The C Language

… you wouldn’t start from here!

Andrew Banks

Explainable Real-Time Systems and Their Analysis (ERSA) 2024

The Speaker

Andrew Banks

Andrew.Banks@LDRA.com

- Software Engineer & Standards Evangelist

Focus: helping YOU to get your software right, first time!

- Biography
  - Over 35 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
- Standards
  - Chairman of MISRA C Working Group since June 2013...
    ... Working Group member since 2007
  - Chairman of the BSI Software Testing Working Group
    … UK Head-of-Delegation to ISO/IEC JTC1/SC7
  - Contributor to ISO 29119 “Software Testing”
  - Contributor to ISO 26262 2nd Edition “Functional Safety”
  - etc

Agenda

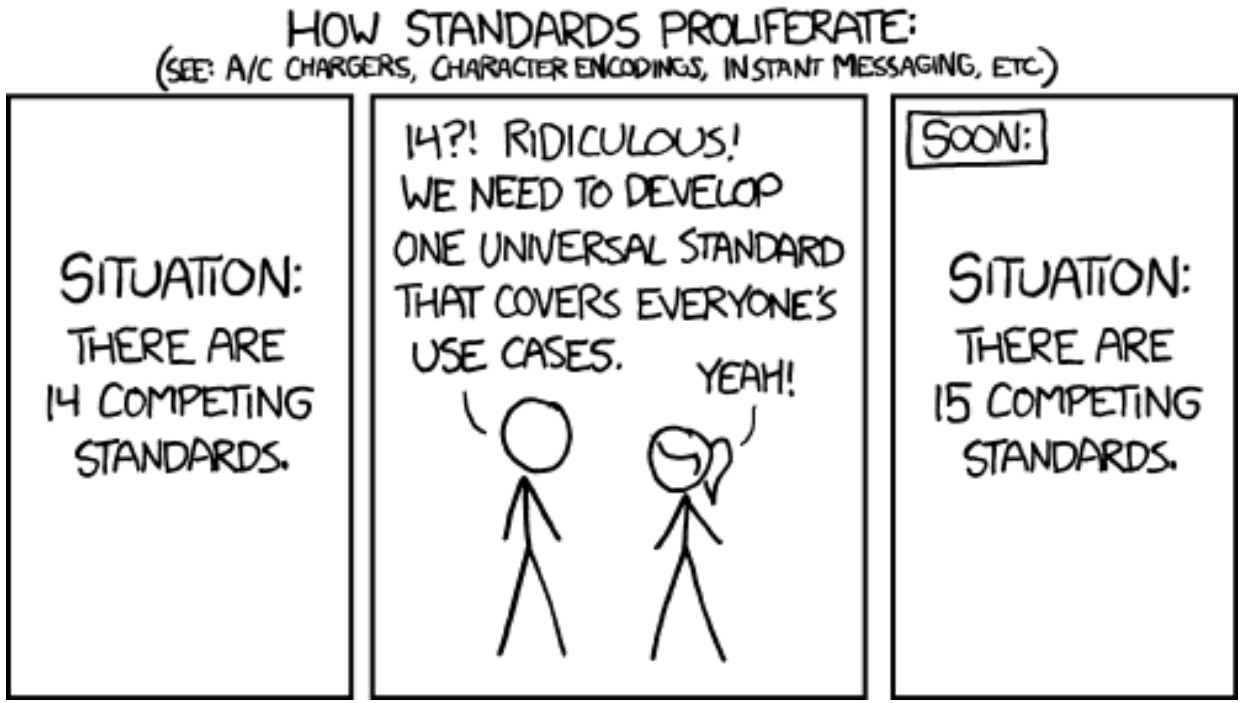
1. Functional Safety Standards
   … and how MISRA fits in
2. The C Language...
   ... and what is wrong with it
3. An introduction to MISRA C
4. MISRA C in an
   ... ISO 26262 context
5. Why Use MISRA…
   ... Or, in fact, any other Static Analysis

International Standards
Safety, Security and AI

Setting the scene…

> The nice thing about standards
> ... is that you have so many to choose from!

