자동차 SW 테스팅 세미나

SW 테스팅 표준의 활용 방안과 실무 사례

2016.09.29(목)

ISO 26262 – How Effective is the Software Testing?

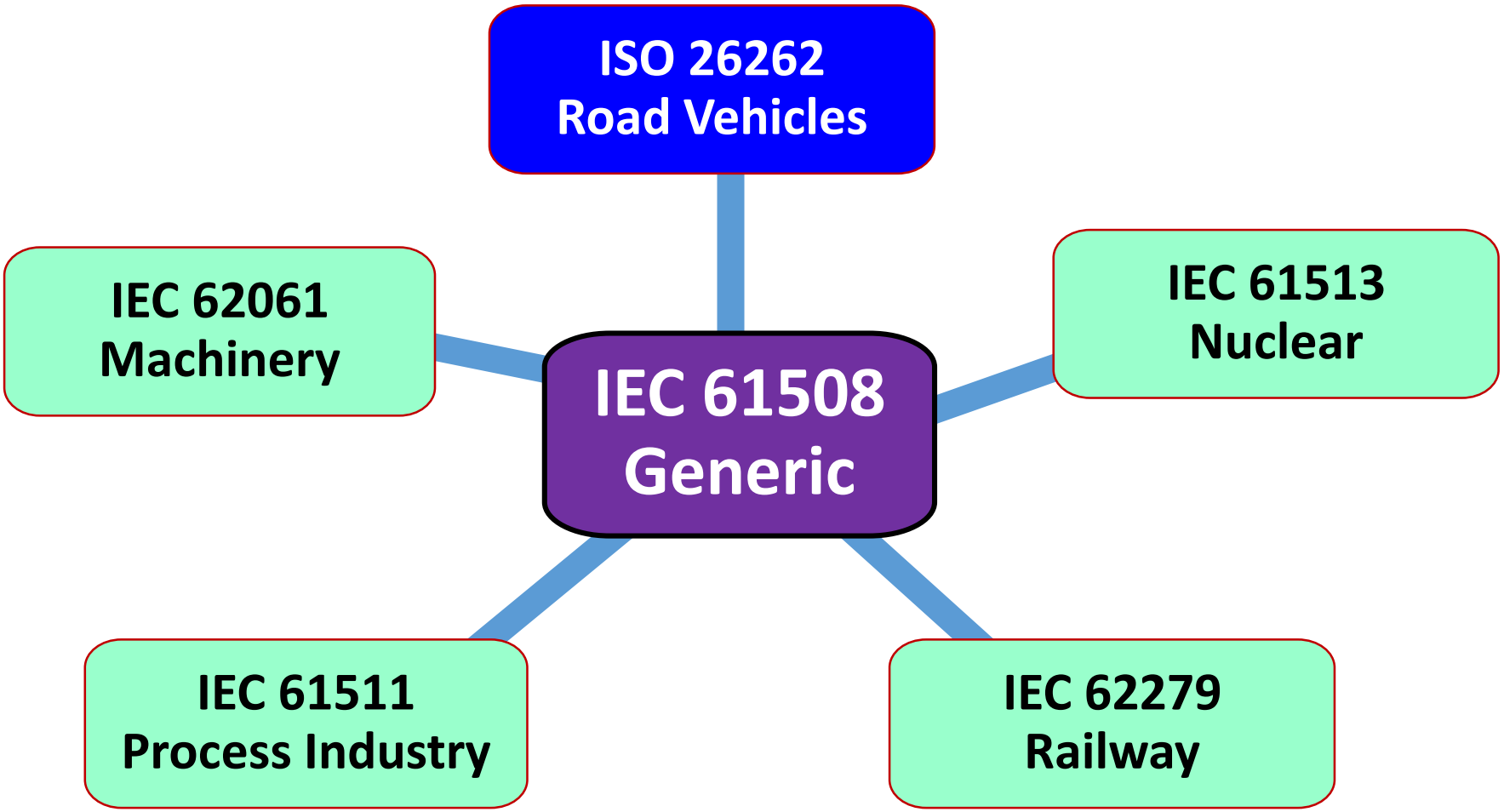
PhD. Stuart Reid / STA테스팅컨설팅

INFINIQ Infinitely yours

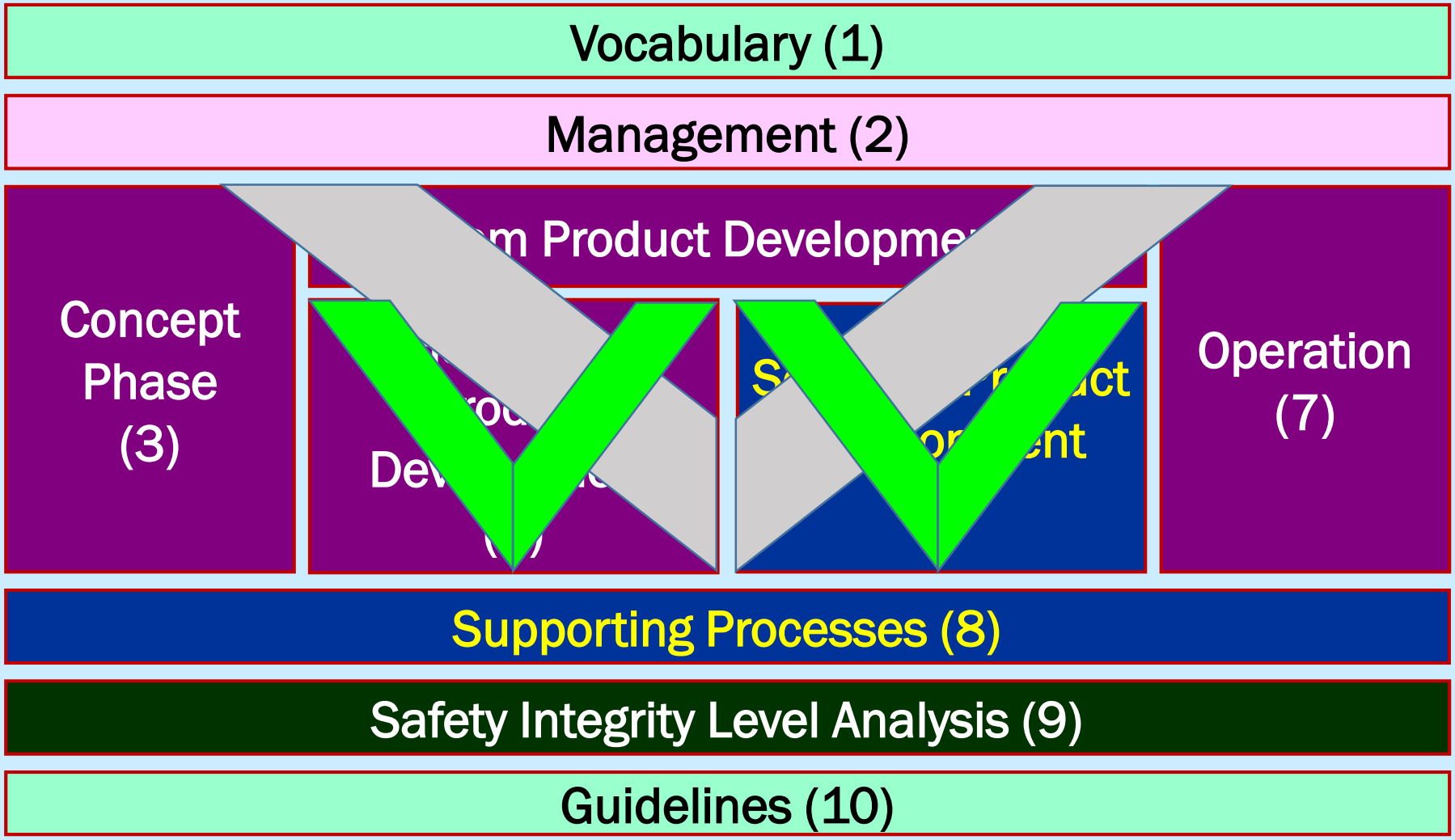
iSTA Global Testing Leader 테스팅컨설팅

CONTENTS

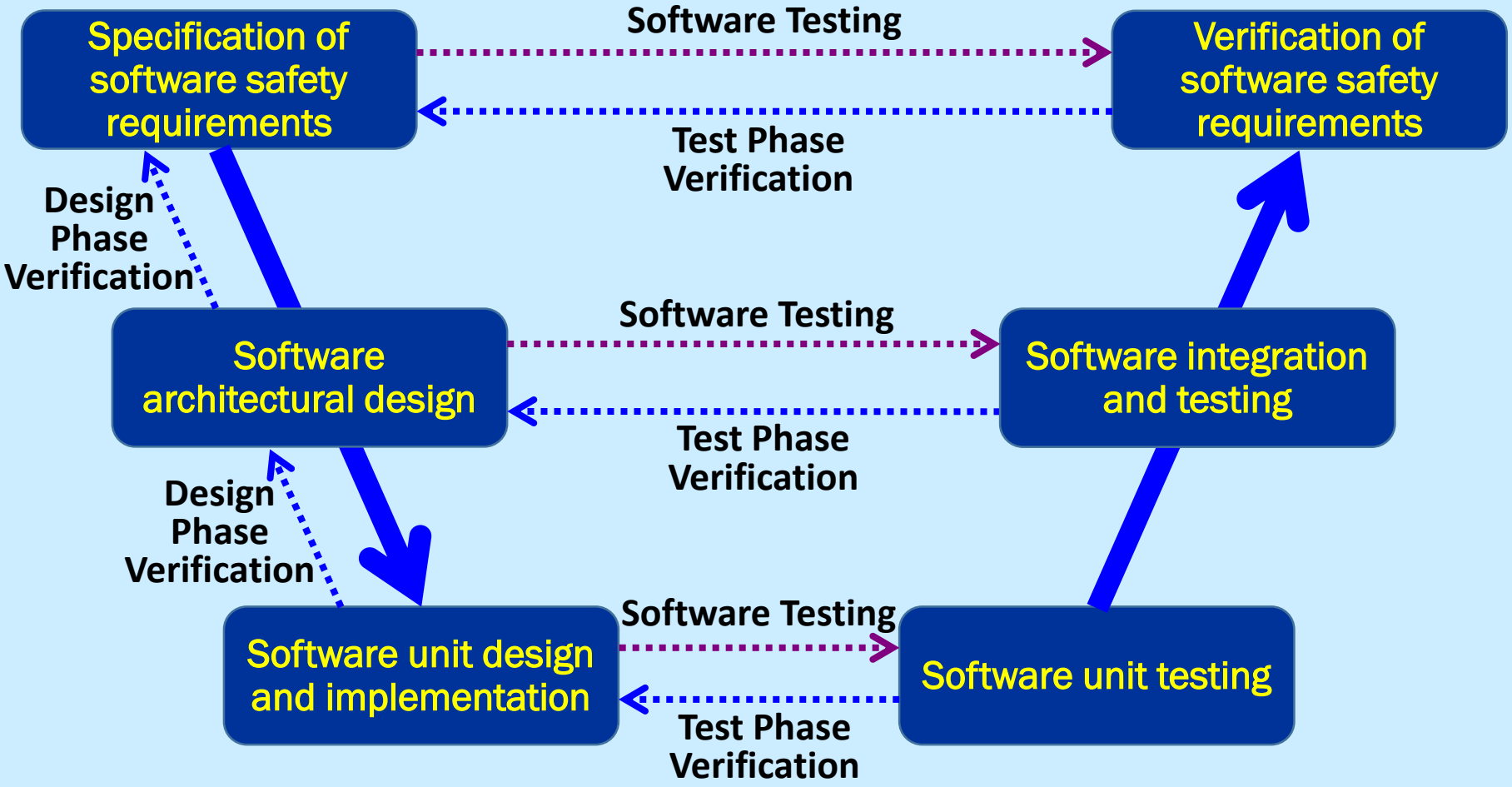
IEC 61508 - Functional safety of systems



