-9-1	The std::bind shall not be used	Yes	Automated	Required	
A18-9-2	Forwarding values to other functions	Yes	Automated	Required	

	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	The std::move shall not be used on objects declared const or const&	Yes	Automated	Required	
A18-9-4	An argument to std::forward shall not be subsequently used	Yes	Automated	Required	
A20-8-1	An already-owned pointer value shall not be stored in an unrelated smart pointer	Yes	Automated	Required	
A20-8-2	A std::unique_ptr shall be used to represent exclusive ownership	Yes	Automated	Required	
A20-8-3	A std::shared_ptr shall be used to	Yes	Automated	Required	

	represent shared ownership				
A20-8-4	A std::unique_ptr shall be used over std::shared_ptr if ownership sharing is not required	Yes	Automated	Required	
A20-8-5	std::make_unique shall be used to construct objects owned by std::unique_ptr	Yes	Automated	Required	
A20-8-6	std::make_shared shall be used to construct objects owned by std::shared_ptr	Yes	Automated	Required	
A20-8-7	Cyclic Structure of std::shared_ptr	Yes	Non-automated	Required	
A21-8-1	Arguments to character-handling functions shall be representable as an unsigned char	Yes	Automated	Required	
A23-0-1	An iterator	Yes	Automated	Required	

	shall not be implicitly converted to const_iterator				
A23-0-2	Elements of a container shall only be accessed via valid references, iterators, and pointers	No	Automated	Required	
A25-1-1	Predicate Function Objects Copied Incorrectly	Yes	Automated	Required	
A25-4-1	Ordering predicates used with associative containers and STL sorting and related algorithms shall adhere to a strict weak ordering relation	No	Non-automated	Required	
A26-5-1	Pseudorandom numbers shall not be generated using std::rand()	Yes	Automated	Required	
A26-5-2	Random number engines shall not be	Yes	Automated	Required	

	default-initialized				
A27-0-1	Inputs from independent components shall be validated	Yes	Non-automated	Required	
A27-0-2	A C-style string shall guarantee sufficient space for data and the null terminator	No	Automated	Advisory	
A27-0-3	Alternate input and output operations on a file stream shall not be used without an intervening flush or positioning call	Yes	Automated	Required	
A27-0-4	C-style strings shall not be used	Yes	Automated	Required	
AC_00	No Control Code Characters	Yes			
AC_01	No Direct or Indirect Recursion	Yes			
AC_HIS_02	Number of Paths(PATH)	Yes			
AC_HIS_04	Cyclomatic Complexity (v(G))	Yes			
AC_HIS_05	Calling	Yes			

	Functions (CALLING)				
AC_HIS_06	Called Functions (CALLS)	Yes			
AC_HIS_07	Function Parameters (PARAM)	Yes			
AC_HIS_08	Number of Staments (STMT)	Yes			
AC_HIS_09	Number of call levels (LEVEL)	Yes			
AC_HIS_10	Number of return points (RETURN)	Yes			
AC_HIS_11	Language scope (VOCF)	Yes			
AC_HIS_12	Recursion (AP_CG_CYCLE)	Yes			
AC_HIS_13	Statements Changed (SCHG)	Yes			
AC_HIS_14	Statements Deleted (SDEL)	Yes			
AC_HIS_15	New Statements (SNEW)	Yes			
AC_HIS_16	Stability Index (S)	Yes			
ARR30-C	Do not form or use out-of-bounds pointers or array subscripts	No			High
ARR32-C	Ensure size	No			High

	arguments for variable length arrays are in a valid range				
ARR38-C	Guarantee that library functions do not form invalid pointers	No			High
CON32-C	Prevent data races when accessing bit-fields from multiple threads	No			Medium
CON34-C	Declare objects shared between threads with appropriate storage durations	No			Medium
CON35-C	Avoid deadlock by locking in a predefined order	No			Low
CON43-C	Do not allow data races in multithreaded code	No			Medium
CON50-CPP	Do not destroy a mutex while it is locked	Yes			Medium
CON51-CPP	Ensure actively held locks are released on	Yes			Low

	exceptional conditions				
CON52-CPP	Prevent data races when accessing bit-fields from multiple threads	Yes			Medium
CON53-CPP	Avoid deadlock by locking in a predefined order	No			Low
CON54-CPP	Wrap functions that can spuriously wake up in a loop	Yes			Medium
CON55-CPP	Preserve thread safety and liveness when using condition variables	Yes			Low
CON56-CPP	Do not speculatively lock a non-recursive mutex that is already owned by the calling thread	Yes			Low
CPP_A000	Assembler instructions only use asm keyword	Yes			
CPP_A001	Assembly language shall be encapsulated and isolated.	Yes			

CPP_A002	Assignment Operator Return This	Yes			
CPP_A003	Assignment Operator Self Assignment	Yes			
CPP_A004	Parameter of assignment operator is a constant reference	Yes			
CPP_A005	Move and copy assignment operators shall either move or respectively copy base classes and data members of a class, without any side effects	Yes			
CPP_A006	The asm declaration shall not be used.	Yes			
CPP_A007	Assembler instructions shall only be introduced using the asm declaration.	Yes			
CPP_A008	Assembly Language Code Usage not Documented	Yes			
CPP_A009	User-defined	Yes			

	copy and move assignment operators should use user-defined no-throw swap function.				
CPP_A010	Move constructor shall not initialize its class members and base classes using copy semantics.	Yes			
CPP_A011	A copy assignment and a move assignment operators shall handle self-assignment.	Yes			
CPP_A012	Copy and move constructors and copy assignment and move assignment operators shall be declared protected or defined "=delete" in base class.	Yes			
CPP_A013	Assignment operators	Yes			

	should be declared with the ref-qualifier &.				
CPP_A014	The semantic equivalence between a binary operator and its assignment operator form shall be preserved	Yes			
CPP_A015	An assignment operator shall return a reference to "this"	Yes			
CPP_A016	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->	Yes			
CPP_A017	A template should check if a specific template argument is suitable for this template	Yes			
CPP_AO000	Assignment operators	Yes			

	shall not be used in sub-expressions				
CPP_B000	Bool, Unsigned, or Signed Bit-fields	Yes			
CPP_B001	(Fuzzy parser) Bit-fields shall only be declared with an appropriate type	Yes			
CPP_B002	Enum Bit-fields	Yes			
CPP_B003	The underlying bit representations of floating-point values shall not be used	Yes			
CPP_B004	(Fuzzy parser) Named bit-fields with signed integer type shall have a length of more than one bit.	Yes			
CPP_B005	(Fuzzy parser) Single-bit named bit fields shall not be of a signed type	Yes			

CPP_B006	Bit-field Length	Yes			
CPP_C000	Commented Out Code	Yes			
CPP_C001	Line-splicing shall not be used in // comments	Yes			
CPP_C002	No Nested Comments	Yes			
CPP_C003	Only use /* comments	Yes			
CPP_C004	Parameter of copy constructor is a constant reference	Yes			
CPP_C005	Members in function-try-blocks in constructors or destructors	Yes			
CPP_C006	Explicitly call all immediate and virtual base classes	Yes			
CPP_C007	A copy constructor shall be declared when there is a template constructor with a single parameter that is a generic parameter	Yes			
CPP_C008	A copy constructor shall only	Yes			

	initialize its base classes and the non-static members of the class of which it is a member				
CPP_C009	Explicit Constructors	Yes			
CPP_C010	Incomplete constructor initialization list	Yes			
CPP_C011	An object's dynamic type shall not be used from the body of its constructor or destructor	Yes			
CPP_C012	Virtual Function Call In Constructor	Yes			
CPP_C013	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement	Yes			
CPP_C014	Dangling Else	Yes			
CPP_C015	A for loop shall contain a single loop-counter which shall	Yes			

	not have floating-point type				
CPP_C016	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	Yes			
CPP_C017	The body of an iteration-statement or a selection-statement shall be a compound-statement	Yes			
CPP_C018	Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement	Yes			
CPP_C019	A loop-control-variable other than	Yes			

	the loop-counter shall not be modified within condition or expression				
CPP_C020	If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to <=, <, > or >=	Yes			
CPP_C021	The loop-counter shall be modified by one of: --, ++, -= n, or += n; where n remains constant for the duration of the loop	Yes			
CPP_C022	The loop-counter shall not be modified within condition or statement	Yes			
CPP_C023	The goto statement shall jump to a label declared later in the same	Yes			

	function body				
CPP_C024	No Continue Statements	Yes			
CPP_C025	Goto Statements	Yes			
CPP_C026	There should be no more than one break or goto statement used to terminate any iteration statement	Yes			
CPP_C027	Member data in non-POD class types shall be private	Yes			
CPP_C028	A null statement shall only occur on a line by itself	Yes			
CPP_C029	Single exit point at end	Yes			
CPP_C030	A switch-label shall only be used when the most closely-enclosing compound statement is the body of a switch statement	Yes			
CPP_C031	Switch Has Default	Yes			
CPP_C032	Every switch statement	Yes			

	shall have at least two switch-clauses				
CPP_C033	An unconditional throw or break statement shall terminate every non-empty switch-clause	Yes			
CPP_C034	Unreachable Code	Yes			
CPP_C035	No Backslash at End of Comment	Yes			
CPP_C036	If statements shall not have assignments in the conditions	Yes			
CPP_C037	Documentation	Yes			
CPP_C038	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	Yes			

	null statement is a white-space character				
CPP_C039	A switch statement shall have at least two case-clauses, distinct from the default label	Yes			
CPP_C040	A loop-control-variable other than the loop-counter which is modified in statement shall have type bool	Yes			
CPP_C041	Do statements should not be used	Yes			
CPP_C042	For-init-statement and expression should not perform actions other than loop-counter initialization and modification	Yes			
CPP_C043	Checked	Yes			

	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.				
CPP_C044	Continue Statement Used in a not Well-formed For Loop	Yes			
CPP_C045	Write constructor member initializers in the canonical order	Yes			
CPP_C046	Switch Statement not Well-formed	Yes			
CPP_C047	All if and else if constructs shall be terminated with an else clause	Yes			
CPP_C048	Transferring Control to a Try or Catch Block Using Goto or	Yes	Non-automated	Required	

	Switch Statement				
CPP_C049	Class Constructor with Parameter Type std::initializer_list	Yes			
CPP_C050	A for-loop that loops through all elements of the container and does not use its loop-counter shall not be used	Yes			
CPP_C051	Constructors that are not noexcept shall not be invoked before program startup	Yes			
CPP_C052	If a constructor is not noexcept and the constructor cannot finish object initialization, then it shall deallocate the object's resources and it shall throw an exception	Yes			

CPP_C053	Explicit Calls to Constructors of Temporary Objects	Yes			
CPP_C054	When a "deep copy" constructor is not implemented, comments in the class header shall describe this fact	Yes			
CPP_C055	Constructors that can be used with one argument should be declared explicit.	Yes			
CPP_C056	Move and copy constructors shall move and respectively copy base classes and data members of a class, without any side effects	Yes			
CPP_CF000	The condition of a switch statement shall not have bool type	Yes			

CPP_CF001	All cases in a switch statement shall have a break or it shall be well commented	Yes			
CPP_CF002	Switch statements should have a default case	Yes			
CPP_CF003	Switch label unstructured	Yes			
CPP_CF004	The std::terminate() function shall not be called implicitly	Yes			
CPP_CF005	Program shall not be abruptly terminated	Yes			
CPP_CF006	Simple Control Flow	Yes			
CPP_CF007	Loops with Fixed Limits	Yes			
CPP_CM000	Comments shall precede code being commented and shall align with code they represent	Yes			
CPP_CM001	Each function shall end with a comment	Yes			
CPP_CM002	Timing delays shall be preceded	Yes			

	by comments explaining the delay				
CPP_CM003	Class headers shall include a short description for every member function declaration and a comment for every data member declared	Yes			
CPP_CT_BUGPRONE_ASSERT_SIDE_EFFECT	Assert Side Effect	Yes			High
CPP_CT_BUGPRONE_BRANCH_CLONE	Branch Clone	Yes			High
CPP_CT_BUGPRONE_COPY_CONSTRUCTOR_INIT	Copy Constructor Init	Yes			High
CPP_CT_BUGPRONE_INFINITE_LOOP	Infinte Loop	Yes			High
CPP_CT_BUGPRONE_MACRO_REPEATED_SIDE_EFFECTS	Macro Side Effects	Yes			High
CPP_CT_BUGPRONE_NOT_NULL_TERMINATED_RESULT	Missing Null Terminator	Yes			High
CPP_CT_BUG	Redundant	Yes			High

PRONE_REDUNDANT_BRANCH_CONDITION	Condition				
CPP_CT_MODERNIZE_USE_DEFAULT_MEMBER_INIT	Default Member Init	Yes			
CPP_CT_MODERNIZE_USE_EQUALS_DEFAULT	Default Member Function	Yes			
CPP_CT_MODERNIZE_USE_EQUALS_DELETE	Delete Member Function	Yes			
CPP_CT_MODERNIZE_USE_NULLPTR	Null Pointer Keyword	Yes			
CPP_CT_READABILITY_DELETE_NULL_POINTER	Delete Null Pointer	Yes			High
CPP_CT_READABILITY_REDUNDANT_CASTING	Redundant Cast	Yes			High
CPP_D000	An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes			
CPP_D001	Do not delete a polymorphic object without a	Yes			

	virtual destructor				
CPP_D002	Single Declarations	Yes			
CPP_D003	When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialisation	Yes			
CPP_D004	A u or U suffix shall be applied to all integer constants that are represented in an unsigned type	Yes			
CPP_D005	A base class shall only be declared virtual if it is used in a diamond hierarchy	Yes			
CPP_D006	Class Derived From Virtual Bases	Yes			
CPP_D007	A compatible declaration shall be visible when an object or function with external	Yes			

	linkage is defined				
CPP_D008	A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter	Yes			
CPP_D009	Multiple declarations for an identifier in the same namespace shall not straddle a using-declaration for that identifier	Yes			
CPP_D010	= construct in enumerator list shall only be used on either the first item alone, or all items explicitly.	Yes			
CPP_D011	Use the static keyword for internal linkage	Yes			
CPP_D012	An external	Yes			

	object or function shall be declared in one and only one file				
CPP_D013	An identifier with external linkage shall have exactly one definition	Yes			
CPP_D015	Externals shall have the same type in the declaration and definition	Yes			
CPP_D017	Non-static Inline Functions	Yes			
CPP_D018	Literal suffixes shall be upper case	Yes			
CPP_D019	The comma operator, && operator and the || operator shall not be overloaded	Yes			
CPP_D020	The lowercase character L shall not be used in a literal suffix	Yes			
CPP_D021	Narrow and wide string literals shall not be concatenated	Yes			
CPP_D022	Functions	Yes			

	and objects should not be defined with external linkage if they are referenced in only one translation unit				
CPP_D023	Single-Function Global Objects	Yes			
CPP_D024	The restrict type qualifier shall not be used	Yes			
CPP_D026	The register keyword shall not be used	Yes			
CPP_D027	The unary & operator shall not be overloaded	Yes			
CPP_D028	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique	Yes			
CPP_D029	Destructor Set Data Ptr to 0	Yes			
CPP_D030	Exceptions in Destructors	Yes			
CPP_D031	Non-Virtual Destructors	Yes			

	in Base Classes				
CPP_D032	Virtual Function Call In Destructor	Yes			
CPP_D033	A function shall not be declared implicitly	Yes			
CPP_D034	Datamembers should be declared private	Yes			
CPP_D035	Destructor of a base class shall be public virtual, public override or protected non-virtual	Yes			
CPP_D036	Volatile keyword shall not be used	Yes			
CPP_D037	Functions shall not be declared at block scope	Yes			
CPP_D038	When an array with external linkage is declared, its size shall be stated explicitly	Yes			
CPP_D039	A function definition shall only be placed in a class definition if	Yes			

	(1) the function is intended to be inlined (2) it is a member function template (3) it is a member function of a class template				
CPP_D040	All declarations of an object or function shall have compatible types	Yes			
CPP_D041	The One Definition Rule	Yes			
CPP_D042	If a function has internal linkage then all redeclarations shall include the static storage class specifier	Yes			
CPP_D043	Static and thread-local objects shall be constant-initialized	Yes			
CPP_D044	Declarations at Lowest Scope	Yes			
CPP_D045	A type,	Yes			

	object or function that is used in multiple translation units shall be declared in one and only one file				
CPP_D046	Constexpr or const specifiers shall be used for immutable data declaration	Yes			
CPP_D047	The constexpr specifier shall be used for values that can be determined at compile time	Yes			
CPP_D048	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	Yes			

	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				
CPP_D049	A class, structure, or enumeration shall not be declared in the definition of its type	Yes			
CPP_D050	Enumerations shall be declared as scoped enum classes	Yes			
CPP_D051	A non-type specifier shall be placed before a type specifier in a declaration.	Yes			
CPP_D052	Use the same identifier in definition and declaration of functions.	Yes			
CPP_D053	Multiple Base	Yes			

