MISRA C Evolution

Tokyo – 12th June 2023

Presented by
Andrew Banks
Technical Specialist, LDRA
Chair, MISRA C

Andrew.Banks@ldra.com

About the speaker

- Biography
  - Over 30 years experience in developing real-time embedded software systems, across a number of industries
  - Chartered Fellow of the British Computer Society
  - Member of the Institution of Engineering & Technology
    ... Member of the System Safety Technical Professional Network Executive
  - Technical Specialist / Field Application Engineer, LDRA
- Standards
  - Chairman of MISRA C Working Group since May 2013...
    ... Working Group member since 2007
  - Chairman of the British Standards Institute's Software Testing Working Group
    - Contributor to ISO/IEC JTC1/SC7 and WG26
    - Contributor to ISO 29119 "Software Testing"
  - Contributor to ISO 26262 "Road Vehicles - Functional Safety" 2nd Edition
  - etc

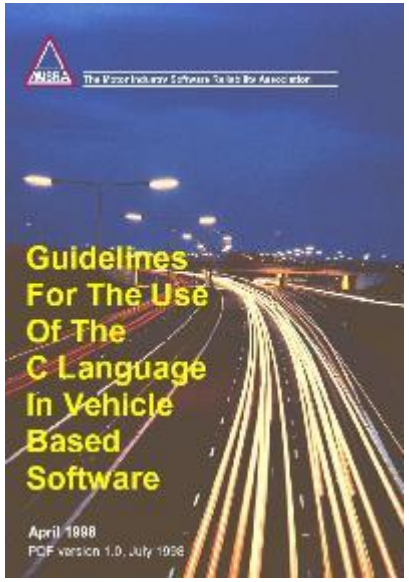
Andrew Banks
IEng MIET FBCS CITP

@AndrewBanks
AndrewBanks

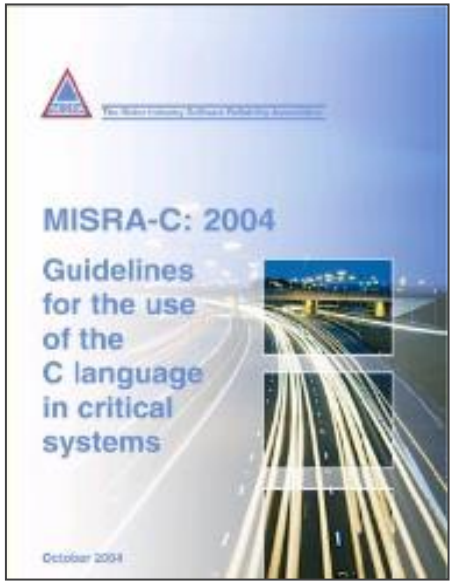
MISRA C Evolution

MISRA C Evolution – 1998 to 2019

MISRA-C:1998
1st Edition
(April 1998)



MISRA-C:2004
2nd Edition
(October 2004)



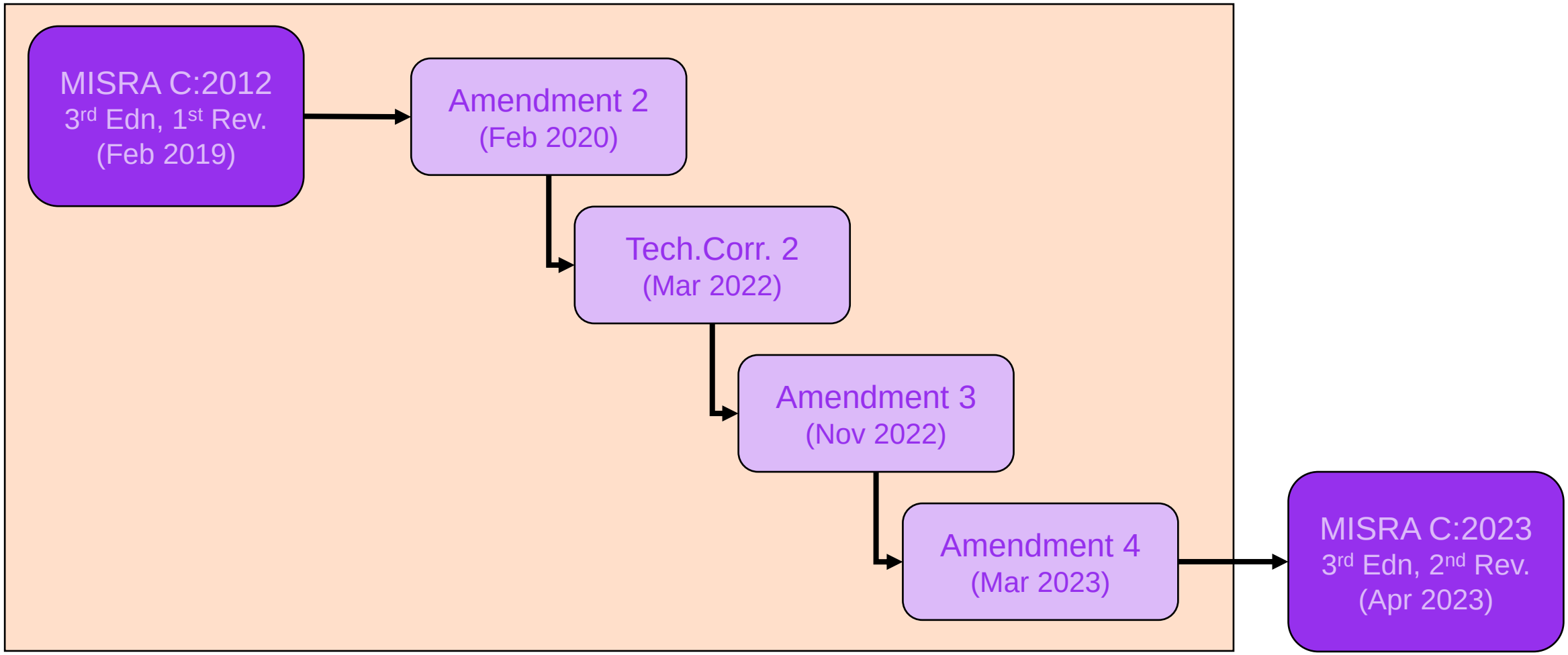
MISRA C:2012
3rd Edition
(March 2013)