ISO 26262 - Overview

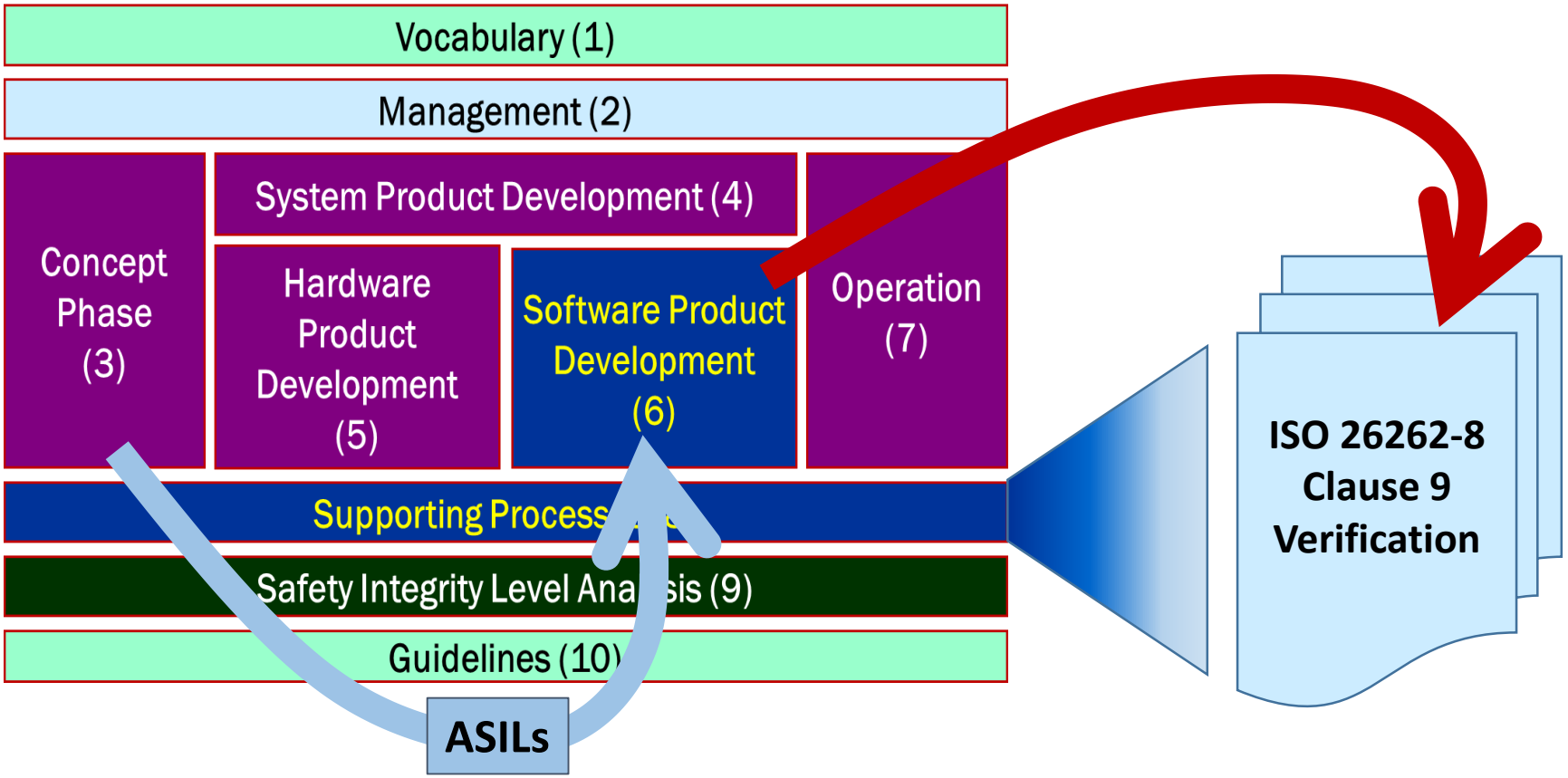


ISO 26262 – Software development process



ISO 26262 Verification

9.4.2/10.4.2/11.4.1 Software unit testing/integration testing/verification of software safety requirements shall be planned, specified and executed in accordance with ISO 26262-8:2011, Clause 9.



ISO 26262 – Safety Integrity Level (ASIL)

- **Severity**
  - S1 – light/moderate injuries
  - S2 – severe/life threatening injuries
  - S3 – life threatening/fatal injuries
- **Probability of exposure**
  - E1 – v. low probability
  - E2 – low probability
  - E3 – medium probability
  - E4 – high probability
- **Controllability**
  - C1 – simply controllable
  - C2 – normally controllable
  - C3 – difficult/uncontrollable

Severity	Probability	Controllability		
		C1	C2	C3
S1	E1	ASILs		
	E2			
	E3			
	E4			
S2	E1			
	E2			
	E3			
	E4			
S3	E1			
	E2			
	E3			
	E4			

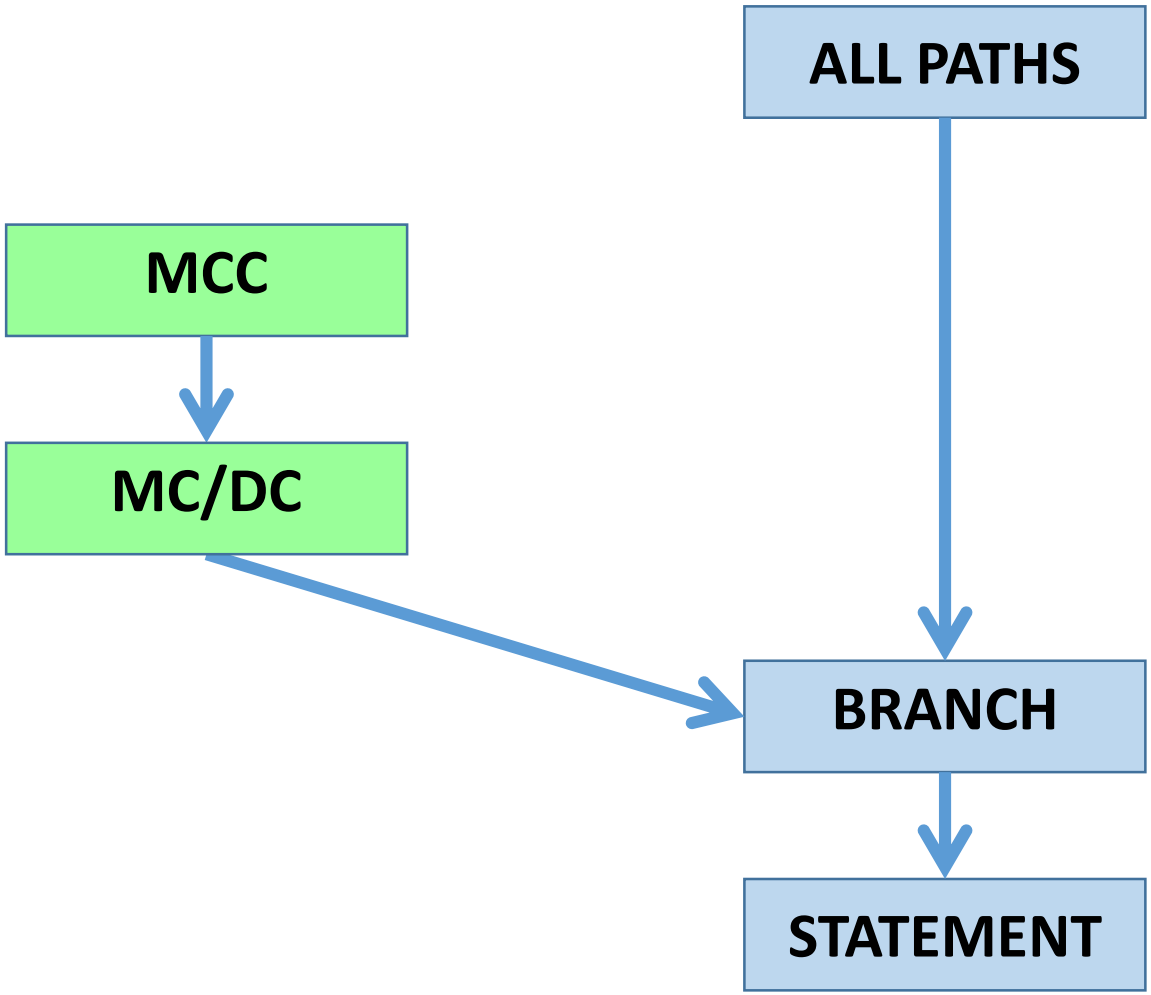
ISO 26262 –Test coverage

Table 12 — Structural coverage metrics at the software unit level

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Statement coverage	++	++	+	+
1b	Branch coverage	+	++	++	++
1c	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++

