Karlsruher Institut für Technologie

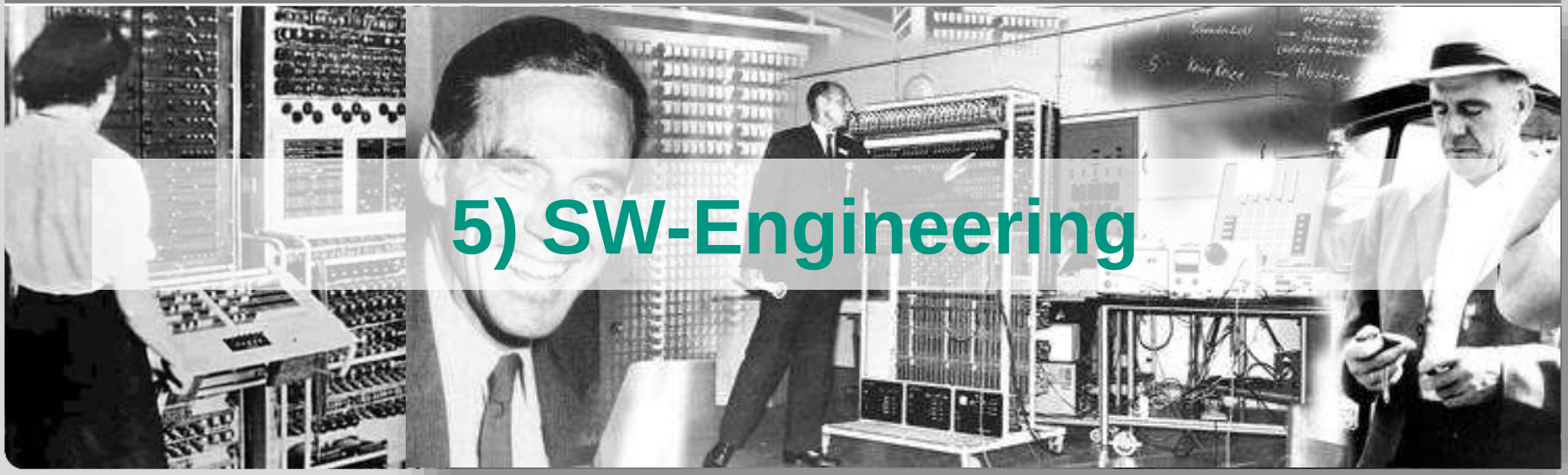
Institut für Technik der Informationsverarbeitung

Institutsleitung
Prof. Dr.-Ing. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Vorlesung Informationstechnik (IT)

Sommersemester 2018

Institut für Technik der Informationsverarbeitung (ITIV)



5) SW-Engineering

IT Inhalt

0. Einleitung und Motivation

- Organisatorisches
- Begriffe
- Typische Anwendungen

1. Rechnerarchitekturen

- Einführung
- Allgemeiner Aufbau der internen Hardware Architektur
- Instruction Set Architecture (ISA) am Beispiel AVR8
- Klassifikation von Rechnerarchitekturen
- Performanzsteigerung
- ausgewählte Beispiele

IT Inhalt

2. Programmiersprachen

- Software-Komponenten
- Was ist eine Programmiersprache?
- Formale Sprachen
- Höhere Sprachen
- Programmierparadigmen
- Wichtige Programmiersprachen

IT Inhalt

.

IT Inhalt

SW-Engineering

5.1 Qualität von Software

5.2 Entwicklungsablauf

5.3 Verifikation & Validierung

5.4 Tests

5.5 Kundenakzeptanz

Wesentliche Merkmale guter Software

Zuverlässigkeit	• Verlässlichkeit • Zugriffsschutz • Betriebssicherheit, keine körperlichen, seelischen oder wirtschaftlichen Schäden
Wartungsfreundlichkeit	• Softwareveränderung • Weiterentwicklung • Diagnostizierbarkeit
Effizienz	• keine Verschwendung von Systemressourcen wie Speicher oder Prozessorkapazität • Kosteneffizienz
Benutzerfreundlichkeit	• nutzbar ohne unangemessene Anstrengung • Benutzeroberfläche • Dokumentation
Nützlichkeit	• Erfüllung der Benutzeranforderungen

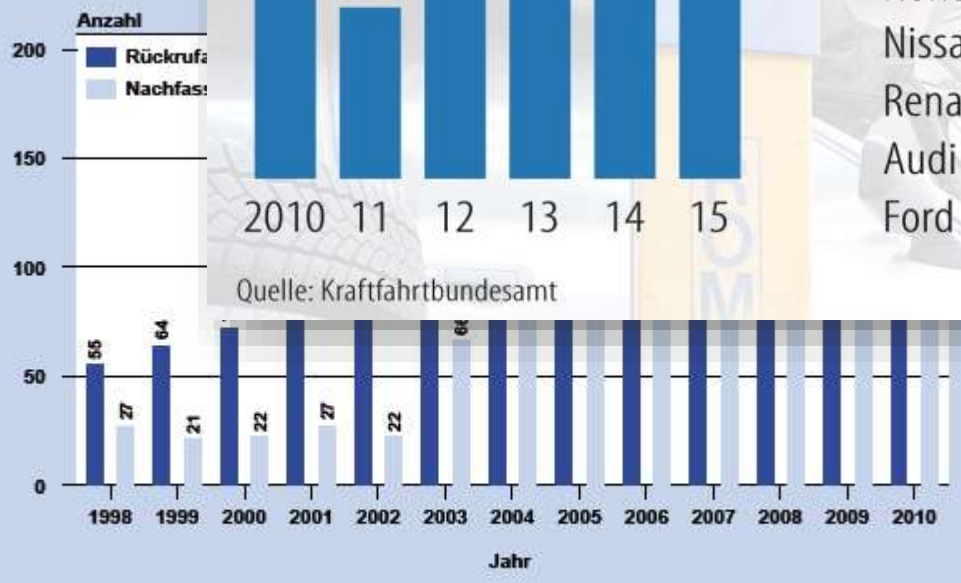
prime error source: requirements specification

More than 40%
during require

Percentage of er
problem type on

Environm
Human
Document-
ation
2%

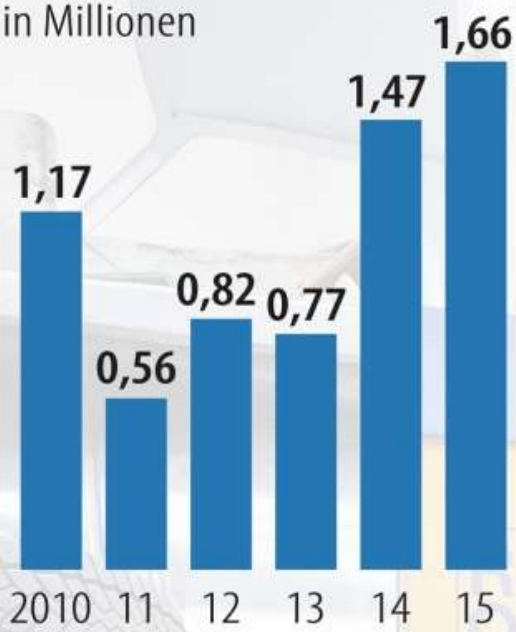
Anzahl der Rück



Quelle: Jahresbericht 2010, Kraftfahrt-Bundesamt (KBA) Auszug, S. 62

Rückrufe von Fahrzeugen in Deutschland

Zahl der Rückrufe
in Millionen



Quelle: Kraftfahrtbundesamt

Rückrufe und Neuzulassungen nach Marken
2010 bis 2015 in Millionen

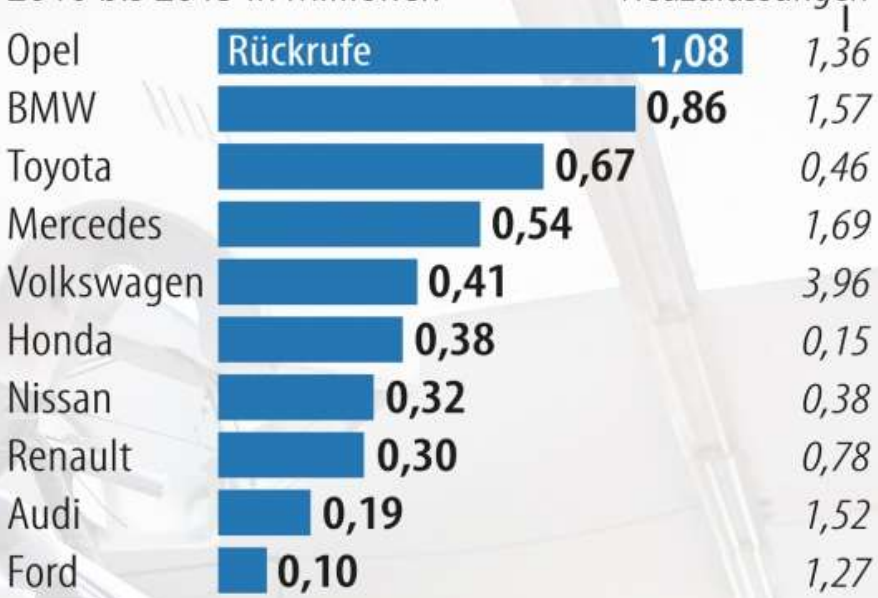


Foto Felix Seuffert/F.A.Z.-Grafik Walter

Software Engineering

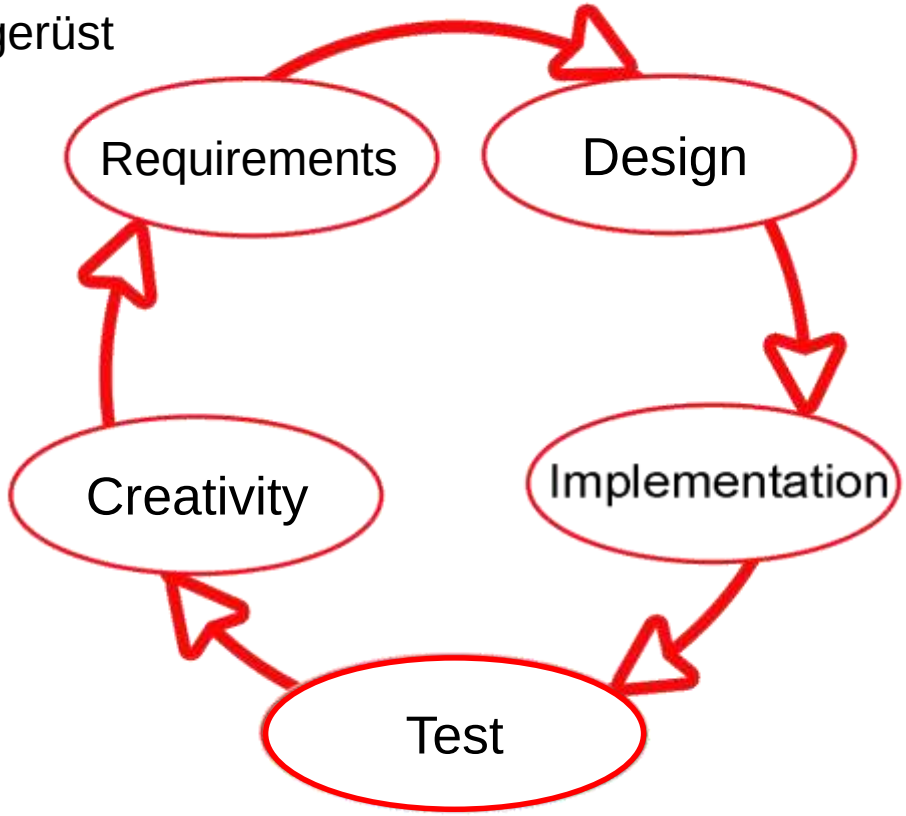
- *„Zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen.“*
  (Balzert, S.36)
- Technische Disziplin, die sich mit allen Aspekten der Software-Herstellung beschäftigt:
  - Von den frühen Phasen der Systemspezifikation, der Analyse, dem Design, der Implementierung, dem Test, bis hin zur Wartung des Systems, nachdem sein Betrieb aufgenommen wurde

Grundsätzliche Einsicht

- Ein gleiches Verständnis der gemeinsamen Vorgehensweise, die nicht jedesmal neu erfunden wird, sondern sich auf eine Referenz bzw. etwas Bestehendes abstützt, ist wertvoll.
- Für Projekte einer Organisation ist daher ein einheitliches Prozessmodell bei ähnlicher Aufgabenstellung sinnvoll.
- Dieses Prozessmodell ist ein generisches Muster und wird pro Aufgabenstellung instantiiert bzw. zugeschnitten (*tailored*).
- Es gibt keinen idealen Software-Entwicklungsprozess.
- Prozessmodelle sind nicht gut oder schlecht, sondern zum Projekt bzw. der Aufgabe passend oder unpassend.
- Allen Vorgehensmodellen gemein sind Phasen:
  Kreativität, Analyse/Anforderungen, Design, Implementierung, Test

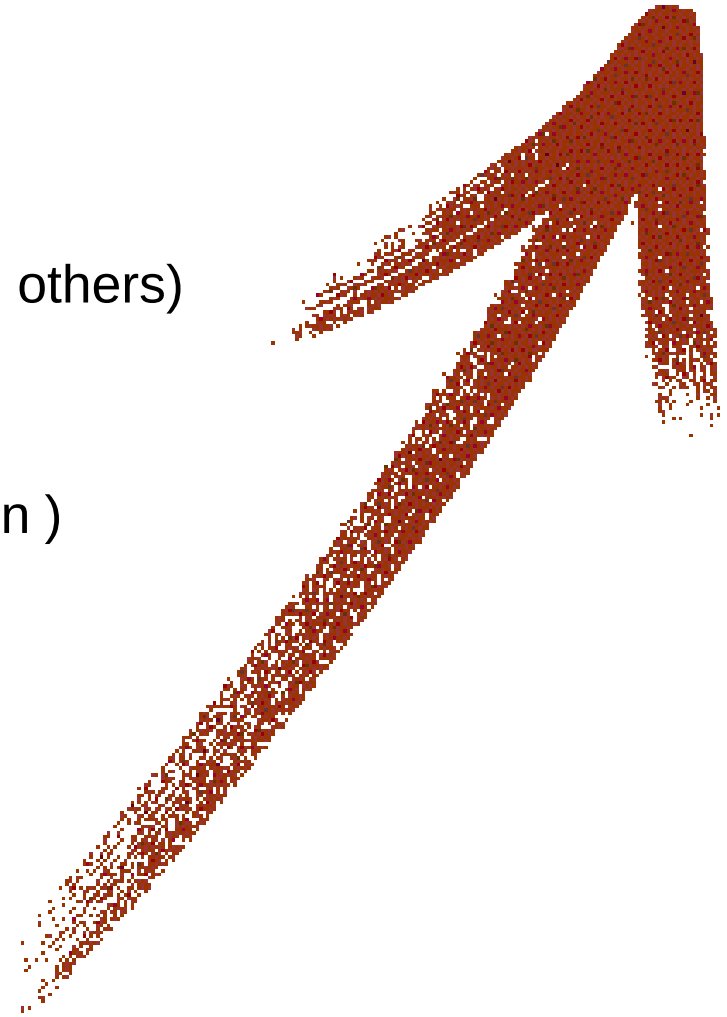
Was ist ein Phasenmodell?

- Reihe von Aufgaben (design steps)
  - Passende, auf einander aufbauende Aktivitäten
  - Definition von erwarteten Teilergebnissen
  - Kriterien zum Phasenende
  - Passendes Kosten- und Mengengerüst
- Definierte Mitarbeiterfähigkeiten
- Geklärte Rollen, Kompetenzen und Verantwortlichkeiten
- Standardisiertes, wiederholbares Vorgehen



Historie von Lebenszyklusmodellen

- 2001 : Agile Programming
- 1992 : V-Model
- 1988 : Spiral model (Boehm)
- 1983 : Y-Diagramm (Gajski)
- 1977 : Prototyping model (Bally and others)
- 1971 : Incremental model (Mills)
- 1970 : Waterfall model (Royce)
- 1956 : Stagewise model (Bengington)
- 1950 : Code-and-fix model



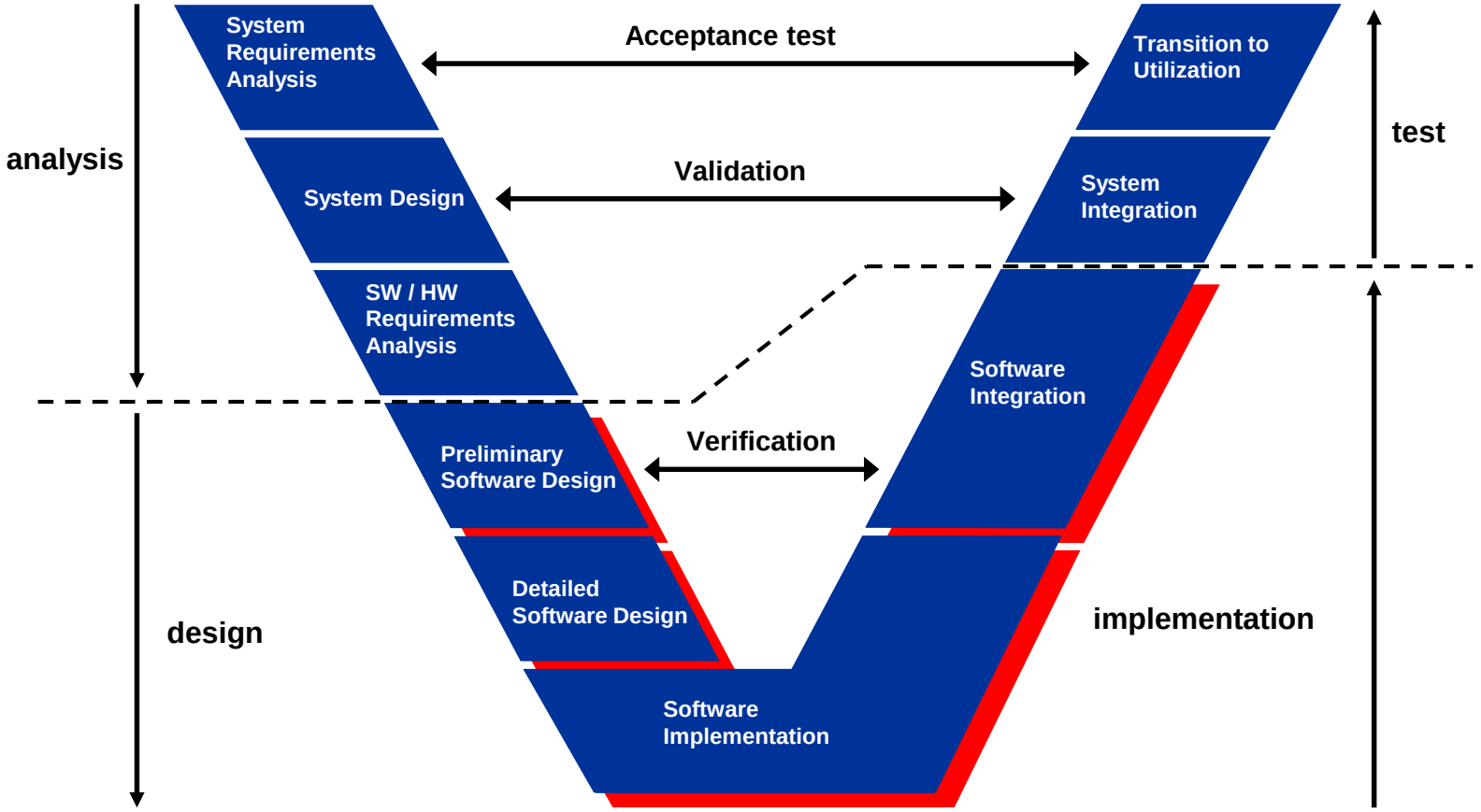
Historie von Lebenszyklusmodellen

- 2001 : Agile Programming
- 1992 : V-Model
- 1988 : Spiral model (Boehm)
- 1983 : Y-Diagramm (Gajski)
- 1977 : Prototyping model (Bally and others)
- 1971 : Incremental model (Mills)
- 1970 : Waterfall model (Royce)
- 1956 : Stagewise model (Bengington)
- 1950 : Code-and-fix model

