INFORMATIK ▪ CONSULTING ▪ SYSTEMS AG

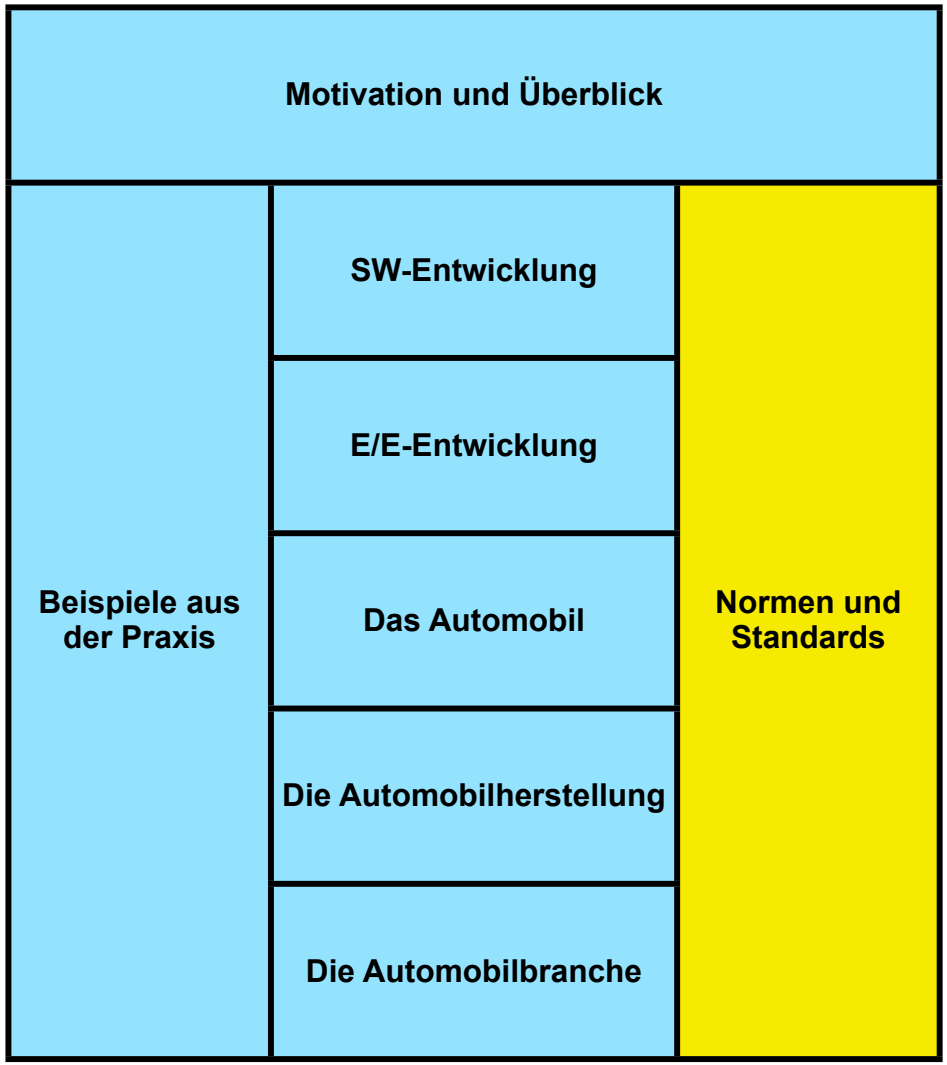
Vorlesung
Automotive Software Engineering
Teil 7 Normen und Standards (3)

TU Dresden, Fakultät Informatik

Sommersemester 2012

Prof. Dr. rer. nat. Bernhard Hohlfeld

bernhard.hohlfeld@daad-alumni.de



AUTOSAR

MOST

CAN

lin

HIS

OSEK/ VDX

ASAM

FlexRay

ISO 26262
Road vehicles - Functional safety

7. Normen und Standards

1. AUTOSAR
2. ARTOP
3. **Vorgehensmodelle und funktionale Sicherheit**

Gliederung

- Vorgehensmodelle
- Funktionale Sicherheit
- Vorgehensmodelle und funktionale Sicherheit - ISO 26262

Gliederung

- **Vorgehensmodelle**
- Funktionale Sicherheit
- Vorgehensmodelle und funktionale Sicherheit - ISO 26262

Vorgehensmodell

Allgemeiner Vorgehensrahmen

- benennt und beschreibt allgemein die Art und Anordnung von Aktivitäten im Rahmen der SW-Entwicklung
- man unterscheidet
  - sequenzielle, streng phasenorientierte Modelle
  - nichtlineare evolutionäre Modelle
- wird in der Praxis für konkrete Projekte und Unternehmen näher spezifiziert

→ konkretes Prozessmodell festlegen

Prozessmodell

Konkrete Instanz eines Vorgehensmodells

- definiert die konkreten Aktivitäten, ihre Reihenfolge und ihre Produkte für die Durchführung eines Projektes
- Definition ist abhängig von
  - Projekttyp und
  - Unternehmenspolitik und -kompetenz
- die Aktivitäten müssen durch Methoden und Werkzeuge unterstützt werden

→ abhängig vom Vorgehensparadigma

Vorgehensmodelle im Software Engineering
Dr. Marco Kuhrmann, TU München

- TU München - Fakultät für Informatik Lehrstuhl IV: Software & Systems Engineering (Prof. Manfred Broy)
- Vorlesung | Wintersemester 2010/2011 (2+1, 5 ETCS-Punkte)
- http://www4.in.tum.de/lehre/vorlesungen/vgmse/ws1011/
- Inhalt

  Die Vorlesung vermittelt die grundlegenden Techniken und Methoden der Projektorganisation und des Projektmanagements für die Entwicklung großer Softwaresysteme.

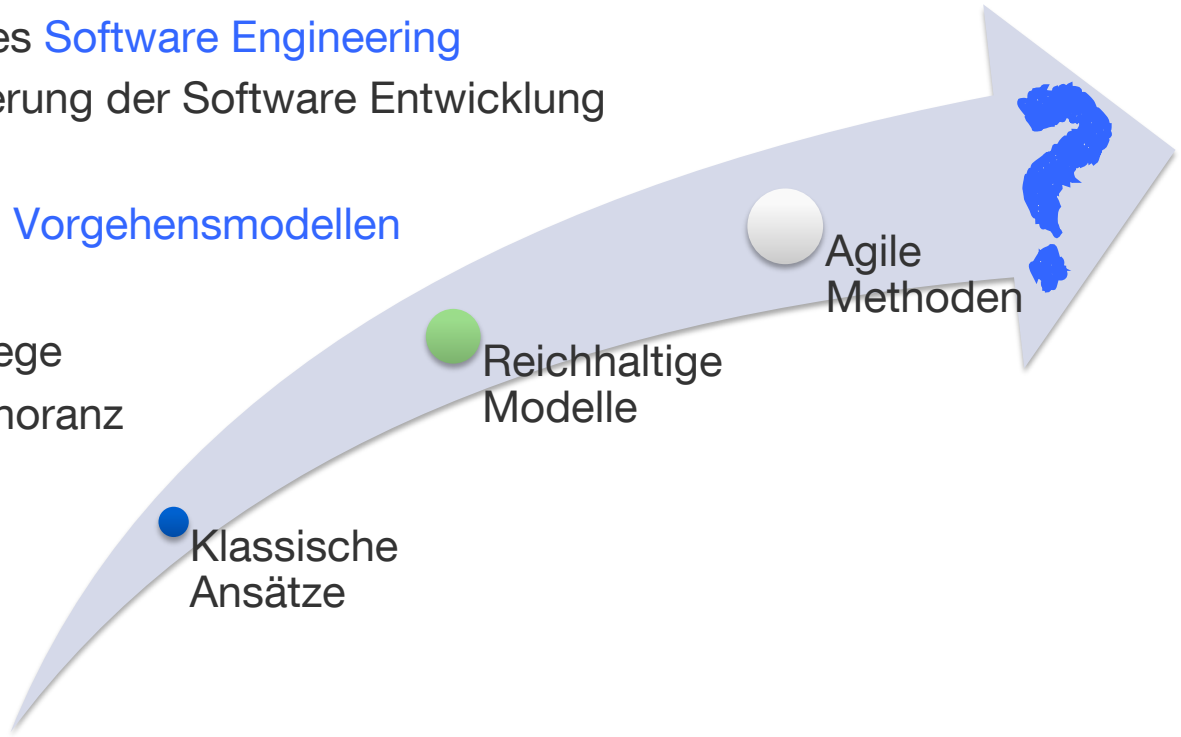
Vorgehensmodelle im Software Engineering
Dr. Marco Kuhrmann, TU München

- Grundlagen Was ist ein Vorgehensmodell?
  - Einordnung von Vorgehensmodellen in Prozessladschaften
- Übersicht Vorgehensmodelle
  - Klassische Ansätze
  - Agile Modelle/Methoden
  - Strukturierte Modelle
  - Weitere Modelle
  - Werkzeugunterstützung
- Lebenszyklus von Vorgehensmodellen Phasenübersicht (im Folgenden vertieft)
- Analyse
  - Analyse und Optimierung von Prozessen
  - Stakeholder und Rollen
  - Vorgänge/Abläufe
  - Artefakte/Produkte
  - Strukturierung von Prozessen
- Konstruktion und Anpassung
  - Konstruktionsoptionen
  - Vorgehensmetamodelle
  - Umsetzung und Bereitstellung
- Einführung
  - Einführungsstrategien
  - Planung und Maßnahmen
  - Schulungen
  - Etablieren von Feedbackschleifen
- Weiterentwicklung und Pflege
  - Prozessreife
  - Audits, Assessments und Zertifizierung
  - Planung von Prozessverbesserungen
  - Ausgewählte Modelle und Methoden

Inhalte und Ziele der Vorlesung (1)

- Vorgehensmodelle existieren seit den frühen 70‘ern
  - Ziel: „Raus aus der Software Krise“
    → Geburt des Software Engineering
  - Systematisierung der Software Entwicklung

- Entwicklung von Vorgehensmodellen
  - Inflation
  - Glaubenskriege
  - Trend zur Ignoranz



Inhalte und Ziele der Vorlesung (2)

- Ziele
  - Organisations- und Projektprozesse verstehen
  - Vorgehensmodelle verstehen
  - Prozesse planen, abbilden, definieren und umsetzen
  - Handhabung von Werkzeugen und Infrastrukturen für Prozessanpassungen erlernen

- Praktischer Anwendungskontext
  - Prozessoptimierendes Projektmanagement
    - Projekt- und Teamorganisation
    - Effektivitätsfeststellung und Verbesserung
  - Prozessanalyse und Beratung

Inhalte und Ziele der Vorlesung (3)

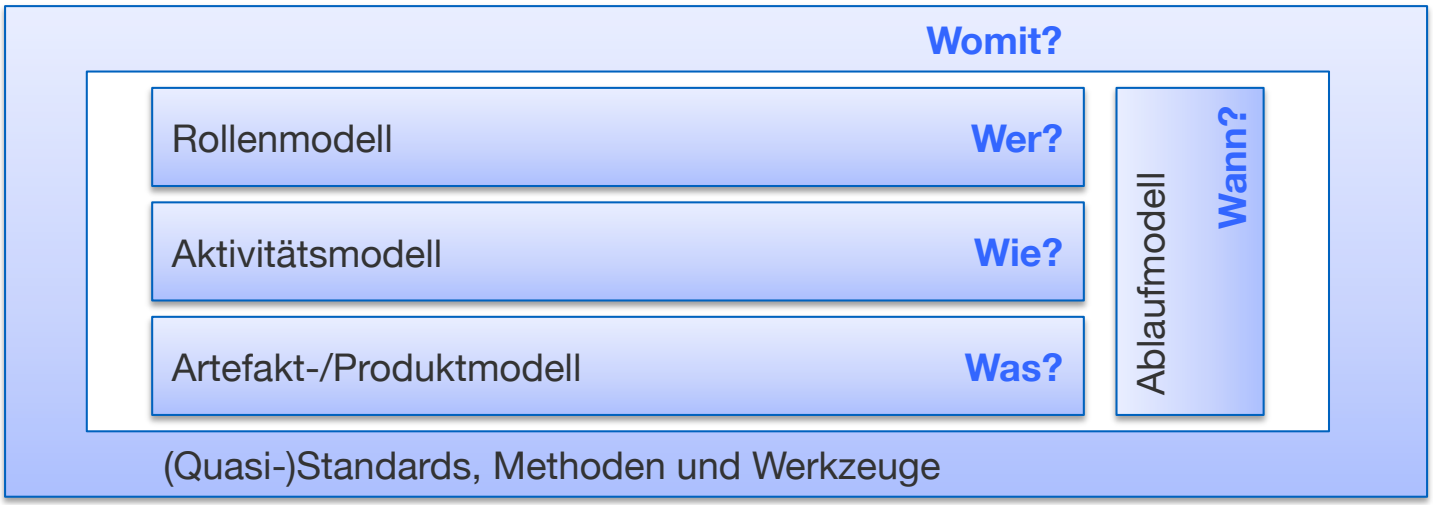
1. Grundlagen
   → *Begriffsklärung und Einordnung*
2. Übersicht Vorgehensmodelle
   → *Aufstellung/Bewertung existierender Ansätze*
3. Lebenszyklus von Vorgehensmodellen
   → *Überblick über die Phasen (Vertiefung im Anschluss)*
4. Analyse
   → *Analyse und Optimierung von Prozessen, Stakeholder, Modelle etc.*
5. Konstruktion und Anpassung
   → *Metamodellierung und Modellierung von Vorgehensmodellen*
6. Einführung
   → *Einführungsmaßnahmen, Planung, Schulung etc.*
7. Weiterentwicklung und Pflege
   → *Prozessreife, Planung, Übersicht Reifegradmodelle*

Was ist ein Vorgehensmodell?

Definition

Ein Vorgehensmodell beschreibt systematische, ingenieurmäßige und quantifizierbare Vorgehensweisen, um Aufgaben einer bestimmten Klasse wiederholbar zu lösen.

Submodelle

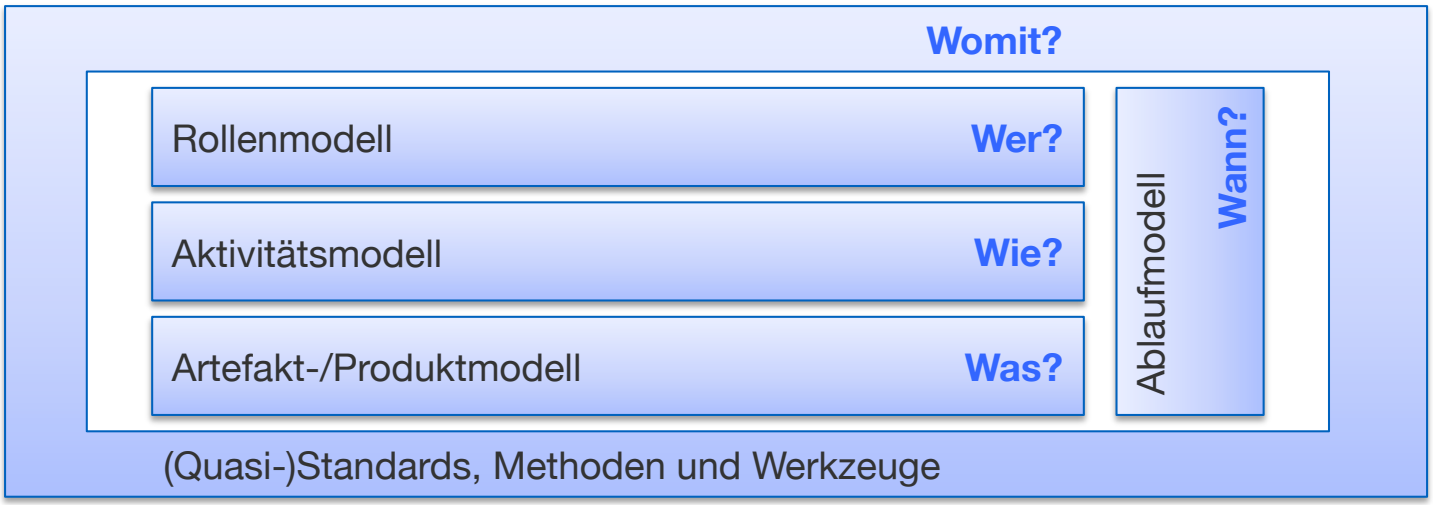


Was ist ein Vorgehensmodell?

Definition

Ein Vorgehensmodell beschreibt systematische, ingenieurmäßige und quantifizierbare Vorgehensweisen, um Aufgaben einer bestimmten Klasse wiederholbar zu lösen.

Submodelle **1. Definition der Aktivitäten (wer, was, wie, wann)**



Sichten auf das Vorgehensmodell

- Wer?
  Rollenorientierte Sicht:
  Wer ist für welche Arbeitsergebnisse (Produkte, Work Products) verantwortlich?
- Was?
  Produktorientierte Sicht:
  Was sind die zu erarbeitenden Produkte?
- Wie?
  Aktivitätenorientierte Sicht:
  Wie werden die Produkte erarbeitet?
- Wann?
  Phasenorientierte Sicht:
  Wann werden welche Arbeitsschritte durchgeführt?
  Meilensteinorientierte Sicht:
  Wann werden welche Produkte fertiggestellt und geprüft?
- Womit?
  Methoden- und werkzeugorientierte Sicht:
  Womit (Methoden und Werkzeuge) werden die Produkte erarbeitet?

Gliederung

- Vorgehensmodelle
- **Funktionale Sicherheit**
- Vorgehensmodelle und funktionale Sicherheit - ISO 26262

- Dieser Abschnitt basiert auf dem Vortrag

  Entwicklung und Zulassung von sicherheitskritischen Systemen -
  was kann die Automobilbranche von Bahnen und Luftfahrt lernen?

  Dr. Bernhard Hohlfeld, ICS AG, Ulm (Vortragender)
  Dr. Paul Linder, ICS AG, Stuttgart
  Udo Hipp, ICS AG, Stuttgart

  Tagung

  Elektronik im Kraftfahrzeug

  16./17. Juni 2010, Dresden

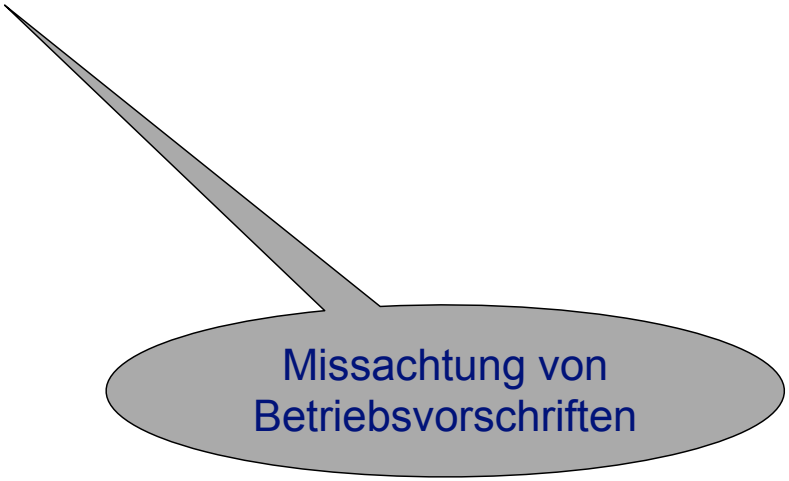
Gliederung

Katastrophen mit technischen Systemen

- 1986 Explosion im Kernkraftwerk Tschernobyl
- 1987 Explosion des Space Shuttle Challenger
- 1998 ICE-Unglück bei Eschede
- 1999 Feuer im Mont Blanc Tunnel
- 2000 Absturz der Concorde bei Paris
- 2010 Absturz der Tupolew 154 bei Smolensk

Katastrophen mit technischen Systemen

- 1986 Explosion im Kernkraftwerk Tschernobyl
- 1987 Explosion des Space Shuttle Challenger
- 1998 ICE-Unglück bei Eschede
- 1999 Feuer im Mont Blanc Tunnel
- 2000 Absturz der Concorde bei Paris
- 2010 Absturz der Tupolew 154 bei Smolensk

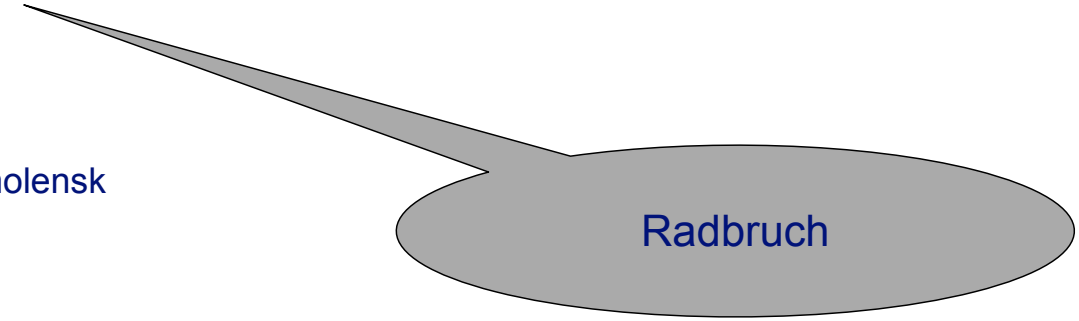


Katastrophen mit technischen Systemen

- 1986 Explosion im Kernkraftwerk Tschernobyl
- 1987 Explosion des Space Shuttle Challenger
- 1998 ICE-Unglück bei Eschede
- 1999 Feuer im Mont Blanc Tunnel
- 2000 Absturz der Concorde bei Paris
- 2010 Absturz der Tupolew 154 bei Smolensk

Katastrophen mit technischen Systemen

- 1986 Explosion im Kernkraftwerk Tschernobyl
- 1987 Explosion des Space Shuttle Challenger
- 1998 ICE-Unglück bei Eschede
- 1999 Feuer im Mont Blanc Tunnel
- 2000 Absturz der Concorde bei Paris
- 2010 Absturz der Tupolew 154 bei Smolensk

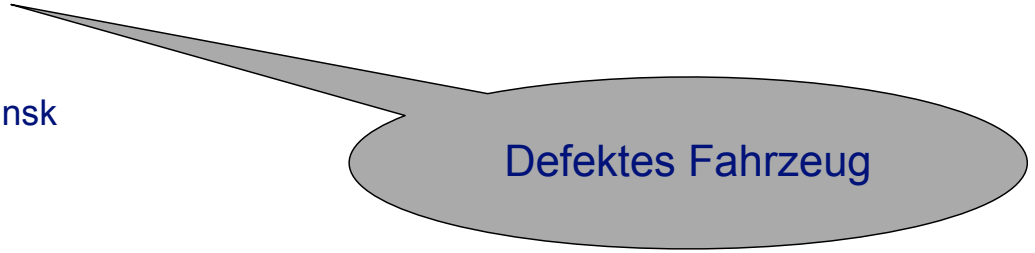


Katastrophen mit technischen Systemen

- 1986 Explosion im Kernkraftwerk Tschernobyl
- 1987 Explosion des Space Shuttle Challenger
- 1998 ICE-Unglück bei Eschede
- 1999 Feuer im Mont Blanc Tunnel
- 2000 Absturz der Concorde bei Paris
- 2010 Absturz der Tupolew 154 bei Smolensk

Katastrophen mit technischen Systemen

- 1986 Explosion im Kernkraftwerk Tschernobyl
- 1987 Explosion des Space Shuttle Challenger
- 1998 ICE-Unglück bei Eschede
- 1999 Feuer im Mont Blanc Tunnel
- 2000 Absturz der Concorde bei Paris
- 2010 Absturz der Tupolew 154 bei Smolensk

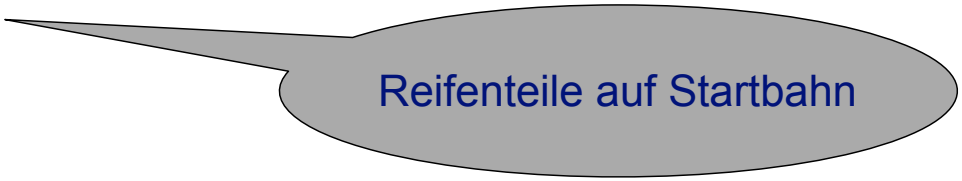


Katastrophen mit technischen Systemen

- 1986 Explosion im Kernkraftwerk Tschernobyl
- 1987 Explosion des Space Shuttle Challenger
- 1998 ICE-Unglück bei Eschede
- 1999 Feuer im Mont Blanc Tunnel
- 2000 Absturz der Concorde bei Paris
- 2010 Absturz der Tupolew 154 bei Smolensk

Katastrophen mit technischen Systemen

- 1986 Explosion im Kernkraftwerk Tschernobyl
- 1987 Explosion des Space Shuttle Challenger
- 1998 ICE-Unglück bei Eschede
- 1999 Feuer im Mont Blanc Tunnel
- 2000 Absturz der Concorde bei Paris
- 2010 Absturz der Tupolew 154 bei Smolensk



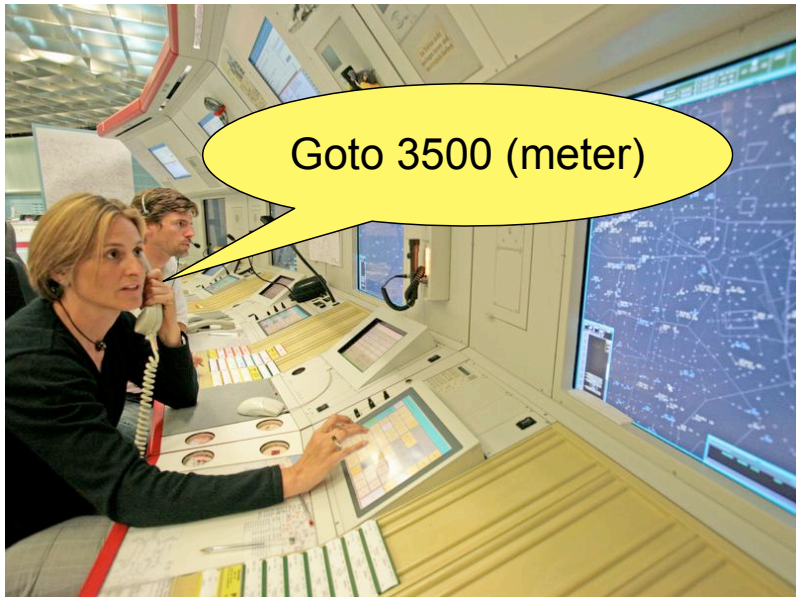
Katastrophen mit technischen Systemen

- 1986 Explosion im Kernkraftwerk Tschernobyl
- 1987 Explosion des Space Shuttle Challenger
- 1998 ICE-Unglück bei Eschede
- 1999 Feuer im Mont Blanc Tunnel
- 2000 Absturz der Concorde bei Paris
- 2010 Absturz der Tupolew 154 bei Smolensk

Igel

Eagle

Unterschiedliche Normen und Standards



Nach Josef Börcsök:
Funktionale Sicherheit,
Hüthig Verlag, Heidelberg, 2008.

Unvollständige Abdeckung

Der automatische Vortriebsregler unserer B737 hatte die Eigenschaft, sich manchmal während des Startvorgangs bei exakt 60 Konten zu verabschieden. Es waren unsere Werkstätten - und nicht etwa der Gerätehersteller -, die anhand des glücklicherweise vorhandenen Listings die Ursache fanden: Der Programmierer hatte festgelegt, was der Vortriebsregler unter und was er über 60 Knoten Fahrt tun sollte. Nur ihm zu sagen, wie er bei 60 Knoten reagieren sollte, dass hatte er vergessen. Wenn der Computer nun bei exakt 60 Knoten die entsprechende Bedingung abfragte, fand er keine Anweisung vor, war verwirrt und schaltete ab.

Nach J.P. Hach:
Digitale Elektronik in Verkehrsflugzeugen, in DGLR (Hrsg.): Test und Verifkation von Software bei digitalen Systemen der Luft- und Raumfahrt, DGLR-Bericht 83-02.

Unvollständige Abdeckung

Der automatische Vortriebsregler unserer B737 hatte die Eigenschaft, sich manchmal während des Startvorgangs bei exakt 60 Konten zu verabschieden. **Es waren unsere Werkstätten** - und nicht etwa der Gerätehersteller -, **die anhand des glücklicherweise vorhandenen Listings die Ursache fanden**: Der Programmierer hatte festgelegt, was der Vortriebsregler unter und was er über 60 Knoten Fahrt tun sollte. Nur ihm zu sagen, wie er bei 60 Knoten reagieren sollte, dass hatte er vergessen. Wenn der Computer nun bei exakt 60 Knoten die entsprechende Bedingung abfragte, fand er keine Anweisung vor, war verwirrt und schaltete ab.

Nach J.P. Hach:
Digitale Elektronik in Verkehrsflugzeugen, in DGLR (Hrsg.): Test und Verifkation von Software bei digitalen Systemen der Luft- und Raumfahrt, DGLR-Bericht 83-02.

Softwarefehler

- Fehlerursache
  Verwechslung von Punkt und Komma in FORTRAN
  - Richtig mit Komma: DO 10 i = 1,3 . . . (Schleife)
  - Falsch mit Punkt: DO 10 i = 1.3 . . . (Zuweisung)
- Fehlerauswirkung:
  Die Mission eines zum Planet Venus gestarteten Satelliten scheiterte (laut NASA).

- Programmierfehler oder ungeeignete Programmiersprache?
- PASCAL
  - for I := 1 to 3 do ...;
  - I := 1.3;

- Nach Rudolf M. Konakovsky:
  Zuverlässigkeit und Sicherheit von Automatisierungssystemen,
  Institut für Automatisierungs- und Softwaretechnik, Universität Stuttgart,
  Vorlesung, 2005.

Kein Rosenmontagsscherz

```
if <condition>
then
        ....
        goto L2;
        ....
        L1:
        ....
else
        ....
        L2:
        ....
        goto L1;
        ....
end if;
```

```
while <condition>
do
        ...
        goto L;
        -- irgendwo ausserhalb der
        -- Schleife
        ...
end while;
```

Gliederung

Ansätze und Prinzipien der Funktionalen Sicherheit

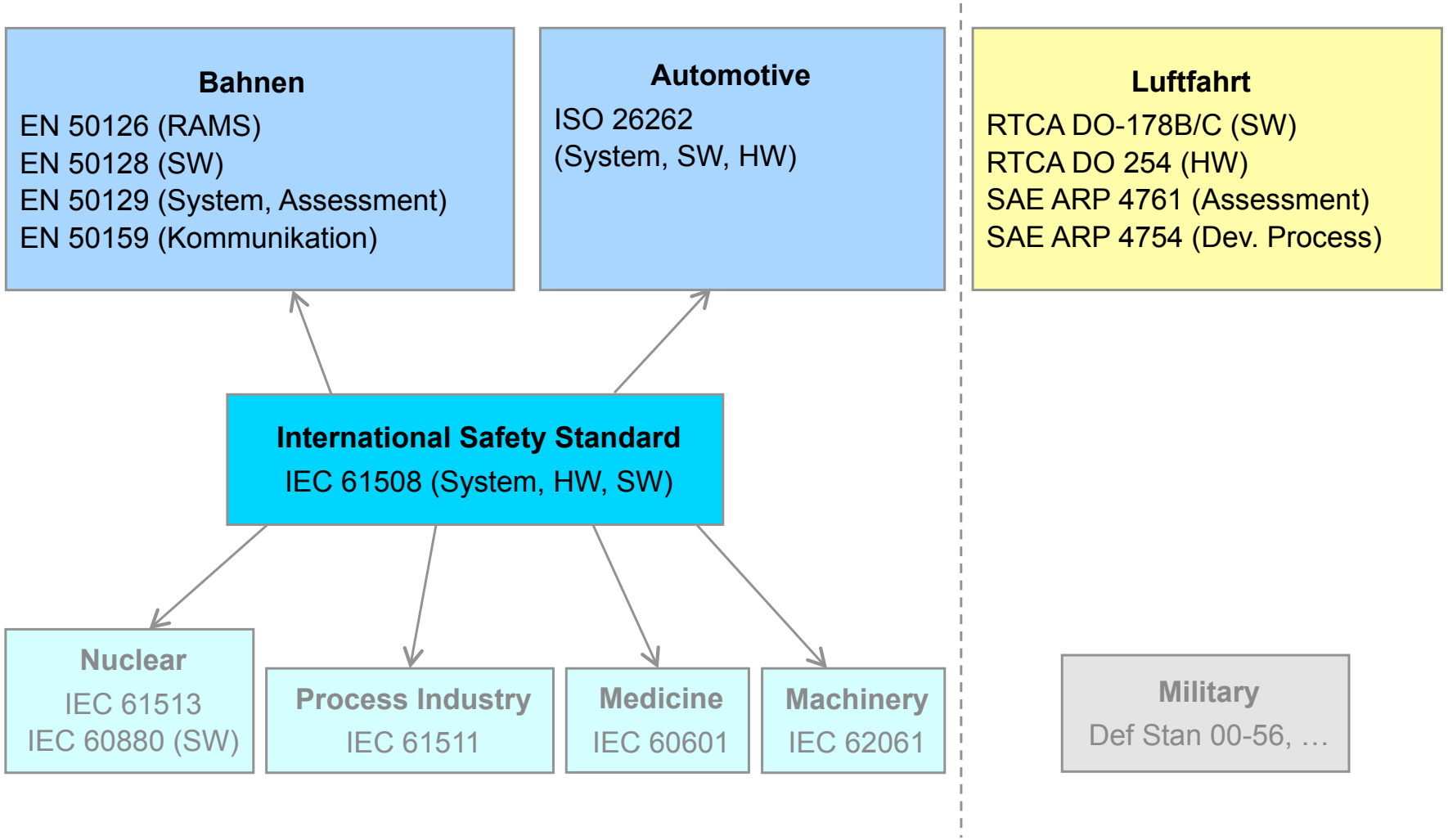
Ansätze und Prinzipien der Funktionalen Sicherheit

	Zufällige Fehler	Systematische Fehler
Beispiele	■ Hardwareausfall ■ Übertragungsfehler	■ Designfehler ■ Spezifkationsfehler ■ Programmierfehler
Strategie	Beherrschung der Auswirkungen	Fehlervermeidung
Ansatz	*Quantitative* Analysen	Vorgeschriebene Methoden abhängig vom (*qualitativen)* Safety Integrity Level (SIL)

Prinzipien	■ Fehlererkennung
	  ■ Selbsttests
	■ Fail-safe
	(Sicherer Zustand bei Ausfall)
	■ Redundanz
	■ Ziel: Beherrschung jedes einzelnen Fehlers
		■ Entwicklung nach Stand der Wissenschaft und Technik
		■ Umfangreiche Verifikation
		■ Nachvollziehbarkeit
		■ Abdeckung
		■ Unabhängigkeit
		  ■ Technisch: Diversität, ...
		  ■ Personell: Entwickler und Prüfer verschiedene Personen
		  ■ Organisatorisch: Entwickler und Prüfer in verschiedenen Organisationen
		■ Ziel: Vermeidung von Fehlern

Sicherheitsstandards im Überblick

Sicherheitsstandards im Überblick



Vergleich der Sicherheitsstandards

Vergleich der Sicherheitsstandards

	IEC 61508	EN 50126 EN 50128 EN 50129 EN 50159	ISO 26262	DO-178B DO-254 ARP 4761 ARP 4754
Anwendungsbereich	Generisch	Bahnen (1-dimensional)	Automotive (2-dimensional)	Luftfahrt (3-dimensional)
Sicherheitsansatz	Sicherer Zustand, Fail-safe	Sicherer Zustand, Fail-safe im Fehlerfall	Sicherer Zustand oder sichere Fortsetzung mit Restfunktionalität	Sichere Fortsetzung des Fluges und sichere Landung
Betrachtete Gefahren	Gefährdungen von Menschen und Umwelt		Nur Gefährdungen von Menschen	
Abdeckung	System, Umwelt, Wartung		System	
Safety Integrity Levels (SIL)	SIL 4 (hoch) SIL 3 SIL 2 SIL 1 (niedrig) --	SIL 4 (hoch) SIL 3 SIL 2 SIL 1 (niedrig) SIL 0 (nicht sicherheitsrelevant)	-- ASIL D (hoch) ASIL C ASIL B ASIL A (niedrig) (QM)	Level A (hoch) Level B Level C Level D Level E (niedrig) --
Organisatorische Aspekte	Teilweise	Ja	Ja	Nein
Werkzeug-qualifizierung	Nein	Nein	Ja	Ja

Gliederung

Systemfunktion „Anfahren“

- Als einfaches Beispiel wird die Systemfunktion „Anfahren“ bei einem PKW mit Automatikgetriebe genommen. Gewolltes Anfahren bei laufendem Motor wird durch die folgenden Bedienschritte erreicht:
- Auf die Betriebsbremse („Fussbremse“) treten und diese gedrückt halten.
- Den Wählhebel in die Stellung „D“ oder „R“ bringen.
- Ggf. die Feststellbremse („Handbremse“) lösen.
- Die Betriebsbremse lösen.
- Gas geben.
- Fehlverhalten der Systemfunktion „Anfahren“ wäre „nicht gewolltes Anfahren“.