	Classes				
CPP_D054	Virtual function declaration shall contain exactly one of the three specifiers: (1) virtual, (2) override, (3) final	Yes			
CPP_D055	All Checks/ Language Specific/C and C++/ Destructors/ Non-Virtual Destructors in Base Classes	Yes			
CPP_D056	User-defined assignment operator shall not be virtual	Yes			
CPP_D057	Hierarchies should be based on interface classes	Yes			
CPP_D058	A non-POD type should be defined as class	Yes			
CPP_D059	Friend declarations shall not be used.	Yes			
CPP_D060	If a class declares a copy or move operation, or a destructor,	Yes			

	either via "=default", "=delete", or via a user-provided declaration, then all others of these five special member functions shall be declared as well.				
CPP_D061	Constructors shall explicitly initialize all virtual base classes, all direct non-virtual base classes and all non-static data members.	Yes			
CPP_D062	Both NSDMI and a non-static member initializer in a constructor shall not be used in the same type.	Yes			
CPP_D063	If all user-defined constructors of a class initialize data members	Yes			

	with constant values that are the same across all constructors, then data members shall be initialized using NSDMI instead.				
CPP_D064	All constructors that are callable with a single argument of fundamental type shall be declared explicit.	Yes			
CPP_D065	Common class initialization for non-constant members shall be done by a delegating constructor.	Yes			
CPP_D066	If a public destructor of a class is non-virtual, then the class should be declared final.	Yes			
CPP_D067	All class data members that are	Yes			

	initialized by the constructor shall be initialized using member initializers.				
CPP_D068	If the behavior of a user-defined special member function is identical to implicitly defined special member function, then it shall be defined =default or be left undefined.	Yes			
CPP_D069	Member Data in Non-POD Class not Private	Yes			
CPP_D070	Template specialization shall be declared in the same file as the primary template	Yes			
CPP_D071	All user-provided class destructors, deallocation	Yes			

	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				
CPP_D072	Non-standard entities shall not be added to standard namespaces	Yes			
CPP_D073	There shall be one variable declaration per line	Yes			
CPP_D074	An external variable shall not be set to a value where it is being declared	Yes			
CPP_D075	Incorrect Order of Initialization	Yes			
CPP_D076	If a class requires a	Yes			

	virtual destructor but has nothing to undo from a constructor, an empty implementation should be provided.				
CPP_DD000	The defines, typedefs, structures, externals, globals, statics, external prototypes, and local prototypes shall be grouped by category.	Yes			
CPP_DD001	Use of global functions and variables shall be limited	Yes			
CPP_DD002	Globals should not be used in macros	Yes			
CPP_DD003	There shall be a function prototype for all functions	Yes			
CPP_DD004	Prototypes for static functions shall include the static storage class	Yes			

CPP_DD005	Any defined constants that are used as argument or return variables shall be placed in an include file	Yes			
CPP_DD006	Initializer lists shall be written in the order in which they are declared	Yes			
CPP_DD007	The private keyword should be used in class definitions	Yes			
CPP_DD008	Nesting template class definitions should be avoided.	Yes			
CPP_DD009	Assignment operators should check for self-assignment	Yes			
CPP_DD010	The use of friend classes should be avoided	Yes			
CPP_DD011	If the subscript operator (operator[]) is overloaded,	Yes			

	both the const and non-const versions should be defined.				
CPP_DD012	Layering techniques, where applicable, should be used instead of private inheritance.	Yes			
CPP_DD013	Public Inheritance not Used in a "is-a" Relationship	Yes			
CPP_DD014	Use the same parameter names and type qualifiers for all declarations and definitions	Yes			
CPP_DD015	Overload allocation and deallocation functions as a pair in the same scope	Yes			
CPP_DD016	Do not write syntactically ambiguous declarations	Yes			
CPP_DD017	Avoid cycles during initialization	Yes			

	of static objects				
CPP_DD018	Obey the one-definition rule	Yes			
CPP_DD019	Arrays shall not be partially initialized	Yes			
CPP_DD020	An element of an object shall not be initialized more than once	Yes			
CPP_DD021	Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	Yes			
CPP_DD022	Make sure that objects are initialized before they are used	Yes			
CPP_DD023	Use the same form in correspondin g uses of new and delete	Yes			
CPP_DD024	Postpone variable definitions as long as possible	Yes			

CPP_DD025	Avoid hiding inherited names	Yes			
CPP_DD026	Never redefine an inherited non-virtual function	Yes			
CPP_E000	A class type exception shall always be caught by reference	Yes			
CPP_E001	There should be at least one exception handler to catch all otherwise unhandled exceptions	Yes			
CPP_E002	Catch-All Statement Before Last	Yes			
CPP_E003	Catch Const References	Yes			
CPP_E004	Destructors Not Throw Exceptions	Yes			
CPP_E005	An empty throw (throw;) shall only be used in the compound-statement of a catch handler	Yes			
CPP_E006	Order of Catch Blocks with Derived	Yes			

	Classes				
CPP_E007	An exception object should not have pointer type	Yes			
CPP_E008	Exceptions shall be raised only after start-up and before termination of the program	Yes			
CPP_E009	Exceptions thrown shall be the type indicated by the function	Yes			
CPP_E010	Inconsistent Exception-Specification	Yes			
CPP_E011	No "errno" allowed	Yes			
CPP_E012	NULL shall not be thrown explicitly	Yes			
CPP_E013	Throw exceptions by value, not by pointer	Yes			
CPP_E014	The assignment-expression of a throw statement shall not itself cause an exception to be thrown	Yes			
CPP_E015	Expressions with type	Yes			

	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				
CPP_E016	Character Operators	Yes			
CPP_E017	Code Slicing Should Not Occur	Yes			
CPP_E018	Expressions with type enum or enum class shall not be used as operands to built-in and overloaded operators other than the subscript operator [ ], the assignment operator =, the equality	Yes			

	operators == and ! =, the unary & operator, and the relational operators <, <=, >, >=				
CPP_E019	Avoid Trigraphs	Yes			
CPP_E020	Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Yes			
CPP_E021	Octal and Hexadecimal Sequences	Yes			
CPP_E022	Escape sequences are standardized	Yes			
CPP_E023	Expression uses operand of side-effect more than once	Yes			
CPP_E024	Signed operands to modulus or division operator	Yes			
CPP_E025	Floating Equality Test	Yes			
CPP_E026	Minimization of run-time failures shall be ensured by the use of	Yes			

	static analysis tools				
CPP_E027	Only those escape sequences that are defined in ISO/IEC 14882:2014 shall be used	Yes			
CPP_E028	Hexadecimal constants should be upper case	Yes			
CPP_E029	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Yes			
CPP_E030	Concatenating String Literals of Different Encodings	Yes	Automated	Required	
CPP_E031	Traditional C-style casts shall not be used	Yes			
CPP_E032	Infeasible Paths	Yes			
CPP_E033	Do not rely on the value of a moved-from object	Yes			
CPP_E034	Limited dependence should be placed on C+	Yes			

	+ operator precedence rules in expressions				
CPP_E035	Parameter list (possibly empty) shall be included in every lambda expression	Yes			
CPP_E036	Specify Lambda Return Type	Yes			
CPP_E037	Lambda expressions should not be defined inside another lambda expression	Yes			
CPP_E038	Identical unnamed lambda expressions shall be replaced with a named function or a named lambda expression	Yes			
CPP_E039	A lambda shall not be an operand to decltype or typeid	Yes			
CPP_E040	dynamic_cast should not be used	Yes			
CPP_E041	reinterpret_c	Yes			

	ast shall not be used				
CPP_E042	Operands of Logical Boolean Operators	Yes			
CPP_E043	The increment (++) and decrement (--) operators shall not be mixed with other operators in an expression	Yes			
CPP_E044	Each operand of the ! operator, the logical && or the logical || operators shall have type bool	Yes			
CPP_E045	Evaluation of the operand to the sizeof operator shall not contain side effects	Yes			
CPP_E046	The right hand operand of a shift operator shall lie between zero and one less than the width in bits	Yes			

	of the underlying type of the left hand operand.				
CPP_E047	The ternary conditional operator shall not be used as a sub-expression	Yes			
CPP_E048	Each expression statement and identifier declaration shall be placed on a separate line	Yes			
CPP_E049	The comma operator shall not be used.	Yes			
CPP_E050A	Evaluation of the operand to the typeid operator shall not contain side effects	Yes			
CPP_E050B	The right hand operand of the integer division or remainder operators shall not be equal to zero	Yes			
CPP_E051	Unary Minus Operator Applied to an Expression with an	Yes			

	Unsigned Type				
CPP_E052	The right-hand operand of a logical && or || operator should not contain persistent side effects	Yes			
CPP_E053	Empty Throw	Yes			
CPP_E054	NULL Throw	Yes			
CPP_E055	Exception Object	Yes			
CPP_E056	A lambda expression object shall not outlive any of its reference-captured objects	Yes			
CPP_E057	Only instances of types derived from std::exception should be thrown	Yes			
CPP_E058	An exception object shall not be a pointer	Yes			
CPP_E059	All thrown exceptions should be unique	Yes			
CPP_E060	If a function exits with an exception, then before a	Yes			

	throw, the function shall place all objects/ resources that the function constructed in valid states or it shall delete them				
CPP_E061	Dynamic exception-specification shall not be used	Yes			
CPP_E062	A class type exception shall be caught by reference or const reference	Yes			
CPP_E063	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	Yes			

	exceptions not according to AUTOSAR C++14 guidelines				
CPP_E064	Unhandled Exceptions on Main Function	Yes			
CPP_E065	Condition of if statement shall be bool	Yes			
CPP_E066	Const Should be placed on the left-hand side of the comparison	Yes			
CPP_E067	Floats shall not be tested for direct equality	Yes			
CPP_E068	Provide a valid ordering predicate	Yes			
CPP_E069	Assignment in SubExpressions	Yes			
CPP_E070	Boolean operators	Yes			
CPP_E072	Int to Float Conversion	Yes			
CPP_E073	An implicit integral conversion shall not change the signedness of the underlying type	Yes			

CPP_E074	Operands shall not be of an inappropriate essential type	Yes			
CPP_E075	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category	Yes			
CPP_E077	The value of a composite expression shall not be assigned to an object with wider essential type	Yes			
CPP_E078	The value of a composite expression shall not be cast to a different essential type category or a wider essential type	Yes			
CPP_E079	Conversions	Yes			

	shall not be performed between a pointer to an incomplete type and any other type				
CPP_E080	A cast shall not be performed between a pointer to object type and a pointer to a different object type	Yes			
CPP_E081	A conversion should not be performed between a pointer to object and an integer type	Yes			
CPP_E082	Initializer lists shall not contain persistent side effects	Yes			
CPP_E083	The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type	Yes			

CPP_E084	The macro NULL shall be the only permitted form of integer null pointer constant	Yes			
CPP_E085	The result of an assignment operator should not be used	Yes			
CPP_E086	A loop counter shall not have essentially floating type	Yes			
CPP_E087	Minimize casting	Yes			
CPP_EH000	Program shall not be abruptly terminated	Yes			
CPP_EH001	The std::terminate() function shall not be called implicitly	Yes			
CPP_EH002	Library objects shall not generate error messages directly	Yes			
CPP_EH003	Destructors should not throw exceptions	Yes			
CPP_EH004	Exceptions	Yes			

	should be caught only by reference				
CPP_EH005	A declaration of non-throwing function shall contain noexcept specification	Yes			
CPP_EH006	If a function is declared to be noexcept, noexcept(true) or noexcept(<truecondition>), then it shall not exit with an exception	Yes			
CPP_EH007	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	Yes			
CPP_EH008	Exceptions thrown across execution boundaries	Yes			
CPP_EH009	New Method Throwing an Exception	Yes			
CPP_EH010	Use Assertion	Yes			

	Statements				
CPP_EH011	Catch exceptions by lvalue reference	Yes			
CPP_F000	All prototype parameters must have an identifier.	Yes			
CPP_F001	All class templates, function templates, class template member functions and class template static members shall be instantiated at least once	Yes			
CPP_F002	Const member functions shall not return non-const pointers or references to class-data	Yes			
CPP_F003	Unused Functions	Yes			
CPP_F004	Functions with no parameters need explicit void keyword	Yes			
CPP_F005	Declare functions at	Yes			

	file scope				
CPP_F006	A Function identifier shall either be used to call the function or it shall be preceded by &	Yes			
CPP_F007	Functions must not return objects by value.	Yes			
CPP_F008	Functions shall not be defined using the ellipsis notation	Yes			
CPP_F009	Use Named Parameters and Prototype Form	Yes			
CPP_F010	Functions shall not be declared implicitly	Yes			
CPP_F011	Inline functions defined in the class body	Yes			
CPP_F012	The identifier main shall not be used for a function other than the global function main	Yes			
CPP_F013	Member	Yes			

	functions shall not return non-const handles to class-data				
CPP_F014	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	Yes			
CPP_F015	Missing parameter name in function declarations	Yes			
CPP_F016	variable numbers of arguments shall not be used.	Yes			
CPP_F017	Overloaded function templates shall not be explicitly specialized	Yes			
CPP_F018	Parameters in an overriding virtual function shall either use the same default arguments as the function	Yes			

	they override, or else shall not specify any default arguments.				
CPP_F019	A pointer or reference parameter in a function shall be declared as pointer to const or reference to const if the correspondin g object is not modified	Yes			
CPP_F020	use the same identifier in definition and declaration of functions.	Yes			
CPP_F021	The features of <stdarg.h> shall not be used	Yes			
CPP_F022	Objects should not be passed by reference	Yes			
CPP_F023	A function parameter should not be modified	Yes			
CPP_F025	All functions with void return type shall have external side effect(s)	Yes			
CPP_F026	Every	Yes			

	function defined in an anonymous namespace, or static function with internal linkage, or private member function shall be used				
CPP_F027	There shall be no unused named parameters in non-virtual functions	Yes			
CPP_F028	There shall be no unused named parameters in the set of parameters for a virtual function and all the functions that override it	Yes			
CPP_F029	operator "new" and operator "delete" shall be defined together	Yes			
CPP_F030	If a project has a sized or unsized version of operator "delete"	Yes			

	globally defined, then both sized and unsized versions shall be defined				
CPP_F031	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Yes			
CPP_F032	A function shall not return a reference or a pointer to a parameter that is passed by reference to const.	Yes			
CPP_F033	Always return a value in non-void functions	Yes			
CPP_F034	Trivial accessor and mutator functions should be inlined.	Yes			
CPP_F035	Non-virtual public or protected member	Yes			

	functions shall not be redefined in derived classes				
CPP_F036	Use Override	Yes			
CPP_F037	Time Handling Functions of <ctime>	Yes			
CPP_F038_A	Check Parameters and Return Values - Ignored Return Values	Yes			
CPP_F039	A function that contains "forwarding reference" as its argument shall not be overloaded	Yes			
CPP_F040	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual	Yes			
CPP_F041	Member functions shall not return non-const raw pointers or references to private or	Yes			

	protected data owned by the class				
CPP_F042	If two opposite operators are defined, one shall be defined in terms of the other	Yes			
CPP_F043	Comparison operators shall be non-member functions with identical parameter types and noexcept	Yes			
CPP_F044	Overloaded Function Not Visible From Where it is Called	Yes			
CPP_F045	Virtual functions shall not be introduced in a final class	Yes			
CPP_F046	Predicate Function Objects Copied Incorrectly	Yes			
CPP_F047	A template constructor shall not participate in overload resolution for a single	Yes			

	argument of the enclosing class type				
CPP_F048	A non-member generic operator shall only be declared in a namespace that does not contain class (struct) type, enum type or union type declarations	Yes			
CPP_F049	Explicit specializations of function templates shall not be used	Yes			
CPP_F050	The noexcept specification of a function shall either be identical across all translation units, or identical or more restrictive between a virtual member function and an overrider	Yes			
CPP_F051	A function should be inlined only if it has one or	Yes			

	two lines of code				
CPP_F052	The function gets() should not be used	Yes			
CPP_F053	Every function shall have an explicitly declared return type.	Yes			
CPP_F054	Boolean functions shall explicitly return true or false	Yes			
CPP_F055	The default parameter list, when redeclaring or overriding methods, should be kept constant	Yes			
CPP_F056	Each function shall contain a prologue	Yes			
CPP_F057	Function prologue shall be in header or source	Yes			
CPP_F058	Function prologue shall contain certain specific information	Yes			
CPP_F059	Variable-length argument	Yes			

	lists should not be used				
CPP_F060	A method that does not change the visible properties of a class shall be declared const	Yes			
CPP_F061	The type of the return and all method arguments (even type void) shall be specified when defining a method	Yes			
CPP_F062	When overloading standardized operators (e.g., a += b, a-=b etc.), the resulting behavior should remain consistent with the expected outcome of the operator.	Yes			
CPP_F063	Member function arguments should not share the same name	Yes			

	as class state variables				
CPP_F064	Member functions should always be declared const unless they modify state variables	Yes			
CPP_F065	Any parameter not modified by a method should be passed to the method as a const.	Yes			
CPP_F066	Tail-Call Optimization	Yes			
CPP_F067	Functions declared with the [[noreturn]] attribute shall not return	Yes			
CPP_F069	A signal handler must be a plain old function	Yes			
CPP_F070	Consider alternatives to virtual functions	Yes			
CPP_H000	The #include directive shall be followed by either a <filename> or "filename"	Yes			

	sequence				
CPP_H001	The backslash character should not occur in a header file name	Yes			
CPP_H002	The ', ", /* or // characters shall not occur in a header file name	Yes			
CPP_H003	Definitions in Header Files	Yes			
CPP_H004	There shall be no unnamed namespaces in header files.	Yes			
CPP_H005	Objects or functions with external linkage shall be declared in a header file	Yes			
CPP_H006	It shall be possible to include any header file in multiple translation units without violating the One Definition Rule	Yes			
CPP_H007	Unnecessary #Includes	Yes			