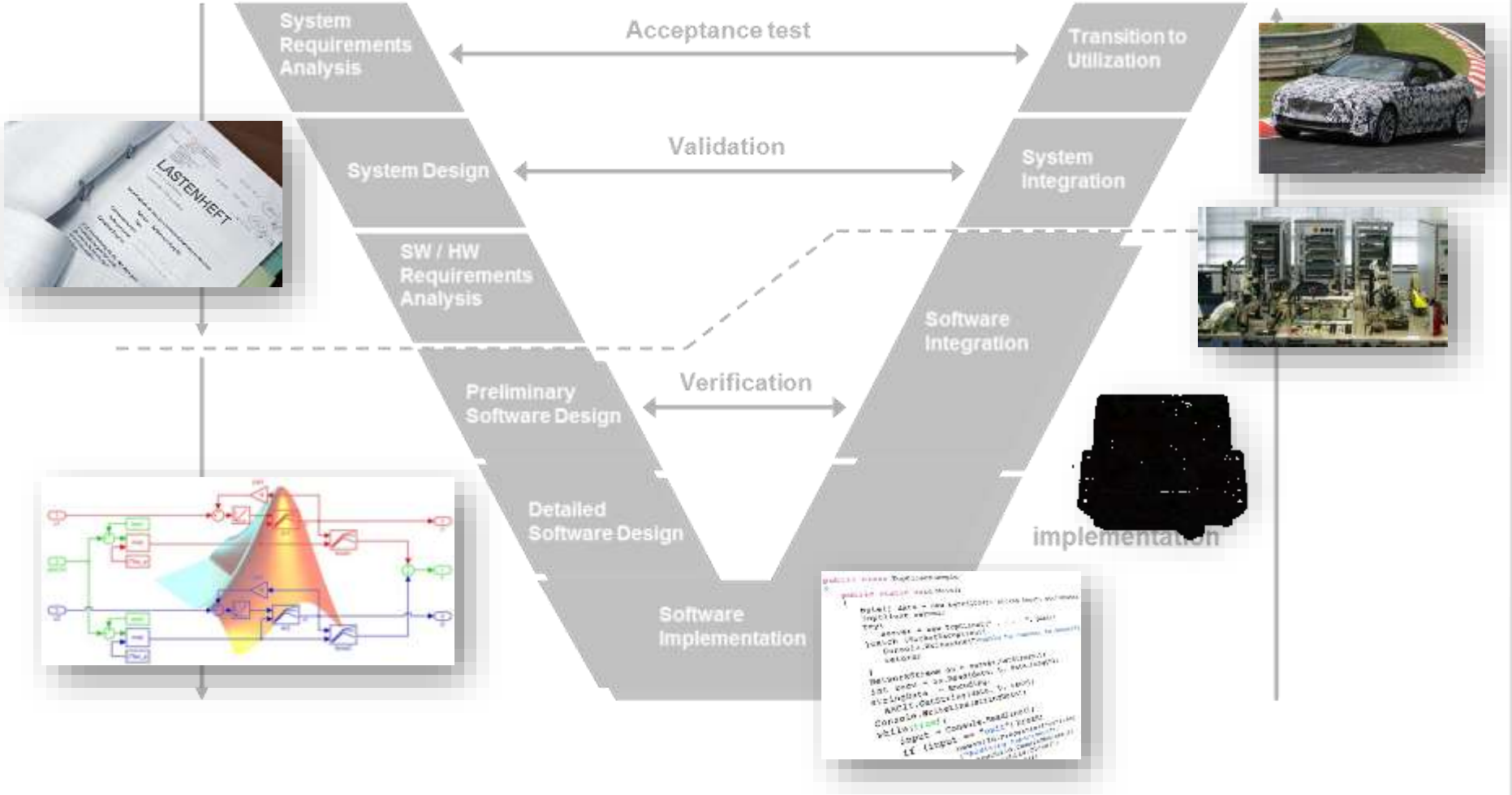
V-Modell Historie (Zsf.)

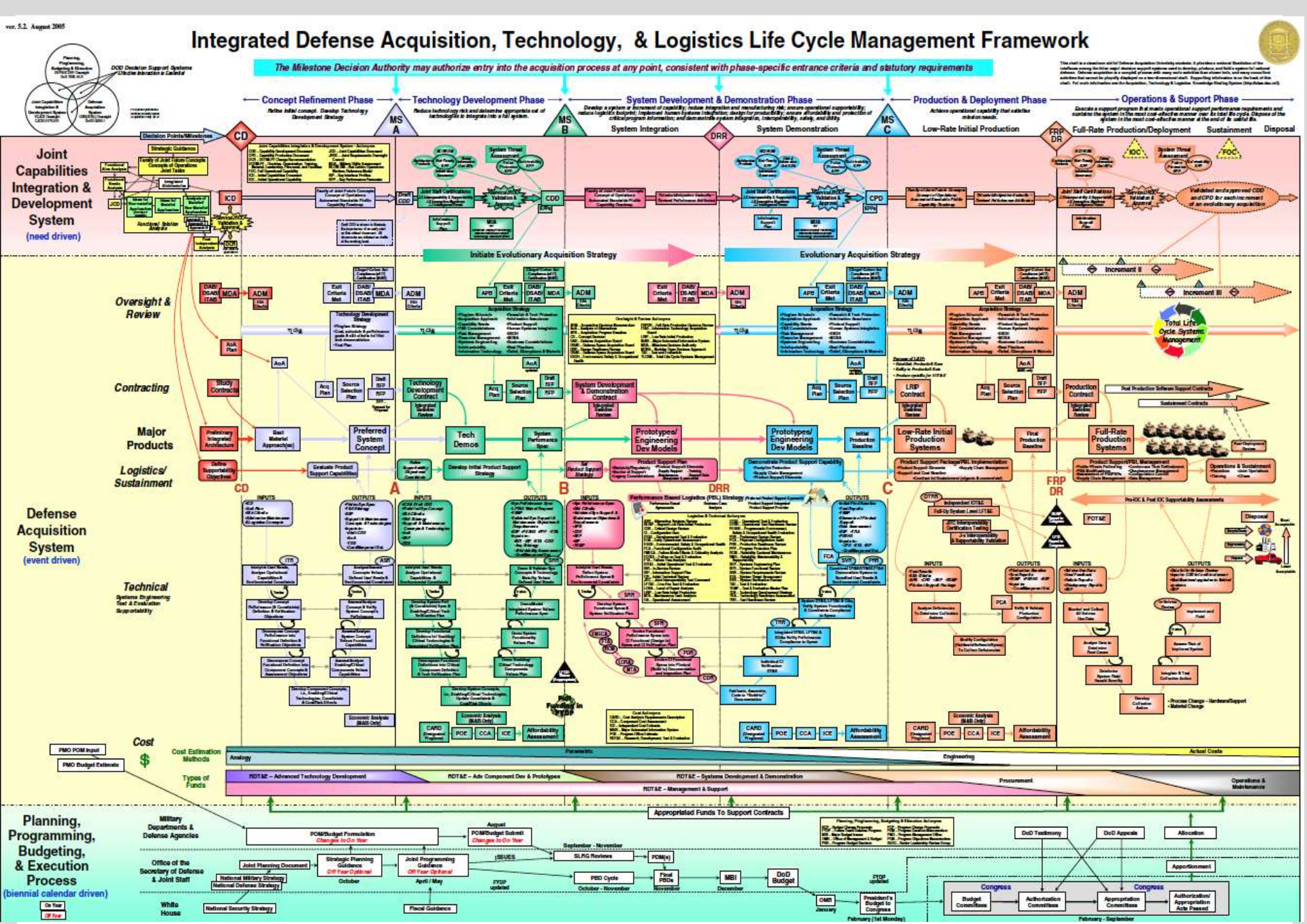
- Das V-Modell
  - Das V-Modell wurde 1992 im Auftrag des Bundesministeriums für Verteidigung entwickelt
- V-Modell 97
  - Überarbeitung u.a. wegen neuer Software-Entwicklungsansätze (z. B. Objektorientierung)
- V-Modell XT
  - Version 1.0 des V-Modell XT (XT = Extreme Tailoring) (2005)

Das V-Model


System Requirements Analysis
System Design
SW / HW Requirements Analysis
Preliminary Software Design
Detailed Software Design
Software Implementation
Software Integration
System Integration
Transition to Utilization
Acceptance test
Validation
Verification
analysis
design
test
implementation

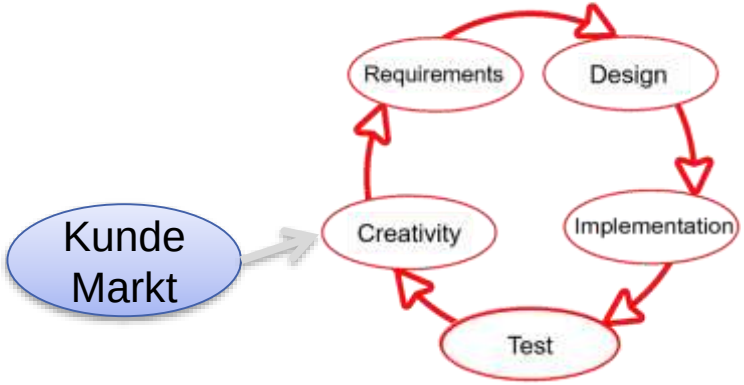
Das V-Model


System Requirements Analysis
Acceptance test
Transition to Utilization
System Design
Validation
System Integration
SW / HW Requirements Analysis
Software Integration
Preliminary Software Design
Verification
Detailed Software Design
Software Implementation
implementation
LASTENHEFT


ver. 5.2. August 2005
Integrated Defense Acquisition, Technology, & Logistics Life Cycle Management Framework
The Milestone Decision Authority may authorize entry into the acquisition process at any point, consistent with phase-specific entrance criteria and statutory requirements
Concept Refinement Phase
Technology Development Phase
System Development & Demonstration Phase
Production & Deployment Phase
Operations & Support Phase
MS A
MS B
MS C
DRR
FRP DR
System Integration
System Demonstration
Low-Rate Initial Production
Full-Rate Production/Deployment
Sustainment
Disposal
Joint Capabilities Integration & Development System
(need driven)
Defense Acquisition System
(event driven)
Planning, Programming, Budgeting, & Execution Process
(biennial calendar driven)
Oversight & Review
Contracting
Major Products
Logistics/ Sustainment
Technical
Cost
Initiate Evolutionary Acquisition Strategy
Evolutionary Acquisition Strategy
Increment II
Increment III
Total Life Cycle Systems Management
Preferred System Concept
Tech Demos
Prototypes/ Engineering Dev Models
Low-Rate Initial Production Systems
Full-Rate Production Systems
Appropriated Funds To Support Contracts
Military Departments & Defense Agencies
Office of the Secretary of Defense & Joint Staff
White House
Congress

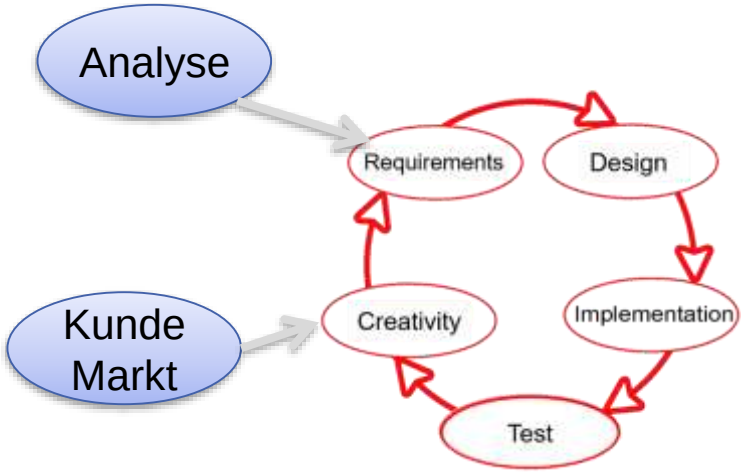
Ideen entwickeln / Bedürfnisse aufnehmen

- Es finden Kundengespräche und Marktanalysen statt.
- Es werden Kreativitätstechniken verwendet.
- Kunde erhält Einschätzung über Umfang und Form der Lösung.
  - Ggf. Testpersonen bei neuen Produkten.

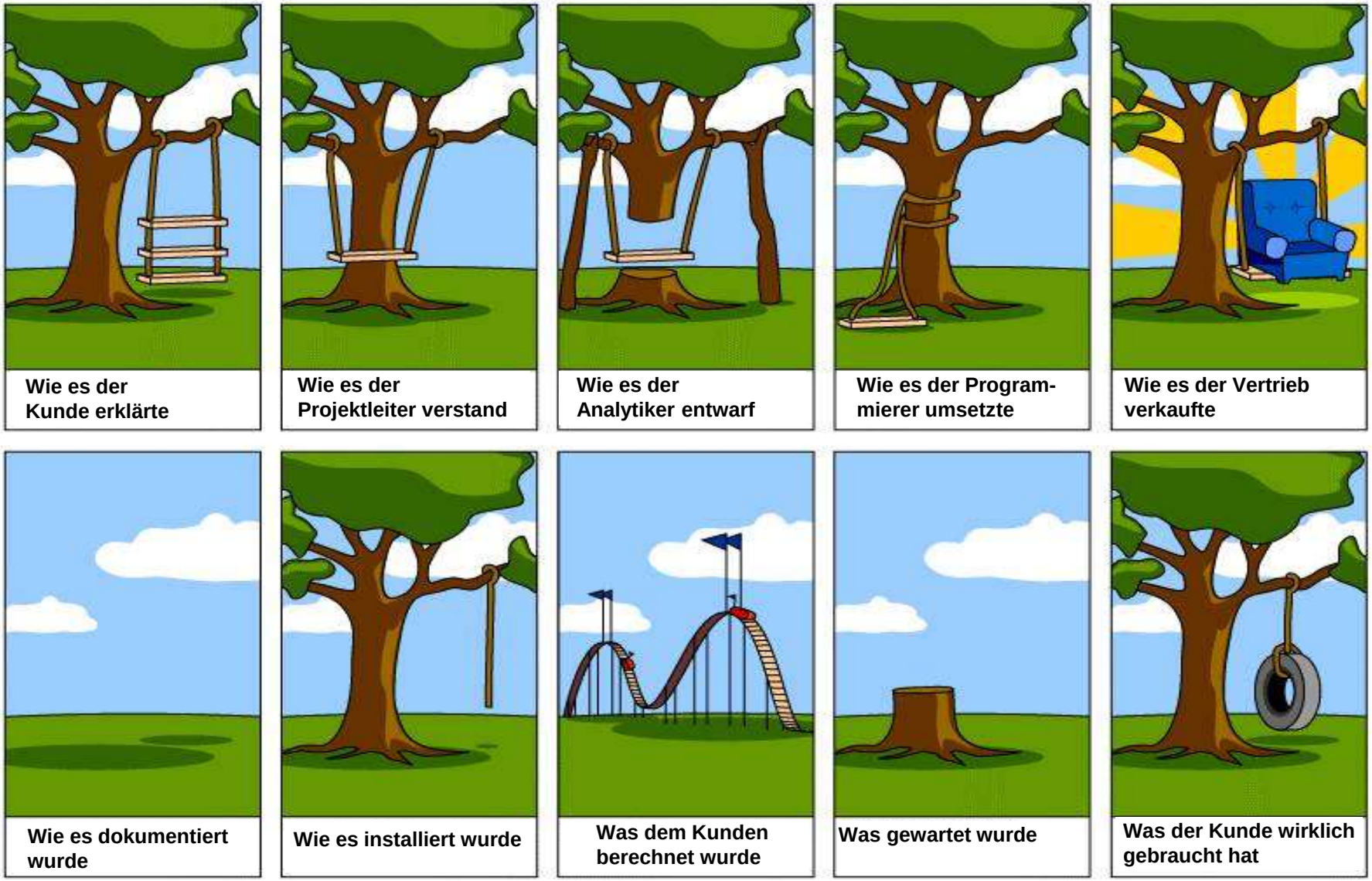


Analyse: Anforderungen fixieren

- Warum braucht der Kunde/Markt diese Lösung?
  - Warum von uns?
- Wo liegen die Herausforderungen bei der Umsetzung?
  - Systematische Erfassung und Analyse der Anforderunge
  - siehe Vorlesung “Systems and Software Engineering”
- Welche Ressourcen benötigt man?
- Der Bedarf wird festgestellt
  - Lastenheft
- Festhalten von (schriftlich! da Vertragsgrundlage):
  - Pflichtenheft
  - Zielvereinbarungen
- Beschreibung des Problems (Analyse)