Functional Hazard Assessment (FHA)
Gefährdungsanalyse

Bei der FHA wird der Systementwurf aus funktionaler Sicht analysiert. Ziel ist die Identifikation von

- Möglichem Fehlverhalten
- Betriebszustand, in dem das Fehlverhalten auftritt
- Auswirkung des Fehlverhaltens
- Klassifizierung der Auswirkungen (z.B. gefährlich, bedeutend, ungefährlich)
- Gegenmassnahmen (wenn sinnvoll)
- Überprüfungsmethode

Das Ergebnis der FHA wird meist in Tabellenform dokumentiert. Abbildung 3 zeigt das Ergebnis der FHA für das Beispiel „Anfahren“ (in Anlehnung an [2]).

Systemfunktion	Fehlverhalten	Betriebszustand	Auswirkung der Fehlerbedingung	Klassifizierung	Gegenmassnahmen	Überprüfungsmethode
Anfahren	Nicht gewolltes Anfahren	Motor aus, Bremsen gelöst, Stellung "N"	Fahrzeug kann anfahren (je nach Strassenneigung)	bedeutend	Schlüssel kann nur bei Stellung "P" abgezogen werden, evtl. Warnsignal	
	Nicht gewolltes Anfahren	Motordrehzahl über Grenzwert, Bremsen gelöst, Stellung "D" oder "R"	Fahrzeug fährt an	gefährlich		FMEA

Abbildung 3: Ergebnis der FHA für die Systemfunktion „Anfahren“

FMEA
Failure Mode and Effects Analysis
Fehler Möglichkeits- und Einflussanalyse

Beispiel FMEA - Motorenentwicklung

- **Motor „Typ 12“**
- **Biturbo-System mit mit zwei Ladeluftkühlern**
- **405 kW / 550 PS**
- **900 Nm**

Failure Mode and Effects Analysis									Blatt Nr.:	
Produkt-feature	Möglicher Fehler	Mögliche Folgen	Mögliche Fehler-ursache	Aktueller Status: Aktuelle Maß-nahme	Auftreten	Bedeutung	Entdeckung	RPZ	Maß-nahmen / Ver-ant-wortlich	Termin
Feder Nr. 103-5	Bruch	Zylinder-ausfall	Ermüdung	Festigkeits-test	6	7	10	420	versch. R.B.Shaw	08/07/01
Öldicht-schraube	Leck	Ölverlust, Über-hitzung	Dichtung nicht fest genug	Höheres Montage-moment	7	9	9	567	dickere Dichtung R.Frost	05/09/01

Bewertungszahlen:

A - Auftretenswahrscheinlichkeit
1 (unwahrscheinlich)
10 (hoch)

B - Bedeutung
1 (keine Bedeutung)
10 (sehr hohe Bedeutung)

E - Entdeckungswahrscheinlichkeit
1 (hoch)
10 (unwahrscheinlich)

FMEA
Failure Mode and Effects Analysis
Fehler Möglichkeits- und Einflussanalyse

Beispiel FMEA - Motorenentwicklung

- **Motor „Typ 12“**
- **Biturbo-System mit mit zwei Ladeluftkühlern**
- **405 kW / 550 PS**
- **900 Nm**

RPZ
Risiko Prioritätszahl

Failure Mode and Effects Analysis									Blatt Nr.:		
Produkt-feature	Möglicher Fehler	Mögliche Folgen	Mögliche Fehler-ursache	Aktueller Status: Aktuelle Maß-nahme	Auftreten	Bedeutung	Entdeckung	RPZ	Maß-nahmen	Ver-ant-wortlich	Termin
Feder Nr. 103-5	Bruch	Zylinder-ausfall	Ermüdung	Festigkeits-test	6	7	10	420	versch.	R.B.Shaw	08/07/01
Öldicht-schraube	Leck	Ölverlust, Über-hitzung	Dichtung nicht fest genug	Höheres Montage-moment	7	9	9	567	dickere Dichtung	R.Frost	05/09/01

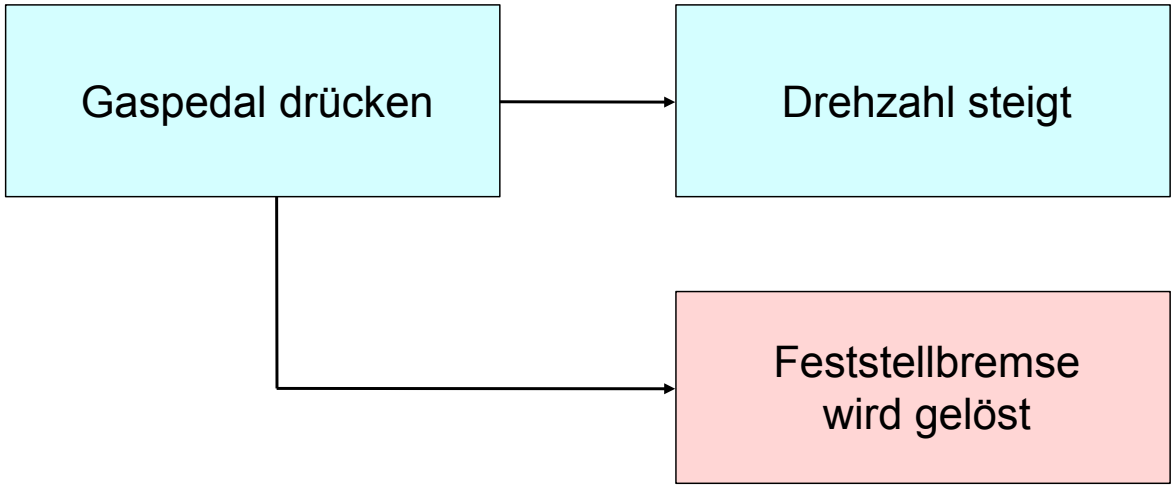
Bewertungszahlen:

A - Auftretenswahrscheinlichkeit
1 (unwahrscheinlich)
10 (hoch)

B - Bedeutung
1 (keine Bedeutung)
10 (sehr hohe Bedeutung)

E - Entdeckungswahrscheinlichkeit
1 (hoch)
10 (unwahrscheinlich)

Angedachte Lösung

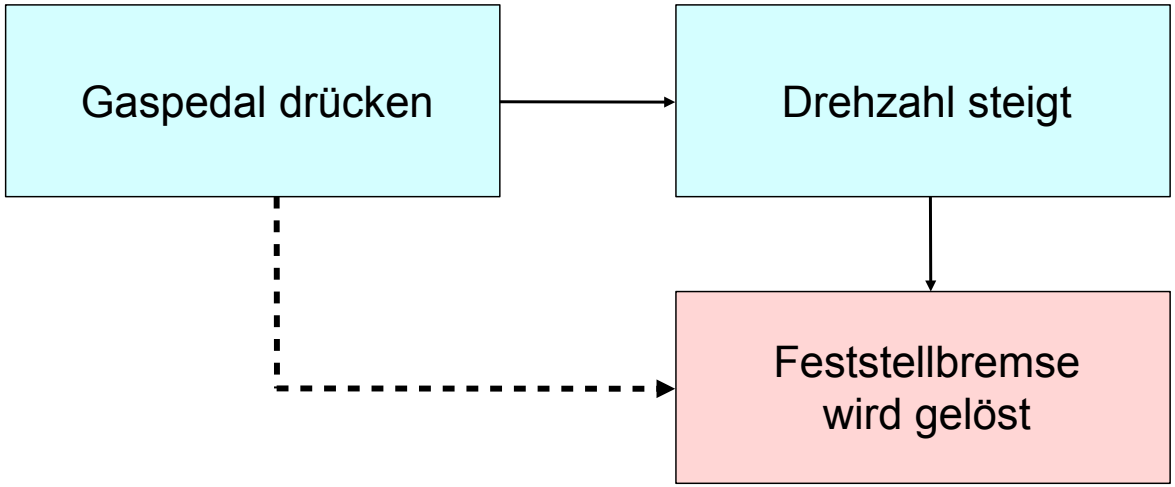


Realisierte Lösung

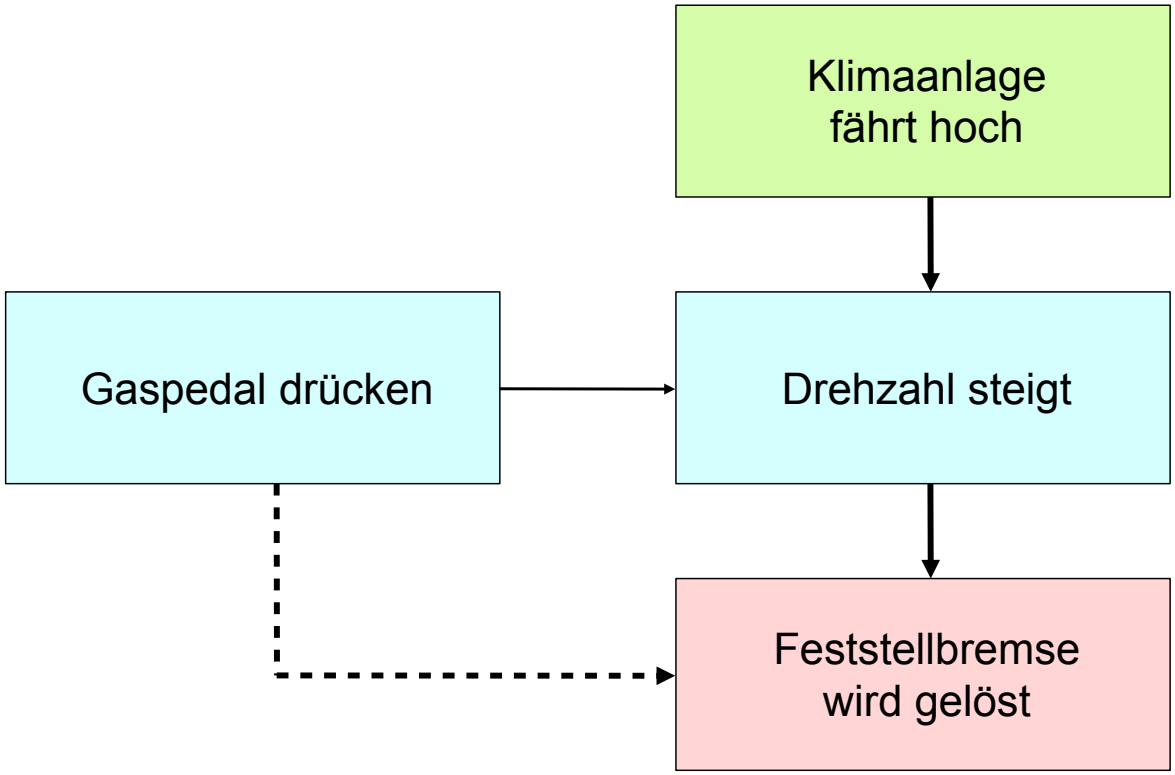
- Kostenziel
- Gewichtssziel
- Drehzahl liegt schon auf CAN

Realisierte Lösung

- Kostenziel
- Gewichtsssziel
- Drehzahl liegt schon auf CAN



Auswirkung

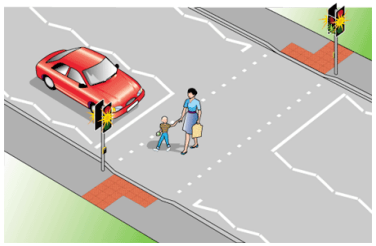

Klimaanlage
fährt hoch
Gaspedal drücken
Drehzahl steigt
Feststellbremse
wird gelöst

Beispiel Micro-Hybrid Stop-Start Feature

- Verifikation einer sicherheitskritischen Anwendung im Automobilbau
  Dr. Thomas Rambow, Ford Forschungszentrum Aachen GmbH
  7. SafeTRANS Industrial Day am 19. November 2009 bei EADS in Friedrichshafen

Example

Hazard: Unintended vehicle lurch



Safety Goal:

- Cranking the engine by the Micro-Hybrid Stop-Start Feature shall not contribute to vehicle movement (transfer torque to wheels) in other than vehicle pull-away maneuvers.

Functional and Technical Safety Concept

SW Safety Requirement:

- If the starter command is CRANK and the gear state is not NEUTRAL then the starter command shall be reset

Hybrid Antrieb

- Ladestrategien
  - Mindestladung der Batterie erhalten
  - Ab definierter Motordrehzahl laden
  - Nur Bremsenergie laden
  - Konstanter Ladestrom
  - Mindestreichweite
  - ...
- Zuschaltung Elektromotor
  - Bis Richtgeschwindigkeit
  - Ortsbezogen
  - Ladungsbezogen
  - Booster (siehe SPIEGEL 13.02.2010)
  - ...

Gliederung

- Vorgehensmodelle
- Funktionale Sicherheit
- **Vorgehensmodelle und funktionale Sicherheit - ISO 26262**

<table>
<tr><td colspan="3">1. Vocabulary</td></tr>
<tr><td colspan="3">2. Management of functional safety</td></tr>
<tr><td rowspan="2">3. Concept phase</td><td colspan="2">4. Product development at the system level</td><td rowspan="2">7. Production and operation</td></tr>
<tr><td>5. Product development at the hardware level</td><td>6. Product development at the software level</td></tr>
<tr><td colspan="3">8. Supporting processes</td></tr>
<tr><td colspan="3">9. ASIL-orientes and safety-oriented analyses</td></tr>
<tr><td colspan="3">10. Guideline on ISO 26262</td></tr>
</table>

Struktur der Teile 4 – 6: Produktentwicklung

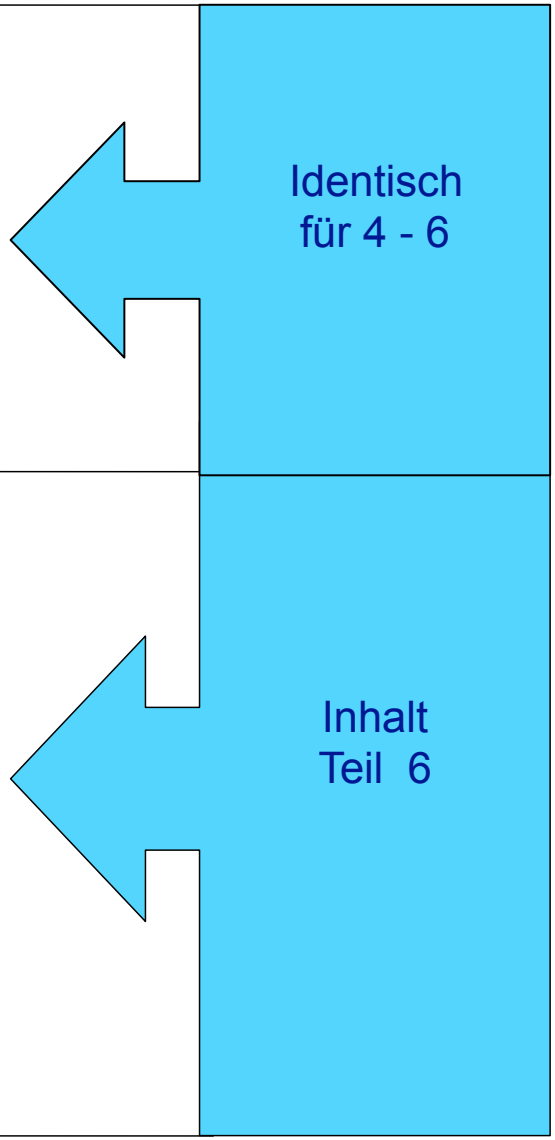
- Part 4: Product development at the system level
- Part 5: Product developmen at the hardware level
- Part 6: Product development: software level

haben die gleiche Struktur und bis 4 Requirements for compliance nahezu identischen Inhalt:

- Foreword
- Introduction
- 1 Scope
- 2 Normative references
- 3 Terms, definitions and abbreviated terms
- 4 Requirements for compliance
- 5 – n: Zusammen mit Annex der eigentliche Inhalt
- Annex A – m
- Bibliography

ISO 26262 Road vehicles - Functional safety
Part 6: Product development: software level

- Foreword
- Introduction
- 1 Scope
- 2 Normative references
- 3 Terms, definitions and abbreviated terms
- 4 Requirements for compliance
- 5 Initiation of product development at the software level
- 6 Specification of software safety requirements
- 7 Software architectural design
- 8 Software unit design and implementation
- 9 Software unit testing
- 10 Software integration and testing
- 11 Verification of software safety requirements
- Annex A – D
- Bibliography



Foreword

Allgemeine Informationen zur Normierung der 26262-6 durch die ISO (the International Organization for Standardization), z.B.

- „ISO 26262-6 was prepared by Technical Committee ISO/TC 22, Roadvehicles, Subcommittee SC3, Electrical and electronic equipment..“

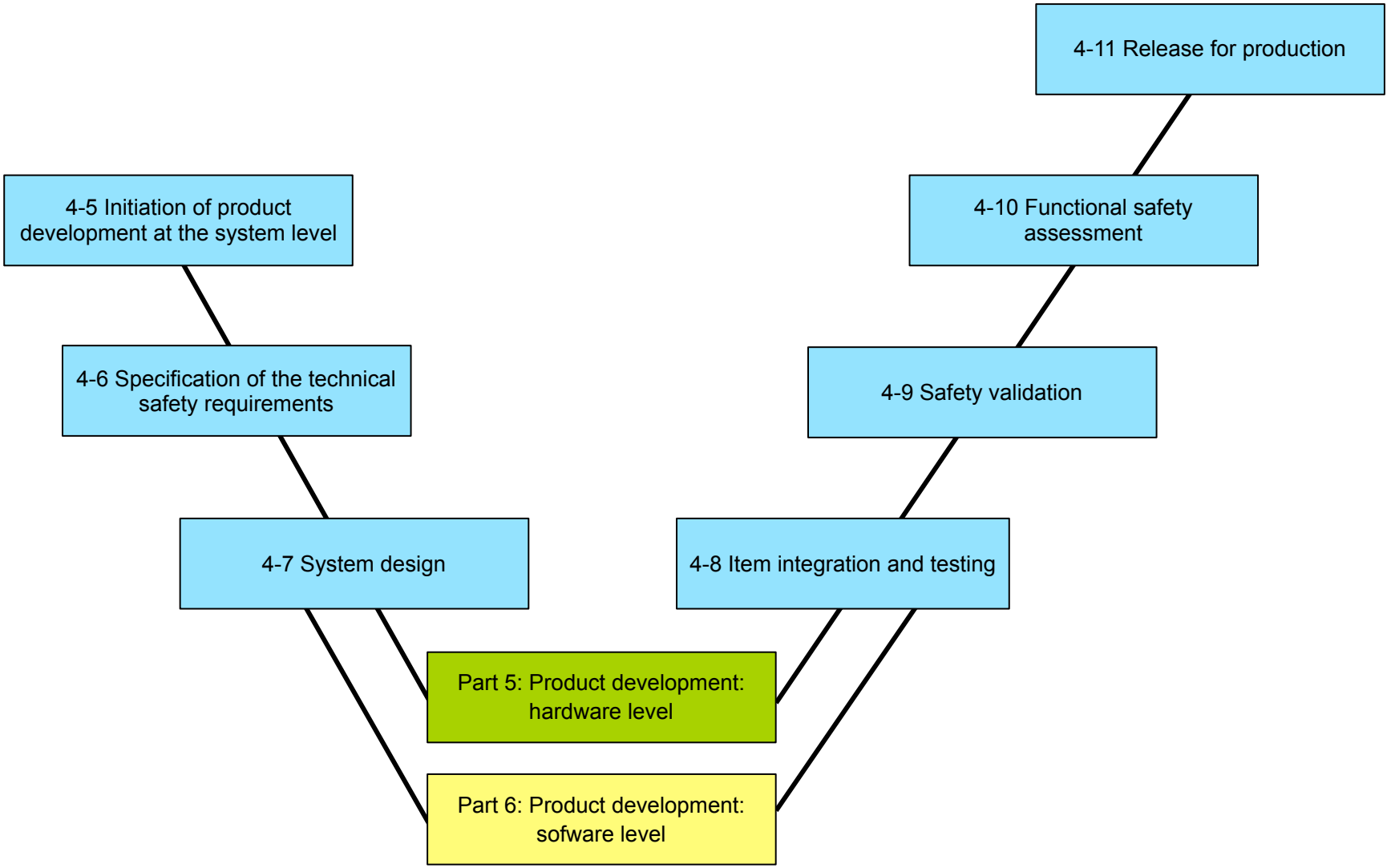
Informationen zur Struktur des Standards:

- Part 1: Vocabulary
- Part 2: Management of functional safety
- Part 3: Concept phase
- Part 4: Product development at the system level
- Part 5: Product development at the hardware level
- Part 6: Product development at the software level
- Part 7: Production and operation
- Part 8: Supporting processes
- Part 9: ASIL-oriented and safety-oriented analyses
- Part 10: Guideline

Introduction

- ISO 26262 ist eine Anpassung der IEC 61508 „Functional safety of electrical/electronic/programmable electronic safety-relatedsystems“ an die speziellen Anforderungen von E/E-Systemen in Strassenfahrzeugen.
- ISO 26262 gibt Leitlinien zum Umgang mit Sicherheitsrisiken, die aus systematischen Fehlern bei der E/E-Entwicklung und aus statistischen Hardware-Ausfällen resultieren.
- ISO 26262 definiert einen Sicherheits-Lebenszyklus
  - Managment
  - Entwicklung
  - Produktion
  - Betrieb
  - Wartung
  - Stilllegung
- ISO 26262 basiert auf einem V-Modell als Referenz für die verschiedenen Phasen der Produktentwicklung

Part 4: Product development at the system level



1 Scope

- Anwendung bei sicherheitsrelevanten Systemen, die ein oder mehrere E/E Systeme einschließen und in PKW bis max. 3,5t installiert sind (die Ausweitung auf grössere Fahrzeuge ist jedoch nicht ausdrücklich verboten)
- Entwicklungen vor der Veröffentlichung sind ausgenommen, bei Weiterentwicklungen und Modifikationen ist die ISO 26262 jedoch anzuwenden.
- ISO 26262 adressiert mögliche Gefahren die durch Störungen im Verhalten von sicherheitsrelevanten E/E-Systemen verursacht werden, einschließlich der Wechselwirkungen zwischen diesen Systemen.
- Es geht nicht um Gefährdungen durch elektrischen Schlag, Feuer, Rauch, Hitze, Strahlung, Giftigkeit, Brennbarkeit, Reaktionsfähigkeit, Korrosion, Freisetzung von Energie, und ähnliche Gefahren, es sei denn, diese sind direkt verursacht durch ein fehlerhaften Verhalten von sicherheitsrelevante E/E-Systemen
- ISO 26262 befasst sich nicht mit den Grundfunktion von E/E-Systeme, auch wenn eigene funktionale Leistungsvorgaben für diese Systeme existieren (z.B. aktive und passive Sicherheitssysteme, Bremssysteme, ACC).

2 Normative references

- Verweise auf weitere Normen, die für die Anwendung von Teil 6 benötigt werden. Dies sind die folgenden Teile der ISO 26262:
- Part 1: Vocabulary
- Part 2: Management of functional safety
- Part 4: Product development at the system level
- Part 5: Product development at the hardware level
- Part 7: Production and operation
- Part 8: Supporting processes
- Part 9: ASIL-oriented and safety-oriented analyses

3 Terms, definitions and abbreviated terms

- Hier wird auf iSO 26262 - Part 1: Vocabulary verwiesen

4 Requirements for compliance

- Erfüllung der ISO 26262 bei der Software-Entwicklung heisst Erfüllung aller Anforderungen aus den Abschnitten / Prozessphasen / Arbeitsschritten
  - 5 Initiation of product development at the software level
  - 6 Specification of software safety requirements
  - 7 Software architectural design
  - 8 Software unit design and implementation
  - 9 Software unit testing
  - 10 Software integration and testing
  - 11 Verification of software safety requirements
- Die konkreten Anforderungen (“Was soll mit welchen Methoden erreicht werden?”) stehen jeweils in den Unterabschnitten x.4 “Requirements and recommendations”
- Ausnahmen
  - Tailoring der Safety-Akitivitäten nach ISO 26262-2 hat ergeben, dass die Anforderung nicht zutrifft
  - Assessment nach ISO 26262-2 mit entsprechender Begründung hat ergeben, dass die Nichterfüllung der Anforderung akzeptabel ist

4 Requirements for compliance

- Interpretation der Tabellen
- Die in den Tabellen aufgeführten Methoden sind entweder
  - aufeinander folgende Einträge (Durchnummeriert mit 1, 2, 3, ...) oder
  - alternative Einträge (Durchnummeriert mit x.a, x.b, x.c, ...)
- Aufeinander folgende Einträge
  - Alle aufgeführten Methoden müssen angewendet werden, sofern sie für den angestrebten ASIL empfohlen sind.
  - Wenn weitere Methoden angewendet werden muss begründet werden, warum mit diesen Methoden die Anforderungen erfüllt werden.
- Alternative Einträge
  - Eine angemessene Kombination von Methoden ist anzuwenden (mit Begründung).
  - Wenn weitere Methoden angewendet werden muss begründet werden, warum mit diesen Methoden die Anforderungen erfüllt werden.
- In Teil 6 SW-Entwicklung gibt es ausschliesslich alternative Einträge

Recommentations / Empfehlungen

Darstellung	Vorgabe (Englisch)	Vorgabe (Deutsch)
++	highly recommended	Verbindlich
+	recommended	Empfohlen
o	no recommendation for or against its usage	keine Empfehlung für oder gegen ihre Verwendung

Beispiel für Methodenauswahl

- 11.4.2 Table 16 - Test environments for conducting the software safety requirements verification

- Steuergerätelieferant / Tier-2 Supplier: 1a, evtl. 1b
- Systemintegrator / Tier-1 Supplier: 1b, evtl. 1a und 1c
- Fahrzeughersteller / OEM: 1c, evtl. 1b

Beispiel für Methodenauswahl

- 11.4.2 Table 16 - Test environments for conducting the software safety requirements verification

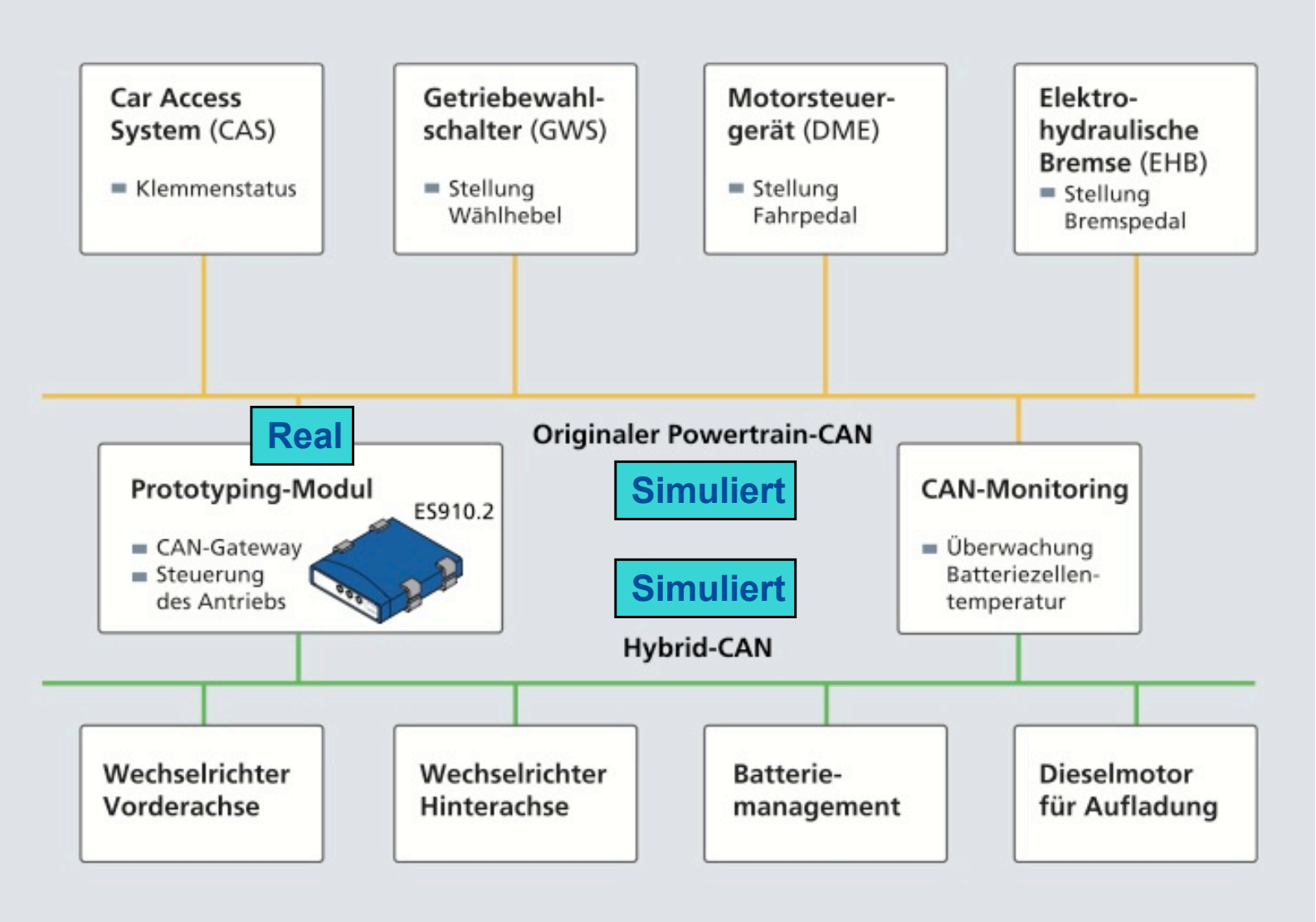
Methods		ASIL			
		A	B	C	D
1a	Hardware-in-the-loop	+	+	++	++
1b	Electronic control unit network environments	++	++	++	++
1c	Vehicles	++	++	++	++

- Steuergerätelieferant / Tier-2 Supplier: 1a, evtl. 1b
- Systemintegrator / Tier-1 Supplier: 1b, evtl. 1a und 1c
- Fahrzeughersteller / OEM: 1c, evtl. 1b

Testumgebungen für integrierte Software

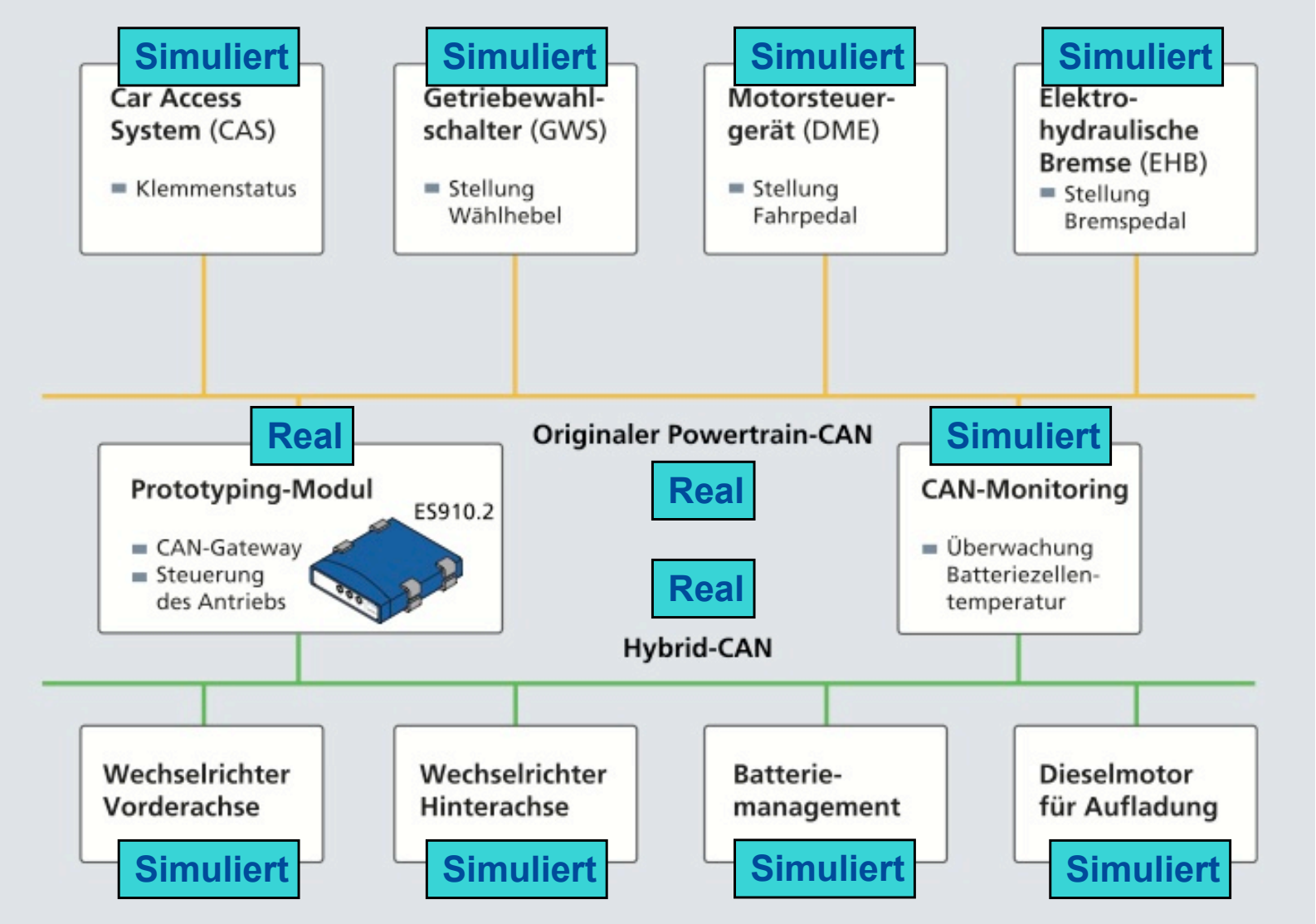
	Steuer-gerät incl. Software	Netz	Weitere Steuer-geräte	Fahrzeug
Hardware-in-the-loop	Real	Simuliert	Simuliert	Simuliert
Electronic control unit network environments	Real	Real	Teils real, teils simuliert	Simuliert
Vehicles	Real	Meist real	Teils real, teils simuliert	Real

Hardware-in-the-loop (HIL)



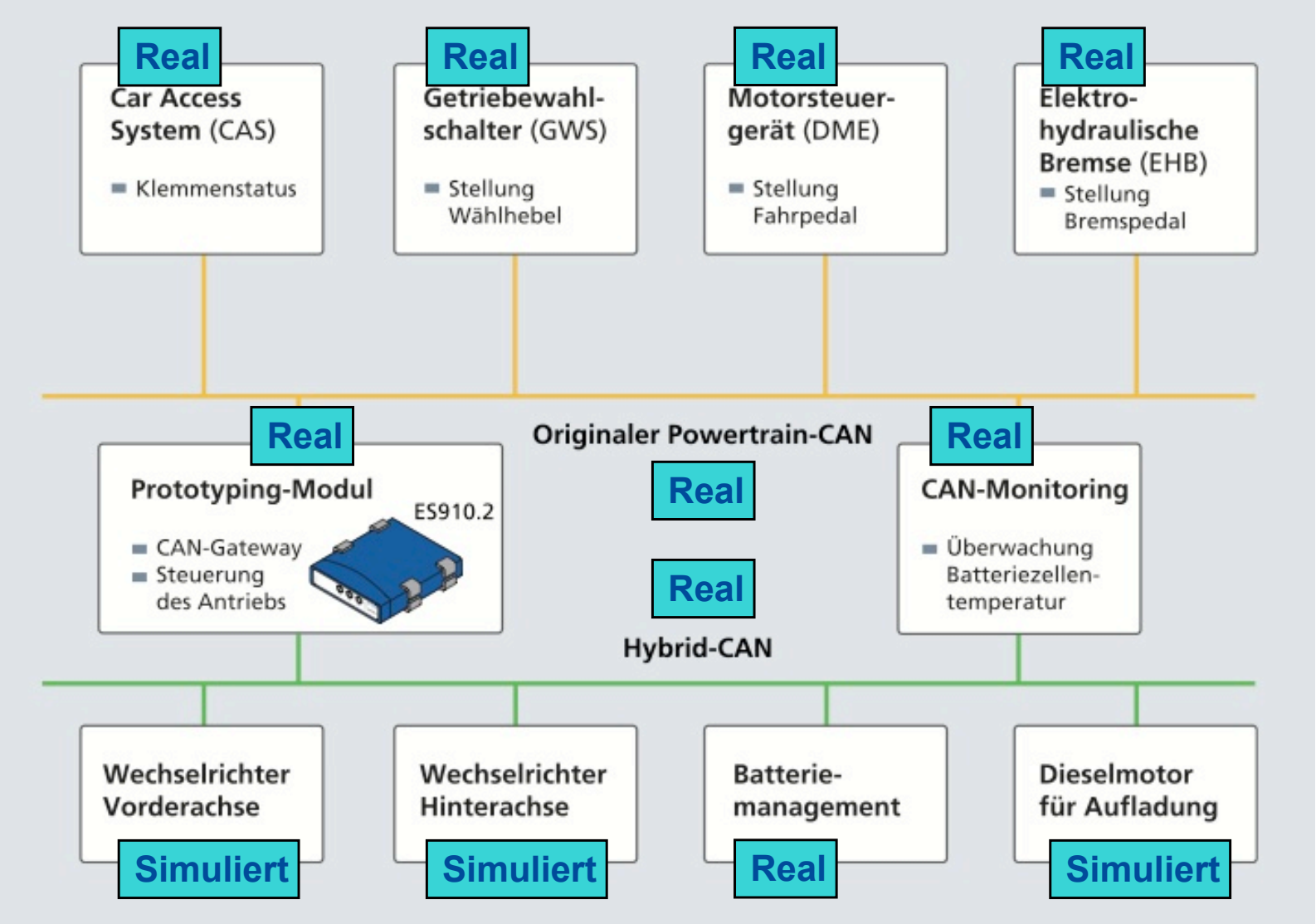
Quelle: Prof. Dr. Dieter Nazareth, FH Landshut
Entwicklung einer Antriebssteuerung für ein Hybridfahrzeug in einer Rapid Prototyping-Umgebung

