Automotive System Safety Engineering Practitioner Knowledge

Joseph D'Ambrosio

ISO 26262 Automotive Functional Safety

Technical Expert

(ISO/TC22/SC3/WG16 Member)

Lab Group Manager

GM Research Laboratories

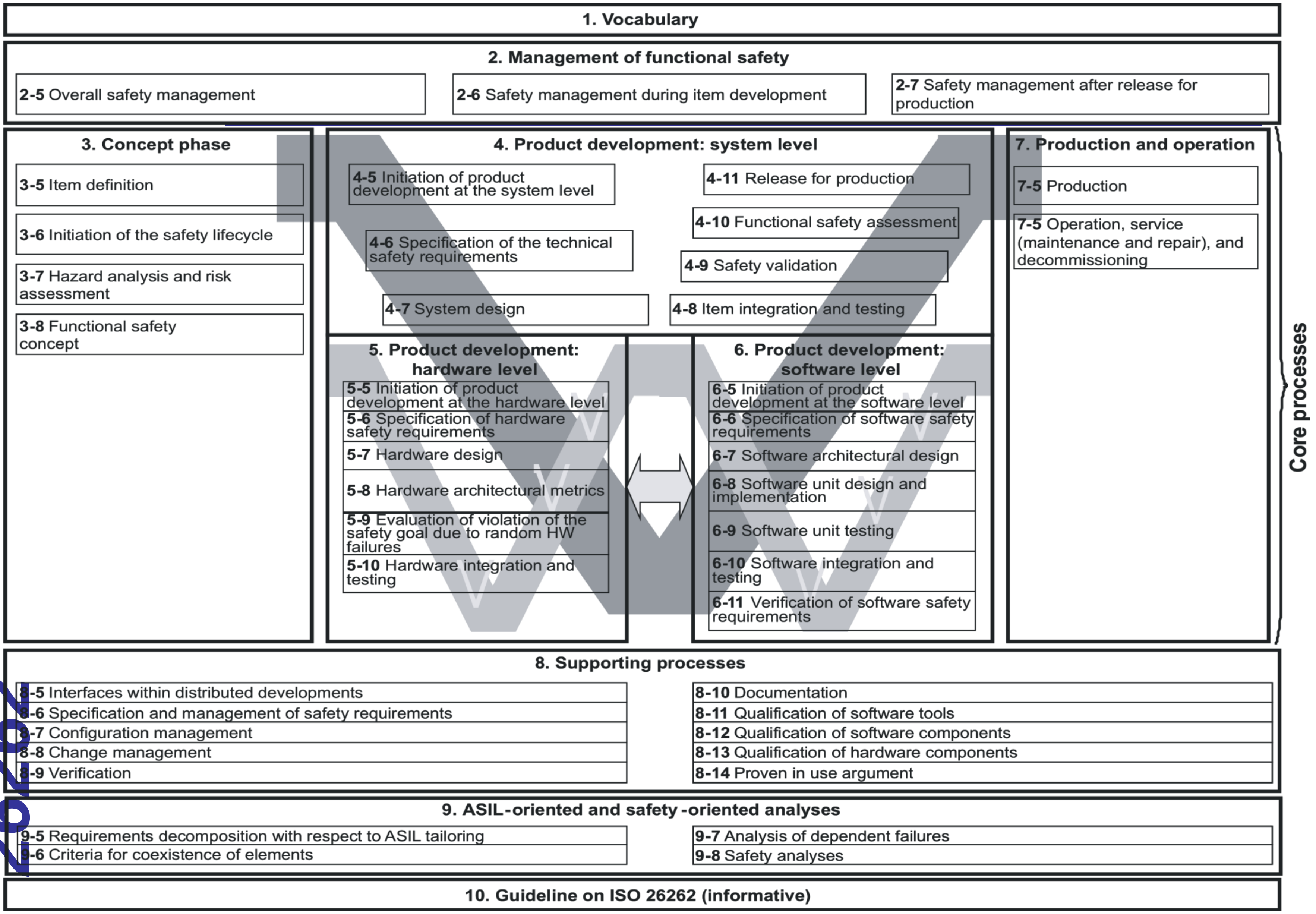
Outline

- ISO 26262 Overview
- ISO 26262 Competence Management
- ISO 26262 Safe SW Development

What is ISO 26262?

- Adaptation of IEC 61508 to comply with the specific needs of E/E systems within road vehicles
  - Specifies a functional safety life-cycle for automotive products
- Applies to all activities during the safety lifecycle of safety-related systems comprised of electrical, electronic, and software components
- Is a standard, not a regulation
  - Broad industry participation in its development
  - Indication of broad industry adoption
  - Expected Publication Date: Nov. 15
- Key concept: Automotive Safety Integrity Level (ASIL)
  - Specify risk associated with a potential hazard
  - Specifies development requirements to achieve targeted integrity levels with respect to systematic and random hardware failures

Overview of ISO/DIS 26262



1. Vocabulary

2. Management of functional safety
- **2-5** Overall safety management
- **2-6** Safety management during item development
- **2-7** Safety management after release for production

