Entwicklung sicherheitskritischer Systeme

Anforderungen der ISO 26262 an die Softwareentwicklung

Auswahl der Programmiersprache

Tabelle C.1 – Empfehlungen für bestimmte Programmiersprachen

	Programmiersprache	SIL1	SIL2	SIL3	SIL4
1	ADA	++	++	+	+
2	ADA mit Teilmenge	++	++	++	++
3	MODULA-2	++	++	+	+
4	MODULA-2 mit Teilmenge	++	++	++	++
5	PASCAL	++	++	+	+
6	PASCAL mit Teilmenge	++	++	++	++
7	FORTRAN 77	+	+	+	+
8	FORTRAN 77 mit Teilmenge	++	++	++	++
9	C	+	o	--	--
10	C mit Teilmenge und Codierrichtlinie und Verwendung von statischen Analysewerkzeugen	++	++	++	++
11	PL/M	+	o	--	--
12	PL/M mit Teilmenge und Codierrichtlinie	++	+	+	+

Klassische Entscheidung für laufendes Beispiel: Verwendung von C mit MISRA-Regeln und statischer Analyse zur Prüfung

Bewertungen der Programmiersprachen in der IEC 61508

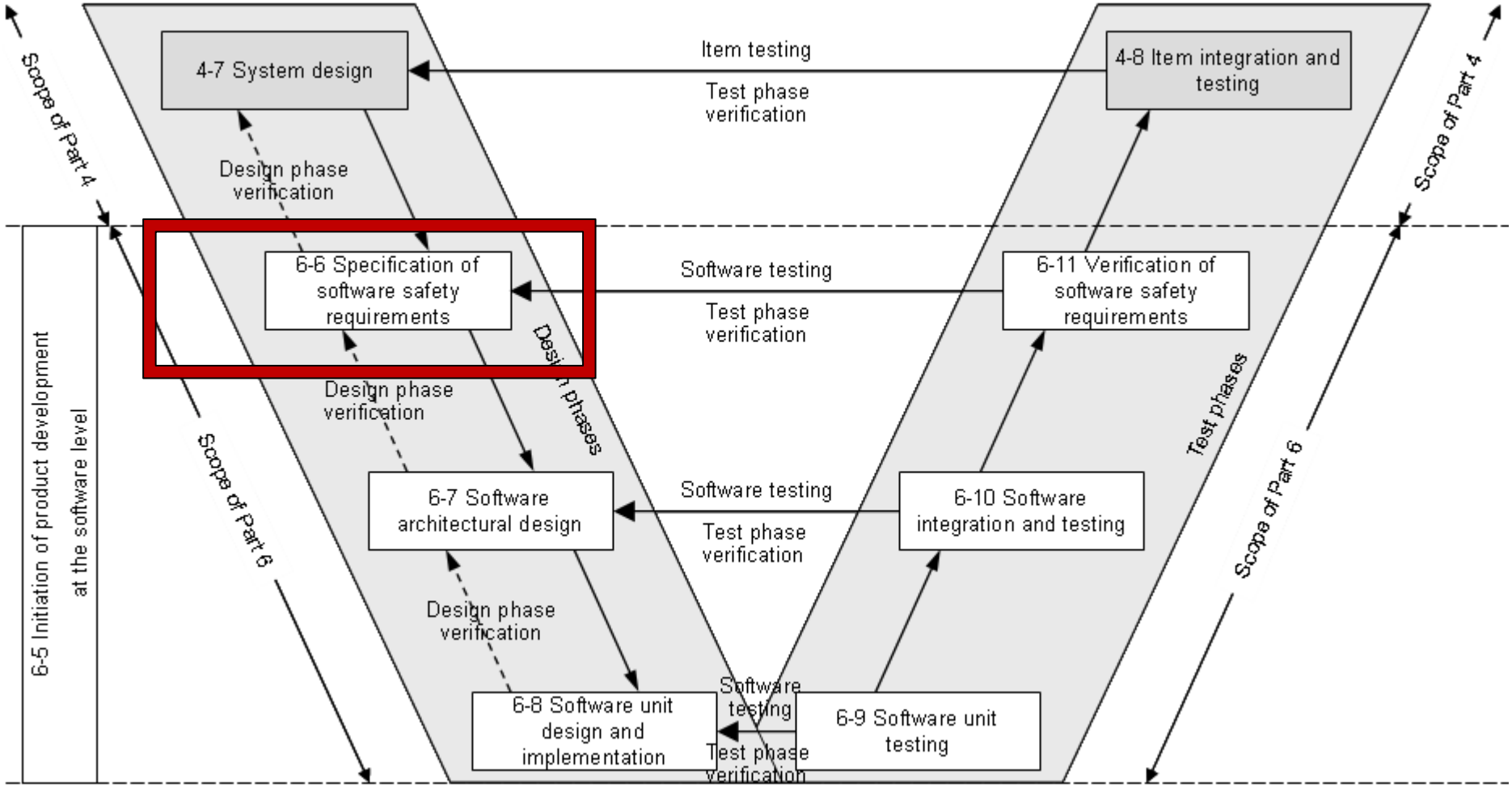
"++": Das Verfahren oder die Maßnahme wird für diesen Sicherheits-Integritätslevel besonders empfohlen. Wenn dieses Verfahren oder diese Maßnahme nicht verwendet wird, dann sollte der Grund während der Sicherheitsplanung ausführlich dargelegt und mit dem Gutachter abgestimmt werden.