Amendment 1
(April 2016)

Tech.Corr. 1
(June 2017)

MISRA C:2012
3rd Edn, 1st Rev.
(Feb 2019)

MISRA C Evolution



Happy 25th anniversary, MISRA C!

Introducing MISRA C:2023

Available to purchase as a PDF **now** via:
https://www.misra.org.uk/product/misra-c2023/

Coming soon… Print On Demand

Comparison

	D	R	
• MISRA C:2012	16	143	3rd Edition
• AMD1 (2016 – Security)	+1	+13	
• **MISRA C:2012 (2019)**	**17**	**156**	3rd Edition, 1st Revision
• AMD2 (2020 – C11)	0	+2	Mods to most other guidelines
• AMD3 (2022)	+1	+23	
• AMD4 (2023)	+3	+19	
• **MISRA C:2023**	**21**	**200**	3rd Edition, 2nd Revision

The MISRA C Supporting Cast

- Not being maintained
  - Addendum 1 Rule Mapping (MISRA C:2012 v MISRA C:2004)

- Coverage of MISRA C:2012 against…
  - Addendum 2 ISO/IEC TS 17961:2013 “C Secure” Published (*)
  - Addendum 3 CERT C 2016 Edition Published (*)
  - Addendum 4 ISO/IEC 24772 “Language Vulnerabilities” In progress
  - Addendum 5 MITRE Common Weakness Enumeration In progress

- MISRA Compliance
  - Permits Deviation permits for MISRA Compliance Published (*)

(*) = will be revised for MISRA C:2023

MISRA C:2023
New Guidelines

MISRA C:2023 – General　全般

- R.1.4　R　Emergent language features shall not be used
  新興の言語機能を使用してはならない
- R.1.5　R　Obsolescent language features shall not be used
  時代遅れの言語機能を使用してはならない

- R.21.21　R　The Standard Library function *system* shall not be used (previously part of R.21.8)
  標準ライブラリ関数*system*を使用してはならない
  （旧R.21.8の一部）

- R.21.24　R　The random number generator functions of <stdlib.h> shall not be used
  <stdlib.h>の乱数生成関数を使用してはならない

MISRA C:2023 – Objects and Initialization
オブジェクトと初期化

- R.2.8 A A project should not contain unused object definitions
  プロジェクトは使用されないオブジェクト定義を含むべきではない
- R.9.6 R An initializer using chained designators shall not contain initializers without designators
  ネストした指示付き初期化子による初期化には指示なし初期化を含んではならない

  Note: R.9.4 on designated initializers modified 指示付き初期化子に対するR.9.4は変更されました

- R.18.9 R An object with temporary lifetime shall not undergo array-to-pointer conversion
  一時的な生存期間をもつオブジェクトに配列からポインタへの変換を行ってはならない

- R.18.10 R Pointers to variably-modified array types shall not be used
  可変的変更配列型へのポインタを使用してはならない

  Note: existing R.18.8 modified to focus on just variable-length arrays, and exclude variably-modified arrays
  既存のR.18.8は可変長配列だけを対象とし、可変的変更配列を除外するよう変更されました

MISRA C:2023 – Functions and NoReturn
関数と NoReturn

- R.17.9 M A function declared with a _*Noreturn* function specifier shall not return to its caller
  _*Noreturn* 関数指定子付きで宣言された関数を呼出し元にリターンしてはならない
- R.17.10 R A function declared with a _*Noreturn* function specifier shall have void return type
  _*Noreturn* 関数指定子付きで宣言された関数は void 返却型を持たねばならない
- R.17.11 A A function that never returns should be declared with a _*Noreturn* function specifier
  決してリターンしない関数は _*Noreturn* 関数指定子付きで宣言されるべきである
- R.17.12 A A function identifier should only be used with either a preceding &, or with a parenthesized parameter list
  関数識別子は & を前につけるか、丸括弧で囲まれたパラメーターリストをつけるか、いずれかだけで使用されるべきである
- R.17.13 R A function type shall not be type qualified 関数型を型修飾してはならない