3. Concept phase
- **3-5** Item definition
- **3-6** Initiation of the safety lifecycle
- **3-7** Hazard analysis and risk assessment
- **3-8** Functional safety concept

4. Product development: system level
- **4-5** Initiation of product development at the system level
- **4-6** Specification of the technical safety requirements
- **4-7** System design
- **4-8** Item integration and testing
- **4-9** Safety validation
- **4-10** Functional safety assessment
- **4-11** Release for production

5. Product development: hardware level
- **5-5** Initiation of product development at the hardware level
- **5-6** Specification of hardware safety requirements
- **5-7** Hardware design
- **5-8** Hardware architectural metrics
- **5-9** Evaluation of violation of the safety goal due to random HW failures
- **5-10** Hardware integration and testing

6. Product development: software level
- **6-5** Initiation of product development at the software level
- **6-6** Specification of software safety requirements
- **6-7** Software architectural design
- **6-8** Software unit design and implementation
- **6-9** Software unit testing
- **6-10** Software integration and testing
- **6-11** Verification of software safety requirements

7. Production and operation
- **7-5** Production
- **7-5** Operation, service (maintenance and repair), and decommissioning

8. Supporting processes
- **8-5** Interfaces within distributed developments
- **8-6** Specification and management of safety requirements
- **8-7** Configuration management
- **8-8** Change management
- **8-9** Verification
- **8-10** Documentation
- **8-11** Qualification of software tools
- **8-12** Qualification of software components
- **8-13** Qualification of hardware components
- **8-14** Proven in use argument

9. ASIL-oriented and safety-oriented analyses
- **9-5** Requirements decomposition with respect to ASIL tailoring
- **9-6** Criteria for coexistence of elements
- **9-7** Analysis of dependent failures
- **9-8** Safety analyses

10. Guideline on ISO 26262 (informative)

Source ISO/FDIS 26262

Outline

- ISO 26262 Overview
- ISO 26262 Competence Management
- ISO 26262 Safe SW Development

ISO 26262 Competence Management

- Part 2, 5.4.3.1 “The organization shall ensure that the persons involved in the execution of the safety lifecycle have a sufficient level of skills, competences and qualifications corresponding to their responsibilities.”

Source ISO/FDIS 26262

ISO 26262 Competence Management

- Part 2, 5.4.3.1 Note 1 – “One of the possible means to achieve a sufficient level of skills and competences in development is a training and qualification programme that considers the following knowledge areas:
  - usual safety practices, concepts and designs;
  - ISO 26262 and, if applicable, further safety standards;
  - organization-specific rules for functional safety;
  - functional safety processes instituted in the organization.”

Source ISO/FDIS 26262

ISO 26262 Competence Management

- Part 2, 5.4.3.1 Note 2 – "To evaluate the skills, competences and qualifications to carry out activities to comply with ISO 26262, the experience from previous professional activities can be considered, e.g.
  - domain knowledge of the item;
  - expertise on the environment of the item;
  - management experience."

ISO 26262 Lifecycle Steps

- Safety Management
  - Process Management
  - Design Confirmation Including Reviewers
- Safety-Critical Systems Development
  - Systems Development & Testing
  - SW Development & Testing
  - HW Development & Testing
- Distributed Development Management
- …