"+": Das Verfahren oder die Maßnahme wird für diesen Sicherheits-Integritätslevel empfohlen. Es handelt sich um eine geringere Empfehlung als die mit "++" gekennzeichnete.

"o": Das Verfahren oder die Maßnahme hat keine Empfehlung für oder gegen die Verwendung.

"--": Das Verfahren oder die Maßnahme wird für diesen Sicherheits-Integritätslevel ausdrücklich nicht empfohlen. Wenn dieses Verfahren oder diese Maßnahme verwendet wird, dann sollte der Grund während der Sicherheitsplanung ausführlich dargelegt und mit dem Gutachter abgestimmt werden.

Vorgehensmodell bei der Softwareentwicklung



Spezifikation der Softwaresicherheitsanforderungen (SRS)

- Ziele:
  - Spezifikation aller Sicherheitsfunktionen der Software mit zugehörigen SIL
- Quelle der Sicherheitsanforderungen:
  - Systemsicherheitsanforderungsspezifikation
  - Hardwaresicherheitsanforderungsspezifikation
    - Beispiel: Erkennen des Ausfalls von Hardware (z.B. CPU- oder Speichertests)
- Regeln:
  - Präzise, quantitative Aussagen:
    - **Nicht**: Nach Betätigung des Lichtschalters wird das Ablendlicht eingeschalten.
    - **Sondern**: Maximal 1 Sekunde nach Betätigung des Lichtschalters wird das Licht eingeschalten. Die volle Lichtleistung wird nach maximal 5 Sekunden erreicht.
  - Präzise Benennung des Subjekts und der Verbindlichkeiten:
    - **Nicht**: Der Schalterstatus kann dem Fahrer signalisiert werden.
    - **Sondern**: Der Status des Schalters wird durch eine Schalterbeleuchtung angezeigt.

Methoden zur Definition und Verifikation der SRS

Tabelle A.1 – Spezifikation der Software-Sicherheitsanforderung (siehe 7.2)

	Verfahren/Maßnahme *	siehe	SIL1	SIL2	SIL3	SIL4
1	Rechnergestützte Spezifikationswerkzeuge	B.2.4	+	+	++	++
2a	Semi-formale Methoden	Tabelle B.7	+	+	++	++
2b	Formale Methoden mit z. B., CCS, CSP, HOL, LOTOS, OBJ, temporäre Logik, VDM und Z	C.2.4	o	+	+	++

Vorschlag zur Spezifikation der Softwaresicherheitsanforderungen zusätzlich zur prosaischen Beschreibung im IEC 61508-Standard.
Semi-Formale Methoden: z.B. Ablauf-, Datenfluss-, Zustandsübergangsdiagramme