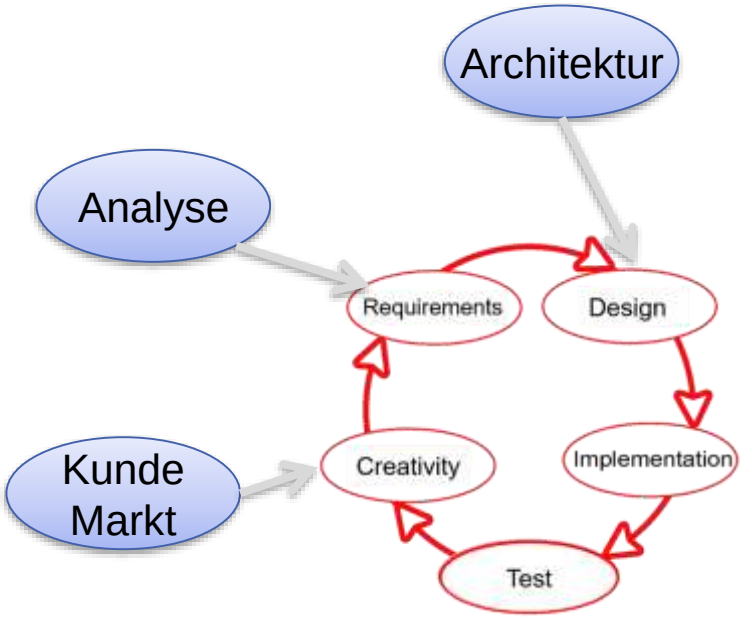


Konkrete Anforderungen …



Design

- Wie lässt sich das Problem gliedern?
- Wie sehen die Schnittstellen aus
  - … zu bestehender Software beim Kunden
  - … zu Hardware
  - … zu anderen Programmteilen (untereinander)
- Modularisierung, Entwurf für und mit Wiederverwendung
- Welche Bibliotheken können eingesetzt werden
- Was lässt sich kostengünstiger/besser extern einkaufen
- Passt das Design zum Problem? → Verfeinerung

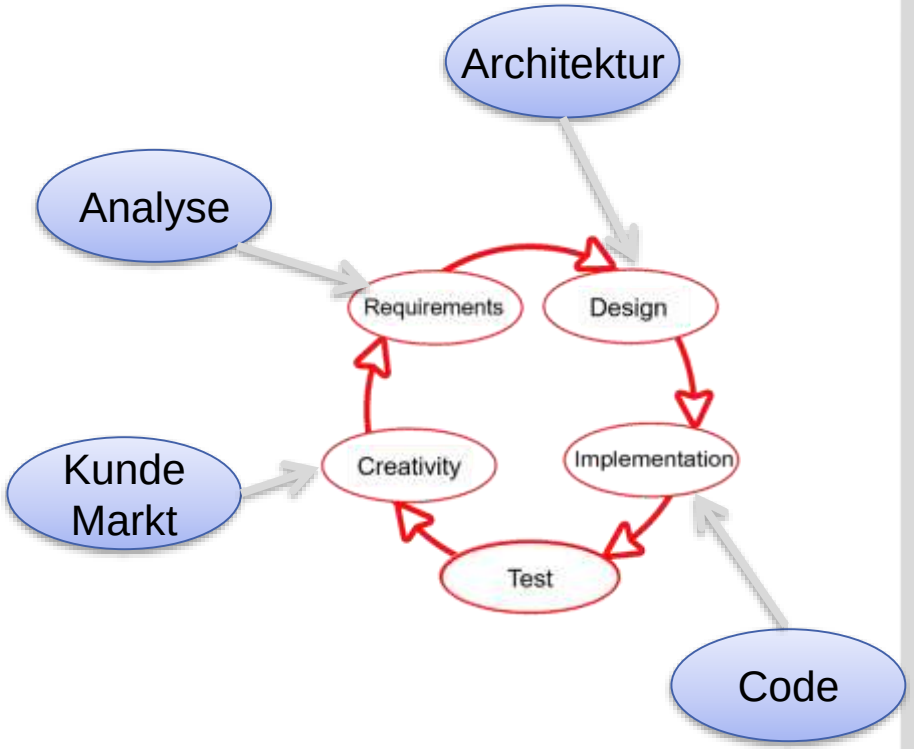


→ Architektur

Umsetzung

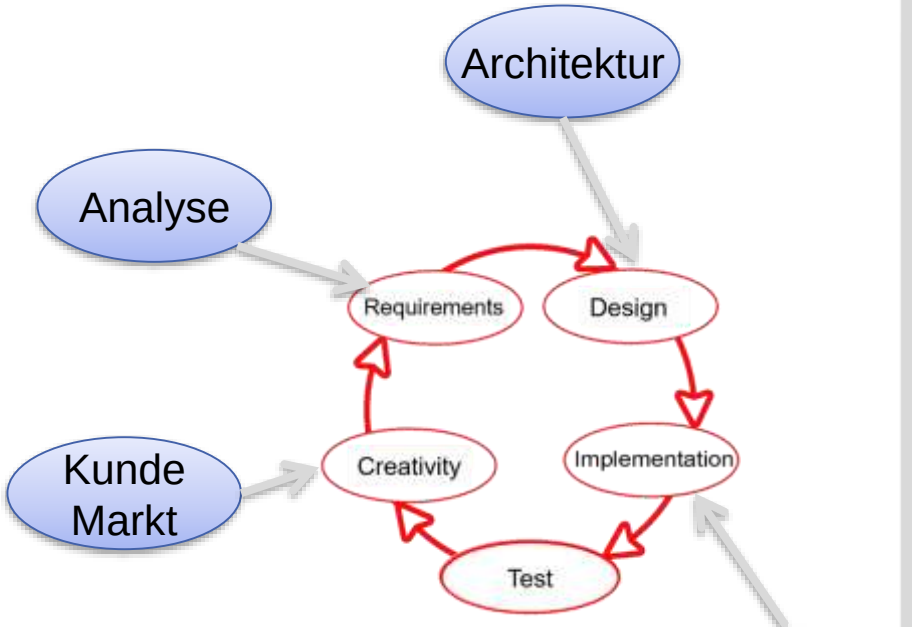
- Programmierer setzen den ihnen zugeteilten Systemteil um.
- Aufgabe ergibt sich aus Design, Beschreibung des Problems.
- Verifikation, dass Design eingehalten wird
- Unter Umständen: Anpassung des Designs

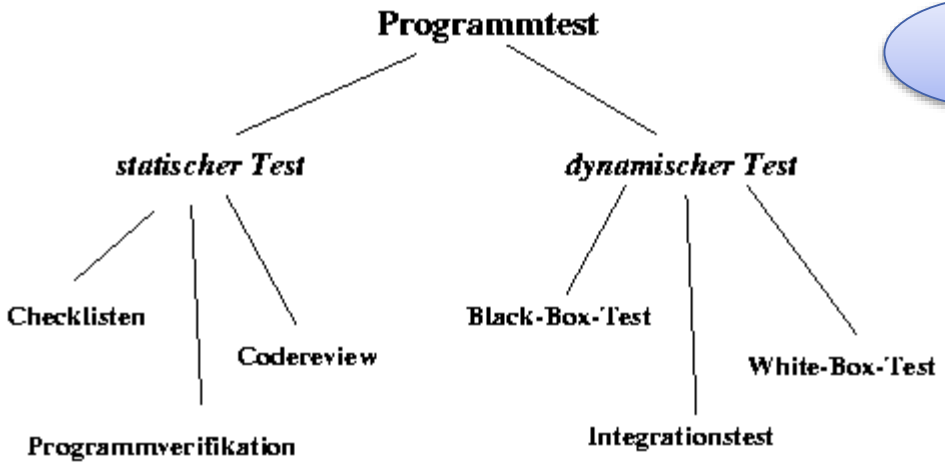
➡ Code



Testen

- Fehler (Bugs) können sich immer einschleichen:
  - menschliches Versagen
  - widersprüchliche Anforderungen / Design
  - nicht spezifikationsgemäßes Verhalten von Software aus Bibliotheken, Hardware, anderen Komponenten
- Notwendig für Stabilität
- Erheblicher Zeitaufwand





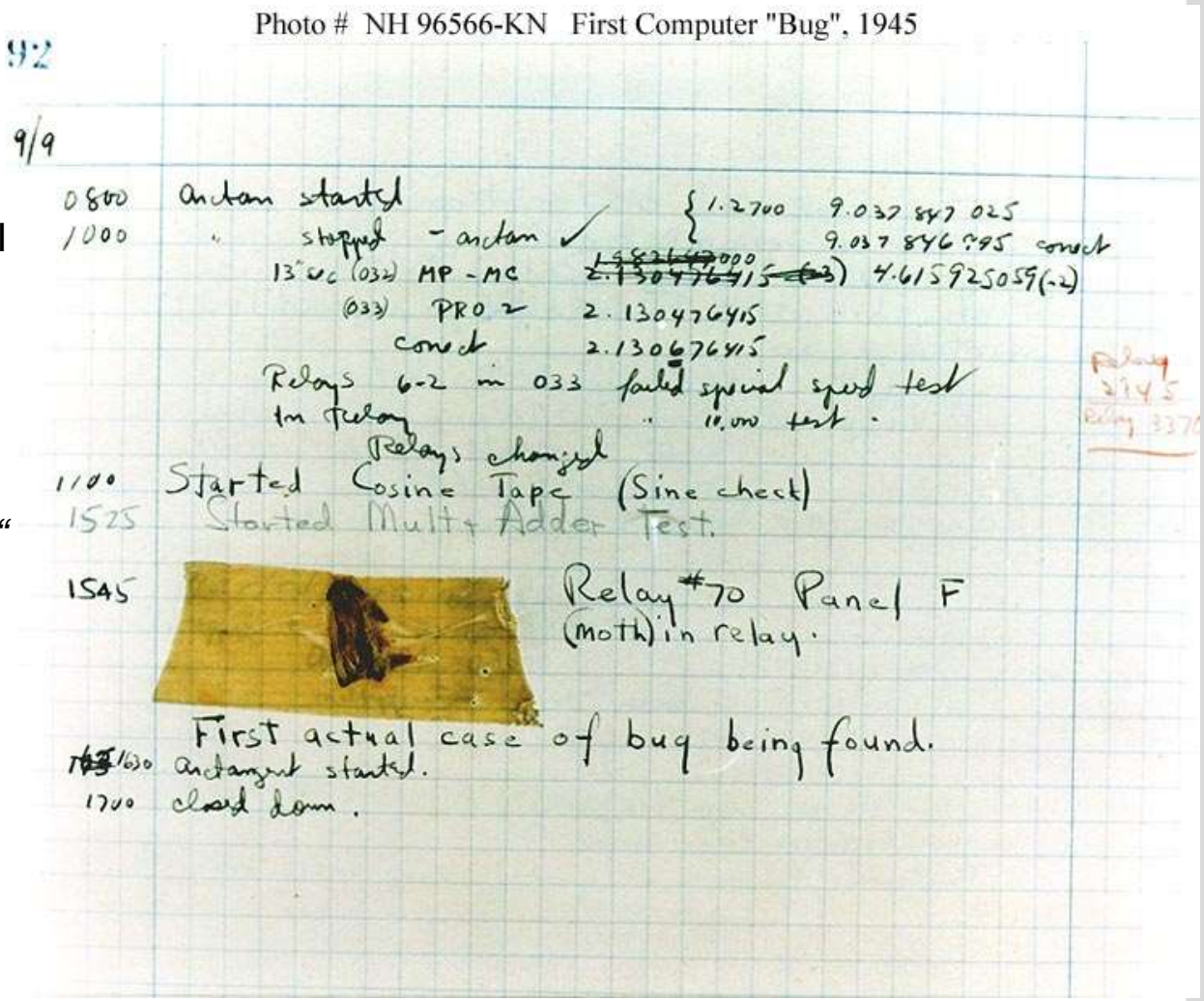
Was ist Testen?

- Testen ist das Vorgehen eine größere Sicherheit zu erlangen, ob ein technischer Apparat oder Prozess innerhalb der vorgegebenen Randbedingungen gemäß der Anforderungen funktioniert oder nicht.
- Beim Testen werden vordefinierte Antworten erwartet. Damit unterscheidet es sich vom Experiment, wo die Ergebnisse offen sind.
- Vom Prüfen unterscheidet sich das Testen insofern, dass beim Prüfen relative Aussagen getroffen werden, wie gut die Anforderungen erfüllt sind. Beim Testen nicht → geht oder geht nicht
- In Extremfällen kann ein Test auch einem Beweis gleich kommen.

Debuggen

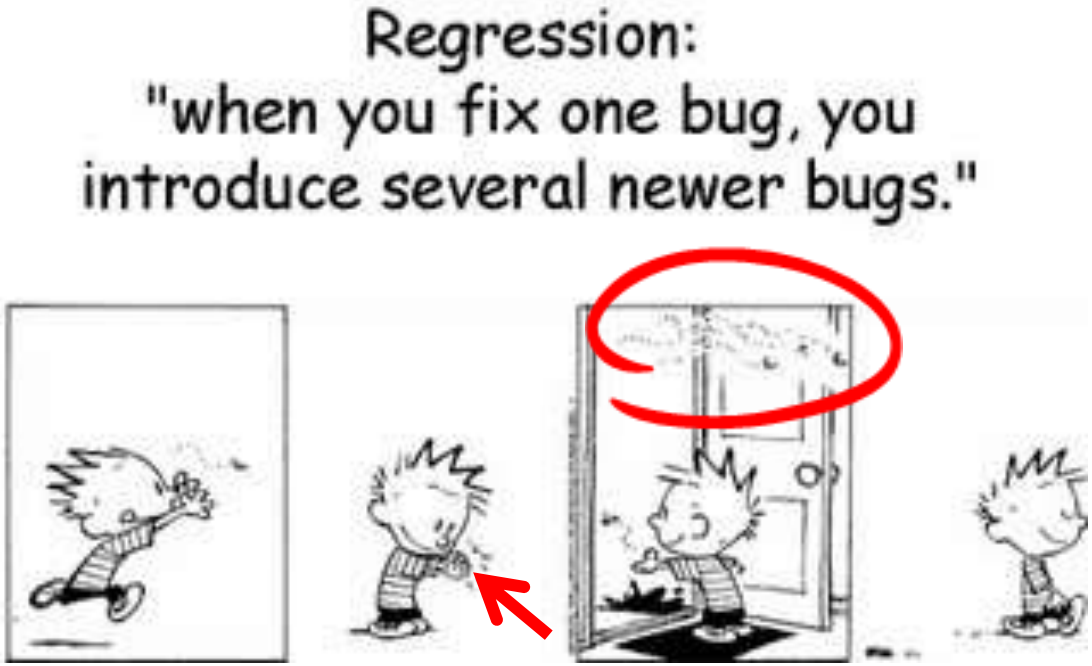
Der Begriff „debugging“ (*entwanzen*) basiert auf der von Grace Hopper eingeführten Bezeichnung für Fehler in Computersystemen. 1947 hatte während der Arbeiten am MarkII eine Motte für den Ausfall eines Relais dieses Computers gesorgt. Grace Hopper hat die (tote) Motte in das Logbuch geklebt und mit dem Satz *„First actual case of bug being found.“* kommentiert.

Remark: *Der Begriff „Bug“ (für Insekt, Käfer, Schädling) war im Englischen unter Ingenieuren bereits seit längerer Zeit als Bezeichnung für Fehlfunktionen in Gebrauch.*



Was ist ein Bug?

- Ein **Programmfehler** (**Bug**) bezeichnet ein Fehlverhalten von Computer-Programmen.
- Tritt auf, wenn der Programmierer eine bestimmte Festlegung der Spezifikation nicht oder falsch umgesetzt hat, oder wenn die Laufzeitumgebung fehlerhaft bzw. anders als erwartet arbeitet.
- Weiterhin können auch Unvollständigkeit, Ungenauigkeit oder Mehrdeutigkeiten in der Spezifikation des Programms zu Fehlern führen.