Electronic control unit network environments
Restbussimulation



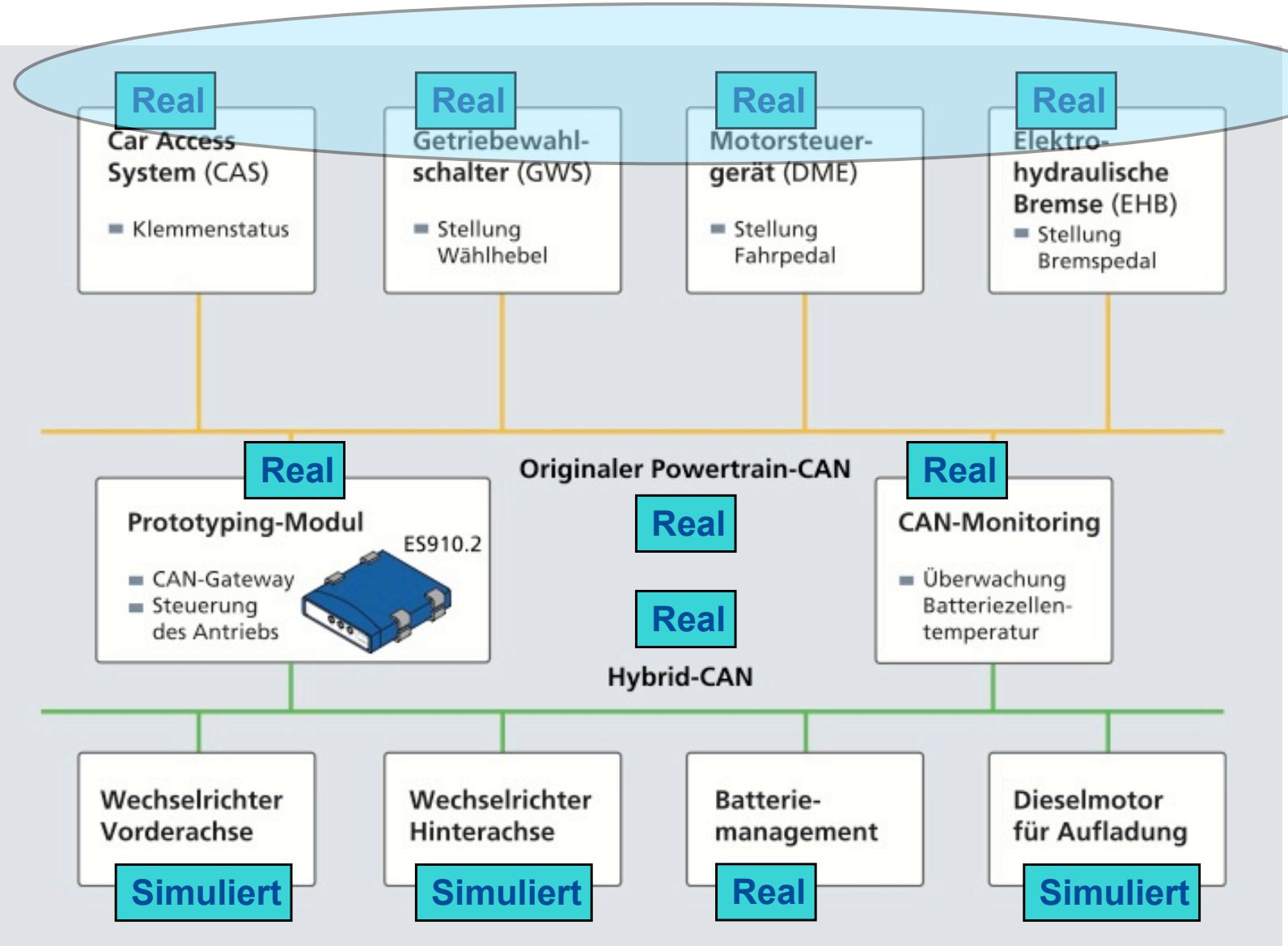
Quelle: Prof. Dr. Dieter Nazareth, FH Landshut
Entwicklung einer Antriebssteuerung für ein Hybridfahrzeug in einer Rapid Prototyping-Umgebung

Electronic control unit network environments mit (teilweise) realen weiteren Steuergeräten



Quelle: Prof. Dr. Dieter Nazareth, FH Landshut
Entwicklung einer Antriebssteuerung für ein Hybridfahrzeug in einer Rapid Prototyping-Umgebung

Electronic control unit network environments mit (teilweise) realen weiteren Steuergeräten

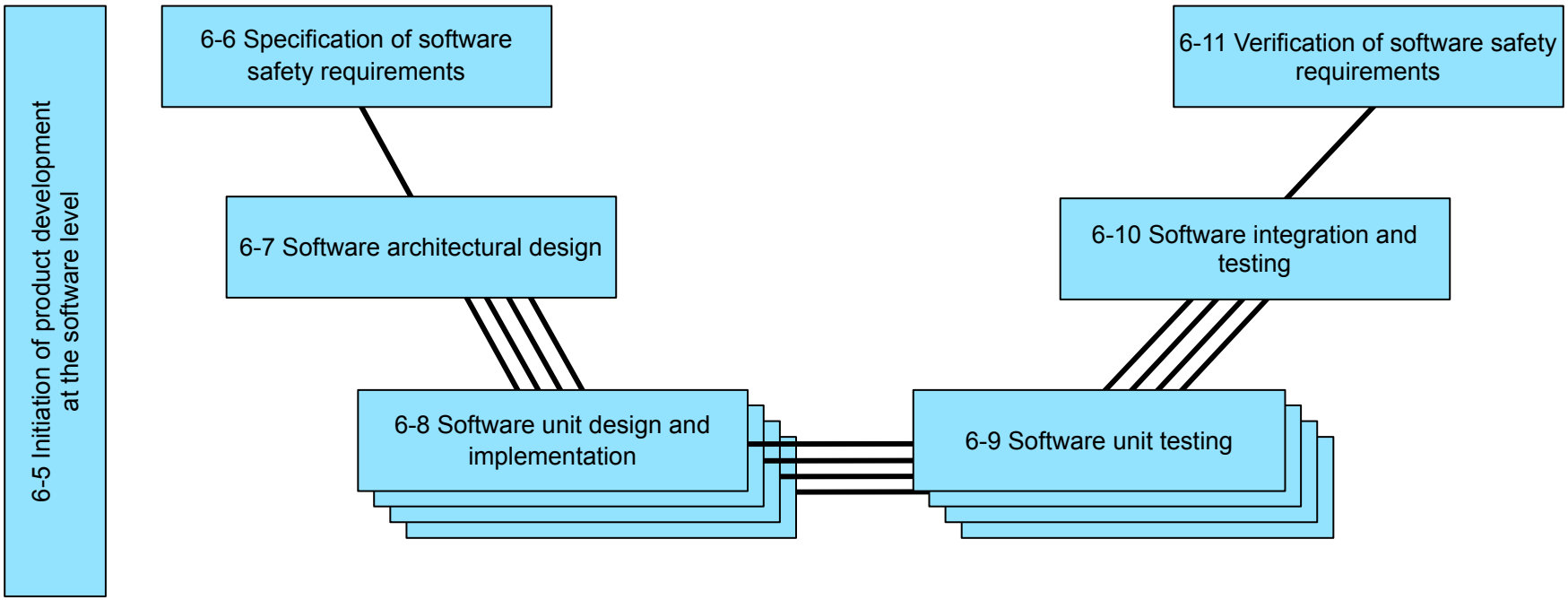


Quelle: Prof. Dr. Dieter Nazareth, FH Landshut
Entwicklung einer Antriebssteuerung für ein Hybridfahrzeug in einer Rapid Prototyping-Umgebung

Vehicle, Vehicle-in-the-loop (VIL), Prüfstand
siehe Teil 8, Beispiele aus der Praxis

Quelle: Prof. Dr. Dieter Nazareth, FH Landshut
Entwicklung einer Antriebssteuerung für ein Hybridfahrzeug in einer Rapid Prototyping-Umgebung

Figure 2
Reference phase model for the software development



Prozessphasen, Voraussetzungen, Arbeitsergebnisse

- Produkte / Work products sind die Arbeitsergebnisse der einzelnen Prozessphasen
- Beispiel: Das Produkt „Software architectural design specification” ist Arbeitsergebnis der Prozessohase 6-7 „Software architectural design”

- Voraussetzungen / Prerequisites sind Arbeitsergebnisse früherer Phasen, die benötigt werden
- Beispiel: Das Produkt „Software architectural design specification” ist Voraussetzung für die Prozessphase 6-8 „Software unit design and implementation”

- Zusatzinformationen / Further supporting informations müssen keine Arbeitsergenisse früherer Prozessphasen sein sondern können auch aus externen Quellen kommen.
- Beispiel: Design and coding guidelines, z.B. MISRA-C

ISO FDIS 26262 Road vehicles - Functional safety
Produkte, Prozessschritte, Rollen
Teil 6: Product development: software level

Information legend for the matrix of Workproducts:
I = Input of a process phase (prerequisites)
O = Output of a process phase
I/O = Rework of the workproduct in that process phase
FI = Further supporting information as input of the process phase
OR = Reviewed Output of the process phase

	Teil 6: Product development: software level	5 Initiation of product development at the software level	6 Specification of software safety requirements	7 Software architectural design	8 Software unit design and implementation	9 Software unit testing	10 Software integration and testing	11 Verification of software safety requirements
1	Overall project plan (refined)	I						
2	Safety plan (refined)	I/O	I	I/O	I	I	I	I
3	Item integration and testing plan	I						
4	Technical safety concept	I	I	FI	FI			FI
5	System design specification	I	I	FI	FI			FI
6	Hardware Software Interface Specification (HSI)		I/O	I	FI	I	I	
7	Validation plan							FI
8	Integration testing report							I
9	Software safety requirements specification		O	I/O	I			I
10	Hardware design specification		FI					
11	Software verification plan	O	I/O	I	I	O	I/O	I/O
12	Software verification report		O	I/O	I/O	I/OR	I/O	I/O
13	Design and coding guidelines for modelling and programming languages	FI/O		I	I			
14	Tool application guidelines	O		FI	FI	FI	FI	FI
15	Guidelines for the application of methods	FI	FI	FI	FI	FI	FI	FI
16	Guidelines for the application of tools	FI						
17	Qualified software components available	FI		FI			FI	
18	Qualified software tools available	FI						
19	Software architectural design specification			O	I		I	I
20	Safety analysis report			O	FI			
21	Dependant failures analysis report			O				
22	Software unit design specification				O	I		
23	Software unit implementation				O	I	I	
24	Software verifcation specification					O	I/O	I/O
25	Embedded software						O	
26	Software tool qualification report						FI	

Explizit in 26262:
Prozessschritte mit Voraussetzungen und Arbeitsergenissen

Implizit in 26262:
Arbeitsergenisse mit Prozessschritten, in denen sie gebraucht bzw verändert werdem

Sichten auf das Vorgehensmodell

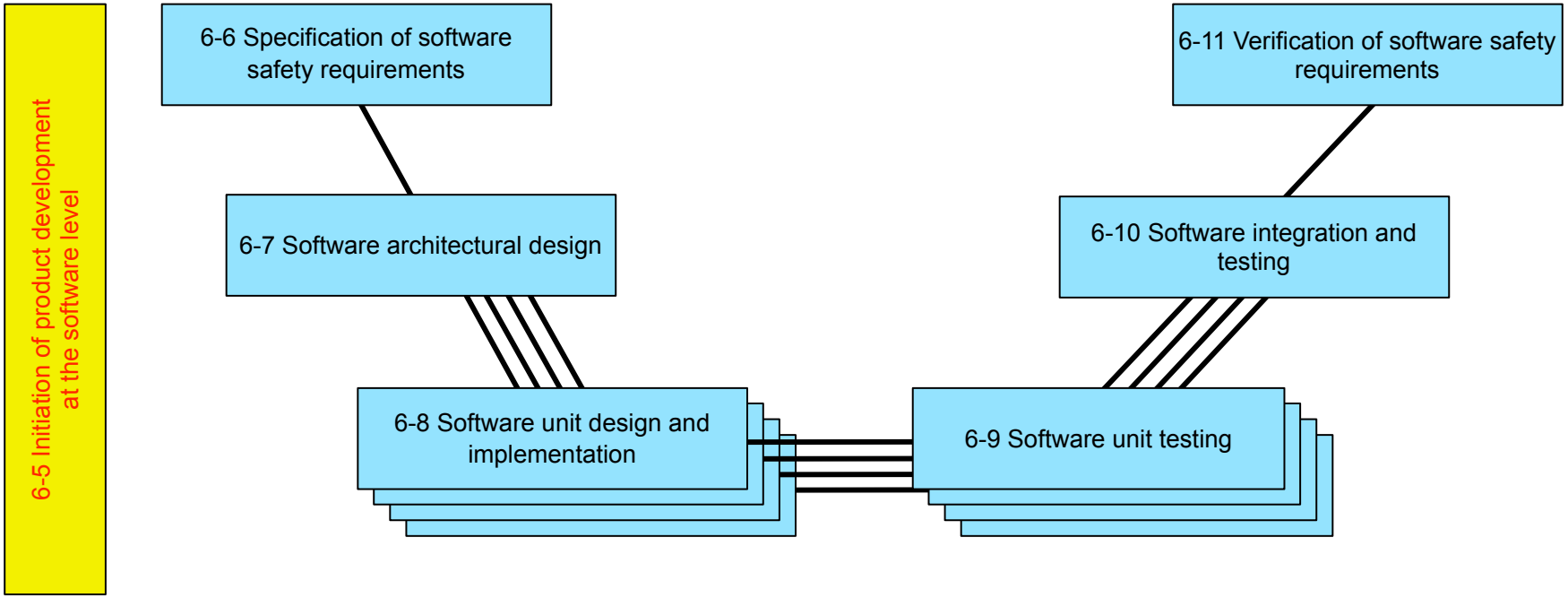
- Wer?
  Rollenorientierte Sicht:
  Wer ist für welche Arbeitsergebnisse (Produkte, Work Products) verantwortlich?
- Was?
  Produktorientierte Sicht:
  Was sind die zu erarbeitenden Produkte?
- Wie?
  Aktivitätenorientierte Sicht:
  Wie werden die Produkte erarbeitet?
- Wann?
  Phasenorientierte Sicht:
  Wann werden welche Arbeitsschritte durchgeführt?
  Meilensteinorientierte Sicht:
  Wann werden welche Produkte fertiggestellt und geprüft?
- Womit?
  Methoden- und werkzeugorientierte Sicht:
  Womit (Methoden und Werkzeuge) werden die Produkte erarbeitet?

Sichten auf das Vorgehensmodell

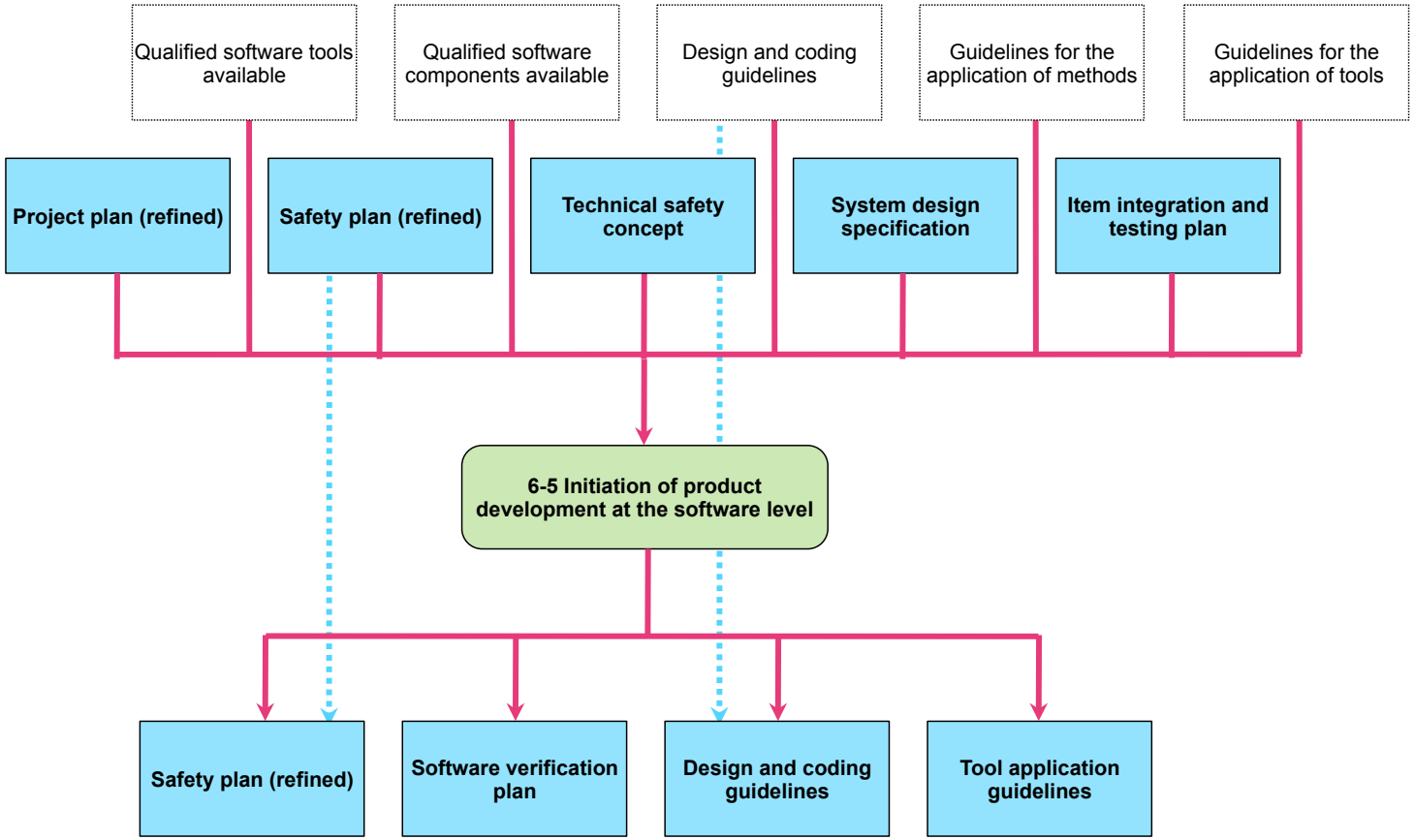
- Wer?
  Rollenorientierte Sicht:
  Wer ist für welche Arbeitsergebnisse (Produkte, Work Products) verantwortlich?
- Was?
  Produktorientierte Sicht:
  Was sind die zu erarbeitenden Produkte?
- Wie?
  Aktivitätenorientierte Sicht:
  Wie werden die Produkte erarbeitet?
- Wann?
  Phasenorientierte Sicht:
  Wann werden welche Arbeitsschritte durchgeführt?
  Meilensteinorientierte Sicht:
  Wann werden welche Produkte fertiggestellt und geprüft?
- Womit?
  Methoden- und werkzeugorientierte Sicht:
  Womit (Methoden und Werkzeuge) werden die Produkte erarbeitet?

✓

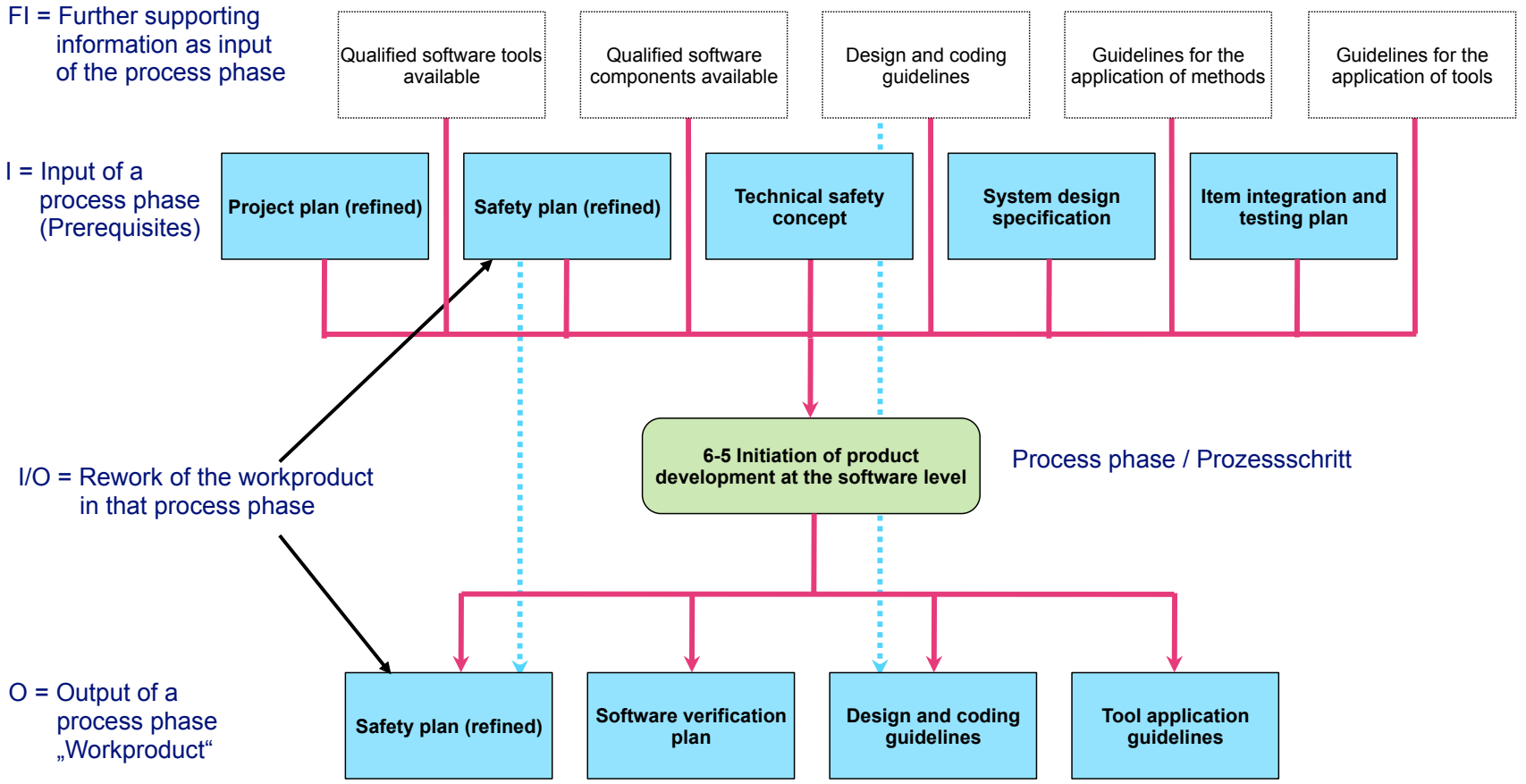
Part 6: Product development: software level
6-5 Initiation of product development at the software level



6-5 Initiation of product development at the software level

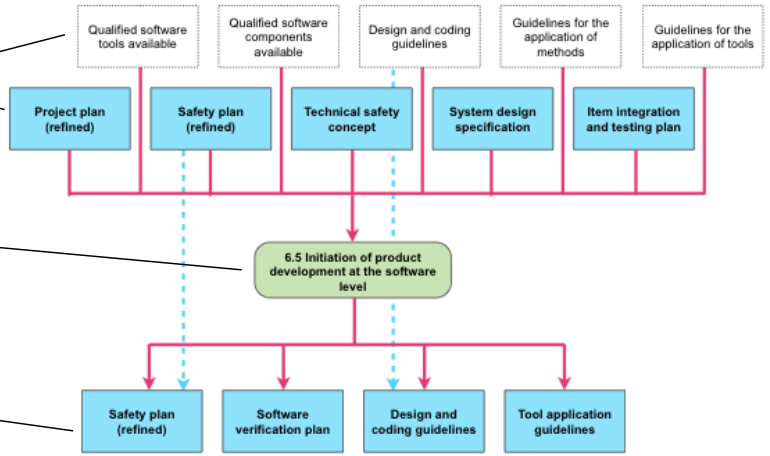


6-5 Initiation of product development at the software level



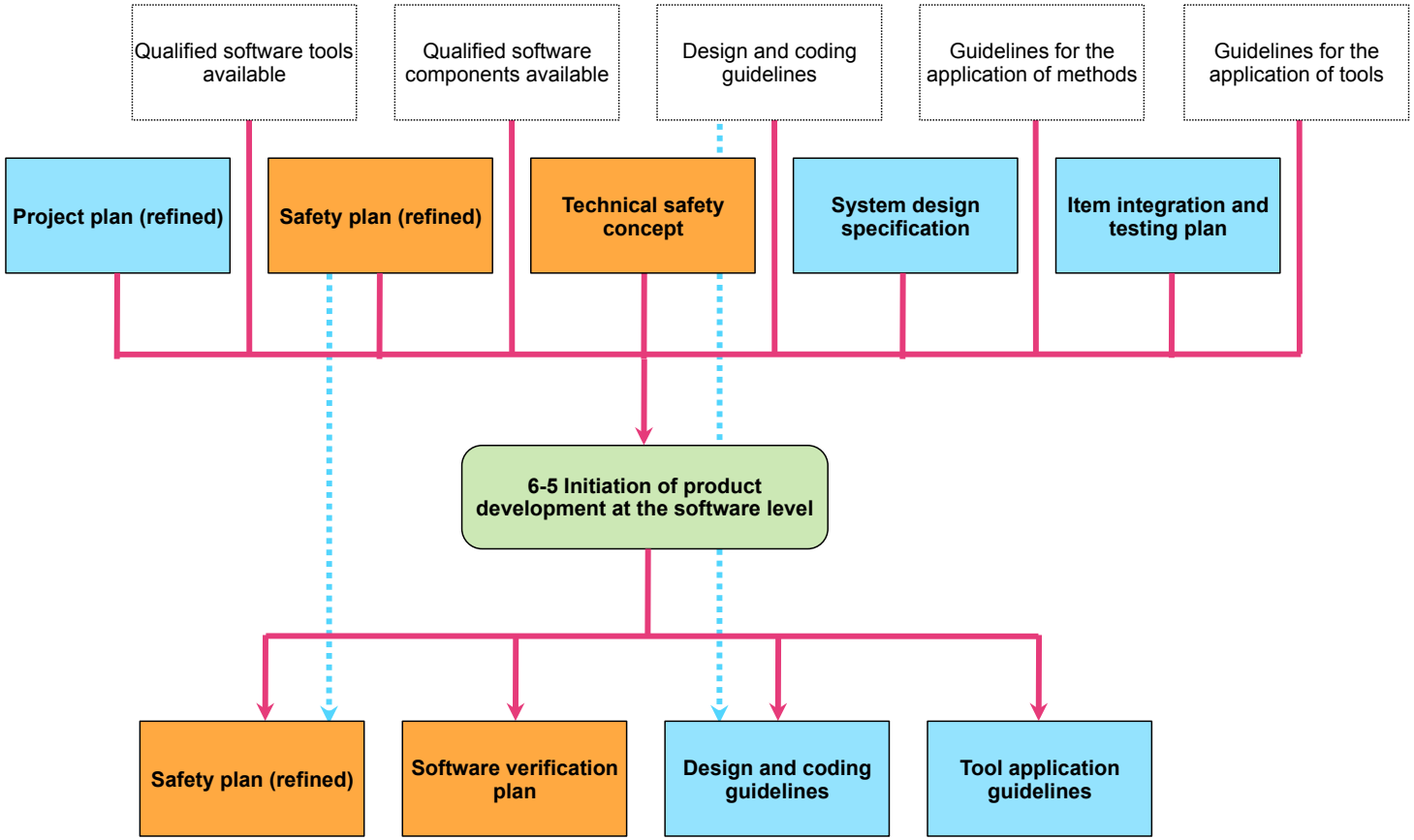
6-5 Initiation of product development at the software level

- Part 6: Product development: software level
- 5 Initiation of product development at the software level
- 5.1 Objectives
- 5.2 General
- 5.3 Inputs to this clause
- 5.3.1 Prerequisites
  Voraussetzungen für diesen Prozessschritt
- 5.3.2 Further supporting information
  Zusatzinformationen
- 5.4 Requirements and recommendations
  Inhalte und Methoden
- 5.5 Work products
  Arbeitsergebnisse dieses Prozessschrittes
- Gleiche Struktur bei den anderen sechs Prozessschri



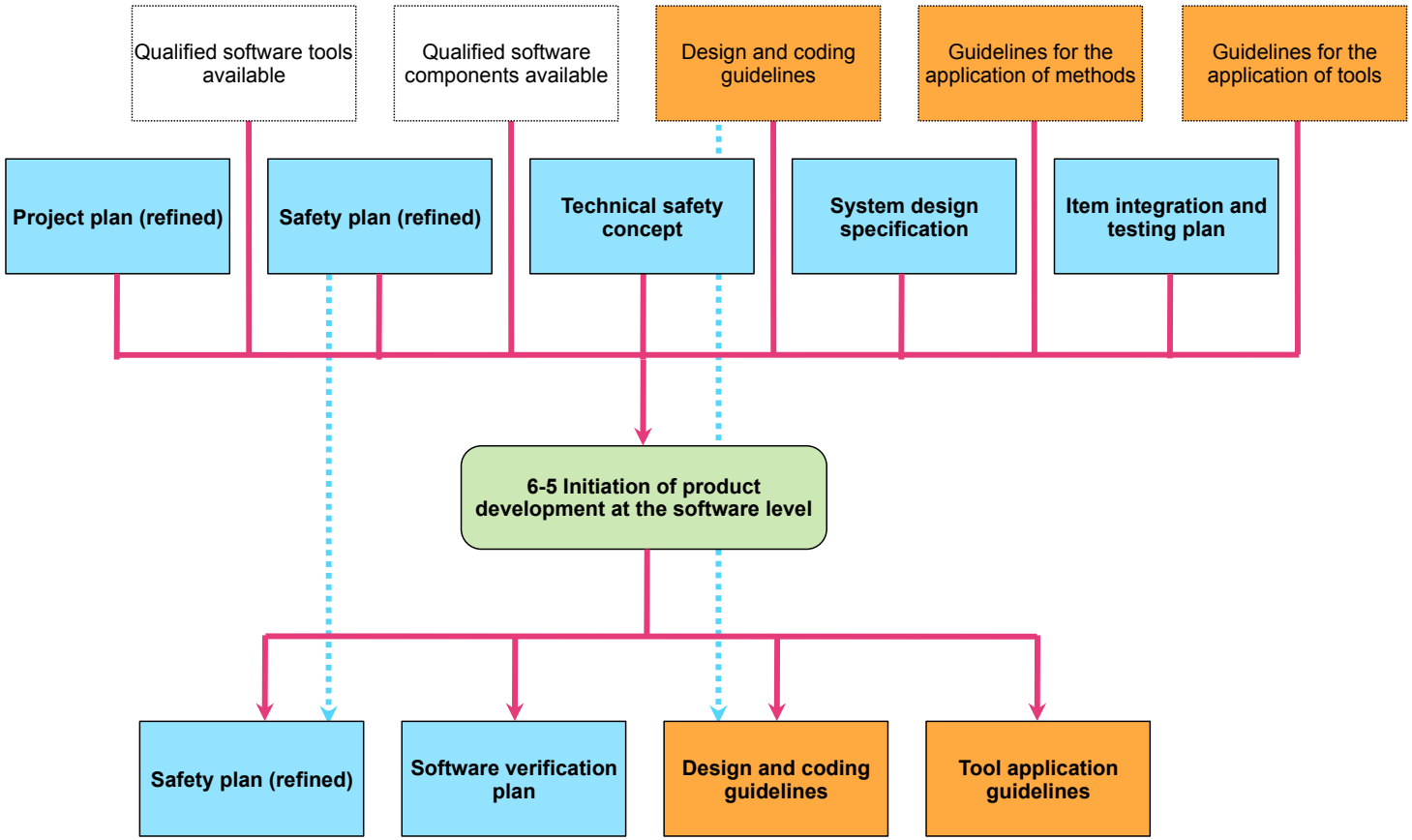
6-5 Initiation of product development at the software level

- Identifikation und Auswahl geeigneter Methoden zur Erfüllung der Anforderungen entsprechend ASIL.



6-5 Initiation of product development at the software level

- Auswahl entsprechender Anwendungsrichtlinien und Werkzeuge.



Typisches Arbeitsergebnis
Design and coding guidelines for modelling and programming languages

- Kriterien für die Auswahl von Modellierungs- oder Programmiersprachen
- Eindeutige Definition
  - Beispiel: Syntax und Semantik der Sprache
- Unterstützung für eingebettete Realzeit-Software
- Behandlung von Laufzeitfehlern
- Unterstützung von Modularität
- Unterstützung von Abstraktion
- Unterstützung von Strukturierung.
- Punkte, die nicht von der Sprache abgedeckt werden, sollen durch entsprechende Richtlinien oder durch die Entwicklungsumgebung abgedeckt werden.
- Richtlinien für Modellierungs- oder Programmiersprachen sollen die Punkte in Tabelle 1 abdecken.