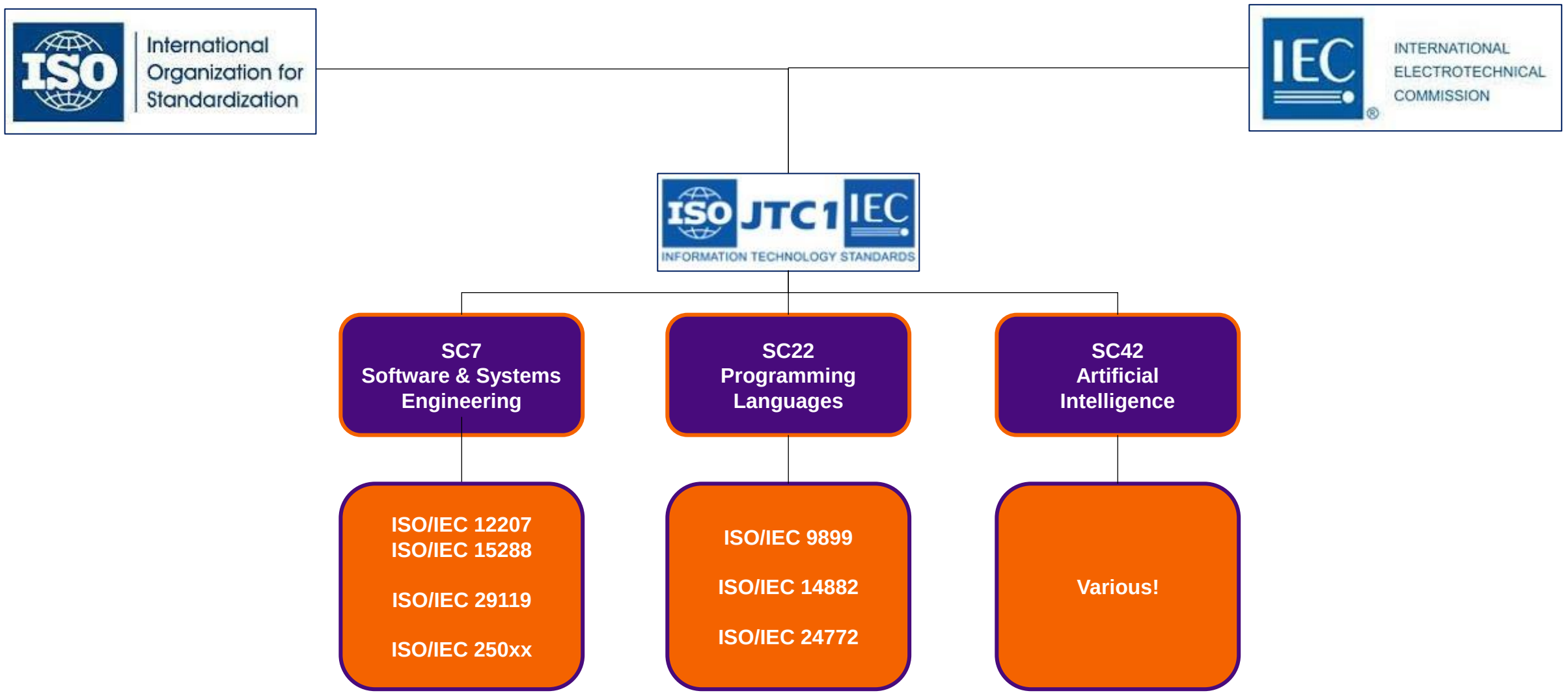
Andrew S. Tanenbaum



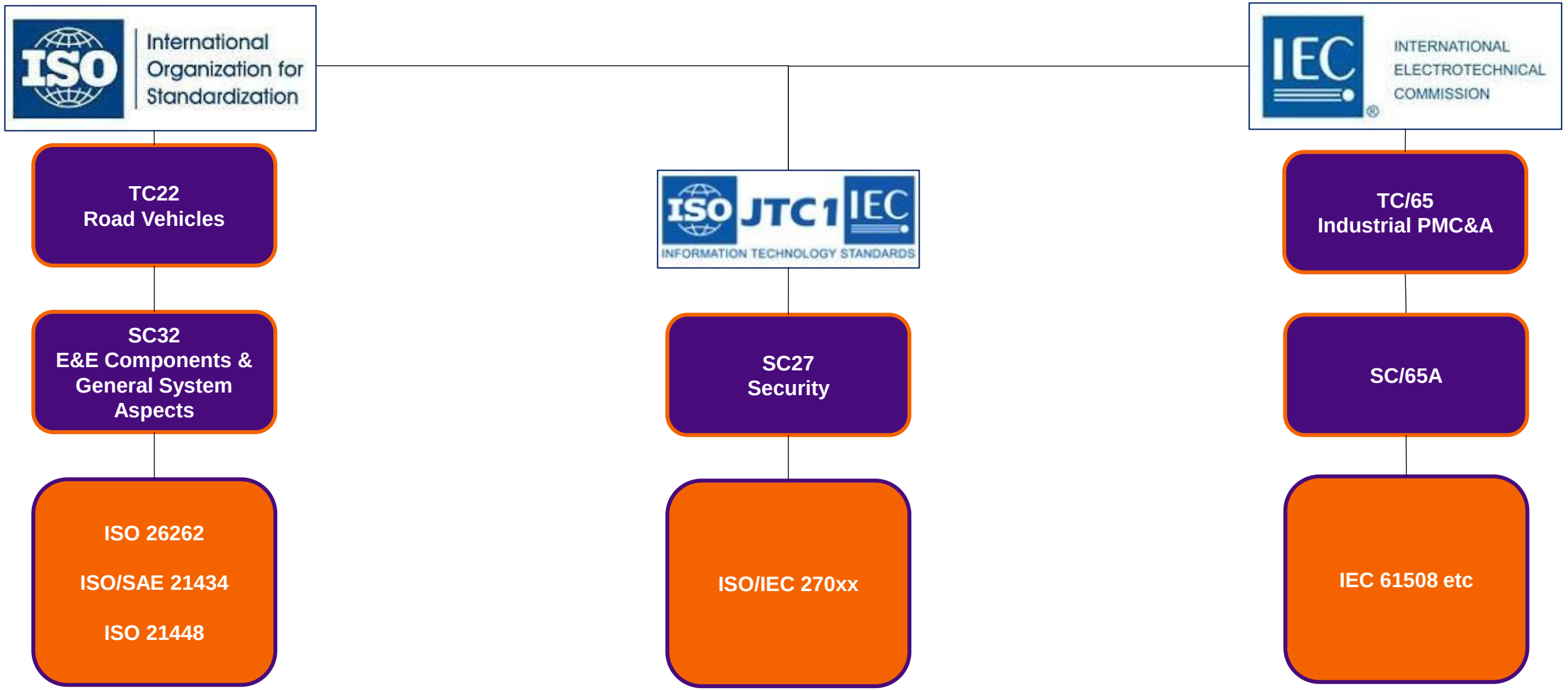
Functional Safety Standards

		LDRA
▪ IEC 61508	Functional Safety of E/E/PES	X
▪ IEC 61511	Industrial Process Equipment	[X]
▪ IEC 61513	Nuclear	X
▪ IEC 62304	Medical Devices	X
▪ ISO 13849	Machinery	[X]
▪ ISO 25119	Agriculture & Forestry	[X]
▪ ISO 26262	Road Vehicles	X
▪ EN 50126/7/8/9	Railway	X
▪ DO-178	Airborne Systems	X