- Multiple conditions are a known source of defects
  - hence the high recommendation for their coverage for ASIL D
- Achieving 100% MC/DC ensures that all branches and statements are also exercised (it subsumes them)

Structure Testing Techniques Hierarchy – Subsumes Ordering



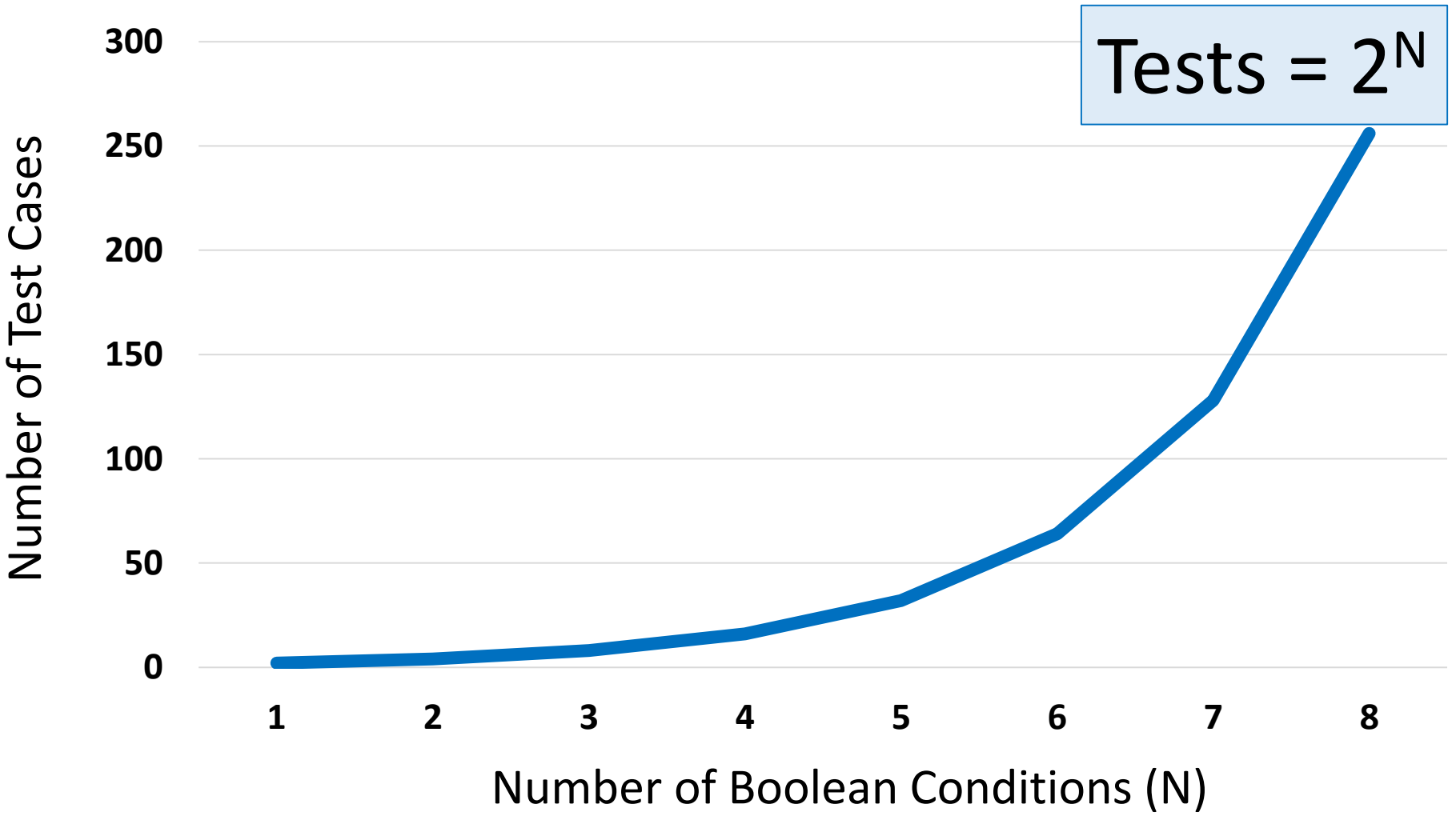
MCC Testing

if (A OR B) AND C then…

Test	COND_1 A	COND_2 B	COND_3 C	OUTCOME (A OR B) AND C
1	TRUE	TRUE	TRUE	TRUE
2	TRUE	TRUE	FALSE	FALSE
3	TRUE	FALSE	TRUE	TRUE
4	TRUE	FALSE	FALSE	FALSE
5	FALSE	TRUE	TRUE	TRUE
6	FALSE	TRUE	FALSE	FALSE
7	FALSE	FALSE	TRUE	FALSE
8	FALSE	FALSE	FALSE	FALSE

Multiple Condition Coverage (MCC) Testing exercises ALL combinations of conditions

Achieving 100% Multiple Condition Coverage



Testing of Multiple Conditions (MCC)

- As we can see, the number of tests required to achieve 100% multiple condition coverage (MCC) can be prohibitive…
- …so instead…
- DO-178B (the avionics software standard) first required the use of MC/DC (published 1992)
- MC/DC is now required by:
  - avionics DO-178C for the most critical software (Level A)
  - IEC 61508 (generic safety standard) for SIL 4
  - ISO 26262-6 for ASIL D

Modified Condition Decision Testing

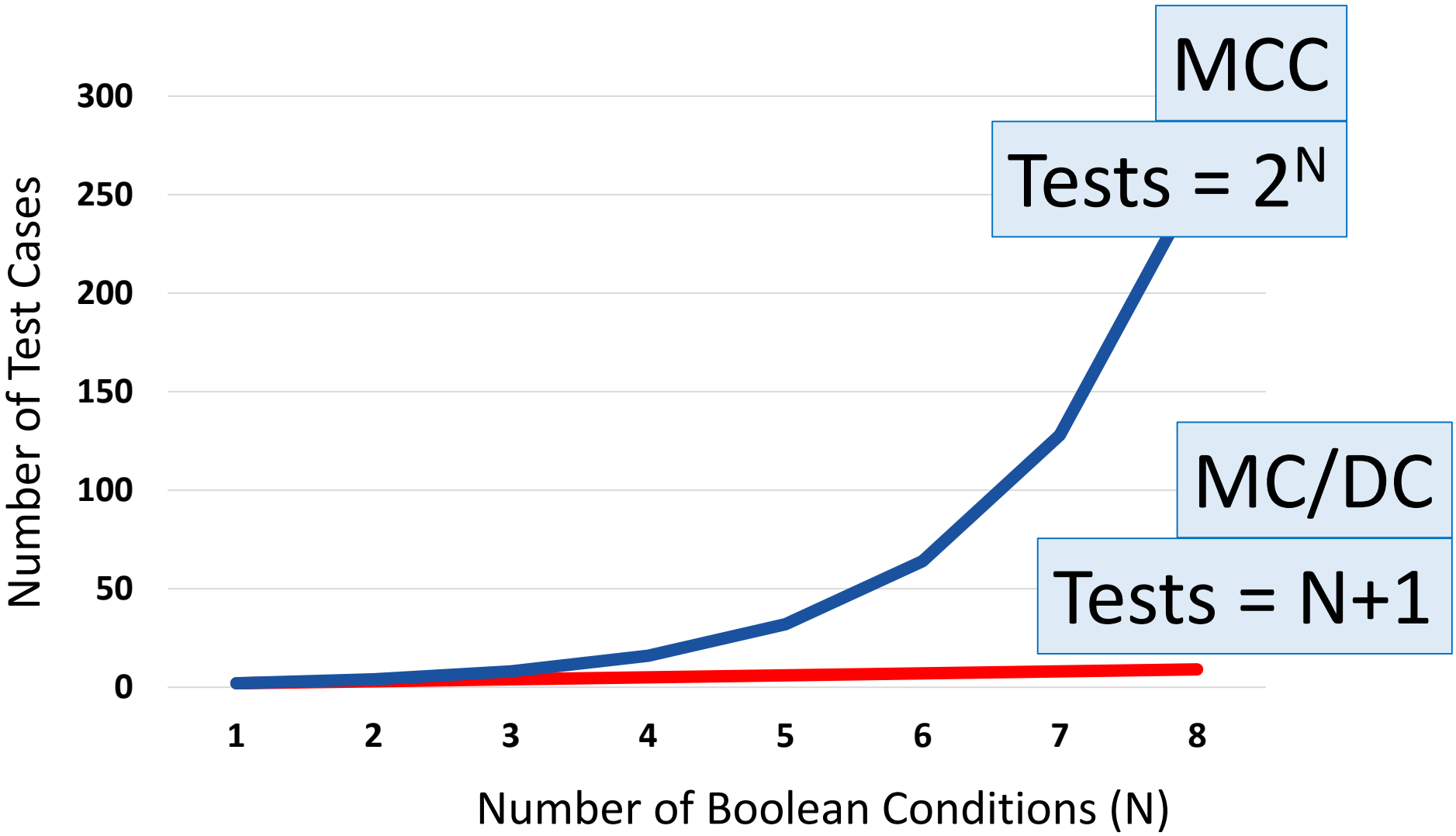
- ISO 29119-4, 2015
  - Test cases shall be designed to demonstrate that Boolean operands within a decision condition can independently affect the outcome of the decision
- An assumed benefit of MC/DC is that it requires a much smaller number of test cases than for multiple condition coverage (MCC), while sustaining a quite high defect-detection probability
- The safety-related standards define MC/DC only at 100% - there are no lower levels possible - you either achieve it or not