CPP_H008	using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.	Yes			
CPP_H009	Header files, that are defined locally in the project, shall have a file name extension of one of: ".h", ".hpp" or ".hxx"	Yes			
CPP_H010	Header File Name	Yes			
CPP_H011	Absolute path names shall not be used for header files	Yes			
CPP_H012	All references to header files shall be listed one per line	Yes			
CPP_H013	Names of private header files should not be identical to names of library	Yes			

	header files				
CPP_H014	All public header files shall be capable of being included by a C++ file as well as a C file	Yes			
CPP_H016	If prototypes, typedefs, macros, structure definitions, or enums are needed in multiple modules, they shall be placed in header files	Yes			
CPP_H017	C++ version of the header file should be used	Yes			
CPP_H018	When including C Standard Library header files, C++ Standard Library header files without a '.h' appended should be used	Yes			
CPP_H019	Forward referencing should be	Yes			

	used, when appropriate, over direct inclusion when documenting dependencies within a header file.				
CPP_H020	The standard header file <tgmath.h> shall not be used	Yes			
CPP_H021	The standard header file <setjmp.h> shall not be used	Yes			
CPP_I000	A class, union or enum name (including qualification, if any) shall be a unique identifier	Yes			
CPP_I001	Different identifiers shall be typographically unambiguous	Yes			
CPP_I002	External identifiers shall be distinct	Yes			
CPP_I003	Identifiers that define objects or functions with external	Yes			

	linkage shall be unique				
CPP_I004	Global Namespace Declarations	Yes			
CPP_I005	Identifier name reuse	Yes			
CPP_I006	Identifiers shall be distinct from macro names	Yes			
CPP_I007	Identifiers declared in the same scope and name space shall be distinct	Yes			
CPP_I008	Identifiers that define objects or functions with internal linkage should be unique	Yes			
CPP_I009	Macro identifiers shall be distinct	Yes			
CPP_I010	The identifier name of a non-member object or function with static storage duration should not be reused	Yes			
CPP_I011	Identifier name significance	Yes			

CPP_I012	Static name reuse	Yes			
CPP_I013	A tag name shall be a unique identifier	Yes			
CPP_I014	A typedef name shall be a unique identifier.	Yes			
CPP_I015	No identifier in one name space should have the same spelling as an identifier in another name space.	Yes			
CPP_I016	Reserved Identifiers or Macros	Yes			
CPP_I017	Shadowed Identifiers	Yes			
CPP_I018	A class or enumeration name shall not be hidden by a variable, function or enumerator declaration in the same scope	Yes			
CPP_I019	The identifier name of a non-member object with static storage duration or static	Yes			

	function shall not be reused within a namespace				
CPP_I020	An identifier name of a function with static storage duration or a non-member object with external or internal linkage should not be reused	Yes			
CPP_I021	Universal character names shall be used only inside character or string literals	Yes			
CPP_I022	Similiar Entity Names within Multiple Inheritance	Yes			
CPP_I023	Uppercase 'O' shall not be used as an identifier	Yes			
CPP_I024	Lowercase 'l' shall not be used as an identifier	Yes			
CPP_I025	The using namespace directive should be used only at the method or function	Yes			

	scope.				
CPP_L000	Calls to COTS library functions that might throw an exception must be enclosed in a try block.	Yes			
CPP_L001	The C library shall not be used	Yes			
CPP_L002	The signal handling facilities of <csignal> shall not be used	Yes			
CPP_L003	The stream input/output library <cstdio> shall not be used	Yes			
CPP_L004	<cstdlib> Library Functions	Yes			
CPP_L005	Avoid atof, atoi, atol, and atoll from <cstdlib> or <stdlib.h>	Yes			
CPP_L006	Unbounded Functions of <cstring>	Yes			
CPP_L007	Avoid using the library <ctime>	Yes			
CPP_L008	No "errno" allowed	Yes			
CPP_L009	No offsetof	Yes			

	allowed				
CPP_L010	The setjmp macro and the longjmp function shall not be used	Yes			
CPP_L011	Signal.h should not be used	Yes			
CPP_L012	Standard Library Function Names	Yes			
CPP_L013	Avoid including stdio.h	Yes			
CPP_L014	Library stdlib.h - avoid: abort, exit, getenv and system	Yes			
CPP_L015	Guarantee that library functions do not overflow	Yes			
CPP_L016	The library <time.h> shall not be used	Yes			
CPP_L017	Inputs from independent components shall be validated	Yes			
CPP_L018	Ensure your random number generator is properly seeded	Yes			
CPP_L019	Random number	Yes			

	engines shall not be default-initialized				
CPP_L020	Do not unlock or destroy another POSIX thread's mutex	Yes			
CPP_L021	An iterator shall not be implicitly converted to const_iterator	Yes			
CPP_L022	An argument to std::forward shall not be subsequently used	Yes			
CPP_L023	The std::move shall not be used on objects declared const or const&	Yes			
CPP_L024	Forwarding values to other functions shall be done via: (1) std::move if the value is an rvalue reference, (2) std::forward	Yes			

	if the value is forwarding reference				
CPP_L025	The std::bind shall not be used	Yes			
CPP_L026	Alternate input and output operations on a file stream shall not be used without an intervening flush or positioning call	Yes			
CPP_L027	All std::hash specializations for user-defined types shall have a noexcept function call operator	Yes			
CPP_L028	The std::auto_ptr type shall not be used	Yes			
CPP_L029	Library <clocale> (locale.h)	Yes			
CPP_L030	Avoid deadlock with POSIX threads by locking in predefined order	Yes			
CPP_L031	Evaluation of the operand	Yes			

	to the typeid operator shall not contain side effects.				
CPP_L033	Reserved Builtin Macros	Yes			
CPP_L034	Use of the iostream library is preferred over stdio.h	Yes			
CPP_M000	Dynamic heap memory allocation	Yes			
CPP_M001	The form of the delete expression shall match the form of the new expression used to allocate the memory	Yes			
CPP_M002	Non-placement new or delete expressions shall not be used	Yes			
CPP_M003	Bitwise operations and operations that assume data representation in memory shall not be performed on objects.	Yes			

CPP_M004	Moved-from object shall not be read-accessed.	Yes			
CPP_M005	Uninitialized Memory Read	Yes			
CPP_M006	Functions malloc, calloc, realloc and free shall not be used	Yes			
CPP_M007	When reading strings a maximum field width should be specified	Yes			
CPP_M008	Dynamically allocated memory shall be set to some value prior to its use as an rvalue or in a test	Yes			
CPP_M009	Memory that has been freed shall not be referenced	Yes			
CPP_M010	The new[] and delete[] operators shall be used for the allocation and deallocation of memory	Yes			

	resources				
CPP_M011	The delete[] operator shall be used to deallocate arrays that have been allocated with the new[] operator	Yes			
CPP_M012	The delete[] operator shall be called in the destructor for all member pointers in an object that are pointing to memory that was dynamically allocated by that object	Yes			
CPP_M013	Users shall provide a copy constructor, destructor and assignment operator for a class that uses dynamic memory allocation	Yes			
CPP_M014	The operator new should be called with the	Yes			

	nothrow option.				
CPP_M015	When overloading the new[] operator, a correspondin g delete[] operator should be provided.	Yes			
CPP_M016	Overloaded new operator should not hide the global new operator	Yes			
CPP_M017	All local allocations made in a method, other than the destructor, should be deallocated prior to exiting the method.	Yes			
CPP_M018	Dynamic Memory Usage on Realtime Phase	Yes			
CPP_M019	No Dynamic Memory Allocation	Yes			
CPP_M020	Properly pair allocation and deallocation functions	Yes			

CPP_M021	Declare objects shared between POSIX threads with appropriate storage durations	Yes			
CPP_N000	Naming Convention: Classes	Yes			
CPP_N001	Naming Convention: Enumerator	Yes			
CPP_N002	Naming Convention: Enums	Yes			
CPP_N003	Naming Convention: Files	Yes			
CPP_N004	Naming Convention: Functions	Yes			
CPP_N005	Naming Convention: Macros	Yes			
CPP_N006	Naming Convention: Namespaces	Yes			
CPP_N007	Naming Convention: Parameters	Yes			
CPP_N008	Naming Convention: Structs	Yes			
CPP_N009	Naming Convention: Typedefs	Yes			
CPP_N010	Naming Convention: Unions	Yes			

CPP_N011	Naming Convention: Variables	Yes			
CPP_N012	Only those characters specified in the C++ Language Standard basic source character set shall be used in the source code	Yes			
CPP_N013	Naming Convention: Header File Names	Yes			
CPP_N014	Naming Convention: Implementation File Names	Yes			
CPP_N015	Implementation files, that are defined locally in the project, should have a file name extension of ".cpp"	Yes			
CPP_N016	User defined suffixes of the user defined literal operators shall start with underscore followed by one or more	Yes			

	letters				
CPP_N017	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	Yes			
CPP_N018	All macros shall be fully capitalized	Yes			
CPP_N019	Function and variable names shall not be fully capitalized	Yes			
CPP_P000	No more than 2 levels of pointer indirection	Yes			
CPP_P001	Hide Implementation of Pointers Not Dereferenced	Yes			
CPP_P002	Pointer initialization must use 0, not NULL.	Yes			
CPP_P003	Pointer function parameters must be tested for equality to 0	Yes			

	before accessing the data being pointed to				
CPP_P004	Pointers Must Be Initialized	Yes			
CPP_P005	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives	Yes			
CPP_P006	std::make_unique shall be used to construct objects owned by std::unique_ptr	Yes			
CPP_P007	A std::unique_ptr shall be used over std::shared_ptr if ownership sharing is not required	Yes			
CPP_P008	Do Not Use #define	Yes			
CPP_P009	In the definition of a function-like macro, each instance of a parameter	Yes			

	shall be enclosed in parentheses, unless it is used as the operand of # or ##				
CPP_P010	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related	Yes			
CPP_P011	Ifndef Wrappers or Pragma Once	Yes			
CPP_P012	File Include Matching Header	Yes			
CPP_P013	Function-like macros shall not be defined	Yes			
CPP_P014_A	Restrict Pointer Usage - Multiple Dereferences	Yes			
CPP_P014_B	Restrict Pointer Usage - Other	Yes			
CPP_P015	Inactive Code	Yes			
CPP_P017	#include directives in	Yes			

	a file shall only be preceded by other preprocessor directives or comments				
CPP_P018	A macro shall not be defined with the same name as a keyword	Yes			
CPP_P019	Macros in Blocks	Yes			
CPP_P020	C++ macros shall only be used for include guards, type qualifiers, or storage class specifiers	Yes			
CPP_P021	Before dereferencing a pointer, compare it with NULL	Yes			
CPP_P022	The pre-processor shall only be used for file inclusion and include guards	Yes			
CPP_P023	Reserved identifiers, macros and functions in the standard library shall not be	Yes			

	defined, redefined or undefined				
CPP_P024	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.	Yes			
CPP_P025	All macro identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef preprocessor directives and the defined() operator	No			
CPP_P026	avoid #undef	Yes			
CPP_P028	A smart pointer shall only be used as a parameter type if it expresses lifetime semantics	Yes			

CPP_P029	A project should not contain unused macro declarations	Yes			
CPP_P030	Invalid Use of std::shared_ptr	Yes			
CPP_P031	Invalid Use of std::unique_ptr	Yes			
CPP_P032	Cyclic Structure of std::shared_ptr	Yes			
CPP_P033	For pointer declarations, the asterisk shall be placed with the variable	Yes			
CPP_P034	Const Member Function Returning Non-Const Pointer or Reference	Yes			
CPP_P035	std::make_shared shall be used to construct objects owned by std::shared_ptr	Yes			
CPP_P036	A std::shared_ptr shall be used to represent	Yes			

	shared ownership				
CPP_P037	A std::unique_ptr shall be used to represent exclusive ownership	Yes			
CPP_P038	An already-owned pointer value shall not be stored in an unrelated smart pointer	Yes			
CPP_P039	String literals shall not be assigned to non-constant pointers	Yes			
CPP_P040	Only nullptr literal shall be used as the null-pointer-constant	Yes			
CPP_P041	Subtraction between pointers shall only be applied to pointers that address elements of the same array	Yes			
CPP_P042	Pointer arithmetic shall not be used with pointers to	Yes			

	non-final classes				
CPP_P043	>, >=, <, <= shall not be applied to objects of pointer type, except where they point to the same array	Yes			
CPP_P044	Deleting Pointers to Incomplete Class Types	Yes			
CPP_P045	Array indexing over pointer arithmetic	Yes			
CPP_P046	A pointer pointing to an element of an array of objects shall not be passed to a smart pointer of single object type	Yes			
CPP_P047	A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type	Yes			
CPP_P048	A pointer to member virtual function shall	Yes			

	only be tested for equality with null-pointer-constant				
CPP_P049	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array	Yes			
CPP_P050	Literal zero (0) shall not be used as the null-pointer-constant.	Yes			
CPP_P051	Pointer to Integer Cast	Yes			
CPP_P052	A parameter shall be passed by reference if it can't be NULL	Yes			
CPP_P053	A pointer to member shall not access non-existent class members	Yes			
CPP_P054	References should be used instead of pointers when	Yes			

	possible.				
CPP_P055	For pointer declarations, the placement of the * shall be consistent	Yes			
CPP_P056	Pointer functions shall return a valid pointer on success and a zero pointer on failure	Yes			
CPP_P057	A pointer to dynamic memory that is declared and allocated locally should be declared as an auto_ptr.	Yes			
CPP_P058	Store newed objects in smart pointers in standalone statements	Yes			
CPP_P059	A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast	Yes			
CPP_P060	Prefer pass-by-	Yes			

	reference-to-const to pass by value				
CPP_P061	Shared Pointer Capture	Yes			
CPP_PR001	Include guards shall be provided	Yes			
CPP_PR002	Constants defined by #define shall be explicitly declared with uppercase suffixes	Yes			
CPP_PR003	Macros shall not be used to change language syntax	Yes			
CPP_PR004	Limit Preprocessor Usage	Yes			
CPP_PR005	#include directives should only be preceded by preprocessor directives or comments	Yes			
CPP_PR006	There shall be at most one occurrence of the # or ## operators in a single macro definition	Yes			
CPP_PR007	The defined	Yes			

	preprocessor operator shall only be used in one of the two standard forms				
CPP_PR021	The names of standard library macros and objects shall not be reused	Yes			
CPP_PR030	The #pragma directive shall not be used	Yes			
CPP_PR031	#error directive shall not be used	Yes			
CPP_PR032	The # and ## operators should not be used	Yes			
CPP_PR033	The macro offsetof shall not be used	Yes			
CPP_PR034	There shall be no unused include directives (slow)	Yes			
CPP_PR036	Invalid Preprocessor Directives	Yes			
CPP_PR037	Undefined macro identifiers shall not be used in #if or #elif	Yes			

	preprocessor directives, except as operands to the defined operator				
CPP_PR038	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 ##	Yes			
CPP_PR039	Function-like Macro Containing Preprocessing Directives	Yes			
CPP_PR040	#include Directives Not Grouped Together	Yes			
CPP_PR041	Incorrect Use of Pre-processor	Yes			
CPP_S000	no unions	Yes			
CPP_S001	Flexible array members shall not be declared	Yes			
CPP_S002	Incorrect Initializer Lists	Yes			
CPP_S003	A type defined as struct shall:	Yes			