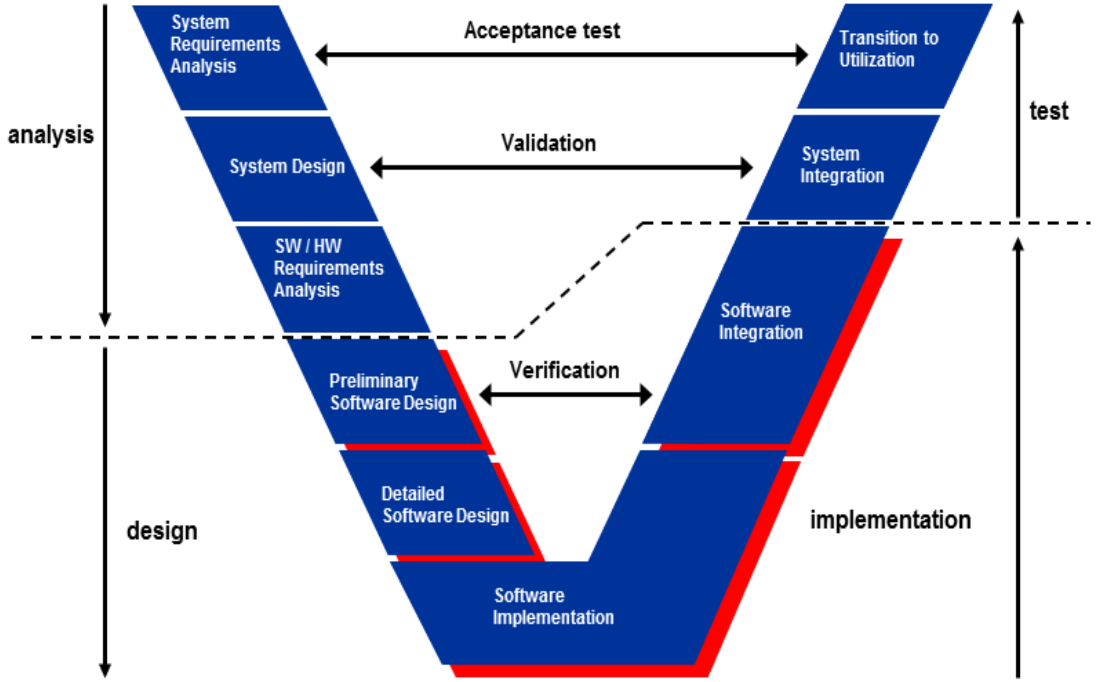
Verifikation und Validierung

Verifikation

- **Baue ich das Produkt richtig?**

Validierung

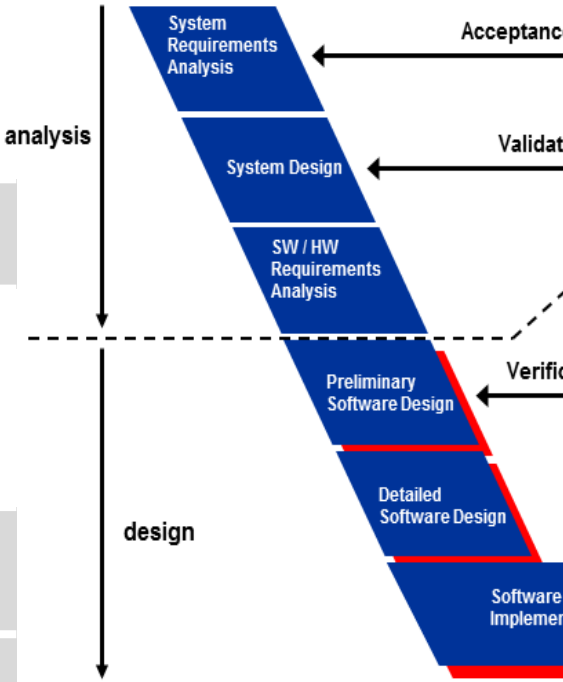
- **Baue ich das richtige Produkt?**



Überblick über die Absicherungsphasen

Dokumente im Entwurfsablauf

Gesamt-system	Sub-system	Funktionen	Kompo-nenten
Vehicle-“Rahmenheft“			
Def. of. architecture			
	Systems-“Lastenheft“		
	Systems-modeling	Functional-“Lastenheft“	„Component“-Lastenheft
	Systems-Test-Spec.	Application-modeling	
		Rapid-Prototyping	
		Production code-generation	Software-Integration
		Functional Test-Spec.	Component Test-Spec.

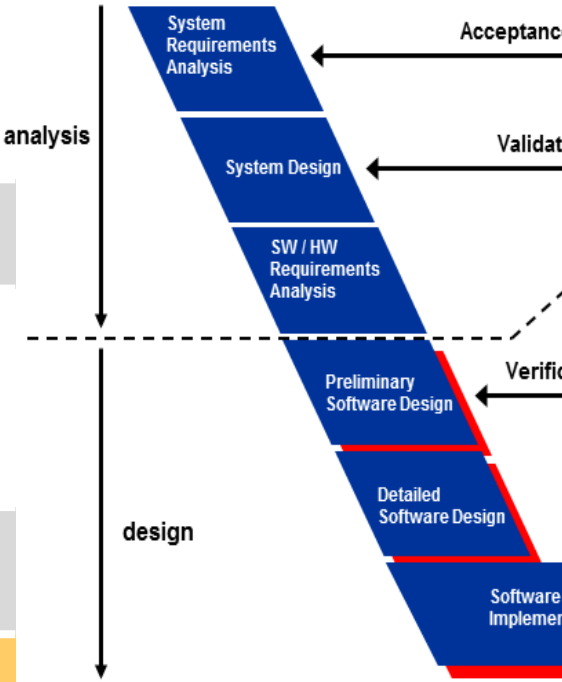


Überblick über die Absicherungsphasen

Dokumente im Entwurfsablauf

Gesamt-system	Sub-system	Funktionen	Kompo-nenten
Vehicle-“Rahmenheft“			
Def. of. architecture			
	Systems-“Lastenheft“		
	Systems-modeling	Functional-“Lastenheft“	„Component“-Lastenheft
	Systems-Test-Spec.	Application-modeling	
		Rapid-Prototyping	
		Production code-generation	Software-Integration
		Functional Test-Spec.	Component Test-Spec.

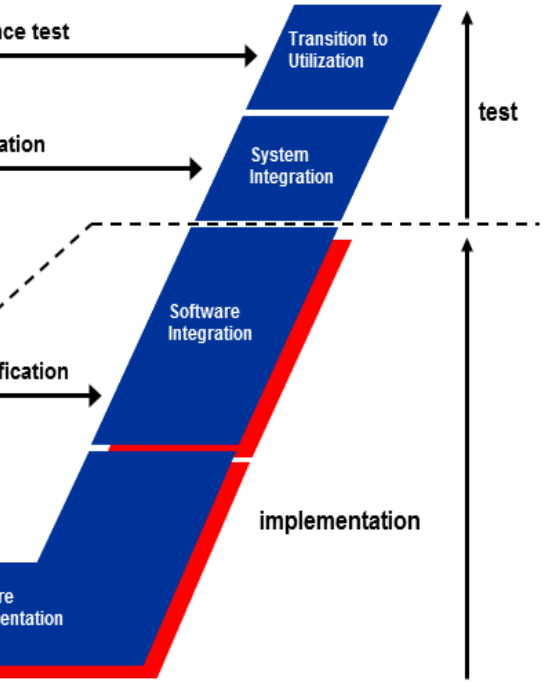
Dokument



Überblick über die Absicherungsphasen

Dokumente im Entwurfsablauf

Kompo-nenten → **Funktionen** → **Sub-System** → **Gesamt-System**



Unit-Test

Diagnosis Tests

Conformance-tests

Module-Tests

Funktional-tests

Sub-Systemtest

Systemtest

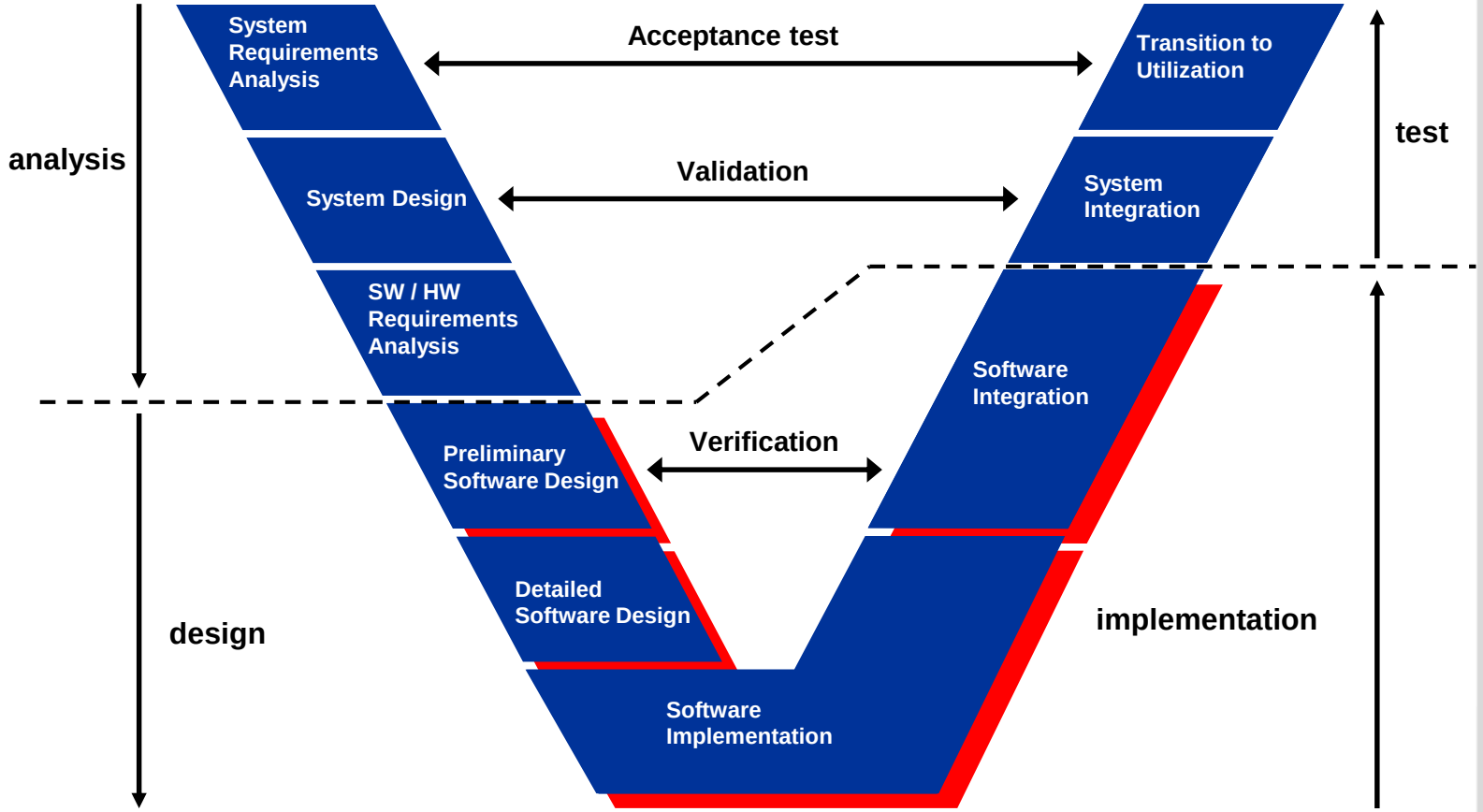
Akzeptanztest

Electrical-systems test

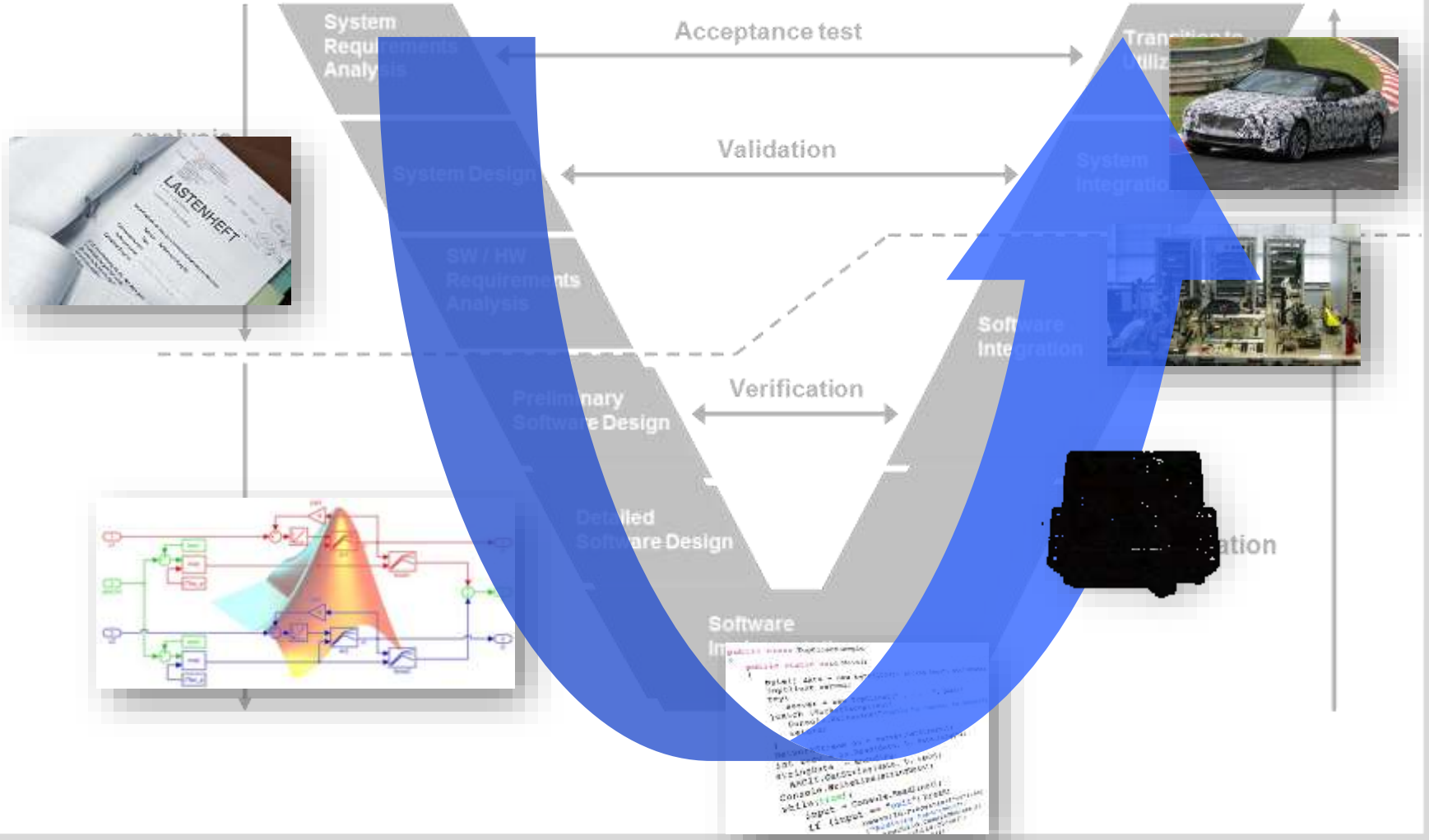
Test-Aktivitäten

- Unit-Test
  - Einzelne Komponenten werden isoliert auf spezifikationsgerechte Funktionsweise geprüft (z.B. einzelne Funktionen).
- Module-Test
  - Verbundene Komponenten werden in ihrem internen Zusammenspiel geprüft (z.B. Klassen).
- Subsystemtest
  - Verknüpfte Module werden auf Interoperabilität geprüft. Hierbei treten häufig Probleme wegen nicht übereinstimmender Schnittstellen zutage.
- Funktionstest
  - Die verteilte Funktion wird auf ihr korrektes Verhalten hin getestet.
- Systemtest
  - Subsysteme werden integriert und machen nun das Gesamtsystem aus. Es geht darum, Fehler im Zusammenspiel zwischen (abstraktem) Gesamtsystem und seinen Subsystemen zu finden; außerdem wird validiert, ob das System die Anforderungen des Kunden erfüllt.
- Akzeptanztest:
  - Das finale Produkt wird gegen die initialen Anforderungen validiert.

Durchgängiges Testen



Durchgängiges Testen



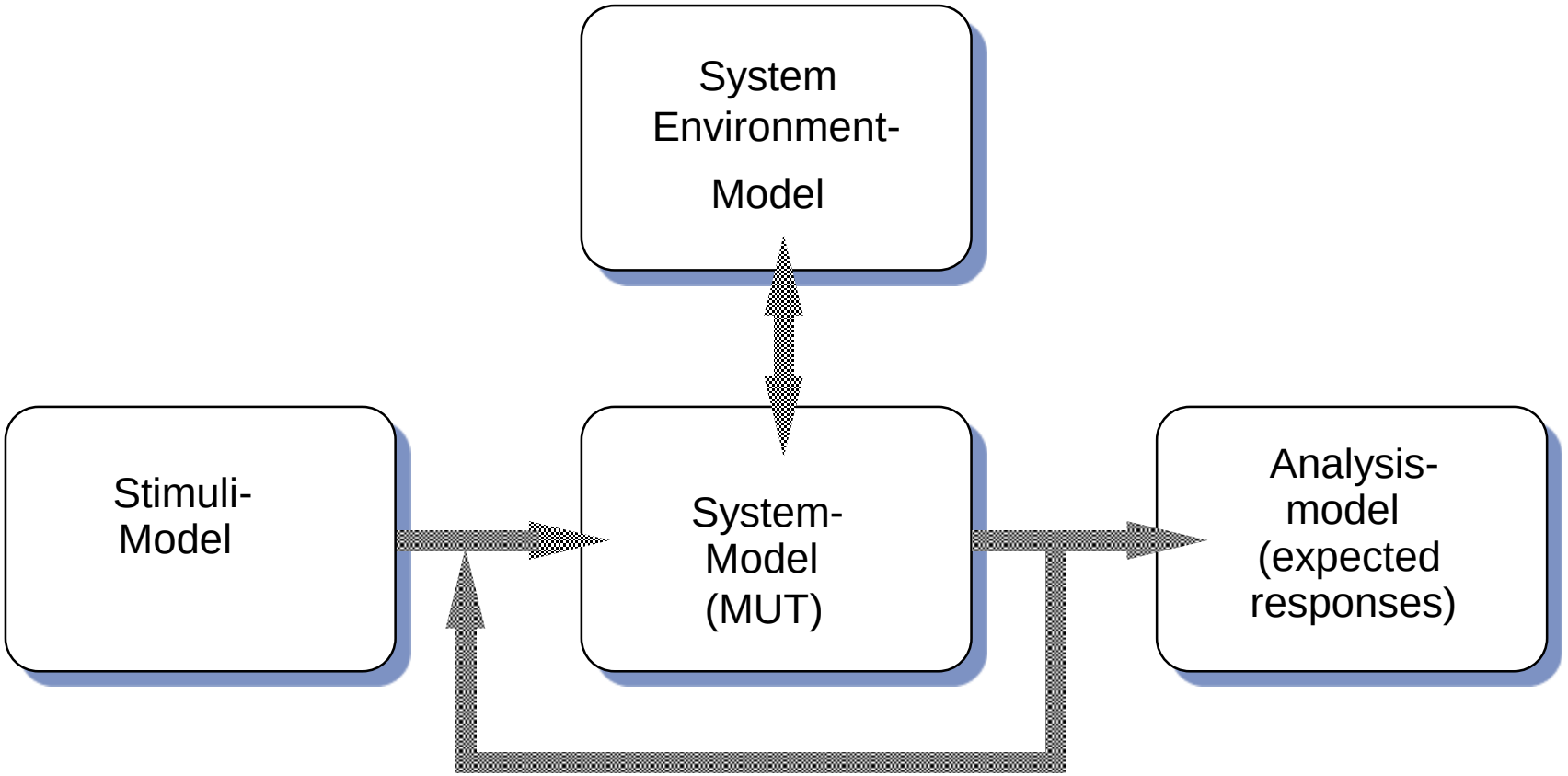
Test-Prinzipien

- Testen ist ein Prozess
- Testen erfordert Disziplin – intern und im Zusammenspiel mit den Partnern
- Testen hat auch etwas mit Vertrauen zu tun
- „Tools follow Process“
- 4-Augenprinzip: Entwickeln und Testten sind 2 unterschiedliche Rollen
- Qualität kann nicht hineingetestet werden