Information Technology

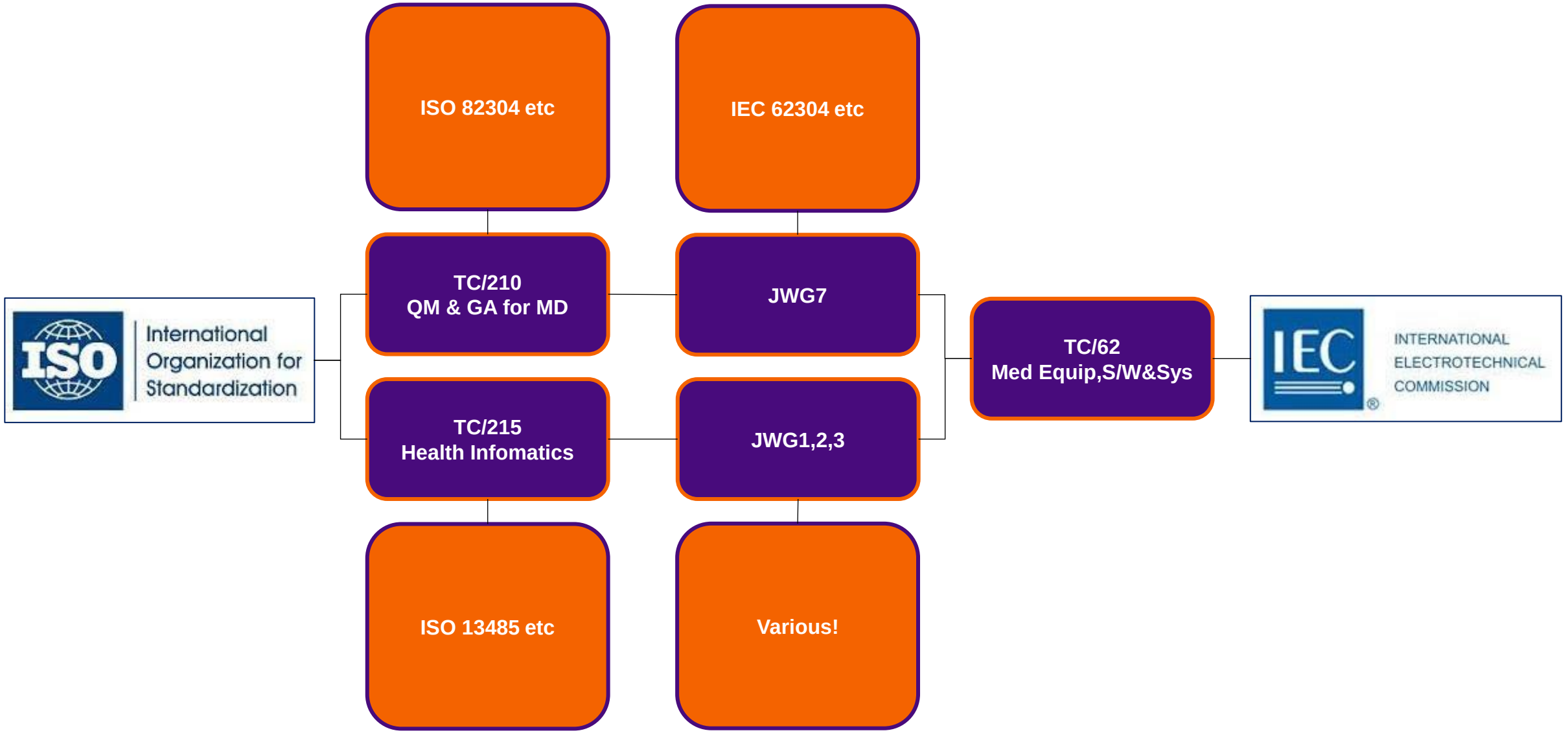


Functional Safety and Security 1/2

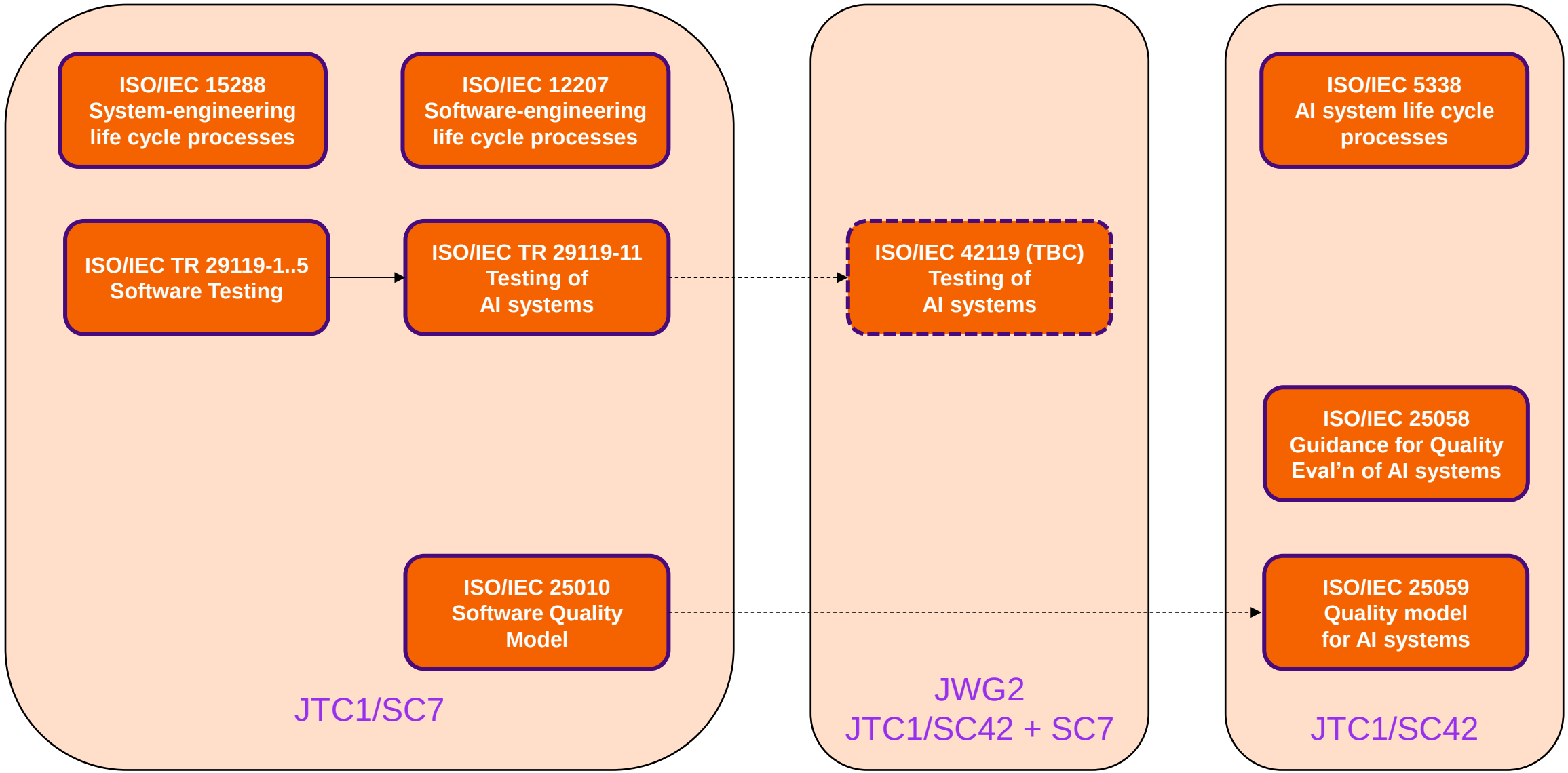


PMC&A = Industrial process measurement, control and automation

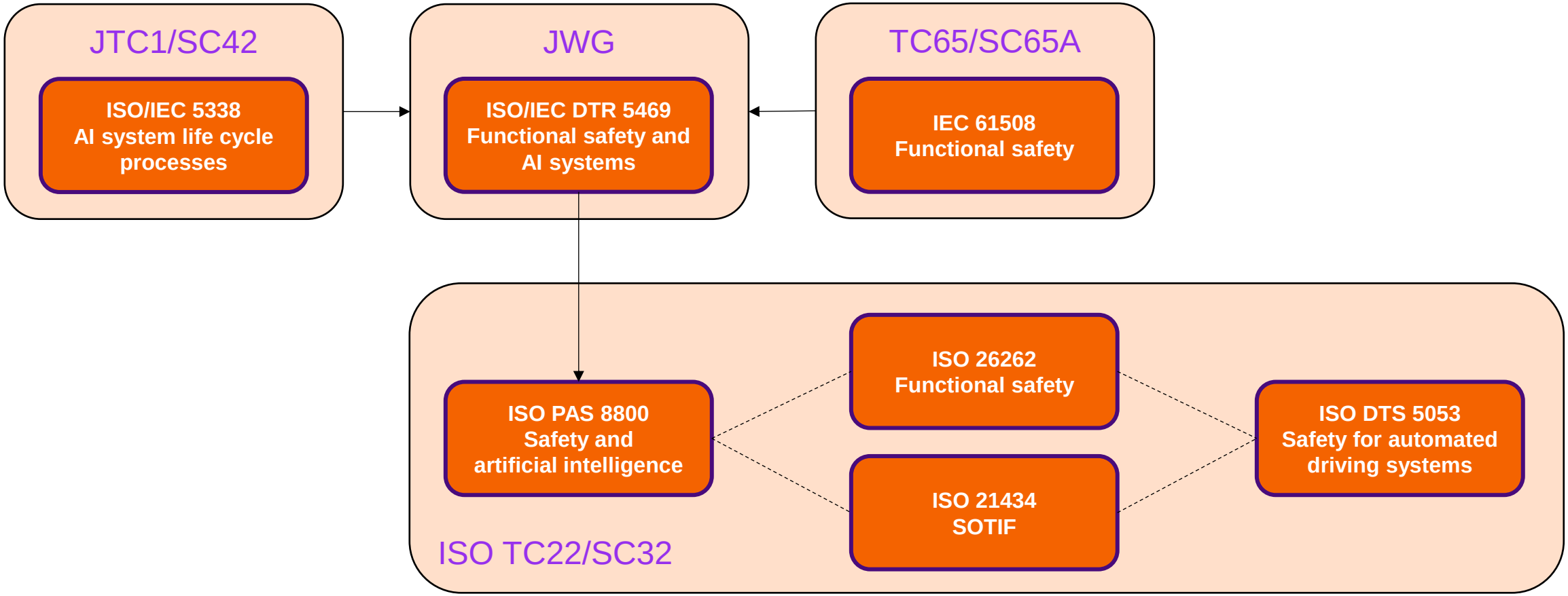
Functional Safety and Security (2/2)



Artificial Intelligence Life Cycle Processes



Safety and Artificial Intelligence



The C Language...

... and what is wrong with it

“

In the beginning, the Universe was created.

This has made a lot of people very angry
… and been widely regarded as a bad move.