Typisches Arbeitsergebnis
Design and coding guidelines for modelling and programming languages

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

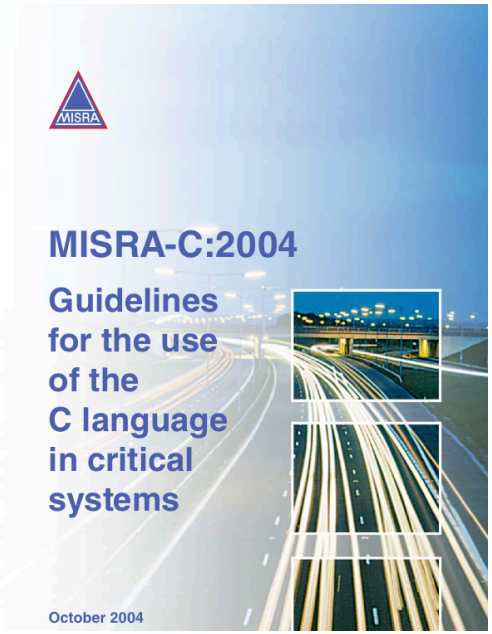
Typisches Arbeitsergebnis
Design and coding guidelines for modelling and programming languages

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

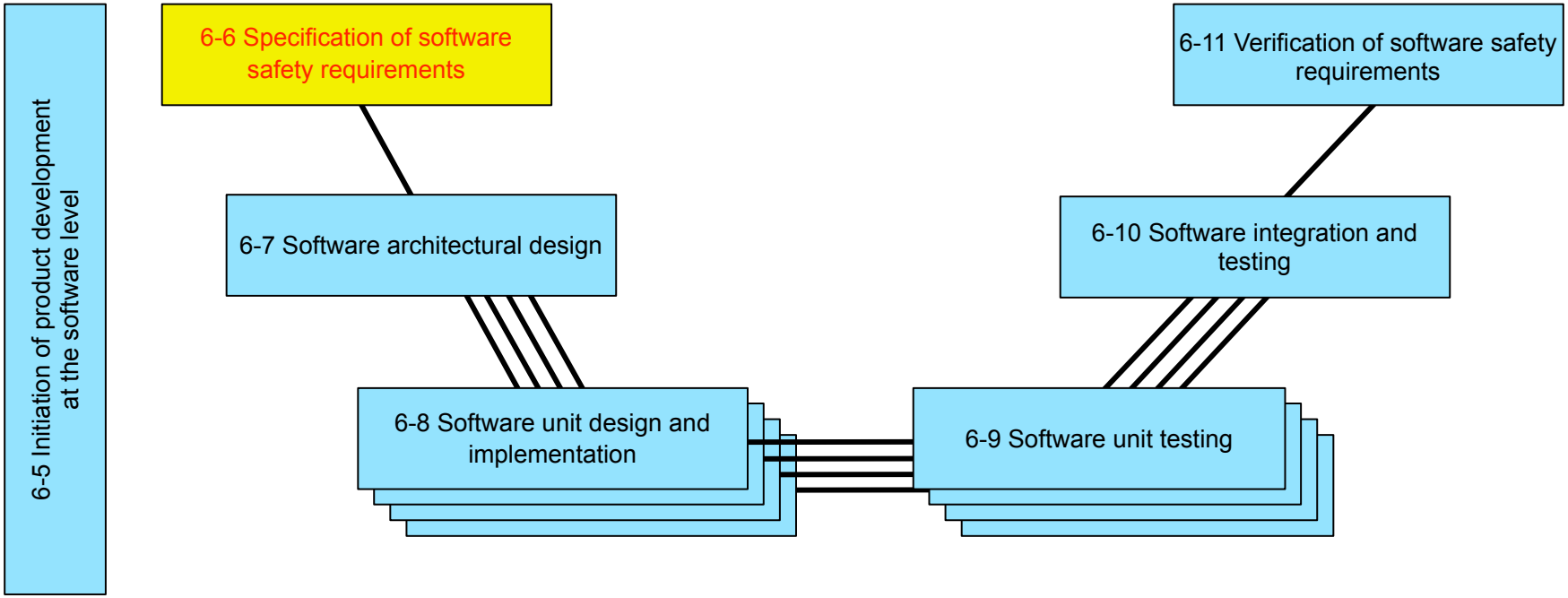
Topics		ASIL			
		A	B	C	D
1a	Enforcement of low complexity	++	++	++	++
1b	Use of language subsets	++	++	++	++
1c	Enforcement of strong typing	++	++	++	++
1d	Use of defensive implementation techniques	0	+	++	++
1e	Use of established design principles	+	+	+	++
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+	++	++	++
1h	Use of naming conventions	++	++	++	++

Typisches Arbeitsergebnis von 6-5
Design and coding guidelines for modelling and programming languages

- „Design and coding guidelines for modelling and programming languages” sind sowohl Zusatzinformationen / Further supporting informations als auch Arbeitsergenis der Prozessphase „6-5 Initiation of product development at the software level”.
- In der Praxis werden allgemeine oder firmen- bzw. anwendungsspezifische Richtlinien übernommen und ggf. für ein konkretes Projekt angepasst.
- Beispiel:
  MISRA-C:2004
  Guidelines for the use of the C language
  in critical systems
- MISRA Mission Statement: To provide assistance to the automotive industry in the application and creation within vehicle systems of safe and reliable software.
- Internet MISRA: www.misra.org.uk
- Internet MISRA-C: www.misra-c.com

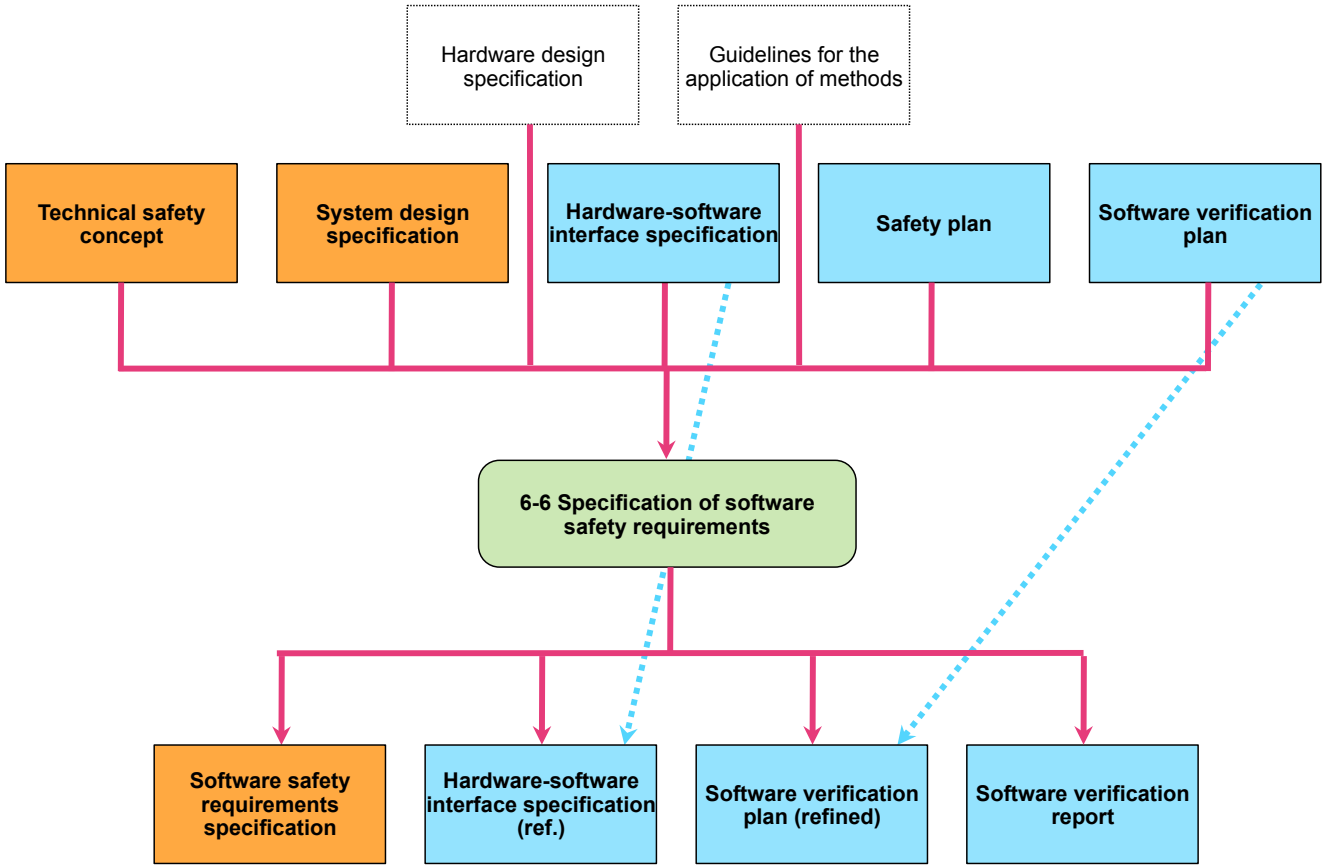


Part 6: Product development: software level
6-6 Specification of software safety requirements



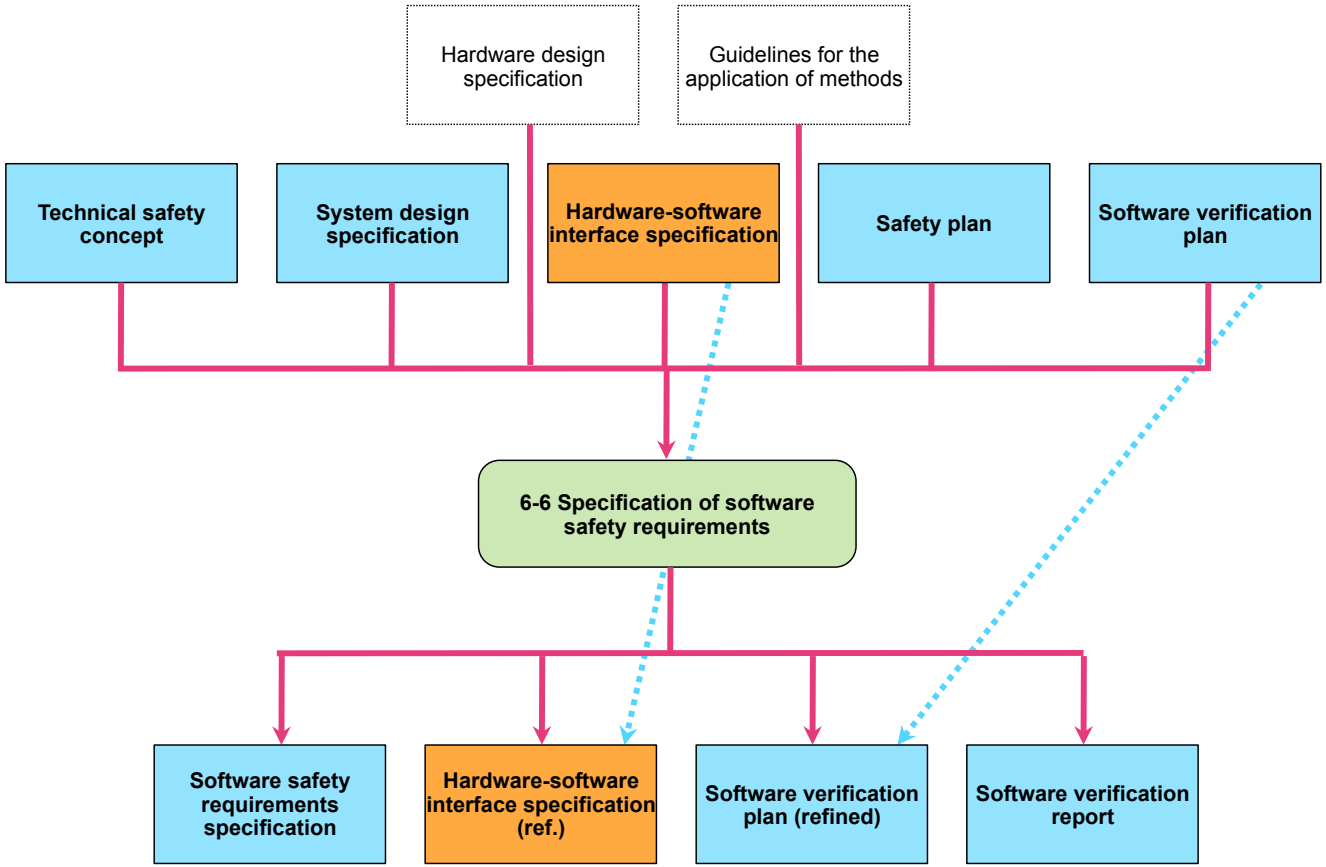
6-6 Specification of software safety requirements

- Spezifikation der Sicherheitsanforderungen an die Software, abgeleitet aus dem Technischen Sicherheitskonzept und dem Systementwurf.



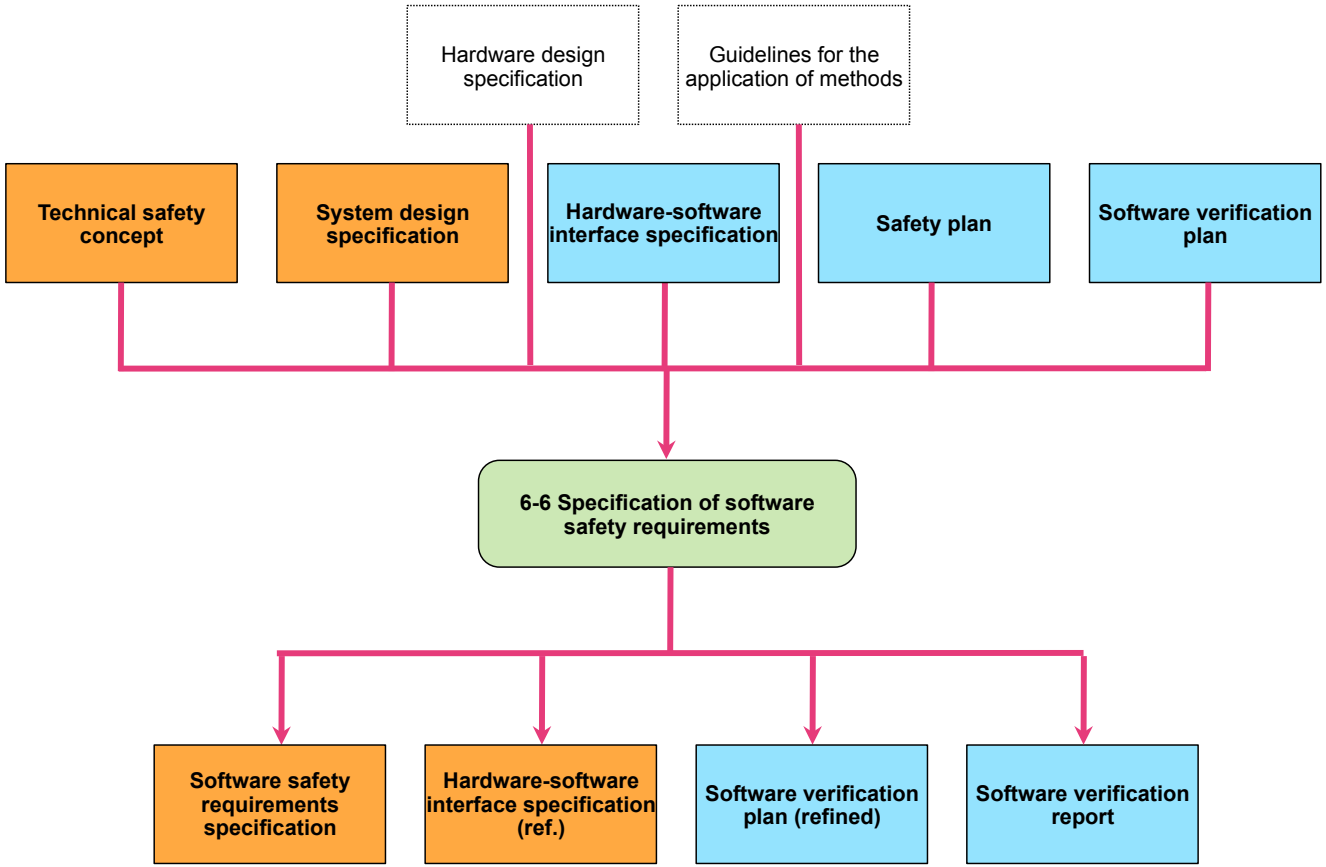
6-6 Specification of software safety requirements

- Verfeinerung der Hardware-Software Schnittstellenanforderungen aus ISO 26262-4-7 7 System design



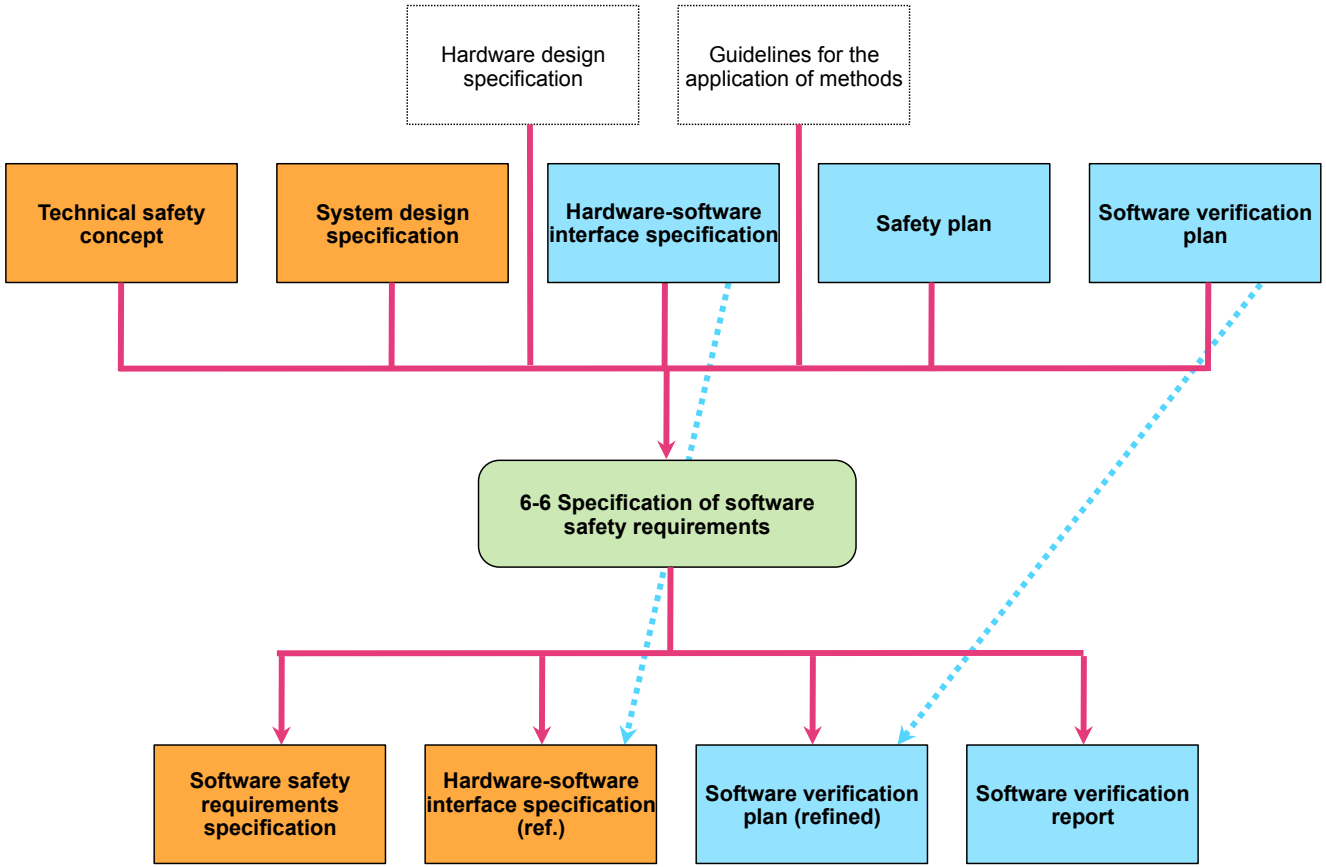
6-6 Specification of software safety requirements

- Konsistenz der Sicherheitsanforderungen an die Software und der Hardware-Software Schnittstellenanforderungen mit dem Technischen Sicherheitskonzept und dem System-Entwurf.



6-6 Specification of software safety requirements

- Konsistenz der Sicherheitsanforderungen an die Software und der Hardware-Software Schnittstellenanforderungen mit dem Technischen Sicherheitskonzept und dem System-Entwurf.



Typisches Arbeitsergebnis von 6-6
Software safety requirements specification

- 6.5 Work products
  6.5.1 “Software safety requirements specification resulting from requirements 6.4.1 to 6.4.3 and 6.4.5.”
- 6.4.1 Die “Software safety requirements” sollen jede software-basierte Funktion umfassen, deren Fehlverhalten zur Verletzung einer durch Software realisierten Tecnischen Sicherheitsanforderung führen kann.
- Beispiele
  - Funktionen, die die Erreichung oder das Beibehalten eines sicheren Zustands ermöglichen
    - Sicherer Zustand: Stehendes Fahrzeug
    - Funktionen: Bremsfunktionen (Betriebsbremse, Feststellbremse)
  - Funktionen, die das Fehlverhalten von sicherheitsrelevanten Hardware-Elementen entdecken, anzeigen und handhaben
    - Sicherheitsrelevantes Hardware-Element: Batterie
    - Funktionen: Messen, Anzeigen und Regeln der Batterietemperatur
  - Entsprechend für Software, z.B. das Monitoring im Betriebssystem

Typisches Arbeitsergebnis von 6-6
Software safety requirements specification

- 6.4.2 Die Spezifikation der “Software safety requirements” soll aus dem Technischen Sicherheitskonzept (ISO 26262-4:—, 7.4.1) und aus dem System-Entwurf (ISO 26262-4:—, 7.4.5) abgeleitet werden.
- Folgendes soll berücksichtigt werden (Auswahl):
  - Spezifikation und Management der Sicherheitsanforderungen (ISO 26262-8:—, Clause 6 “Specification and management of safety requirements”);
  - Die relevanten Anforderungen aus der “Hardware design specification”;
- 6.4.3 Behandelt ASIL Dekomposition, siehe Teil 4. „Konzeptphase und Ableitung des ASIL“
- 6.4.5 Wenn in der Software weitere nicht sicherheitsrelevante Funktionen realisiert sind, so sind diese zu spezifizieren bzw. Ihre Spezifikation ist zu referenzieren.

Beispiel Zeitbeschränkungen

- Ziele von 6-6 Specification of software safety requirements
- Bei der Spezifikation der Sicherheitsanforderungen an die Software sind Beschränkungen durch die Hardware zu berücksichtigen
  - Beispiele: Rechenzeit, Speicherplatz
- Anforderung 6.4.2

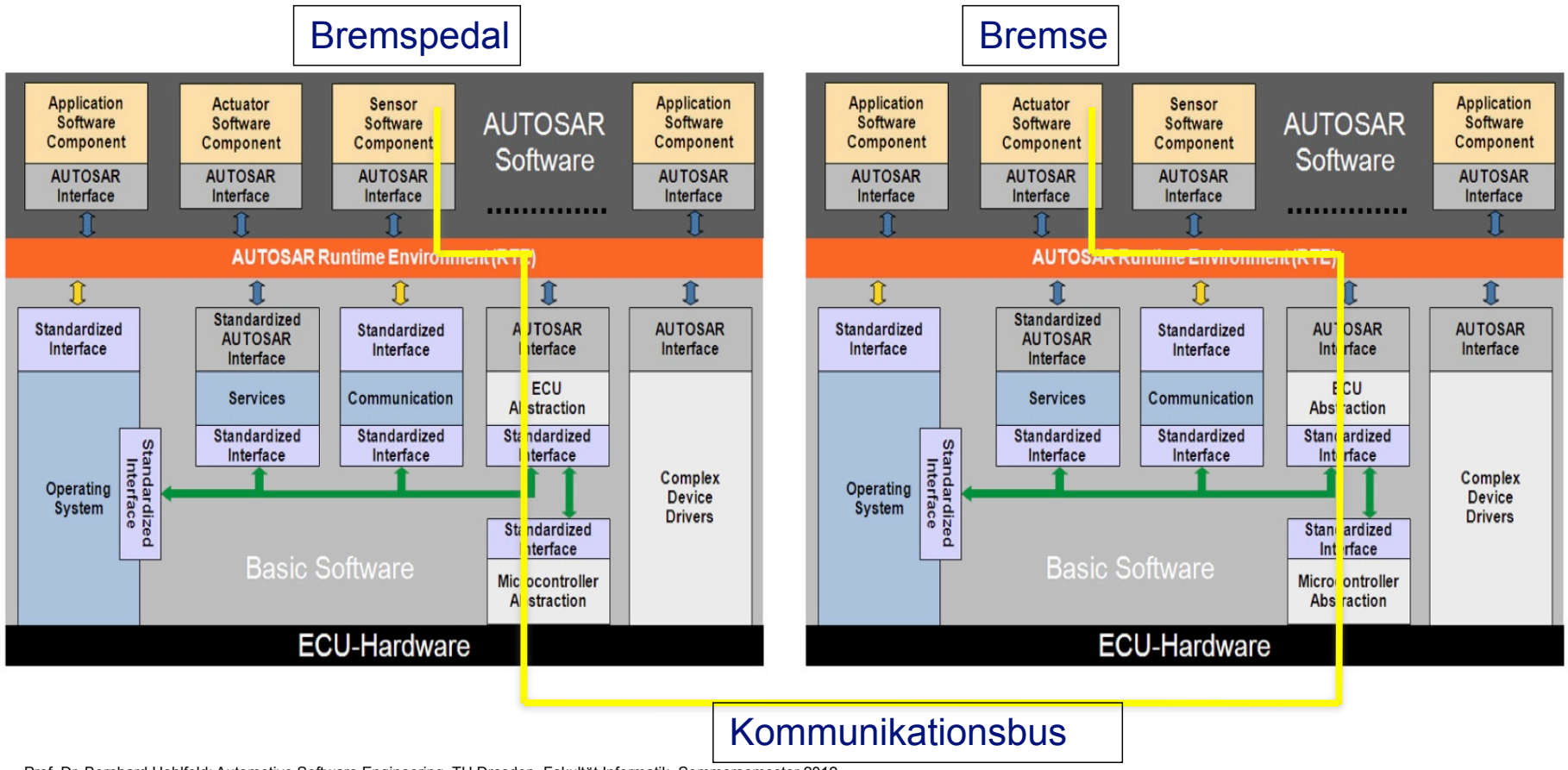
  Zeitbeschränkungen
  Beispiel: Ausführungs- oder Reaktionszeit, abgeleitet aus der geforderten Antwortzeit auf Systemebene

  Der Bremsvorgang muss innerhalb von 1/100 Sekunde eingeleitet werden.
  Anmerkung: Schätzwert, entspricht 10 cm Weg bei 36 km/h und 50 cm Weg bei 180 km/h

  Wenn der Fahrer bei 180 km/h 1 Sekunde nicht aufmerksam ist: Welche Strecke legt das Fahrzeug in dieser Sekunde zurück?

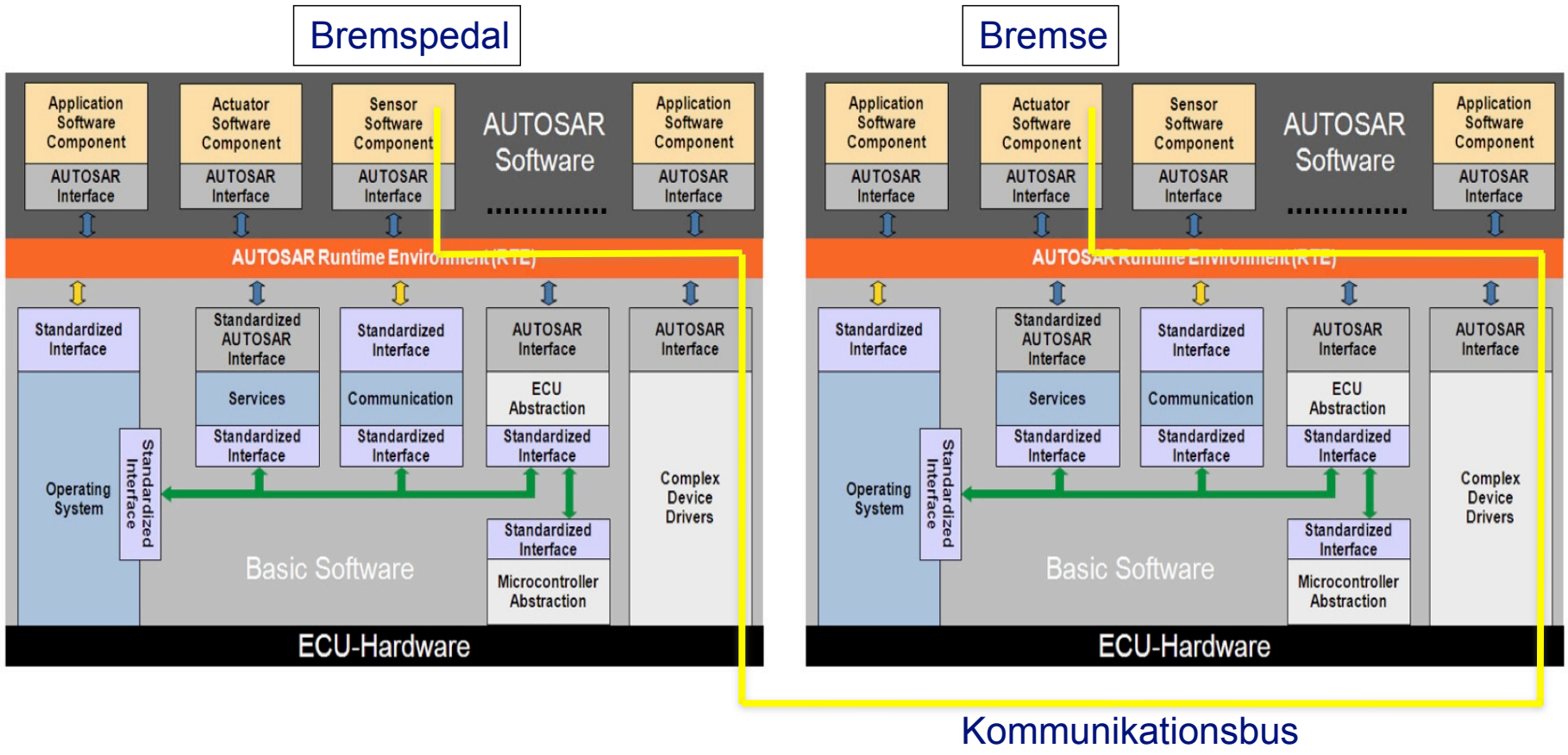
Beispiel Zeitbeschränkungen

- Bremsen mit Verwendung des AUTOSAR-Schichtenmodells:
  Zu langsam durch die vielen Software-Schichten

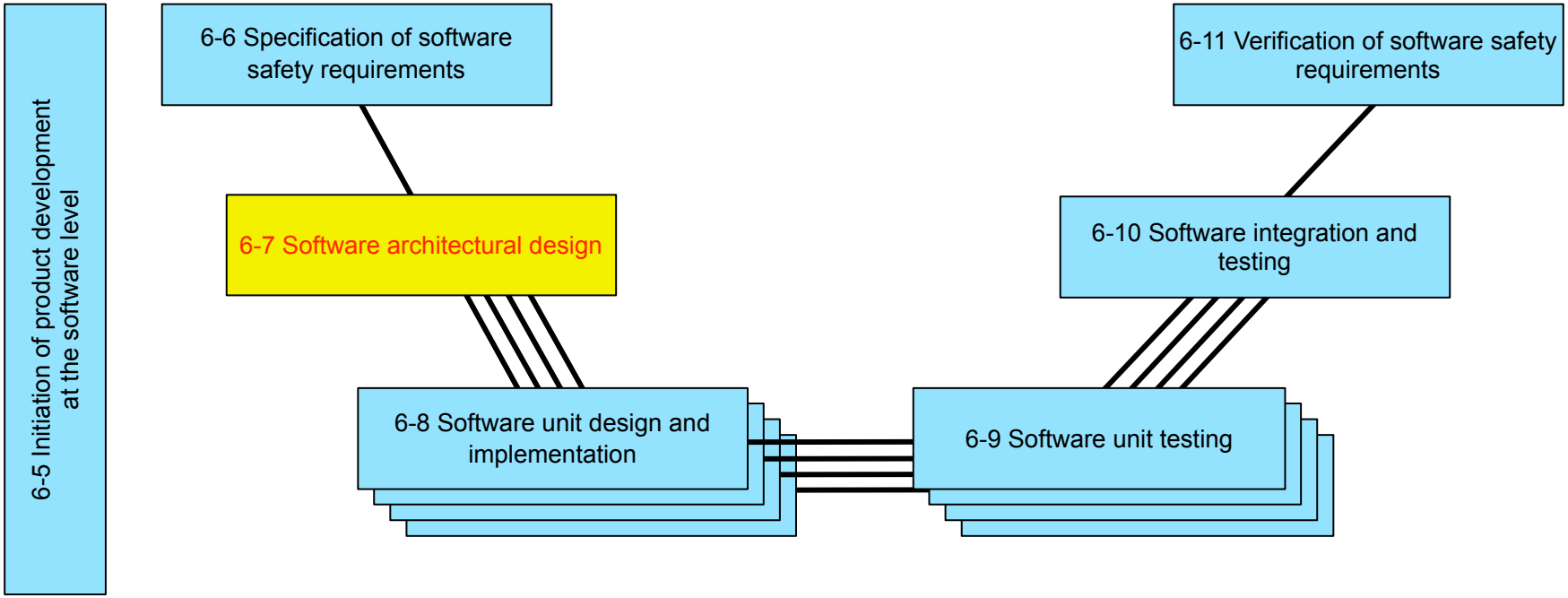


Beispiel Zeitbeschränkungen

- Lösung: Direkter Zugriff auf die ECU-Hardware mit „Complex Drivers“



Part 6: Product development: software level
6-7 Software architectural design

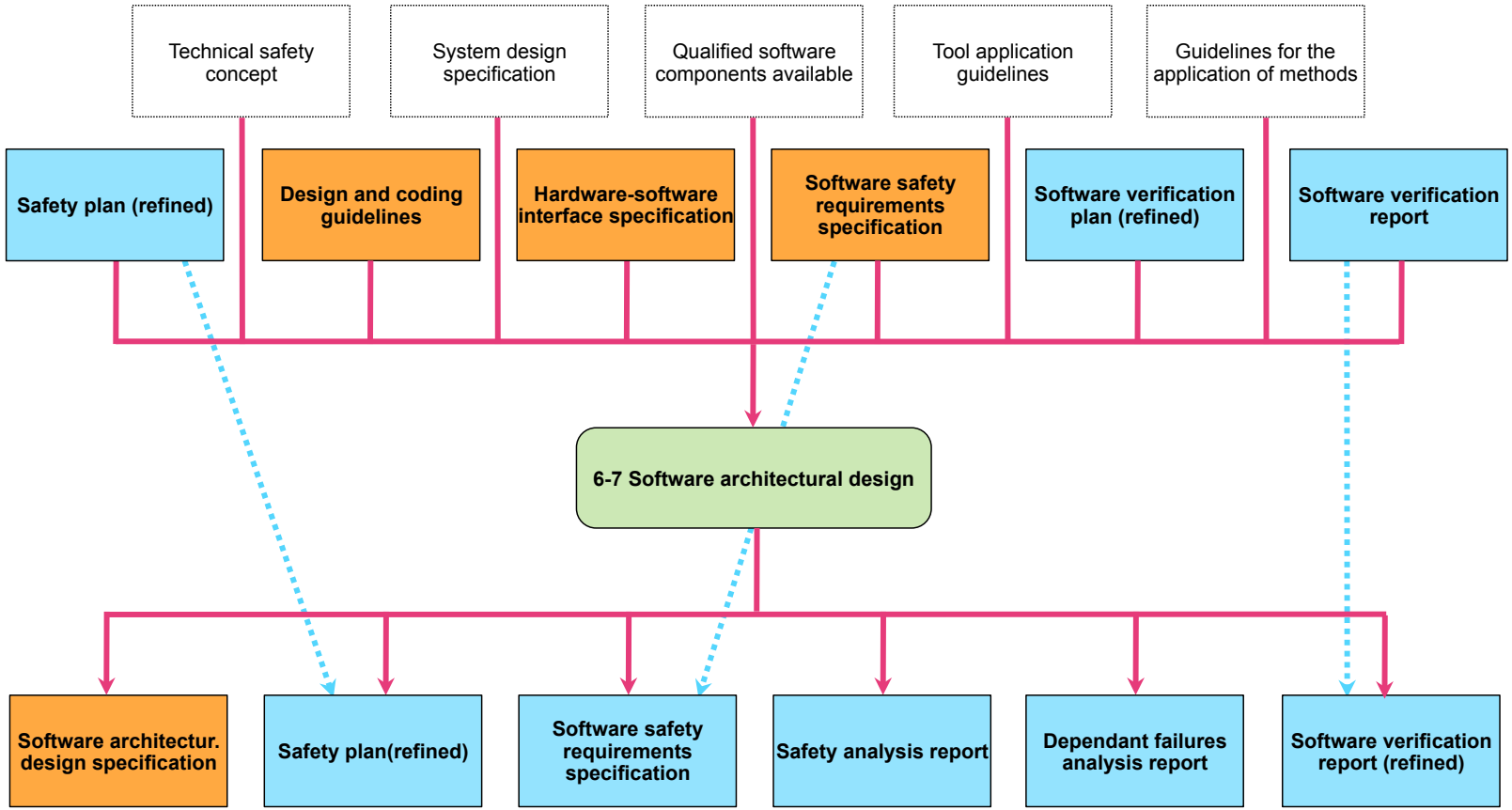


6-7 Software architectural design

- Ziele
  - Entwicklung einer SW-Architektur, die die Sicherheitsanforderungen an die SW umsetzt
  - Verifikation des Entwurfs der SW-Architektur
- Die SW-Architektur beschreibt / ist
  - Alle SW-Komponenten und ihr Zusammenwirken
  - Statische Aspekte
    - Schnittstellen
    - Datenfluss
  - Dynamische Aspekte
    - Reihenfolge der Abarbeitung der Prozesse
    - Zeitverhalten
  - Verteilung auf ECU und Microcontroller (-> AUTOSAR)
  - Umsetzung von sicherheitsrelevanten und anderen Anforderungen (Funktional, Wartung, Wiederverwendung, ...):
    - Ein Entwicklungsprozess
  - Voraussetzung für die Implementierung
  - Beherrschung der Komplexität

6-7 Software architectural design

- Entwicklung einer SW-Architektur, die die Sicherheitsanforderungen an die SW umsetzt
- Verifikation des Entwurfs der SW-Architektur



Typisches Arbeitsergebnis von 6-7
Software architectural design specification

- 7.5 Work products
  7.5.1 Software architectural design specification resulting from requirements 7.4.1 to 7.4.6, 7.4.9, 7.4.10, 7.4.14, 7.4.15 and 7.4.17.
  Im Folgenden wird eine Auswahl betrachtet.
- 7.4 Requirements and recommendations
- 7.4.1 Beschreibungsmethoden für SW-Architekturen

Typisches Arbeitsergebnis von 6-7
Software architectural design specification

- 7.5 Work products
  7.5.1 Software architectural design specification resulting from requirements 7.4.1 to 7.4.6, 7.4.9, 7.4.10, 7.4.14, 7.4.15 and 7.4.17.
  Im Folgenden wird eine Auswahl betrachtet.
- 7.4 Requirements and recommendations
- 7.4.1 Beschreibungsmethoden für SW-Architekturen

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

Typisches Arbeitsergebnis von 6-7
Software architectural design specification

- 7.4.2 Folgendes soll beim Entwurf der SW-Architektur beachtet werden
  - Verifizierbarkeit der SW-Architektur, insbesondere bi-direktionale Rückverfolgbarkeit (traceability) zwischen SW-Architektur und Sicherheitsanforderungen an die SW.
  - Geeignet für konfigurierbare SW
  - Unterstützung bei Entwurf und Implementierung der SW-Einheiten / Elemente der SW-Architektur
  - Testbarkeit (SW Integrationstest)
  - Wartbarkeit
- 7.4.3 Zur Vermeidung von unnötiger Komplexität und daraus folgendem Fehlverhalten der SW soll die SW-Architektur die folgenden Eigenschaften haben:
  - Modularität
  - Kapselung
  - Einfachheit
- Dies wird durch Befolgung der Prinzipien ausTabelle 3 erreicht.