- Anforderungen
  - Klare Spezifikationen
  - Komplette Dokumentation
  - Definierte Prozessschnittstellen
  - Definierte Rollen (Aufgaben, Kompetenzen, Verantwortlichkeiten)

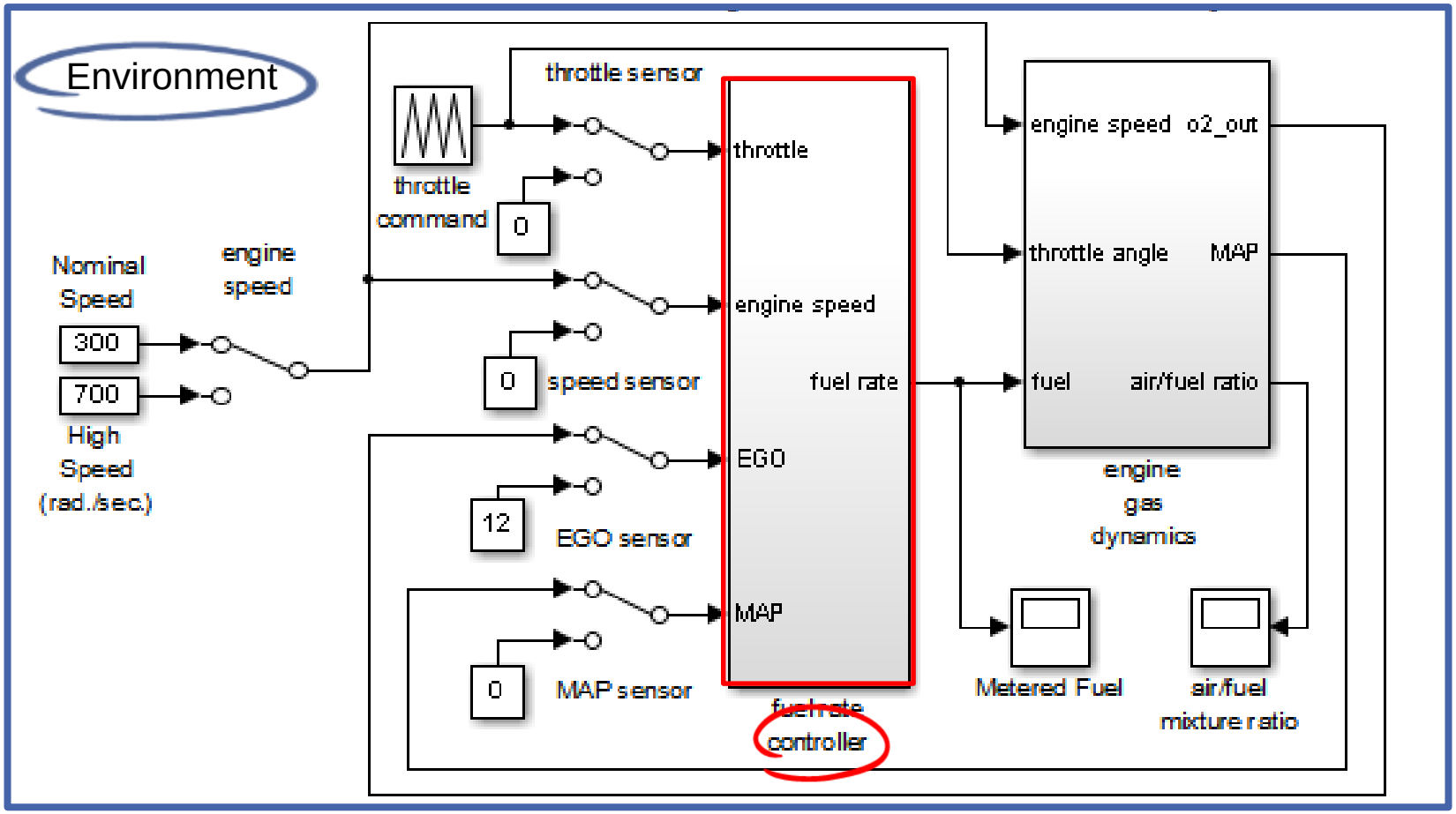
- Das kann auditiert werden!

Testbench Idee: X-in-the-loop

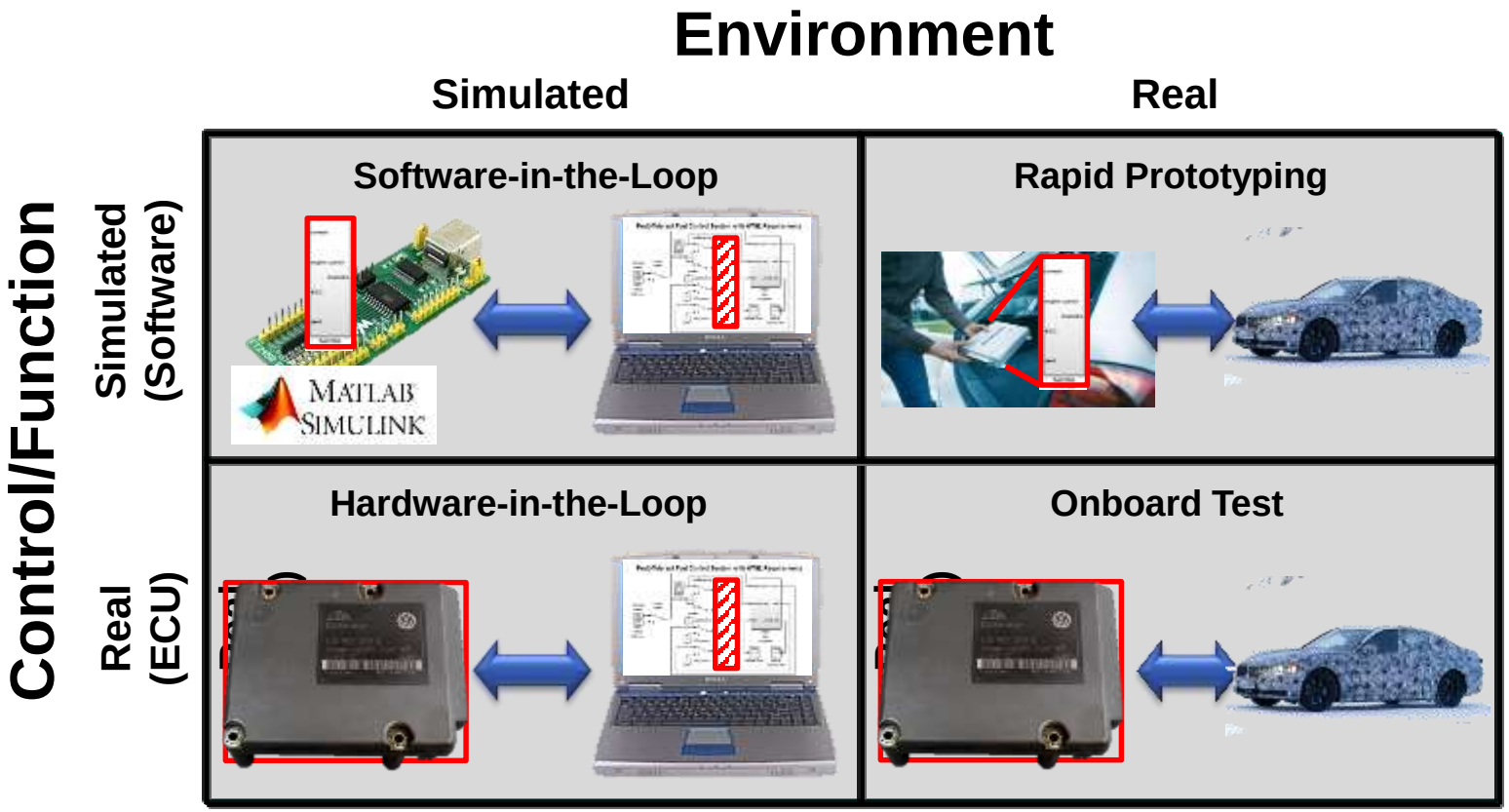


Beispiel

- Kraftstoff-Regelung

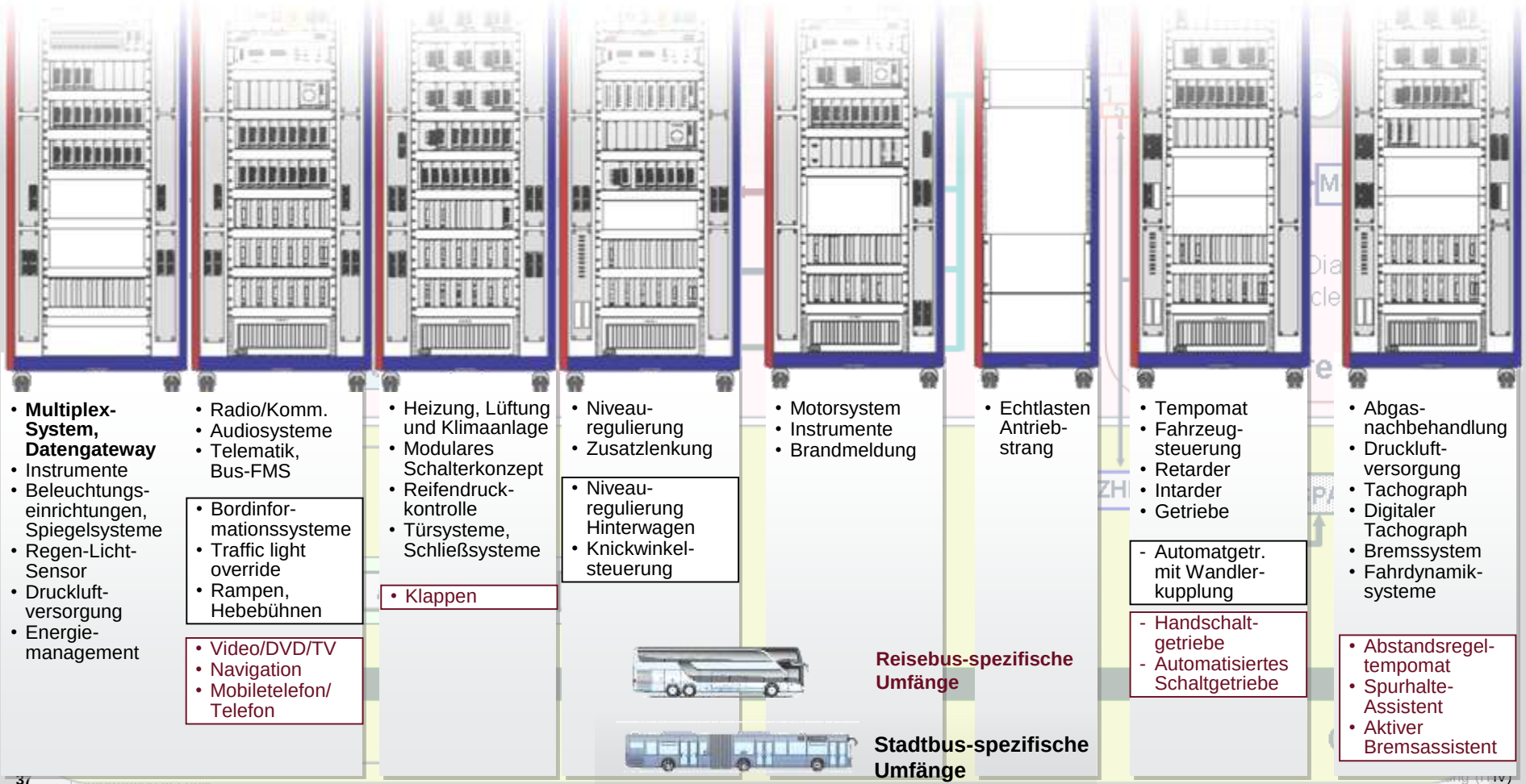


X-in-the-Loop


Environment
Simulated
Real
Control/Function
Simulated (Software)
Real (ECU)
Software-in-the-Loop
Rapid Prototyping
Hardware-in-the-Loop
Onboard Test
MATLAB
SIMULINK

Hardware-in-the-Loop (HiL):
Auslegung und Konzeption

- Die einzelnen SG werden via Switch-Matrix den jeweiligen CAN-Bussen in unterschiedlichen Fahrzeugkonfigurationen zugeordnet

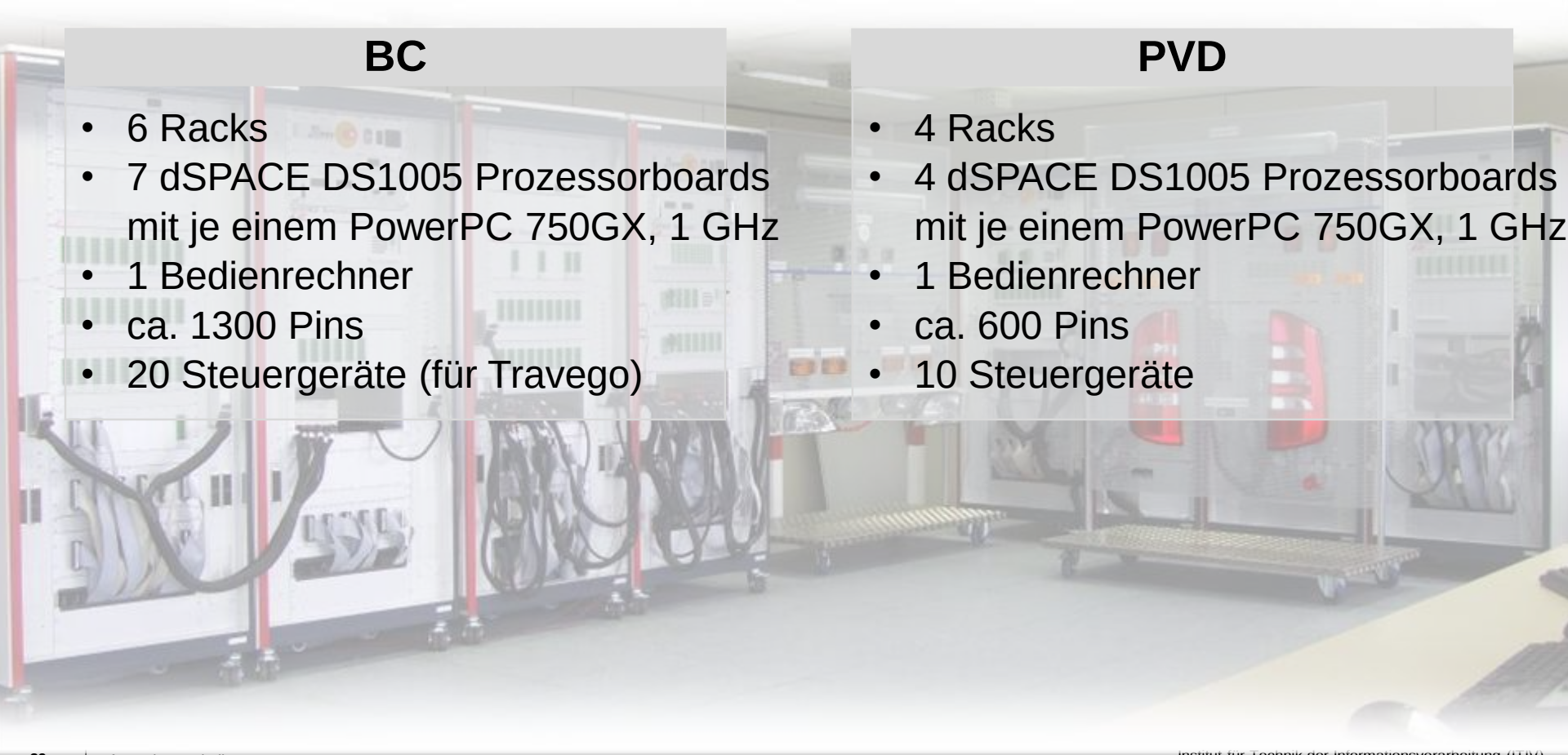


Hardware-in-the-Loop (HiL): Zahlen, Daten, Fakten

Der Prüfstand ist in die Domänen Body-Chassis (BC) und Powertrain-VehicleDynamics (PVD) aufgeteilt

Hardware-in-the-Loop (HiL): Zahlen, Daten, Fakten

Der Prüfstand ist in die Domänen Body-Chassis (BC) und Powertrain-VehicleDynamics (PVD) aufgeteilt



BC

- 6 Racks
- 7 dSPACE DS1005 Prozessorboards mit je einem PowerPC 750GX, 1 GHz
- 1 Bedienrechner
- ca. 1300 Pins
- 20 Steuergeräte (für Travego)

PVD

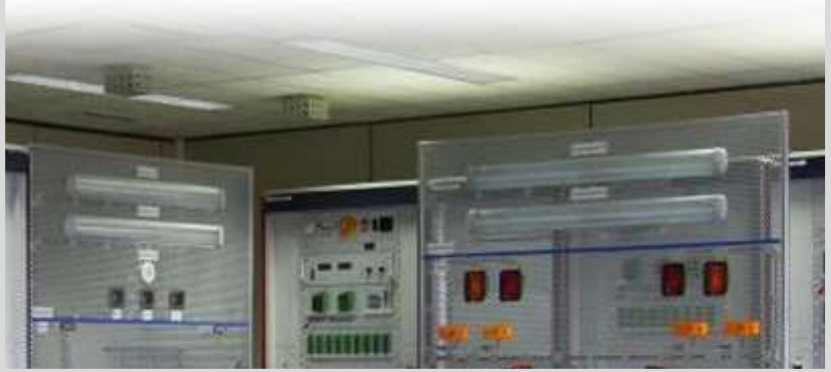
- 4 Racks
- 4 dSPACE DS1005 Prozessorboards mit je einem PowerPC 750GX, 1 GHz
- 1 Bedienrechner
- ca. 600 Pins
- 10 Steuergeräte