MISRA C:2023 – Alignment　アライメント

- R.8.15　R　All declarations of an object with an explicit alignment specification shall specify the same alignment
  アライメント指定が明示されたオブジェクトのすべての宣言は、同じアライメントを指定しなければならない
- R.8.16　A　The alignment specification of zero should not appear in an object declaration
  オブジェクトの宣言で 0 のアライメント指定を行うべきではない
- R.8.17　A　At most one explicit alignment specifier should appear in an object declaration
  オブジェクトの宣言では高々1つの明示的なアライメント指定を行うべきである

MISRA C:2023 – Integer Constants　整数定数

- R.7.5　M　The argument of an integer constant macro shall have an appropriate form
  整数定数マクロの引数は適切な形式でなければならない
- R.7.6　R　The small integer variants of the minimum-width integer constant macros shall not be used
  最小幅整数の定数マクロの小桁整数バリアントを使用してはならない

MISRA C:2023 – Atomics　アトミック

- R.9.7　R　Atomic objects shall be initialized before creation of the C threads accessing it
  アトミックオブジェクトは、それをアクセスするスレッドの生成前に初期化しなければならない
- R.11.10　R　The *_Atomic* qualifier shall not be applied to the incomplete type void
  _Atomic 修飾子を不完全型 void に適用してはならない
- R.12.6　R　Structure and union members of atomic objects shall not be directly accessed
  アトミックオブジェクトの構造体と共用体のメンバを直接アクセスしてはならない

- Note: a number of other guidelines are modified to include additional atomics related requirements.　多くのガイドラインがアトミックに関連する追加の要件を含むように修正されました

MISRA C:2023 – Floating Point and Complex
浮動小数点と複素数

- D.4.15 R Evaluation of floating-point expressions shall not lead to the undetected generation of infinities and NaNs
  浮動小数点式の評価は、検出されない無限大(inf)や非数(NaN)の生成を起こしてはならない

Notes:

1. R.21.12 (use of the floating point environment <fenv.h>) strengthened, and now *Mandatory*
   (浮動小数点環境 <fenv.h> の使用) は強化され、*Mandatory* になりました
2. The essential type model guidance (R.10.x) enhanced to include complex types
   エッセンシャル型モデルのガイダンス(R.10.x)は複素数型を含むよう拡張されました

MISRA C:2023 – Type Generic Macros　型総称マクロ

- R.21.22　M　All operand arguments to any type-generic macros declared in <tgmath.h> shall have an appropriate essential type

  <tgmath.h> で宣言されるすべての型総称マクロのオペランド引数は、すべて適切なエッセンシャル型を持たなければならない
- R.21.23　R　All operand arguments to any multi-argument type-generic macros declared in <tgmath.h> shall have the same standard type

  <tgmath.h> で宣言されるすべての複数引数の型総称マクロのオペランド引数は、すべて同じ標準型を持たなければならない

Note: existing R.21.11 now *Advisory*　既存の R.21.11 は *Advisory* になりました

MISRA C:2023 – Concurrency (1)　並行性

- D.5.1　R　There shall be no data races between threads
  スレッド間でデータ競合があってはならない
- D.5.2　R　There shall be no deadlocks between threads
  スレッド間でデッドロックがあってはならない
- D.5.3　R　There shall be no dynamic thread creation
  動的なスレッド生成を行ってはならない

- R.21.25　R　All memory synchronization operations shall be executed in sequentially consistent order
  すべてのメモリ同期操作は逐次的に整合性のある順序で実行されなければならない
- R.21.26　R　The Standard Library function *mtx_timedlock()* shall only be invoked on mutex variables of appropriate mutex type
  標準ライブラリ関数 *mtx_timedlock()* は、適切なミューテックス型のミューテックス変数に対してだけ呼び出さなければならない

MISRA C:2023 – Concurrency (2)　並行性

- R.22.11　R　A thread that was previously either joined or detached shall not be subsequently joined nor detached
  以前に join またはデタッチされたスレッドは、その後に join やデタッチしてはならない
- R.22.12　R　Thread objects, thread synchronization objects, and thread-specific storage pointers shall only be accessed by the appropriate Standard Library functions
  スレッドオブジェクト、スレッド同期オブジェクト、スレッド固有の記憶域ポインタは、適切な標準ライブラリ関数だけによってアクセスしなければならない
- R.22.13　R　Thread objects, thread synchronization objects and thread-specific storage pointers shall have static storage duration
  スレッドオブジェクト、スレッド同期オブジェクト、スレッド固有の記憶域ポインタは、静的記憶域期間を持たなければならない
- R.22.14　R　Thread synchronization objects shall be initialized before being accessed
  スレッド同期オブジェクトはアクセスされる前に初期化しなければならない
- R.22.15　R　Thread synchronization objects and thread-specific storage pointers shall not be destroyed until after all threads accessing them have terminated
  スレッド同期オブジェクトとスレッド固有の記憶域ポインタは、それらをアクセスするすべてのスレッドが終了するまで破棄してはならない