Methoden zur Verifikation (ISO 26262) gilt ähnlich für die nächste Phase

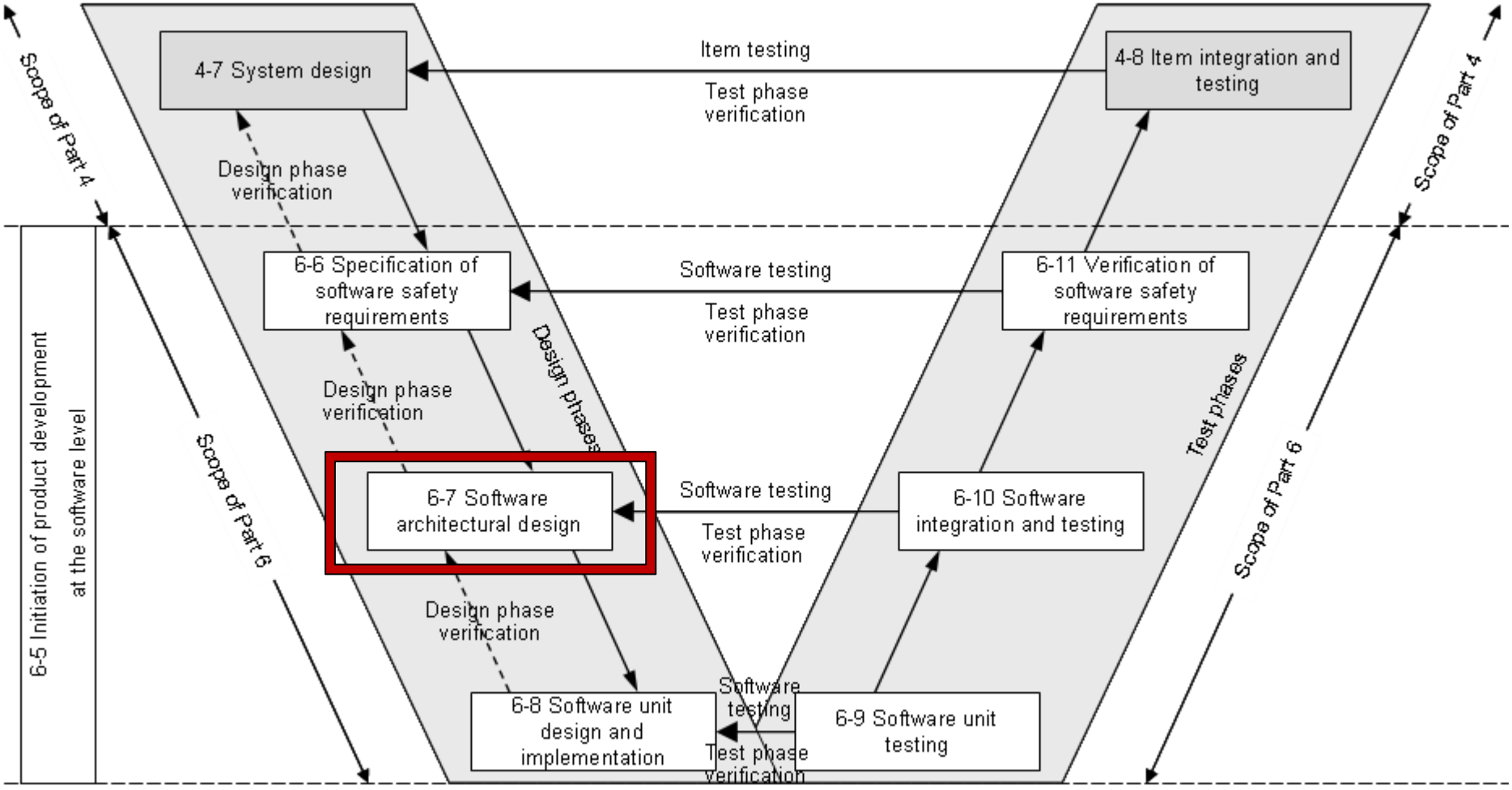
Table 2 — Methods for the verification of requirements

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Informal verification by walkthrough	++	+	o	o
1b	Informal verification by inspection	+	++	++	++
1c	Semi-formal verification[a]	+	+	++	++
1d	Formal verification	o	+	+	+

[a] Method 1c can be supported by executable models.