Typisches Arbeitsergebnis von 6-7
Software architectural design specification

Table 3 - Principles for software architectural design

Typisches Arbeitsergebnis von 6-7
Software architectural design specification

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	++	++	++	++
1c	Restricted size of interfaces	+	+	+	+
1d	High cohesion within each software component	+	++	++	++
1e	Restricted coupling between software components	+	++	++	++
1f	Appropriate scheduling properties	++	++	++	++
1g	Restricted use of interrupts	+	+	+	++

Typisches Arbeitsergebnis von 6-7
Software architectural design specification

- 7.4.4 Die SW-Architektur soll soweit verfeinert werden, dass alle SW-Elnheiten identifiziert sind.
- 7.4.5 Die SW-Architektur soll folgendes beschreiben
  - Statische Aspekte
    - SW-Struktur mit Hhierachie-Ebenen
    - Logische Reihenfolge der Datenverarbeitung
    - Datentypen
    - Externe Schnittstellen der SW-Einheiten untereinander
    - Externe Schnittstellen der SW zur Umgebung
  - Dynamische Aspekte
    - Funktionalität und Verhalten
    - Kontrollfluss und Nebenläufigkeit der Prozesse
    - Datenfluss zwischen den SW-Einheiten
    - Datenfluss nach aussen
    - Zeitanforderungen

Typisches Arbeitsergebnis von 6-7
Software architectural design specification

- 7.4.9 Die Sicherheitsanforderungen an die SW sollen den SW-Einheiten zugeordnet werden. Jede SW-Einheit soll gemäss dem höchsten ASIL der zugeordneten Sicherheitsanforderungen entwickelt werden.
- 7.4.14 Die folgenden Mechanismen zur Fehlerentdeckung sollen angewendet werden:

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

Typisches Arbeitsergebnis von 6-7
Software architectural design specification

- 7.4.9 Die Sicherheitsanforderungen an die SW sollen den SW-Einheiten zugeordnet werden. Jede SW-Einheit soll gemäss dem höchsten ASIL der zugeordneten Sicherheitsanforderungen entwickelt werden.
- 7.4.14 Die folgenden Mechanismen zur Fehlerentdeckung sollen angewendet werden:

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	+	+	+	++
1c	Detection of data errors	+	+	+	+
1d	External monitoring facility	0	+	+	++
1e	Control flow monitoring	0	+	++	++
1f	Diverse software design	0	0	+	++

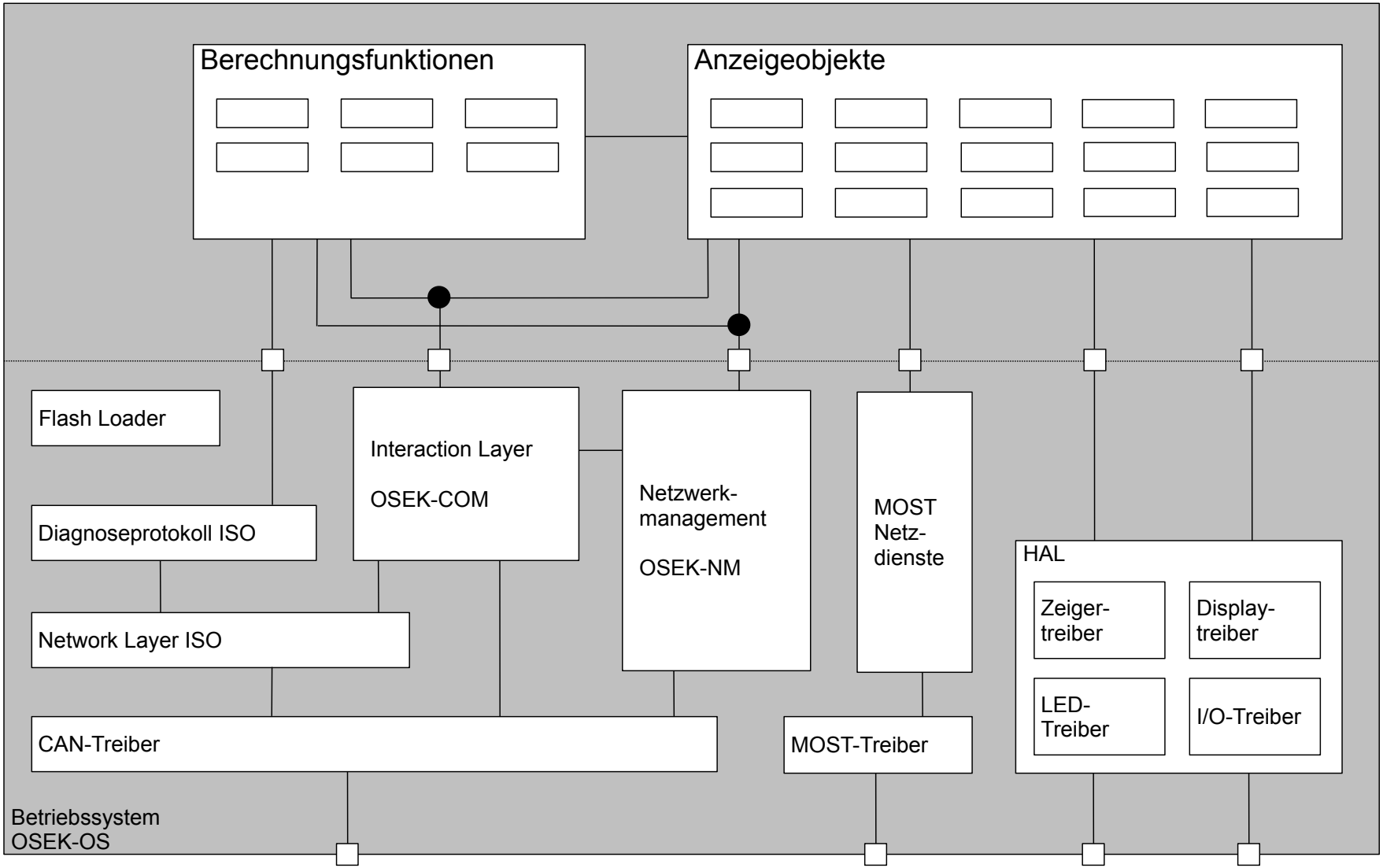
Typisches Arbeitsergebnis von 6-7
Software architectural design specification

- 7.4.17 Die von der SW benötigten Ressourcen sollen abgeschätzt werden (Obere Grenze):
  - Rechenzeit
  - Speicherbedarf und Speicherort (RAM, ROM)
  - Kommunikation (Datenrate)

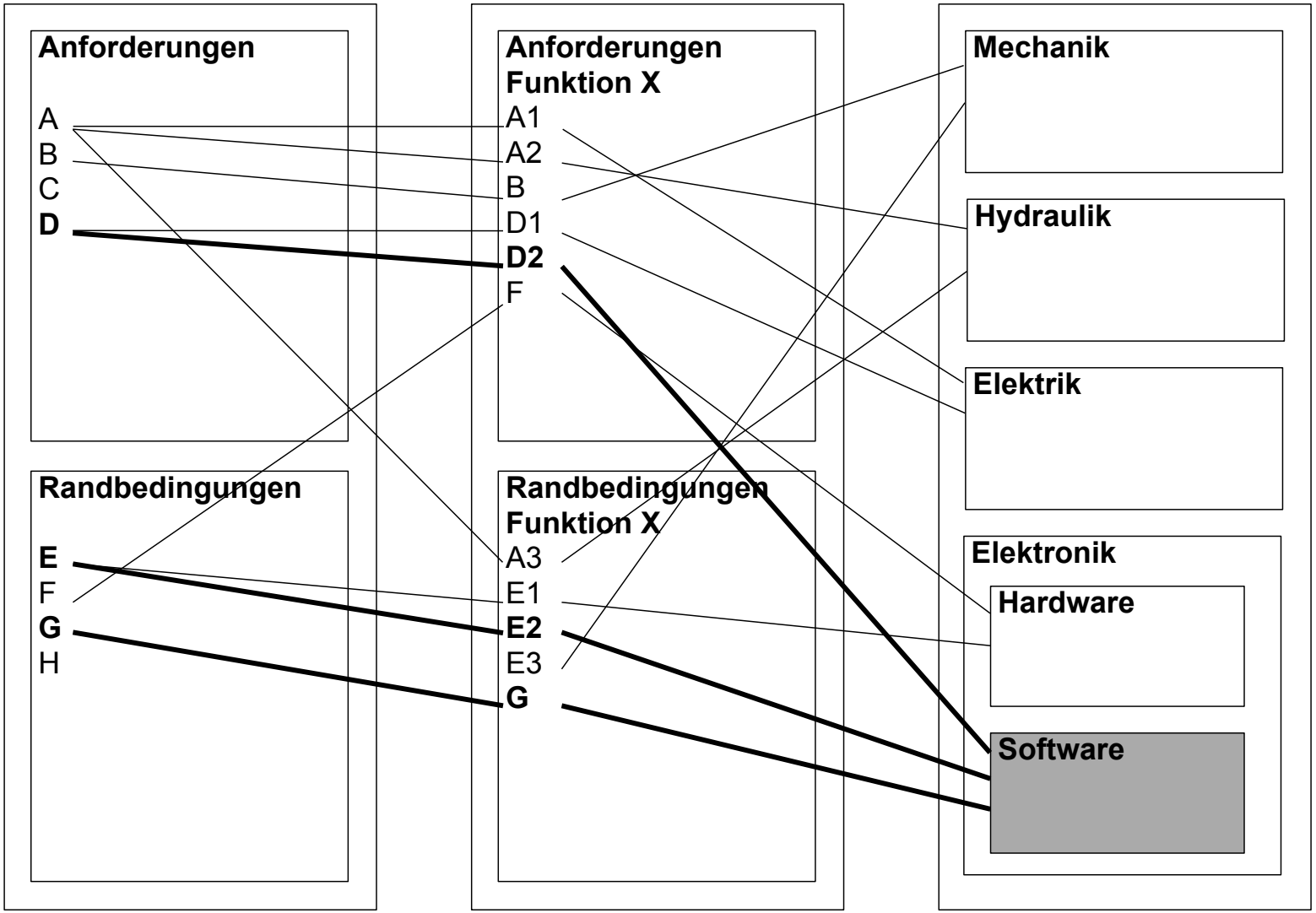
Beispiel Kombiinstrument

- Tachometer
- Odometer
- Drehzahlmesser
- Tankanzeige
- Kühlmitteltemperaturanzeige
- Kontrollleuchten
- ...

Softwarearchitektur des Kombiinstrumentes (Nach Schäuffele, Zurawka)

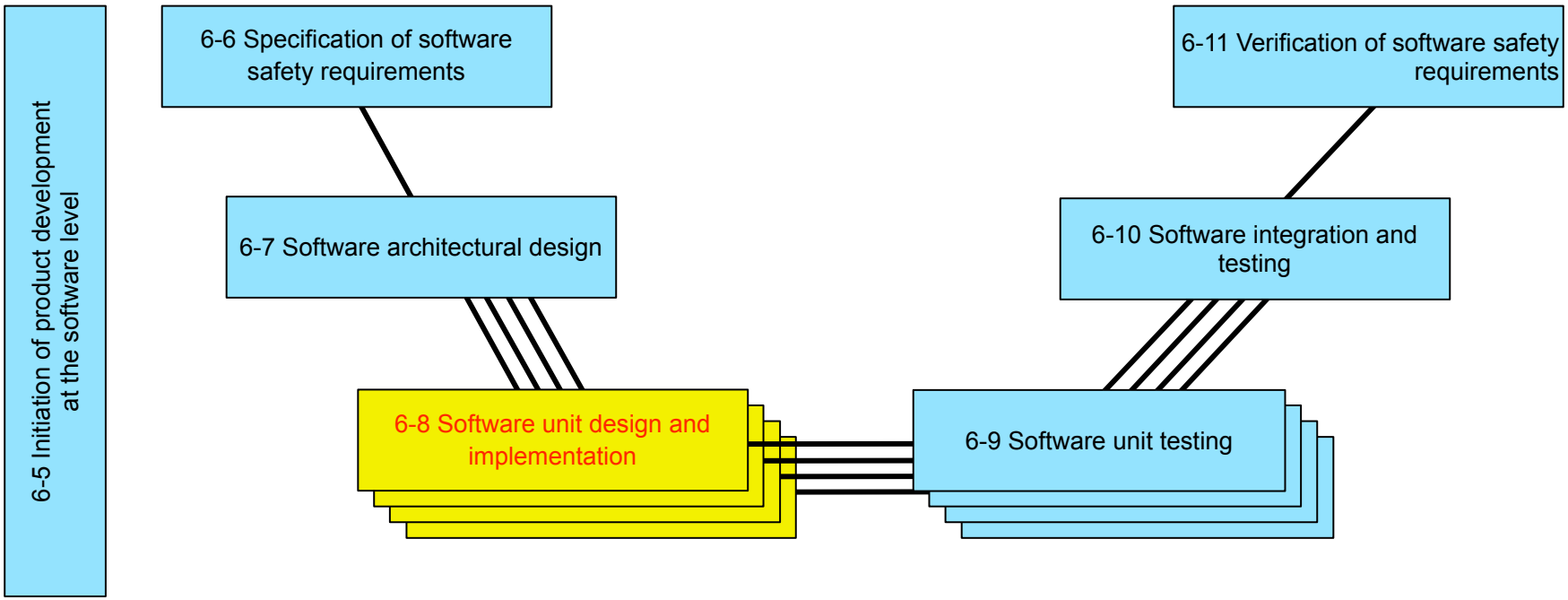


Rückverfolgbarkeit von Anforderungen (traceability)
Hier auf System-Ebene



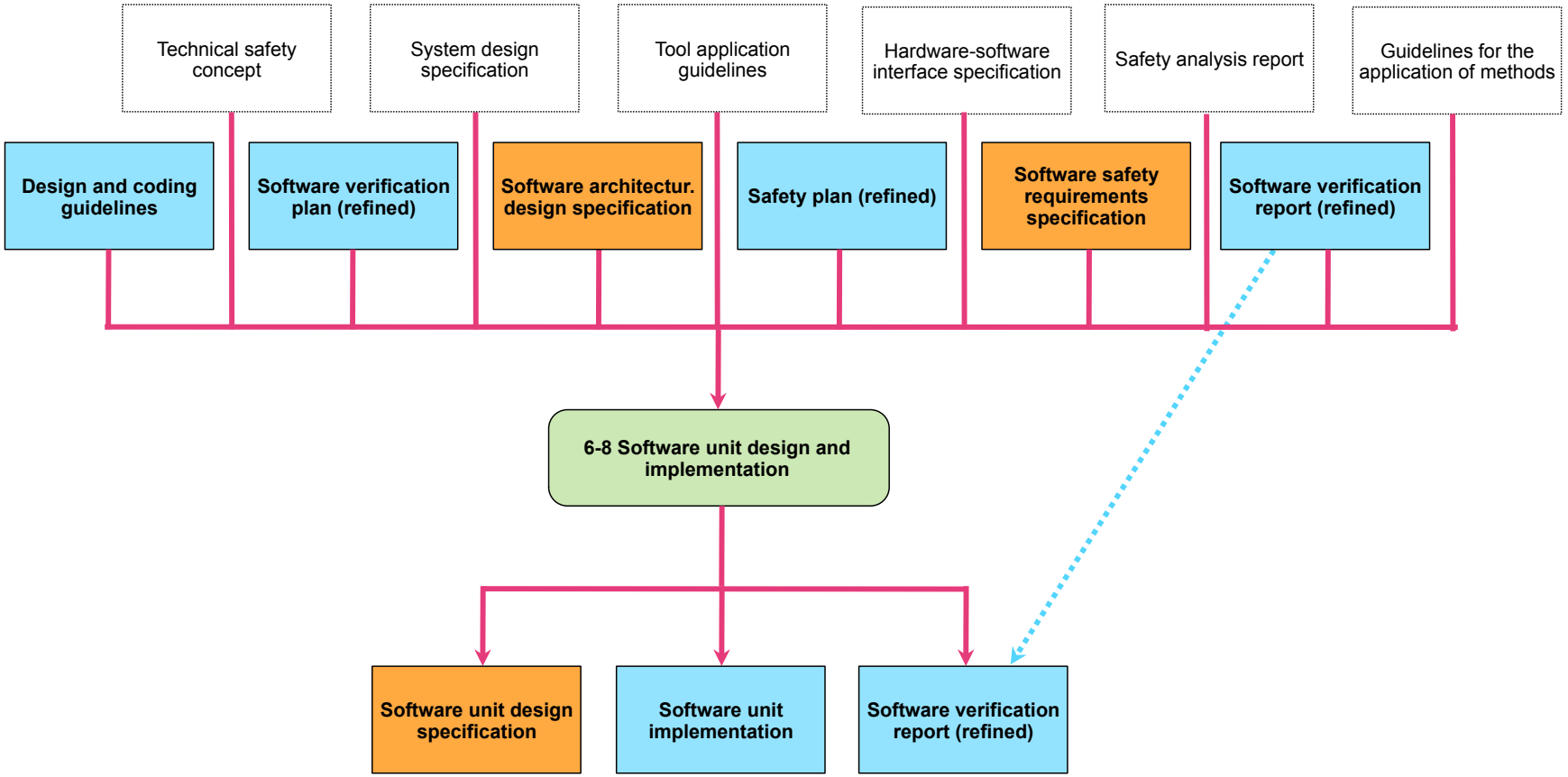
(Nach Schäuffele, Zurawka)

Part 6: Product development: software level
6-8 Software unit design and implementation



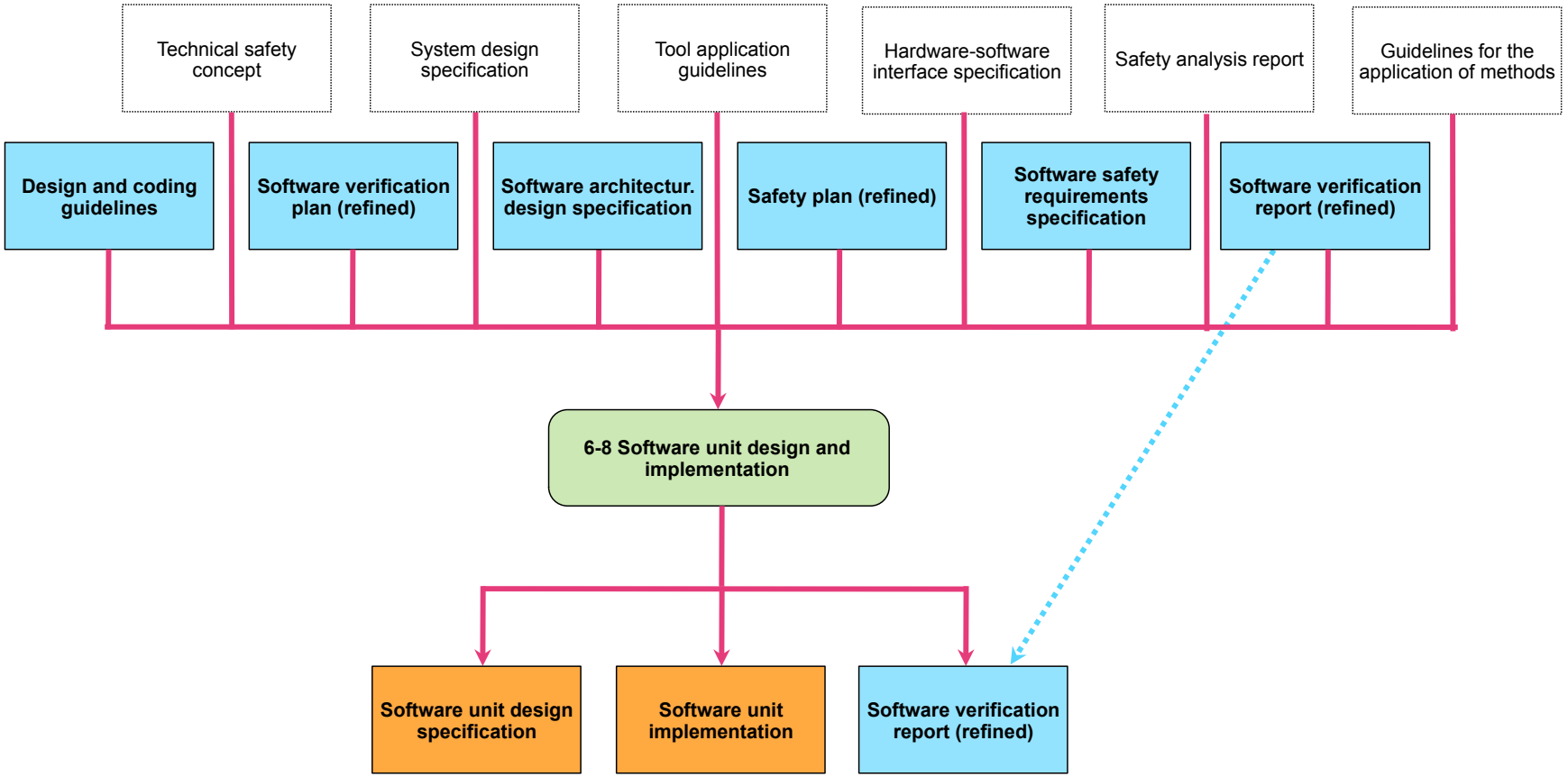
6-8 Software unit design and implementation

- Spezifikation der SW-Einheiten (SW-Module) in Übereinstimmung mit der SW-Architektur und den Sicherheitsanforderungen an die SW.



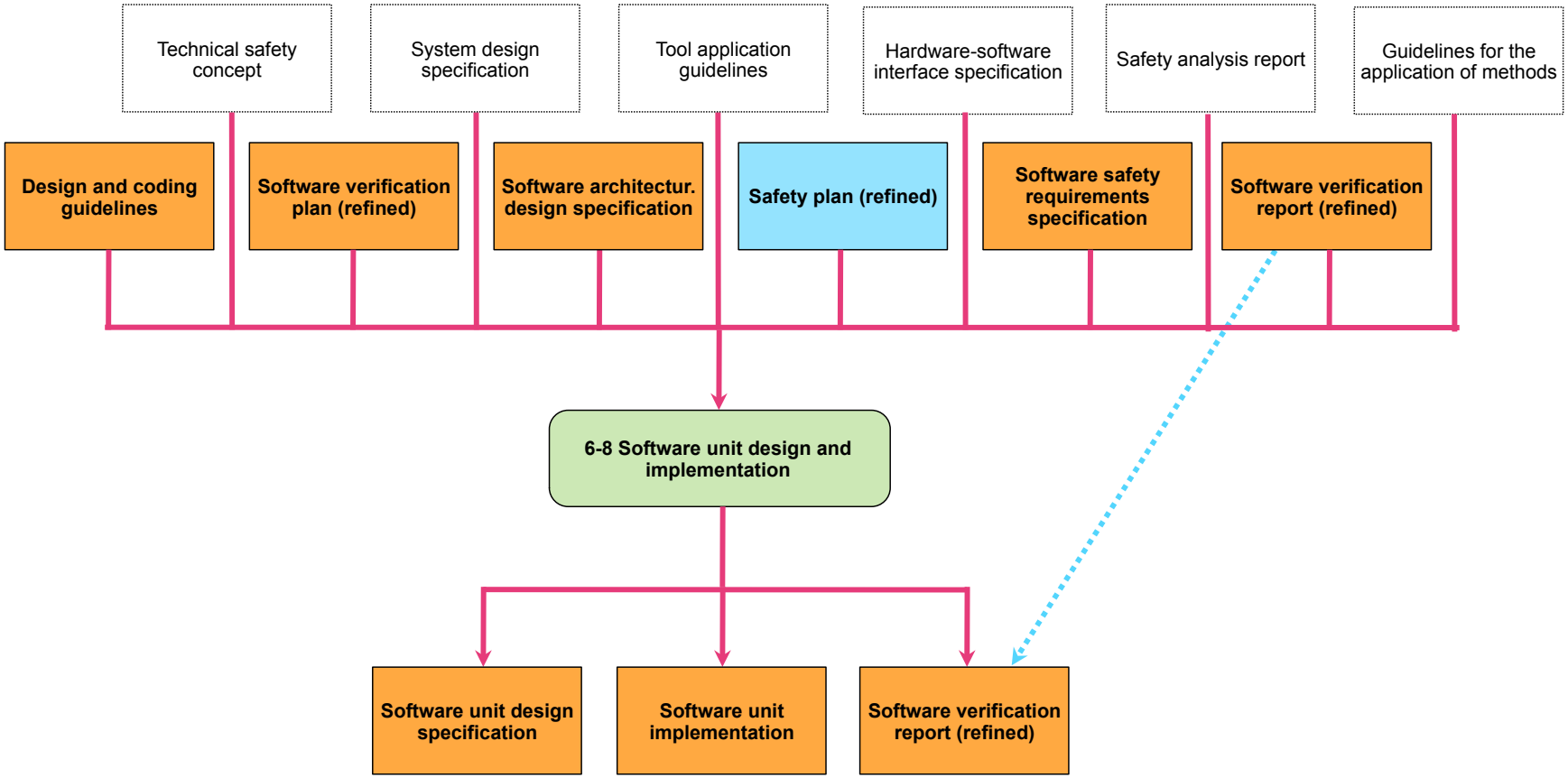
6-8 Software unit design and implementation

- Implementierung der SW-Einheiten gem. Spezifikation.



6-8 Software unit design and implementation

- Statische Verifikation der SW-Einheiten (Entwurf und Implementierung).



6-8 Software unit design and implementation

- Feinentwurf ausgehend von der SW-Architektur
- Implementierung als Modell (mit automatischer Code-Generierung)
  - Einhaltung von Modellierungs-Richtlinien
  - Prüfung auf Modell-Ebene
- Implementierung als Quellcode
  - Einhaltung von Codierungs-Richtlinien
  - Prüfung auf Code-Ebene
- Implementierung von sicherheitsrelevanten und anderen Anforderungen (Funktional, Wartung, ...):
  - Ein Entwicklungsprozess
- Die Implementierung beinhaltet die Erzeugung von Quellcode und die Übersetzung in Objektcode.

Typisches Arbeitsergebnis von 6-8
Software unit implementation

- 8.5 Work products
  8.5.2 Software unit implementation resulting from requirement 8.4.4.
- 8.4 Requirements and recommendations
- 8.4.4 Prinzipien für Entwurf und Implementierung auf Quellcode-Ebene
  Prinzipien gem. Tabelle 8 sollen angewendet werden um Folgendes zu erreichen:
  - Korrekte Ausführungsreihenfolge von Prozeduren und Funktionen gem. SW-Architektur
  - Schnittstellenkonsistenz
  - Korrektheit von Daten- und Kontrollfluss
  - Einfachheit
  - Lesbarkeit und Verständlichkeit
  - Robustheit
    Beispiel: Methoden zur Vermeidung von unplausiblen Werten, Laufzeitfehlern, Division durch 0, Fehler in Daten- und Kontriollfluss
  - Änderbarkeit
  - Testbarkeit
- Viele der Prinzipien nach Tabelle 8 werden durch MISRA-C abgedeckt

Typisches Arbeitsergebnis von 6-8
Software unit implementation

Table 8 — Design principles for software unit design and implementation

Typisches Arbeitsergebnis von 6-8
Software unit implementation

Table 8 — Design principles for software unit design and implementation

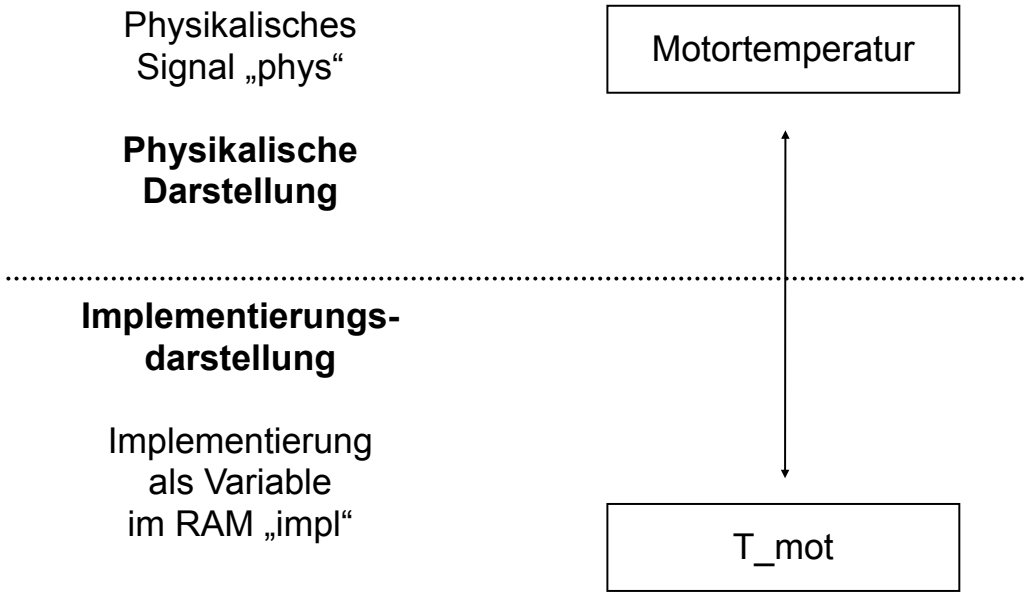
Methods		ASIL			
		A	B	C	D
1a	One entry and one exit point in subprograms and functions (a)	++	++	++	++
1b	No dynamic objects or variables, or else online test during their creation	+	++	++	++
1c	Initialisation of variables	++	++	++	++
1d	No multiple use of variable names	+	++	++	++
1e	Avoid global variables or else justify their usage	+	+	++	++
1f	Limited use of pointers	0	+	+	++
1g	No implicit type conversions (Nicht bei Assembler)	+	++	++	++
1h	No hidden data flow or control flow	+	++	++	++
1i	No unconditional jumps (“GOTO”, Nicht bei Assembler)	++	++	++	++
1j	No recursions	+	+	++	++

Beispiel: Implementierung der Motortemperatur

- Unterscheidung zwischen Variablen und durch das Programm nicht veränderbaren Parametern
- Design-Entscheidungen
  - Prozessorinterne Darstellung
  - Speichersegment für die Ablage
    - RAM = Random[-]access memory, Direktzugriffsspeicher:
      jede Speicherzelle kann über ihre Speicheradresse direkt angesprochen werden
    - ROM = Read-only memory; Festwertspeicher:
      ein Datenspeicher, der nur lesbar ist, im normalen Betrieb aber nicht beschrieben werden kann und nicht flüchtig ist.

Beispiel: Implementierung der Motortemperatur

- Beispiel Motortemperatur:
  Abbildung der physikalischen Spezifikation auf die Implementierung



Design und Implementierung der Software-Komponenten

- Beispiel Motortemperatur:
  Abbildung der physikalischen Spezifikation auf die Implementierung
- Physik
  - Bezeichnung im Klartext: Motortemperatur
  - Physikalische Einheit: °C
- Umrechnung
  - Umrechnungsformel: impl = f(phys) = 40 + 1 x phys
  - Quantisierung: 1 Bit = 1 °C
  - Offset: 40 °C
  - Minimal-/Maximalwert
    Physik: -40 °C - 215 °C
    Implementierung: 0 - 255
- Imlementierung
  - Bezeichnung im Code: T_mot
  - Wortlänge: 8 Bit
  - Speichersegment: Internes RAM

Design und Implementierung der Software-Komponenten

- Beispiel Motortemperatur:
  Abbildung der physikalischen Spezifikation auf die Implementierung
- Physik
  - Bezeichnung im Klartext: Motortemperatur
  - Physikalische Einheit: °C
- Umrechnung
  - Umrechnungsformel: impl = f(phys) = 40 + 1 x phys
  - Quantisierung: 1 Bit = 1 °C
  - Offset: 40 °C
  - Minimal-/Maximalwert
    Physik: -40 °C - 215 °C
    Implementierung: 0 - 255
- Imlementierung
  - Bezeichnung im Code: T_mot
  - Wortlänge: 8 Bit
  - Speichersegment: Internes RAM

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

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

Methods		ASIL			
		A	B	C	D
1a	Walk-through (a)	++	+	0	0
1b	Inspection (a)	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	0	0	+	+
1e	Control flow analysis (b), (c)	+	+	++	++
1f	Data flow analysis (b), (c)	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis (d)	+	+	+	^

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

Methods		ASIL			
		A	B	C	D
1a	Walk-through (a)	++	+	0	0
1b	Inspection (a)	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	0	0	+	+
1e	Control flow analysis (b), (c)	+	+	++	++
1f	Data flow analysis (b), (c)	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis (d)	+	+	+	^

Überspringen?

Motivation

- Tests können die Anwesenheit von Fehlern zeigen, nie aber deren Abwesenheit.

  Edsger W. Dijkstra

- Ein fehlender Programmzweig ... wird auch durch Austesten aller vorhandenen Programmzweige nicht gefunden.

  Bernhard Hohlfeld

- Software wird hauptsächlich getestet
- Testverfahren können nie alle Systemzustände erreichen
  - Wurde ausreichend getestet?
- Software muss für Tests vorhanden und ausführbar sein
  - Ist die Spezifikation korrekt?
- Interne Qualitätseigenschaften werden selten geprüft
- Ist die Software zuverlässig, wartbar, änderbar etc. ?
- Ist die Software effizient?
- Wie viel Speicher/Zeit verbraucht das System maximal?

- Quelle: Der Beitrag über statische Codeanalysen basiert auf einem Vortrag von Dr. Steffen Görzig

Motivation

- Anforderungen an Software Qualität steigen
- ISO 26262 (Funktionale Sicherheit)
  - Funktionalität
  - Zuverlässigkeit
  - Benutzbarkeit
  - Effizienz
  - Änderbarkeit
  - Übertragbarkeit
- Zulieferergeschäft erfordert objektive Qualitätsaussagen
- Qualitätsvorgaben erfordern Qualitätsmaße
- Keine Fehler gefunden
  - Schlecht getestet?
  - Keine Fehler vorhanden?

Statische Analysen

- Grundprinzip: Analyse von Artefakten ohne deren Ausführung
- Manuell
  - Review, Inspection, Walkthrough, etc.
  - Vorteil
    - Artefakte müssen nicht formal spezifiziert sein
  - Nachteil
    - Hoher Aufwand, menschliche Fehler
- Automatisch / Werkzeuggestützt
  - Modelltransformation, Datenflussanalyse, Symbolische Ausführung
  - Vorteil
    - Formaler Nachweis möglich, objektive Aussagen, automatisierbar
  - Nachteil
    - Teilweise hoher Aufwand
    - Unterschiedliche Werkzeuge für unterschiedliche Einsatzgebiete

Statische Analysen

- Pro
- Analyseobjekt muss nicht formal sein
  - Qualitätssicherung im gesamten Entwicklungszyklus
- Analyseobjekt muss nicht ausführbar sein
  - Testfälle nicht notwendig
- Komplette Pfadabdeckung möglich (inkl. Varianten)
  - Formaler Nachweis

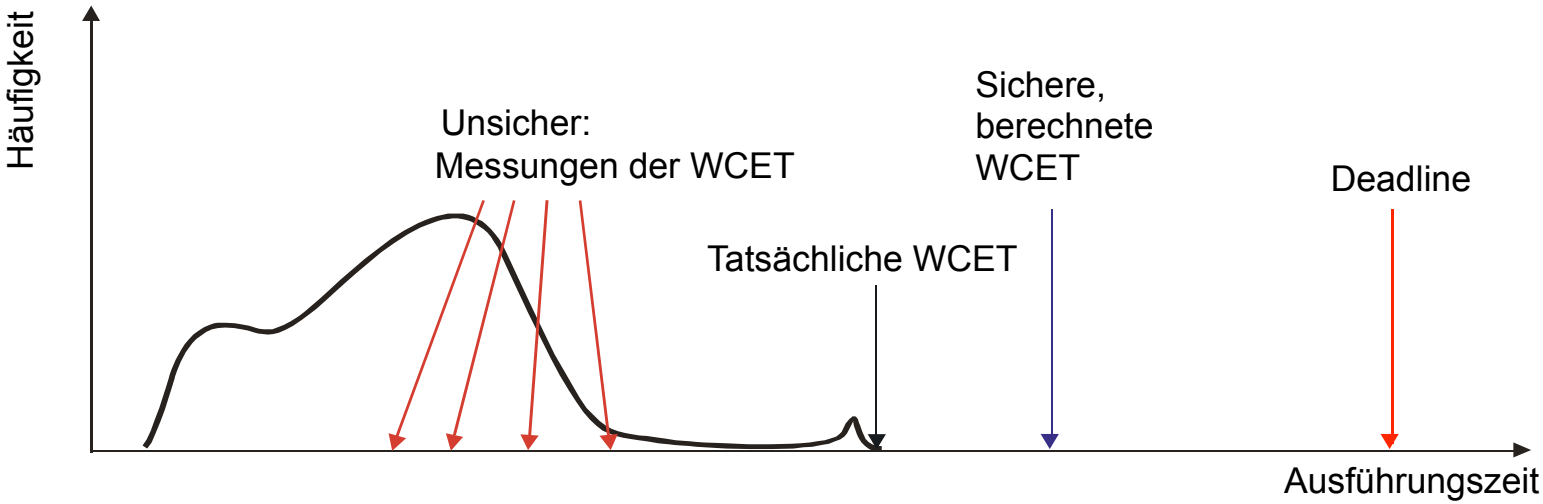
- Contra
- Prüfung funktionaler Eigenschaften nur schwer möglich
- Teilweise hoher manueller Aufwand
- Systemtests weiter notwendig
  - Hardwareanteile
  - Sensorrauschen
  - ...