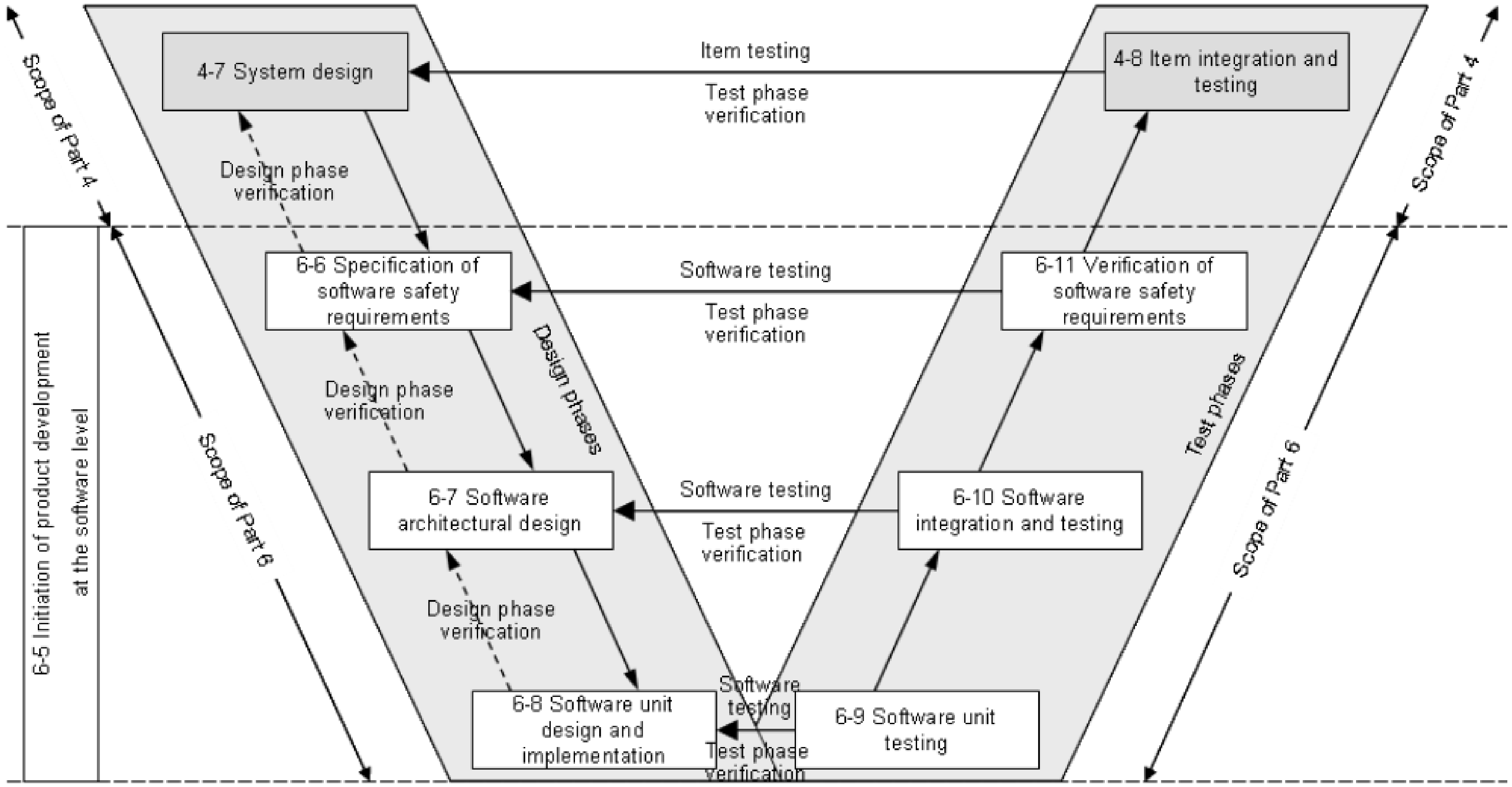
“Usual System Safety Concept” from ISO 26262

- Hazard Analysis and Risk Assessment
- Safety Concept Development
  - Functional safety requirements
  - Technical safety requirements
- Safety Analysis
  - HW & SW FMEA, FTA, Modeling & Simulation Tools
- Diagnostic & Remediation Strategies
  - Diagnostic Methods
  - Microcontroller and circuit board concepts
- Verification & Validation
  - HW / SW testing methods, including unit, integration, bench, vehicle
- Functional Safety Assessment / Safety Case

Outline

- ISO 26262 Overview
- ISO 26262 Competence Management
- ISO 26262 Safe SW Development

Software Development


Scope of Part 4
4-7 System design
Item testing
Test phase verification
4-8 Item integration and testing
Scope of Part 4
Design phase verification
6-6 Specification of software safety requirements
Software testing
Test phase verification
6-11 Verification of software safety requirements
6-5 Initiation of product development at the software level
Scope of Part 6
Design phase verification
Design phases
Test phases
6-7 Software architectural design
Software testing
Test phase verification
6-10 Software integration and testing
Scope of Part 6
Design phase verification
6-8 Software unit design and implementation
Software testing
Test phase verification
6-9 Software unit testing

Reference Phase Model for the Software Development

Source ISO/FDIS 26262

SW Development Work Products

- ❑ Safety plan (refined)
- ❑ Software verification plan
- ❑ Design and coding guidelines for modelling and programming languages
- ❑ Software tool application guidelines
- ❑ Software safety requirements specification
- ❑ Hardware-software interface specification (refined)
- ❑ Software verification plan (refined)
- ❑ Software verification report
- ❑ Software architectural design specification
- ❑ Safety analysis report
- ❑ Dependent failures analysis report
- ❑ Software unit design specification
- ❑ Software unit implementation
- ❑ Software verification specification (refined)
- ❑ Embedded software