MISRA C:2023 – Concurrency (3)　並行性

- R.22.16　R　All mutex objects locked by a thread shall be explicitly unlocked by the same thread
  あるスレッドによってロックされたミューテックスオブジェクトはすべて、同じスレッドによって明示的にアンロックされなければならない
- R.22.17　R　No thread shall unlock a mutex or call cnd_wait() for a mutex it has not locked before
  以前にロックされていないミューテックスに対して、いかなるスレッドもアンロックしたり cnd_wait() を呼び出してはならない
- R.22.18　R　Non-recursive mutex variables shall not be recursively locked
  非再帰的ミューテックス変数を再帰的にロックしてはならない
- R.22.19　R　A condition variable shall be associated with at most one mutex variable
  1つの条件変数は、高々1つのミューテックス変数と関連付けられなければならない
- R.22.20　R　Thread-specific storage pointers shall be created before being accessed
  スレッド固有の記憶域ポインタは、アクセスされる前に生成しなければならない

MISRA C:2023 – Generics (1)　ジェネリック

A new section, s8.23:

- R.23.1　A　A generic selection should only be expanded from a macro
  総称選択はマクロから展開されるべきである
- R.23.2　R　A generic selection that is not expanded from a macro shall not contain *potential side effects* in the controlling expression
  マクロから展開されない総称選択は、制御式内で*副作用の可能性*があってはならない
- R.23.3　A　A generic selection should contain at least one non-default association
  総称選択は少なくとも1つのデフォルトではないアソシエーションを含むべきである
- R.23.4　R　A generic association shall list an appropriate type
  総称アソシエーションは適切な方をリストしなければならない

MISRA C:2023 – Generics (2)　ジェネリック

- R.23.5　A　A generic selection should not depend on implicit pointer type conversion
  総称選択はポインタの暗黙の型変換に依存するべきではない
- R.23.6　R　The controlling expression of a generic selection shall have an essential type that matches its standard type
  総称選択の制御式は、その標準型に適合するエッセンシャル型を持たなければならない
- R.23.7　A　A generic selection that is expanded from a macro should evaluate its argument only once
  マクロから展開された総称選択は、その引数を1度だけ評価するべきである
- R.23.8　R　A default association shall appear as either the first or the last association of a generic selection
  デフォルトのアソシエーションは、総称選択の最初か最後のアソシエーションのどちらかとして現れなければならない

Looking Forward…

Publication Process

- Previously, infrequent new editions
  (1998, 2004, 2012, 2019)

- Recent incremental approach speeds up development, with updates coinciding with the Embedded World Conference
  - 2019 3rd Edition, 1st Revision
  - 2020 Amendment 2
  - 2021 Permits
  - 2022 Amendment 3, Technical Corrigendum 2
  - 2023 Amendment 4

- Print-on-Demand allows flexibility in producing new full documents…

Work in Progress

- Work in Progress
  - Enhancements for Automatically Generated Code
  - Updated coverage mappings for:
    - ISO/IEC TS 17961 “C Secure”
    - CERT C
  - New coverage mappings for:
    - ISO/IEC 24772 Language Vulnerabilities
    - MITRE CWE Common Weakness Enumeration

What Next? – the Incremental

- ISO/IEC 9899:202x (aka C2x)
  - New and revised features
- Improvements for *undecidability*
  - Use of annotations to aid analysis, eg [[misra::xxx]]
- Enhanced guidance for the Standard Library
  - Use of dynamic memory
  - Character data handling
  - Use of stdio.h

What Next? – More fundamental

- A Layered Approach?
  - Predictable Subset v Good/Best Practice
  - Bare Metal v Application
  - Freestanding v Hosted
- What about C90?
  - Do we still need to continue C90 support?

Software Verification (aka “Static Analysis”)