Statische Analysewerkzeuge - Kategorisierung

- Analyseziel
  - Fehlerfindung (Laufzeitfehler, Verletzung von Regeln und Guidelines)
  - Metriken (Komplexität, Wiederverwendbarkeit)
  - Codeverständnis (Aufrufhierachien, Typbeziehungen)
  - Ressourcenverbrauch (Zeit, Speicher)
- Analyseobjekt
  - Modell (Simulink, Stateflow)
  - Quellcode (C, C++, Java)
  - Maschinencode (obj, asm)
- Analysetiefe
  - Modelltransformation (Kontrollflussgraph, Aufrufgraph)
  - Datenflussanalyse (intraprozedural, interprozedural, Abstrakte Interpretation)

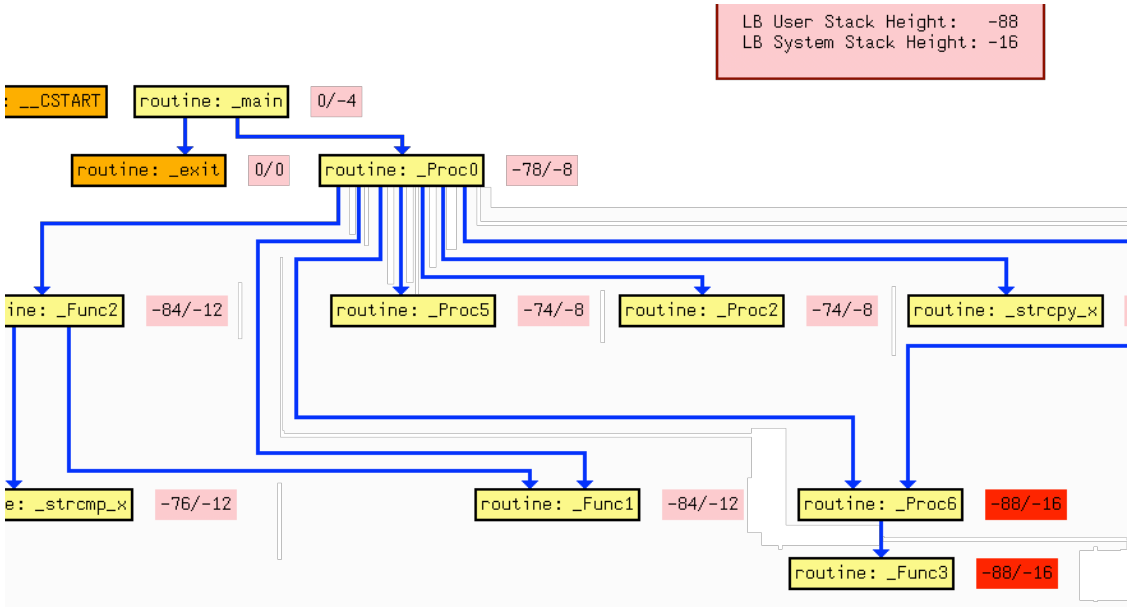
Werkzeug	Analyseziel
aiT	Ressourcenverbrauch (Zeit)
Stackanalyzer	Ressourcenverbrauch (Stackspeicher)
PolySpace Verifier	Fehler (Laufzeitfehler)
C/C++ Analysator	Codeverständnis
Sotograph	Fehler (Architektur, Design)
QA-C/C++	Metriken, Fehler

Statische Analysewerkzeuge
aiT



Analyseziel:	Verifikation der Worst-Case Execution Time (WCET)
Analyseobjekt:	Maschinencode
Analysetiefe:	Abstrakte Interpretation

Statische Analysewerkzeuge
Stackanalyzer



Analyseziel: Verifikation des Stackspeicherbedarfs

Analyseobjekt: Maschinencode

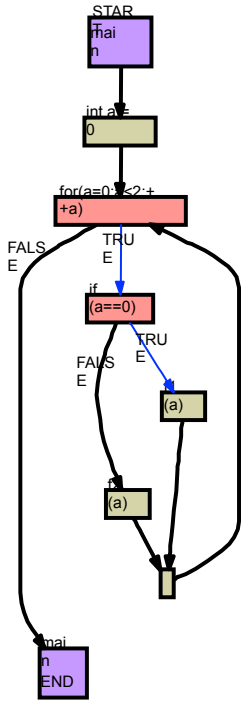
Analysetiefe: Interprozedurale Datenflussanalyse

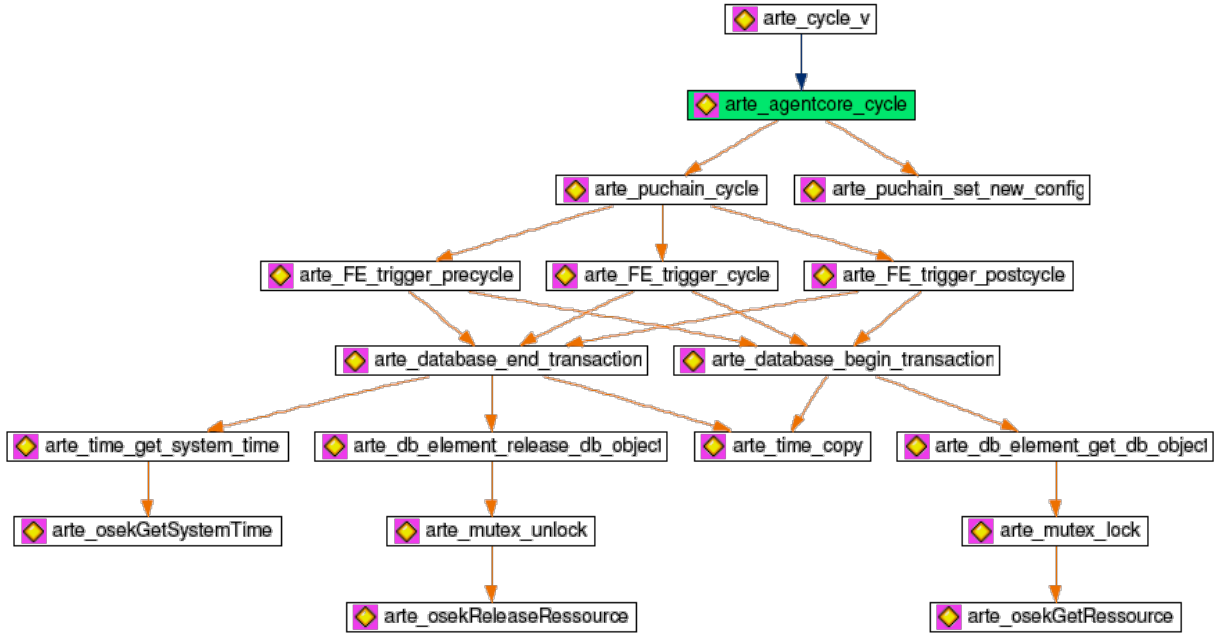
Statische Analysewerkzeuge
PolySpace Verifier

```
6     void functionl(int inputl)
7     {
8
9             char * p = ch;
10            int i;
11
12            // initilize:
13            tmpl = ((*p+*(p+1)) >> 1) + ((*(p+2)+*(p+3)) >> 1);
14
15            for (i = 0; i <= 6; i++)
16            {
17                    ch[i] = inputl&0xFF;
18                    inputl >>= 8;
19            }
20
21            if (inputl < -128 || inputl >= 0) return;
22
23            assert(tmpl<0);
24    }
```

Analyseziel:	Verifikation von Laufzeitfehlern
Analyseobjekt:	C, C++ Quellcode
Analysetiefe:	Abstrakte Interpretation

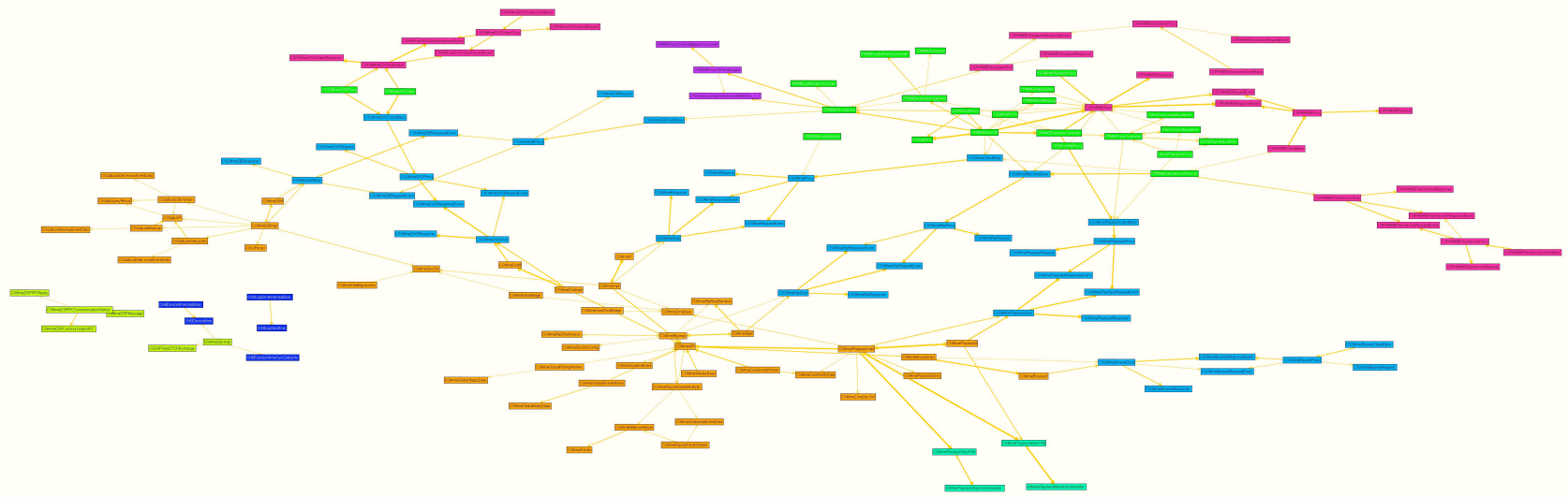
Statische Analysewerkzeuge
C/C++ Analysator





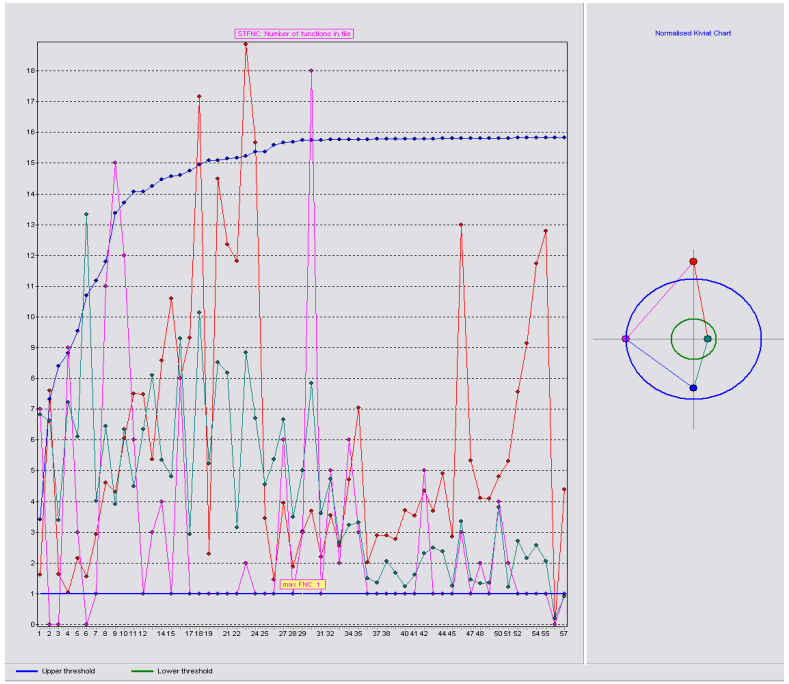
Analyseziel:	Analyse benutzerdefinierter Eigenschaften
Analyseobjekt:	C/C++ Quellcode
Analysetiefe:	Modelltransformation

Statische Analysewerkzeuge
Sotograph



Analyseziel:	Design und Architekturanalyse
Analyseobjekt:	C/C++/Java Quellcode
Analysetiefe:	Modelltransformation

Statische Analysewerkzeuge
QA-C/C++



Analyseziel:	Metrik- und Fehlerprüfung
Analyseobjekt:	C/C++ Quellcode
Analysetiefe:	Interprozedurale Datenflussanalyse

Informationen zu den Werkzeugen

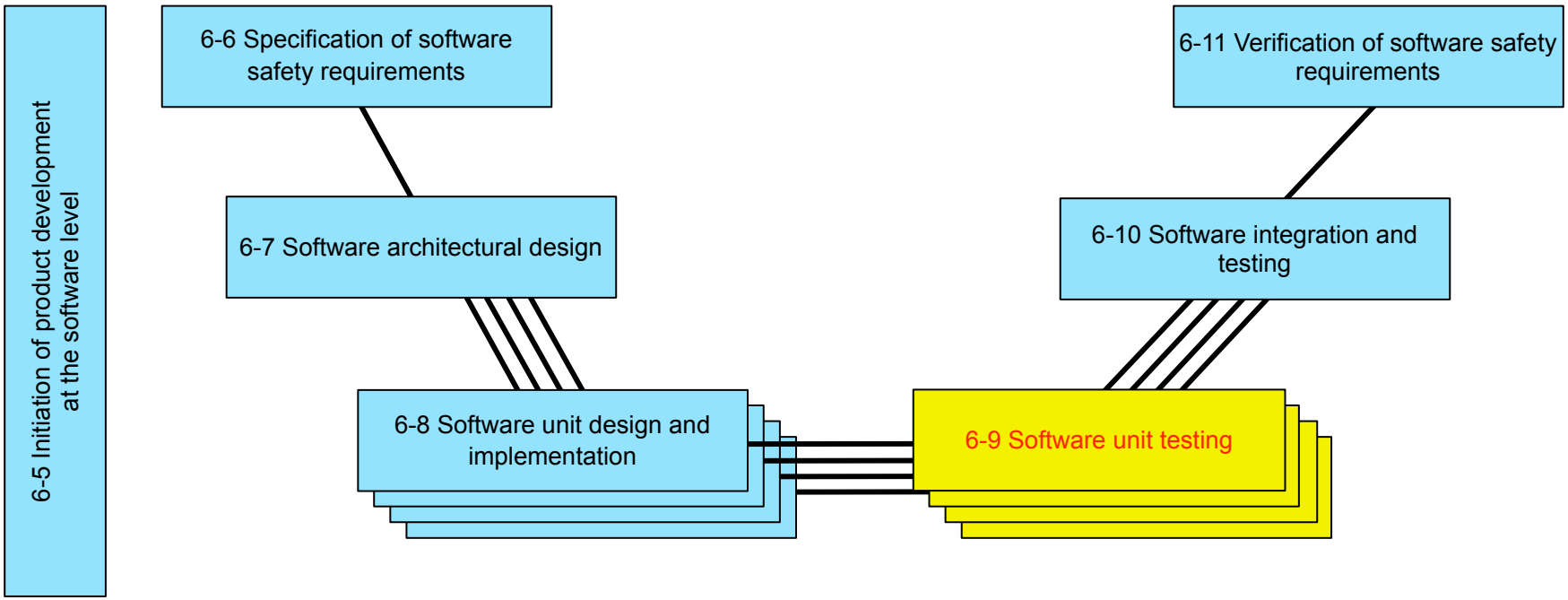
- aiT
  - http://www.absint.com/ait/
- Stackanalyzer
  - http://www.absint.com/stackanalyzer/
- PolySpace Verifier
  - www.mathworks.com/polyspace/
- C/C++ Analysator
  - Interner Einsatz bei Daimler AG (und bei EADS in der Ada-Variante)
- Sotograph
  - http://www.hello2morrow.com/products/sotograph
- QA-C/C++
  - http://www.programmingresearch.com/QAC_MAIN.html
  - http://www.programmingresearch.com/QACPP_MAIN.html
  - http://www.vectorcast.com/industries/automotive.php

Weitere Informationen (Stand Oktober 2008)

- Qualitätssicherung – Die Hüter der Software
  Fehlersuche und Qualitätssicherung im Softwarelabor der Daimler-Forschung
  Daimler HighTechReport 1/2008
  (pdf über http://www.daimler.com / Technologie & Innovation)
  Populärwissenschaftlich
- Sicherheit: Sicherheitsrelevante Software fehlerfrei und effizient entwickeln
  Daimler HighTechReport 2/2007
  (pdf über http://www.daimler.com / Technologie & Innovation)
  Populärwissenschaftlich

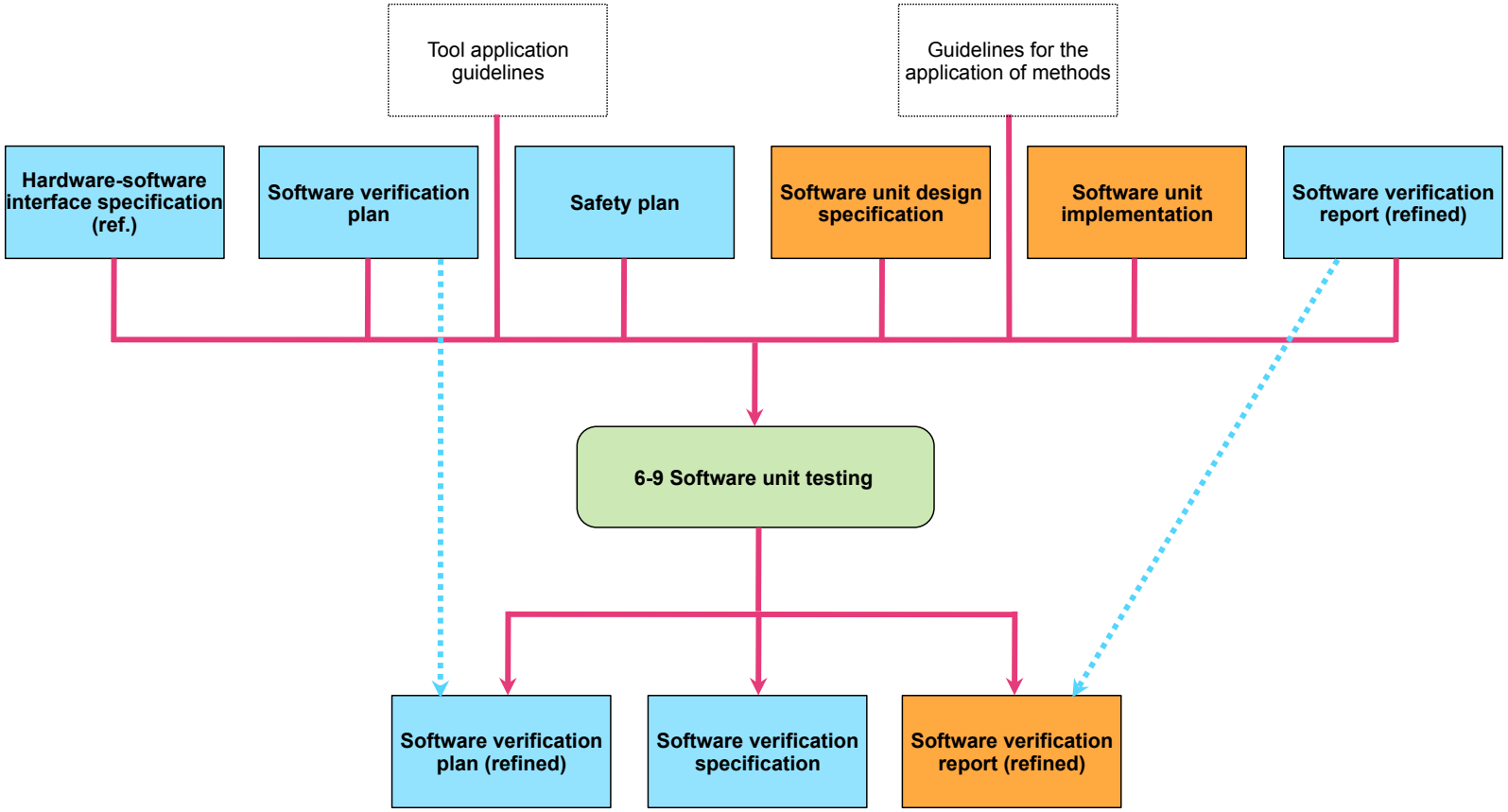
- ES_PASS (ITEA 2 06042)
  Improving safety-critical engineering processes
  Integrating static-analysis techniques into quality assurance processes for the transport industries
  http://www.itea2.org/public/project_leaflets/ES_PASS_profile_oct-07.pdf

Part 6: Product development: software level
6-9 Software unit testing



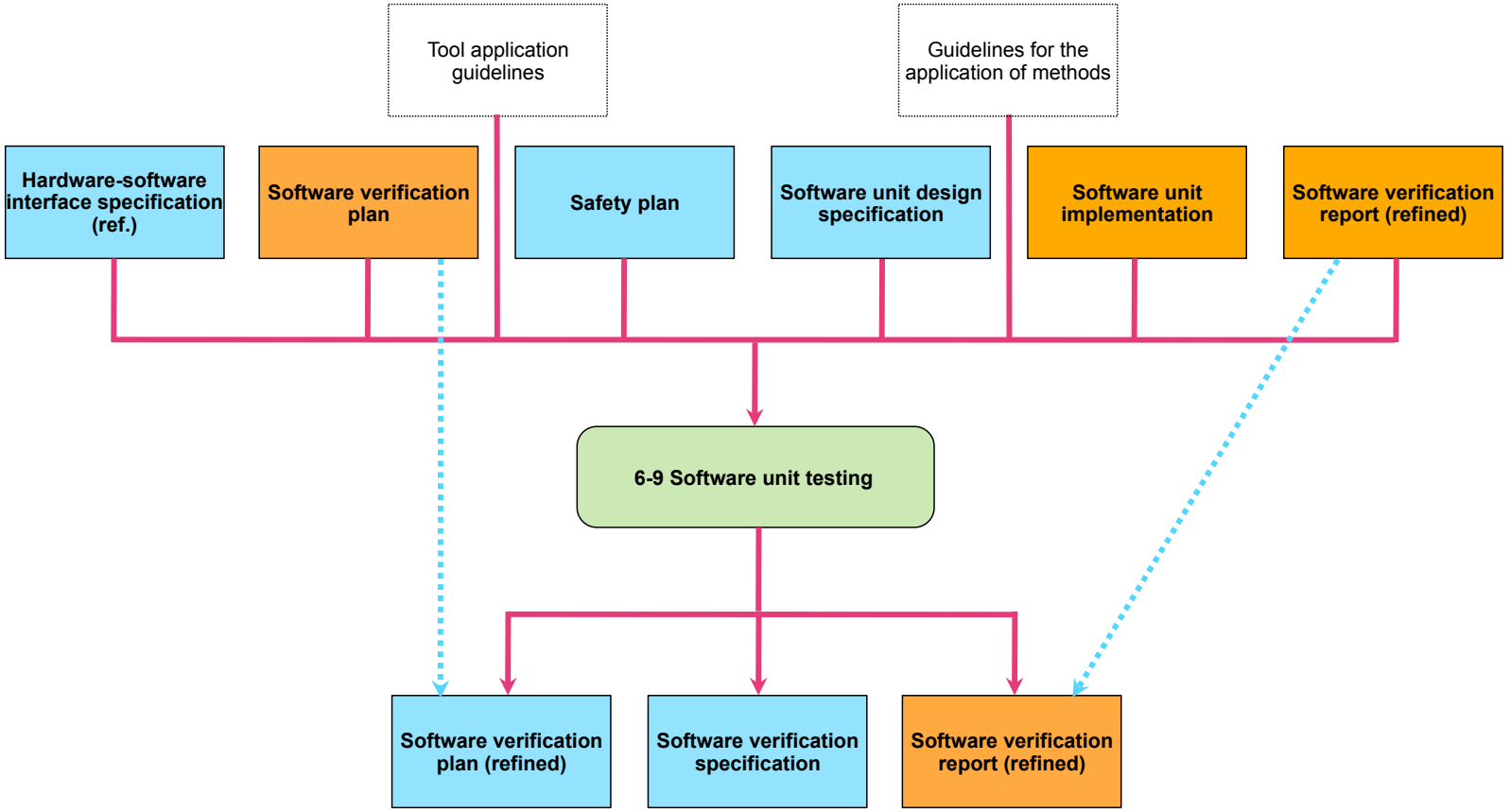
6-9 Software unit testing

- Nachweis der Erfüllung der Anforderungen durch die Implementierung auf Ebene SW-Einheit.
- Keine nicht gewünschte Funktionalität.



6-9 Software unit testing

- Testverfahren werden festgelegt und die Tests entsprechend durchgeführt.



Typisches Arbeitsergebnis von 6-9
Software verification plan (refined)

- 9.5 Work products
- 9.5.1 Software verification plan (refined) resulting from requirements 9.4.2 to 9.4.6.
  Im Folgenden wird eine Auswahl betrachtet.
- 9.4 Requirements and recommendations
- 9.4.2 Verweis auf den Unterstützungsprozess Verifikation
  Der Softwaretest soll in Übereinstimmung mit ISO 26262-8 (Supporting Processes):—, Clause 9 (Verification) geplant, spezifiziert und durchgeführt werden.

Typisches Arbeitsergebnis von 6-9
Software verification plan (refined)

- 9.4.3 Testmethoden
  Die in Tabelle 10 aufgeführten Tesrtmethoden sollen angewendet werden um Folgendes nachzuweisen:
  - Erfüllung der Spezifikation der Software
  - Einhaltung der Hardware-Software-Schnittstelle
  - Erfüllung der spezifizierten Funktionlität
  - Keine nicht beabsichtigte Funktionalität
  - Robustheit
    - Beispiele:
      - Keine nicht erreichbaren Programmteile (eher statische Analysen),
      - Massnahmenn zur Laufzeitfehlerentdeckung und – behandlung.
  - Genügende Ressourcen (Speicher, Rechenzeit)

Typisches Arbeitsergebnis von 6-9
Software verification plan (refined)

Table 10 — Methods for software unit testing

Typisches Arbeitsergebnis von 6-9
Software verification plan (refined)

Table 10 — Methods for software unit testing

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

Typisches Arbeitsergebnis von 6-9
Software verification plan (refined)

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

(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.

Typisches Arbeitsergebnis von 6-9
Software verification plan (refined)

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

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.

Typisches Arbeitsergebnis von 6-9
Software verification plan (refined)

Table 12 — Structural coverage metrics at the software unit level

Typisches Arbeitsergebnis von 6-9
Software verification plan (refined)

Table 12 — Structural coverage metrics at the software unit level

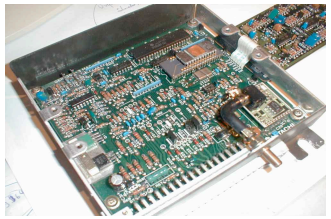
Methods		ASIL			
		A	B	C	D
1a	Statement coverage	++	++	+	+
1b	Branch coverage	+	++	++	++
1c	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++

Typisches Arbeitsergebnis von 6-9
Software verification plan (refined)

- 9.4.6 Testumgebungen
  Die Testumgebung soll der Zielumgebung soweit wie möglich entsprechen.
- Beispiele für Testumgebungen
  - Model-in-the-loop tests
    Beispiel: Ausführbare ML/SL-Modell
  - Software-in-the-loop tests
    Beispiel: Test des C-Quellcodes in PC-Umgebung
  - Processor-in-the-loop tests
    Quellcode auf Zielprozessor
  - Hardware-in-the-loop tests
    Quellcode auf Ziel-Steuergerät (= Zielprozessor(en) + Peripherie, insb. A/D- und D/A-Wandler)
- Siehe auch
  4 Requirements for compliance “Testumgebungen für integrierte Software“

Begriffsdefinitionen

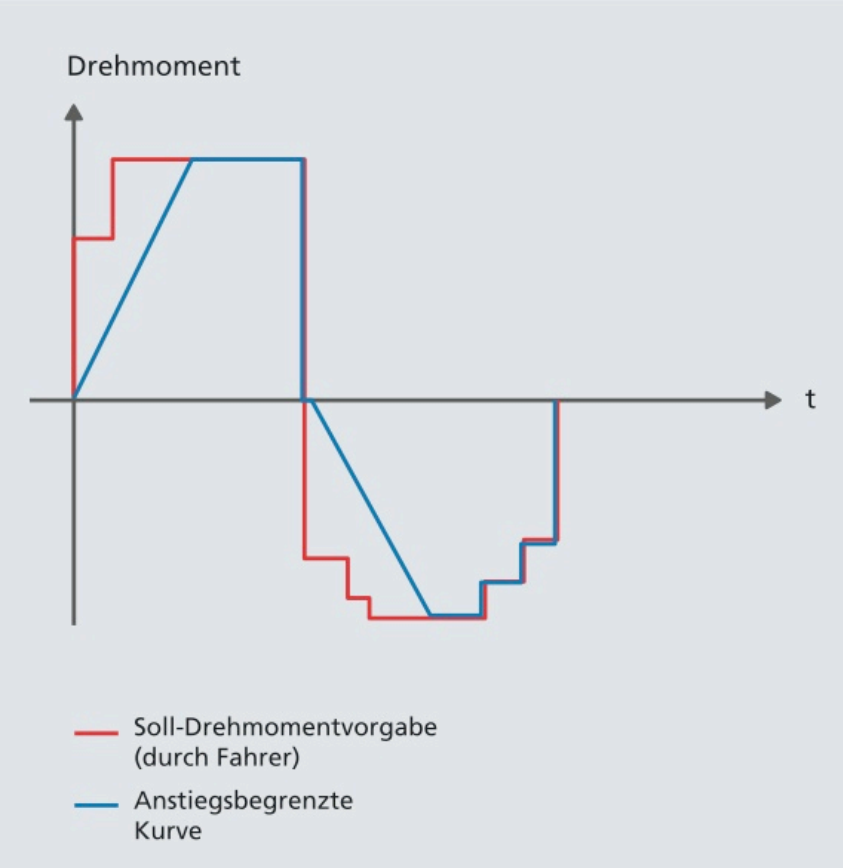
- MIL Model in the Loop, SIL Software in the Loop
  - Modell / SG-Software läuft auf simuliertem SG, das von simulierter Fahrzeugumgebung (rechnergenerierten Sensor- und Bussignalen) gespeist wird.
- HIL Hardware in the Loop
  - Reale SG-Hardware wird von simulierter Fahrzeugumgebung gespeist
- Prototyp
  - Simuliertes Steuergerät im Fahrzeug (“PC im Kofferraum”)
- Prüfstand, Fahrversuch
  - Steuergerät im Fahrzeug unter Laborbedingungen bzw. Auf der Strasse (Testgelände, öffentliches Strassennetz)



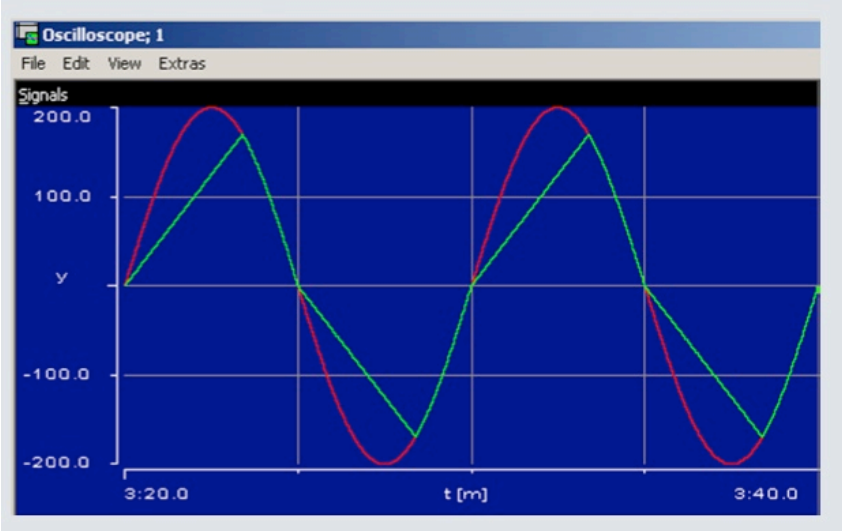
	Steuergerät	
Umgebung, Fahrzeug	simuliert	real
simuliert	MIL, SIL	HIL
real	Prototyp	Prüfstand, Fahrversuch

Offline-Simulation

Spezifikation: Begrenzung des Drehmomentanstiegs



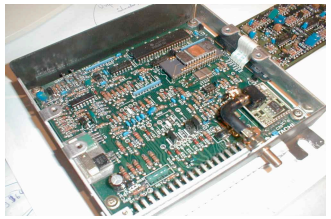
Modellierung des Sollwertes („Gas geben“) durch Sinuskurve



Quelle: Prof. Dr. Dieter Nazareth, FH Landshut
Entwicklung einer Antriebssteuerung für ein Hybridfahrzeug in einer Rapid Prototyping-Umgebung

Begriffsdefinitionen

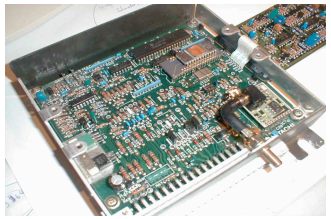
- MIL Model in the Loop, SIL Software in the Loop
  - Modell / SG-Software läuft auf simuliertem SG, das von simulierter Fahrzeugumgebung (rechnergenerierten Sensor- und Bussignalen) gespeist wird.
- HIL Hardware in the Loop
  - Reale SG-Hardware wird von simulierter Fahrzeugumgebung gespeist
- Prototyp
  - Simuliertes Steuergerät im Fahrzeug (“PC im Kofferraum”)
- Prüfstand, Fahrversuch
  - Steuergerät im Fahrzeug unter Laborbedingungen bzw. Auf der Strasse (Testgelände, öffentliches Strassennetz)



	Steuergerät	
Umgebung, Fahrzeug	simuliert	real
simuliert	MIL, SIL	HIL
real	Prototyp	Prüfstand, Fahrversuch

Begriffsdefinitionen

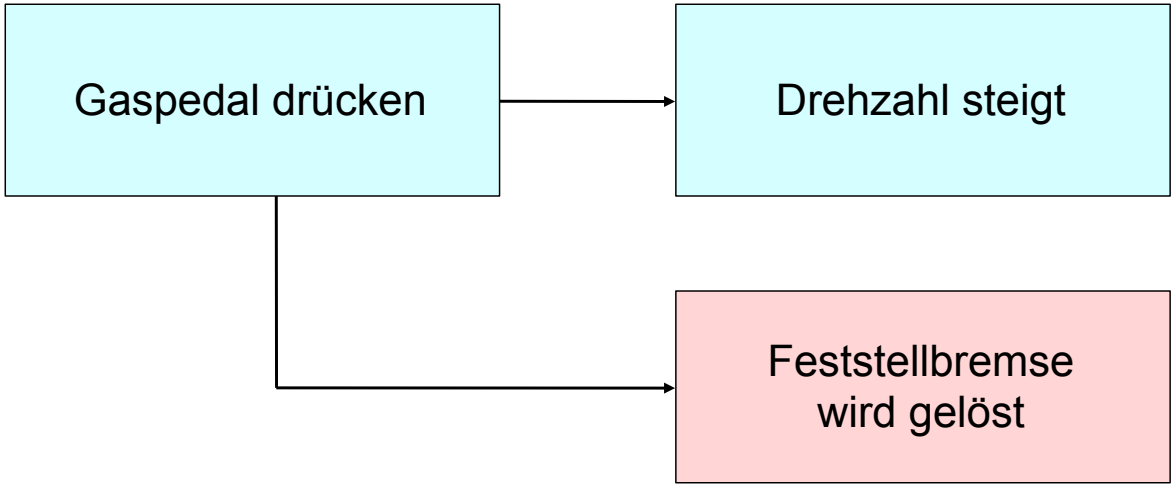
- MIL Model in the Loop, SIL Software in the Loop
  - Modell / SG-Software läuft auf simuliertem SG, das von simulierter Fahrzeugumgebung (rechnergenerierten Sensor- und Bussignalen) gespeist wird.
- HIL Hardware in the Loop
  - Reale SG-Hardware wird von simulierter Fahrzeugumgebung gespeist
- Prototyp
  - Simuliertes Steuergerät im Fahrzeug (“PC im Kofferraum”)
- Prüfstand, Fahrversuch
  - Steuergerät im Fahrzeug unter Laborbedingungen bzw. Auf der Strasse (Testgelände, öffentliches Strassennetz)



	Steuergerät	
Umgebung, Fahrzeug	simuliert	real
simuliert	MIL, SIL	HIL
real	Prototyp	Prüfstand, Fahrversuch

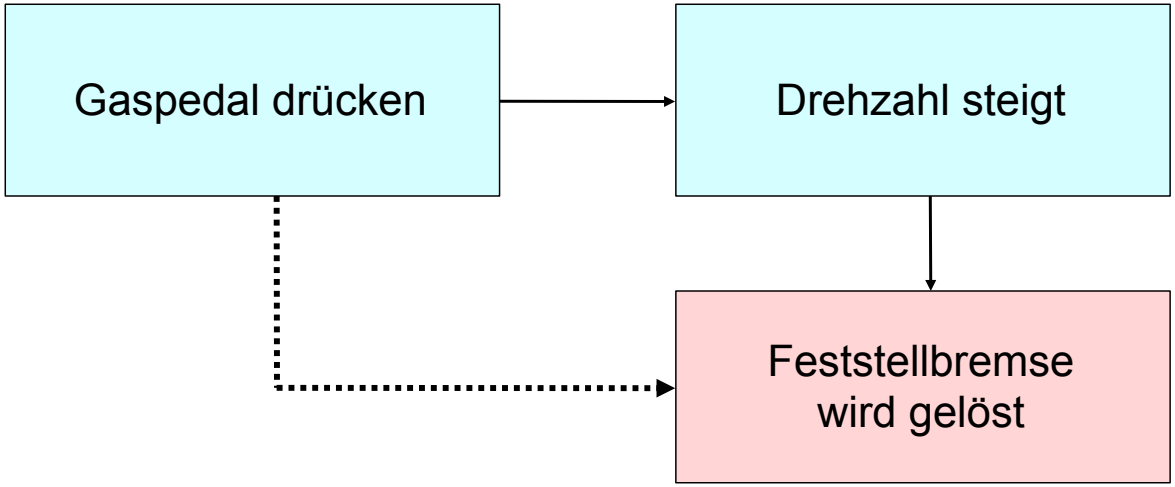
Keine nicht beabsichtigte Funktionalität
Beispiel: Lösen der elektrischen Festellbremse