**Interpretation der Tabellen: Alternative Methoden sind durch Buchstaben gekennzeichnet. Für die IEC 61508 reicht die Anwendung einer Methode aus, bei der ISO 26262 muss der Entwickler eine Auswahl anwenden und die Auswahl begründen (es sei denn alle höchstgewerteten Methoden sind gewählt).*

Vorgehensmodell bei der Softwareentwicklung



Allgemeine Anforderungen an die Softwarearchitektur

- Die Architektur beschreibt
  - Die Identifikation der Softwarekomponenten, der Struktur und der Schnittstellen
  - Die Interaktion zwischen den Softwarekomponenten, aber auch den Hardwarekomponenten durch Spezifikation der Datenstrukturen, der Kommunikation, Ablaufsequenzen, etc.
  - Das Systemumfeld (Ein-/Ausgangsdaten, interne Datenstruktur, Kommunikation, Schnittstellen, Wartungsdaten)
  - Datensicherungsmethoden
  - Resultierende Testfälle
- Die Anforderungen der Software-SRS müssen durch die Architektur erfüllt werden
  - Insbesondere gilt: Bei mehreren Software-Sicherheitsfunktionen mit unterschiedlichen ASIL ist das Gesamtsystem stets nach dem höchsten SIL zu entwickeln
    - Ausnahme: die Unabhängigkeit (Rückwirkungsfreiheit) kann nachgewiesen werden
  - Bei Verwendung von Standard- bzw. Wiederverwendbaren Komponenten:
    - Nachweis durch Betriebsbewährtheit (Beachtung der Randbedingungen – siehe ARIANE 5 - Unfall) oder entsprechende ASIL bezogene Implementierung (Safety-Element-out-of-Context)

Anforderungen an die Softwarearchitektur (ISO 26262)

Table 4 — 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 minimise 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.

Zu verwendende Mechanismen (ISO 26262)

- Fehlererkennung:

Table 5 — Mechanisms for error detection at the software architectural level

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

[a] Plausibility checks include assertion checks. Complex plausibility checks can be realised by using a reference model of the desired behaviour.

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

[c] Diverse software design is not intended to imply n-version programming.

- Fehlerbehebung:

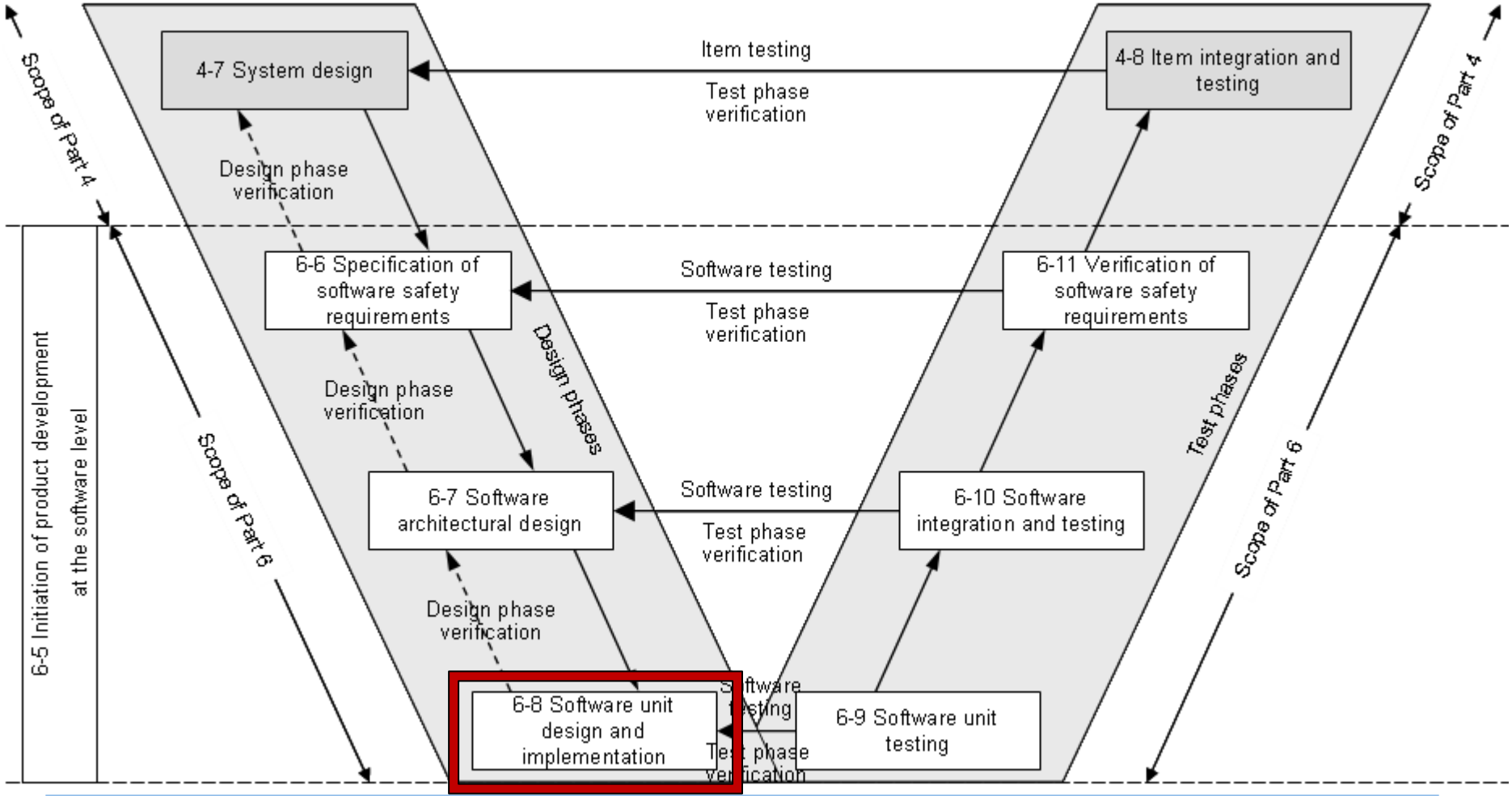
	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Static recovery mechanism[a]	+	+	+	+
1b	Graceful degradation[b]	+	+	++	++
1c	Independent parallel redundancy[c]	o	o	+	++
1d	Correcting codes for data	+	+	+	+