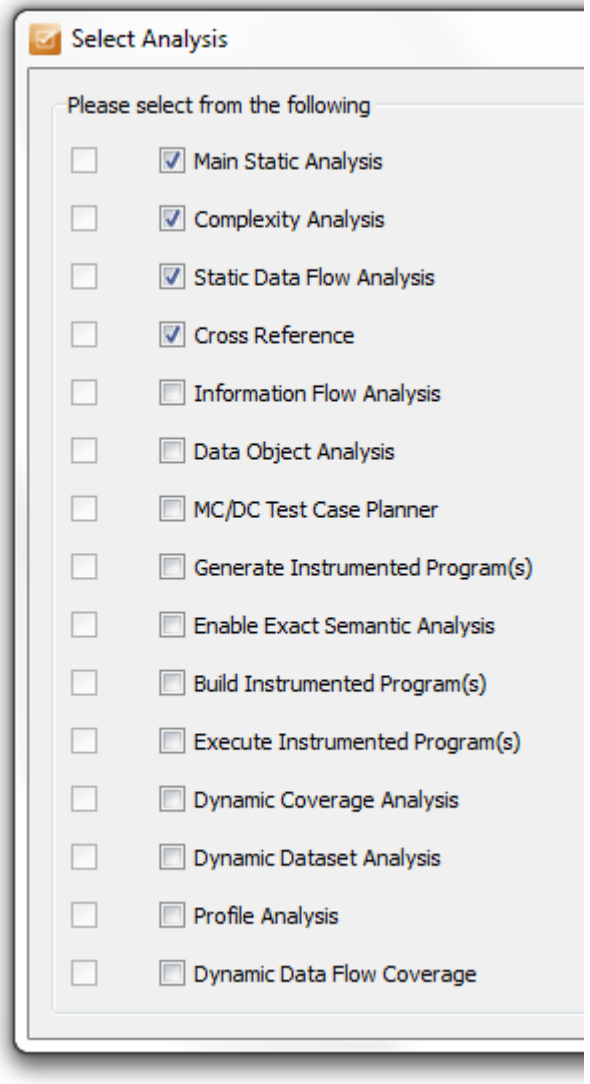
ソフトウェア検証（静的解析による）

Software Verification ソフトウェアの検証

Software Verification is used to identify: 検証で解析できること

- Static Analysis 静的解析
  - Programming standards violations コーディング規約違反
  - Inappropriate use of the programming language
    プログラミング言語の不適切な使用
  - Inconsistent interfaces between modules/functions
    一貫性がないモジュール/機能間のインターフェイス
  - Unreachable (or dead) code 到達不能(またはデッド)コード
- Complexity Analysis 複雑度解析
- Data Flow Analysis データフロー解析
  - Unused variables and constants etc 未使用の変数や定数など
  - Uninitialised data 初期化されていないデータ

It also provides some useful (and some not so useful) metrics
いくつかの有用な (およびあまり役に立たない) メトリクスもある



Software Verification ソフトウェアの検証

#1 Heed compiler warnings コンパイラの警告にご注意を

- Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code.

  最高の警告レベルでコンパイルし、コードを修正して警告を排除する

  - Static analysis of code with reference to secure coding standards such as CWE, CERT C and MISRA C, to minimize the use of constructs and techniques likely to introduce vulnerabilities into the system

    CWE、CERT C、MISRA C などセキュアなコーディング規約で、システムに脆弱性をもたらす構成や手法の使用を最小限に抑える
  - Data and Control Coupling analysis, providing a means to recognize how data injection could potentially infiltrate a system, and to ensure that it is defended accordingly

    データ結合・制御結合の分析。データがシステムにどのように侵入する可能性があるかを認識し、それに応じてシステムが確実に防御されるようにする手段を提供する
  - Unit test to show that defensive mechanisms have been implemented and are traceable back to security requirements 防御メカニズムが実装され、セキュリティ要件まで追跡可能であることを示す単体テスト
  - Automatically generated robustness test cases, ensuring that such as boundary conditions, null pointers, and default conditions are handled adequately and hence introduce no vulnerabilities

    堅牢性テストケースの自動生成により、境界条件、ヌルポインター、デフォルト条件などが適切に処理され、脆弱性が生じないことを保証

#2 Adopt an appropriate coding standard 適切なコーディン規約の採用

- Develop and/or apply a coding standard for your target development language and platform. Use static and dynamic analysis tools to detect and eliminate additional security flaws

  開発言語およびプラットフォーム用のコーディング規約を開発または採用する。静的および動的解析ツールでセキュリティ上の欠陥を検出して排除する

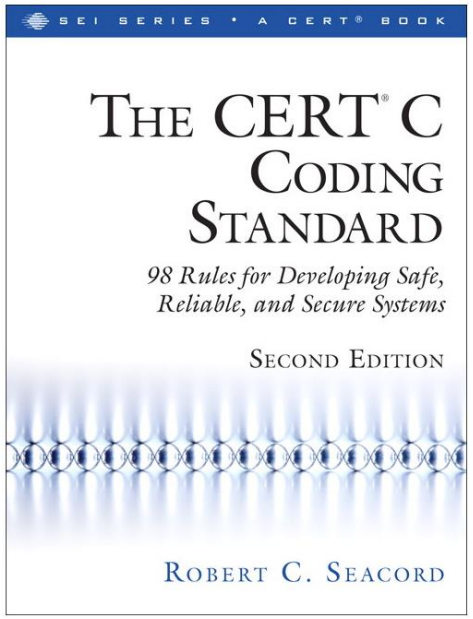
Dir 4.14 The validity of values received from external sources shall be checked

C90 [Undefined 15, 19, 26, 30, 31, 32, 94]
C99 [Undefined 15, 16, 33, 40, 43-45, 48, 49, 113]

Category Required

Applies to C90, C99

MISRA



#3 Validate inputs 入力を検証する

- Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities.

  全ての信頼できないデータソースからの入力を検証する
  適切な入力検証により、ソフトウェアの脆弱性の大部分を排除できる

- Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files

  コマンドライン引数、ネットワークインターフェイス、環境変数、ユーザー制御ファイルなど、ほとんどの外部データソースを疑う

```
static void
  print_buffer ( void )
  {
    char
      input [ 128 ] ;
    ( void ) scanf ( "%128c" , input ) ;
    ( void ) printf ( "%s" , input ) ;
 86 D : User input not checked before use. : input (MISRA-C:2012/AMD1 D.4.14)
  }
```

#3 Validate inputs 入力を検証する

```
169  #pragma vector=ADC12_VECTOR
170  __interrupt void ADC12ISR (void)
171  {
172    switch(__even_in_range(ADC12IV,34))
173    {
174    case  0: break;
175    case  2: break;
176    case  4: break;
177    case  6:
178          adc12_result = ADC12MEM0;
179          adc12_data_ready = 1;
180          _BIC_SR_IRQ(LPM3_bits);
181          break;
182    case  8: break;
```