The Restaurant at the End of the Universe
Book 2 of the Douglas Adams’ 5-part Trilogy
The Hitch Hiker’s Guide To The Galaxy

Far back in the mists of ancient time…

- 1963 CPL — *Cambridge Programming Language*
  Christopher Strachey et al of University of Cambridge

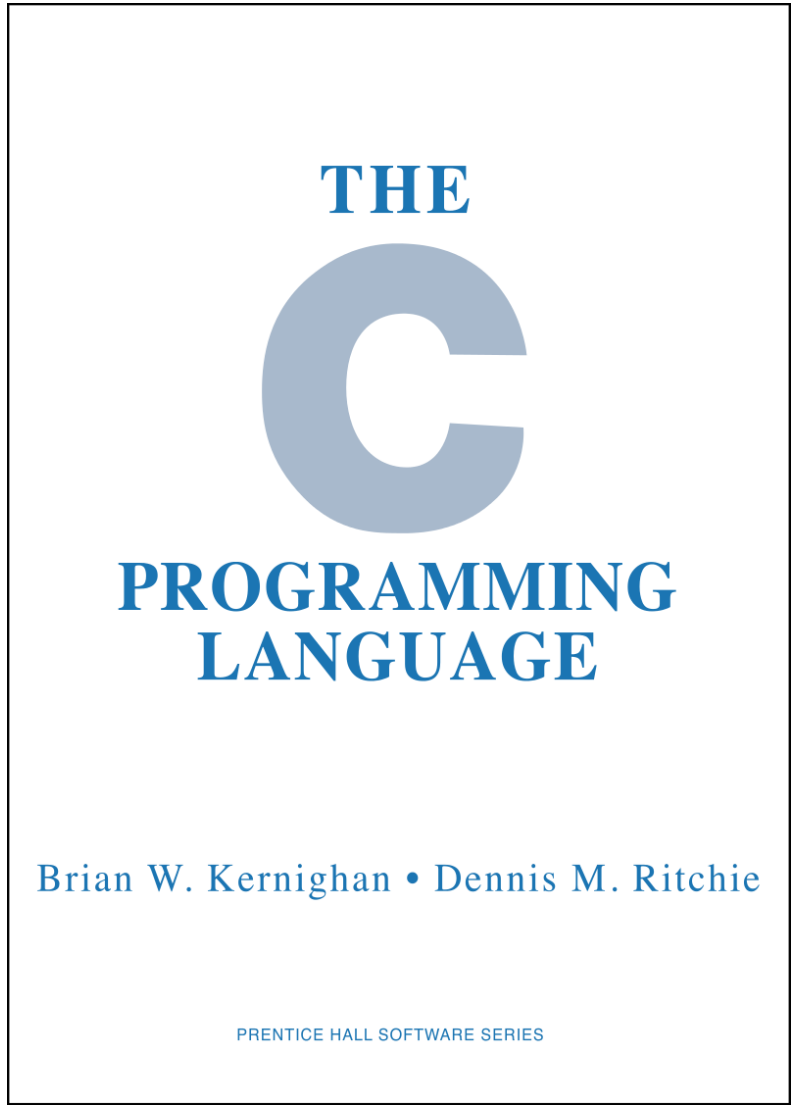
  Later known as the *Combined Programming Language*
  with the involvement of the University of London

- 1967 BCPL — *Bootstrap CPL* (or *Basic CPL*)
  Martin Richards at the University of Cambridge

- 1969 B — Ken Thompson and Dennis Ritchie at Bell Labs.