MC/DC Testing Example

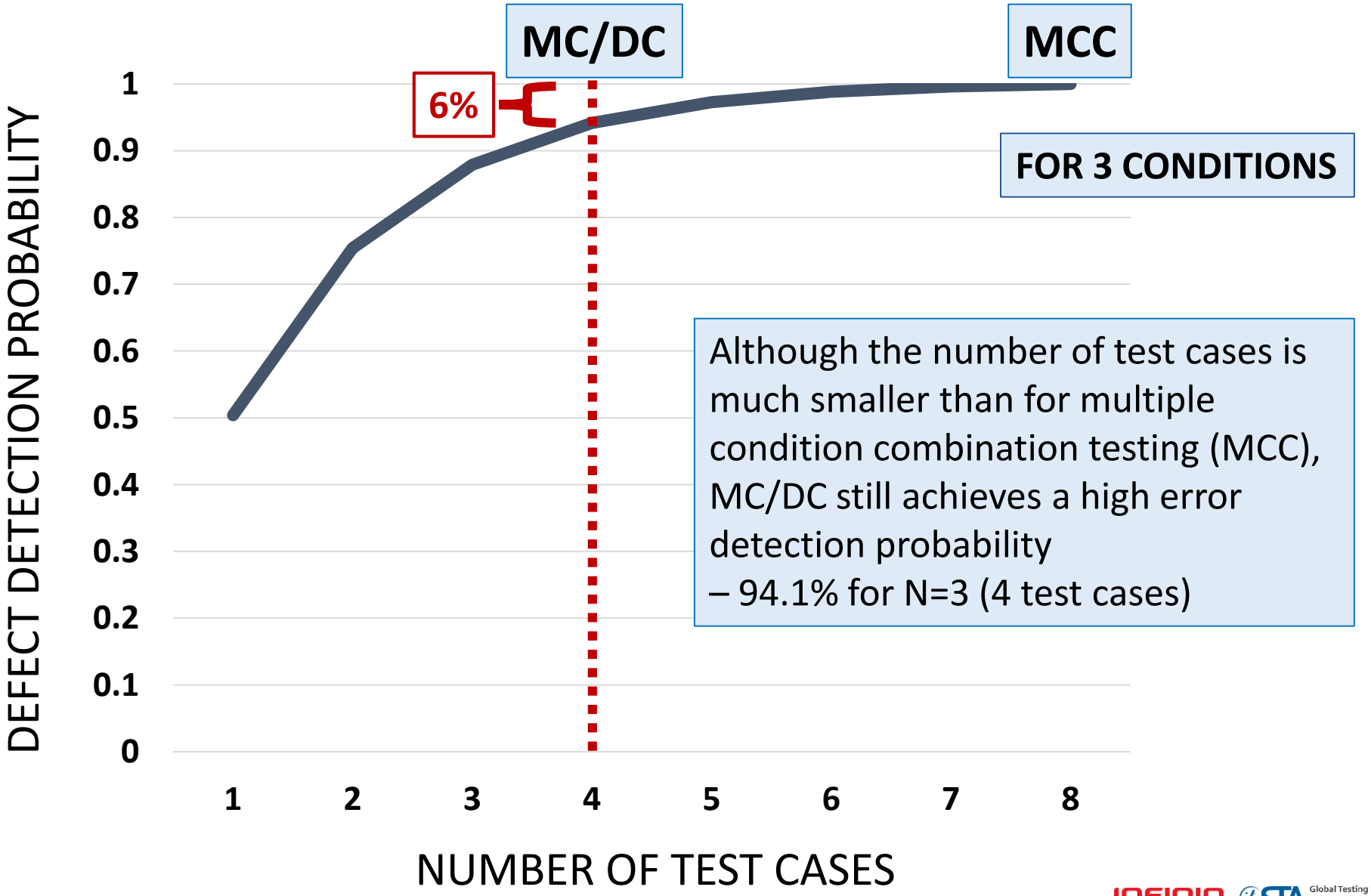
if (A OR B) AND C then…

Test	COND_1 A	COND_2 B	COND_3 C	OUTCOME (A OR B) AND C
	TRUE	TRUE	TRUE	TRUE
	TRUE	TRUE	FALSE	FALSE
	TRUE	FALSE	TRUE	TRUE
	TRUE	FALSE	FALSE	FALSE
	FALSE	TRUE	TRUE	TRUE
	FALSE	TRUE	FALSE	FALSE
	FALSE	FALSE	TRUE	FALSE
	FALSE	FALSE	FALSE	FALSE
1	TRUE	FALSE	TRUE	TRUE
2	FALSE	TRUE	TRUE	TRUE
3	FALSE	TRUE	FALSE	FALSE
4	FALSE	FALSE	TRUE	FALSE

#Tests - MC/DC vs MCC



Multiple Condition Testing – Effectiveness vs Test Cases



MC/DC – Practical Issues – Independent Variables

- Sometimes the conditions in a decision are not independent:

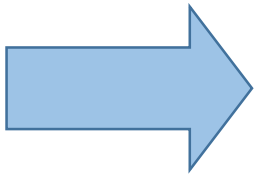
- if (A OR B) AND (A OR B) then
  do W
  do Y

- Now we cannot independently affect the outcome of the decision by varying each condition while keeping the rest the same – as A appears twice when we change it the other instance of A will also change

MC/DC – Practical Issues – Temporary Variables

- Developers (who have to test) may be tempted to move the logic away from the decision
  - deliberately (naughty programmers); or
  - accidentally

- if (A OR B) AND C then
  do W
  do Y



```
temp = (A OR B) AND C
  if temp then
    do W
    do Y
```

MC/DC – Practical Issues – Short-circuit evaluation

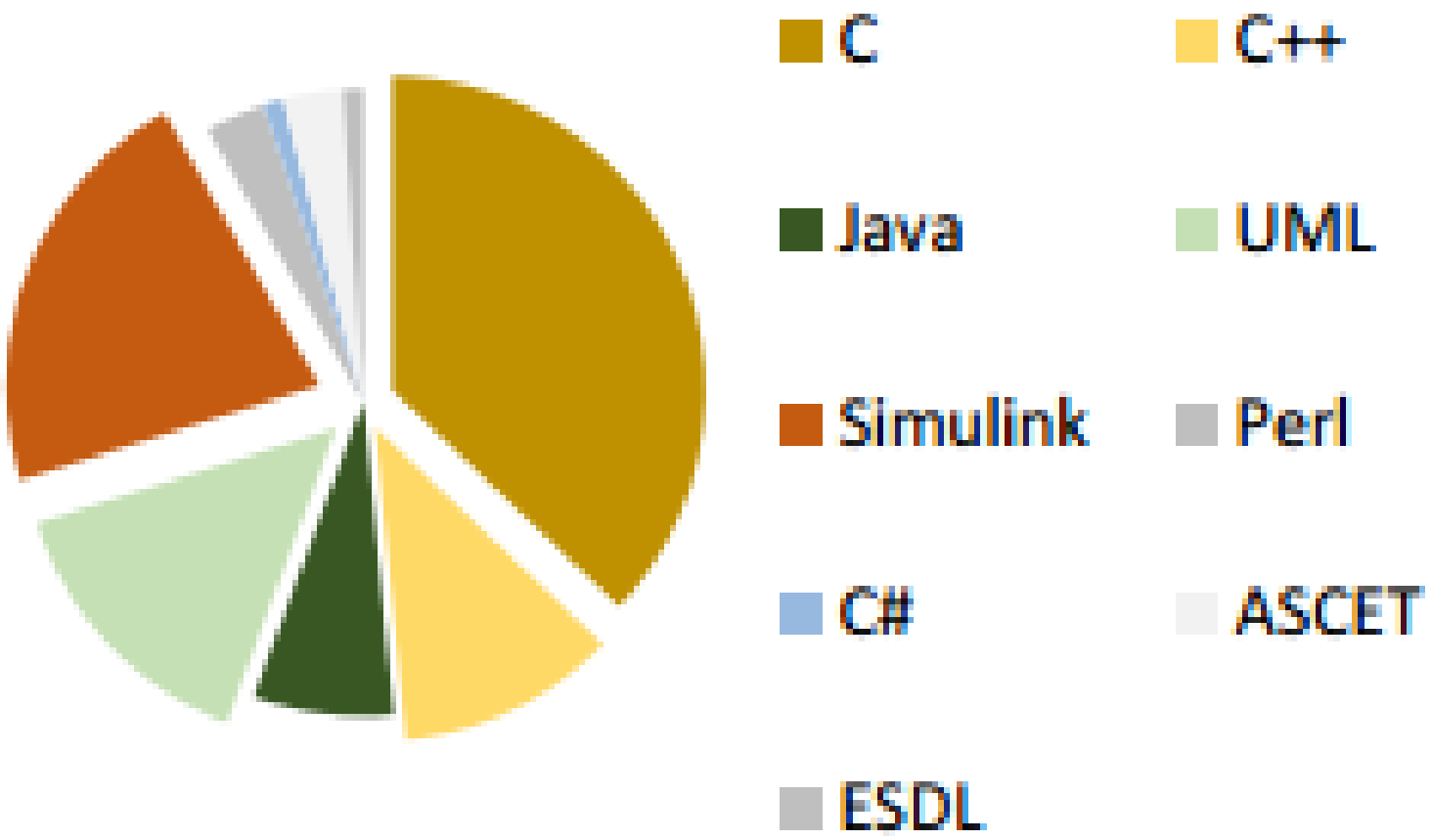
- Short-circuit evaluation is used on the semantics of applicable Boolean operations in some programming languages (notably C)
  - the second argument is evaluated only if the first argument does not determine the value of the expression on its own

- For instance,
  - if X OR Y then
  - whenever X is evaluated as TRUE then we can 'short-circuit' the evaluation of Y as whatever its value (TRUE or FALSE) the overall result (TRUE) will be the same

  - if P AND Q then
  - whenever P is evaluated as FALSE then we can 'short-circuit' the evaluation of Q as whatever its value (TRUE or FALSE) the overall result (FALSE) will be the same

VDA – Languages in the Automotive Industry, May 2016

"C, a technology from the 1960s, is the absolute champion-language in modern automotive industry."
– VDA, 2016

