HIC++v4 Standards Model Summary for C++

The LDRA tool suite® is developed and certified to BS EN ISO 9001:2000 and SGS-TÜV Saar.

This information is applicable to version 9.7.1 of the LDRA tool suite®.
It is correct as of 25th September 2017.

Compliance is measured against
"High Integrity C++ Coding Standard, Version 4.0"
3 October 2013
Copyright © The Programming Research Group

Further information is available at http://www.codingstandard.com/HICPPCM/index.html

Classification	Enhanced Enforcement	Fully Implemented	Partially Implemented	Not yet Implemented	Not statically Checkable	Total
Rule	11	75	30	4	35	**155**
Total	**11**	**75**	**30**	**4**	**35**	**155**

HIC++v4 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
1.1.1	Rule	Ensure that code complies with the 2011 ISO C++ Language Standard	22 S	Use of obsolete language feature (use = -).
			113 S	Non standard character in source.
			293 S	Non ANSI/ISO construct used.
			387 S	Enum init not integer-constant-expression.
			502 S	Delete contains number of elements.
			646 S	Struct initialisation has too many items.
1.2.1	Rule	Ensure that all statements are reachable	139 S	Construct leads to infeasible code.
			140 S	Infeasible loop condition found.
1.2.2	Rule	Ensure that no expression or sub-expression is redundant	65 D	Void function has no side effects.
			105 D	DU anomaly dead code, var value is unused on all paths.
			57 S	Statement with no side effect.
1.3.1	Rule	Do not use the increment operator (++) on a variable of type bool	389 S	Bool value incremented/decremented.
1.3.2	Rule	Do not use the register keyword	84 S	Register variable declared.
1.3.3	Rule	Do not use the C Standard Library .h headers	130 S	Included file is not permitted.
1.3.4	Rule	Do not use deprecated STL library features	44 S	Use of banned function, type or variable.
1.3.5	Rule	Do not use throw exception specifications	281 S	Throw keyword found.
2.1.1	Rule	Do not use tab characters in source files	187 S	Tab character in source.
2.2.1	Rule	Do not use digraphs or trigraphs	81 S	Use of trigraph.
			295 S	Use of digraph.
2.3.1	Rule	Do not use the C comment delimiters /* .. */	207 S	Use of old style /* comments in C++.
2.3.2	Rule	Do not comment out code	302 S	Comment possibly contains code.
2.4.1	Rule	Ensure that each identifier is distinct from any other visible identifier	217 S	Names only differ by case.
			67 X	Identifier is typographically ambiguous.
2.5.1	Rule	Do not concatenate strings with different encoding prefixes	450 S	Wide string and string concatenated.
2.5.2	Rule	Do not use octal constants (other than zero)	83 S	Octal number found.
2.5.3	Rule	Use nullptr for the null pointer constant	348 S	Use of the NULL macro.
			530 S	NULL used in integer context.
			531 S	Literal zero used in pointer context.
			18 D	Identifier name reused.

HIC++v4 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
3.1.1	Rule	Do not hide declarations	92 S	Duplicate use of a name in an enumeration.
			128 S	Parameter has same name as global variable.
			131 S	Name reused in inner scope.
			358 S	Class member name reused.
3.2.1	Rule	Do not declare functions at block scope	296 S	Function declared at block scope.
3.3.1	Rule	Do not use variables with static storage duration	10 D	Globals used inside procedure.
			95 D	Named global function pointer used in procedure.
			100 D	Named static member object used in procedure.
			38 S	Use of static class member.
3.4.1	Rule	Do not return a reference or a pointer to an automatic variable defined within the function	42 D	Local pointer returned in function result.
			77 D	Local structure returned in function result.
			564 S	Reference assignment to wider scope.
3.4.2	Rule	Do not assign the address of a variable to a pointer with a greater lifetime	71 S	Pointer assignment to wider scope.
			564 S	Reference assignment to wider scope.
3.4.3	Rule	Use RAII for resources	503 S	Function returns local resources.
3.5.1	Rule	Do not make any assumptions about the internal representation of a value or object	74 S	Union declared.
			345 S	Bit operator with floating point operand.
			437 S	< > <= >= used on different object pointers.
			554 S	Cast to an unrelated type.
4.1.1	Rule	Ensure that a function argument does not undergo an array-to-pointer conversion	459 S	Array passed as actual parameter.
4.2.1	Rule	Ensure that the U suffix is applied to a literal used in a context requiring an unsigned integral	331 S	Literal value requires a U suffix.
			434 S	Signed/unsigned conversion without cast.
4.2.2	Rule	Ensure that data loss does not demonstrably occur in an integral expression	51 S	Shifting value too far.
			96 S	Use of mixed mode arithmetic.
			107 S	Type mismatch in ternary expression.
			114 S	Expression is not Boolean.
			403 S	Negative (or potentially negative) shift.
			433 S	Type conversion without cast.
			434 S	Signed/unsigned conversion without cast.
			446 S	Narrower int conversion without cast.
			488 S	Value outside range of underlying type.