Hardware-in-the-Loop (HiL): Komponententräger

Ergänzung der Teilsysteme des Integrationsprüfstandes

z.B. Druckluftaufbau der Bremse

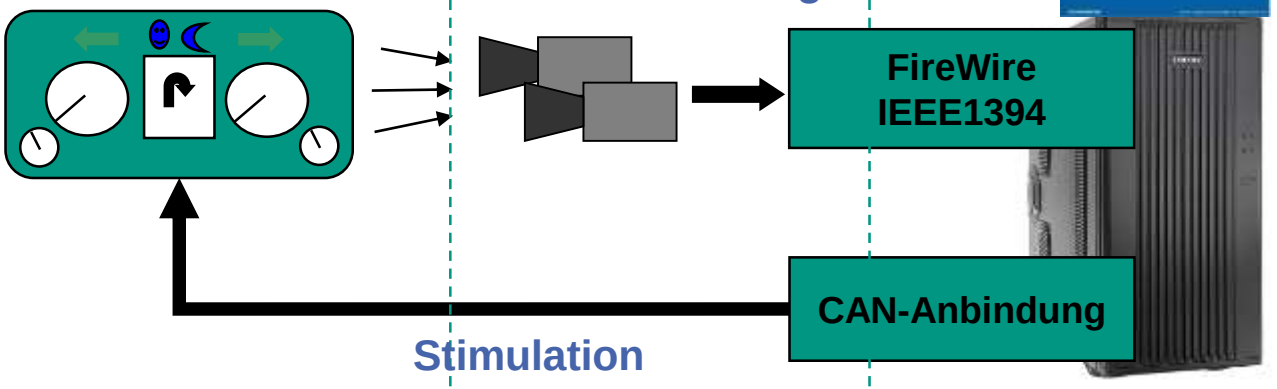
z.B. Beleuchtungseinrichtungen



z.B. Cluster-Auswe

Hardware-in-the-Loop (HiL):

Besonderheiten Bildverarbeitung für den vollautomatisierten Cluster-Test



PROVEtech:TA

Device under Test

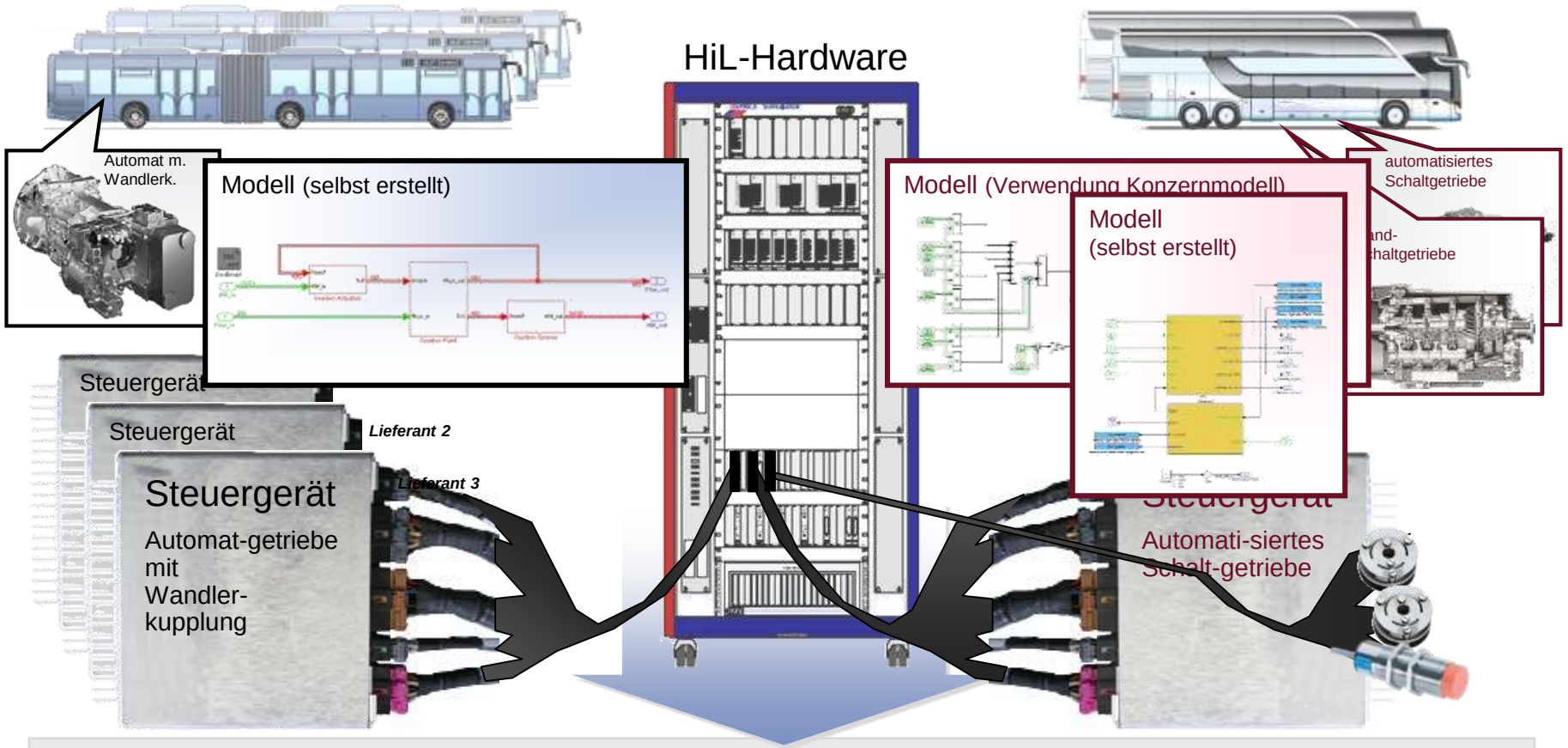
Bild-aufnahme

HW-Schnittstellen & Bilderkennung

Test-automatisierung

Hardware-in-the-Loop (HiL): Fahrzeugvarianten

Prüfstand variabel anpassbar – Beispiel Getriebevarianten

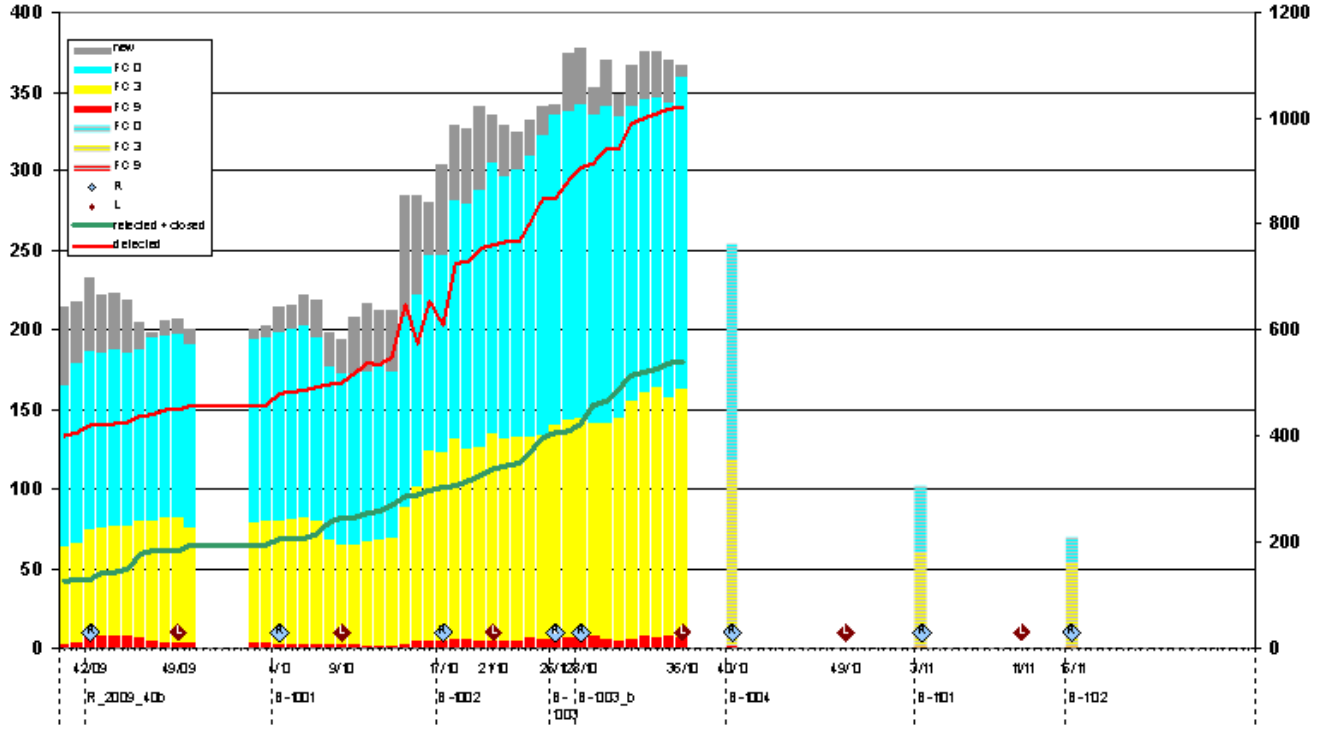


HiL-Prüfstand ermöglicht alternative Darstellung der Fahrzeugvarianten aus dem Baukasten

- 3 Getriebearten (Handschalter, automatisiertes Schaltgetriebe und Stufenautomat)
- Konzerneigene Getriebe plus unterschiedliche Lieferanten

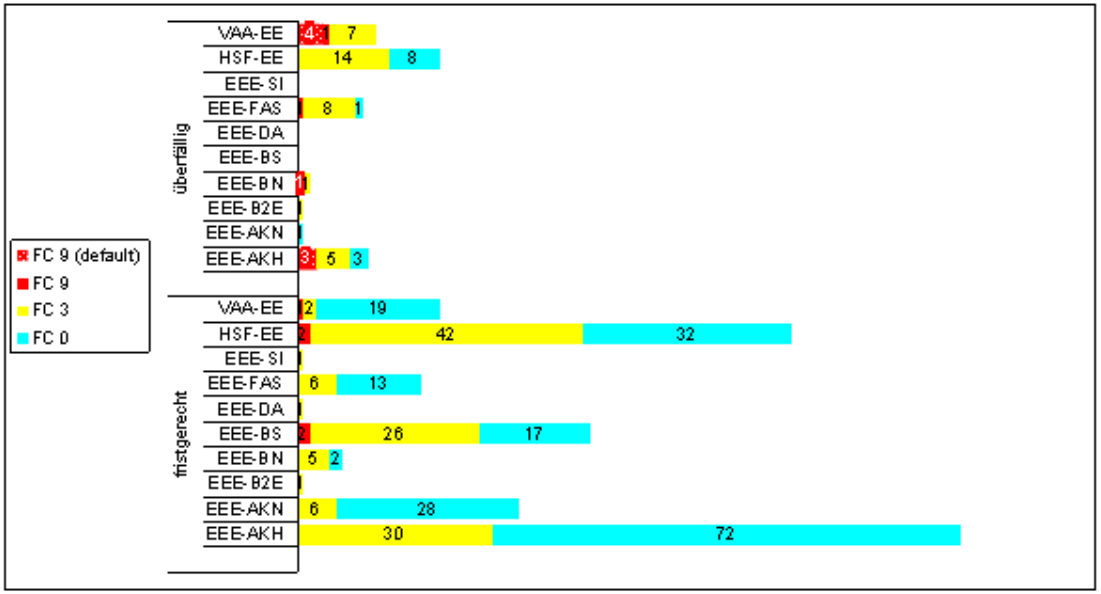
Konsequentes Verfolgen der Issues

- Wöchentlich:
  - Gesamt-Issueverlaufs und teambezogene detaillierte Issues
  - (Metrik: Fehlerklassen und Laufzeiten) an die zuständigen Entwickler und Vorgesetzter



Konsequentes Verfolgen der Issues

- Wöchentlich:
  - Gesamt-Issueverlaufs und teambezogene detaillierte Issues
  - (Metrik: Fehlerklassen und Laufzeiten) an die zuständigen Entwickler und Vorgesetzten

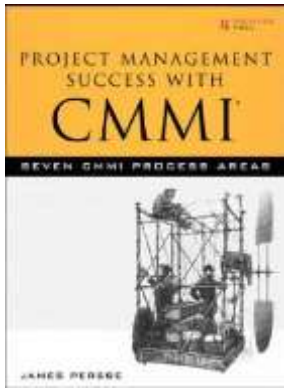


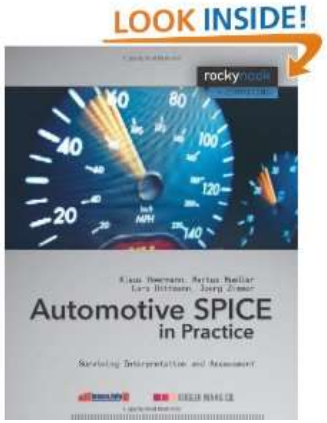
Regeln zur Klassifikation als "überfällig"

Laufzeit	Fehlerklasse	Issue Status	Planned fixing release	Ergebnis
>3 Werktage	FC 9 **(default)**	new	-	überfällig
>6 Werktage	FC 9	open	keine Angabe	überfällig
>13 Werktage	FC 3	open	keine Angabe	überfällig
>13 Werktage	FC 0	open	keine Angabe	überfällig
-	-	open	< aktuelles Release	überfällig

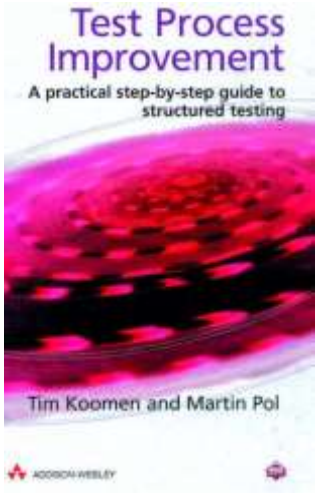
Bestehende Reifegradmodelle

- CMMI/SPICE
  - Verification and Validation are not entirely adressed
  - CMMI® mainly only reviews the documents but does not care about „how“
  - (Automotive) SPICE demands strategic and generic approach
  - Both models do not adress test equipment and planing sufficiently





- Test Process Improvement (TPI®)
  - Combination with existing assessment critical
  - TPI® Automotive defines no reference process
  - To much open and flexible issues to make a comparison of different methods

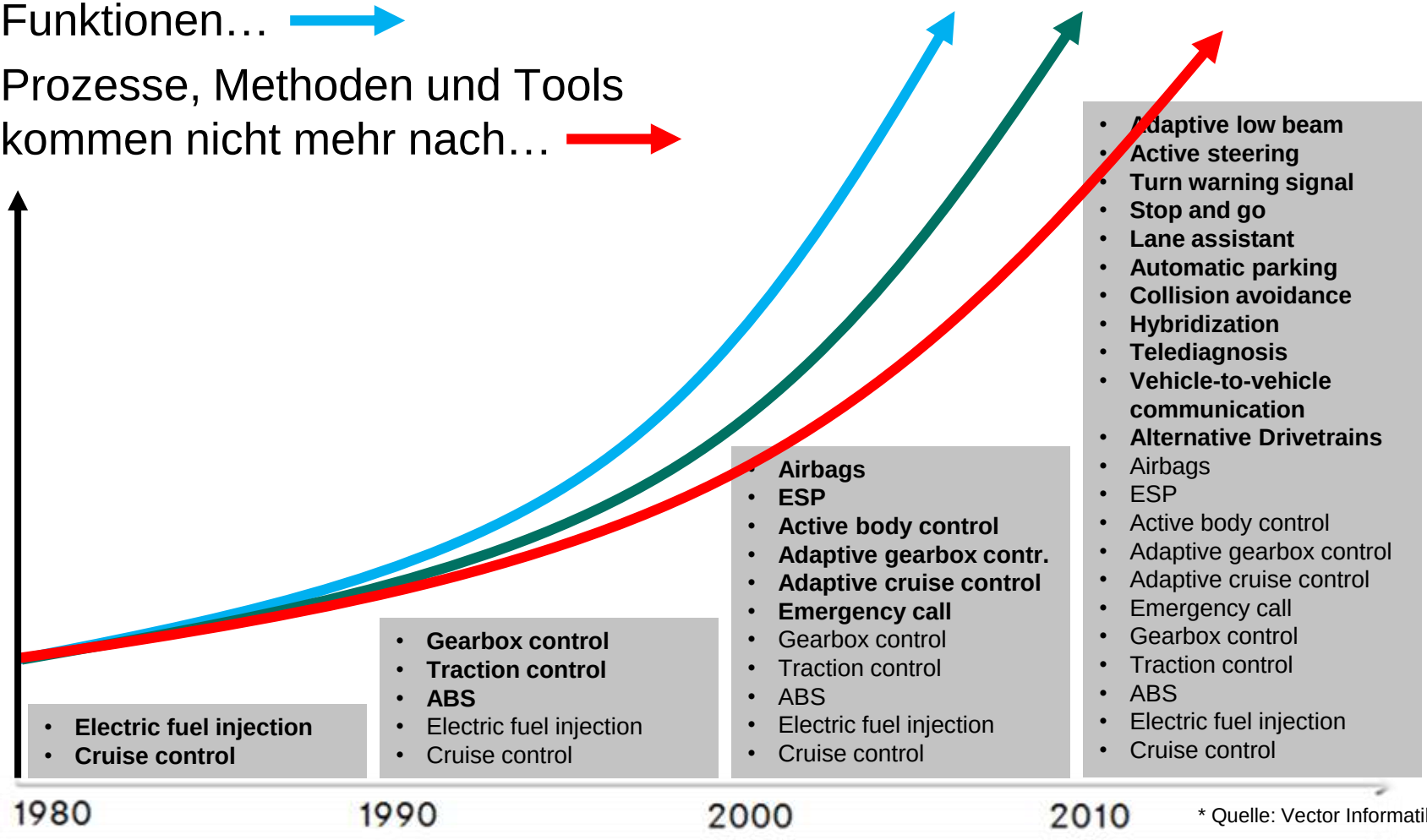



Hardware in the loop
Zur fachlichen Freigabe – nur zum internen Gebrauch.