	(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				
CPP_S004	Unions Shall not be Used	Yes			
CPP_SA_DEAD_STORES	Dead Stores	Yes			
CPP_ST001	Not more than one space should precede a ";" with the exception of the null statement	Yes			
CPP_ST002	Equal signs should be aligned when they occur in a series of assignment operators	Yes			
CPP_ST003	Placement of braces for functions shall adhere to one of the following	Yes			

	formats and shall be consistent				
CPP_ST004	Code between the beginning and ending braces of a function shall start with one level of indentation	Yes			
CPP_ST005	Enum lists should not contain a trailing comma	Yes			
CPP_ST006	No line of code should extend beyond column 80	Yes			
CPP_ST007	Declarations shall not be made within an individual block but shall be placed at the function level or at the module level.	Yes			
CPP_ST008	Blank lines should be used to separate distinct algorithmic parts	Yes			
CPP_ST009	Parentheses should be used in	Yes			

	lengthy logical and arithmetic expressions for clarity.				
CPP_ST010	Items should be logically grouped	Yes			
CPP_ST011	Inline functions should be used instead of macros	Yes			
CPP_ST012	Names that differ in case only or that look similar should not be used.	Yes			
CPP_ST013	Switch statements should be used instead of deeply nested else-ifs when testing a variable for multiple values	Yes			
CPP_ST014	No line of code should extend beyond 80 characters	Yes			
CPP_ST015	Incrementing and decrementing control variables in loops	Yes			
CPP_ST016	Calls to free	Yes			

	should have an if test around them if it is uncertain that the pointer has been properly allocated.				
CPP_ST017	White space shall not be used in the following places	Yes			
CPP_ST018	Continuation lines shall be indented at least one level from the line being continued	Yes			
CPP_ST019	Statements under case labels shall be indented one level	Yes			
CPP_ST020	For the if-else, while, do, and for control structure, the statement(s) shall be indented one level	Yes			
CPP_ST021	Placement of braces for constructs shall be consistent within a	Yes			

	module				
CPP_ST022	Boolean expressions involving non-boolean values should always use an explicit test for equality or non-equality.	Yes			
CPP_ST023	At least one blank line shall be placed before comments	Yes			
CPP_ST024	Functions shall have at least one blank line between them	Yes			
CPP_ST025	Each area of declarations shall have at least one blank line before and after it	Yes			
CPP_ST026	Class naming conventions	Yes			
CPP_ST027	Naming conventions for class data members vs. member function internal data	Yes			
CPP_ST028	Data type naming conventions	Yes			
CPP_ST029	Immutable	Yes			

	data naming conventions				
CPP_ST030	Class design should include the following format	Yes			
CPP_ST031	Separate lines should be used for each member declaration	Yes			
CPP_ST032	Indentation shall be at least three spaces, and consistent across modules	Yes			
CPP_ST033	Short Functions	Yes			
CPP_T000	Typedefs that indicate size and signedness should be used in place of the basic numerical types	Yes			
CPP_T001	Arguments to character-handling functions shall be representable as an unsigned char	Yes			
CPP_T002	The std::vector<bool>	Yes			

	specialization shall not be used				
CPP_T003	There should be no unused type declarations	Yes			
CPP_T004	Type long double shall not be used	Yes			
CPP_T005	Type wchar_t shall not be used	Yes			
CPP_T006	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	Yes			
CPP_T007	A cvalue expression shall not be implicitly converted to a different underlying type	Yes			
CPP_T008	An implicit integral conversion shall not change the signedness of the	Yes			

	underlying type				
CPP_T009	There shall be no implicit floating-integral conversions	Yes			
CPP_T010	An implicit integral or floating-point conversion shall not reduce the size of the underlying type	Yes			
CPP_T011	There shall be no explicit floating-integral conversions of a cvalue expression	Yes			
CPP_T012	An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue expression	Yes			
CPP_T013	An explicit integral conversion shall not change the signedness of the underlying	Yes			

	type of a cvalue expression				
CPP_T014	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	Yes			
CPP_T015	The plain char type shall only be used for the storage and use of character values	Yes			
CPP_T016	Signed char and unsigned char type shall only be used for the storage and use of numeric values	Yes			
CPP_T017	The first operand of a conditional-operator shall	Yes			

	have type bool				
CPP_T018	Bitwise operators shall only be applied to operands of unsigned underlying type	Yes			
CPP_T019	C-style Array	Yes			
CPP_T020	Casts from a base class to a derived class should not be performed on polymorphic types	Yes			
CPP_T021	A cast shall not remove any const or volatile qualification from the type of a pointer or reference	Yes			
CPP_T022	An object with integer type or pointer to void type shall not be converted to an object with pointer type.	Yes			
CPP_T023	Array to Pointer Decay	Yes			
CPP_T024	NULL shall not be used	Yes			

	as an integer value				
CPP_T025	CV-qualifiers shall be placed on the right hand side of the type that is a typedef or a using name	Yes			
CPP_T026	The typedef specifier shall not be used	Yes			
CPP_T027	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration	Yes			
CPP_T028	Enumeration underlying base type shall be explicitly defined	Yes			
CPP_T029	In an enumeration, either (1) none, (2) the first or (3) all enumerators shall be initialized	Yes			
CPP_T030	When declaring	Yes			

	function templates, the trailing return type syntax shall be used if the return type depends on the type of parameters.				
CPP_T031	Common ways of passing parameters should be used.	Yes			
CPP_T032	Multiple output values from a function should be returned as a struct or tuple.	Yes			
CPP_T033	"consume" parameters declared as X && shall always be moved from.	Yes			
CPP_T034	"forward" parameters declared as T && shall always be forwarded.	Yes			
CPP_T035	"in" parameters for "cheap to copy" types shall be passed by	Yes			

	value.				
CPP_T036	Output parameters shall not be used.	Yes			
CPP_T037	"in-out" parameters declared as T & shall be modified.	Yes			
CPP_T038	Fixed Width Integers	Yes			
CPP_T039	Non-constant operands to a binary bitwise operator shall have the same underlying type	Yes			
CPP_T040	User defined literals operators shall only perform conversion of passed parameters	Yes			
CPP_T041	A binary arithmetic operator and a bitwise operator shall return a "prvalue"	Yes			
CPP_T042	A relational operator shall return a boolean value	Yes			

CPP_T043	If "operator[]" is to be overloaded with a non-const version, const version shall also be implemented	Yes			
CPP_T044	Undocumented Use of Floating-point Arithmetic	Yes			
CPP_T045	Undocumented Use of Scaled-integer or Fixed-point Arithmetic	Yes			
CPP_T046	Assigning Object to an Overlapping Object	Yes			
CPP_T047	Data types used for interfacing	Yes			
CPP_T048	All user-defined conversion operators shall be defined explicit	Yes			
CPP_T049	User-defined conversion operators should not be used	Yes			
CPP_T050	Types shall be explicitly	Yes			

	specified				
CPP_T051	C-style strings shall not be used	Yes			
CPP_T052	String-to-Number Conversion Handling	Yes			
CPP_T053	A type used as a template argument shall provide all members that are used by the template	Yes			
CPP_T054A	An array or container shall not be accessed beyond its range (Part A)	Yes			
CPP_T054B	An array or container shall not be accessed beyond its range Part B	Yes			
CPP_T055	A value should not be retrieved from a structure or union with a different type than with which it was stored	Yes			
CPP_T056	Explicit type casting shall be used	Yes			

	when performing calculations with a mix of signed and unsigned values.				
CPP_T057	Actual arguments shall be explicitly type cast to the appropriate type	Yes			
CPP_T058	Simple integers shall be used to test and set booleans	Yes			
CPP_T059	Width-sensitive types should be typedef'd and placed in a header file	Yes			
CPP_T060	Converting a pointer to integer or integer to pointer	Yes			
CPP_T061	All Checks/Language Specific/C and C++/Types/Use Const whenever possible	Yes			
CPP_U000	Digraphs shall not be used	Yes			

CPP_U001	Discarded return values.	Yes			
CPP_U002	Inline Functions have more than X LOC	Yes			
CPP_U003	Unused Parameters in Non-virtual Functions	Yes			
CPP_U004	Unused Static Globals	Yes			
CPP_U005	A project should not contain unused tag declarations	Yes			
CPP_U006	A project shall not contain unused type declarations	Yes			
CPP_U007	Unused Labels	Yes			
CPP_U008	Unnecessary Friends	Yes			
CPP_U009	Special Member Functions	Yes			
CPP_U010	Unused Entities	Yes			
CPP_V000	Magic Numbers	Yes			
CPP_V001	One Variable per Line	Yes			
CPP_V002	Reference Symbols Spacing, (& *)	Yes			
CPP_V003	Declare each	Yes			

	variable in a separate declaration statement				
CPP_V004	A project shall not contain non-volatile POD variables having only one use.	Yes			
CPP_V005	Types or externals declared at the function level.	Yes			
CPP_V006	A variable which is not modified shall be const qualified	Yes			
CPP_V007	Unused Local Variables	Yes			
CPP_V008	Unused Static Global	Yes			
CPP_V009	Using-directives shall not be used.	Yes			
CPP_V010	Variables should be commented	Yes			
CPP_V011	All variables shall have a defined value before they are used	Yes			
CPP_V012	Explicit Virtual	Yes			
CPP_V013	There shall be no more	Yes			

	than one definition of each virtual function on each path through the inheritance hierarchy				
CPP_V014	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual	Yes			
CPP_V015	There shall be no unused parameters (named or unnamed) in the set of parameters for a virtual function and all the functions that override it	Yes			
CPP_V016	Virtual Call in Constructor/ Destructor	Yes			
CPP_V017	A project shall not contain instances of non-volatile variables being given values that	Yes			

	are not subsequently used				
CPP_V018	Auto Variable	Yes			
CPP_V019	Initializing Variables Without Using Braced-Initialization	Yes			
CPP_V020	Class members that are not dependent on template class parameters should be defined in a separate base class	Yes			
CPP_V021	Variables should not be altered more than once in an expression	Yes			
CPP_V022	Variables shall not be implicitly captured in a lambda expression	Yes			
CPP_V023	Literal values shall not be used apart from type initialization, otherwise symbolic names shall be used	Yes			

	instead				
CPP_V024	Variables of type char shall be explicitly qualified as signed or unsigned when used to store numbers	Yes			
CPP_V025	Every variable shall be declared with a specific type	Yes			
CPP_V026	Local variables shall be initialized when declared	Yes			
CPP_V027	Globals in header files shall be ifdef'd	Yes			
CPP_V028	Constants should be declared as const values as opposed to #define directives.	Yes			
CPP_V029	The const_cast operator should be used exclusively for altering the constness	Yes			

	attribute of a variable.				
CPP_V030	The dynamic_cast operator should be used exclusively for casting within an inheritance hierarchy.	Yes			
CPP_V031	The static_cast operator should be used for routine cast operations not provided by const_cast and dynamic_cast.	Yes			
CPP_V032	Use of the reinterpret_cast operator should be avoided	Yes			
CPP_V033	Typedef'd variables in a class shall be placed in an include file	Yes			
CPP_V034	STL containers (vector, list, map, etc.) should be used instead of C-style	Yes			

	arrays whenever possible.				
CPP_V035	Objects that do not outlive a function shall have automatic storage duration	Yes			
CPP_V036	Static data member initialization should be placed in the class .cpp file	Yes			
CPP_V037	Initializer lists should be used to initialize member variables over direct assignment.	Yes			
CPP_V038	The concept of information hiding should be implemented.	Yes			
CPP_V039	Within an object, most instance variables should be accessed directly. Methods should be used to set variables whose values	Yes			

	are determined by an algorithm.				
CPP_V042	An object shall not be accessed outside of its lifetime	No			
CPP_VF000	Every class that contains virtual functions shall provide a virtual destructor	Yes			
CPP_VF001	Access levels should not be mixed (public, protected, private) when overriding virtual functions.	Yes			
CPP_VF002	Virtual Call in Constructor/ Destructor	Yes			
CTR50-CPP	Guarantee that container indices and iterators are within the valid range	Yes			High
CTR51-CPP	Use valid references, pointers, and iterators to reference elements of a container	Yes			High

CTR52-CPP	Guarantee that library functions do not overflow	Yes			High
CTR53-CPP	Use valid iterator ranges	Yes			High
CTR54-CPP	Do not subtract iterators that do not refer to the same container	Yes			Medium
CTR55-CPP	Do not use an additive operator on an iterator if the result would overflow	Yes			
CTR56-CPP	Do not use pointer arithmetic on polymorphic objects	Yes			High
CTR57-CPP	Provide a valid ordering predicate	Yes			Low
CTR58-CPP	Predicate function objects should not be mutable	Yes			Low
DCL31-C	Declare identifiers before using them	Yes			Low
DCL36-C	Do not declare an identifier with conflicting linkage	Yes			Medium

	classification s				
DCL38-C	Use the correct syntax when declaring a flexible array member	Yes			Low
DCL39-C	Avoid information leakage when passing a structure across a trust boundary	No			Low
DCL40-C	Do not create incompatible declarations of the same function or object	Yes			Low
DCL50-CPP	Do not define a C-style variadic function	Yes			High
DCL52-CPP	Never qualify a reference type with const or volatile	Yes			Low
DCL53-CPP	Do not write syntactically ambiguous declarations	Yes			Low
DCL54-CPP	Overload allocation and deallocation functions as a pair in the same scope	Yes			Low
DCL55-CPP	Avoid	No			Low

	information leakage when passing a class object across a trust boundary				
DCL56-CPP	Avoid cycles during initialization of static objects	Yes			Low
DCL57-CPP	Do not let exceptions escape from destructors or deallocation functions	Yes			Low
DCL58-CPP	Do not modify the standard namespaces	Yes			High
DCL60-CPP	Obey the one-definition rule	Yes			High
EFFECTIVECPP_02	2. Do Not Use #define	Yes			
EFFECTIVECPP_03	3. Use Const whenever possible	Yes			
EFFECTIVECPP_04	4. Make sure that objects are initialized before they are used	Yes			
EFFECTIVECPP_07	7. Non-Virtual Destructors in Base Classes	Yes			
EFFECTIVECPP_08	8. Exceptions in Destructors	Yes			

EFFECTIVECPP_09	9. Virtual Call in Constructor/ Destructor	Yes			
EFFECTIVECPP_10	10. Assignment Operator Return This	Yes			
EFFECTIVECPP_11	11. Assignment Operator Self Assignment	Yes			
EFFECTIVECPP_16	16. Use the same form in corresponding uses of new and delete	Yes			
EFFECTIVECPP_17	17. Store newed objects in smart pointers in standalone statements	Yes			
EFFECTIVECPP_20	20. Prefer pass-by-reference-to-const to pass by value	Yes			
EFFECTIVECPP_22	22. Datamembers should be declared private	Yes			
EFFECTIVECPP_26	26. Postpone variable definitions as long as possible	Yes			
EFFECTIVECPP_27	27. Minimize casting	Yes			

EFFECTIVECPP_33	33. Avoid hiding inherited names	Yes			
EFFECTIVECPP_35	35. Consider alternatives to virtual functions	Yes			
EFFECTIVECPP_36	36. Never redefine an inherited non-virtual function	Yes			
ERR32-C	Do not rely on indeterminate values of errno	No			Low
ERR33-C	Detect and handle standard library errors	Yes			High
ERR34-C	Detect errors when converting a string to a number	Yes			Medium
ERR50-CPP	Do not abruptly terminate the program	Yes			Low
ERR51-CPP	Handle all exceptions	Yes			Low
ERR52-CPP	Do not use setjmp() or longjmp()	Yes			Low
ERR53-CPP	Do not reference base classes or class data members in a constructor	Yes			Low

	or destructor function-try-block handler				
ERR54-CPP	Catch handlers should order their parameter types from most derived to least derived	Yes			Medium
ERR55-CPP	Honor exception specifications	Yes			Low
ERR57-CPP	Do not leak resources when handling exceptions	Yes			Low
ERR58-CPP	Handle all exceptions thrown before main() begins executing	Yes			Low
ERR59-CPP	Do not throw an exception across execution boundaries	Yes			High
ERR60-CPP	Exception objects must be nothrow copy constructible	Yes			Low
ERR61-CPP	Catch exceptions by lvalue reference	Yes			Low
ERR62-CPP	Detect errors	Yes			Medium

	when converting a string to a number				
EXP35-C	Do not modify objects with temporary lifetime	No			Low
EXP40-C	Do not modify constant objects	No			Low
EXP42-C	Do not compare padding data	Yes			Medium
EXP43-C	Avoid undefined behavior when using restrict-qualified pointers	No			Medium
EXP50-CPP	Do not depend on the order of evaluation for side effects	Yes			Medium
EXP51-CPP	Do not delete an array through a pointer of the incorrect type	Yes			Low
EXP52-CPP	Do not rely on side effects in unevaluated operands	Yes			Low
EXP53-CPP	Do not read uninitialized memory	Yes			High