The C Language – A Quick History

- K&R C
  - 1972 First created by Dennis Ritchie
  - 1976 Lint, the first C static analyser
    ... created by Stephen Johnson
  - 1978 The C Programming Language published
- ANSI C
  - 1989 First standardized version
    ANSI X3.159-1989 (aka C89)



The C Language – A Quick History

- ISO C
  - 1990 ISO/IEC 9899:1990 aka C90 Equivalent to C89
  - 1995 Amendment 1 aka C95
  - 1999 ISO/IEC 9899:1999 aka C99
  - 2011 ISO/IEC 9899:2011 aka C11
  - 2018 ISO/IEC 9899:2018 aka C18 A “TC” in all but name (aka C17)
  - 2024 ISO/IEC 9899:2024 aka C24 Also known as C23!

Very few (if any) of you will be using ANSI C any more!

“We don't demand solid facts!

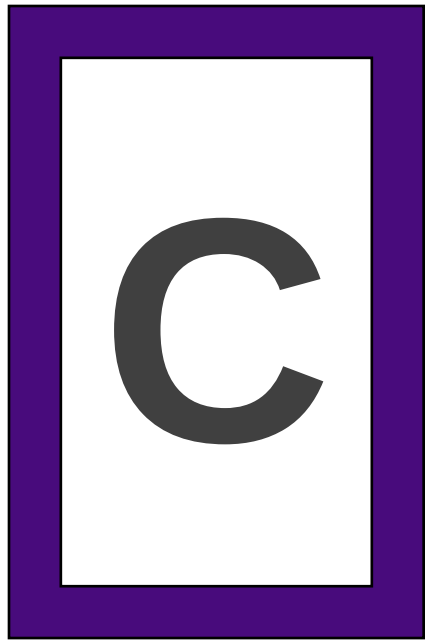
What we demand is a total absence of solid facts.

We demand rigidly defined areas of doubt and uncertainty!

Vroomfondel the Philosopher, in
The Hitch Hiker’s Guide To The Galaxy
by Douglas Adams

The C Language – What is the problem?

- Despite its popularity, there are several drawbacks with the C language, eg:
  - The ISO Standard language definition is incomplete:
    - Behaviour that is *Undefined* 61 incidences
    - Behaviour that is *Unspecified* 211 incidences
    - Behaviour that is *Implementation Defined* 120 incidences
    - Behaviour that is *Locale-dependant* 15 incidences
  - Language misuse and obfuscation
  - Language misunderstanding
  - Run-time error checking
- MISRA C is one solution...



The C Language – What is the problem?

- The ISO Standard language definition is incomplete
  - Undefined behaviour is *behaviour, upon use of a nonportable or erroneous program construct or of erroneous data, for which the International Standard imposes no requirements*
  - An example of *undefined behaviour* is the behaviour on integer overflow

    211 instances
  - Unspecified behaviour invokes *the use of an unspecified value, or other behaviour where the International Standard provides two or more possibilities and imposes no further requirements on which is chosen in any instance*
  - An example of *unspecified behaviour* is the order in which the arguments to a function are evaluated.

    61 instances

If not C, then what?

So, if not C, then what?

- 1977 etc Modula, Modula-2
- 1980(?) Perspective Pascal
- 1983 Ada (also including SPARK)
- 1993 Lua
- 1995 Java
- 2010 Rust
- 2024 ?

There is another way…

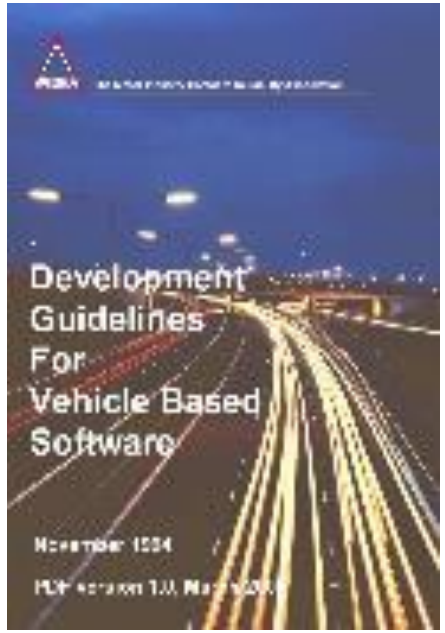
An introduction to MISRA C

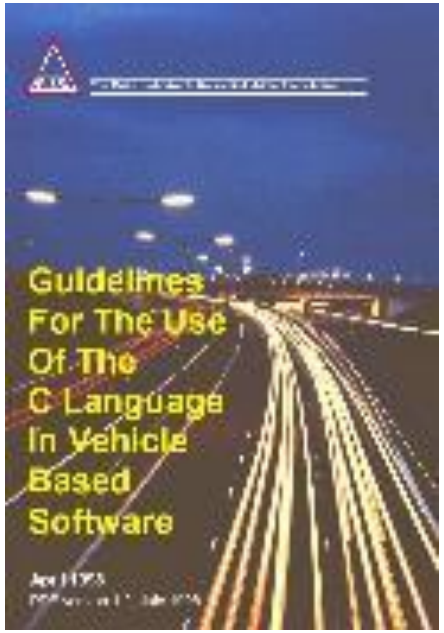
Original MISRA publications

- **November 1994**

Development guidelines for vehicle based software (aka The MISRA Guidelines)

  - The first automotive publication concerning functional safety
  - Commenced more than 10 years before work started on ISO 26262





- **April 1998**

Guidelines for the use of the C language in vehicle based software (aka MISRA C)

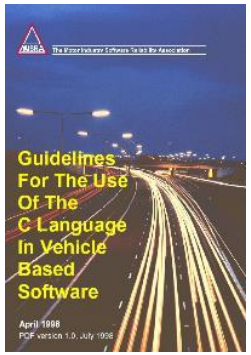
- **December 1998**

IEC 61508 (first edition) published!