Source ISO/FDIS 26262

SW Architecture Design Reorientation

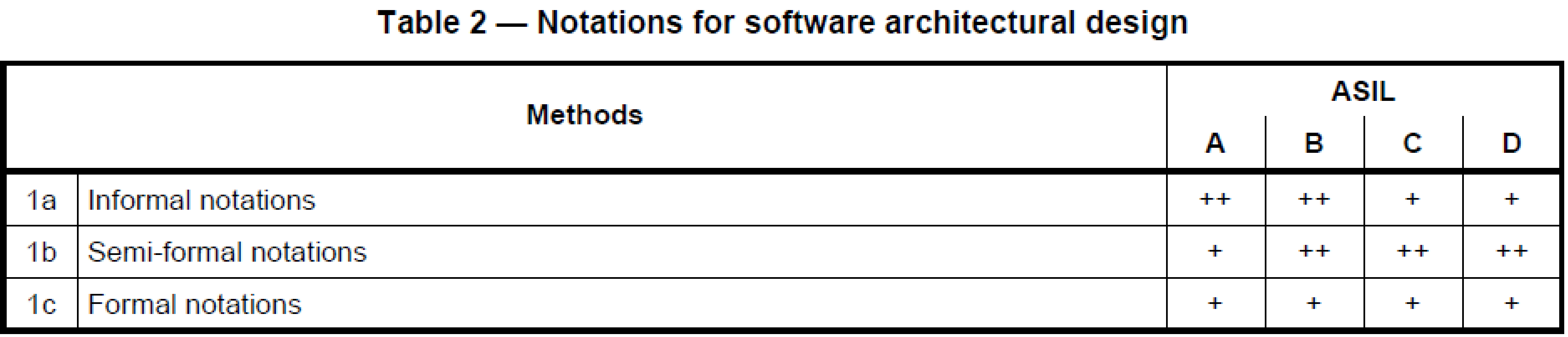


Table 2 — Notations for software architectural design

Methods		ASIL			
		A	B	C	D
1a	Informal notations	++	++	+	+
1b	Semi-formal notations	+	++	++	++
1c	Formal notations	+	+	+	+

Source ISO/FDIS 26262

SW Architecture Design

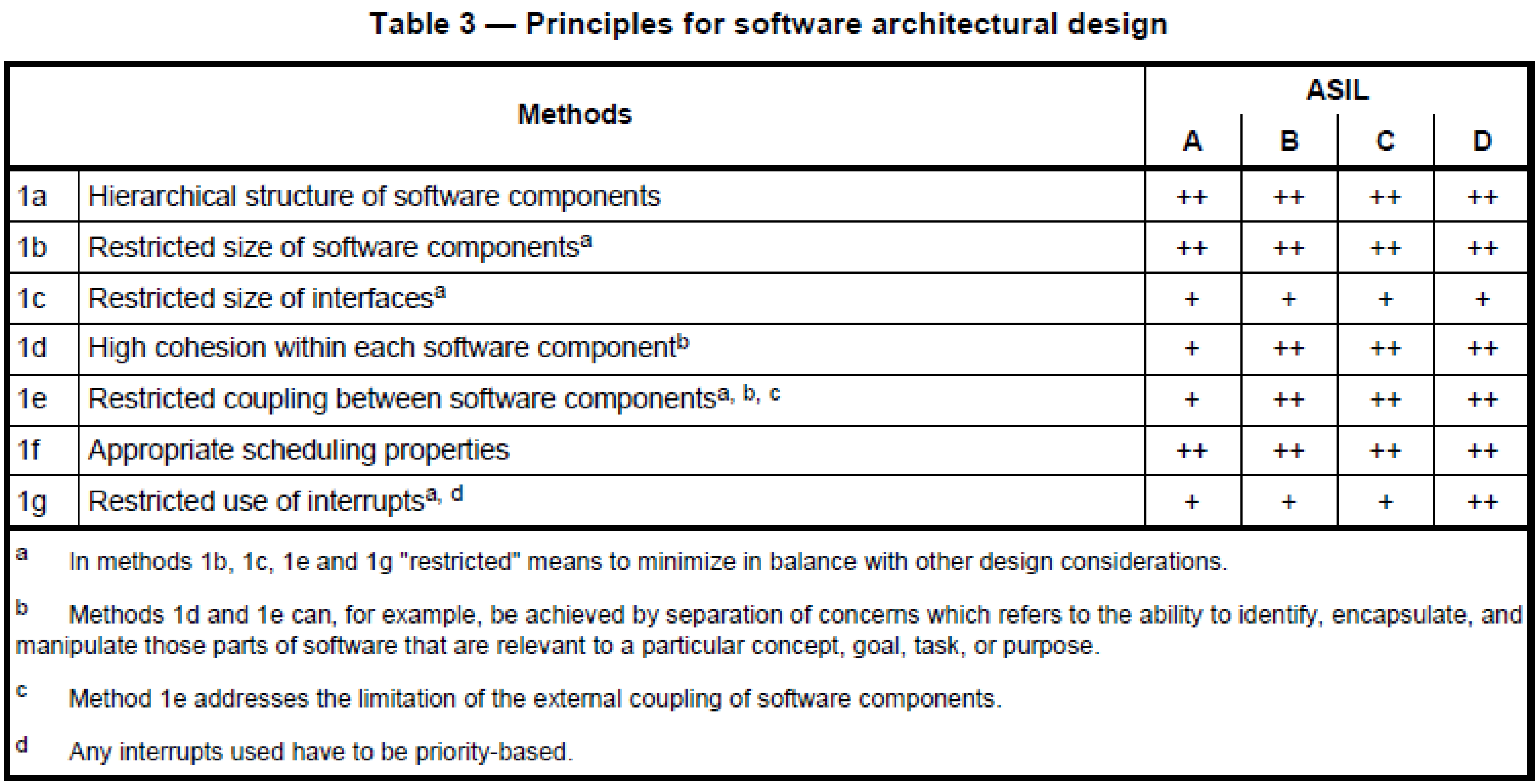


Table 3 — Principles for software architectural design

Methods		ASIL			
		A	B	C	D
1a	Hierarchical structure of software components	++	++	++	++
1b	Restricted size of software components[a]	++	++	++	++
1c	Restricted size of interfaces[a]	+	+	+	+
1d	High cohesion within each software component[b]	+	++	++	++
1e	Restricted coupling between software components[a, b, c]	+	++	++	++
1f	Appropriate scheduling properties	++	++	++	++
1g	Restricted use of interrupts[a, d]	+	+	+	++

[a] In methods 1b, 1c, 1e and 1g "restricted" means to minimize in balance with other design considerations.