HIC++v4 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
4.3.1	Rule	Do not convert an expression of wider floating point type to a narrower floating point type	425 S	float literal with no F suffix.
			445 S	Narrower float conversion without cast.
4.4.1	Rule	Do not convert floating values to integral types except through use of standard library functions	435 S	Float/integer conversion without cast.
5.1.1	Rule	Use symbolic names instead of literal values in code	58 D	Character or string literal re-used.
			201 S	Use of numeric literal in expression.
			604 S	Use of numeric literal as array bound/subscript.
5.1.2	Rule	Do not rely on the sequence of evaluation within an expression	35 D	Expression has side effects.
			72 D	Potential side effect problem in expression.
			1 Q	Call has execution order dependant side effects.
			9 S	Assignment operation in expression.
			30 S	Deprecated usage of ++ or -- operators found.
			53 S	Use of comma operator.
			132 S	Assignment operator in boolean expression.
			134 S	Volatile variable in complex expression.
5.1.3	Rule	Use parentheses in expressions to specify the intent of the expression	49 S	Logical conjunctions need brackets.
			361 S	Expression needs brackets.
5.1.4	Rule	Do not capture variables implicitly in a lambda		
5.1.5	Rule	Include a (possibly empty) parameter list in every lambda expression	657 S	Lambda parameter list omitted.
5.1.6	Rule	Do not code side effects into the right-hand operands of: &&, ||, sizeof, typeid or a function passed to condition variable::wait	35 D	Expression has side effects.
			1 Q	Call has execution order dependant side effects.
			54 S	Sizeof operator with side effects.
			133 S	Assignment operator in RHS of && or ||.
			406 S	Use of ++ or -- on RHS of && or || operator.
			408 S	Volatile variable accessed on RHS of && or ||.
5.2.1	Rule	Ensure that pointer or array access is demonstrably within bounds of a valid object	45 D	Pointer not checked for null before use.
			47 S	Array bound exceeded.
			436 S	Declaration does not specify an array.
			567 S	Pointer arithmetic is not on array.
			64 X	Array bound exceeded at call.
			68 X	Parameter indexing array too big at call.

HIC++v4 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
			69 X	Global array bound exceeded at use.
			72 X	Parameter indexing array too small at call.
5.2.2	Rule	Ensure that functions do not call themselves, either directly or indirectly	6 D	Recursion in procedure calls found.
			1 U	Inter-file recursion found.
5.3.1	Rule	Do not apply unary minus to operands of unsigned type	52 S	Unsigned expression negated.
5.3.2	Rule	Allocate memory using new and release it using delete	44 S	Use of banned function, type or variable.
			651 S	delete operation on object created with malloc.
			668 S	free called on object created with new.
5.3.3	Rule	Ensure that the form of delete matches the form of new used to allocate the memory	485 S	Array deletion without [].
5.4.1	Rule	Only use casting forms: static cast (excl void*), dynamic cast or explicit constructor call	203 S	Cast on a constant value.
			241 S	Use of reinterpret_cast.
			242 S	Use of const_cast.
			306 S	Use of C type cast.
			344 S	Cast on volatile value.
			440 S	Cast from integral type to pointer.
5.4.2	Rule	Do not cast an expression to an enumeration type	411 S	Inappropriate value assigned to enum.
5.4.3	Rule	Do not convert from a base class to a derived class	448 S	Base class pointer cast to derived class.
5.5.1	Rule	Ensure that the right hand operand of the division or remainder operators is demonstrably non-zero	43 D	Divide by zero found.
			127 D	Local or member denominator not checked before use.
			131 D	Global denominator not checked within this procedure.
			137 D	Parameter used as denominator not checked before use.
			248 S	Divide by zero in preprocessor directive.
			629 S	Divide by zero found.
			80 X	Divide by zero found.
5.6.1	Rule	Do not use bitwise operators with signed operands	50 S	Use of shift operator on signed type.
			120 S	Use of bit operator on signed type.

