



MISRA C:2012: What is it and How to Continuously Enforce it

Jason Masters

www.programmingresearch.com



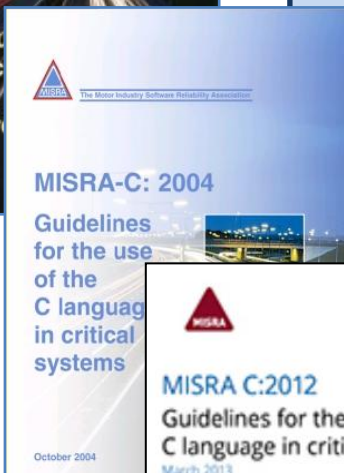
- Introduction to MISRA
- What's new in MISRA C:2012
- MISRA C Compliance
- Demonstration





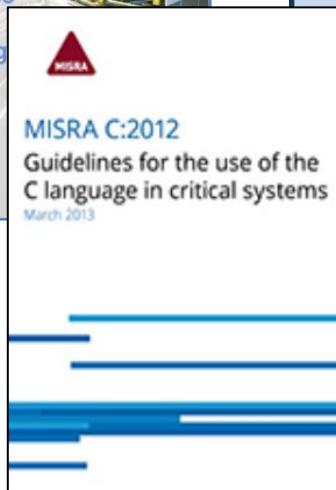
MISRA C:1998

- Derived from PRQA standards developed for Ford and Rover
- Developed in UK for automotive applications



MISRA C:2004

- Amended and extended
- Supplemented with an Exemplar suite



MISRA C:2012

- A committee of 10 representing many years of experience
- 4 years of effort
- Released 18th March 2013
- A further step forward ...

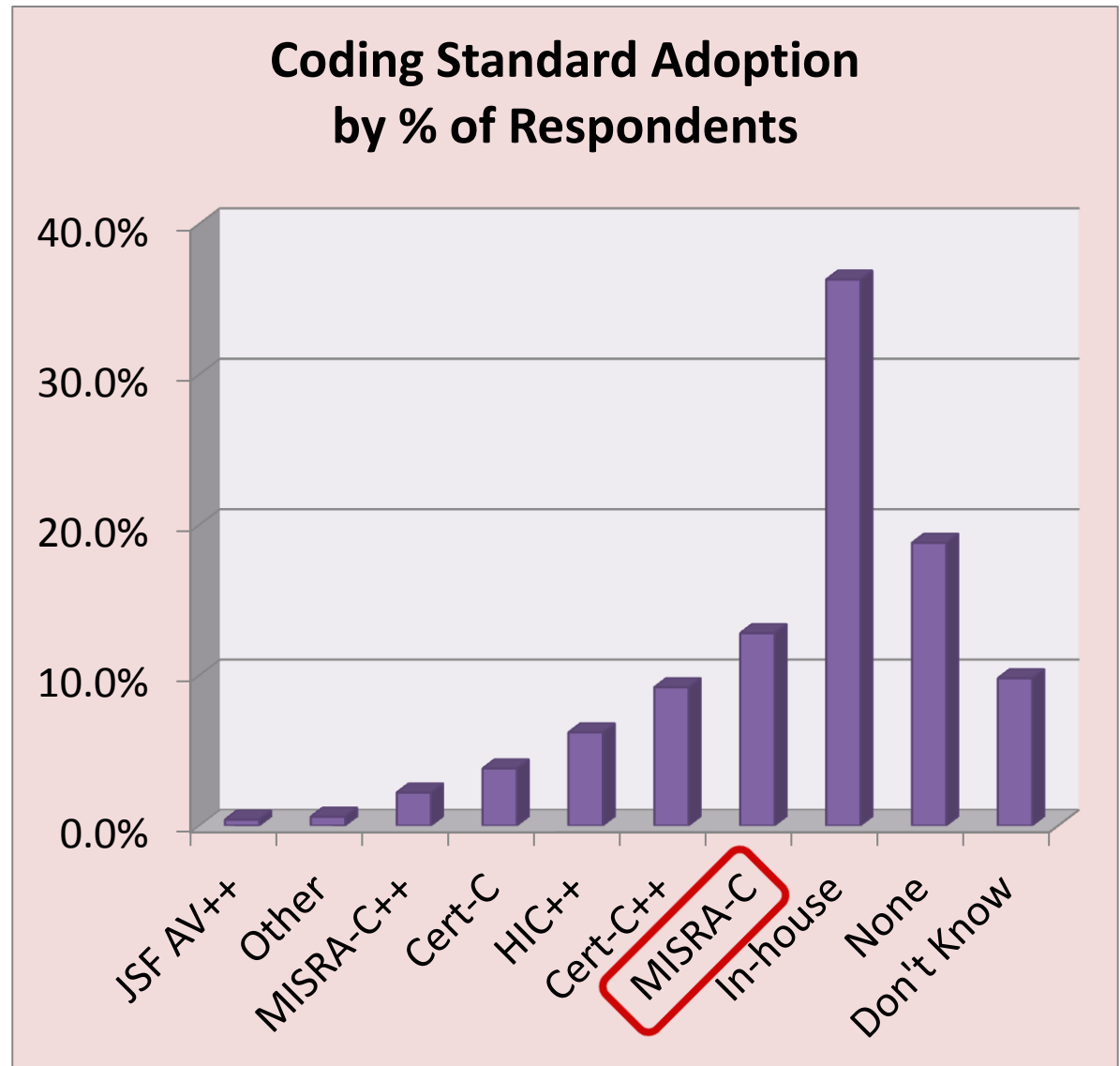
VDC Research White Paper* April 2011:

"Re-evaluation of Development and Testing Practices April 2011"

Available at:

www.programmingresearch.com/whitepapers

Data obtained from 600 respondents in Embedded and Enterprise/IT software and systems development.





MISRA C is now the most widely used coding standard for the C language - worldwide

- Automotive
- Aerospace
- Defence
- Medical
- Nuclear power
- Railways
- Consumer electronics
- Process control
- etc.





MISRA C:2012 compared to MISRA C:2004

- It's bigger
 - a few more rules (159 instead of 142)
 - and the content is better
- Many guidelines are unchanged
 - they may be reworded and better specified
 - guideline/rule numbering has changed
- Legacy code may not be compliant
 - there are new requirements – not many
 - but some restrictions have been removed





ISO:C90

- Well supported by compilers and tools
- The dangers are well understood
- Limitations – e.g. absence of Boolean type

MISRA C:1998
MISRA C:2004
MISRA C:2012

ISO:C99

- More features, e.g. `_Bool` and inline functions
- More dangers, e.g. additional undefined behaviour
- Most compilers do not support all features of C99

MISRA C:2012

ISO:C11

- Still relatively new
- Very limited tool support



Headline text clarification

An expanded explanation of the requirement

Why the guideline is necessary

More extensive code examples

Exceptions

Rule 8.8 The *static* storage class specifier shall be used in all declarations of objects and functions that have internal linkage

Category Required

Analysis Decidable, Single Translation Unit

Applies to C90, C99

Amplification

Since definitions are also declarations, this rule applies equally to definitions.

Rationale

The Standard states that if an object or function is declared with the *extern* storage class specifier and another declaration of the object or function is already visible, the linkage is that specified by the earlier declaration. This can be confusing because it might be expected that the *extern* storage class specifier creates external linkage. The *static* storage class specifier shall therefore be consistently applied to objects and functions with internal linkage.

Example

```
static int32_t x = 0;      /* definition: internal linkage */
extern int32_t x;          /* Non-compliant */

static int32_t f ( void ); /* declaration: internal linkage */
int32_t f ( void )        /* Non-compliant */
{
    return 1;
}

static int32_t g ( void ); /* declaration: internal linkage */
extern int32_t g ( void )  /* Non-compliant */
{
    return 1;
}
```




- Introduction to MISRA
- **What's new in MISRA C:2012**
- MISRA C Compliance
- Demonstration



Guideline Type

- Directive
- Rule

Category

- Advisory
- Required
- Mandatory

Rule 8.8 The *static* storage class specifier shall be used in all declarations of objects and functions that have internal linkage

Category Required

Analysis Decidable, Single Translation Unit

Applies to C90, C99

Amplification

Since definitions are also declarations, this rule applies equally to definitions.