[b] Methods 1d and 1e can, for example, be achieved by separation of concerns which refers to the ability to identify, encapsulate, and manipulate those parts of software that are relevant to a particular concept, goal, task, or purpose.

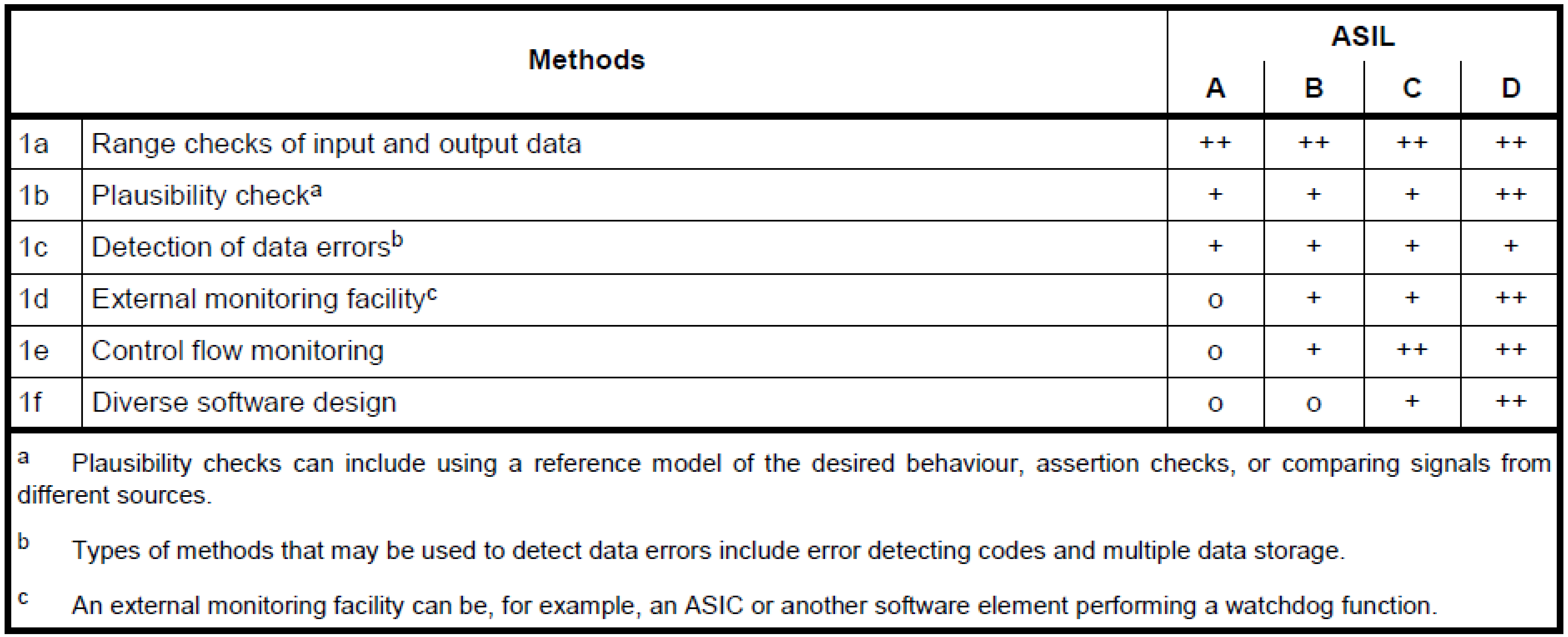
[c] Method 1e addresses the limitation of the external coupling of software components.

[d] Any interrupts used have to be priority-based.

Source ISO/FDIS 26262

SW Architecture Design

Table 4 — Mechanisms for error detection at the software architectural level



Methods		ASIL			
		A	B	C	D
1a	Range checks of input and output data	++	++	++	++
1b	Plausibility check[a]	+	+	+	++
1c	Detection of data errors[b]	+	+	+	+
1d	External monitoring facility[c]	o	+	+	++
1e	Control flow monitoring	o	+	++	++
1f	Diverse software design	o	o	+	++

[a] Plausibility checks can include using a reference model of the desired behaviour, assertion checks, or comparing signals from different sources.

[b] Types of methods that may be used to detect data errors include error detecting codes and multiple data storage.

[c] An external monitoring facility can be, for example, an ASIC or another software element performing a watchdog function.