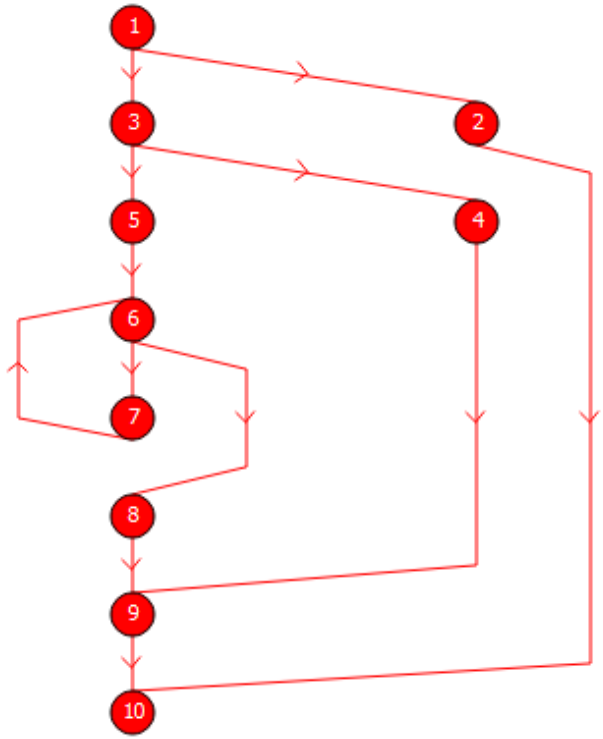
adc12_result (Pri)		u16	adc12.c	<Global scope>	G	E	60				
			adc12.h/adc12.c		G	E	62				
				ADC12ISR	G	D	178				
				adc12_single_conversion	G	R	157				

#4 Keep it simple シンプルに保つ

- Keep the design as simple and small as possible.
  デザインはできるだけシンプルで小さいものに
- Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use
  複雑な設計では、実装、構成、使用時にエラーが発生する可能性が高くなる

Code Quality View (Maintainability)

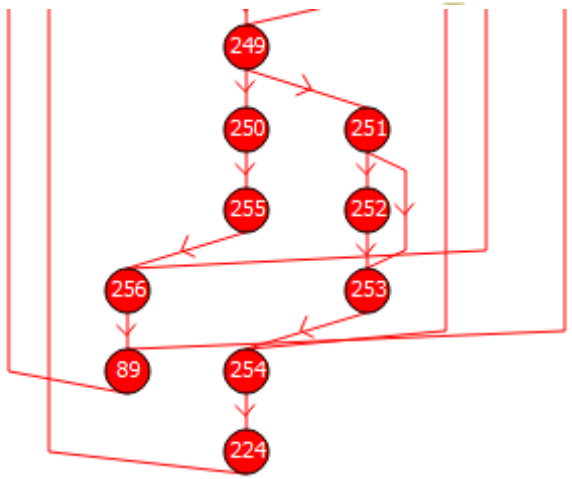
Procedure Calls	Cyclomatic Complexity
main	6
Containers::StaticLinkList::removeLast	4
Containers::DynamicLinkList::removeLast	4
Containers::DynamicLinkList::addLast	3
Containers::StaticLinkList::addLast	3
Containers::DynamicLinkList::removeFirst	3
Containers::StaticLinkList::removeFirst	3
Containers::StaticLinkList::addFirst	3
Containers::DynamicLinkList::addFirst	3
Containers::DynamicLinkList::removeAll	2
Containers::StaticLinkList::removeAll	2
Containers::DynamicLinkList::getCount	2

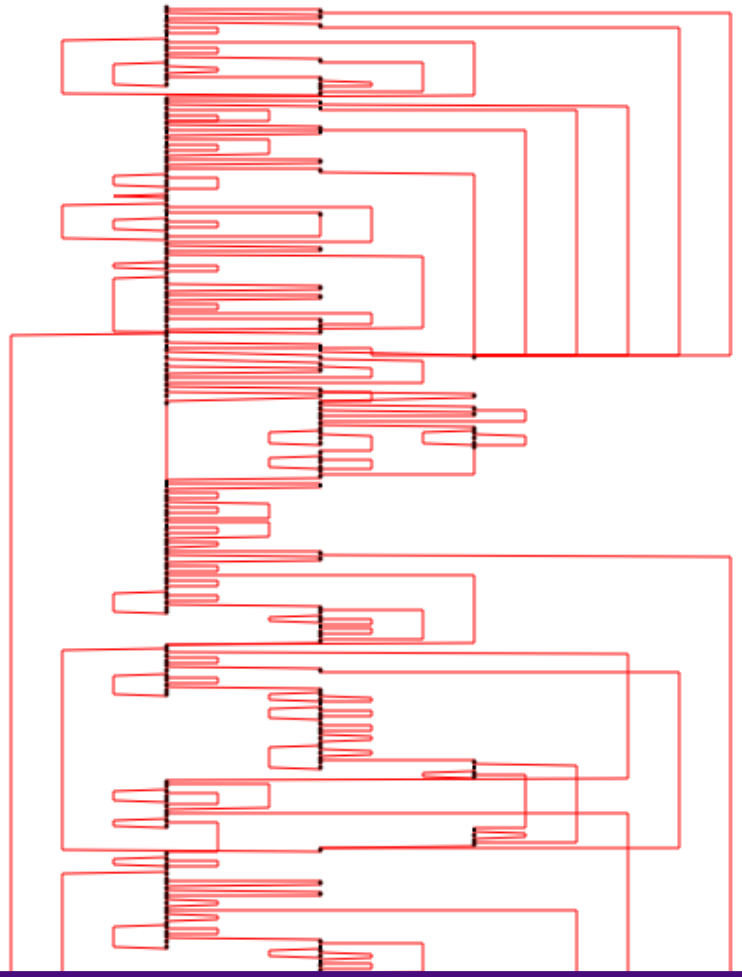


Software Verification ソフトウェアの検証

Code Quality View (Maintainability)