Rationale

Language

- C90
- C99
- C90, C99

Analysis Scope

- Single Translation Unit
- System

Decidability

- Decidable
- Undecidable

```
static int32_t x = 0;      /* definition: internal linkage */
extern int32_t x;          /* Non-compliant */

static int32_t f ( void ); /* declaration: internal linkage */
int32_t f ( void )        /* Non-compliant */
{
    return 1;
}

static int32_t g ( void ); /* declaration: internal linkage */
extern int32_t g ( void )  /* Non-compliant */
{
    return 1;
}
```



There are now 2 types of guideline:

Rules

- Have well defined requirements
- Are statically enforceable (subject to certain limitations)

Directives

- May be loosely defined – allowing alternative interpretations
- May address "process" or "documentation" requirements





Rules

Rule 8.5 An external object or function shall be declared once in one and only one file

Rule 11.3 A cast shall not be performed between a pointer to object type and a different pointer to object type

Directives

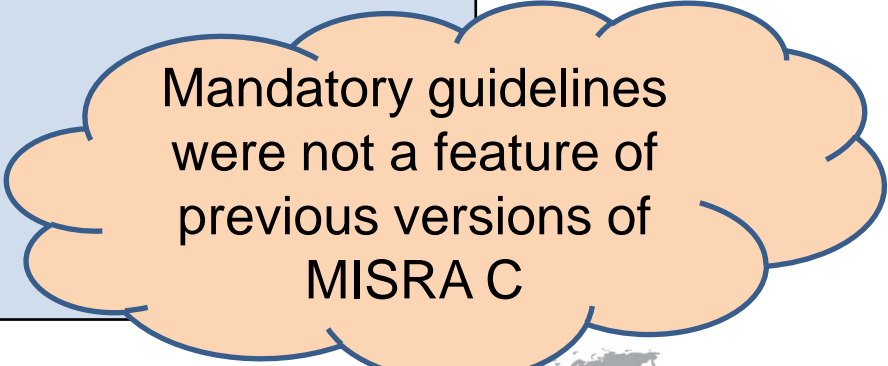
Dir 3.1 All code shall be traceable to documented requirements

Dir 4.3 Assembly language shall be encapsulated and isolated



There are now 3 guideline categories:

- **Advisory** guidelines
 - These are recommendations
 - Non-compliance is permitted at the user's discretion
 - Non-compliance should be documented
 - Formal deviations are not required
- **Required** guidelines
 - Non-compliance must be supported by a formal "deviation"
- **Mandatory** guidelines
 - Must always be obeyed
 - Compliance is always required



Mandatory guidelines
were not a feature of
previous versions of
MISRA C

CLASSIFICATION SUMMARY		DIRECTIVES (16)	RULES (143)
CATEGORY	Advisory	7	32
	Required	9	101
	Mandatory	0	10
LANGUAGE	C90	0	2
	C99	0	11
	C90 or C99	16	130
DECIDABILITY	Decidable	-	117
	Undecidable	-	26
ANALYSIS SCOPE	Single Translation Unit	-	104
	System	-	39

Some rules are now classified as "mandatory"

Some rules are only relevant for a specific language version

Some rules cannot be statically enforced with certainty

Rules that can be enforced within a single translation unit are decidable



- Introduction to MISRA
- What's new in MISRA C:2012
- **MISRA C Compliance**
- Demonstration





How do I claim Compliance?