Source ISO/FDIS 26262

Software Unit Design Representation

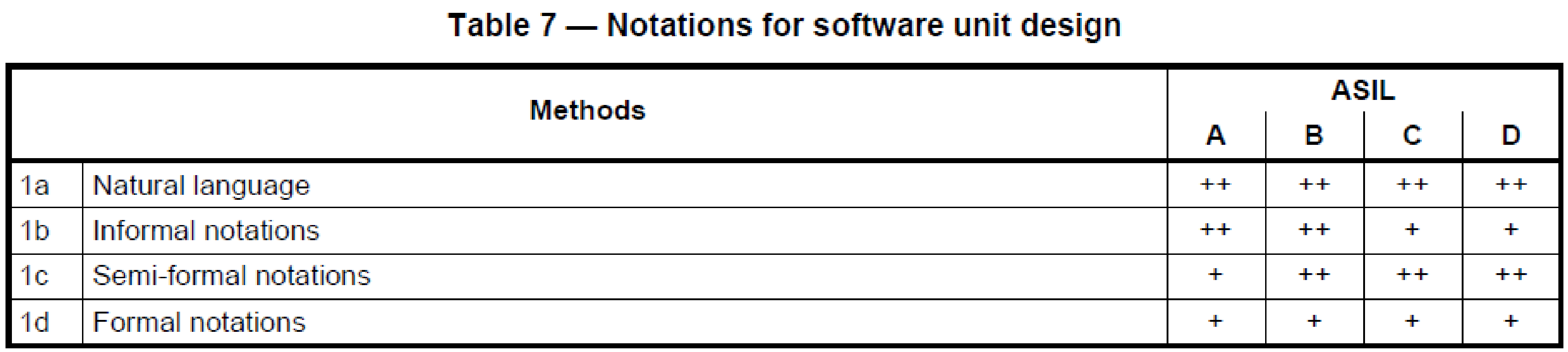


Table 7 — Notations for software unit design

Methods		ASIL			
		A	B	C	D
1a	Natural language	++	++	++	++
1b	Informal notations	++	++	+	+
1c	Semi-formal notations	+	++	++	++
1d	Formal notations	+	+	+	+