HIC++v4 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
			136 S	Bit operator with boolean operand.
5.7.1	Rule	Do not write code that expects floating point calculations to yield exact results	56 S	Equality comparison of floating point.
5.7.2	Rule	Ensure that a pointer to member that is a virtual function is only compared (==) with nullptr		
5.8.1	Rule	Do not use the conditional operator (?:) as a sub-expression	33 S	Use of ternary operator found.
6.1.1	Rule	Enclose the body of a selection or an iteration statement in a compound statement	11 S	No brackets to loop body (added by Testbed).
			12 S	No brackets to then/else (added by Testbed).
			428 S	No {} for switch (added by Testbed).
6.1.2	Rule	Explicitly cover all paths through multi-way selection statements	48 S	No default case in switch statement.
			59 S	Else alternative missing in if.
			278 S	Switch has missing or extra cases.
6.1.3	Rule	Ensure that a non-empty case statement block does not fall through to the next label	62 S	Switch case not terminated with break.
6.1.4	Rule	Ensure that a switch statement has at least two case labels, distinct from the default label	60 S	Empty switch statement.
			61 S	Switch contains default only.
			314 S	Switch has only 1 case and default.
6.2.1	Rule	Implement a loop that only uses element values as a range-based loop		
6.2.2	Rule	Ensure that a loop has a single loop counter, an optional control variable, and is not degenerate	28 D	Potentially infinite loop found.
			38 D	More than one control variable for loop.
			66 D	Non boolean control variable modified in loop.
			542 S	Extra loop control variable is not bool.
6.2.3	Rule	Do not alter a control or counter variable more than once in a loop	55 D	Modification of loop counter in loop body.
6.2.4	Rule	Only modify a for loop counter in the for expression	55 D	Modification of loop counter in loop body.
6.3.1	Rule	Ensure that the label(s) for a jump statement or a switch condition appear later, in the same or an enclosing block	245 S	Case statement in nested block.
			509 S	goto label is backwards.
			511 S	Jump into nested block.
		Ensure that execution of a function with a non-void	2 D	Function does not return a value on all paths.

HIC++v4 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
6.3.2	Rule	Ensure that execution of a function with a non-void return type ends in a return statement with a value	36 S	Function has no return statement.
			66 S	Function with empty return expression.
6.4.1	Rule	Postpone variable definitions as long as possible	25 D	Scope of variable could be reduced.
			40 S	Loop index is not declared locally.
			505 S	Control variable not declared in for loop.
			560 S	Scope of variable could be reduced.
7.1.1	Rule	Declare each identifier on a separate line in a separate declaration	177 S	Identifier not declared on new line.
			178 S	Declaration statement not on new line.
			422 S	More than one synonym in typedef.
			579 S	More than one variable per declaration.
7.1.2	Rule	Use const whenever possible	59 D	Parameter should be declared const.
			150 S	Volatile or const used in function type.
			603 S	Parameter should be declared * const.
			119 D	Pointer param should be declared const pointer.
			120 D	Pointer param should be declared pointer to const.
7.1.3	Rule	Do not place type specifiers before non-type specifiers in a declaration		
7.1.4	Rule	Place CV-qualifiers on the right hand side of the type they apply to		
7.1.5	Rule	Do not inline large functions	353 S	Inline member has more than *** statements.
7.1.6	Rule	Use class types or typedefs to abstract scalar quantities and standard integer types	90 S	Basic type declaration used.
			329 S	Operation not appropriate to plain char.
			495 S	Typedef name has no size indication.
7.1.7	Rule	Use a trailing return type in preference to type disambiguation using typename		
7.1.8	Rule	Use auto id = expr when declaring a variable to have the same type as its initializer function call		
7.1.9	Rule	Do not explicitly specify the return type of a lambda	658 S	Lambda trailing return type found.
7.1.10	Rule	Use static_assert for assertions involving compile time constants		