Procedure Calls	Cyclomatic Complexity
GetOptimum	98 : (Fail)
LzmaDec_DecodeReal	82 : (Fail)
LzmaDec_TryDummy	61 : (Fail)
GetOptimumFast	41 : (Fail)
LzmaDec_DecodeToDic	30 : (Fail)
LzmaEnc_CodeOneBlock	30 : (Fail)
main2	23
LzmaEncProps_Normalize	20

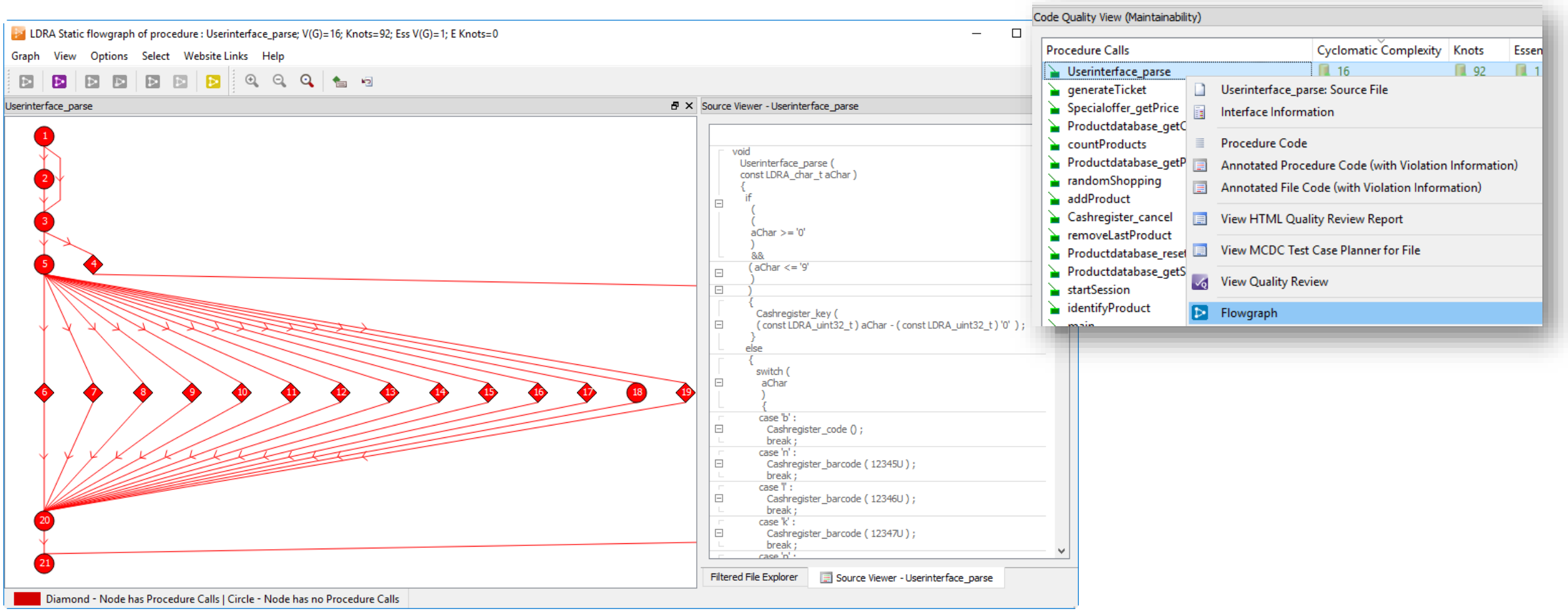




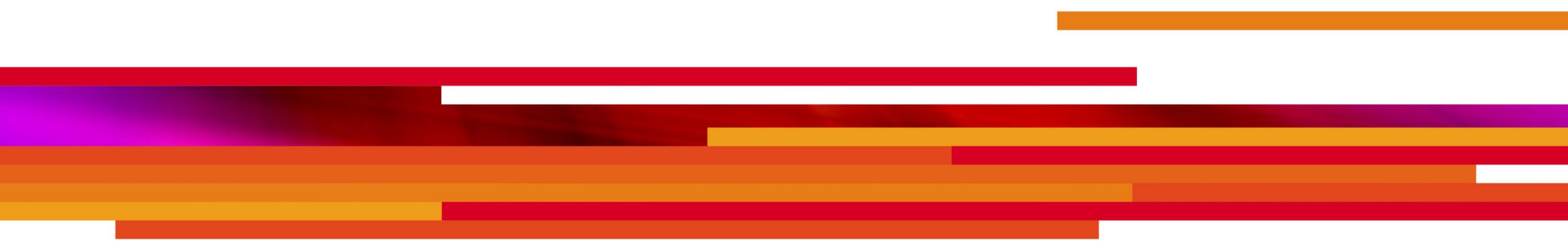
Impossible to understand, maintain or test
これでは理解、保守、テストが不可能

Software Verification ソフトウェアの検証

- Treat Code Complexity with caution... For example a switch() construct has a high calculated complexity!
  コードの複雑さには注意... 例えば、switch() コンストラクトは計算の複雑さが高くなる



In conclusion...