MISRA C – A Quick History

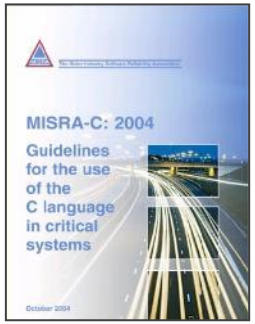
MISRA-C:1998

- “Guidelines for the use of the C language in vehicle based software”
- Compatible with ISO/IEC 9899:1990 (aka C90)



MISRA-C:2004

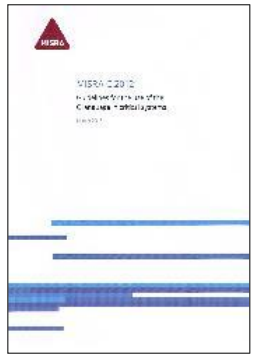
- “Guidelines for the use of the C language in critical systems”
- Remains compatible with ISO/IEC 9899:1990 (aka C90)



MISRA C – A Quick History

MISRA C:2012 (3rd Edition)

- Adds compatibility with ISO/IEC 9899:1999 (aka C99)
- Amendment 1 in 2016 included additional security guidelines



MISRA C:2012 (3rd Edition, 1st Edition) [published 2019]

- Consolidated enhancements introduced by AMD1 and TC1
- Further enhancements in 2020 (AMD2), 2022 (TC2, AMD3) and 2024 (AMD4)

MISRA C:2023 (3rd Edition, 2nd Revision)

- Latest version, consolidating all recent work, to mark 25th anniversary!

“The MISRA C Guidelines define a subset of the C language in which the opportunity to make mistakes is either removed or reduced.

Many standards for the development of safety-related software require, or recommend, the use of a language subset, and this can also be used to develop any application with security, high integrity or high reliability requirements.”

MISRA C in one slide!

- 221 Guidelines: 21 Directives (5 sections) and 200 Rules (23 sections)

Directives

- The implementation
- Compilation and build
- Requirements traceability
- Code design
- Concurrency considerations

Rules

- A standard C environment
- Unused code
- Comments
- Character sets & lexical conventions
- Identifiers
- Types
- Literals & constants
- Declarations & definitions
- Initialization
- The essential type model
- Pointer type conversions
- Expressions
- Side effects
- Control statement expressions
- Control flow
- Switch statements
- Functions
- Pointers and arrays
- Overlapping storage
- Preprocessing directives
- Standard libraries
- Resources
- Generic Selections

3

MISRA C in an ... ISO 26262 context

MISRA C – In an ISO 26262 Context

ISO 26262-6:2018, Section 5.4.3

- Criteria for suitable modelling, design or programming languages that are not sufficiently addressed by the language itself shall be covered by the corresponding guidelines, or by the development environment, considering the topics listed in Table 1

- Example 1: MISRA C is a coding guideline for the programming language C and includes guidance on automatically generated code

> **5.4.3** Criteria for suitable modelling, design or programming languages (see 5.4.2) that are not sufficiently addressed by the language itself shall be covered by the corresponding guidelines, or by the development environment, considering the topics listed in Table 1.
>
> EXAMPLE 1 MISRA C (see Reference [3]) is a coding guideline for the programming language C and includes guidance for automatically generated code.

MISRA C – In an ISO 26262 Context

ISO 26262-6:2018, Table 1

Table 1 — Topics to be covered by modelling and coding guidelines

Topics		ASIL			
		A	B	C	D
1a	Enforcement of low complexity[a]	++	++	++	++
1b	Use of language subsets[b]	++	++	++	++
1c	Enforcement of strong typing[c]	++	++	++	++
1d	Use of defensive implementation techniques[d]	+	+	++	++
1e	Use of well-trusted design principles[e]	+	+	++	++
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+	++	++	++
1h	Use of naming conventions	++	++	++	++
1i	Concurrency aspects[f]	+	+	+	+

MISRA C – In an ISO 26262 Context

- ISO 26262-6:2018, Section 8.4.5
  - Design principles for software unit design and implementation at the source code level as listed in Table 6 shall be applied to achieve the following properties:
    - correct order of execution of subprograms and functions within the software units, based on the software architectural design;
    - consistency of the interfaces between the software units;
    - correctness of data flow and control flow between and within the software units;
    - simplicity;
    - readability and comprehensibility;
    - robustness;
    - suitability for software modification; and
    - verifiability

MISRA C – In an ISO 26262 Context

ISO 26262-6:2018, Table 6

Table 6 — Design principles for software unit design and implementation

Principle		ASIL			
		A	B	C	D
1a	One entry and one exit point in subprograms and functions[a]	++	++	++	++
1b	No dynamic objects or variables, or else online test during their creation[a]	+	++	++	++
1c	Initialization of variables	++	++	++	++
1d	No multiple use of variable names[a]	++	++	++	++
1e	Avoid global variables or else justify their usage[a]	+	+	++	++
1f	Restricted use of pointers[a]	+	++	++	++
1g	No implicit type conversions[a]	+	++	++	++
1h	No hidden data flow or control flow	+	++	++	++
1i	No unconditional jumps[a]	++	++	++	++
1j	No recursions	+	+	++	++

[a] Principles 1a, 1b, 1d, 1e, 1f, 1g and 1i may not be applicable for graphical modelling notations used in model-based development.

NOTE For the C language, MISRA C (see Reference [3]) covers many of the principles listed in Table 6.