HIC++v4 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
7.2.1	Rule	Use an explicit enumeration base and ensure that it is large enough to store all enumerators		
7.2.2	Rule	Initialize none, the first only or all enumerators in an enumeration	85 S	Incomplete initialisation of enumerator.
7.3.1	Rule	Do not use using directives	513 S	Use of using directive.
7.4.1	Rule	Ensure that any objects, functions or types to be used from a single translation unit are defined in an unnamed namespace in the main source file	311 S	Non local declaration not in a namespace.
7.4.2	Rule	Ensure that an inline function, a function template, or a type used from multiple translation units is	352 S	Declaration of type not in header file.
			460 S	Inline or template function not in header.
7.4.3	Rule	Ensure that an object or a function used from multiple translation units is declared in a single header file	26 D	Variable should be defined once in only one file.
			33 D	No real declaration for external variable.
			34 D	Procedure name re-used in different files.
			60 D	External object should be declared only once.
			106 D	No prototype for non-static function.
			172 S	Variable declared multiply.
			354 S	Extern declaration is not in header file.
7.5.1	Rule	Do not use the asm declaration	17 S	Code insert found.
			88 S	Procedure is not pure assembler.
8.1.1	Rule	Do not use multiple levels of pointer indirection	80 S	Pointer indirection exceeds 2 levels.
			356 S	Pointer to pointer declared.
8.2.1	Rule	Make parameter names absent or identical in all declarations	36 D	Prototype and definition name mismatch.
8.2.2	Rule	Do not declare functions with an excessive number of parameters	18 S	More than *** parameters in procedure.
8.2.3	Rule	Pass small objects with a trivial copy constructor by value	215 S	Struct or class called by value.
8.2.4	Rule	Do not pass std::unique_ptr by const reference	660 S	STL smart pointer std::unique_ptr passed by const ref.
8.3.1	Rule	Do not write functions with an excessive McCabe Cyclomatic Complexity	1 C	Cyclomatic complexity greater than ***.

HIC++v4 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
8.3.2	Rule	Do not write functions with a high static program path count	2 Q	LCSAJ density exceeds ***.
8.3.3	Rule	Do not use default arguments	359 S	Default parameter use.
8.3.4	Rule	Define =delete functions with parameters of type rvalue reference to const		
8.4.1	Rule	Do not access an invalid object or an object with indeterminate value	53 D	Attempt to use uninitialised pointer.
			69 D	UR anomaly, variable used before assignment.
			652 S	Object created by malloc used before initialisation.
8.4.2	Rule	Ensure that a braced aggregate initializer matches the layout of the aggregate object	105 S	Initialisation brace { } fault.
			397 S	Array initialisation has insufficient items.
			462 S	Struct initialisation has insufficient items.
9.1.1	Rule	Declare static any member function that does not require this. Alternatively, declare const any member function that does not modify the externally visible state of the object	46 D	Member function may be declared const.
9.1.2	Rule	Make default arguments the same or absent when overriding a virtual function	364 S	Inherited default parameter redefined.
9.1.3	Rule	Do not return non-const handles to class data from const member functions	392 S	Class data accessible thru non const member.
9.1.4	Rule	Do not write member functions which return non-const handles to data less accessible than the member function	671 S	Class data accessible thru non const handle.
9.1.5	Rule	Do not introduce virtual functions in a final class		
9.2.1	Rule	Declare bit-fields with an explicitly unsigned integral or enumeration type	346 S	Bit field is not unsigned integral or enum type.
10.1.1	Rule	Ensure that access to base class subobjects does not require explicit disambiguation	28 S	Duplicated Base Classes in a Derived class.
10.2.1	Rule	Use the override special identifier when overriding a virtual function	659 S	Overriding a virtual function without override specifier.
10.3.1	Rule	Ensure that a derived class has at most one base class which is not an interface class	283 S	Multiple direct inheritance found.
11.1.1	Rule	Declare all data members private	202 S	Class data is not explicitly private.