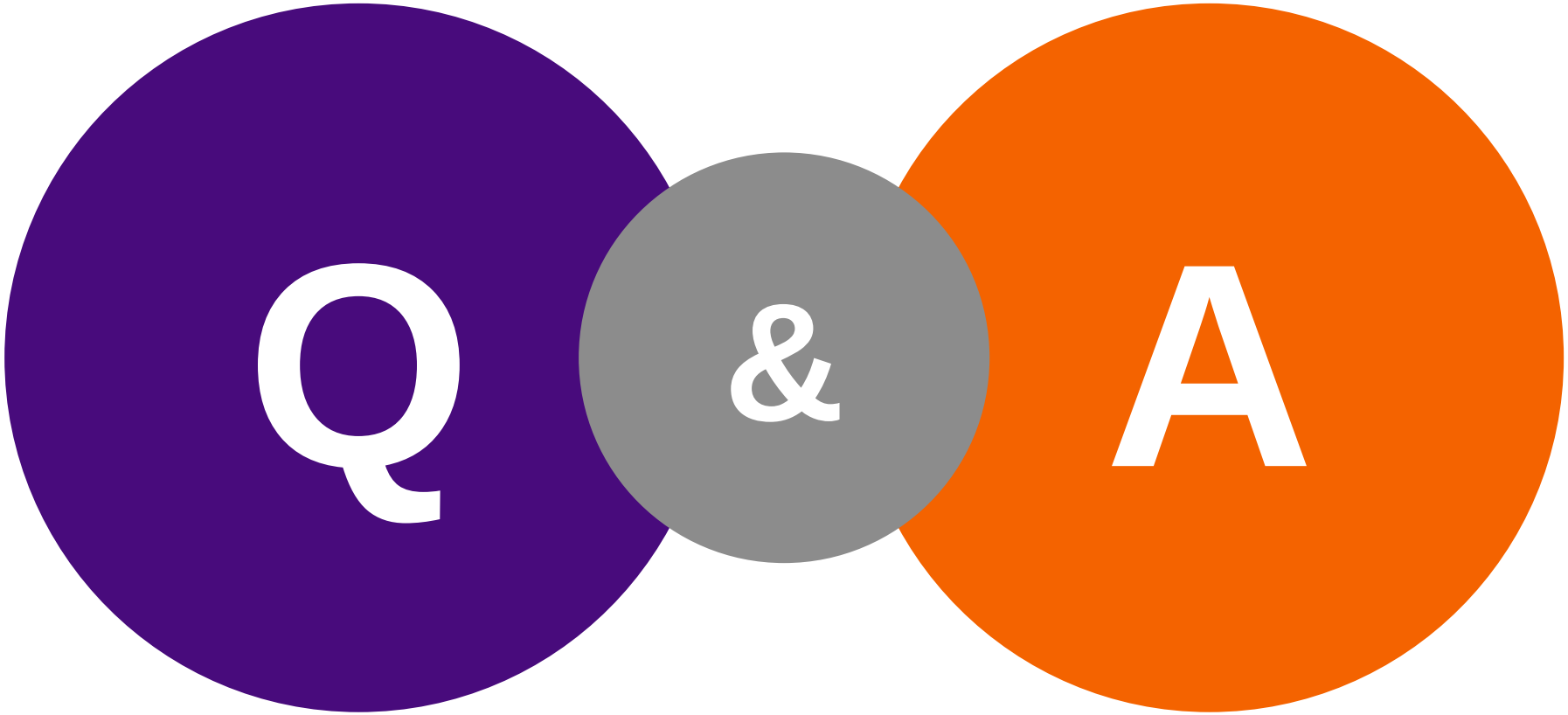


MISRA C – In Summary

- MISRA C and MISRA C++ are
  - widely respected as a safety-related coding standard
  - equally applicable as a security-related coding standard
  - appropriate for use in all high-integrity and high-reliability environments
- MISRA C and MISRA C++ have
  - evolved from being automotive guidance into a pan-industry best practice
- MISRA C and MISRA C++ will
  - continue to evolve as new editions of the C/C++ standards are produced
  - seek to address other constraints as they become identified
- The MISRA C Working Group welcomes feedback from all users

MISRA Working Groups – Get Involved

- The MISRA C and MISRA C++ Working Groups are made up of volunteers.
- Most work is still done via Zoom meetings
- Face-to-Face meetings are typically held at Sibson (near Nuneaton).
- Membership is open to anyone with the appropriate technical ability.
- Anyone interested in joining should contact the respective Working Group Chairman:
  - For MISRA C chair.c@misra.org.uk
  - For MISRA C++ chair.cpp@misra.org.uk
  - For MISRA SQM chair.sqm@misra.org.uk


Q
&
A

Contact Us

Need more information?

LDRA.com

LDRA Software Technology

LDRA Limited

@ldra_technology

LDRA Tools

info@ldra.com