EXP54-CPP	Do not access an object outside of its lifetime	Yes			High
EXP55-CPP	Do not access a cv-qualified object through a cv-unqualified type	Yes			Medium
EXP56-CPP	Do not call a function with a mismatched language linkage	No			Low
EXP57-CPP	Do not cast or delete pointers to incomplete classes	Yes			Medium
EXP58-CPP	Pass an object of the correct type to va_start	Yes			Medium
EXP59-CPP	Use offsetof() on valid types and members	Yes			Medium
EXP61-CPP	A lambda object must not outlive any of its reference captured objects	Yes			High
EXP62-CPP	Do not access the bits of an object	Yes			High

	representation that are not part of the object's value representation				
EXP63-CPP	Do not rely on the value of a moved-from object	Yes			Medium
FIO32-C	Do not perform operations on devices that are only appropriate for files	No			Medium
FIO34-C	Distinguish between characters read from a file and EOF or WEOF	No			High
FIO38-C	Do not copy a FILE object	Yes			Low
FIO50-CPP	Do not alternately input and output from a file stream without an intervening positioning call	Yes			Low
FIO51-CPP	Close files when they are no longer needed	Yes			Medium
FLP30-C	Do not use floating-point variables as	Yes			Low

	loop counters				
FLP32-C	Prevent or detect domain and range errors in math functions	No			Medium
FLP34-C	Ensure that floating-point conversions are within range of the new type	No			Low
FLP36-C	Preserve precision when converting integral values to floating-point type	No			Low
FLP37-C	Do not use object representations to compare floating-point values	Yes			Low
HIS_01	1. Comment Density (COMF)	Yes			
HIS_02	2. Number of Paths(PATH)	Yes			
HIS_03	3. Number of Goto Statements(GOTO)	Yes			
HIS_04	4. Cyclomatic Complexity (v(G))	Yes			
HIS_05	5. Calling Functions	Yes			

	(CALLING)				
HIS_06	6. Called Functions (CALLS)	Yes			
HIS_07	7. Function Parameters (PARAM)	Yes			
HIS_08	8. Number of Staments(STMT)	Yes			
HIS_09	9. Number of call levels(LEVEL)	Yes			
HIS_10	10. Number of return points (RETURN)	Yes			
HIS_11	11. Language scope(VOCF)	Yes			
HIS_12	12. Recursion (AP_CG_CYCLE)	Yes			
HIS_13	13. Statements Changed (SCHG)	Yes			
HIS_14	14. Statements Deleted (SDEL)	Yes			
HIS_15	15. New Statements (SNEW)	Yes			
HIS_16	16. Stability Index (S)	Yes			
HIS_17	17. MISRA-HIS Violations (NOMV)	Yes			
HIS_18	18. MISRA-HIS	Yes			

	Violations per Rule (NOMVPR)				
INT32-C	Ensure that operations on signed integers do not result in overflow	No			High
INT34-C	Do not shift an expression by a negative number of bits or by greater than or equal to the number of bits that exist in the operand	No			Low
INT35-C	Use correct integer precisions	No			Low
INT36-C	Converting a pointer to integer or integer to pointer	Yes			Low
INT50-CPP	Do not cast to an out-of-range enumeration value	Yes			Medium
M0-1-1	A project shall not contain unreachable code	Yes	Automated	Required	
M0-1-2	A project shall not contain	Yes	Automated	Required	

	infeasible paths				
M0-1-3	A project shall not contain unused variables	Yes	Automated	Required	
M0-1-4	A project shall not contain non-volatile POD variables having only one use.	Yes	Automated	Required	
M0-1-8	All functions with void return type shall have external side effect(s)	Yes	Automated	Required	
M0-1-9	There shall be no dead code	No	Automated	Required	
M0-1-10	Every defined function shall be called at least once.	Yes	Automated	Advisory	
M0-2-1	Assigning Object to an Overlapping Object	Yes	Automated	Required	
M0-3-1	Minimization of run-time failures shall be ensured by the use of static analysis tools	Yes	Non-automated	Required	
M0-3-2	If a function generates error	No	Non-automated	Required	

	information, then that error information shall be tested				
M0-4-1	Undocumented Use of Scaled-integer or Fixed-point Arithmetic	Yes	Non-automated	Required	
M0-4-2	Undocumented Use of Floating-point Arithmetic	Yes	Non-automated	Required	
M1-0-2	Multiple compilers shall only be used if they have a common, defined interface	No	Non-automated	Required	
M2-7-1	The character sequence /* shall not be used within a C-style comment.	Yes	Automated	Required	
M2-10-1	Different identifiers shall be typographically unambiguous	Yes	Automated	Required	
M2-13-2	Octal constants (other than zero) and	Yes	Automated	Required	

	octal escape sequences (other than "\0") shall not be used.				
M2-13-3	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Yes	Automated	Required	
M2-13-4	Literal suffixes shall be upper case	Yes	Automated	Required	
M3-1-2	Functions shall not be declared at block scope	Yes	Automated	Required	
M3-2-1	All declarations of an object or function shall have compatible types	Yes	Automated	Required	
M3-2-2	The One Definition Rule	Yes	Automated	Required	
M3-2-3	A type, object or function that is used in multiple translation units shall be declared in one and only one file	Yes	Automated	Required	

M3-2-4	An identifier with external linkage shall have exactly one definition	Yes	Automated	Required	
M3-3-2	If a function has internal linkage then all redeclarations shall include the static storage class specifier	Yes	Automated	Required	
M3-4-1	Declarations at Lowest Scope	Yes	Automated	Required	
M3-9-1	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	Yes	Automated	Required	
M3-9-3	The underlying bit representations of floating-point values shall not be used	Yes	Automated	Required	
M4-5-1	Expressions with type	Yes	Automated	Required	

	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	Character Operators	Yes	Automated	Required	
M4-10-1	NULL shall not be used as an integer value	Yes	Automated	Required	
M4-10-2	Literal zero (0) shall not be used as the null-pointer-constant.	Yes	Automated	Required	
M5-0-2	Limited dependence should be placed on C++ operator precedence rules in expressions	Yes	Automated	Advisory	
M5-0-3	A cvalue expression	Yes	Automated	Required	

	shall not be implicitly converted to a different underlying type				
M5-0-4	An implicit integral conversion shall not change the signedness of the underlying type	Yes	Automated	Required	
M5-0-5	There shall be no implicit floating-integral conversions	Yes	Automated	Required	
M5-0-6	An implicit integral or floating-point conversion shall not reduce the size of the underlying type	Yes	Automated	Required	
M5-0-7	There shall be no explicit floating-integral conversions of a cvalue expression	Yes	Automated	Required	
M5-0-8	An explicit integral or floating-point conversion shall not increase the	Yes	Automated	Required	

	size of the underlying type of a cvalue expression				
M5-0-9	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression	Yes	Automated	Required	
M5-0-10	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	Yes	Automated	Required	
M5-0-11	The plain char type shall only be used for the storage and use of character values	Yes	Automated	Required	

M5-0-12	Signed char and unsigned char type shall only be used for the storage and use of numeric values	Yes	Automated	Required	
M5-0-14	The first operand of a conditional-operator shall have type bool	Yes	Automated	Required	
M5-0-15	Array indexing over pointer arithmetic	Yes	Automated	Required	
M5-0-16	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array	Yes	Automated	Required	
M5-0-17	Subtraction between pointers shall only be applied to pointers that address elements of the same array	Yes	Automated	Required	
M5-0-18	>, >=, <, <=	Yes	Automated	Required	

	shall not be applied to objects of pointer type, except where they point to the same array				
M5-0-20	Non-constant operands to a binary bitwise operator shall have the same underlying type	Yes	Automated	Required	
M5-0-21	Bitwise operators shall only be applied to operands of unsigned underlying type	Yes	Automated	Required	
M5-2-2	A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast	Yes	Automated	Required	
M5-2-3	Casts from a base class to a derived class should not be performed on	Yes	Automated	Advisory	

	polymorphic types				
M5-2-6	A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type	Yes	Automated	Required	
M5-2-8	An object with integer type or pointer to void type shall not be converted to an object with pointer type.	Yes	Automated	Required	
M5-2-9	Pointer to Integer Cast	Yes	Automated	Required	
M5-2-10	The increment (++) and decrement (--) operators shall not be mixed with other operators in an expression	Yes	Automated	Required	
M5-2-11	The comma operator, && operator and the || operator shall not be overloaded	Yes	Automated	Required	

M5-2-12	Array to Pointer Decay	Yes	Automated	Required	
M5-3-1	Each operand of the ! operator, the logical && or the logical || operators shall have type bool	Yes	Automated	Required	
M5-3-2	Unary Minus Operator Applied to an Expression with an Unsigned Type	Yes	Automated	Required	
M5-3-3	The unary & operator shall not be overloaded	Yes	Automated	Required	
M5-3-4	Evaluation of the operand to the sizeof operator shall not contain side effects	Yes	Automated	Required	
M5-8-1	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	Yes	Partially Automated	Required	

	operand.				
M5-14-1	The right hand operand of a logical &&, || operators shall not contain side effects	Yes	Automated	Required	
M5-17-1	The semantic equivalence between a binary operator and its assignment operator form shall be preserved	Yes	Non-automated	Required	
M5-18-1	The comma operator shall not be used.	Yes	Automated	Required	
M5-19-1	Evaluation of constant unsigned integer expressions shall not lead to wrap-around	No	Automated	Required	
M6-2-1	Assignment operators shall not be used in sub-expressions	Yes	Automated	Required	
M6-2-2	Floating-point expressions shall not be directly or indirectly tested for	Yes	Partially Automated	Required	

	equality or inequality				
M6-2-3	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	Yes	Automated	Required	
M6-3-1	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement	Yes	Automated	Required	
M6-4-1	An if (condition) construct shall be followed by a compound statement. The else keyword shall be followed by either a	Yes	Automated	Required	

	compound statement, or another if statement				
M6-4-2	All if and else if constructs shall be terminated with an else clause	Yes	Automated	Required	
M6-4-3	Switch Statement not Well-formed	Yes	Automated	Required	
M6-4-4	A switch-label shall only be used when the most closely-enclosing compound statement is the body of a switch statement	Yes	Automated	Required	
M6-4-5	An unconditional throw or break statement shall terminate every non-empty switch-clause	Yes	Automated	Required	
M6-4-6	The final clause of a switch statement shall be the default-	Yes	Automated	Required	

	clause				
M6-4-7	The condition of a switch statement shall not have bool type	Yes	Automated	Required	
M6-5-2	If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to <=, <, > or >=	Yes	Automated	Required	
M6-5-3	The loop-counter shall not be modified within condition or statement	Yes	Automated	Required	
M6-5-4	The loop-counter shall be modified by one of: --, ++, -= n, or += n; where n remains constant for the duration of the loop	Yes	Automated	Required	
M6-5-5	A loop-control-variable other than the loop-counter shall not be	Yes	Automated	Required	

	modified within condition or expression				
M6-5-6	A loop-control-variable other than the loop-counter which is modified in statement shall have type bool	Yes	Automated	Required	
M6-6-1	Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement	Yes	Automated	Required	
M6-6-2	The goto statement shall jump to a label declared later in the same function body	Yes	Automated	Required	
M6-6-3	Continue Statement Used in a not Well-formed For Loop	Yes	Automated	Required	
M7-1-2	A pointer or	Yes	Automated	Required	

	reference parameter in a function shall be declared as pointer to const or reference to const if the correspondin g object is not modified				
M7-3-1	Global Namespace Declarations	Yes	Automated	Required	
M7-3-2	The identifier main shall not be used for a function other than the global function main	Yes	Automated	Required	
M7-3-3	There shall be no unnamed namespaces in header files.	Yes	Automated	Required	
M7-3-4	Using-directives shall not be used.	Yes	Automated	Required	
M7-3-6	using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be	Yes	Automated	Required	

	used in header files.				
M7-4-1	Assembly Language Code Usage not Documented	Yes	Non-automated	Required	
M7-4-2	Assembler instructions shall only be introduced using the asm declaration.	Yes	Automated	Required	
M7-4-3	Assembly language shall be encapsulated and isolated.	Yes	Automated	Required	
M7-5-1	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Yes	Non-automated	Required	
M7-5-2	The address of an object with automatic storage shall not be assigned to another object that may persist after the first	Yes	Non-automated	Required	

	object has ceased to exist.				
M8-0-1	Single Declarations	Yes	Automated	Required	
M8-3-1	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.	Yes	Automated	Required	
M8-4-2	The identifiers used for the parameters in a re-declaration of a function shall be identical to those in the declaration.	Yes	Automated	Required	
M8-4-4	A function identifier shall either be used to call the function or it shall be preceded by &.	Yes	Automated	Required	
M8-5-2	Incorrect Initializer Lists	Yes	Automated	Required	

M9-3-1	Const Member Function Returning Non-Const Pointer or Reference	Yes	Automated	Required	
M9-3-3	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	Yes	Automated	Required	
M9-6-1	When the absolute positioning of bits representing a bit-field is required, then the behavior and packing of bit-fields shall be documented	No	Non-automated	Required	
M9-6-4	Bit-field Length	Yes	Automated	Required	
M10-1-1	Class Derived From Virtual Bases	Yes	Automated	Advisory	
M10-1-2	A base class shall only be declared virtual if it is used in a diamond	Yes	Automated	Required	

	hierarchy				
M10-1-3	An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes	Automated	Required	
M10-2-1	Similiar Entity Names within Multiple Inheritance	Yes	Automated	Advisory	
M10-3-3	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual	Yes	Automated	Required	
M11-0-1	Member Data in Non-POD Class not Private	Yes	Automated	Required	
M12-1-1	An object's dynamic type shall not be used from the body of its constructor or destructor	Yes	Automated	Required	
M14-5-3	A copy assignment operator shall be declared when there is a template	Yes	Automated	Required	

	assignment operator with a parameter that is a generic parameter				
M14-6-1	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->	Yes	Automated	Required	
M15-0-3	Control shall not be transferred into a try or catch block using a goto or a switch statement	Yes	Non-automated	Required	
M15-1-1	Exception Object	Yes	Automated	Required	
M15-1-2	NULL Throw	Yes	Automated	Required	
M15-1-3	Empty Throw	Yes	Automated	Required	
M15-3-1	Exceptions shall be raised only after start-up and before termination of the program	Yes	Automated	Required	
M15-3-3	Handlers of a function-try-block	Yes	Automated	Required	

	implementation of a class constructor or destructor shall not reference non-static members from this class or its bases				
M15-3-4	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	Yes	Automated	Required	
M15-3-6	Order of Catch Blocks with Derived Classes	Yes	Automated	Required	
M15-3-7	Where multiple handlers are provided in a single try-catch statement or function-try-block, any ellipsis (catch-all) handler shall occur last	Yes	Automated	Required	
M16-0-1	#include Directives	Yes	Automated	Required	

	Not Grouped Together				
M16-0-2	Macros shall only be #define'd or #undef'd in the global namespace.	Yes	Automated	Required	
M16-0-5	Function-like Macro Containing Preprocessing Directives	Yes	Automated	Required	
M16-0-6	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 ##	Yes	Automated	Required	
M16-0-7	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator	Yes	Automated	Required	
M16-0-8	Invalid Preprocessor Directives	Yes	Automated	Required	
M16-1-1	The defined	Yes	Automated	Required	

	preprocessor operator shall only be used in one of the two standard forms				
M16-1-2	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related	Yes	Non-automated	Required	
M16-2-3	Include guards shall be provided	Yes	Automated	Required	
M16-3-1	There shall be at most one occurrence of the # or ## operators in a single macro definition	Yes	Automated	Required	
M16-3-2	The # and ## operators should not be used	Yes	Automated	Advisory	
M17-0-2	The names of standard library macros and objects shall not be reused	Yes	Automated	Required	

M17-0-3	Standard Library Function Names	Yes	Automated	Required	
M17-0-5	The setjmp macro and the longjmp function shall not be used	Yes	Automated	Required	
M18-0-3	<cstdlib> Library Functions	Yes	Automated	Required	
M18-0-4	Time Handling Functions of <ctime>	Yes	Automated	Required	
M18-0-5	Unbounded Functions of <cstring>	Yes	Automated	Required	
M18-2-1	The macro offsetof shall not be used	Yes	Automated	Required	
M18-7-1	The signal handling facilities of <csignal> shall not be used	Yes	Automated	Required	
M19-3-1	The error indicator errno shall not be used	Yes	Automated	Required	
M27-0-1	The stream input/output library <cstdio> shall not be used	Yes	Automated	Required	
MEM30-C	Do not access freed memory	No			High
MEM36-C	Do not	No			Low

	modify the alignment of objects by calling realloc()				
MEM50-CPP	Do not access freed memory	No			High
MEM51-CPP	Properly deallocate dynamically allocated resources	Yes			High
MEM52-CPP	Detect and handle memory allocation errors	Yes			High
MEM53-CPP	Explicitly construct and destruct objects when manually managing object lifetime	No			High
MEM57-CPP	Avoid using default operator new for over-aligned types	Yes			Medium
METRIC_00	Program Unit Call Count	Yes			
METRIC_01	Program Unit Callby Count	Yes			
METRIC_02	Program Unit Comment to Code Ratio	Yes			
METRIC_03	Program Unit Cyclomatic Complexity	Yes			
METRIC_04	Program Unit	Yes			