Wachsende Funktionalität

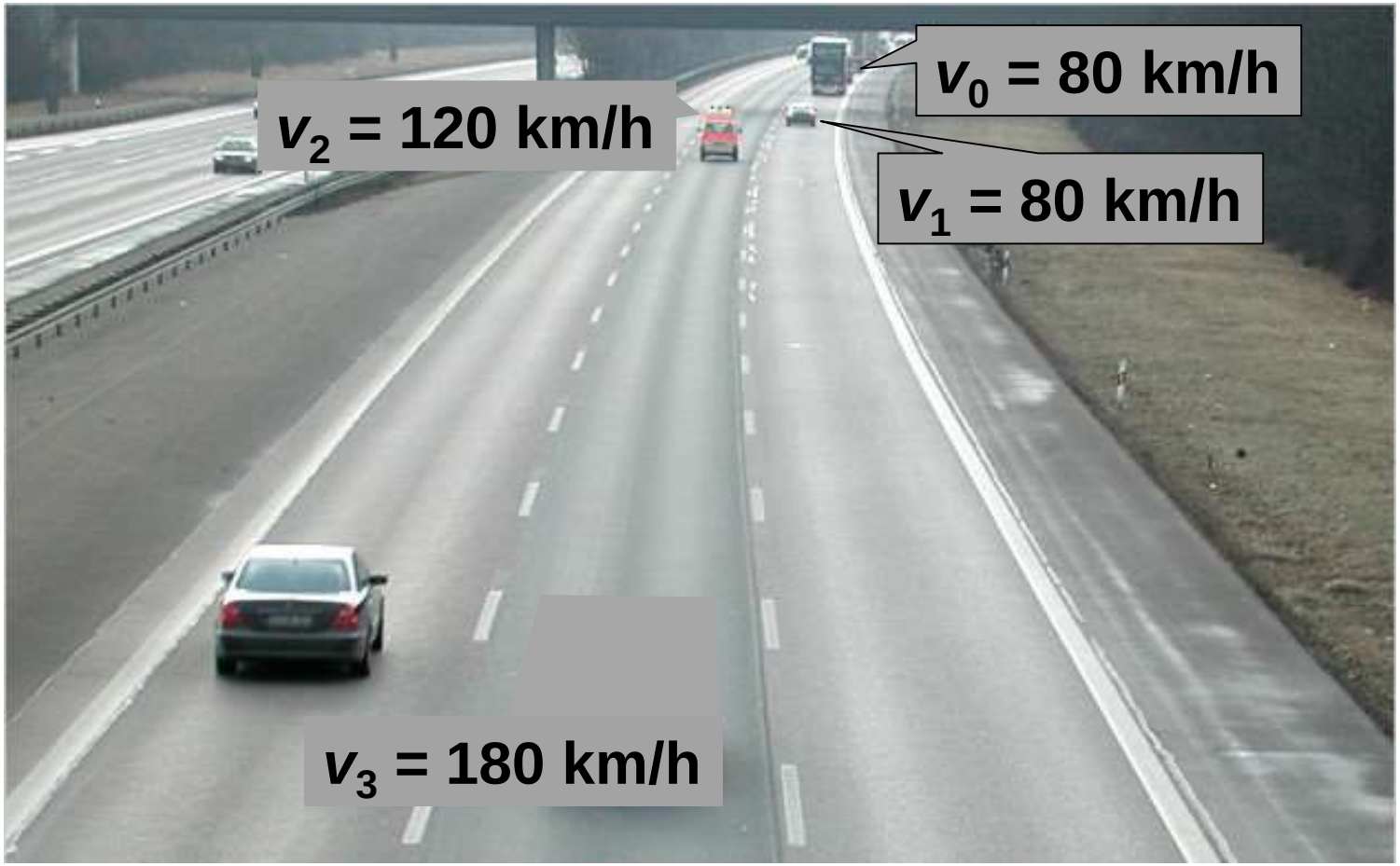
(Automotive)

- Kommunikationsaufwand wächst schneller als die eigentlichen Funktionen…
- Prozesse, Methoden und Tools kommen nicht mehr nach…

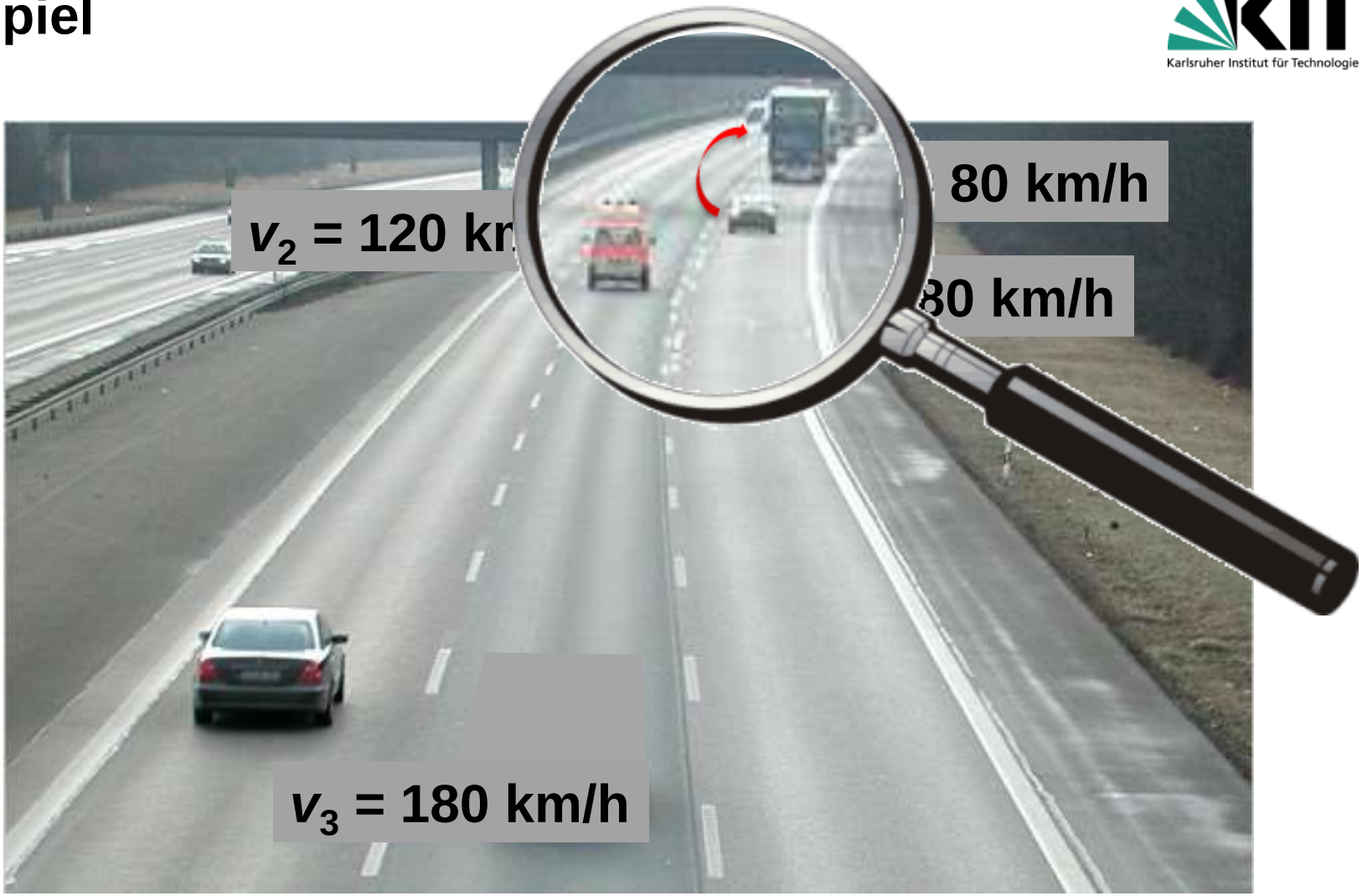


* Quelle: Vector Informatik

Beispiel

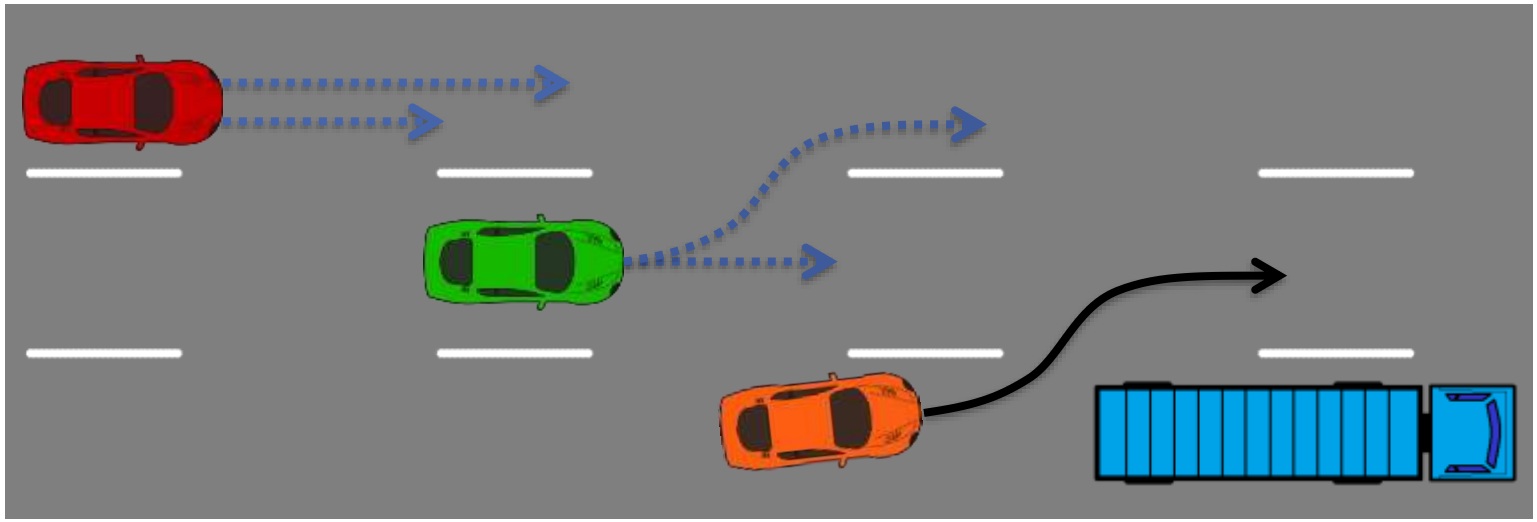


Beispiel



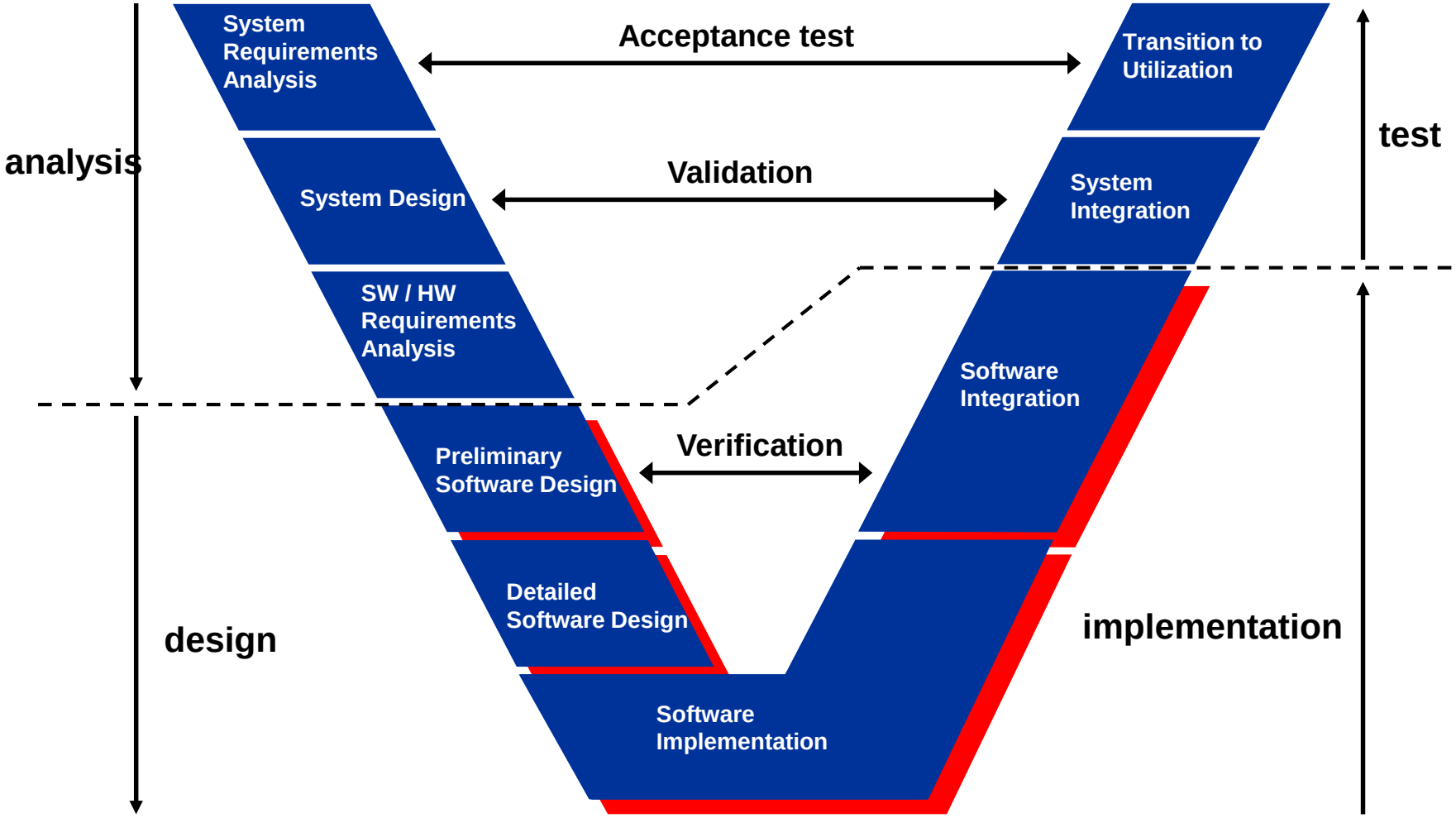
Alternativen

- Anfangssituation
  - Blauer Lkw und orangenes Fzg fahren 80 km/h → orangenes Fzg setzt zum Überholen an

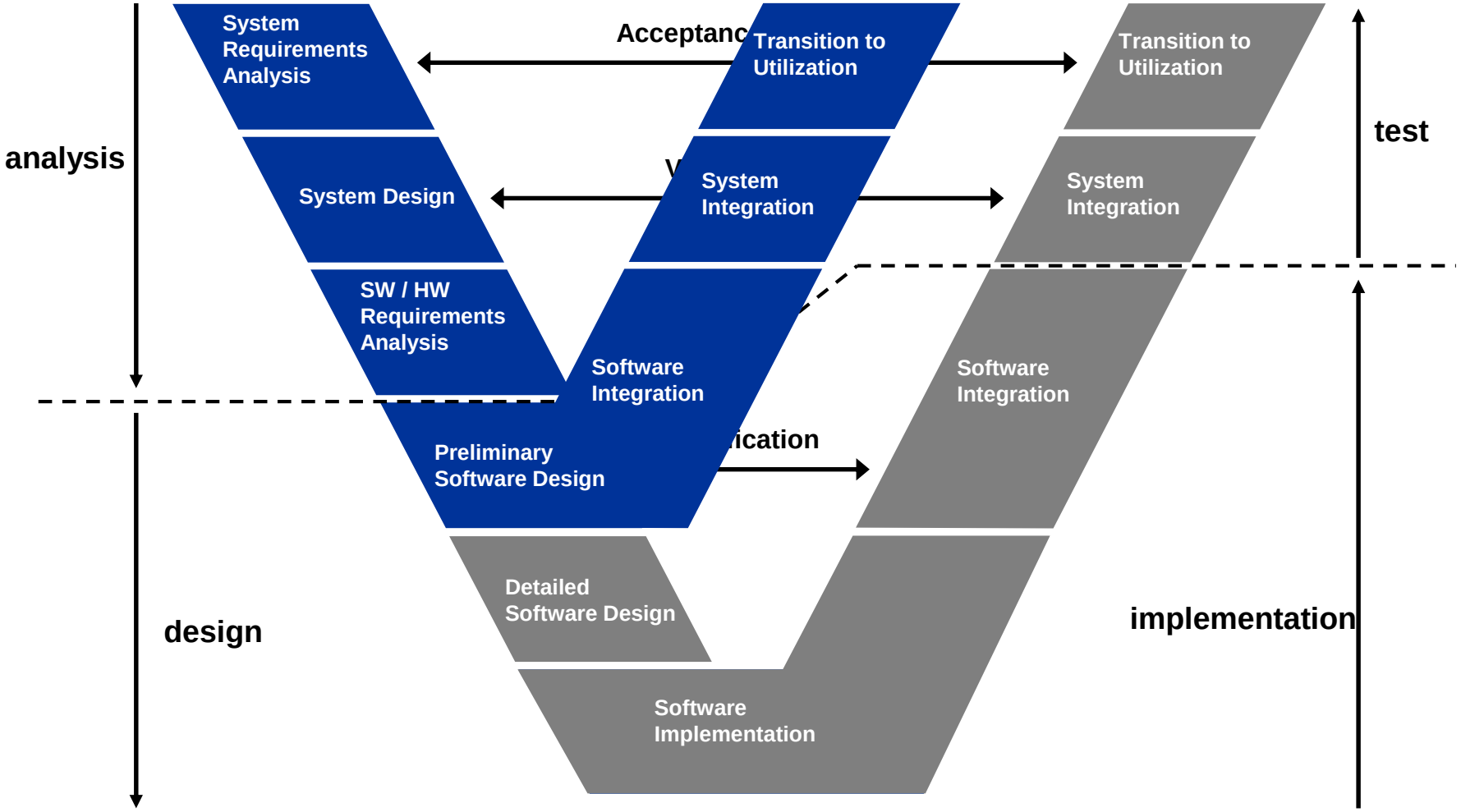


- Δv from 0 km/h to 120 km/h in 10 km/h steps → 12 variants
- Δs from 0 m to 100 m in 10 m steps → 10 variants
- Two pairs → 12*12*10*10* = **14,400 variants** for initial situation