- *Enforceability*
- *Deviations*



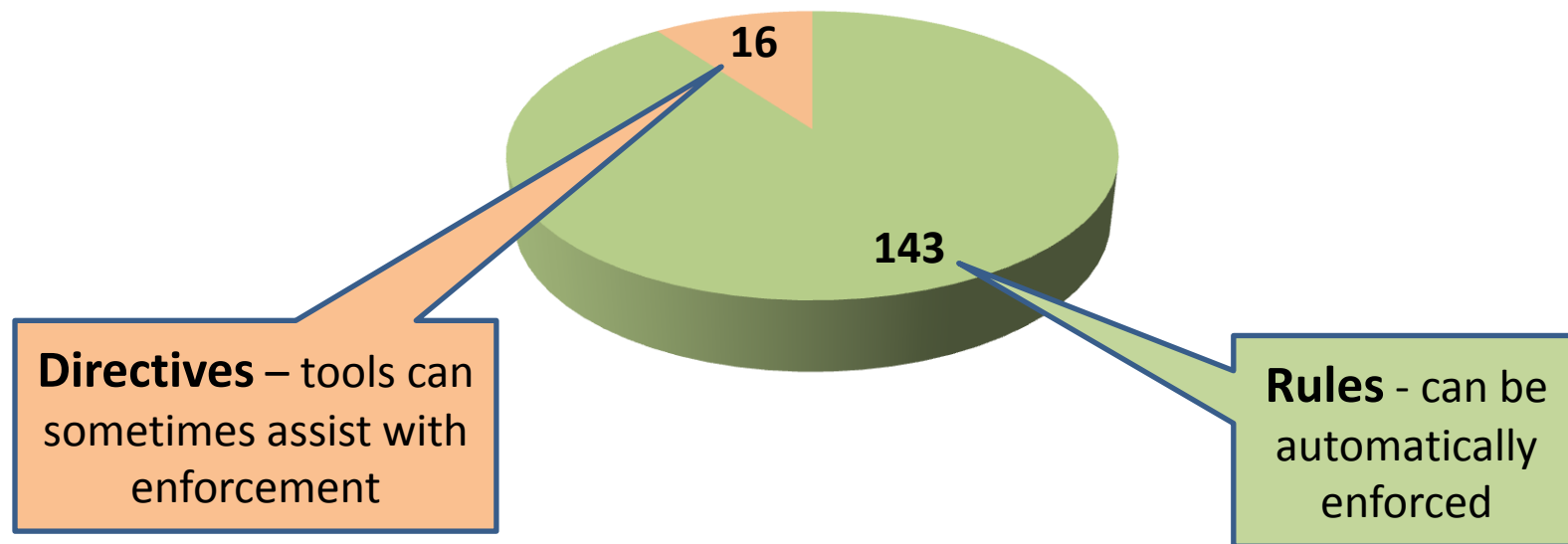
"The vision for the third edition of MISRA C is therefore to:

...

Increase the number of guidelines that can be processed by static analysis tools"

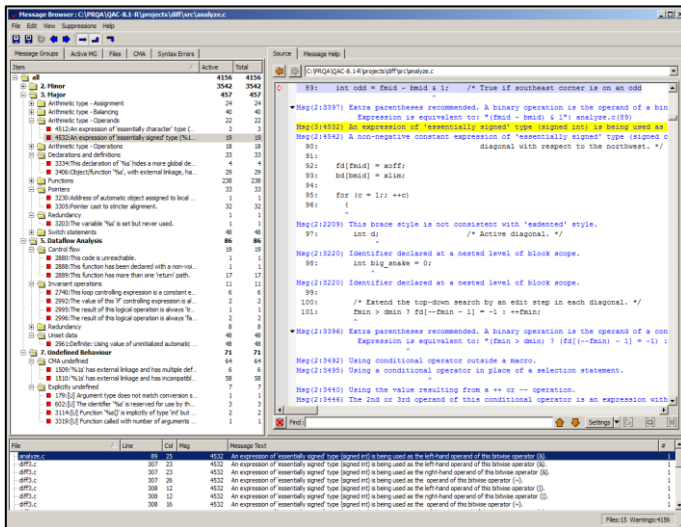
...