HIC++v4 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
11.2.1	Rule	Do not use friend declarations	212 S	Use of friend function in class.
			213 S	Use of friend class.
12.1.1	Rule	Do not declare implicit user defined conversions	393 S	Single parameter constructor not 'explicit'.
12.2.1	Rule	Declare virtual, private or protected the destructor of a type used as a base class	303 S	Virtual class members need virtual destructor.
12.3.1	Rule	Correctly declare overloads for operator new and delete	421 S	New or delete not explicitly static.
			423 S	Operator new and no operator delete.
			424 S	Operator new[] and no operator delete[].
12.4.1	Rule	Do not use the dynamic type of an object unless the object is fully constructed	92 D	C'tor/d'tor calls virtual function.
			467 S	Virtual member called in ctor/dtor.
			561 S	Use of dynamic type in c'tor/d'tor.
12.4.2	Rule	Ensure that a constructor initializes explicitly all base classes and non-static data members	319 S	Constructor has insufficient initialisers.
12.4.3	Rule	Do not specify both an NSDMI and a member initializer in a constructor for the same non static member		
12.4.4	Rule	Write members in an initialization list in the order in which they are declared	206 S	Class initialiser out of order.
12.4.5	Rule	Use delegating constructors to reduce code duplication		
12.5.1	Rule	Define explicitly =default or =delete implicit special member functions of concrete classes	230 S	No copy constructor defined for class.
			231 S	No assignment operator defined for class.
			232 S	No destructor defined for class.
			233 S	No copy constructor for class with pointers.
			234 S	No assignment operator for class with pointers.
			235 S	No destructor for class with pointers.
			236 S	New used in class without assignment op.
			239 S	New used in class without copy constructor.
			260 S	No default constructor declared for class.
			469 S	No copy constructor for complex destructor.
			470 S	No assignment operator for complex destrtor.

HIC++v4 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
12.5.2	Rule	Define special members =default if the behavior is equivalent		
12.5.3	Rule	Ensure that a user defined move/copy constructor only moves/copies base and member objects	529 S	Static member initialised/assigned in constructor.
12.5.4	Rule	Declare noexcept the move constructor and move assignment operator		
12.5.5	Rule	Correctly reset moved-from handles to resources in the move constructor		
12.5.6	Rule	Use an atomic, non-throwing swap operation to implement the copy and move assignment operators		
12.5.7	Rule	Declare assignment operators with the ref-qualifier &		
12.5.8	Rule	Make the copy assignment operator of an abstract class protected or define it =delete	522 S	Public assign operator in abstract class.
13.1.1	Rule	Ensure that all overloads of a function are visible from where it is called	262 S	Non virtual function redefined.
			518 S	Using declaration is straddled by declarations.
			601 S	Insufficient overridden members.
13.1.2	Rule	If a member of a set of callable functions includes a universal reference parameter, ensure that one appears in the same position for all other members		
13.2.1	Rule	Do not overload operators with special semantics	211 S	Overloaded &&, || or comma.
			508 S	Operator & overloaded.
13.2.2	Rule	Ensure that the return type of an overloaded binary operator matches the built-in counterparts		
13.2.3	Rule	Declare binary arithmetic and bitwise operators as non-members	569 S	Operator should be non class member.
13.2.4	Rule	When overloading the subscript operator (operator[]) implement both const and non-const versions		