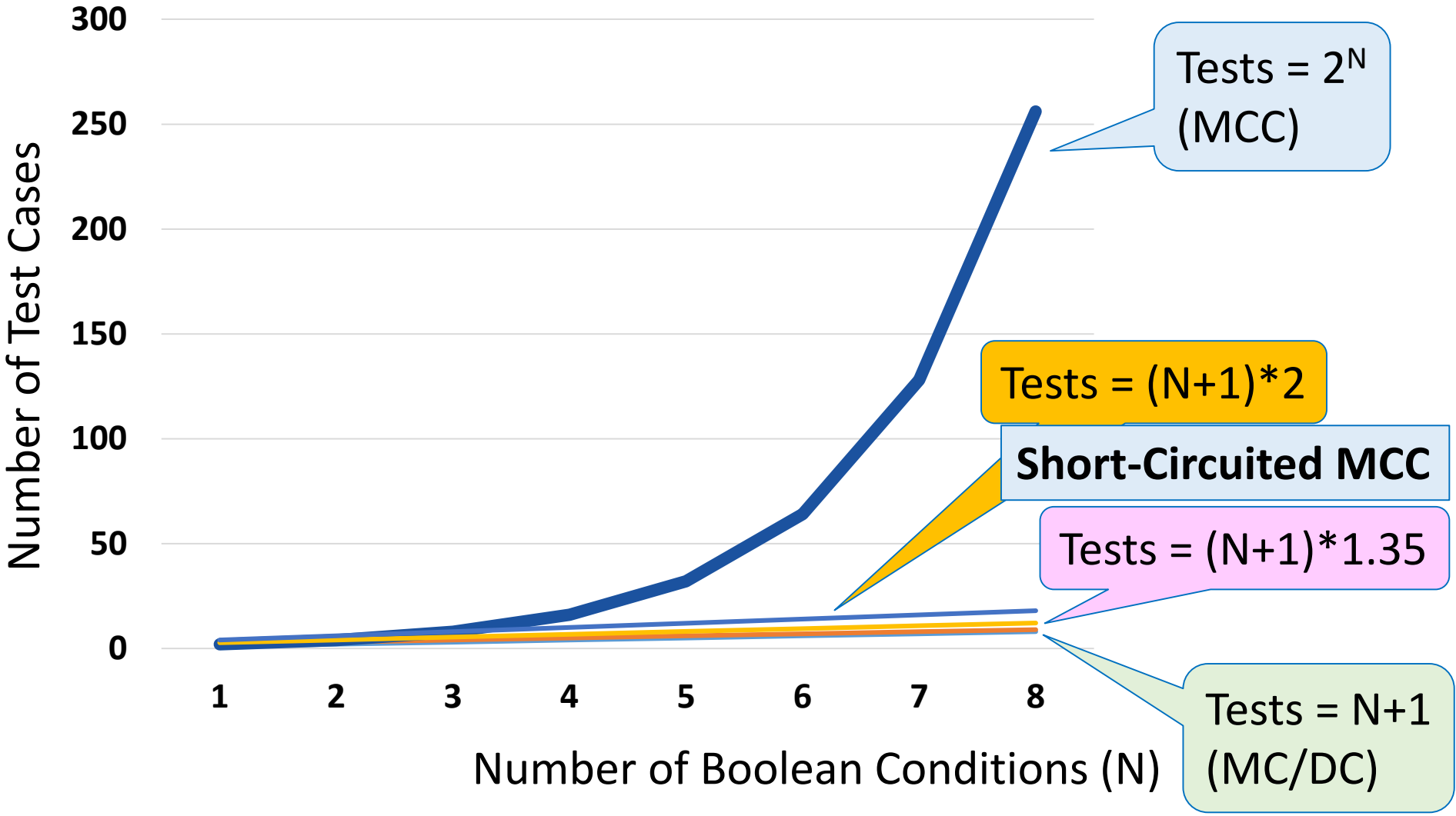


MC/DC – Practical use for Automotive C code

- With short-circuit evaluation the number of test cases for MCC is much smaller because many redundant test cases occur
- Based on case studies, the number of tests required to achieve MCC is (on average) only about 35% higher than the number required to achieve MC/DC
  - the maximum overhead is approximately 100% (for decisions with 5 conditions)
- "Considering the lower error-detection effectiveness of MC/DC compared to MCC, we conclude with the strong recommendation to use MCC as a coverage metric for testing safety-relevant software (with a limited number of conditions) implemented in programming languages with short-circuit evaluation."

Reasonability of MC/DC for safety-relevant software implemented in programming languages with short-circuit evaluation, Computing (2015)

Number of Tests - MC/DC vs MCC (and short-circuited MCC)



MC/DC - Evidence from DO-178B (avionics experience)

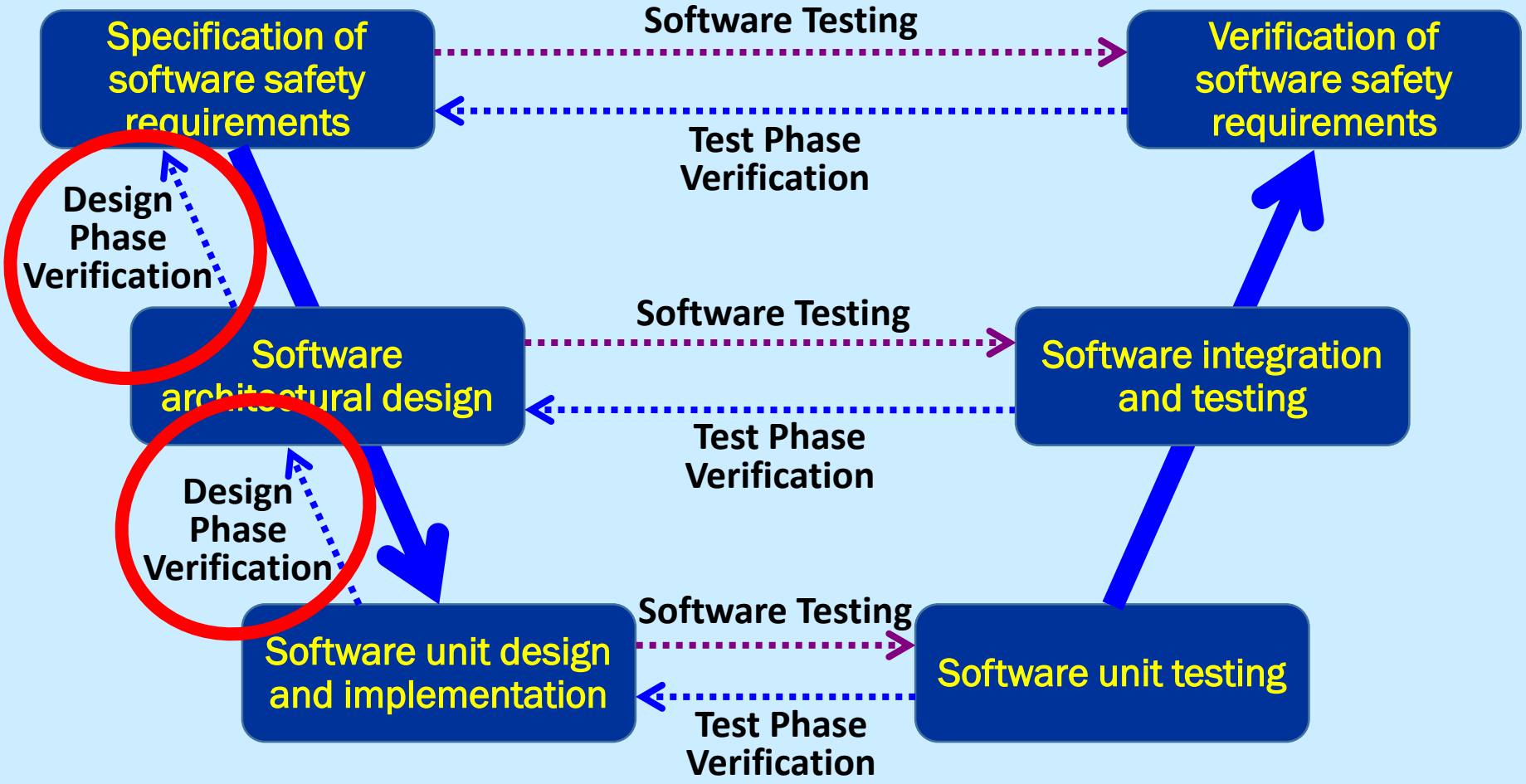
- For DO-178B, a detailed comparison of systems certified to Levels A or B showed that there was no discernible difference between the two levels in the remaining level of serious anomalies in the software
- The main difference between Level A (catastrophic failure) and Level B (severely hazardous) is that Level A requires MC/DC coverage of the software
- So, for the studied systems, MC/DC did not significantly increase the probability of detecting serious defects in the software
- 71% of respondents stated that MC/DC rarely or never revealed defects

German & Mooney, 2001, "Air vehicle software static code analysis— Lessons learnt," Proceedings of the Ninth Safety-Critical Systems Symposium.

MC/DC - Recommendations

- Multiple Condition Coverage (MCC) testing
  - subsumes MC/DC testing
  - finds more bugs
  - is simpler to understand – and perform correctly
  - so use MCC (instead of MC/DC)
    - unless there are more than 5 conditions in a decision – and short-circuiting doesn't apply
- Use tools to measure condition coverage
- As testers, be aware that it is possible for programmers to 'cheat' the tools by moving the multiple conditions into temporary variables

ISO 26262 – Software development process



ISO 26262-1 – Definitions

- verification review
  - verification activity to ensure that the result of a development activity fulfils the project requirements, or technical requirements, or both
    - NOTE 1 Individual requirements on verification reviews are given in specific clauses of individual parts of ISO 26262.
    - NOTE 2 The goal of verification reviews is technical correctness and completeness of the item or element with respect to use cases and failure modes.
    - EXAMPLE Technical review; walk-through; inspection.
- review
  - examination of a work product, for achievement of the intended work product goal, according to the purpose of the review
    - NOTE Reviews can be supported by checklists.