MISRA C – In an ISO 26262 Context

Static Analysis, control flow analysis and data flow analysis are mentioned twice as a set:

- Table 7 ... software unit verification
- Table 10 ... verification of software integration

Control-flow and data-flow analysis are also mentioned in Table 4:

- Table 4 ... verification of software architectural design

MISRA C – In an ISO 26262 Context

ISO 26262-6:2018, Table 7 (unit)

Table 7 — Methods for software unit verification

Methods		ASIL			
		A	B	C	D
1a	Walk-through[a]	++	+	o	o
1b	Pair-programming[a]	+	+	+	+
1c	Inspection[a]	+	++	++	++
1d	Semi-formal verification	+	+	++	++
1e	Formal verification	o	o	+	+
1f	Control flow analysis[b, c]	+	+	++	++
1g	Data flow analysis[b, c]	+	+	++	++
1h	Static code analysis[d]	++	++	++	++
1i	Static analyses based on abstract interpretation[e]	+	+	+	+
1j	Requirements-based test[f]	++	++	++	++
1k	Interface test[g]	++	++	++	++
1l	Fault injection test[h]	+	+	+	++
1m	Resource usage evaluation[i]	+	+	+	++
1n	Back-to-back comparison test between model and code, if applicable[j]	+	+	++	++

MISRA C – In an ISO 26262 Context

ISO 26262-6:2018, Table 10

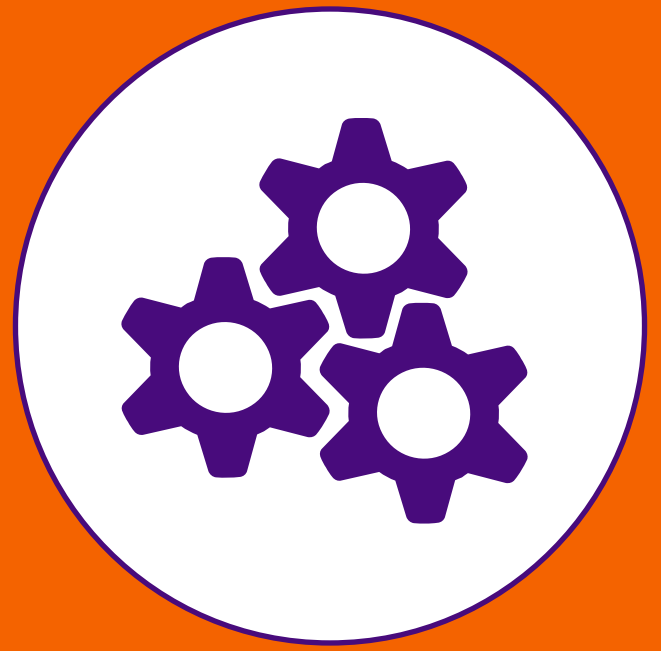
Table 10 — Methods for verification of software integration

Methods		ASIL			
		A	B	C	D
1a	Requirements-based test[a]	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test[b]	+	+	++	++
1d	Resource usage evaluation[c, d]	++	++	++	++
1e	Back-to-back comparison test between model and code, if applicable[e]	+	+	++	++
1f	Verification of the control flow and data flow	+	+	++	++
1g	Static code analysis[f]	++	++	++	++
1h	Static analyses based on abstract interpretation[g]	+	+	+	+

This also maps to the MISRA C guideline scope:

- Unit Verification — Single-translation-unit guidelines
- Integration — System-wide guidelines

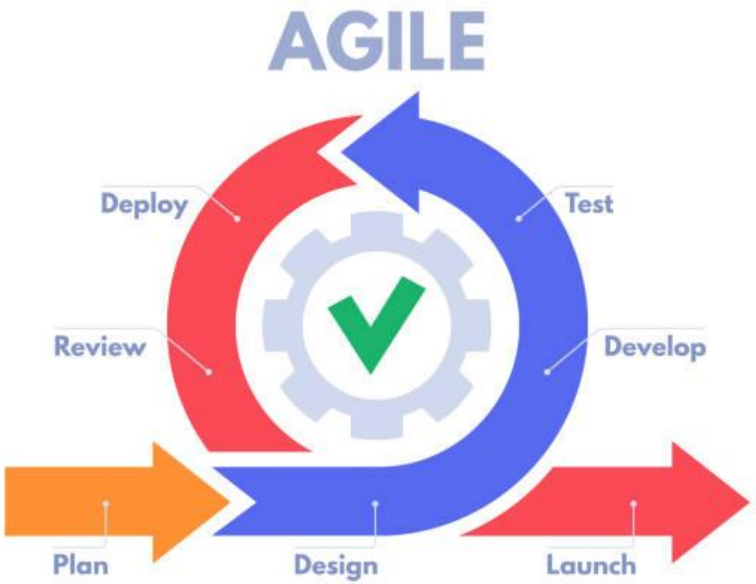
Why Use MISRA…
… Or, in fact, any other Static Analysis



Introduction

Software Testing is…

1. An integral part of the software development life-cycle
2. Often reduced due to budget, resource, and timeline pressures
3. Frequently seen as a mystical black art



Caution!

> Program testing can be used to show the presence of bugs,
> … but never to show their absence!

Edsger Dijkstra

Checking Compliance

- Check the code manually
  - Needs to be done on MISRA C “undecidable” rules
  - But don’t really want to do it on all the code!
- Use a lightweight tool, such as is often built into compilers
  - Fast (Checks just a subset)
  - Detects the easy to find defects
  - Tends to be “Optimistic” – False Negatives

- Use a heavyweight tool
  - Slow (Deep analysis, Check all rules)
  - Detects the easy and hard to find defects (The “once a year” ones!)
  - Tends to be “Pessimistic” – False Positives

Section 8.10 “The Essential Type Model”

- The rules in this section collectively define the essential type model and restrict the C type system so as to:
  - Support a stronger system of type-checking;
  - Provide a rational basis for defining rules to control the use of implicit and explicit type conversions;
  - Promote portable coding practices;
  - Address some of the type conversion anomalies found within ISO C.
- The essential type model does this by allocating an essential type to those objects and expressions which ISO C considers to be of arithmetic type.
- For example, adding an *int* to a *char* gives a result having essentially character type rather than the *int* type that is actually produced by integer promotion.