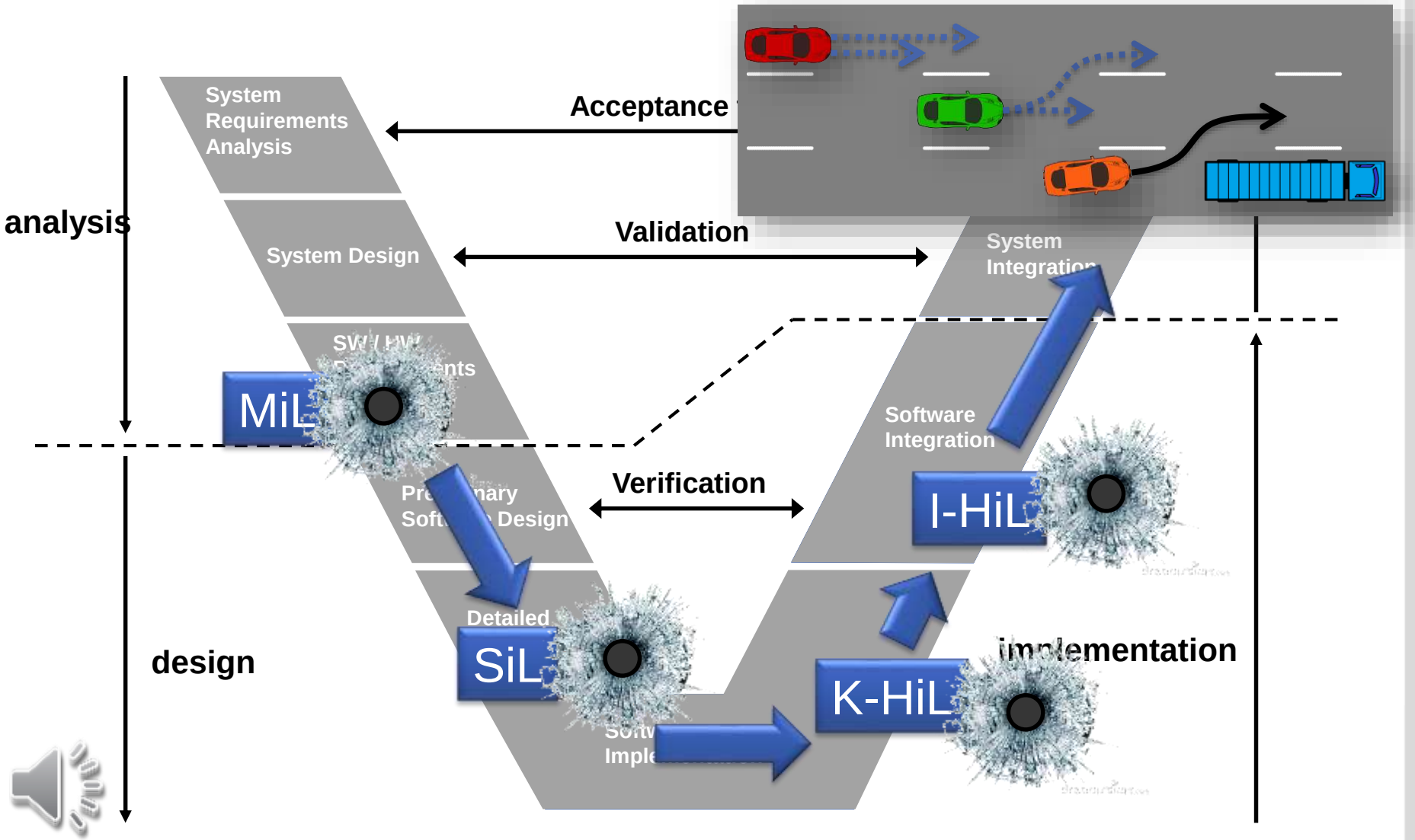
Durchgängiges Testen



Durchgängiges Testen



Durchgängiges Testen



Durchgängiges Testen

analysis

SZENARIO

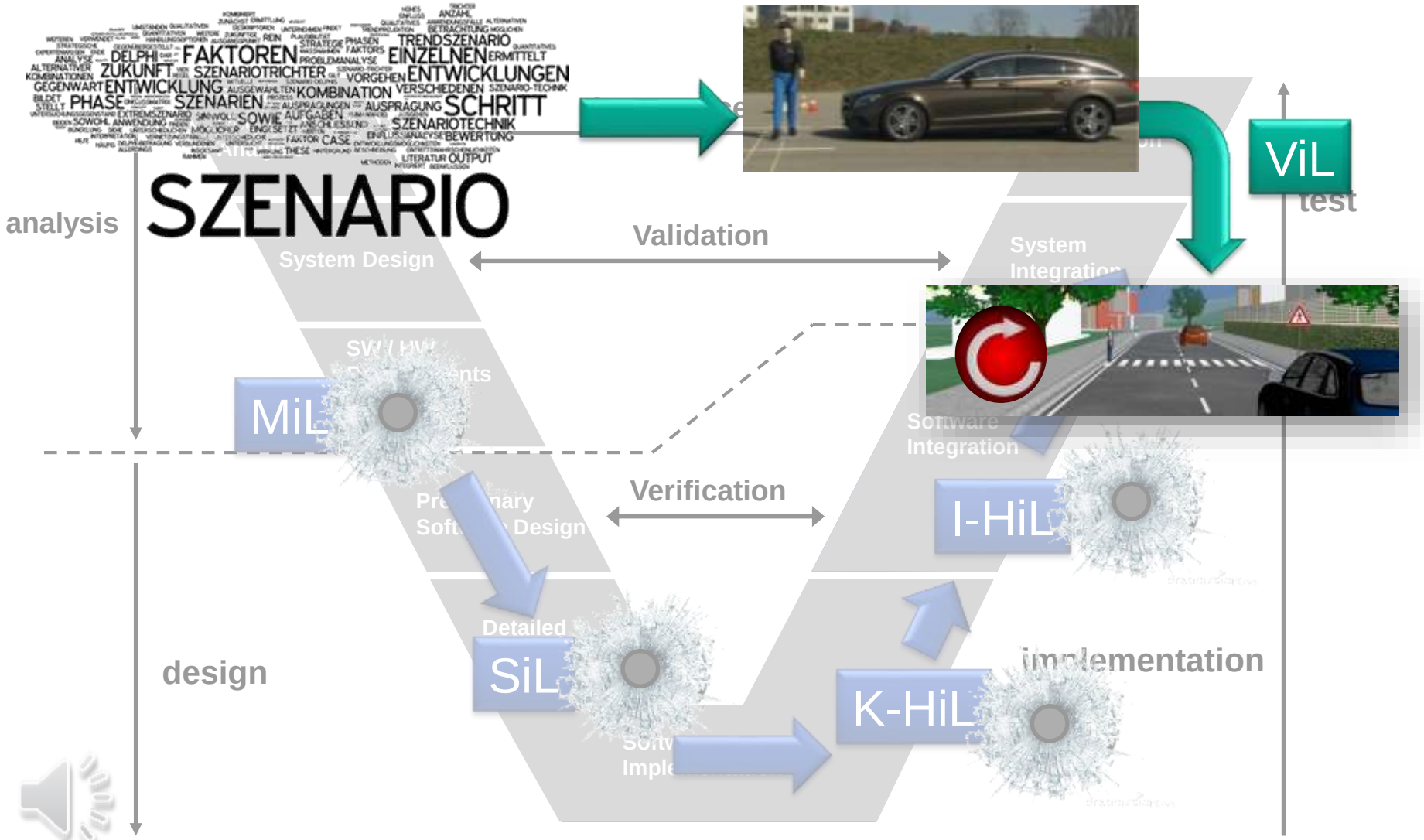
System Design

Validation

Integration

Imp

Durchgängiges Testen



Film von J. Kramer

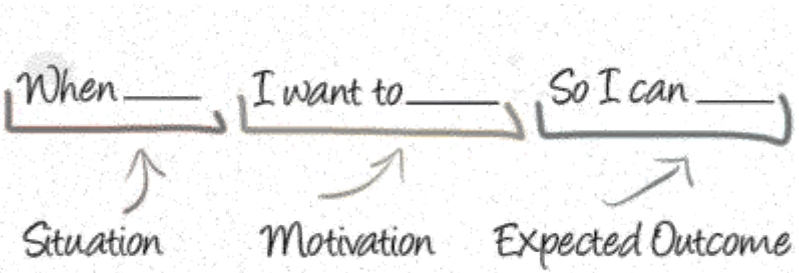
Agile Entwicklung → Agiles Testen

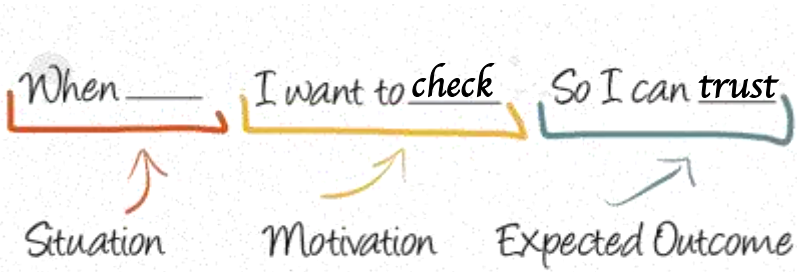
Design Prozess

- User-Stories
  - Konkrete Scenarien

Testing Prozess

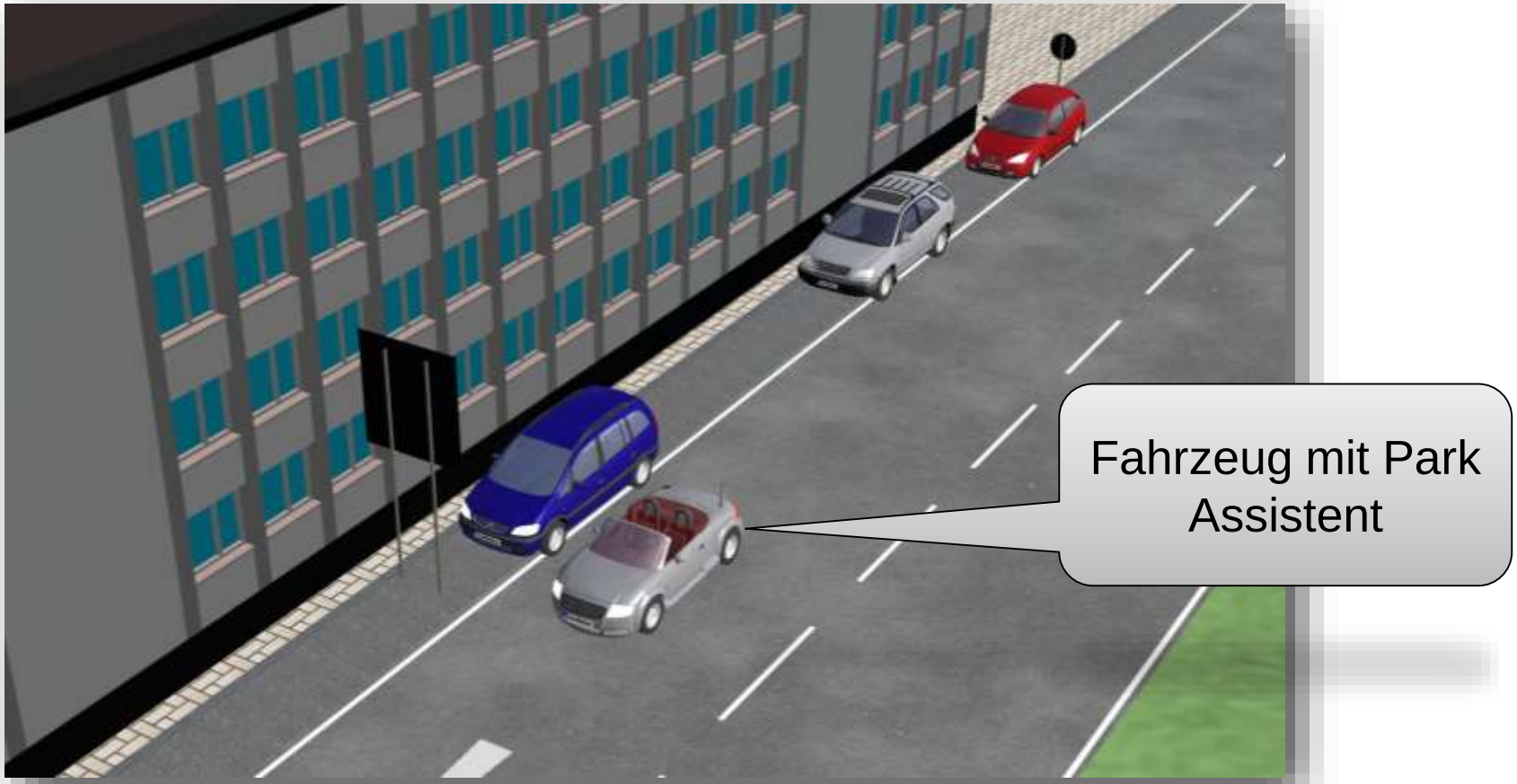
- Test-Stories
  - Initiales szenario
  - Parametervariationen für Start Situation
  - Limitierungen, Toleranzen, schrittweise
  - Pass- / Fail-Criteria





Test-Story für Park Assistent

- Parken parallel zum Seitenstreifen



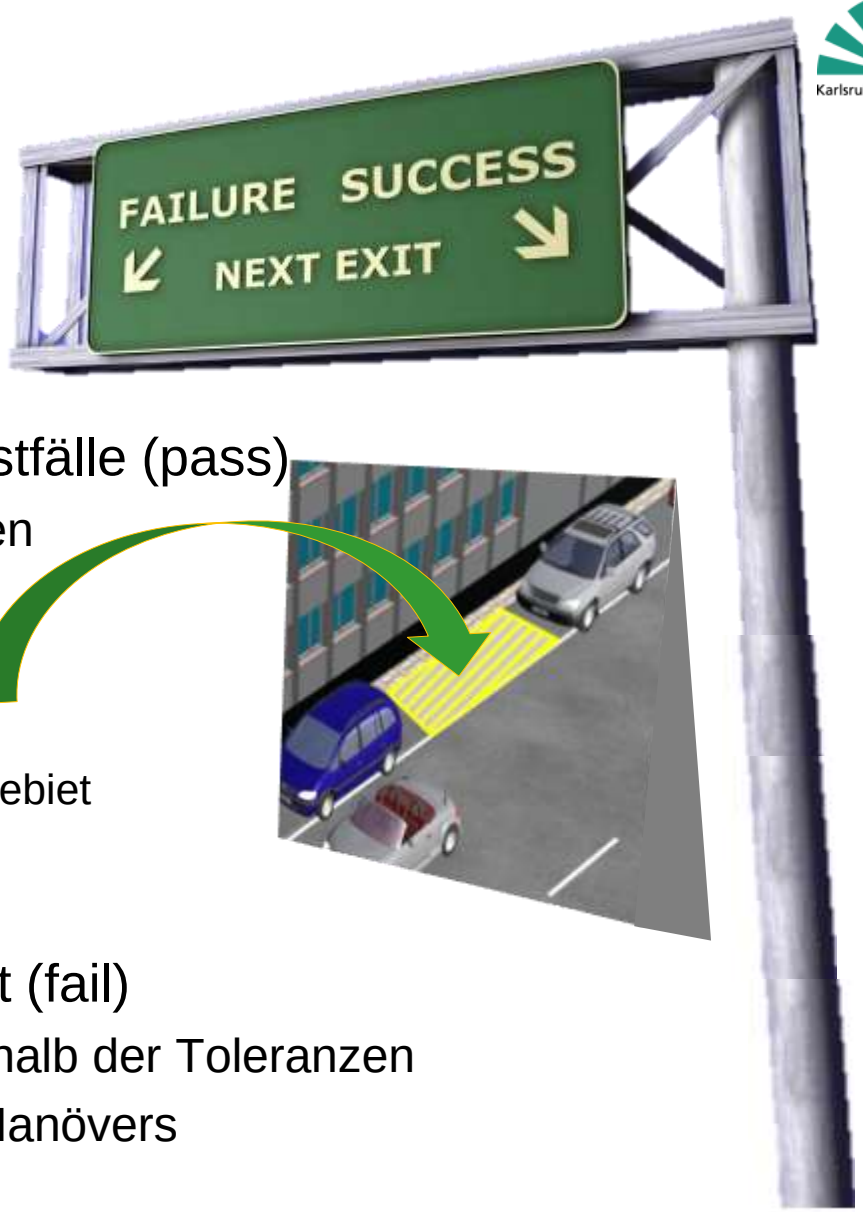
Anzahl Testfälle für eine Test-Story

- Test-Story — Läuft nach Anfangspposition

- Variationen des Startpunktes
  1. Größe der Parklücke
  2. Abstand des zu testenden Fahrzeugs zur Parklücke
  3. Winkel relative zur Parklücke
  4. Vertikale Distanz des zu testenden Fahrzeugs zur Parklücke
- Schritt
  1. 1 bis 2 Fahrzeuglängen in 100 Schritten
  2. 1 bis 2 Fahrzeuglängen in 100 Schritten
  3. Winkel -25 bis +25 Grad in 10 Schritten
  4. 0,5 bis 2 Fahrzeugbreiten in 100 Schritten

Resultierende Anzahl an Test-Stories → 100 * 100 * 10 * 100 = 10 Mio. Fälle

Kriterien für Pass/Fail



- Kriterien für erfolgreiche Testfälle (pass)
  - Berücksichtigen Toleranzen
  - Erlauben keine binäre Entscheidung
  - D.f.
    - Park Position in einem Gebiet
    - Rotation: +/- 5 Grad
    - …
- Kriterien für erfolglosen Test (fail)
  - Finale Parkposition außerhalb der Toleranzen
  - Berührung während des Manövers
  - …

Weitere Herausforderungen

- Eventualitäten
  - Keine oder dynamische Referenz Objekte
  - Zusätzliche, dynamische Objekte in der Parklücke
    - Weiteres Fahrzeug sucht Parkplatz und kommt zuvor
    - Passant, Spielzeug, Haustier kommt dazwischen
    - …
- Weiter User-Stories erfordern weitere Test-Stories
  - Vertikales Parken
  - Parken in Park-Garage
    - Markierungen relevant
  - …

Fazit

- Softwareentwicklung ist ein vielschichtiger Prozess
- Nur ausgiebige Kommunikation mit dem Kunden sichert Erfolg
- Das Zerlegen der Anforderung gliedert das Problem hierarchisch
- Tests:
  - Planung von oben nach unten
  - Durchführung von unten nach oben


Fragen?


Backup
Shift