Source ISO/FDIS 26262

SW Unit Design Methods

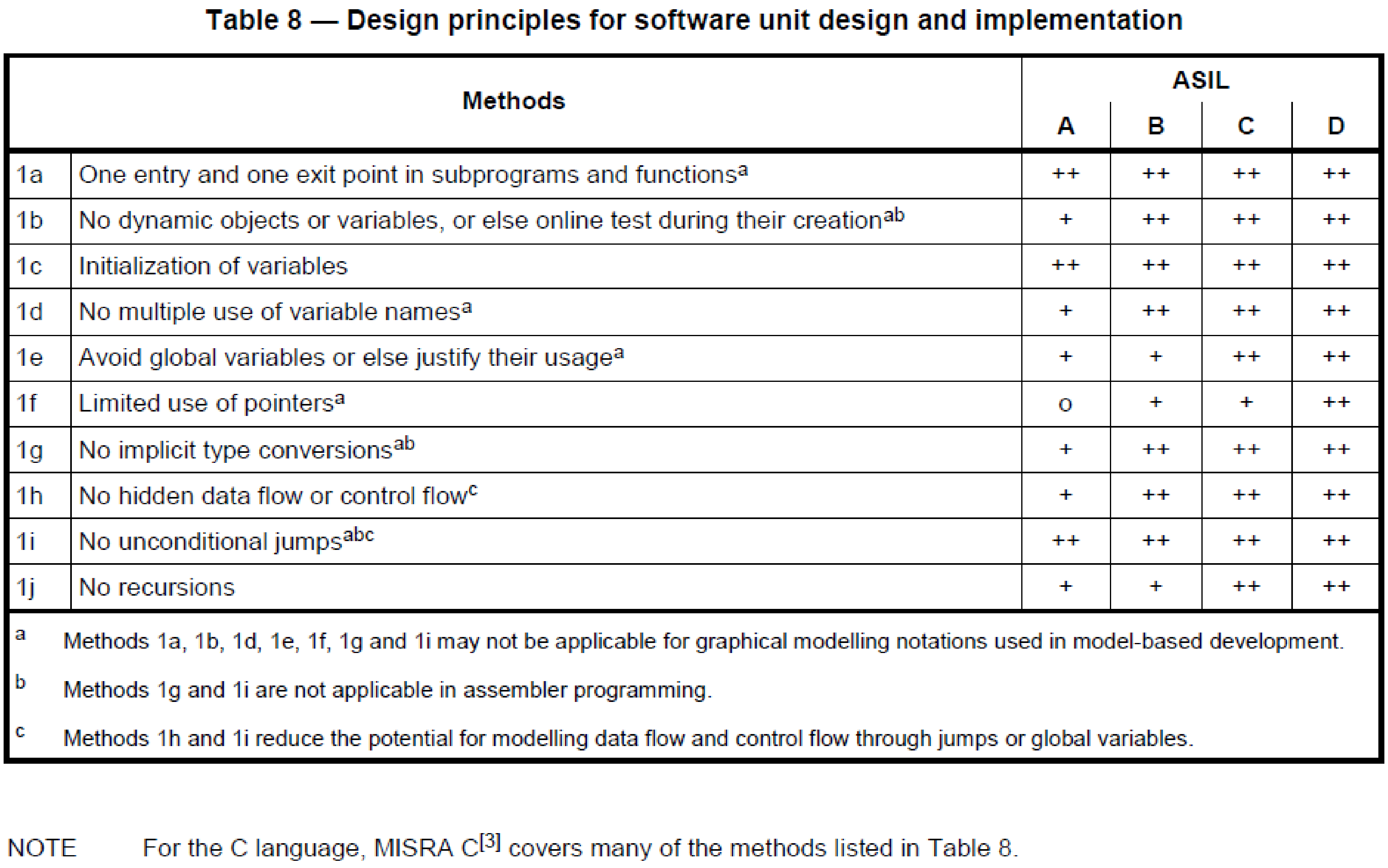


Table 8 — Design principles for software unit design and implementation

Methods		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[ab]	+	++	++	++
1c	Initialization of variables	++	++	++	++
1d	No multiple use of variable names[a]	+	++	++	++
1e	Avoid global variables or else justify their usage[a]	+	+	++	++
1f	Limited use of pointers[a]	o	+	+	++
1g	No implicit type conversions[ab]	+	++	++	++
1h	No hidden data flow or control flow[c]	+	++	++	++
1i	No unconditional jumps[abc]	++	++	++	++
1j	No recursions	+	+	++	++

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

[b] Methods 1g and 1i are not applicable in assembler programming.

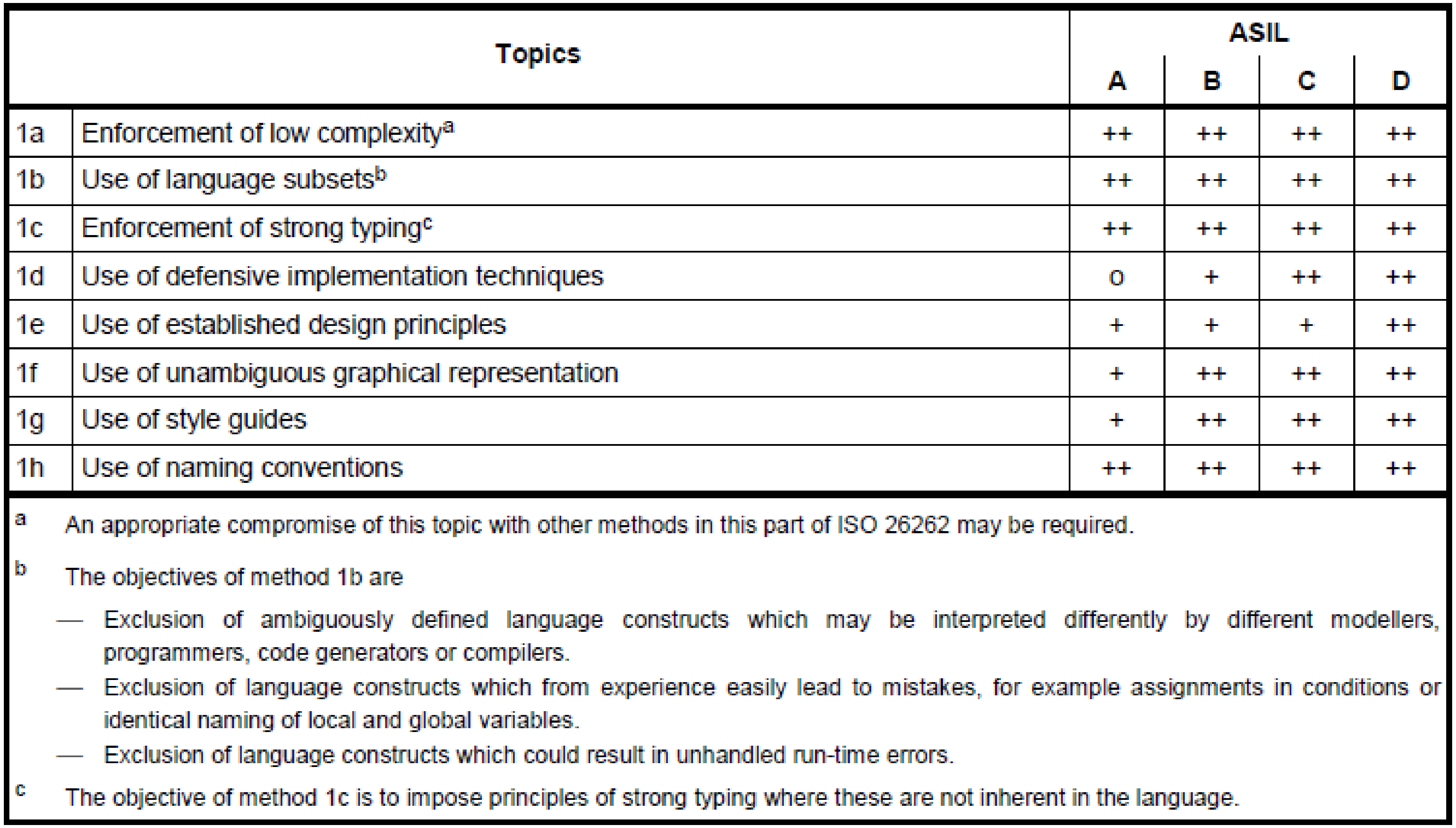
[c] Methods 1h and 1i reduce the potential for modelling data flow and control flow through jumps or global variables.

NOTE For the C language, MISRA C[3] covers many of the methods listed in Table 8.