- Feststellbremse: “Handbremse”
- Betriebsbremse: “Fussbremse”
- Arretieren der Feststellbremse durch Durchtreten der Betriebsbremse im Stand
- Lösen der Feststellbremse durch Gas geben: Angedachte Lösung



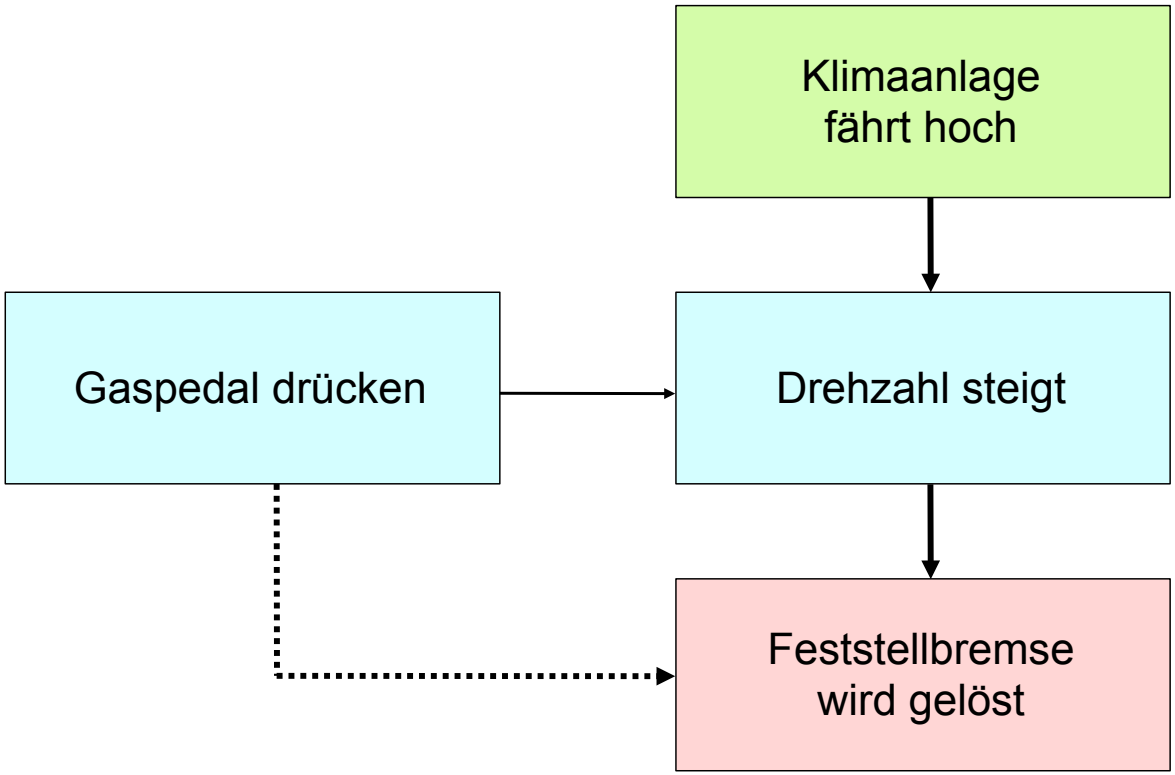
Keine nicht beabsichtigte Funktionalität
Beispiel: Lösen der elektrischen Festellbremse

- Realisierte Lösung
- Kostenziel
- Gewichtssziel
- Drehzahl liegt schon auf CAN, z.B. für Anzeige im Kombiinstrument

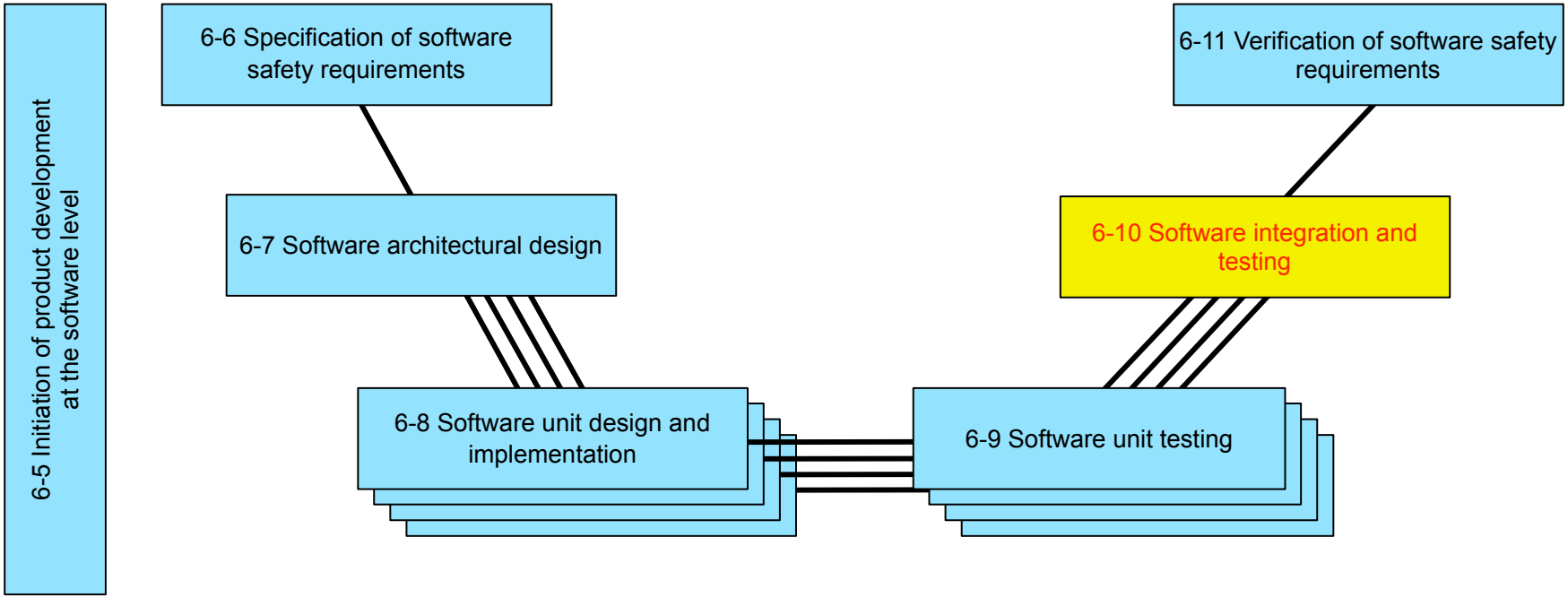


Keine nicht beabsichtigte Funktionalität
Beispiel: Lösen der elektrischen Festellbremse

- Auswirkung

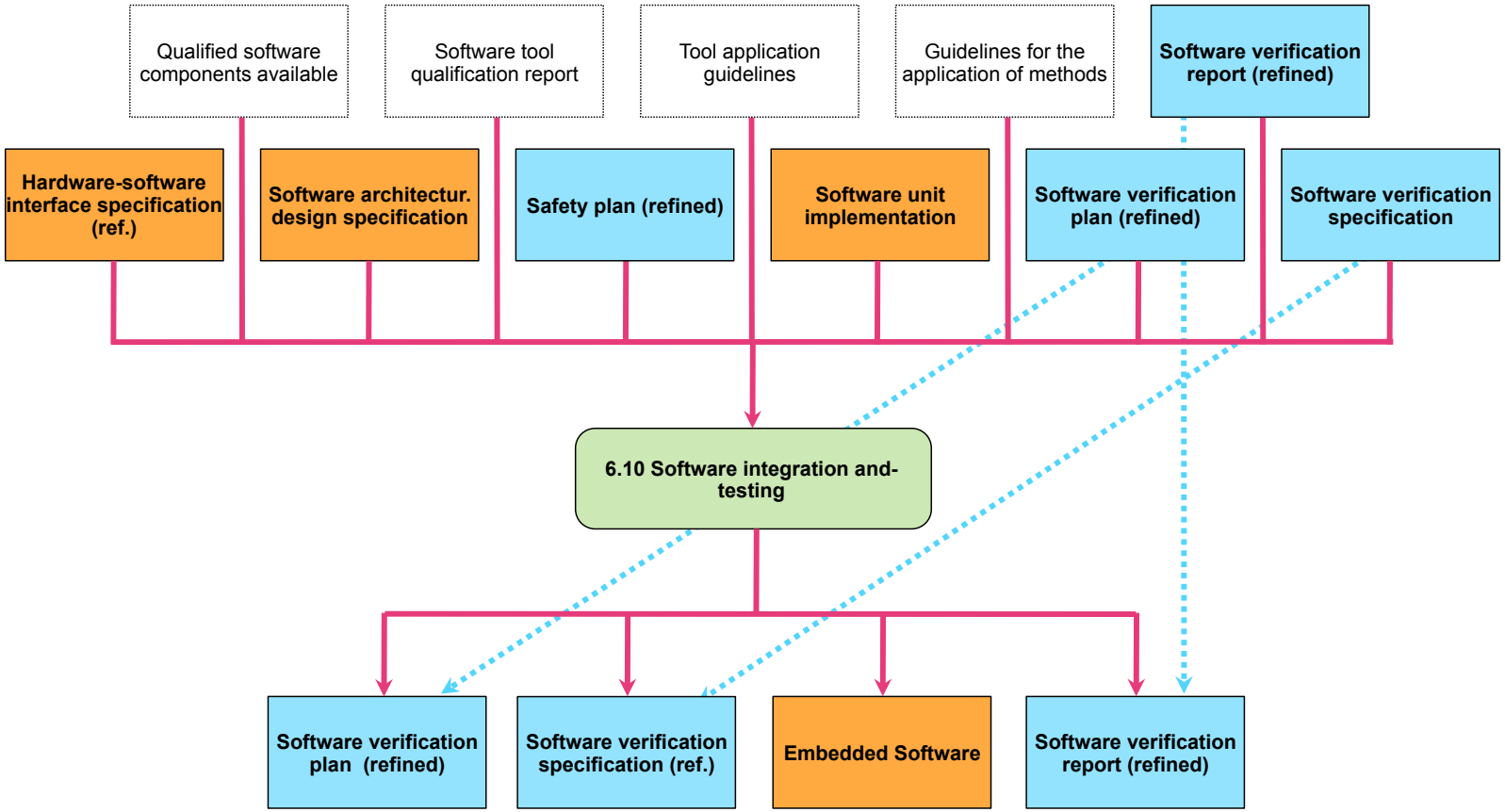


6-10 Software integration and testing



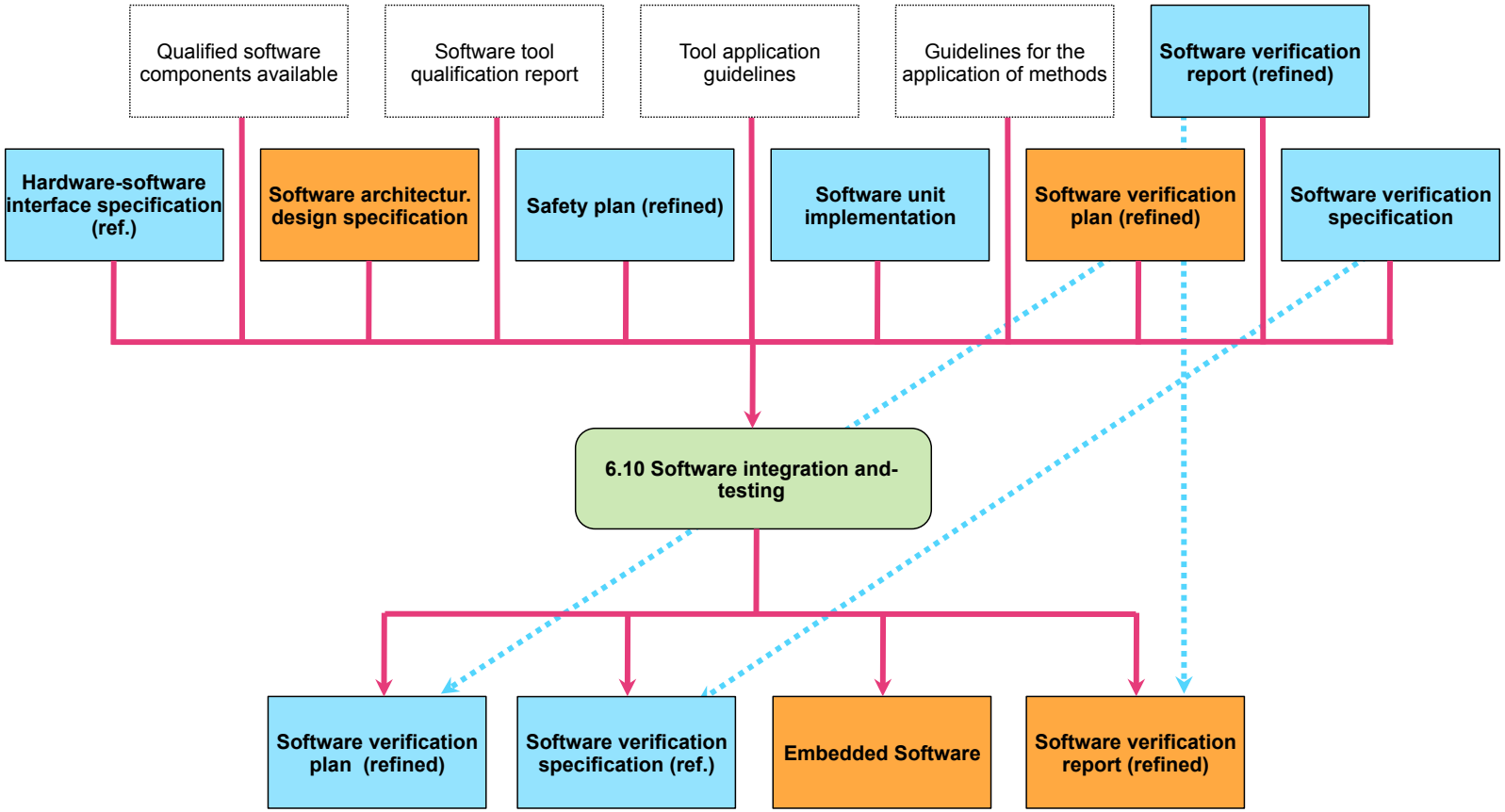
6-10 Software integration and testing

- Integration der Software-Elemente



6-10 Software integration and testing

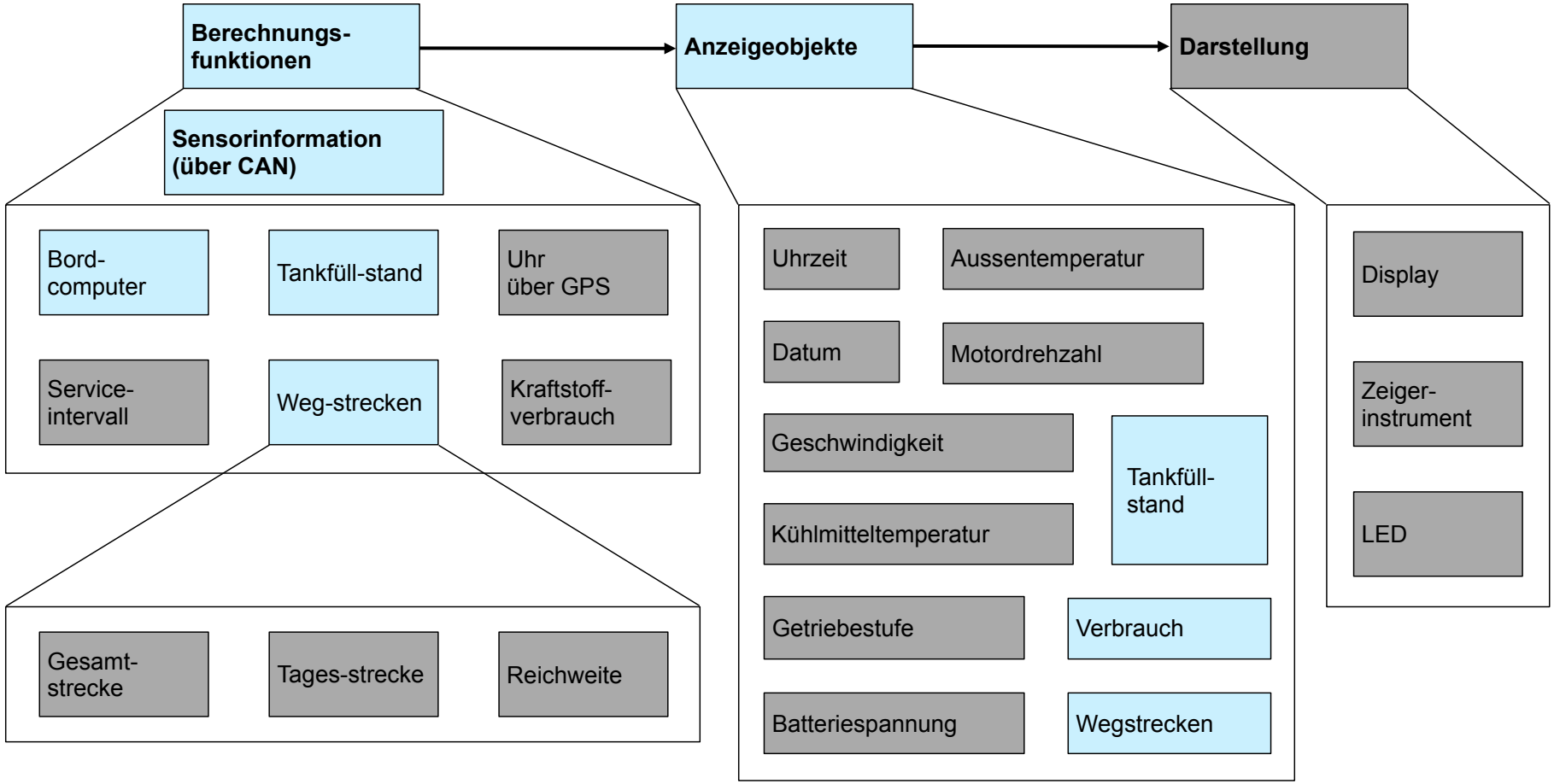
- Nachweis der Umsetzung der SW-Architektur durch die realisierte Software (Embedded software).



Typisches Arbeitsergebnis von 6-10 Embedded SW

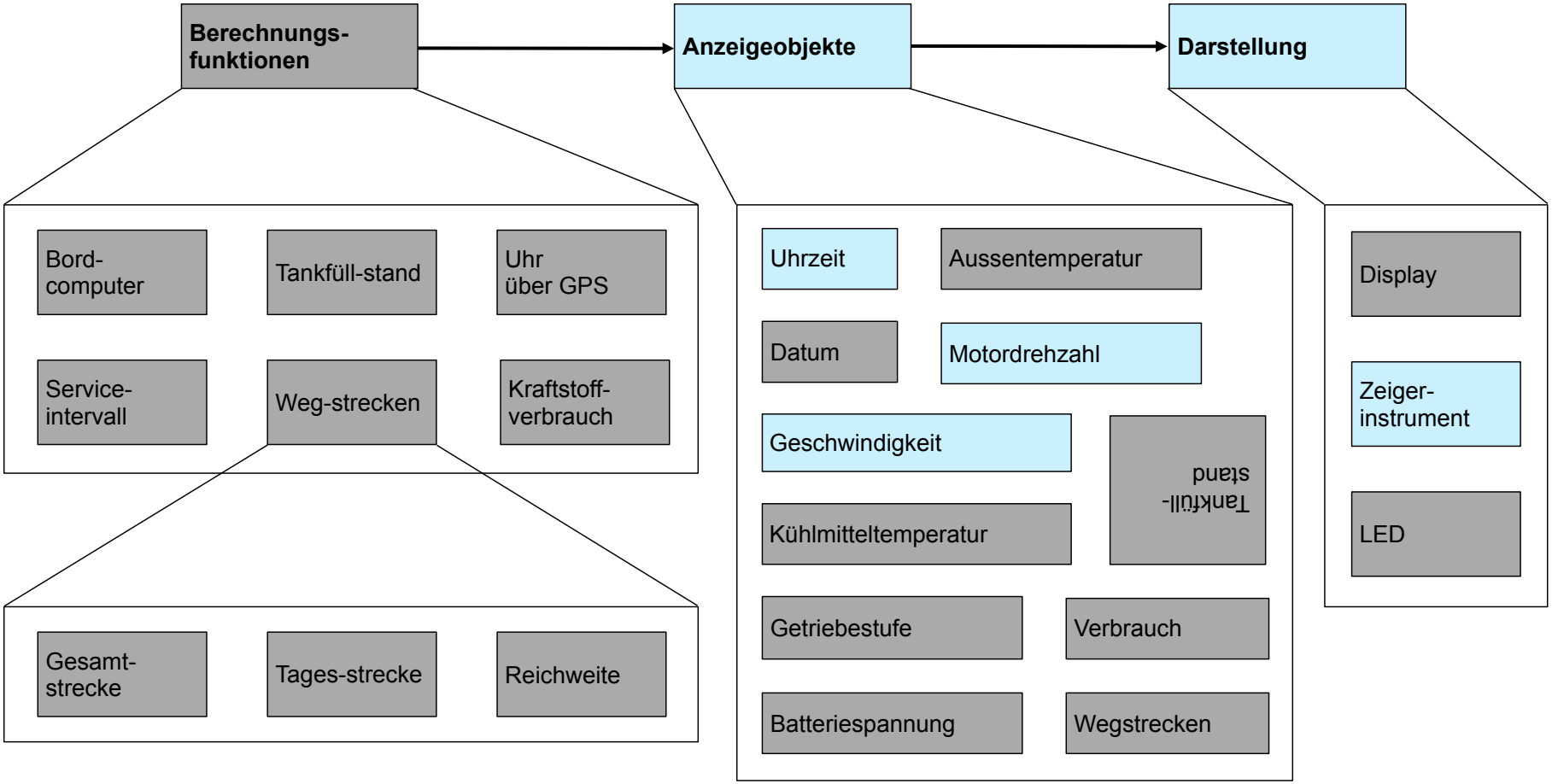
- 10.5 Work products
  10.5.3 Embedded software resulting from requirement 10.4.1.
- 10.4 Requirements and recommendations
- 10.4.1 Planung der SW-Konfiguration
  - Beschreibung der Integrationsschritte
    - SW-Einheiten
    - SW-Komponenten
    - Embedded SW
  - ACHTUNG: KEINE einheitliche Namensgebung, z.B. bei flyXT (V-Modell XT bei EADS):
    - SW-Module
    - SW-Komponenten
    - SW-Einheiten
  - Berücksichtigung von
    - Funktionalen Abhängigkeiten soweit sie für die SW-Integration relevant sind
      Beispiel: Anzeigefunktion benötigt Berechnungsfunktion, Berechnungsfunktion benötigt Sensordaten
    - Abhängigkeiten zwischen SW-Integration und SW-HW-Integration
      Beispiel: Anzeigeinstrument benötigt Anzeigefunktion, Anzeigefunktion benötigt Berechnungsfunktion

Anzeigefunktion benötigt Berechnungsfunktion, Berechnungsfunktion benötigt Sensordaten



Logische Systemarchitektur des Kombiinstruments (Nach Schäuffele, Zurawka)

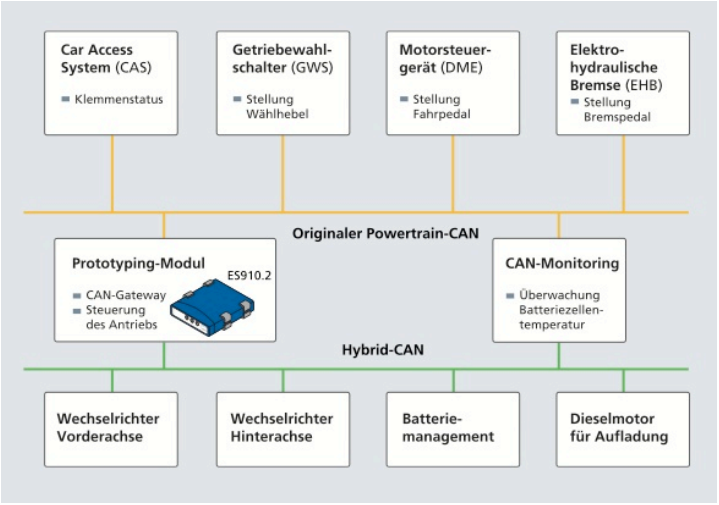
Anzeigeinstrument benötigt Anzeigefunktion

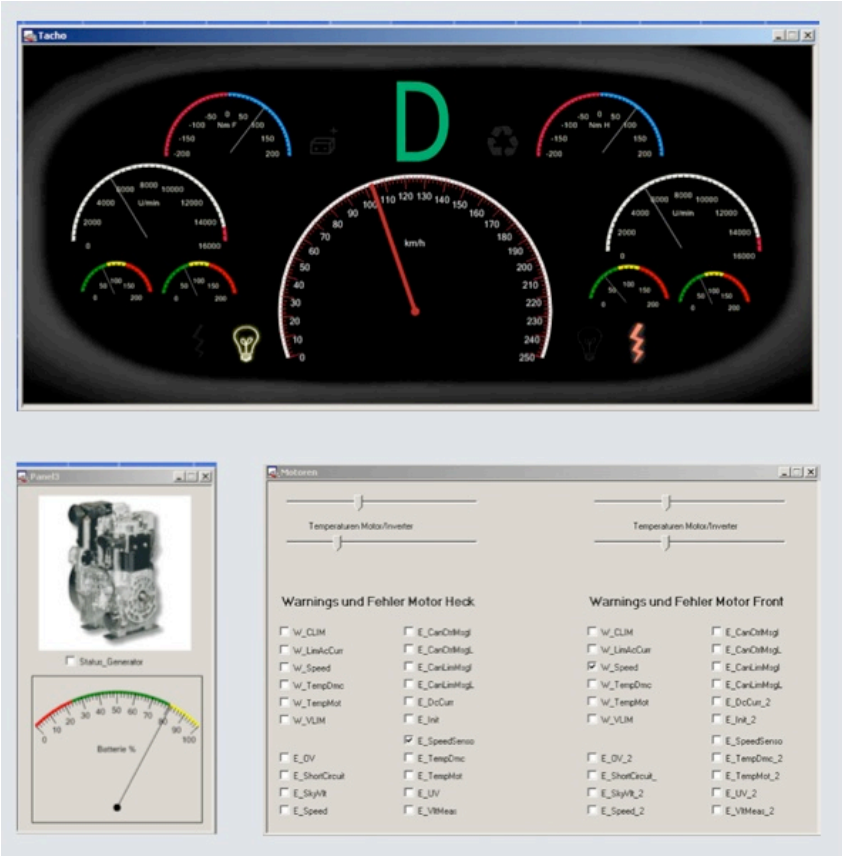


Logische Systemarchitektur des Kombiinstruments (Nach Schäuffele, Zurawka)

Rapid Prototyping als Vorstufe zur Integration (1)

- Im nächsten Schritt wurde das Funktionsmodell mit Hilfe des Werkzeugs INTECRIO auf das Rapid Prototyping-Modul ES910 gebracht. Bevor das Modul im realen Fahrzeug getestet wurde, erfolgte ein weiterer Absicherungsschritt der Gesamt-funktionalität gegenüber einer Restbussimulation (siehe Abbildung).



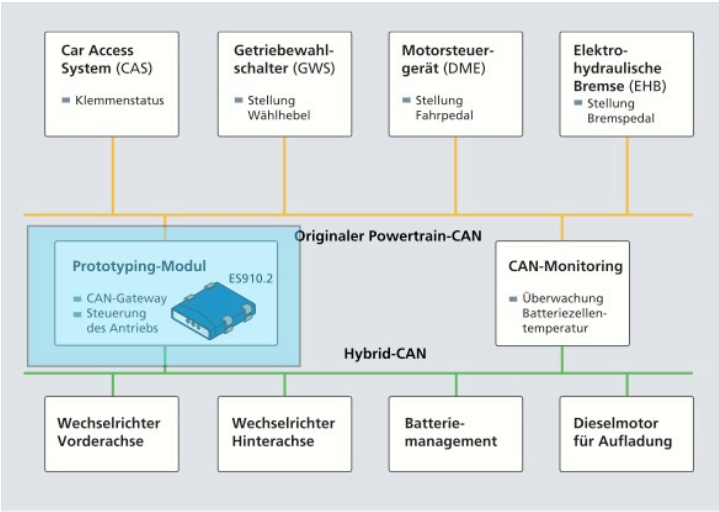


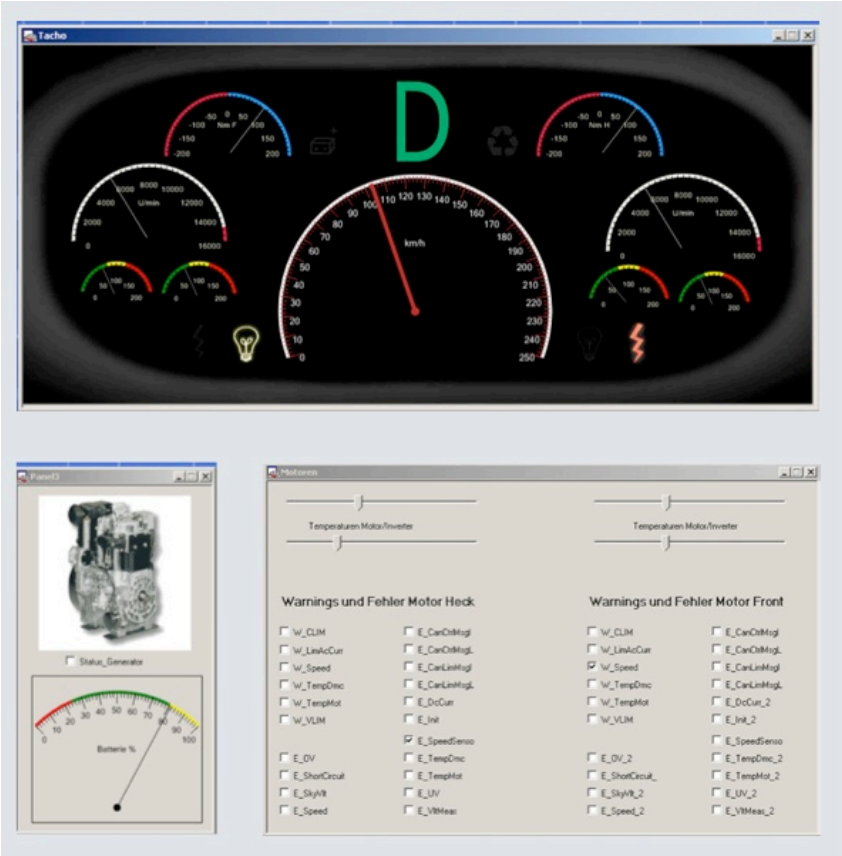
Panels der Restbussimulation

Quelle: Prof. Dr. Dieter Nazareth, FH Landshut
Entwicklung einer Antriebssteuerung für ein Hybridfahrzeug in einer Rapid Prototyping-Umgebung

Rapid Prototyping als Vorstufe zur Integration (2)

- Im nächsten Schritt wurde das Funktionsmodell mit Hilfe des Werkzeugs INTECRIO auf das **Rapid Prototyping-Modul ES910** gebracht. Bevor das Modul im realen Fahrzeug getestet wurde, erfolgte ein weiterer Absicherungsschritt der **Gesamt-funktionalität** gegenüber einer Restbussimulation (siehe Abbildung).



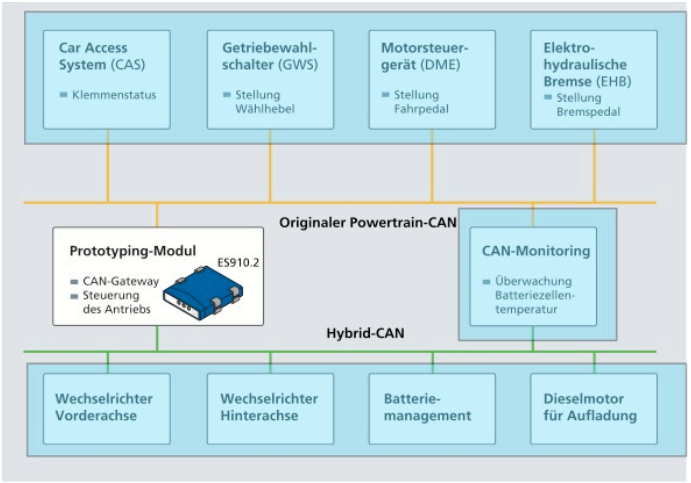


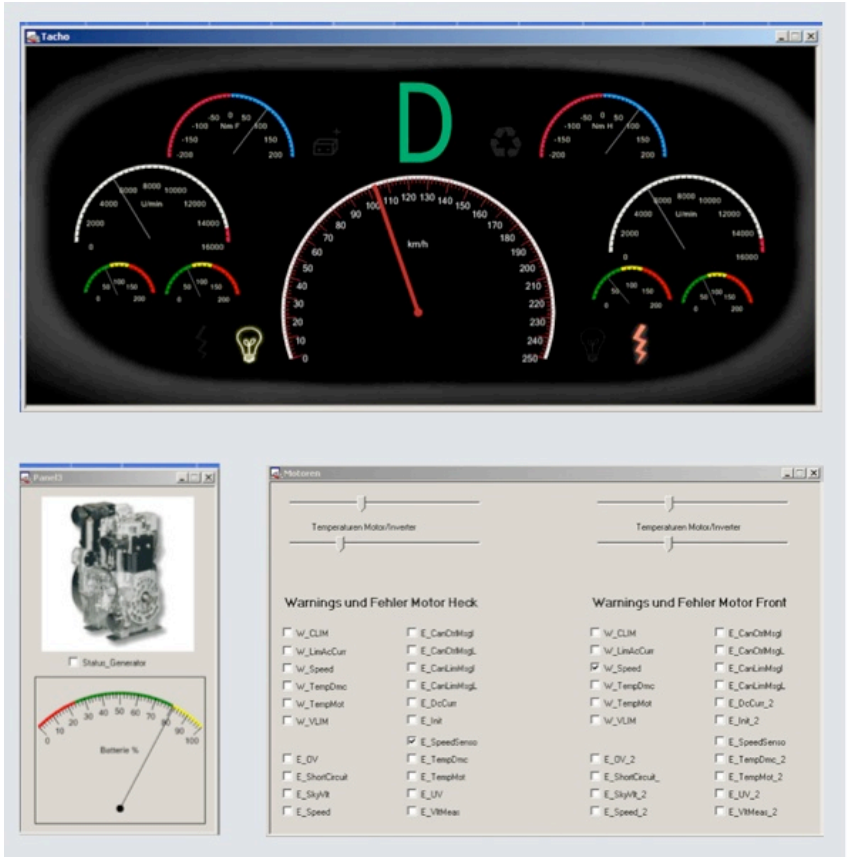
Panels der Restbussimulation

Quelle: Prof. Dr. Dieter Nazareth, FH Landshut
Entwicklung einer Antriebssteuerung für ein Hybridfahrzeug in einer Rapid Prototyping-Umgebung

Rapid Prototyping als Vorstufe zur Integration (3)

- Im nächsten Schritt wurde das Funktionsmodell mit Hilfe des Werkzeugs INTECRIO auf das Rapid Prototyping-Modul ES910 gebracht. Bevor das Modul im realen Fahrzeug getestet wurde, erfolgte ein weiterer Absicherungsschritt der Gesamt-funktionalität gegenüber einer **Restbussimulation** (siehe Abbildung).