HIC++v4 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
13.2.5	Rule	Implement a minimal set of operators and use them to implement all other related operators		
14.1.1	Rule	Use variadic templates rather than an ellipsis	41 S	Ellipsis used in procedure parameter list.
14.2.1	Rule	Declare template specializations in the same file as the primary template they specialize		
14.2.2	Rule	Do not explicitly specialize a function template that is overloaded with other templates	539 S	Specialised overloaded templated function.
14.2.3	Rule	Declare extern an explicitly instantiated template		
15.1.1	Rule	Only use instances of std::exception for exceptions		
15.2.1	Rule	Do not throw an exception from a destructor	453 S	Throw found in destructor.
15.3.1	Rule	Do not access non-static members from a catch handler of constructor/destructor function try block	549 S	Catch in c'tor/d'tor references nonstatic member.
15.3.2	Rule	Ensure that a program does not result in a call to std::terminate	527 S	No master exception handler.
16.1.1	Rule	Use the preprocessor only for implementing include guards, and including header files with include guards	200 S	Define used for numeric constant.
			243 S	Included file not protected with #define.
			255 S	Found #if, #ifdef, #else, #elif .
			272 S	Found #ifndef (ref. removed).
			273 S	Found #define.
			307 S	Use of #line, #error preprocessor directives.
			340 S	Use of function like macro.
16.1.2	Rule	Do not include a path specifier in filenames supplied in #include directives	4 Q	Included file has path.
			100 S	#include filename is non conformant.
			339 S	#include directive with illegal items.
16.1.3	Rule	Match the filename in a #include directive to the one on the filesystem	3 Q	Filename contains upper case letters.
16.1.4	Rule	Use <> brackets for system and standard library headers. Use quotes for all other headers	427 S	Filename in #include not in < > or " ".
			568 S	#include "filename" uses standard library name.
16.1.5	Rule	Include directly the minimum number of headers required for compilation		
17.1.1	Rule	Do not use std::vector<bool>	44 S	Use of banned function, type or variable.

HIC++v4 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
17.2.1	Rule	Wrap use of the C Standard Library	44 S	Use of banned function, type or variable.
			130 S	Included file is not permitted.
17.3.1	Rule	Do not use std::move on objects declared with const or const & type		
17.3.2	Rule	Use std::forward to forward universal references		
17.3.3	Rule	Do not subsequently use the argument to std::forward		
17.3.4	Rule	Do not create smart pointers of array type		
17.3.5	Rule	Do not create an rvalue reference of std::array		
17.4.1	Rule	Use const container calls when result is immediately converted to a const iterator		
17.4.2	Rule	Use API calls that construct objects in place		
17.5.1	Rule	Do not ignore the result of std::remove, std::remove_if or std::unique	91 D	Function return value potentially unused.
			382 S	(void) missing for discarded return value.
18.1.1	Rule	Do not use platform specific multi-threading facilities	130 S	Included file is not permitted.
18.2.1	Rule	Use high integrity::thread in place of std::thread	130 S	Included file is not permitted.
18.2.2	Rule	Synchronize access to data shared between threads using a single lock		
18.2.3	Rule	Do not share volatile data between threads		
18.2.4	Rule	Use std::call_once rather than the Double-Checked Locking pattern		
18.3.1	Rule	Within the scope of a lock, ensure that no static path results in a lock of the same mutex		
18.3.2	Rule	Ensure that order of nesting of locks in a project forms a DAG		
18.3.3	Rule	Do not use std::recursive_mutex	44 S	Use of banned function, type or variable.
18.3.4	Rule	Only use std::unique_lock when std::lock_guard cannot be used	44 S	Use of banned function, type or variable.
18.3.5	Rule	Do not access the members of std::mutex directly		
18.3.6	Rule	Do not use relaxed atomics	44 S	Use of banned function, type or variable.

HIC++v4 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
18.4.1	Rule	Do not use std::condition_variable_any on a std::mutex	44 S	Use of banned function, type or variable.

General Compliance Notes

Enhanced Enforcement: LDRA checks additional cases to those specified by the mapped rule for enhanced safety and security.
Fully Implemented: LDRA checks all statically checkable aspects of the mapped rule.
Partially Implemented: LDRA checks certain aspects of the rule.

The assessment of whether a rule is fully or partially implemented is based on whether the mapped LDRA standards cover all statically checkable aspects of the rule with a high level of coverage or only cover certain statically checkable aspects of the rule. If a rule is undecidable then this assessment is based on what it is deemed reasonable for a static analysis tool to check.