Source ISO/FDIS 26262

Coding Guidelines

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	o	+	++	++
1e	Use of established design principles	+	+	+	++
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+	++	++	++
1h	Use of naming conventions	++	++	++	++

[a] An appropriate compromise of this topic with other methods in this part of ISO 26262 may be required.

[b] The objectives of method 1b are

— Exclusion of ambiguously defined language constructs which may be interpreted differently by different modellers, programmers, code generators or compilers.

— Exclusion of language constructs which from experience easily lead to mistakes, for example assignments in conditions or identical naming of local and global variables.

— Exclusion of language constructs which could result in unhandled run-time errors.

[c] The objective of method 1c is to impose principles of strong typing where these are not inherent in the language.

Source ISO/FDIS 26262

Example Software Unit Design Table

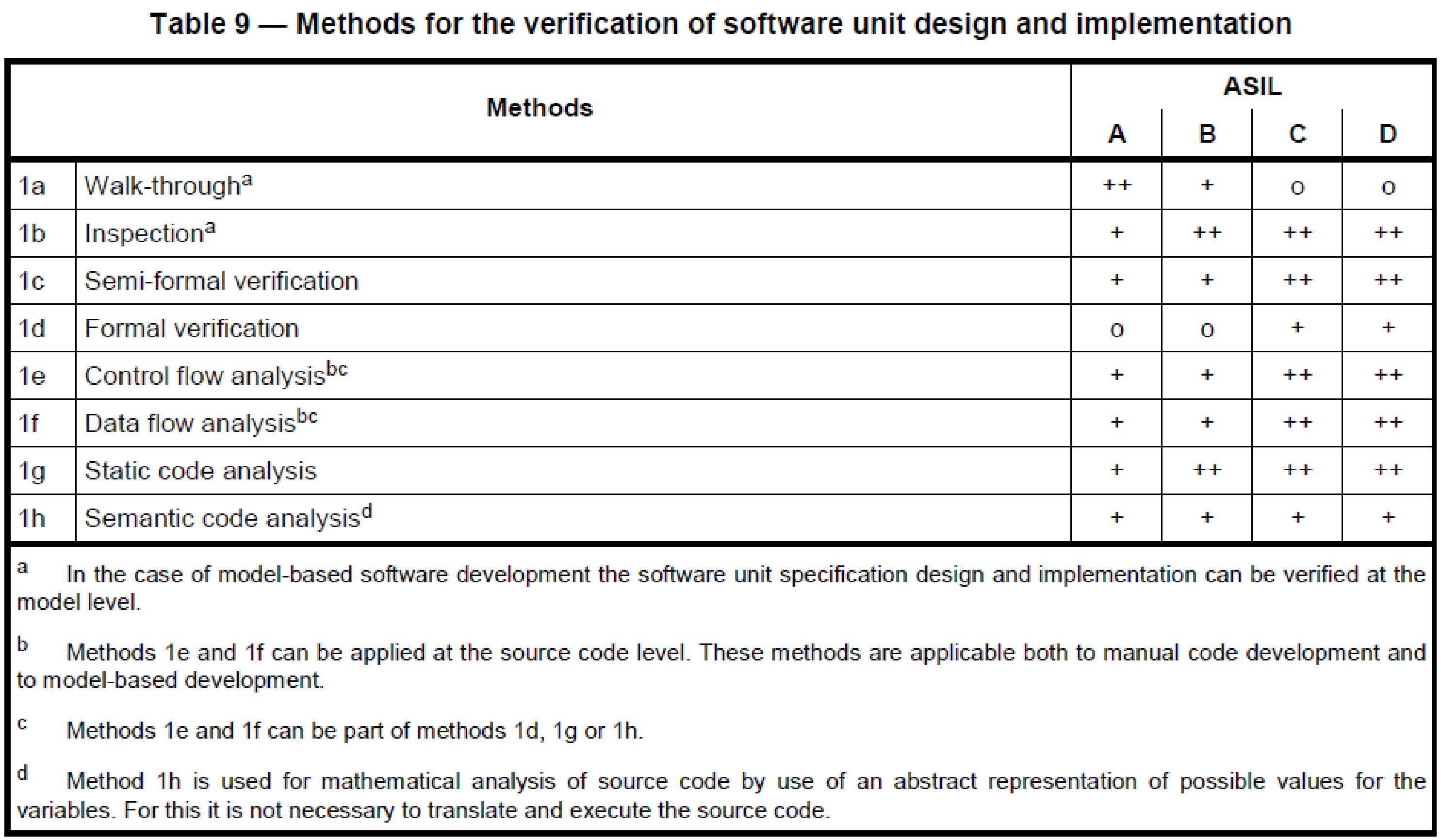


Table 9 — Methods for the verification of software unit design and implementation

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[bc]	+	+	++	++
1f	Data flow analysis[bc]	+	+	++	++
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.

[b] Methods 1e and 1f can be applied at the source code level. These methods are applicable both to manual code development and to model-based development.

[c] Methods 1e and 1f can be part of methods 1d, 1g or 1h.

[d] Method 1h is used for mathematical analysis of source code by use of an abstract representation of possible values for the variables. For this it is not necessary to translate and execute the source code.

Source ISO/FDIS 26262