[a] Static recovery mechanisms can be realised by recovery blocks, backward recovery, forward recovery and recovery through repetition.

[b] Graceful degradation at the software level refers to prioritising functions to minimise the adverse effects of potential failures on functional safety.

[c] For parallel redundancy to be independent there has to be dissimilar software in each parallel path.

Vorgehensmodell bei der Softwareentwicklung



Anforderungen an die Implementierung

Beschreibung des Designs:

Table 8 — Notations for software unit design

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	Documentation of the software unit design in natural language	++	++	++	++
1b	Informal notations	++	++	+	+
1c	Semi-formal notations	+	++	++	++
1d	Formal notations	+	+	+	+

Designprinzipien:

Table 9 — Design principles for software unit design and implementation

Methods		ASIL A	ASIL B	ASIL C	ASIL 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, b]	+	++	++	++
1c	Initialisation 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[a, c]	+	++	++	++
1h	No hidden data flow or control flow[b, d]	+	++	++	++
1i	No unconditional jumps[a, c, d]	++	++	++	++
1j	No recursions	+	+	++	++

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

[b] If these compiler features are "tool qualified" in accordance with ISO 26262-8:—, Clause 10, Method 1b need not be applied if a compiler is used which ensures that there will be enough program storage allocated for all dynamic variables and objects before run-time or which inserts run-time tests for correct online-allocation of program storage, i.e. stack bounds checking.

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

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

Verifikation der Implementierung

Table 10 — Methods for the verification of software unit design and implementation

Methods		ASIL			
		A	B	C	D
1a	Informal verification	See Table 11			
1b	Semi-formal verification[a]	+	+	++	++
1c	Formal verification	o	o	+	+
1d	Control flow analysis[b, c]	+	+	++	++
1e	Data flow analysis[b, c]	+	+	++	++
1f	Static code analysis	+	++	++	++
1g	Semantic code analysis[d]	+	+	+	+