ISO 26262 Review Requirements - Architecture

Table 6 — Methods for the verification of the software architectural design

	Methods	ASIL			
		A	B	C	D
1a	Walk-through of the design[a]	++	+	o	o
1b	Inspection of the design[a]	+	++	++	++
1c	Simulation of dynamic parts of the design[b]	+	+	+	++
1d	Prototype generation	o	o	+	++
1e	Formal verification	o	o	+	+
1f	Control flow analysis[c]	+	+	++	++
1g	Data flow analysis[c]	+	+	++	++

[a] In the case of model-based development these methods can be applied to the model.

ISO 26262 Review Requirements – Unit Design & Implementation

Table 9 — Methods for the verification of software unit design and implementation

	Methods	ASIL			
		A	B	C	D
1a	Walk-through[a]	++	+	o	o
1b	Inspection[a]	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis[b,c]	+	+	++	++
1f	Data flow analysis[b,c]	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis[d]	+	+	+	+

[a] In the case of model-based software development the software unit specification design and implementation can be verified at the model level.

ISO 26262-1 - Definitions

- Walkthrough
  - systematic examination of work products in order to detect anomalies
    - EXAMPLE During a walk-through, the developer explains the work product step-by-step to one or more assessors. The objective is to create a common understanding of the work product and to identify any anomalies within the work product.
    - Both inspections and walk-throughs are types of peer review, where a walk-through is a less stringent form of peer review than an inspection.
    - NOTE Any anomalies that are detected are usually addressed by rework, followed by a walk-through of the reworked work products
- Inspection
  - examination of work products, following a formal procedure, in order to detect anomalies
    - NOTE Any anomalies that are detected are usually addressed by rework, followed by re-inspection of the reworked products
    - NOTE A formal procedure normally includes a previously defined procedure, checklist, moderator and review of the results.

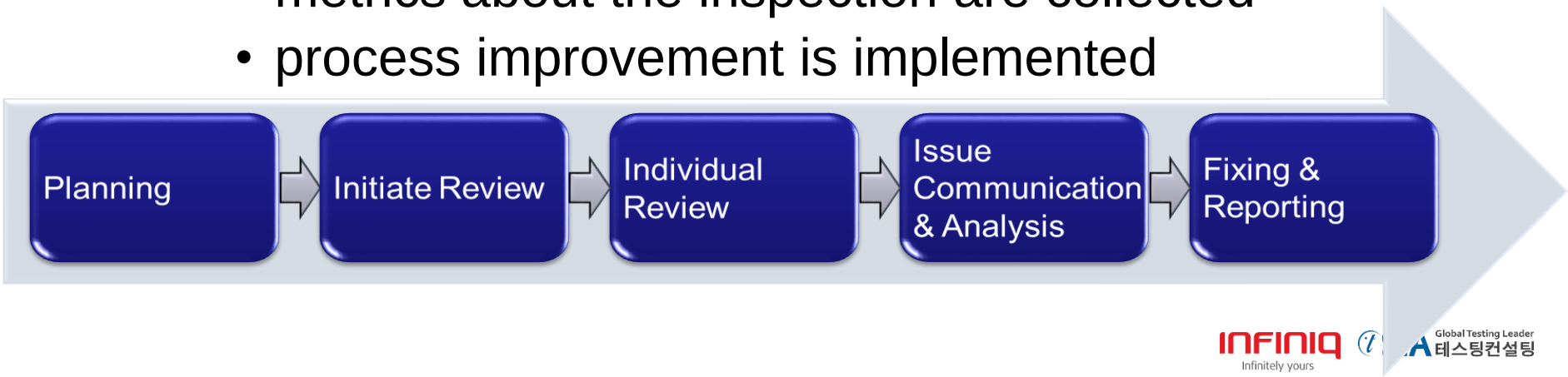
ISO 26262 – Inspections and Walkthroughs

- Walkthroughs
  - to identify anomalies
  - systematic
  - the developer explains the work product step-by-step to one or more assessors [so there must be a review meeting, but only two people may be involved]
    - to create a common understanding
  - detected anomalies are usually addressed by rework, followed by a walkthrough of the reworked products
  - can be supported by checklists
  - less stringent than an inspection

- Inspections
  - to detect anomalies
  - following a formal procedure, including:
    - moderator [so there *must* be a review meeting]
    - review of the results
    - detected anomalies are usually addressed by rework, followed by re-inspection of the reworked products
  - normally includes a checklist
  - more stringent than a walkthrough

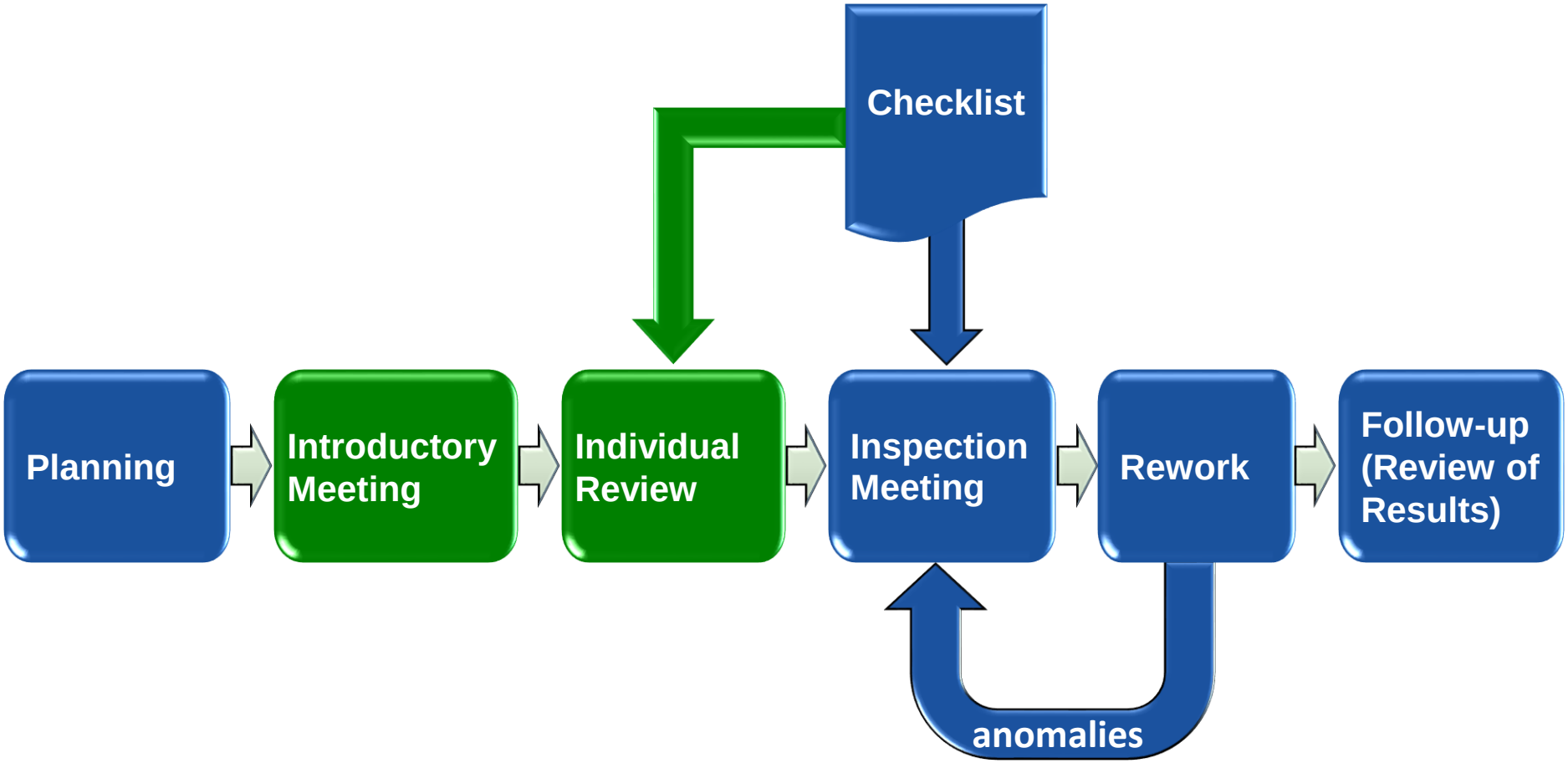
Inspections vs Walkthroughs – ISO 20246

- According to ISO 20246 – Work Product Review (DIS)
- Inspections require (when walkthroughs do not):
  - a moderator
  - the author cannot lead the review meeting
  - individual reviews, with documented issues
  - entry criteria (e.g. passing prior informal reviews, and provision of documents)
  - issues are documented
  - metrics about the inspection are collected
  - process improvement is implemented

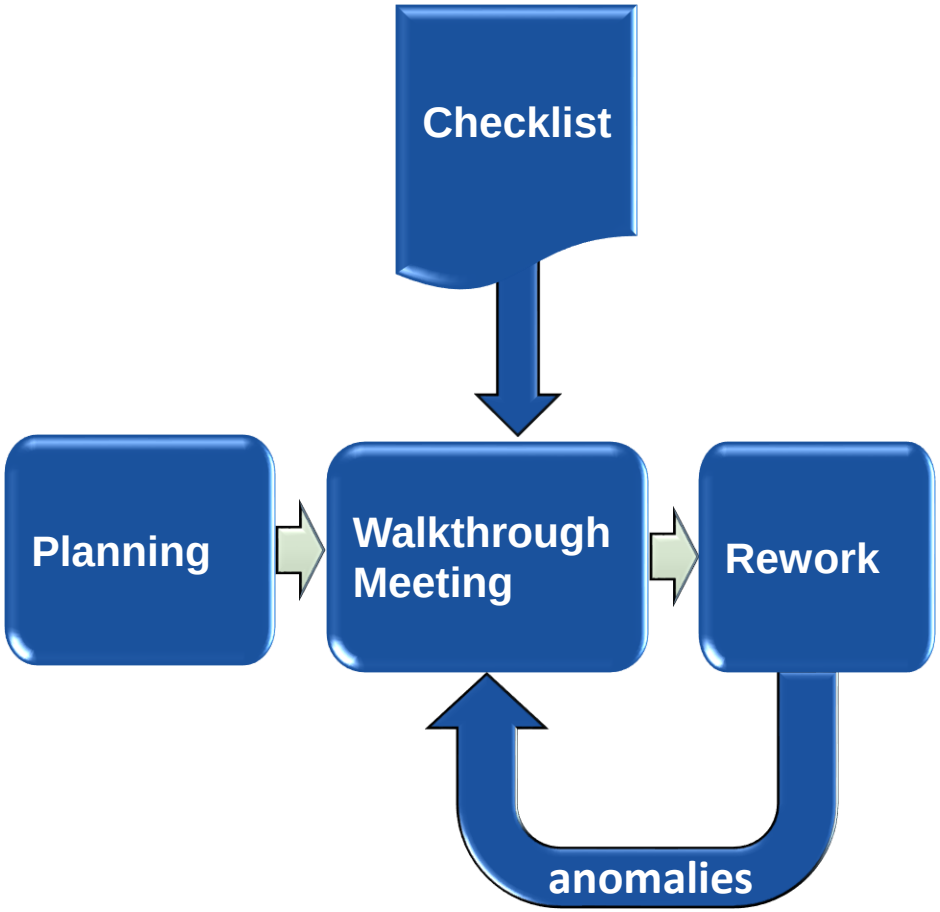


How to review for ISO 26262?