	Max Length				
METRIC_05	Program Unit Max Nesting Depth	Yes			
METRIC_06	Program Unit Parameters Count	Yes			
METRIC_07	Program Unit Path Count	Yes			
METRIC_08	Program Unit Statement Count	Yes			
METRIC_09	Coupling Between Object Classes	Yes			
METRIC_11	Depth of Inheritance Tree	Yes			
METRIC_12	Lack of Cohesion in Methods	Yes			
METRIC_13	Maintainability Index	Yes			
MISRA04_8.7	8.7 Objects shall be local if only accessed from one function	Yes		Required	
MISRA08_0-1-1	0-1-1 A project shall not contain unreachable code	Yes		Required	
MISRA08_0-1-2	0-1-2 Infeasible Paths	Yes		Required	
MISRA08_0-1-3	0-1-3 A project shall not contain	Yes		Required	

	unused variables				
MISRA08_0-1-4	0-1-4 A project shall not contain non-volatile POD variables having only one use.	Yes		Required	
MISRA08_0-1-5	0-1-5 A project shall not contain unused type declarations	Yes		Required	
MISRA08_0-1-7	0-1-7 The value returned by a function having a non-void return type that is not an overloaded operator shall always be used	Yes		Required	
MISRA08_0-1-8	0-1-8 All functions with void return type shall have external side effect(s)	Yes		Required	
MISRA08_0-1-9	0-1-9 There shall be no dead code	No		Required	
MISRA08_0-1-10	0-1-10 All defined functions called	Yes		Required	
MISRA08_0-	0-1-11	Yes		Required	

1-11	Unused Parameters in Non-virtual Functions				
MISRA08_0-1-12	0-1-12 There shall be no unused parameters (named or unnamed) in the set of parameters for a virtual function and all the functions that override it	Yes		Required	
MISRA08_0-3-1	0-3-1 Minimization of run-time failures shall be ensured by the use of static analysis tools	Yes		Document	
MISRA08_0-3-2	0-3-2 If a function generates error information, then that error information shall be tested	No		Required	
MISRA08_2-3-1	2-3-1 Trigraphs shall not be used	Yes		Required	
MISRA08_2-5-1	2-5-1 Digraphs	Yes		Advisory	

	shall not be used				
MISRA08_2-7-1	2-7-1 The character sequence /* shall not be used within a C-style comment.	Yes		Required	
MISRA08_2-7-2	2-7-2 Sections of code shall not be "commented out"	Yes		Required	
MISRA08_2-10-1	2-10-1 Different identifiers shall be typographically unambiguous	Yes		Required	
MISRA08_2-10-2	2-10-2 Shadowed Identifiers	Yes		Required	
MISRA08_2-10-3	2-10-3 A typedef name shall be a unique identifier	Yes		Required	
MISRA08_2-10-4	2-10-4 A class, union or enum name (including qualification, if any) shall be a unique identifier	Yes		Required	
MISRA08_2-10-5	2-10-5 The identifier name of a	Yes		Advisory	

	non-member object or function with static storage duration should not be reused				
MISRA08_2-13-1	2-13-1 escape sequences are standardized	Yes		Required	
MISRA08_2-13-2	2-13-2 Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Yes		Required	
MISRA08_2-13-3	2-13-3 A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Yes		Required	
MISRA08_2-13-4	2-13-4 Literal suffixes shall be upper case	Yes		Required	
MISRA08_2-13-5	2-13-5 Narrow and wide string literals shall not be concatenated	Yes		Required	
MISRA08_3-1-1	3-1-1 It shall be possible	Yes		Required	

	to include any header file in multiple translation units without violating the One Definition Rule				
MISRA08_3-1-2	3-1-2 Functions shall not be declared at block scope	Yes		Required	
MISRA08_3-1-3	3-1-3 When an array is declared, its size shall either be stated explicitly or defined implicitly by initialization	Yes		Required	
MISRA08_3-2-1	3-2-1 All declarations of an object or function shall have compatible types	Yes		Required	
MISRA08_3-2-2	3-2-2 The One Definition Rule	Yes		Required	
MISRA08_3-2-3	3-2-3 A type, object or function that is used in multiple translation	Yes		Required	

	units shall be declared in one and only one file				
MISRA08_3-2-4	3-2-4 An identifier with external linkage shall have exactly one definition	Yes		Required	
MISRA08_3-3-1	3-3-1 Objects or functions with external linkage shall be declared in a header file	Yes		Required	
MISRA08_3-3-2	3-3-2 If a function has internal linkage then all redeclarations shall include the static storage class specifier	Yes		Required	
MISRA08_3-4-1	3-4-1 Declarations at Lowest Scope	Yes		Required	
MISRA08_3-9-1	3-9-1 The types used for an object, a function return type, or a function parameter shall be token-for-	Yes		Required	

	token identical in all declarations and re-declarations				
MISRA08_3-9-2	3-9-2 Typedefs that indicate size and signedness should be used in place of the basic numerical types	Yes		Advisory	
MISRA08_3-9-3	3-9-3 The underlying bit representations of floating-point values shall not be used	Yes		Required	
MISRA08_4-5-1	4-5-1 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 &	Yes		Required	

	operator, and the conditional operator				
MISRA08_4-5-2	4-5-2 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 <, <=, >, >=	Yes		Required	
MISRA08_4-5-3	4-5-3 Character Operators	Yes		Required	
MISRA08_4-10-1	4-10-1 NULL shall not be used as an integer value	Yes		Required	
MISRA08_4-10-2	4-10-2 Literal zero (0) shall not be used as the null-pointer-constant.	Yes		Required	
MISRA08_5-0-2	5-0-2 Limited dependence	Yes		Advisory	

	should be placed on C++ operator precedence rules in expressions				
MISRA08_5-0-3	5-0-3 A cvalue expression shall not be implicitly converted to a different underlying type	Yes		Required	
MISRA08_5-0-4	5-0-4 An implicit integral conversion shall not change the signedness of the underlying type	Yes		Required	
MISRA08_5-0-5	5-0-5 There shall be no implicit floating-integral conversions	Yes		Required	
MISRA08_5-0-6	5-0-6 An implicit integral or floating-point conversion shall not reduce the size of the underlying type	Yes		Required	
MISRA08_5-	5-0-7 There	Yes		Required	

0-7	shall be no explicit floating-integral conversions of a cvalue expression				
MISRA08_5-0-8	5-0-8 An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue expression	Yes		Required	
MISRA08_5-0-9	5-0-9 An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression	Yes		Required	
MISRA08_5-0-10	5-0-10 If the bitwise operators ~ and << are applied to an operand with an underlying type of unsigned char or unsigned	Yes		Required	

	short, the result shall be immediately cast to the underlying type of the operand				
MISRA08_5-0-11	5-0-11 The plain char type shall only be used for the storage and use of character values	Yes		Required	
MISRA08_5-0-12	5-0-12 Signed char and unsigned char type shall only be used for the storage and use of numeric values	Yes		Required	
MISRA08_5-0-14	5-0-14 The first operand of a conditional-operator shall have type bool	Yes		Required	
MISRA08_5-0-17	5-0-17 Subtraction between pointers shall only be applied to pointers that address	Yes		Required	

	elements of the same array				
MISRA08_5-0-18	5-0-18 >, >=, <, <= shall not be applied to objects of pointer type, except where they point to the same array	Yes		Required	
MISRA08_5-0-19	5-0-19 No more than 2 levels of pointer indirection	Yes		Required	
MISRA08_5-0-20	5-0-20 Non-constant operands to a binary bitwise operator shall have the same underlying type	Yes		Required	
MISRA08_5-0-21	5-0-21 Bitwise operators shall only be applied to operands of unsigned underlying type	Yes		Required	
MISRA08_5-2-3	5-2-3 Casts from a base class to a derived class should not be	Yes		Advisory	

	performed on polymorphic types				
MISRA08_5-2-5	5-2-5 A cast shall not remove any const or volatile qualification from the type of a pointer or reference	Yes		Required	
MISRA08_5-2-6	5-2-6 A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type	Yes		Required	
MISRA08_5-2-8	5-2-8 An object with integer type or pointer to void type shall not be converted to an object with pointer type.	Yes		Required	
MISRA08_5-2-9	5-2-9 Pointer to Integer Cast	Yes		Advisory	
MISRA08_5-2-10	5-2-10 The increment (++) and decrement (--) operators shall not be	Yes		Advisory	

	mixed with other operators in an expression				
MISRA08_5-2-11	5-2-11 The comma operator, && operator and the || operator shall not be overloaded	Yes		Required	
MISRA08_5-2-12	5-2-12 Array to Pointer Decay	Yes		Required	
MISRA08_5-3-1	5-3-1 Each operand of the ! operator, the logical && or the logical || operators shall have type bool	Yes		Required	
MISRA08_5-3-3	5-3-3 The unary & operator shall not be overloaded	Yes		Required	
MISRA08_5-3-4	5-3-4 Evaluation of the operand to the sizeof operator shall not contain side effects	Yes		Required	
MISRA08_5-8-1	5-8-1 The right hand operand of a shift operator shall lie	Yes		Required	

	between zero and one less than the width in bits of the underlying type of the left hand operand.				
MISRA08_6-2-2	6-2-2 Floating-point expressions shall not be directly or indirectly tested for equality or inequality	Yes		Required	
MISRA08_6-2-3	6-2-3 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	Yes		Required	
MISRA08_6-3-1	6-3-1 The statement forming the body of a	Yes		Required	

	switch, while, do ... while or for statement shall be a compound statement				
MISRA08_6-4-1	6-4-1 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	Yes		Required	
MISRA08_6-4-2	6-4-2 All if ... else if constructs shall be terminated with an else clause	Yes		Required	
MISRA08_6-4-4	6-4-4 A switch-label shall only be used when the most closely-enclosing compound statement is the body of a switch statement	Yes		Required	
MISRA08_6-4-5	6-4-5 An unconditional	Yes		Required	

	throw or break statement shall terminate every non-empty switch-clause				
MISRA08_6-4-6	6-4-6 The final clause of a switch statement shall be the default-clause	Yes		Required	
MISRA08_6-4-8	6-4-8 Every switch statement shall have at least one case clause	Yes		Required	
MISRA08_6-5-1	6-5-1 A for loop shall contain a single loop-counter which shall not have floating-point type	Yes		Required	
MISRA08_6-5-2	6-5-2 If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to	Yes		Required	

	<=, <, > or >=				
MISRA08_6-5-3	6-5-3 The loop-counter shall not be modified within condition or statement	Yes		Required	
MISRA08_6-5-4	6-5-4 The loop-counter shall be modified by one of: --, ++, -= n, or += n; where n remains constant for the duration of the loop	Yes		Required	
MISRA08_6-5-5	6-5-5 A loop-control-variable other than the loop-counter shall not be modified within condition or expression	Yes		Required	
MISRA08_6-5-6	6-5-6 A loop-control-variable other than the loop-counter which is modified in statement shall have type bool	Yes		Required	
MISRA08_6-	6-6-1 Any	Yes		Required	

6-1	label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement				
MISRA08_6-6-2	6-6-2 The goto statement shall jump to a label declared later in the same function body	Yes		Required	
MISRA08_6-6-4	6-6-4 For any iteration statement there shall be no more than one break or goto statement used for loop termination	Yes		Required	
MISRA08_6-6-5	6-6-5 A function shall have a single point of exit at the end of the function	Yes		Required	
MISRA08_7-1-1	7-1-1 A variable which is not modified	Yes		Required	

	shall be const qualified				
MISRA08_7-1-2	7-1-2 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	Yes		Required	
MISRA08_7-2-1	7-2-1 An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration	Yes		Required	
MISRA08_7-3-1	7-3-1 Global Namespace Declarations	Yes		Required	
MISRA08_7-3-2	7-3-2 The identifier main shall not be used for a function other than the global function main	Yes		Required	
MISRA08_7-3-3	7-3-3 There shall be no	Yes		Required	

	unnamed namespaces in header files.				
MISRA08_7-3-4	7-3-4 Using-directives shall not be used.	Yes		Required	
MISRA08_7-3-5	7-3-5 Multiple declarations for an identifier in the same namespace shall not straddle a using-declaration for that identifier	Yes		Required	
MISRA08_7-3-6	7-3-6 using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.	Yes		Required	
MISRA08_7-4-2	7-4-2 Assembler instructions shall only be introduced using the asm declaration.	Yes		Required	
MISRA08_7-	7-4-3	Yes		Required	

4-3	Assembly language shall be encapsulated and isolated.				
MISRA08_7-5-1	7-5-1 A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Yes		Required	
MISRA08_7-5-2	7-5-2 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.	Yes		Required	
MISRA08_7-5-4	7-5-4 Functions should not call themselves, either directly or indirectly.	Yes		Advisory	
MISRA08_8-0-1	8-0-1 Single Declarations	Yes		Required	
MISRA08_8-	8-3-1	Yes		Required	

3-1	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.				
MISRA08_8-4-1	8-4-1 Functions shall not be defined using the ellipsis notation	Yes		Required	
MISRA08_8-4-2	8-4-2 Use the same identifier in definition and declaration of functions.	Yes		Required	
MISRA08_8-4-3	8-4-3 Always return a value in non-void functions	Yes		Required	
MISRA08_8-4-4	8-4-4 A function identifier shall either be used to call the function or it shall be preceded by &	Yes		Required	
MISRA08_8-5-1	8-5-1 All variables	Yes		Required	

	shall have a defined value before they are used				
MISRA08_8-5-2	8-5-2 Incorrect Initializer Lists	Yes	Automated	Required	
MISRA08_8-5-3	8-5-3 The = construct in enumerator list shall only be used on either the first item alone, or all items explicitly.	Yes		Required	
MISRA08_9-3-1	9-3-1 Const member functions shall not return non-const pointers or references to class-data	Yes		Required	
MISRA08_9-3-2	9-3-2 Member functions shall not return non-const handles to class-data	Yes		Required	
MISRA08_9-3-3	9-3-3 If a member function can be made static then it shall be made static,	Yes		Required	

	otherwise if it can be made const then it shall be made const				
MISRA08_9-5-1	9-5-1 Unions shall not be used	Yes		Required	
MISRA08_9-6-1	9-6-1 When the absolute positioning of bits representing a bit-field is required, then the behavior and packing of bit-fields shall be documented	No		Document	
MISRA08_9-6-4	9-6-4 (Fuzzy parser) Named bit-fields with signed integer type shall have a length of more than one bit	Yes		Required	
MISRA08_10-1-1	10-1-1 Classes should not be derived from virtual bases	Yes		Advisory	
MISRA08_10-1-2	10-1-2 A base class shall only be declared virtual if it is used in a	Yes		Required	

	diamond hierarchy				
MISRA08_10-1-3	10-1-3 An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes		Required	
MISRA08_10-3-1	10-3-1 There shall be no more than one definition of each virtual function on each path through the inheritance hierarchy	Yes		Required	
MISRA08_10-3-2	10-3-2 Each overriding virtual function shall be declared with the virtual keyword.	Yes		Required	
MISRA08_10-3-3	10-3-3 A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual	Yes		Required	
MISRA08_11-	11-0-1	Yes		Required	

0-1	Member data in non-POD class types shall be private				
MISRA08_12-1-1	12-1-1 An object's dynamic type shall not be used from the body of its constructor or destructor	Yes		Required	
MISRA08_12-1-2	12-1-2 Explicitly call all immediate and virtual base classes	Yes		Advisory	
MISRA08_12-1-3	12-1-3 All constructors that are callable with a single argument of fundamental type shall be declared explicit.	Yes		Required	
MISRA08_12-8-1	12-8-1 A copy constructor shall only initialize its base classes and the non-static members of the class of which it is a member	Yes		Required	
MISRA08_14	14-5-2 A	Yes		Required	

-5-2	copy constructor shall be declared when there is a template constructor with a single parameter that is a generic parameter				
MISRA08_14-5-3	14-5-3 A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter	Yes		Required	
MISRA08_14-7-1	14-7-1 All class templates, function templates, class template member functions and class template static members shall be instantiated at least once	Yes		Required	
MISRA08_14	14-8-1	Yes		Required	

-8-1	Overloaded function templates shall not be explicitly specialized				
MISRA08_15-0-2	15-0-2 An exception object should not have pointer type	Yes		Advisory	
MISRA08_15-1-1	15-1-1 The assignment-expression of a throw statement shall not itself cause an exception to be thrown	Yes		Required	
MISRA08_15-1-2	15-1-2 NULL shall not be thrown explicitly	Yes		Required	
MISRA08_15-1-3	15-1-3 An empty throw (throw;) shall only be used in the compound-statement of a catch handler	Yes		Required	
MISRA08_15-3-1	15-3-1 Exceptions shall be raised only after start-up and before termination of the program	Yes		Required	

MISRA08_15-3-2	15-3-2 There should be at least one exception handler to catch all otherwise unhandled exceptions	Yes		Advisory	
MISRA08_15-3-3	15-3-3 Members in function-try-blocks in constructors or destructors	Yes		Required	
MISRA08_15-3-5	15-3-5 A class type exception shall always be caught by reference	Yes		Required	
MISRA08_15-3-6	15-3-6 Order of Catch Blocks with Derived Classes	Yes		Required	
MISRA08_15-3-7	15-3-7 Catch-All Statement Before Last	Yes		Required	
MISRA08_15-4-1	15-4-1 Inconsistent Exception-Specification	Yes		Required	
MISRA08_15-5-1	15-5-1 A class destructor shall not exit with an exception	Yes		Required	
MISRA08_15	15-5-2	Yes		Required	