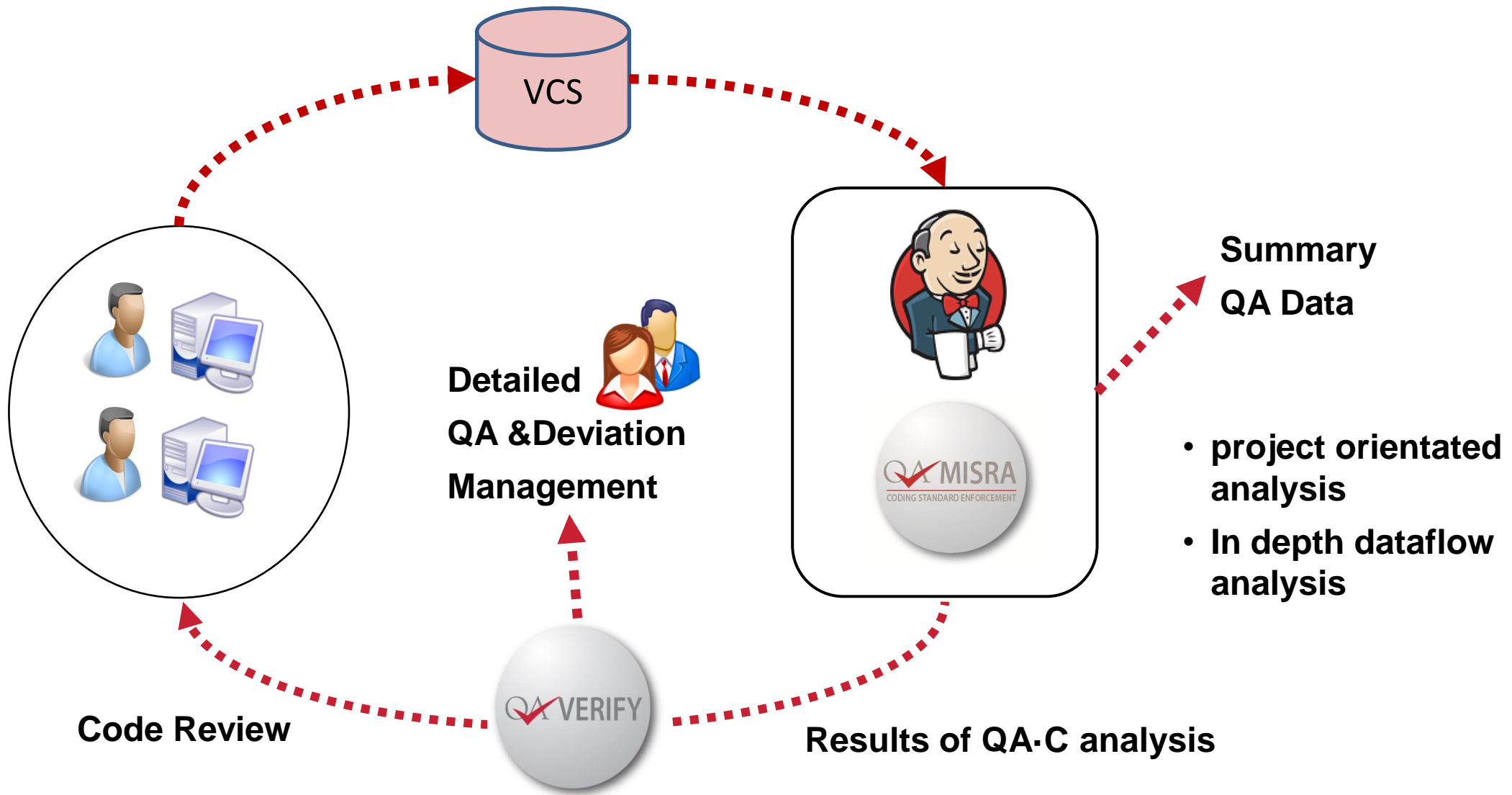
MISRA C:2012 Chapter 1 – The Vision





- ✓ syntactic & semantic analysis
- ✓ dataflow analysis
- ✓ automated code inspection
- ✓ certified ISO 26262 and IEC 61508
- ✓ precise diagnostic information
- ✓ solutions to implement fixes
- ✓ diagnostics, metrics and visualisations
- ✓ low false positives







What do we mean by "MISRA Compliant Code" ?

- *Enforceability*
- *Deviations*



"In order to use MISRA C, it is necessary to develop and document ...

- A deviation process by which justifiable non-compliances can be authorized and recorded "*

MISRA C:2012 Chapter 5.2.1 – Process activities required by MISRA C

" It is important that such deviations are properly recorded and authorized."