ISO 26262 Inspection Process?



ISO 26262 Walkthrough Process?



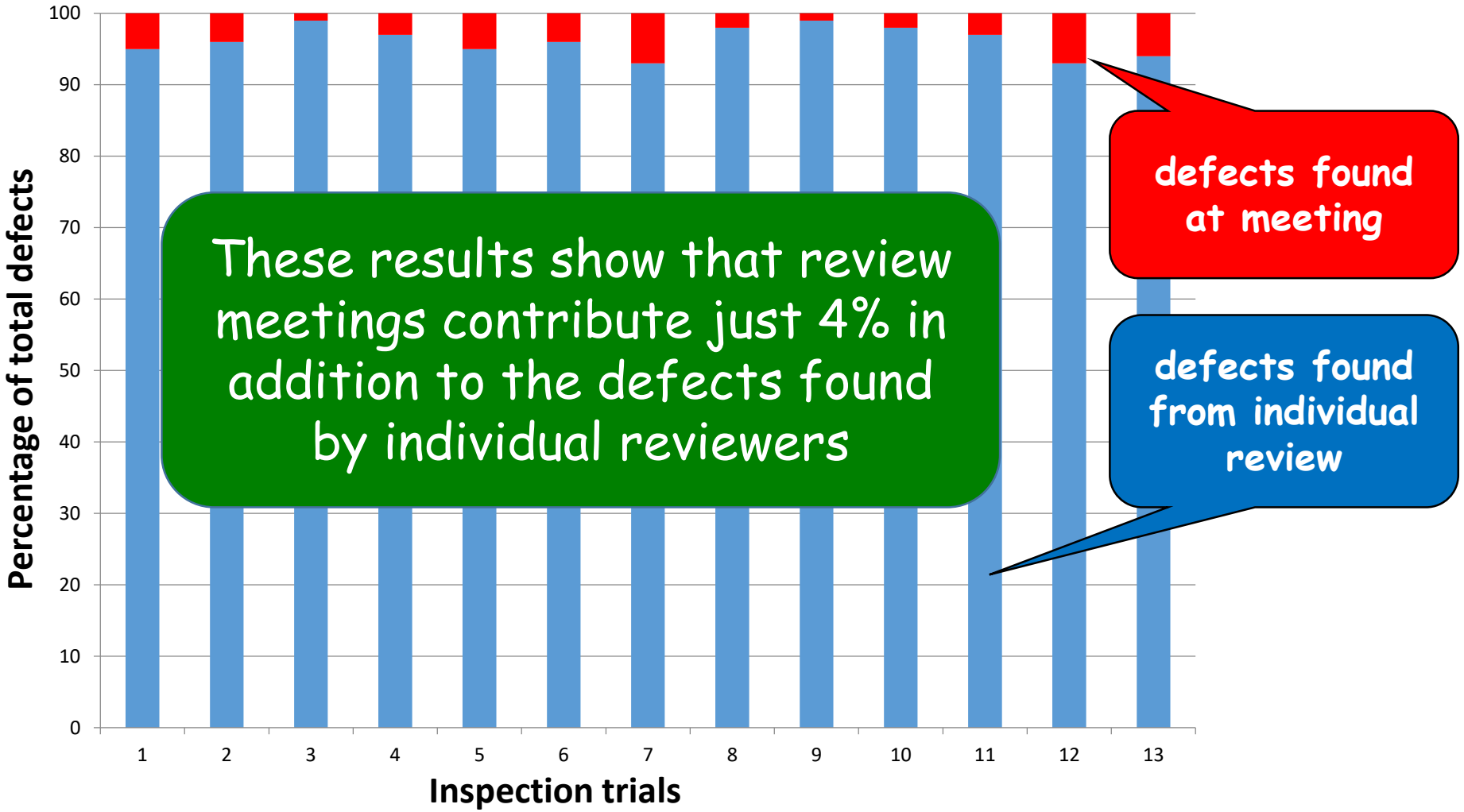
Do we need review meetings?

- “Typical meeting-based review methods are neither more effective nor less effective than non-meeting-based review methods with respect to defect detection effectiveness.

  In fact, the non-meeting inspections found more defects…”
  - [Reference: Porter and Johnson, 1997]

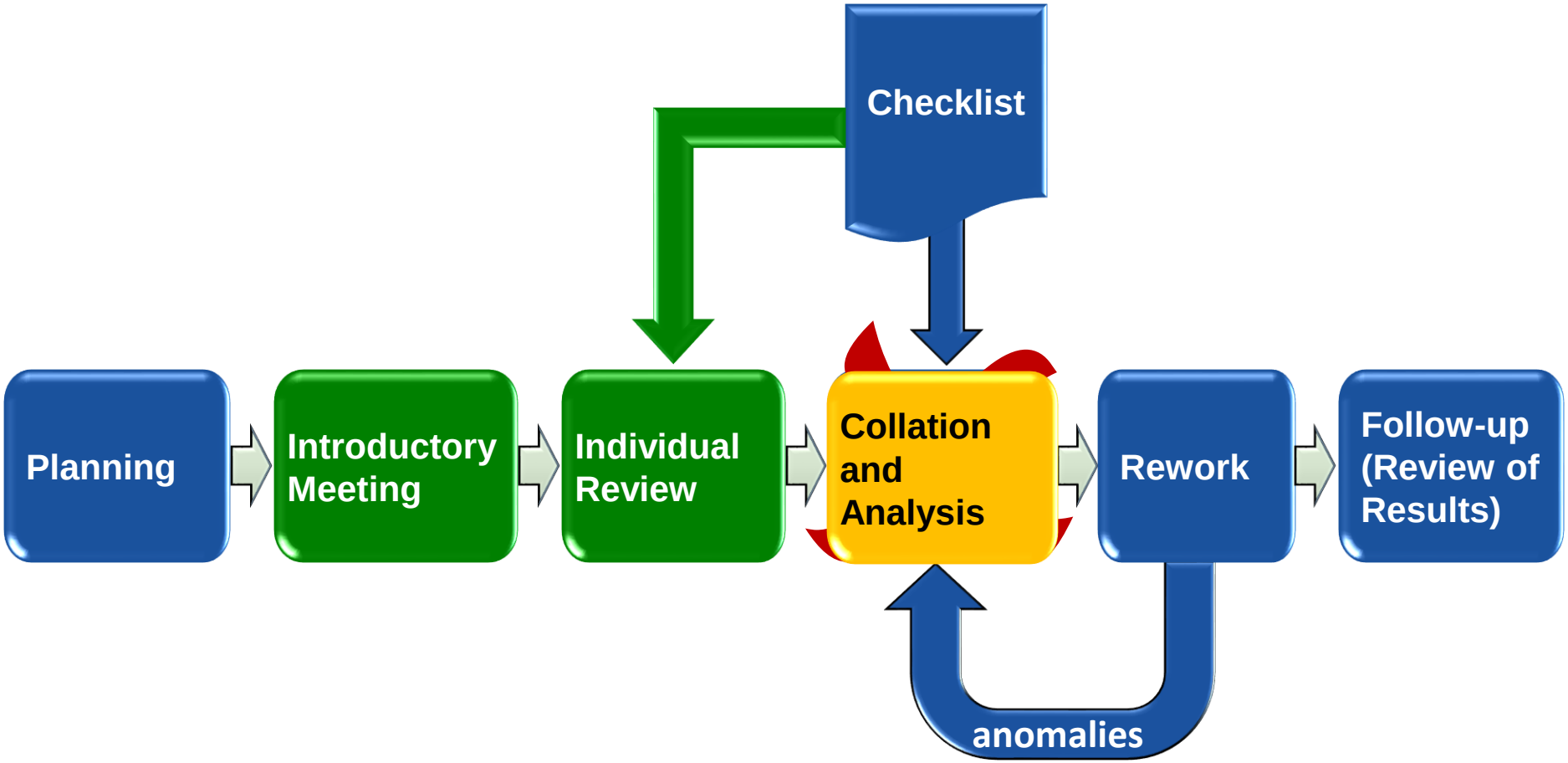
- Several studies have reported results that support the claim that individual preparation for inspections is the most important element contributing to the effectiveness of the inspection
  - [References: Christenson, 1990; Laitenberger, 2002]

Meeting effectiveness vs Individual Review?



Results from a study by Lawrence Votta – Does every inspection need a meeting? Proc ACM SIGSOFT, LA, Dec 1993.

ISO 26262 Inspection Process?



Problems with Checklists

- Tunnel vision
  - Only defects on the checklist are detected (hard-to-find defects requiring deep understanding are often missed)
- Based on the past
  - Checklists only contain defects that have been found before
- Checklist Authors/Reviewers
  - Checklists are only as good as the person writing them
  - Do the reviewers understand the questions?
  - Are the checklists too long?
  - Are the checklists maintained?