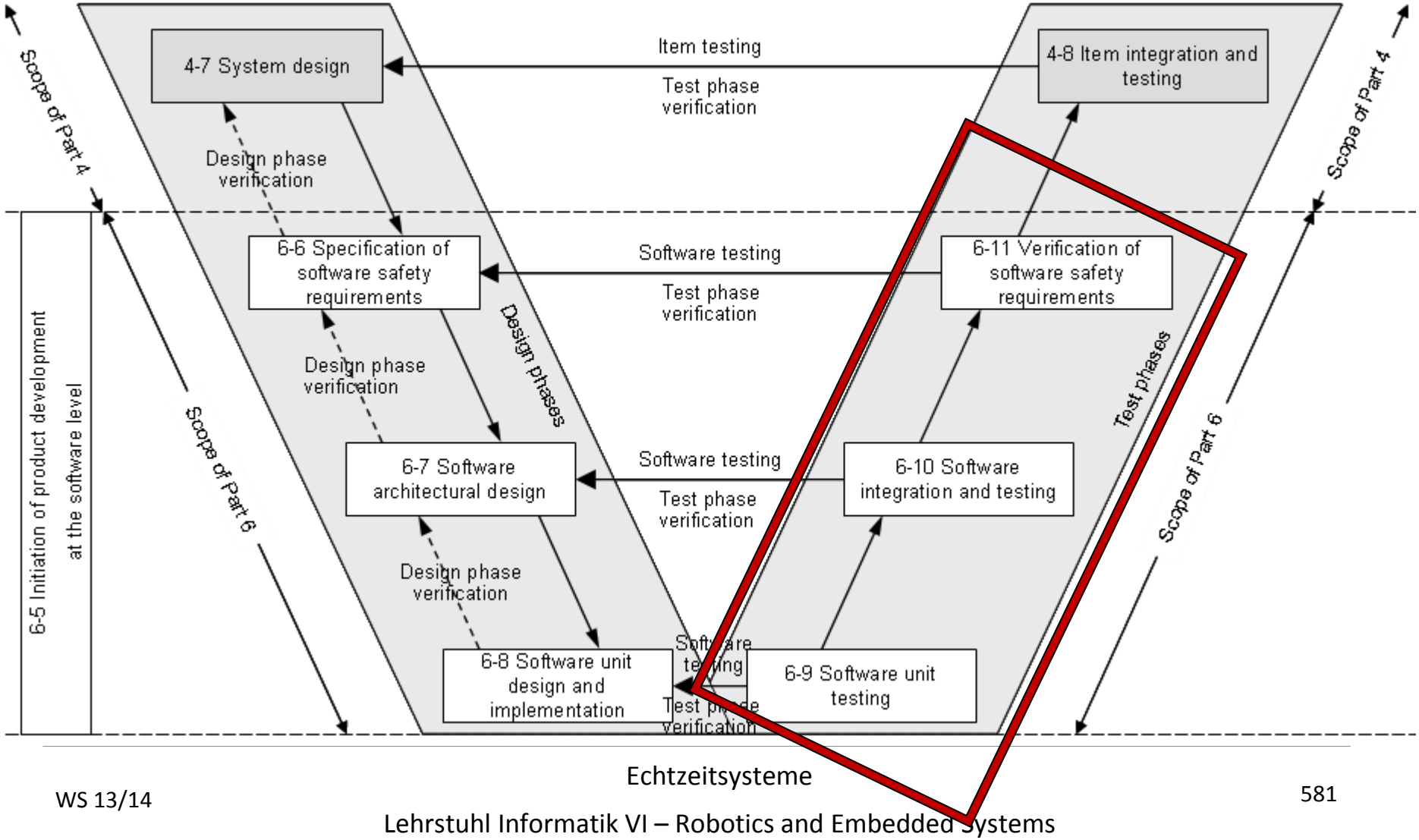
[a] Method 1b requires an executable design or implementation model of the unit to be verified.

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

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

[d] Method 1g 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.

Vorgehensmodell bei der Softwareentwicklung



Anforderungen an die Testarten

Auf Komponenten- & Architekturebene:

Table 12 — Methods for software unit testing

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	Requirement-based test	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test[a]	+	+	+	++
1d	Resource usage test[b]	+	+	+	++
1e	Back-to-back test between model and code, if applicable[c]	+	+	++	++

[a] This includes injection of arbitrary faults in order to test safety mechanisms (e.g. by corrupting values of variables)

[b] Some aspects of the resource usage test can only by evaluated properly when the software unit tests are executed on the target hardware or if the emulator for the target processor supports resource usage tests.

[c] This method requires a model that can simulate the functionality of the software units. Here, the model and code are stimulated in the same way and results compared with each other.

Auf Systemebene.

Table 18 — Test environments for conducting the software safety requirements verification

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	Hardware-in-the-loop	+	+	++	++
1b	Electronic control unit network environments[a]	++	++	++	++
1c	Vehicles	++	++	++	++

[a] Examples are "lab-cars", "rest of the bus" simulations or test benches partially or fully integrating the electrical systems of a vehicle.