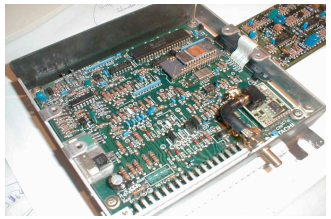


Panels der Restbussimulation

Quelle: Prof. Dr. Dieter Nazareth, FH Landshut
Entwicklung einer Antriebssteuerung für ein Hybridfahrzeug in einer Rapid Prototyping-Umgebung

Begriffsdefinitionen

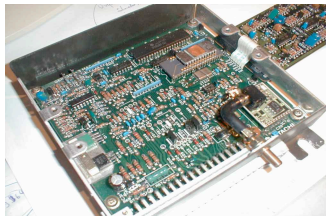
- MIL Model in the Loop, SIL Software in the Loop
  - Modell / SG-Software läuft auf simuliertem SG, das von simulierter Fahrzeugumgebung (rechnergenerierten Sensor- und Bussignalen) gespeist wird.
- HIL Hardware in the Loop
  - Reale SG-Hardware wird von simulierter Fahrzeugumgebung gespeist
- Prototyp
  - Simuliertes Steuergerät im Fahrzeug (“PC im Kofferraum”)
- Prüfstand, Fahrversuch
  - Steuergerät im Fahrzeug unter Laborbedingungen bzw. Auf der Strasse (Testgelände, öffentliches Strassennetz)



	Steuergerät	
Umgebung, Fahrzeug	simuliert	real
simuliert	MIL, SIL	HIL
real	Prototyp	Prüfstand, Fahrversuch

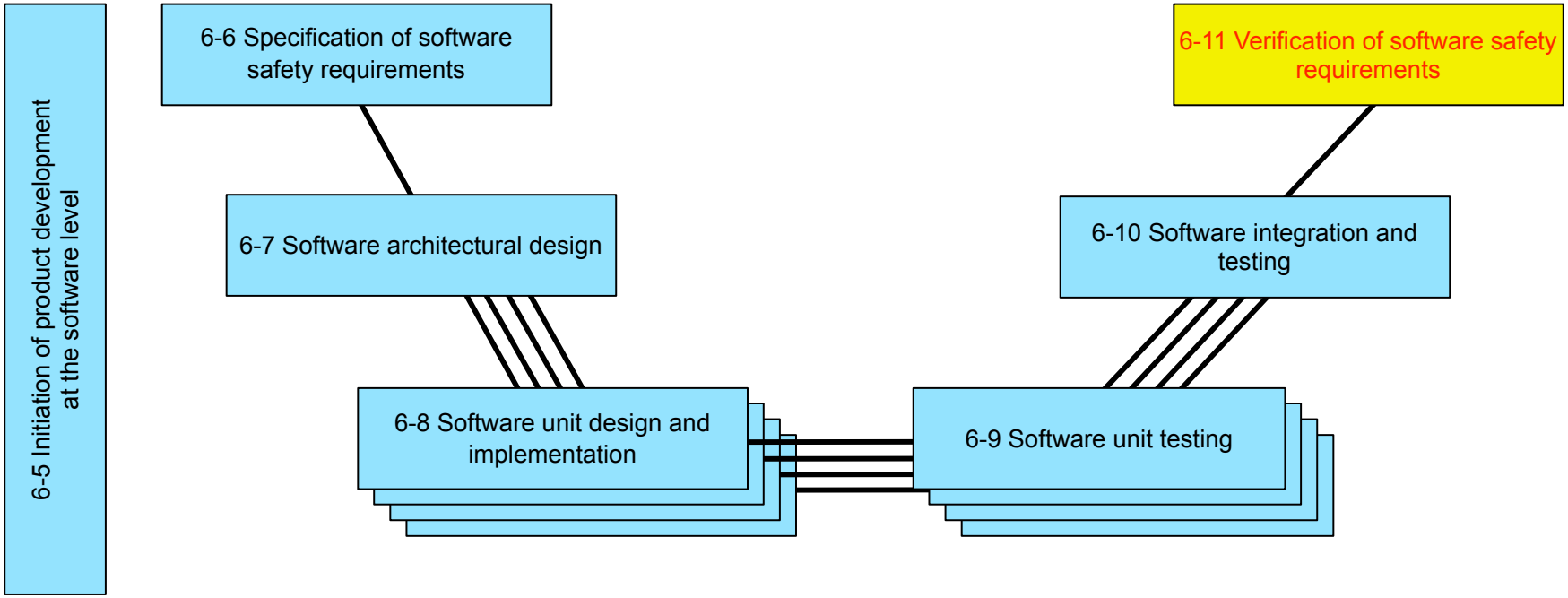
Begriffsdefinitionen

- MIL Model in the Loop, SIL Software in the Loop
  - Modell / SG-Software läuft auf simuliertem SG, das von simulierter Fahrzeugumgebung (rechnergenerierten Sensor- und Bussignalen) gespeist wird.
- HIL Hardware in the Loop
  - Reale SG-Hardware wird von simulierter Fahrzeugumgebung gespeist
- Prototyp
  - Simuliertes Steuergerät im Fahrzeug (“PC im Kofferraum”)
- Prüfstand, Fahrversuch
  - Steuergerät im Fahrzeug unter Laborbedingungen bzw. Auf der Strasse (Testgelände, öffentliches Strassennetz)



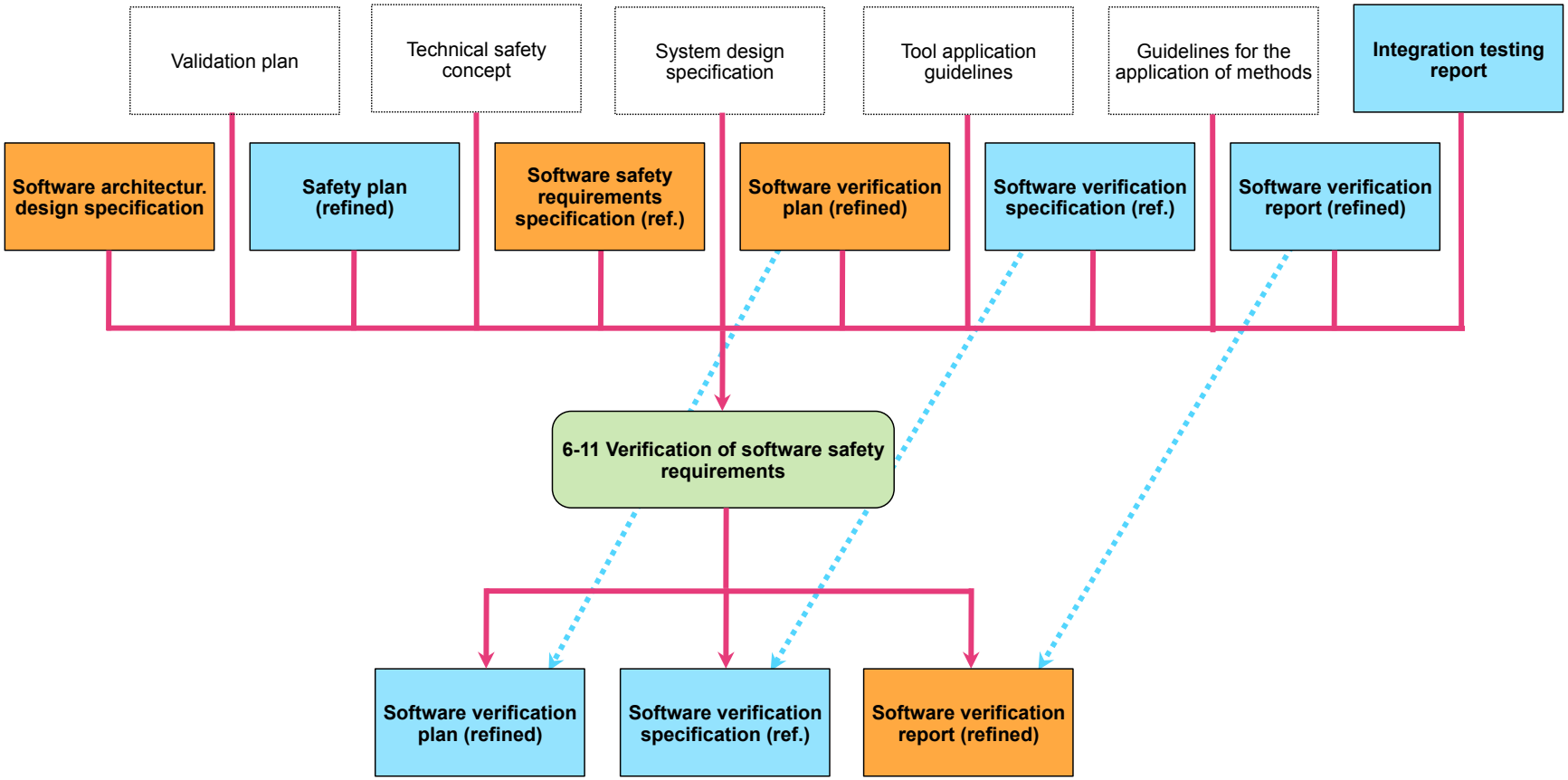
	Steuergerät	
Umgebung, Fahrzeug	simuliert	real
simuliert	MIL, SIL	HIL
real	Prototyp	Prüfstand, Fahrversuch

6-11 Verification of software safety requirements



6-11 Verification of software safety requirements

- Nachweis der Erfüllung der Sicherheitsanforderngen durch die Software.



Typisches Arbeitsergebnis von 6-11
Software verification report

- 11.5 Work products
  11.5.3 Software verification report (refined) resulting from requirements 11.4.1 and 11.4.4.
- 11.4 Requirements and recommendations
- 11.4.1 Verweis auf den Unterstützungsprozess Verifikation
  Der Softwaretest soll in Übereinstimmung mit ISO 26262-8 (Supporting Processes):—, Clause 9 (Verification) geplant, spezifiziert und durchgeführt werden.
- 11.4.4 Verifikation der Sicherheitsanfordeungen an die Software
  Evaluierung der Ergebnisse
  - Übereinstimmung mit den Erwarteten Ergebnissen
  - Abdeckung der Sicherheitsanfordeungen an die Software
  - Kriterien für :-) und :-(

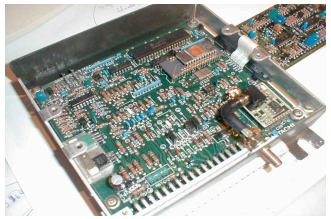
Der Nachweis wird auf der Zielhardware geführt: Erprobung

- Derzeit erfolgt die Erprobung des neuen Antriebssystems sowohl auf einem **Rollenprüfstand** als auch auf der **Straße**. **Ziel der Erprobung ist es, durch Veränderung vielfältiger Parametersätze das Fahrverhalten zu optimieren.**.

Quelle: Prof. Dr. Dieter Nazareth, FH Landshut
Entwicklung einer Antriebssteuerung für ein Hybridfahrzeug in einer Rapid Prototyping-Umgebung

Begriffsdefinitionen

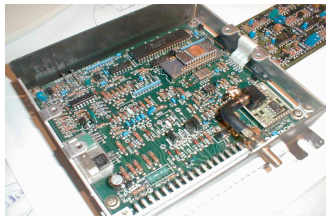
- MIL Model in the Loop, SIL Software in the Loop
  - Modell / SG-Software läuft auf simuliertem SG, das von simulierter Fahrzeugumgebung (rechnergenerierten Sensor- und Bussignalen) gespeist wird.
- HIL Hardware in the Loop
  - Reale SG-Hardware wird von simulierter Fahrzeugumgebung gespeist
- Prototyp
  - Simuliertes Steuergerät im Fahrzeug (“PC im Kofferraum”)
- Prüfstand, Fahrversuch
  - Steuergerät im Fahrzeug unter Laborbedingungen bzw. Auf der Strasse (Testgelände, öffentliches Strassennetz)



	Steuergerät	
Umgebung, Fahrzeug	simuliert	real
simuliert	MIL, SIL	HIL
real	Prototyp	Prüfstand, Fahrversuch

Begriffsdefinitionen

- MIL Model in the Loop, SIL Software in the Loop
  - Modell / SG-Software läuft auf simuliertem SG, das von simulierter Fahrzeugumgebung (rechnergenerierten Sensor- und Bussignalen) gespeist wird.
- HIL Hardware in the Loop
  - Reale SG-Hardware wird von simulierter Fahrzeugumgebung gespeist
- Prototyp
  - Simuliertes Steuergerät im Fahrzeug (“PC im Kofferraum”)
- Prüfstand, Fahrversuch
  - Steuergerät im Fahrzeug unter Laborbedingungen bzw. Auf der Strasse (Testgelände, öffentliches Strassennetz)



	Steuergerät	
Umgebung, Fahrzeug	simuliert	real
simuliert	MIL, SIL	HIL
real	Prototyp	Prüfstand, Fahrversuch

Annex A (informative) Overview of and workflow of management of product development at the software level.

- Tabellarischer Überblick über die sieben Prozessschritte der SW-Entwicklung nach ISO 26262 sowie Annex C Software configuration:
  - Ziele (x.1)
  - Vorraussetzungen (x.3.1)
  - Arbeitsergebnisse (x.5)
- Keine Zusatzinformationen w.r.t. Abschnitte 6-5 – 6-11

Table A.1 — Product development at the software level: overview

Clause	Title	Objectives	Prerequisites	Work products
6-5	Initiation of product development at the software level	Plan and initiate the functional safety activities for the sub-phases of the software development activity.	Project plan (refined) (see ISO 26262-4:—, 5.5.1) Safety plan (refined) (see ISO 26262-4:—, 5.5.2) Technical safety concept (see ISO 26262-4:—, 7.5.1) System design specification (see ISO 26262-4:—, 7.5.2) Item integration and testing plan (refined) (see ISO 26262-4:—, 7.5.4)	5.5.1 Safety plan (refined) 5.5.2 Software verification plan 5.5.3 Design and coding guidelines for modelling and programming languages 5.5.4 Tool application guidelines

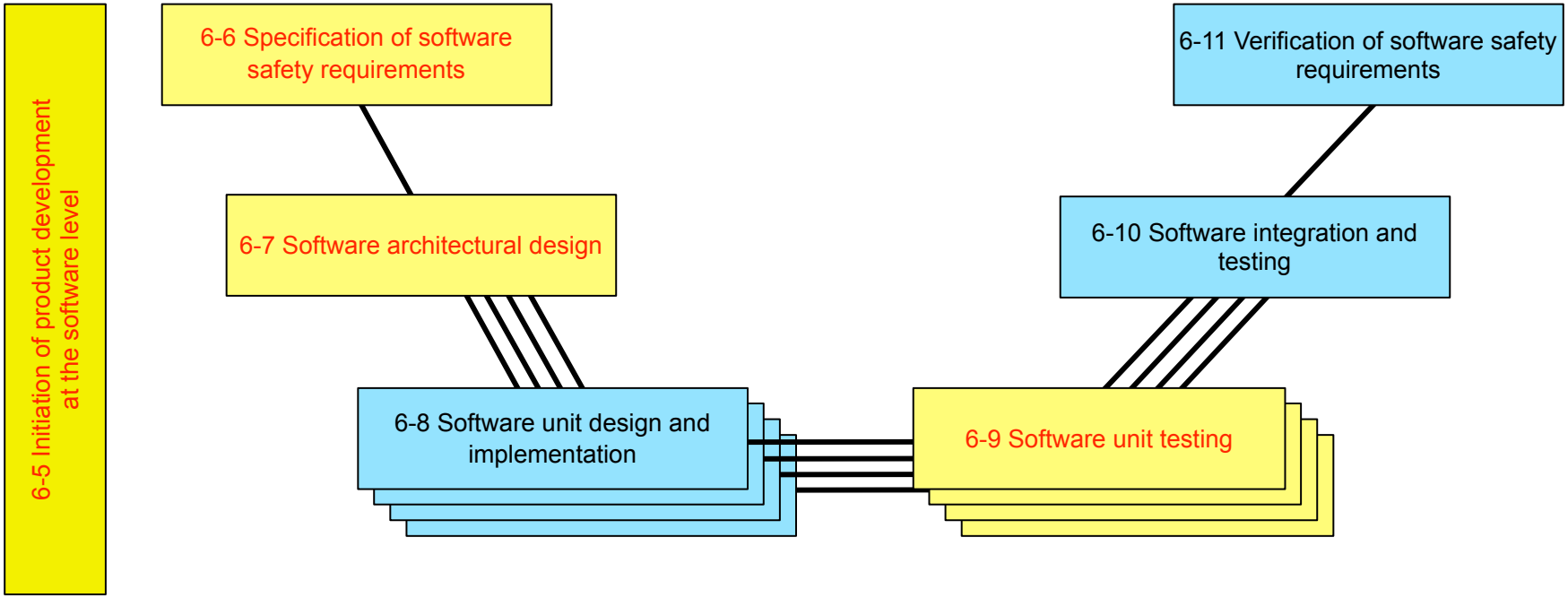
Annex B (informative) Model-based development

- Annex B beschreibt Grundlagen und Vorgehensweise bei modellbasierter Entwicklung.

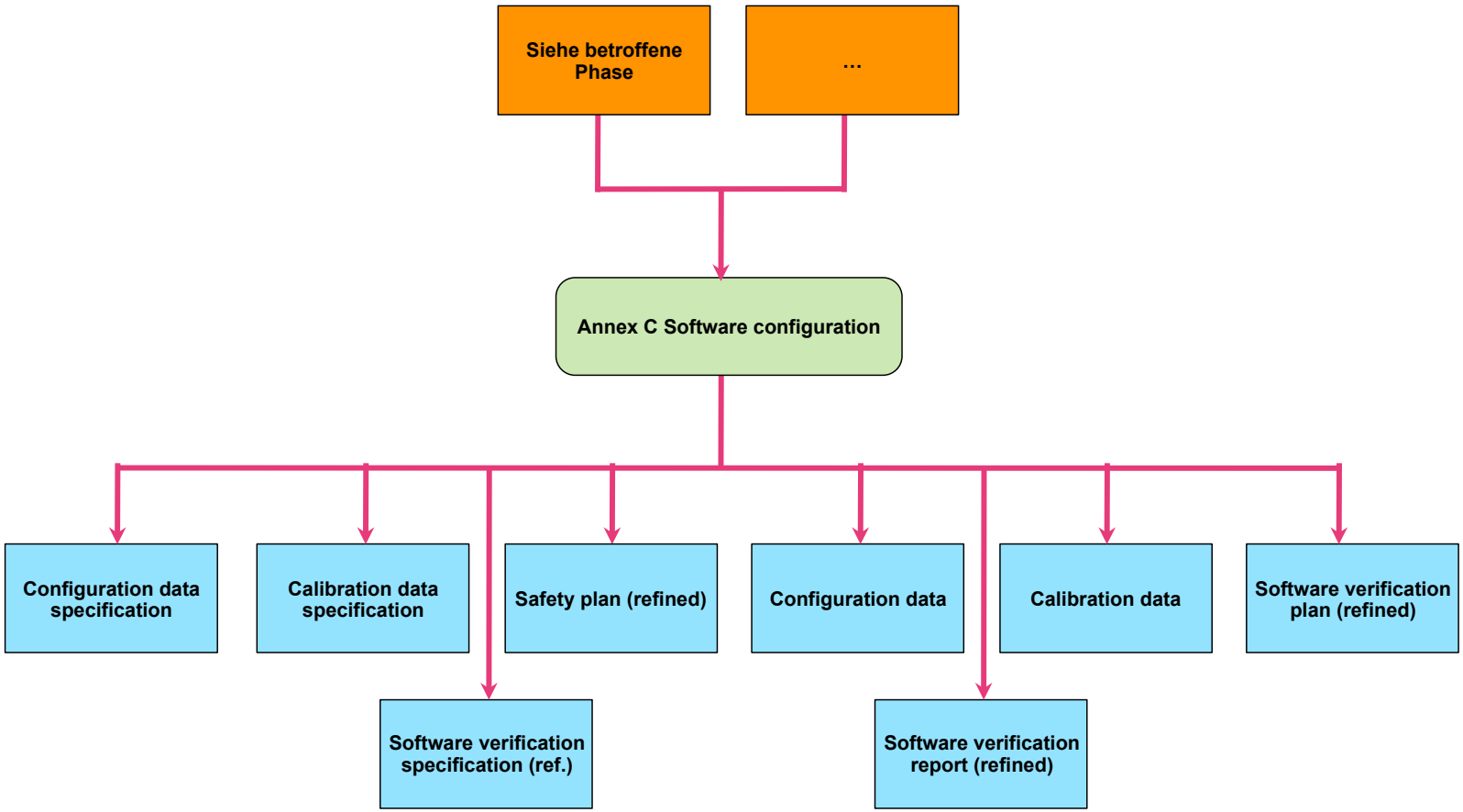
Annex C (normative) Software configuration

- Annex C beschreibt den Umgang mit konfigurierbarer und/oder kalibrierbarer Software.
- Der Aufbau von Annex C entspricht dem Aufbau der Abschnitte 6-5 bis 6-11 (Beschreibungen der sieben Prozessschritte der SW-Entwicklung nach ISO 26262)
- Vorraussetzungen und Zusatzinformationen werden nicht explizit genannt sondern es wird auf die relevanten Phasen verwiesen, in denen Software Konfiguration angewendet wird. (Diese zu finden bleibt dem Leser überlassen :-()
- In den Unterabschnitten x.4 „Requirements and recommendations“ der betreffenden Phasen steht dann z.B.
  5.4.3 If developing configurable software, Annex C shall be applied.
- Betroffen sind
  - 5 Initiation of product development at the software level
  - 6 Specification of software safety requirements
  - 7 Software architectural design
  - 9 Software unit testing

Phasen, in denen Software Konfiguration relevant ist

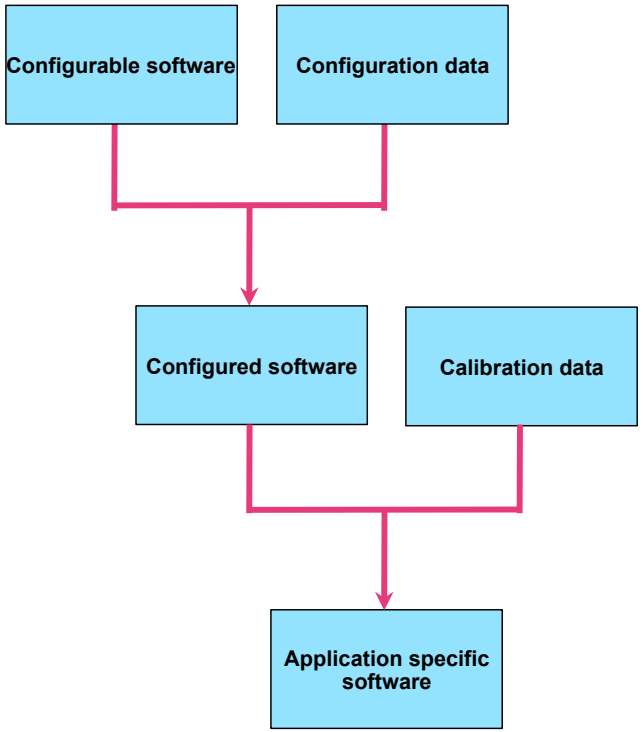


Annex C Software configuration



Annex C Software configuration

- Ziel der Software Konfiguration ist es, die Software für unterschiedliche Anwendungen kontrolliert zu ändern.
- Konfigurierbare Software erlaubt die Entwicklung anwendungsspezifischer Software durch Konfigurations- und Kalibrierungsdaten.
- Beispiel für Konfiguration: Schaltgetriebe oder Automatikgetriebe
- Beispiel für Kalibrierung: Rundlaufen des Motors durch Verstellen der Zündzeitpunkte.
- Konfiguration: Anpassung an unterschiedliche Anforderungen
- Kalibrierung: Anpassung an die Physik; Nachjustieren



Typisches Arbeitsergebnis von Annex C
Configuration data specification

- C.5 Work products
  C.5.1 Configuration data specification resulting from requirements C.4.1 and C.4.3.
- C.4 Requirements and recommendations
- C.4.1 Spezifikation der Konfigurierungsdaten
  Die korrekte Verwendung der Konfigurierungsdaten soll beschrieben werden. Dazu gehören
  - Die gültigen Werte der Konfigurierungsdaten
  - Zweck und Verwendung der Konfigurierungsdaten
  - Bereich, Auflösung und Einheiten
    Beispiel: Motortemperatur aus 6-8 Software unit design and implementation
  - Gegenseitige Abhängigkeit der Konfigurierungsdaten
    Beispiel: Regensensor und sensorgesteuerter Komfortscheibenwischer
- C.4.3 Der ASIL der Konfigurierungsdaten soll dem höchsten ASIL der nutzenden konfigurierbaren Software entsprechen.

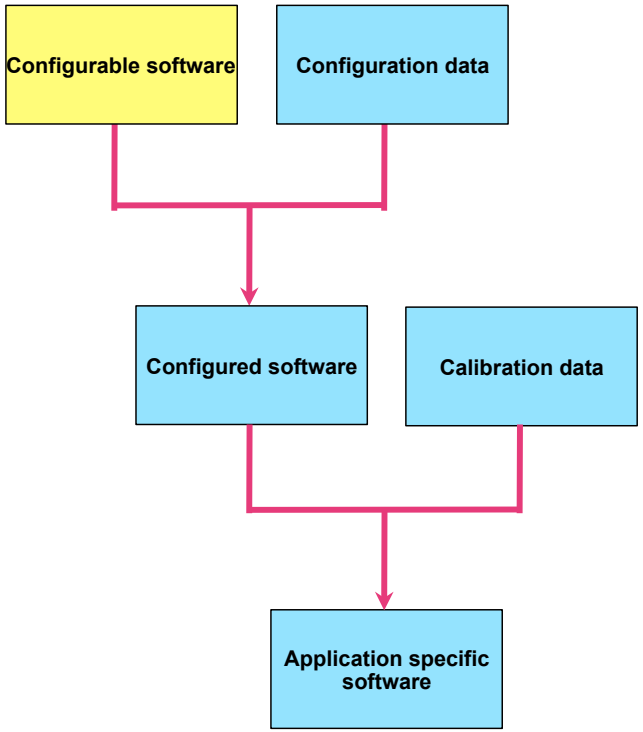
Der Nachweis wird auf der Zielhardware geführt: Erprobung

- Derzeit erfolgt die Erprobung des neuen Antriebssystems sowohl auf einem **Rollenprüfstand** als auch auf der **Straße**. **Ziel der Erprobung ist es, durch Veränderung vielfältiger Parametersätze das Fahrverhalten zu optimieren.**.

Quelle: Prof. Dr. Dieter Nazareth, FH Landshut
Entwicklung einer Antriebssteuerung für ein Hybridfahrzeug in einer Rapid Prototyping-Umgebung

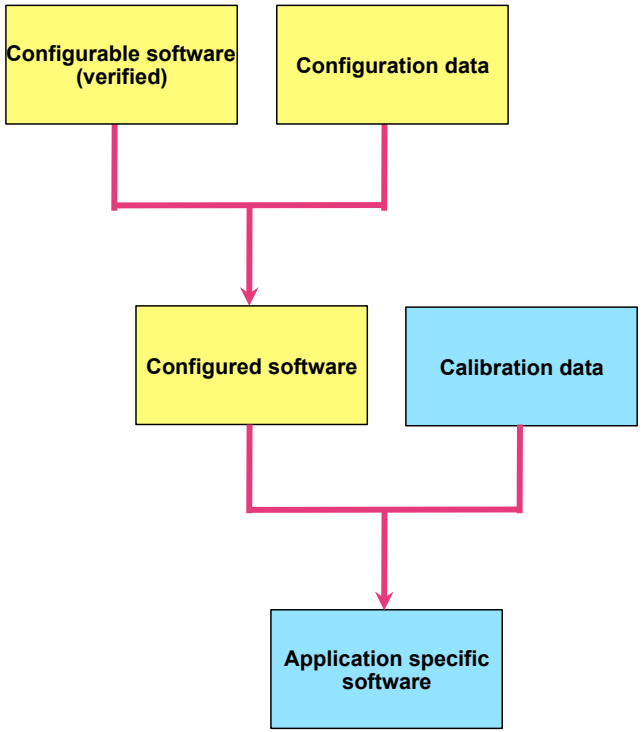
Annex C Software configuration Verifikation

- Schritt 1:
  Verifikation der konfigurierbaren Software



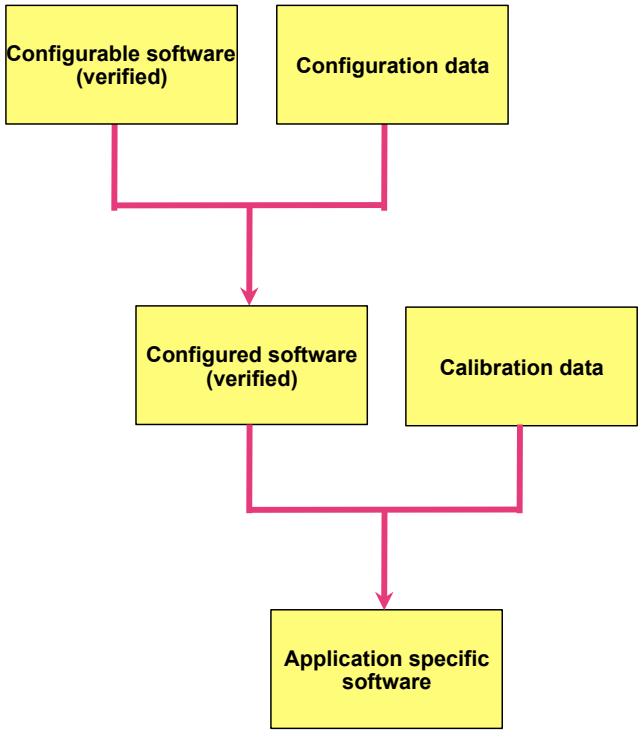
Annex C Software configuration Verifikation

- Schritt 2:
  Verifikation der konfigurierten Software unter Verwendung der verifizierten konfigurierbaren Software und der Konfigurationsdaten
- Die konfigurierbare Software wird für beliebige (gültige) Konfigurationsdaten nur einmal verifiziert



Annex C Software configuration Verifikation

- Schritt 3:
  Verifikation der anwendungs-spezifischen Software unter Verwendung der verifizierten konfigurierten Software und der Kalibrierungsdaten
- Die konfigurierte Software wird für beliebige (gültige) Kalibrierungsdaten nur einmal verifiziert



Annex D (informative) Freedom from interference between software elements

- Annex D behandelt mögliche Fehler durch „Wechselwirkungen“ zwischen Software-Elementen, die auf der selben Hardware laufen, sowie Mechanismen zur Verhinderung, Entdeckung und Abschwächung dieser Fehler.
- Fehler können auftreten bei der gemeinsamen Nutzung von CPU und Speicher sowie beim Informationsaustausch.

Bibliography

- [1] SO/IEC 12207:2008, Systems and software engineering — Software life cycle processes.
- [2] IEC61508-SER:2005,Functional safety ofelectrical/electronic/programmable electronic safety-related systems — all parts.
- [3] IEC 61508-3:1998 Functional safety of electrical/electronic/programmable electronic safety-related systems — Part 3: Software requirements.
- [4] MISRA C Guidelines for the use of the C language in critical systems, ISBN 978-0-9524156-2-6,MIRA, October 2004.
- [5] MISRA AC AGC Guidelines for the application of MISRA-C:2004 in the context of automatic code generation, ISBN 978-1-906400-02-6, MIRA, November 2007.

Sichten auf das Vorgehensmodell ISO 26262 (1)

- Wer?
  Rollenorientierte Sicht:
  Wer ist für welche Arbeitsergebnisse (Produkte, Work Products) verantwortlich?
  - Nur teilweise definiert
  - Beispiel für Definition: Safety Management
    Teil 2, 5.4.2.3 The organization shall institute, execute and maintain processes to ensure that unresolved functional safety anomalies are explicitly communicated to the safety manager, and other responsible persons.
  - Details in Teil 2,
    6 Safety management during the concept phase and the product development
  - Keine explizite Definition z.B. der Rollen
    - Softwareentwicklung
    - Hardwareentwicklung
    - ...

Sichten auf das Vorgehensmodell ISO 26262 (2)

- Was?
  Produktorientierte Sicht:
  Was sind die zu erarbeitenden Produkte?
  - Implizite Sicht auf Produkte: Es wird beschrieben, welche Produkte Ergebnisse eines Arbeitsschrittes sind.
  - Einige Produkte sind Arbeitsergebnisse von genau einer Teilphase. Beispielsweise ist das Produkt „Software architectural design specification" Ergebnis ausschliesslich der Teilphase 6-7 „Software architectural design". In diesem Fall kann die Struktur des Dokumentes zur Beschreibung des entsprechenden Produktes leicht aus der ISO 26262 abgeleitet werden.
  - Einige Produkte werden jedoch in mehreren (zum Teil mehr als fünf) Teilphasen verfeinert. Ein Beispiel hierfür ist das Produkt „Software verification plan". In diesem Fall ist die Information, die für die Strukturierung des beschreibenden Dokumentes benötigt wird, über mehrere Kapitel bzw. Teile der ISO 26262 verteilt.
  - Bei anderen Produkten lässt die ISO 26262 Raum für Interpretationen. Ein Beispiel ist der „Validation plan". Dazu heisst es lediglich: „The validation activities shall be planned" (ISO 26262-4, 5.4.2).
- Wie?
  Aktivitätenorientierte Sicht:
  Wie werden die Produkte erarbeitet?
  - ISO 26262 kennt keine Aktivitäten

Sichten auf das Vorgehensmodell ISO 26262 (3)

- Wann?
  Phasenorientierte Sicht:
  Wann werden welche Arbeitsschritte durchgeführt?
  - Explizit in ISO 26262
- Wann?
  Meilensteinorientierte Sicht:
  Wann werden welche Produkte fertiggestellt und geprüft?
  - ISO 26262 kennt keine Meilensteine
- Womit?
  Methoden- und werkzeugorientierte Sicht:
  Womit (Methoden und Werkzeuge) werden die Produkte erarbeitet?
  - ISO 26262 nennt ein breites Spektrum an Methoden und Werkzeugen

ISO 26262 - Anmerkungen und offene Punkte

- ISO 26262 kennt nur ein rudimentäres Vorgehensmodell (siehe die drei vorigen Folien)
- Informationen stehen oft nicht an den Stellen, wo sie benötigt werden, sondern sind mehrere Kapitel bzw. Teile der ISO 26262 verteilt.
  - Beispiele
    - Produkte allgemein
    - Konfiguration (siehe Annex C (normative) Software configuration)
- Informationen sind teilweise rudimentär.
  - Beispiel „Validation plan“. Dazu heisst es lediglich: „The validation activities shall be planned“ (ISO 26262-4, 5.4.2).

Reinhard Wilhelm: Auf ewig sicher !?

nach 33.000 Jahren hinein bringt, ist noch strittig.

Es wird aber auch bei den Autobauern bald alles besser! ISO-26262 kommt! Diese Norm nimmt die Tradition der Coding Rules auf, also der Einschränkung von Programmierkonzepten. Eine Table 1 empfiehlt unter „Use of language subsets" „the exclusion of language constructs which might result in unhandled runtime errors". Das ist mal eine klare Empfehlung! Arithmetik – kann zu Überläufen führen. Weg damit! Indizierung in Feldern – könnte außerhalb der Feldgrenzen liegen. Raus! Zeiger – es droht die Dereferenzierung von Nullzeigern. Schon weg! Schleifen und Rekursion – terminieren eventuell nicht. Fort damit! Es bleibt eine Sprache übrig, deren Programme zwar nichts mehr machen können, aber dafür leicht zu zertifizieren sind. In Abwandlung von Ludwig Wittgensteins berühmtem Zitat könnte man eine Erfolgsmeldung formulieren, „Was man nicht sagen kann, muss man auch nicht zertifizieren."

Quelle: Kolumne „Einsichten eines Informatikers von geringem Verstande"
Informatik_Spektrum_34_4_2011

ISO 26262 ist veröffentlicht – und jetzt? (Vortrag)

Referent:	Dipl.-Phys. Stefan Kriso , Robert Bosch GmbH
Vortragsreihe:	Sichere Software
Zeit:	07. Dezember 8: 17:30-18:15

Zielgruppe

Entwicklung, Management, Fortgeschrittene, Experten

Themenbereiche

Sichere Software, anderes Themengebiet

Kurzfassung

Mitte 2011 wird die ISO 26262 veröffentlicht und trägt damit zum Stand der Technik bei der Entwicklung sicherheitsrelevanter Systeme im Kfz bei. Wie die Vergangenheit gezeigt hat, rückt das öffentliche Interesse an der Fahrzeugsicherheit immer mehr in den Vordergrund, gleichzeitig wächst aber auch die Komplexität automobiler Systeme. Das Thema „Safety“ gewinnt immer mehr an Bedeutung, insbesondere ergeben sich z.B. im Zusammenspiel von Safety und Security, bei der Anwendung von Open Source Prinzipien oder bei der Entwicklung von E-Fahrzeugen ganz neue Fragestellungen bzgl. der Fahrzeugsicherheit. Dieses rasche Weiterentwickeln des Stands der Technik hat zur Folge, dass die gerade erst veröffentlichte ISO 26262 zwangsläufig diesem neuen Stand der Technik hinterherhinken wird, es zeichnen sich heute schon Fragestellungen ab, die die ISO 26262 (noch) nicht adressiert.

Nutzen und Besonderheiten

Es ist zu beobachten, dass die Automobilbranche damit begonnen hat, die ISO 26262 flächendeckend einzuführen. Viele Themen, die in der Norm einfach zu lesen sind, stellen sich in der Praxis teilweise als schwierig umsetzbar heraus – eine Interpretation der Normanforderung ist hier nicht nur möglich und sinnvoll, sondern geradezu notwendig, um zu einer sinnvollen Normumsetzung zu kommen. Der Vortrag geht darauf ein, welche Themen in der ISO 26262 bislang nicht oder nicht ausreichend adressiert wurden, macht Vorschläge, wie damit umgegangen werden kann und wo die ISO 26262 entsprechend interpretiert werden muss, um sie sinnvoll anwenden zu können.

Über den Referenten

- Mitarbeit bei der Erstellung und Bewertung von Sicherheitskonzepten für X-by-Wire-Systeme - Koordination der Einführung der ISO 26262 bei Bosch - Mitarbeit bei der Ermittlung der Bosch-Interessen bzgl. ISO 26262 und deren Vertretung in den entsprechenden Normungsgremien - Ab 07/2011 Leitung des neu gegründeten "Center of Competence Functional Safety" im Bosch-Konzern