- “The Checklist method was no more effective than the Ad Hoc detection method”
  - [Reference: Porter, 1995]

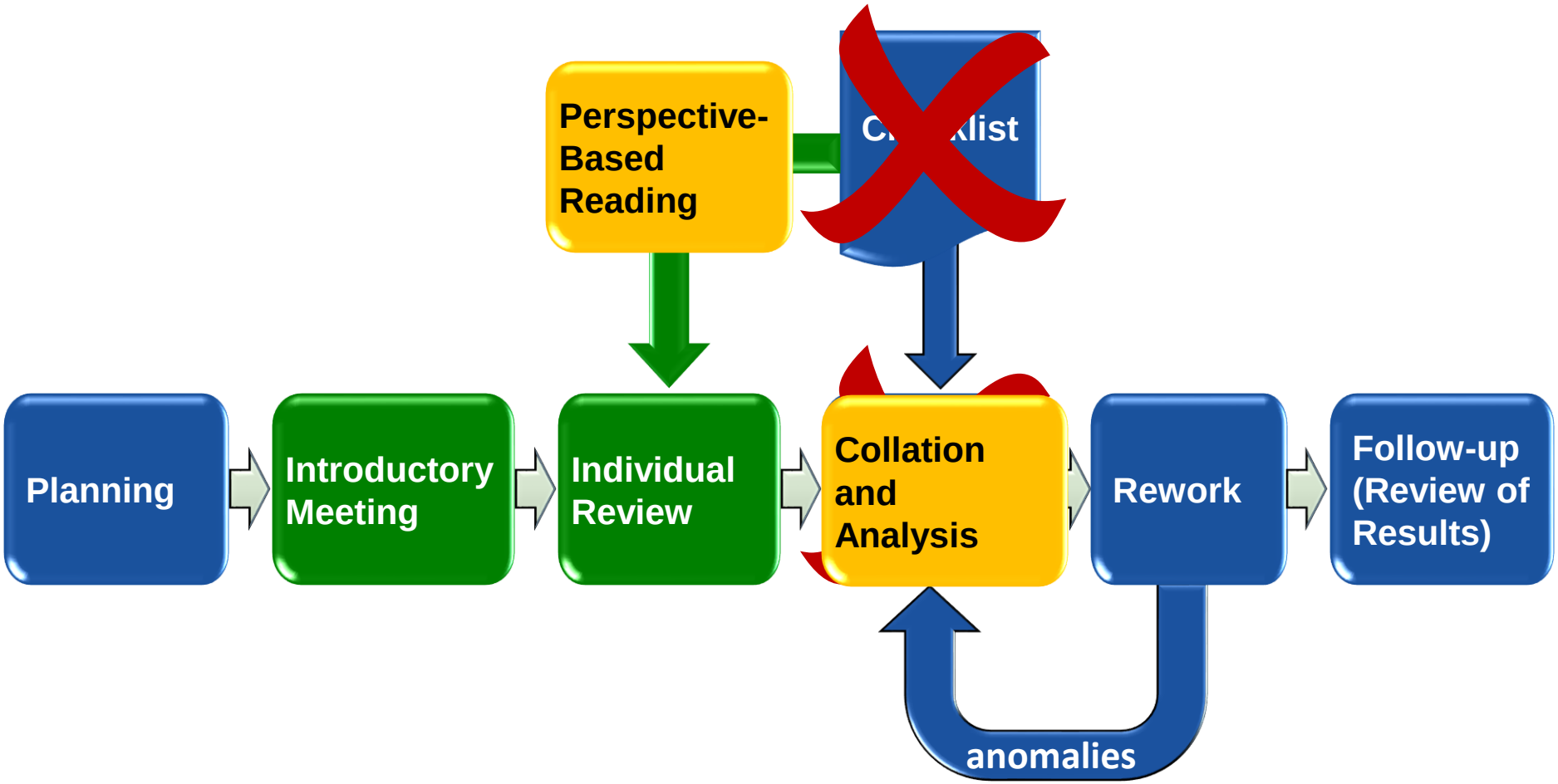
ISO 26262 – Checklists vs Perspective-Based Reading (PBR)

- "The majority of the results indicate that advanced reading techniques such as perspective-based reading can find more defects and are **more cost-effective** than ad-hoc reading and checklist-based reading."
  - [Reference: Lahtine, 2011]
- "Perspective-based reading was statistically found to be **more effective** than Checklist-Based Reading"
  - [Reference: Laitenberger, 2000]
- "The fault detection rate when using scenarios was **superior** to that obtained with Ad Hoc or Checklist methods."
  - [Reference: Porter, 1995]

"50% OF REVIEWERS USE CHECKLISTS"

Is this why its in the standard?

Recommended ISO 26262 Inspection Process



ISO 26262 – Deriving Test Cases

	Methods	ASIL			
		A	B	C	D
1a	Analysis of requirements	++	++	++	++
1b	Generation and analysis of equivalence classes[a]	+	++	++	++
1c	Analysis of boundary values[b]	+	++	++	++
1d	Error guessing[c]	+	+	+	+

[a] Equivalence classes can be identified based on the division of inputs and outputs, such that a representative test value can be selected for each class.

[b] This method applies to interfaces, values approaching and crossing the boundaries and out of range values.

[c] Error guessing tests can be based on data collected through a "lessons learned" process and expert judgment.

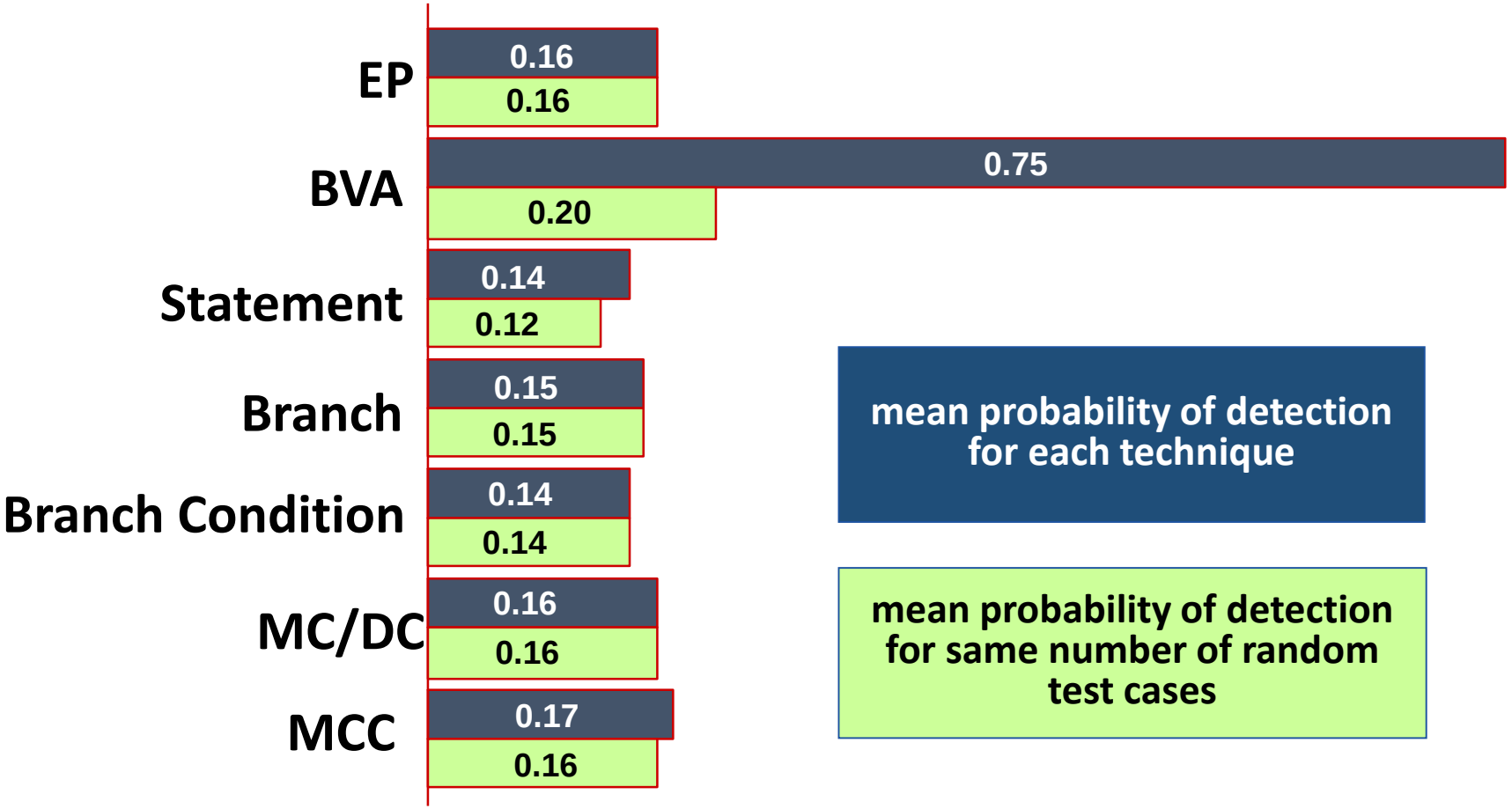
Table 11 — Methods for deriving test cases for software unit testing

Table 14 — Methods for deriving test cases for software integration testing

ISO 26262 – Deriving Test Cases

- "Analysis of Requirements"
  - this is NOT a test design technique – ALL test design techniques (except random testing) require an analysis of the requirements (even white box)
- Equivalence Partitioning – see ISO/IEC/IEEE 29119-4
- Boundary Value Analysis – see ISO/IEC/IEEE 29119-4
  - subsumes equivalence partitioning (except very rarely)
- Error Guessing – see ISO/IEC/IEEE 29119-4
  - this is NOT measurable
  - highly-dependent on the tester's experience

Experimental Results - Mean Probability of Detection



Conclusions

- For ASIL D, where MC/DC is ‘highly-recommended’ seriously consider the use of MCC instead
- If you use MC/DC be aware of temporary variables
- When required to use ‘Inspections’ (for ASIL B, C and D) be sure to use an optimal approach…
  - use ISO/IEC 20246 to design the review process
  - consider replacing the ‘Inspection Meeting’
  - consider replacing the ‘Checklist-Based’ approach
- For black box testing the preferred approach should be Boundary Value Analysis
  - for all ASILs

Thank you