Testabdeckung in den einzelnen Phasen

Auf Komponentenebene:

Table 14 — 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)		+	+	+	++

Auf Architekturebene:

Table 17 — Structural coverage metrics at the software architectural level

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	Function coverage[a]	+	+	++	++
1b	Call coverage[b]	+	+	++	++

[a] The degree of coverage claimed in method 1a requires at least one execution of every software function. This evidence can be achieved by an appropriate software integration strategy.

[b] The degree of coverage claimed in method 1b requires at least one execution of every software function call.

Zusammenfassung

- Was sollen Sie aus diesem Kapitel mitgenommen haben?
  - Verständnis für Zweck und Aufbau von Zertifizierungsstandards
    - Produkthaftung
  - Kenntnisse der Sicherheitslevel und Einstufung einer Anwendung
    - 3 Kriterien: Schadensmaß, Aufenthaltsdauer, Kontrollierbarkeit
  - Wesentliche Elemente / Phasen der Fehlertoleranz
    - Fehlererkennung
    - Fehlerdiagnose
    - Fehlerbehandlung / Fehlerbehebung (Reparatur)
  - Die Auswahl und Realisierung der Fehlertoleranzmechanismen basiert immer auf der Fehlerhypothese (definiert Menge und Art der zu tolerierenden Fehler)
  - Kenntnis der Mechanismen und Vergleich in Bezug auf zu tolerierende Fehler und Echtzeitfähigkeit
  - Anforderungen an die Softwareentwicklung entlang des Entwicklungsprozesses
    - Phasen des V-Modells und die entsprechenden Anforderungen
    - Dokumentationsaufwand
    - Anforderungen an die Entwicklungsmethodik

Klausur WS 08/09 – 10 Punkte

- Gegeben seien folgende Systeme zu Erlangung von Fehlertoleranz:
  1. 3-aus-5-Rechnersystem
  2. Verteiltes System mit TTP als Kommunikationsprotokoll
  3. System mit periodischem Anlegen von Checkpoints und Rückwärtsbehebung im Fehlerfall
  4. Hot-Stand-By-System bestehend aus zwei Komponenten
- Ordnen Sie die Systeme eineindeutig den folgenden Anwendungen zu und begründen Sie kurz Ihre Antwort in Bezug auf die Echtzeitfähigkeit, sowie die tolerierbaren Fehler.
  a. In einem Atomkraftwerk muss die Steuerung fehlertolerant realisiert werden. Es muss mindestens ein beliebiger Rechnerausfall (fehlerhaftes Ergebnis bzw. Komplettausfall) toleriert werden können, zudem muss ein korrektes Ergebnis innerhalb von festen Zeitschranken bestimmt werden.
  b. Ein echtzeitkritisches Steuerungssystem soll zur Erhöhung der Verfügbarkeit fehlertolerant gestaltet werden. Aufgrund von intensiven Selbsttests kann davon ausgegangen werden, dass sich ein einzelner Rechner im Fehlerfall selbst abschaltet und kein fehlerhaftes Ergebnis ausgibt.
  c. Zur Verwaltung des Lagerbestandes wird in einem Logistiksystem ein fehlertolerantes System benötigt, das temporäre Fehler tolerieren muss.
  d. In einem Satelliten soll ein verteiltes System möglichst fehlertolerant gegenüber Strahlungseinflüssen entwickelt werden. Dabei werden speziell für den Weltraumeinsatz entwickelte Rechner mit Strahlenschutz eingesetzt.
- Lösung: 1a, 2d, 3c, 4b

Ausblick

Ausblick

- Vorlesung
  - Cyber-Physical Systems mit Schwerpunkt auf dem Thema Verifikation
  - Verteilte rekonfigurierbare Steuerungssysteme
  - Einführung in die digitale Signalverarbeitung
  - Kognitive Systeme
  - Bewegungsplanung in der Robotik
- Seminare:
  - Real-time Communication Systems
  - Machine Learning II
- Praktikum:
  - Applied Computer Vision for Robotics
  - RACE
  - HW/SW co-design with a LEGO Car
- Studienarbeiten (BA, MA)
- Studentische Hilfskräfte / Promotion (Lehrstuhl bzw. fortiss)