MISRA C:2012 Chapter 5.4 – Deviation procedure

Deviations are often necessary ...
... but the process can also be abused

MISRA C ADC: Approved deviation compliance for MISRA C:2004
ISBN 978-906400-09-5 (PDF), February 2013.
Freely downloadable from www.misra.org.uk





- It is (almost?) impossible to write a useful program that does not break any of the MISRA Guidelines.
- However, the code may still be MISRA Compliant
- Deviations allow the rules to be broken – but in a controlled and safe manner





- Program Performance
- Interfacing with third party code
- Development tools
- Legacy code
- Build configurations
- Hardware
- Defensive Programming
- Language features





- A good reason for breaking the rule
- Description of rule breaking extent
- Argument to support the reason
- Safety measures to be employed to ensure quality/integrity/reliability of the code



- Traceable entity deviation support
- Baselining
- Documentation and audit trail
- Collaborative code review
- Quality profiling



```

+ 116: static int
+ 117: connect_with_timeout (int fd, const struct sockaddr *addr, socklen_t addrlen,
- Msg 3334 This declaration of 'addr' hides a more global declaration.
- MISRA-C:2012 Rule 5.3
+ 118:         double timeout)
+ 119: {
+ 120:     struct cwt_context ctx;
+ 121:     ctx.fd = fd;
+ 122:     ctx.addr = addr;
+ 123:     ctx.addrlen = addrlen;
+ 124:
+ 125:     if (run_with_timeout (timeout, connect_
+ 126:     {
+ 127:         errno = ETIMEDOUT;
+ 128:         return -1;
+ 129:     }
+ 130:     if (ctx.result == -1 && errno == EINTR)
+ 131:         errno = ETIMEDOUT;
+ 132:     return ctx.result;
+ 133: }
+ 134:
+ 135: /* A kludge, but still better than passing the host name all the way
+ 136:     to connect_to_one. */
        
```

Suppress Diagnostic x

Message No: 3334
Line: 117
Column: 54
Deviation:
No Deviation Tag

(add new deviation)

☐ Carry Forward

Add New Deviation

Name:

Short Description

Long Description

Justification

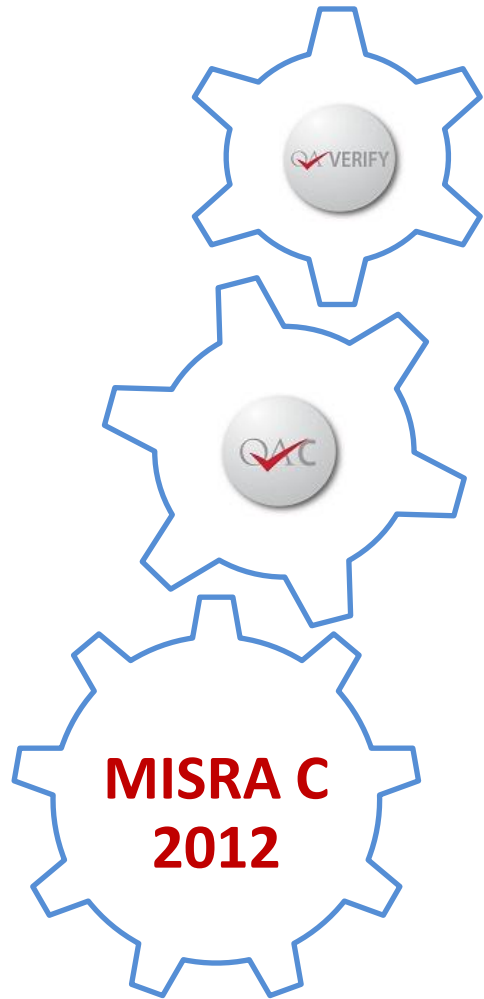
Safety Case

Import Deviations



- Introduction to MISRA
- What's new in MISRA C:2012
- MISRA C Compliance
- **Demonstration**





- *Disciplined compliance and deviation management*
- *Effective, accurate, automatic enforcement*
- *Enforceable, decidable, well-specified coding rules*