Use of defensive implementation techniques

- MISRA C has guidance relating to:
  - Control flow
    - If / else if / else
    - Switch / default
    - While / do
    - For loops
  - Unreachable code
    - There shall be no unreachable code
    - There shall be no unused code

Defensive Implementation Techniques

- Consider the Required MISRA C:2012 Rule 2.1
  - A project shall not contain unreachable code
- Consider the Required MISRA C:2012 Rule 15.6
  - The body of an iteration-statement or a selection-statement shall be a compound-statement. eg:

```
if ( condition )
{
    action();
}
```

- Some suggest that these Rules are (to be polite) unnecessary...
- I wonder if Apple's software team agree?
  - CVE-2014-1266

The Apple iPhone SSL Bug

```
if ( ( err = SSLFreeBuffer( &hashCtx ) ) != 0 )
    goto fail;
if ( ( err = ReadyHash(&SSLHashSHA1, &hashCtx ) ) != 0 )
    goto fail;
if ( ( err = SSLHashSHA1.update( &hashCtx, &clientRandom ) ) != 0 )
    goto fail;
if ( ( err = SSLHashSHA1.update( &hashCtx, &serverRandom ) ) != 0 )
    goto fail;
if ( ( err = SSLHashSHA1.final( &hashCtx, &hashOut ) ) != 0 )
    goto fail;
```

The Apple iPhone SSL Bug

```
if ( ( err = SSLFreeBuffer( &hashCtx ) ) != 0 )
    goto fail;
if ( ( err = ReadyHash(&SSLHashSHA1, &hashCtx ) ) != 0 )
    goto fail;
if ( ( err = SSLHashSHA1.update( &hashCtx, &clientRandom ) ) != 0 )
    goto fail;
if ( ( err = SSLHashSHA1.update( &hashCtx, &serverRandom ) ) != 0 )
    goto fail;
if ( (err = SSLHashSHA1.update( &hashCtx, &signedParams ) ) != 0 )
    goto fail;
    goto fail;
if ( ( err = SSLHashSHA1.final( &hashCtx, &hashOut ) ) != 0 )
    goto fail;
```

← **Now unconditional**

← **Now unreachable!**

The Apple iPhone SSL Bug

```
if ( ( err = SSLFreeBuffer( &hashCtx ) ) != 0 )
        goto fail;
if ( ( err = ReadyHash(&SSLHashSHA1, &hashCtx ) ) != 0 )
        goto fail;
if ( ( err = SSLHashSHA1.update( &hashCtx, &clientRandom ) ) != 0 )
        goto fail;
if ( ( err = SSLHashSHA1.update( &hashCtx, &serverRandom ) ) != 0 )
        goto fail;
 if ( (err = SSLHashSHA1.update( &hashCtx, &signedParams ) ) != 0 )
{                                                          ← would this have helped?
        goto fail;
                                                           ← Now unreachable!
        goto fail;
}                                                          ← not forgetting this one...
if ( ( err = SSLHashSHA1.final( &hashCtx, &hashOut ) ) != 0 )
                                                           ← Now reachable!
        goto fail;
```

In Summary…

In Summary

- The C Language is in widespread use, despite its limitations
  … many attempts have been made to supplant it, without success

- MISRA C is
  - widely respected as guiding good practice
  - appropriate for use in all high-integrity and high-reliability environments

- Writing high-integrity and high-reliability needs the right language
  … C, on its own, leaves a lot to be desired

- You may not want to start from here, but at least with MISRA C, we offer you a guide.

Need more information?

LDRA.com

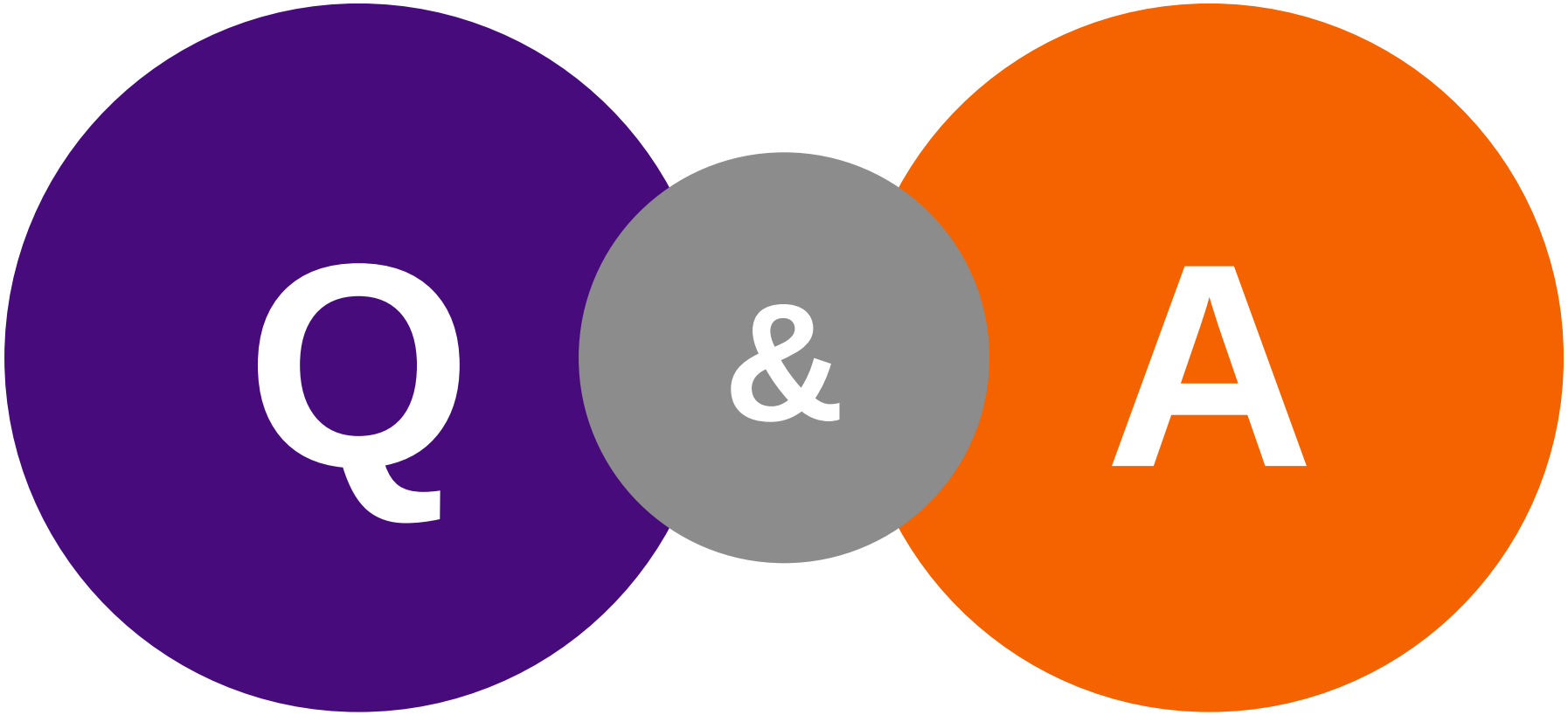
LDRA Software Technology

LDRA Limited

@ldra_technology

LDRA Tools

info@ldra.com


Q
&
A