-5-2	Exceptions thrown shall be the type indicated by the function				
MISRA08_16-0-1	16-0-1 #include directives in a file shall only be preceded by other preprocessor directives or comments	Yes		Required	
MISRA08_16-0-2	16-0-2 Macros shall only be #define'd or #undef'd in the global namespace	Yes		Required	
MISRA08_16-0-3	16-0-3 #undef shall not be used	Yes		Required	
MISRA08_16-0-4	16-0-4 Function-like macros shall not be defined	Yes		Required	
MISRA08_16-0-5	16-0-5 Arguments to a function-like macro shall not contain tokens that look like preprocessing directives	Yes		Required	
MISRA08_16-0-6	16-0-6 In the definition of a	Yes		Required	

	function-like macro, each instance of a parameter shall be enclosed in parentheses, unless it is used as the operand of # or ##				
MISRA08_16-0-7	16-0-7 Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator	Yes		Required	
MISRA08_16-0-8	16-0-8 Invalid Preprocessor Directives	Yes		Required	
MISRA08_16-1-1	16-1-1 The defined preprocessor operator shall only be used in one of the two standard forms	Yes		Required	
MISRA08_16-1-2	16-1-2 All #else, #elif and #endif preprocessor directives shall reside in	Yes		Required	

	the same file as the #if, #ifdef or #ifndef directive to which they are related				
MISRA08_16-2-1	16-2-1 The pre-processor shall only be used for file inclusion and include guards	Yes		Required	
MISRA08_16-2-2	16-2-2 C++ macros shall only be used for include guards, type qualifiers, or storage class specifiers	Yes		Required	
MISRA08_16-2-3	16-2-3 Include guards shall be provided	Yes		Required	
MISRA08_16-2-4	16-2-4 The ', ", /* or // characters shall not occur in a header file name	Yes		Required	
MISRA08_16-2-5	16-2-5 The backslash character should not occur in a header file name	Yes		Advisory	
MISRA08_16	16-2-6 The	Yes		Required	

-2-6	#include directive shall be followed by either a <filename> or "filename" sequence				
MISRA08_16-3-1	16-3-1 There shall be at most one occurrence of the # or ## operators in a single macro definition	Yes		Required	
MISRA08_16-3-2	16-3-2 The # and ## operators should not be used	Yes		Advisory	
MISRA08_17-0-1	17-0-1 Reserved identifiers, macros and functions in the standard library shall not be defined, redefined or undefined	Yes		Required	
MISRA08_17-0-2	17-0-2 The names of standard library macros and objects shall not be reused	Yes		Required	
MISRA08_17	17-0-3	Yes		Required	

-0-3	Standard Library Function Names				
MISRA08_17-0-5	17-0-5 The setjmp macro and the longjmp function shall not be used	Yes		Required	
MISRA08_18-0-1	18-0-1 The C library shall not be used	Yes		Required	
MISRA08_18-0-2	18-0-2 The library functions atof, atoi and atol from library <cstdlib> shall not be used	Yes		Required	
MISRA08_18-0-3	18-0-3 The library functions abort, exit, getenv and system from library <cstdlib> shall not be used	Yes		Required	
MISRA08_18-0-4	18-0-4 The time handling functions of library <ctime> shall not be used	Yes		Required	
MISRA08_18-0-5	18-0-5 Unbounded Functions of <cstring>	Yes		Required	

MISRA08_18-2-1	18-2-1 The macro offsetof shall not be used.	Yes		Required	
MISRA08_18-4-1	18-4-1 Dynamic heap memory allocation shall not be used.	Yes		Required	
MISRA08_18-7-1	18-7-1 The signal handling facilities of <csignal> shall not be used	Yes		Required	
MISRA08_19-3-1	19-3-1 The error indicator "errno" shall not be used.	Yes		Required	
MISRA08_27-0-1	27-0-1 The stream input/output library <cstdio> shall not be used	Yes		Required	
MISRA12_2.3	2.3 A project should not contain unused type declarations	Yes		Advisory	
MISRA12_2.7	2.7 There should be no unused parameters in functions	Yes		Advisory	
MISRA12_3.1	3.1 The character sequences /* and // shall	Yes		Required	

	not be used within a comment				
MISRA12_4.1	4.1 Octal and Hexadecimal Sequences	Yes		Required	
MISRA12_5.9	5.9 Identifiers that define objects or functions with internal linkage should be unique	Yes		Advisory	
MISRA12_6.1	6.1 Bit-fields shall only be declared with an appropriate type	Yes		Required	
MISRA12_8.2	8.2 Use Named Parameters and Prototype Form	Yes			
MISRA12_8.9	8.9 Objects shall be local if only accessed from one function	Yes		Advisory	
MISRA12_8.10	8.10 Non-static Inline Functions	Yes		Required	
MISRA12_9.1	9.1 The value of an object with automatic storage duration shall	Yes		Mandatory	

	not be read before it has been set				
MISRA12_9.2	9.2 The initializer for an aggregate or union shall be enclosed in braces	Yes		Required	
MISRA12_9.3	9.3 Arrays shall not be partially initialized	Yes		Required	
MISRA12_9.4	9.4 An element of an object shall not be initialized more than once	Yes		Required	
MISRA12_9.5	9.5 Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	Yes		Required	
MISRA12_10.1	10.1 Operands shall not be of an inappropriate essential type	Yes		Required	
MISRA12_10.4	10.4 Both operands of an operator in which the	Yes		Required	

	usual arithmetic conversions are performed shall have the same essential type category				
MISRA12_10.6	10.6 The value of a composite expression shall not be assigned to an object with wider essential type	Yes		Required	
MISRA12_10.8	10.8 The value of a composite expression shall not be cast to a different essential type category or a wider essential type	Yes		Required	
MISRA12_11.1	11.1 Conversions shall not be performed between a pointer to a function and any other type	Yes		Required	

MISRA12_11.2	11.2 Conversions shall not be performed between a pointer to an incomplete type and any other type	Yes		Required	
MISRA12_11.3	11.3 A cast shall not be performed between a pointer to object type and a pointer to a different object type	Yes		Required	
MISRA12_11.4	11.4 A conversion should not be performed between a pointer to object and an integer type	Yes		Required	
MISRA12_11.5	11.5 A conversion should not be performed from pointer to void into pointer to object	Yes		Advisory	
MISRA12_11.8	11.8 A cast shall not remove any const or volatile qualification from the type	Yes		Required	

	pointed to by a pointer				
MISRA12_11.9	11.9 The macro NULL shall be the only permitted form of integer null pointer constant	Yes		Required	
MISRA12_12.3	12.3 The comma operator shall not be used.	Yes		Advisory	
MISRA12_13.1	13.1 Initializer lists shall not contain persistent side effects	Yes		Required	
MISRA12_13.4	13.4 The result of an assignment operator should not be used	Yes		Advisory	
MISRA12_13.5	13.5 The right hand operand of a logical && or || operator shall not contain persistent side effects	Yes		Required	
MISRA12_14.1	14.1 A loop counter shall not have essentially floating type	Yes		Required	
MISRA12_14.4	14.4 The controlling	Yes		Required	

	expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type				
MISRA12_21.2	21.2 Reserved Identifiers or Macros	Yes		Required	
MISRA12_DIR_4.8	Directive 4.8 If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden	Yes		Advisory	
MISRA23_0.0.1	0.0.1 A function shall not contain unreachable statements	Yes		Required	
MISRA23_0.0.2	0.0.2 Controlling expressions should not be invariant	No		Required	
MISRA23_0.1.2	0.1.2 The value returned by a function shall	Yes		Required	

	be used				
MISRA23_0.2.1	0.2.1 Variables with limited visibility should be used at least once	Yes		Advisory	
MISRA23_0.2.2	0.2.2 A named function parameter shall be used at least once	Yes		Required	
MISRA23_0.2.3	0.2.3 Types with limited visibility should be used at least once	Yes		Advisory	
MISRA23_0.2.4	0.2.4 Functions with limited visibility should be used at least once	Yes		Advisory	
MISRA23_2.3	2.3 A project should not contain unused type declarations	Yes		Advisory	
MISRA23_2.7	2.7 A function should not contain unused parameters	Yes		Advisory	
MISRA23_3.1	3.1 The character sequences /* and // shall	Yes		Required	

	not be used within a comment				
MISRA23_4.1	4.1 Octal and Hexadecimal Sequences	Yes		Required	
MISRA23_4.1.1	4.1.1 A program shall conform to ISO/IEC 14882:2017 (C++17)	No		Required	
MISRA23_4.1.2	4.1.2 Deprecated features should not be used	No		Advisory	
MISRA23_4.1.3	4.1.3 There shall be no occurrence of undefined or critical unspecified behaviour	No		Required	
MISRA23_5.0.1	5.0.1 Trigraph-like sequences should not be used	Yes		Required	
MISRA23_5.7.1	5.7.1 The character sequence /* shall not be used within a C-style comment	Yes		Required	
MISRA23_5.7.2	5.7.2 Sections of code should not be "commented out"	Yes		Advisory	

MISRA23_5.7.3	5.7.3 Line-splicing shall not be used in // comments	Yes		Required	
MISRA23_5.8	5.8 Identifiers that define objects or functions with external linkage shall be unique	Yes		Required	
MISRA23_5.9	5.9 Identifiers that define objects or functions with internal linkage should be unique	Yes		Advisory	
MISRA23_5.13.1	5.13.1 Within character literals and non raw-string literals, \ shall only be used to form a defined escape sequence or universal character name	Yes		Required	
MISRA23_5.13.2	5.13.2 Octal escape sequences, hexadecimal escape sequences and universal	Yes		Required	

	character names shall be terminated				
MISRA23_5.13.3	5.13.3 Octal constants shall not be used	Yes		Required	
MISRA23_5.13.4	5.13.4 Unsigned integer literals shall be appropriately suffixed	Yes		Required	
MISRA23_5.13.5	5.13.5 The lowercase form of L shall not be used as the first character in a literal suffix	Yes		Required	
MISRA23_5.13.6	5.13.6 An integer-literal of type long long shall not use a single L or l in any suffix	Yes		Required	
MISRA23_5.13.7	5.13.7 String literals with different encoding prefixes shall not be concatenated	Yes		Required	
MISRA23_6.0.1	6.0.1 Block scope declarations shall not be visually	Yes		Required	

	ambiguous				
MISRA23_6.0.2	6.0.2 When an array with external linkage is declared, its size should be explicitly specified	Yes		Advisory	
MISRA23_6.0.3	6.0.3 Global Namespace Declarations	Yes		Required	
MISRA23_6.0.4	6.0.4 The identifier main shall not be used for a function other than the global function main	Yes		Required	
MISRA23_6.1	6.1 Bit-fields shall only be declared with an appropriate type	Yes		Required	
MISRA23_6.2.4	6.2.4 A header file shall not contain definitions of functions or objects that are non-inline and have external linkage	Yes		Required	
MISRA23_6.3	6.3 A bit field shall not be declared as a member of a union	Yes		Required	

MISRA23_6.4.2	6.4.2 Derived classes shall not conceal functions that are inherited from their bases	Yes		Required	
MISRA23_6.5.1	6.5.1 A function or object with external linkage should be introduced in a header file	Yes		Advisory	
MISRA23_6.5.2	6.5.2 Internal linkage should be specified appropriately	Yes		Advisory	
MISRA23_6.7.1	6.7.1 Local variables shall not have static storage duration	Yes		Required	
MISRA23_6.7.2	6.7.2 Global variables shall not be used	Yes		Required	
MISRA23_6.8.1	6.8.1 An object shall not be accessed outside of its lifetime	No		Required	
MISRA23_6.8.2	6.8.2 A function must not return a reference or	Yes		Mandatory	

	a pointer to a local variable with automatic storage duration				
MISRA23_6.9.1	6.9.1 The same type aliases shall be used in all declarations of the same entity	Yes		Required	
MISRA23_6.9.2	6.9.2 The names of the standard signed integer types and standard unsigned integer types should not be used	Yes		Advisory	
MISRA23_7.0.1	7.0.1 There shall be no conversion from type bool	Yes		Required	
MISRA23_7.0.3	7.0.3 The numerical value of a character shall not be used	Yes		Required	
MISRA23_7.11.1	7.11.1 nullptr shall be the only form of the null-pointer-constant	Yes		Required	
MISRA23_7.11.2	7.11.2 Array to Pointer	Yes		Required	

	Decay				
MISRA23_8.1.1	8.1.1 A non-transient lambda shall not implicitly capture this	Yes		Required	
MISRA23_8.1.2	8.1.2 Variables should be captured explicitly in a non-transient lambda	Yes		Advisory	
MISRA23_8.2	8.2 Use Named Parameters and Prototype Form	Yes		Required	
MISRA23_8.2.1	8.2.1 A virtual base class shall only be cast to a derived class by means of dynamic_cast	Yes		Required	
MISRA23_8.2.3	8.2.3 A cast shall not remove any const or volatile qualification from the type accessed via a pointer or by reference	Yes		Required	
MISRA23_8.2.5	8.2.5 reinterpret_cast shall not be used	Yes		Required	
MISRA23_8.2	8.2.6 An	Yes		Required	

.6	object with integral, enumerated, or pointer to void type shall not be cast to a pointer type				
MISRA23_8.2.7	8.2.7 Pointer to Integer Cast	Yes		Advisory	
MISRA23_8.2.8	8.2.8 An object pointer type shall not be cast to an integral type other than std::uintptr_t or std::intptr_t	Yes		Required	
MISRA23_8.2.9	8.2.9 The operand to typeid shall not be an expression of polymorphic class type	Yes		Required	
MISRA23_8.2.10	8.2.10 Functions shall not call themselves, either directly or indirectly	Yes		Required	
MISRA23_8.3.1	8.3.1 The built-in unary - operator should not be applied to an expression of unsigned	Yes		Advisory	

	type				
MISRA23_8.3.2	8.3.2 The built-in unary + operator should not be used	Yes		Advisory	
MISRA23_8.7.2	8.7.2 Subtraction between pointers shall only be applied to pointers that address elements of the same array	Yes		Required	
MISRA23_8.9	8.9 An object should be declared at block scope if its identifier only appears in a single function	Yes		Advisory	
MISRA23_8.10	8.10 Non-static Inline Functions	Yes		Required	
MISRA23_8.14.1	8.14.1 The right-hand operand of a logical && or || operator should not contain persistent side effects	Yes		Advisory	
MISRA23_8.15	8.15 All declarations of an object with an explicit	Yes		Required	

	alignment specification shall specify the same alignment				
MISRA23_8.16	8.16 The alignment specification of zero should not appear in an object declaration	Yes		Advisory	
MISRA23_8.17	8.17 At most one explicit alignment specifier should appear in an object declaration	Yes		Advisory	
MISRA23_8.18.2	8.18.2 The result of an assignment operator should not be used	No		Advisory	
MISRA23_8.19.1	8.19.1 The comma operator shall not be used.	Yes		Advisory	
MISRA23_9.1	9.1 The value of an object with automatic storage duration shall not be read before it has been set	Yes		Mandatory	
MISRA23_9.2	9.2 The initializer for	Yes		Required	

	an aggregate or union shall be enclosed in braces				
MISRA23_9.3	9.3 Arrays shall not be partially initialized	Yes		Required	
MISRA23_9.4	9.4 An element of an object shall not be initialized more than once	Yes		Required	
MISRA23_9.4.1	9.4.1 All if ... else if constructs shall be terminated with an else statement	Yes		Required	
MISRA23_9.5	9.5 Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	Yes		Required	
MISRA23_9.5.2	9.5.2 A for-range-initializer shall contain at most one function call	Yes		Required	
MISRA23_9.6.1	9.6.1 The goto statement	Yes		Advisory	

	should not be used				
MISRA23_9.6.2	9.6.2 A goto statement shall reference a label in a surrounding block	Yes		Required	
MISRA23_9.6.3	9.6.3 The goto statement shall jump to a label declared later in the function body	Yes		Required	
MISRA23_9.6.4	9.6.4 A function declared with the [[noreturn]] attribute shall not return	Yes		Required	
MISRA23_9.6.5	9.6.5 A function with non-void return type shall return a value on all paths	Yes		Required	
MISRA23_9.7	9.7 Atomic objects shall be appropriately initialized before being accessed	Yes		Mandatory	
MISRA23_10.0.1	10.0.1 A declaration should not	Yes		Advisory	

	declare more than one variable or member variable				
MISRA23_10.1	10.1 Operands shall not be of an inappropriate essential type	Yes		Required	
MISRA23_10.1.1	10.1.1 The target type of a pointer or lvalue reference parameter should be const-qualified appropriately	Yes		Advisory	
MISRA23_10.1.2	10.1.2 The volatile qualifier shall be used appropriately	Yes		Required	
MISRA23_10.2.1	10.2.1 An enumeration shall be defined with an explicit underlying type	Yes		Required	
MISRA23_10.2.2	10.2.2 Unscoped enumerations should not be declared	Yes		Advisory	
MISRA23_10.3.1	10.3.1 There should be no unnamed	Yes		Advisory	

	namespaces in header files				
MISRA23_10.4	10.4 Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category	Yes		Required	
MISRA23_10.4.1	10.4.1 The asm declaration shall not be used	Yes		Required	
MISRA23_10.6	10.6 The value of a composite expression shall not be assigned to an object with wider essential type	Yes		Required	
MISRA23_10.8	10.8 The value of a composite expression shall not be cast to a different essential type category or a	Yes		Required	

	wider essential type				
MISRA23_11.1	11.1 Conversions shall not be performed between a pointer to a function and any other type	Yes		Required	
MISRA23_11.2	11.2 Conversions shall not be performed between a pointer to an incomplete type and any other type	Yes		Required	
MISRA23_11.3	11.3 A cast shall not be performed between a pointer to object type and a pointer to a different object type	Yes		Required	
MISRA23_11.3.1	11.3.1 Variables of array type should not be declared	Yes		Advisory	
MISRA23_11.3.2	11.3.2 The declaration of an object should contain no more than	Yes		Advisory	

	two levels of pointer indirection				
MISRA23_11.4	11.4 A conversion should not be performed between a pointer to object and an integer type	Yes		Required	
MISRA23_11.5	11.5 A conversion should not be performed from pointer to void into pointer to object	Yes		Advisory	
MISRA23_11.6.1	11.6.1 All variables should be initialized	Yes		Advisory	
MISRA23_11.6.3	11.6.3 Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique	Yes		Required	
MISRA23_11.8	11.8 A conversion shall not remove any const, volatile or _Atomic qualification	Yes		Required	

	from the type pointed to by a pointer				
MISRA23_11.9	11.9 The macro NULL shall be the only permitted form of integer null pointer constant	Yes		Required	
MISRA23_12.2.1	12.2.1 Bit-fields should not be declared	Yes		Advisory	
MISRA23_12.2.2	12.2.2 A bit-field shall have an appropriate type	Yes		Required	
MISRA23_12.2.3	12.2.3 A named bit-field with signed integer type shall not have a length of one bit	Yes		Required	
MISRA23_12.3	12.3 The comma operator shall not be used.	Yes		Advisory	
MISRA23_12.3.1	12.3.1 The union keyword shall not be used	Yes		Required	
MISRA23_13.1	13.1 Initializer lists shall not contain persistent side effects	Yes		Required	

MISRA23_13.1.1	13.1.1 Classes should not be inherited virtually	Yes		Advisory	
MISRA23_13.1.2	13.1.2 An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes		Required	
MISRA23_13.3.1	13.3.1 User-declared member functions shall use the virtual, override and final specifiers appropriately	Yes		Required	
MISRA23_13.3.2	13.3.2 Parameters in an overriding virtual function shall not specify different default arguments	Yes		Required	
MISRA23_13.3.3	13.3.3 The parameters in all declarations or overrides of a function shall either be unnamed or have identical	Yes		Required	

	names				
MISRA23_13.4	13.4 The result of an assignment operator should not be used	Yes		Advisory	
MISRA23_13.5	13.5 The right hand operand of a logical && or || operator shall not contain persistent side effects	Yes		Required	
MISRA23_14.1	14.1 A loop counter shall not have essentially floating type	Yes		Required	
MISRA23_14.1.1	14.1.1 Non-static data members should be either all private or all public	Yes		Advisory	
MISRA23_14.4	14.4 The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type	Yes		Required	
MISRA23_15.	15.0.2 User-	Yes		Advisory	

0.2	provided copy and move member functions of a class should have appropriate signatures				
MISRA23_15.1.1	15.1.1 An object's dynamic type shall not be used from within its constructor or destructor	Yes		Required	
MISRA23_15.1.3	15.1.3 Conversion operators and constructors that are callable with a single argument shall be explicit	Yes		Required	
MISRA23_15.1.5	15.1.5 A class shall only define an initializer-list constructor when it is the only constructor	Yes		Required	
MISRA23_16.5.1	16.5.1 The logical AND and logical OR operators shall not be overloaded	Yes		Required	

MISRA23_16.5.2	16.5.2 The address-of operator shall not be overloaded	Yes		Required	
MISRA23_16.6.1	16.6.1 Symmetrical operators should only be implemented as non-member functions	Yes		Advisory	
MISRA23_17.8.1	17.8.1 Function templates shall not be explicitly specialized	Yes		Required	
MISRA23_17.10	17.10 A function declared with a _Noreturn function specifier shall have void return type	Yes		Required	
MISRA23_17.12	17.12 A function identifier should only be used with either a preceding &, or with a parenthesized parameter list	Yes		Required	
MISRA23_17.13	17.13 A function type	Yes		Required	

	shall not be type qualified				
MISRA23_18.1.1	18.1.1 An exception object shall not have pointer type	Yes		Required	
MISRA23_18.1.2	18.1.2 An empty throw shall only occur within the compound-statement of a catch handler	Yes		Required	
MISRA23_18.3.1	18.3.1 There should be at least one exception handler to catch all otherwise unhandled exceptions	Yes		Advisory	
MISRA23_18.3.2	18.3.2 An exception of class type shall be caught by const reference or reference	Yes		Required	
MISRA23_18.3.3	18.3.3 Handlers for a function-try-block of a constructor or destructor shall not refer to non-static members	Yes		Required	

	from their class or its bases				
MISRA23_18.5.2	18.5.2 Program-terminating functions should not be used	Yes		Advisory	
MISRA23_19.0.1	19.0.1 A line whose first token is # shall be a valid preprocessing directive	Yes		Required	
MISRA23_19.0.2	19.0.2 Function-like macros shall not be defined	Yes		Required	
MISRA23_19.0.3	19.0.3 #include directives should only be preceded by preprocessor directives or comments	Yes		Advisory	
MISRA23_19.0.4	19.0.4 #undef should only be used for macros defined previously in the same file	Yes		Advisory	
MISRA23_19.1.1	19.1.1 The defined preprocessor operator shall	Yes		Required	

	be used appropriately				
MISRA23_19.1.2	19.1.2 All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related	Yes		Required	
MISRA23_19.2.1	19.2.1 Precautions shall be taken in order to prevent the contents of a header file being included more than once	Yes		Required	
MISRA23_19.2.2	19.2.2 The #include directive shall be followed by either a <filename> or "filename" sequence	Yes		Required	
MISRA23_19.2.3	19.2.3 The ' or " or \ characters and the /* or // character sequences	Yes		Required	

	shall not occur in a header file name				
MISRA23_19.3.1	19.3.1 The # and ## operators should not be used	Yes		Advisory	
MISRA23_19.3.2	19.3.2 A macro parameter immediately following a # operator shall not immediately be followed by a ## operator	Yes		Required	
MISRA23_19.3.3	19.3.3 The argument to a mixed-use macro parameter shall not be subject to further expansion	No		Required	
MISRA23_19.3.4	19.3.4 Parentheses shall be used to ensure macro arguments are expanded appropriately	Yes		Required	
MISRA23_19.3.5	19.3.5 Tokens that look like a preprocessing directive	Yes		Advisory	

	shall not occur within a macro argument				
MISRA23_19.6.1	19.6.1 The #pragma directive and the _Pragma operator should not be used	Yes		Advisory	
MISRA23_21.2	21.2 Reserved Identifiers or Macros	Yes		Required	
MISRA23_21.2.1	21.2.1 The library functions atof, atoi, atol and atoll from library <cstdlib> shall not be used	Yes		Required	
MISRA23_21.2.2	21.2.2 The string handling functions from <cstring>, <cstdlib>, <cwchar> and <cinttypes> shall not be used	Yes		Required	
MISRA23_21.2.3	21.2.3 The library function system from <cstdlib> shall not be	Yes		Required	

	used				
MISRA23_21.2.4	21.2.4 The macro offsetof shall not be used	Yes		Required	
MISRA23_21.6.1	21.6.1 Dynamic memory should not be used	Yes		Advisory	
MISRA23_21.6.2	21.6.2 Dynamic memory shall be managed automatically	Yes		Required	
MISRA23_21.6.4	21.6.4 If a project defines either a sized or unsized version of a global operator delete, then both shall be defined	Yes		Required	
MISRA23_21.6.5	21.6.5 A pointer to an incomplete class type shall not be deleted	Yes		Required	
MISRA23_21.8	21.8 The Standard Library termination functions of <stdlib.h> shall not be used	Yes		Required	
MISRA23_21.10.1	21.10.1 The features of	Yes		Required	

	<cstdarg> shall not be used				
MISRA23_21.10.2	21.10.2 The standard header file <csetjmp> shall not be used	Yes		Required	
MISRA23_21.17	21.17 Use of the string handling functions from <string.h> shall not result in accesses beyond the bounds of the objects referenced by their pointer parameters	Yes		Mandatory	
MISRA23_21.19	21.19 The pointers returned by the Standard Library functions localeconv, getenv, setlocale or, strerror shall only be used as if they have pointer to const-qualified type	Yes		Mandatory	
MISRA23_21.20	21.20 The pointer	Yes		Mandatory	

	returned by the C++ Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror must not be used following a subsequent call to the same function				
MISRA23_21.20.3	21.20.3 The facilities provided by the standard header file <csignal> shall not be used	Yes		Required	
MISRA23_21.24	21.24 The random number generator functions of <stdlib.h> shall not be used	Yes		Required	
MISRA23_21.26	21.26 The Standard Library function mtx_timedlock() shall only be invoked	Yes		Required	

	on mutex objects of appropriate mutex type				
MISRA23_22.3.1	22.3.1 The assert macro shall not be used with a constant-expression	Yes		Required	
MISRA23_22.4.1	22.4.1 The literal value zero shall be the only value assigned to errno	Yes		Required	
MISRA23_22.11	22.11 A thread that was previously either joined or detached shall not be subsequently joined nor detached	Yes		Required	
MISRA23_22.13	22.13 Thread objects, thread synchronization objects and thread-specific storage pointers shall have appropriate storage duration	Yes		Required	
MISRA23_22.17	22.17 No thread shall	Yes		Required	

	unlock a mutex or call cnd_wait() or cnd_timedwait() for a mutex it has not locked before				
MISRA23_24.5.1	24.5.1 The character handling functions from <cctype> and <cwctype> shall not be used	Yes		Required	
MISRA23_24.5.2	24.5.2 The C++ Standard Library functions memcpy, memmove and memcmp from <cstring> shall not be used	Yes		Required	
MISRA23_25.5.1	25.5.1 The setlocale and std::locale::global functions shall not be called	Yes		Required	
MISRA23_25.5.2	25.5.2 The pointers returned by the C++ Standard	Yes		Mandatory	

	Library functions localeconv, getenv, setlocale or strerror must only be used as if they have pointer to const-qualified type				
MISRA23_25.5.3	25.5.3 The pointer returned by the C++ Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror must not be used following a subsequent call to the same function	Yes		Mandatory	
MISRA23_26.3.1	26.3.1 std::vector should not be specialized with bool	Yes		Advisory	
MISRA23_28.6.1	28.6.1 The argument to std::move shall be a non-const	Yes		Required	

	lvalue				
MISRA23_30.0.1	30.0.1 The C Library input/output functions shall not be used	Yes		Required	
MISRA23_30.0.2	30.0.2 Reads and writes on the same file stream shall be separated by a positioning operation	Yes		Required	
MISRA23_DIR_4.8	Directive 4.8 If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden	Yes		Advisory	
MISRA23_DIR_5.3	Directive 5.3 There shall be no dynamic thread creation	Yes		Required	
MSC30-C	Do not use the rand() function for generating pseudorandom numbers	Yes			Medium
MSC37-C	Ensure that control never	Yes			High

	reaches the end of a non-void function				
MSC41-C	Never hard code sensitive information	No			High
MSC50-CPP	Do not use the rand() function for generating pseudorandom numbers	Yes			Medium
MSC51-CPP	Ensure your random number generator is properly seeded	Yes			Medium
MSC52-CPP	Value-returning functions must return a value from all exit paths	Yes			Medium
MSC53-CPP	Do not return from a function declared [[noreturn]]	Yes			Medium
MSC54-CPP	A signal handler must be a plain old function	Yes			High
OOP50-CPP	Do not invoke virtual functions from constructors or destructors	Yes			Low
OOP51-CPP	Do not slice	Yes			Low

	derived objects				
OOP52-CPP	Do not delete a polymorphic object without a virtual destructor	Yes			Low
OOP53-CPP	Write constructor member initializers in the canonical order	Yes			Medium
OOP54-CPP	Gracefully handle self-copy assignment	Yes			Low
OOP55-CPP	Do not use pointer-to-member operators to access nonexistent members	No			High
OOP56-CPP	Honor replacement handler requirements	Yes			
OOP57-CPP	Prefer special member functions and overloaded operators to C Standard Library functions	Yes			High
OOP58-CPP	Copy operations must not mutate the	Yes			Low

	source object				
POS44-C	Do not use signals to terminate threads	Yes			Low
POS47-C	Do not use threads that can be canceled asynchronously	Yes			Medium
POS48-C	Do not unlock or destroy another POSIX thread's mutex	Yes			Medium
POS49-C	When data must be accessed by multiple threads, provide a mutex and guarantee no adjacent data is also accessed	No			Medium
POS50-C	Declare objects shared between POSIX threads with appropriate storage durations	Yes			Medium
POS51-C	Avoid deadlock with POSIX threads by	Yes			Low

	locking in predefined order				
POWER_OF_TEN_01	1 Simple Control Flow	Yes			
POWER_OF_TEN_02	2 Loops with Fixed Limits	Yes			
POWER_OF_TEN_03	3 No Dynamic Memory Allocation	Yes			
POWER_OF_TEN_04	4 Short Functions	Yes			
POWER_OF_TEN_05	5 Use Assertion Statements	Yes			
POWER_OF_TEN_06	6 Declarations at Lowest Scope	Yes			
POWER_OF_TEN_07_A	7A Check Parameters and Return Values - Ignored Return Values	Yes			
POWER_OF_TEN_07_B	7B Check Parameters and Return Values - Unchecked Parameters and Return Values	Yes			
POWER_OF_TEN_08	8 Limit Preprocessor Usage	Yes			
POWER_OF_TEN_09_A	9A Restrict Pointer Usage - Multiple	Yes			

	Dereferences				
POWER_OF_TEN_09_B	9B Restrict Pointer Usage - Other	Yes			
POWER_OF_TEN_10	10 All Compiler Warnings	Yes			
PRE30-C	Do not create a universal character name through concatenation	Yes			Low
RECOMMENDED_00	Commented Out Code	Yes			
RECOMMENDED_01	Definitions in Header Files	Yes			
RECOMMENDED_02	Files too long	Yes			
RECOMMENDED_03	Floating Equality Test	Yes			
RECOMMENDED_04	Functions Too Long	Yes			
RECOMMENDED_05	Functions shall not be declared implicitly	Yes			
RECOMMENDED_06	Goto Statements	Yes			
RECOMMENDED_07	Macros shall not be #define'd or #undef'd within a block	Yes			
RECOMMENDED_08	Magic Numbers	Yes			
RECOMMENDED_09	Nested Comments	Yes			

RECOMMENDED_10	Overly Complex Functions	Yes			
RECOMMENDED_11	Trigraphs shall not be used	Yes			
RECOMMENDED_12	Unreachable Code	Yes			
RECOMMENDED_13	Unused Functions	Yes			
RECOMMENDED_14	Unused C and C++ Local Variables	Yes			
RECOMMENDED_15	Unused Static Globals	Yes			
RECOMMENDED_16	Variables should be commented	Yes			
RECOMMENDED_17	Upper limit shall not be modified within the bounds of the loop	Yes			
RECOMMENDED_19	Comments Indicating Future Fixes	Yes			
RECOMMENDED_20	Duplicate Code	Yes			
SIG35-C	Do not return from a computational exception signal handler	No			Low
STI_FRIENDS	Unnecessary Friends	Yes		Recommended	High
STI_SPECIAL_MEMBER_F	Special Member	Yes		Recommended	High

UNCTIONS	Functions				
STI_UNUSED	Unused Entities	Yes		Recommended	High
STR34-C	Cast characters to unsigned char before converting to larger integer sizes	No			Medium
STR50-CPP	Guarantee that storage for strings has sufficient space for character data and the null terminator	Yes			High
STR51-CPP	Do not attempt to create a std::string from a null pointer	Yes			High
STR52-CPP	Use valid references, pointers, and iterators to reference elements of a basic_string	Yes			High
STR53-CPP	Range check element access	Yes			High
WIN30-C	Properly pair allocation and deallocation functions	Yes			Low