Kapitelübersicht

1.	Einführung	- Prof. Zimmermann
2.	Aspekte des Systementwurfs	- Prof. Zimmermann
3.	Modellbasierter Entwurf	- Prof. Zimmermann
4.	Echtzeitsysteme	- Prof. Zimmermann
5.	Scheduling	- Prof. Zimmermann
6.	Sicherheit und Zuverlässigkeit	- Prof. Zimmermann
7.	Softwaretechnische Aspekte	- Prof. Fengler
8.	Hardware-Software-Codesign	- Prof. Fengler
9.	Rechnerarchitekturaspekte	- Prof. Fengler
10.	Kommunikation	- Prof. Fengler
11.	Energieeffizienz	- Prof. Fengler
12.	Domäne Automotive	- Prof. Fengler

1. Vorgehensmodelle
2. Echtzeit Programmierung
3. Testen
    1. Allgemeines
    2. Remote Debugging
    3. In Circuit Emulation
    4. Hardware in the loop / Software in the loop
    5. JTAG
4. Autosar

Definition:

- Ein Vorgehensmodell definiert einen allgemeinen Rahmen für den Prozess der Softwareerstellung

Vorgehensmodelle legen fest:

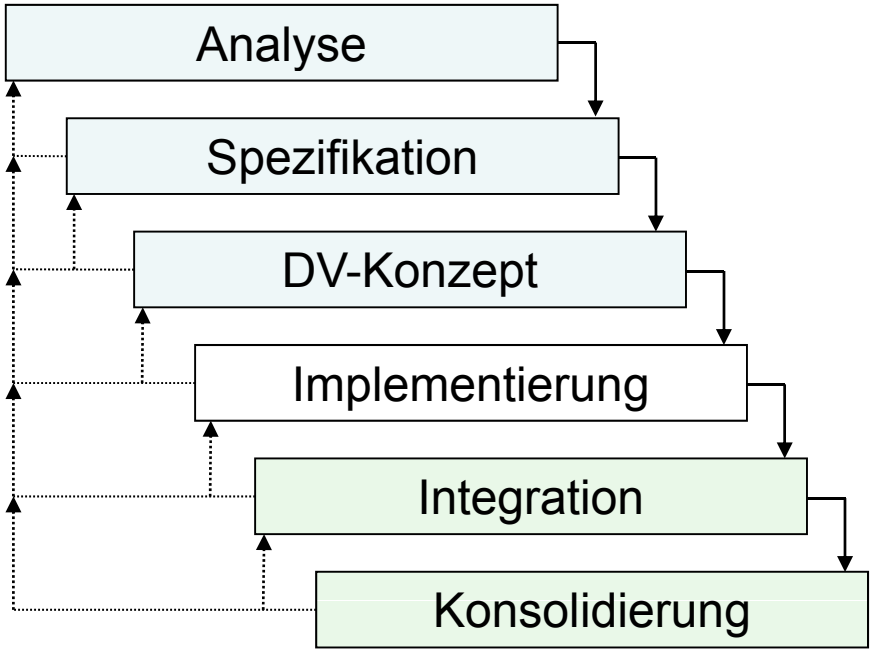
- durchzuführende Arbeiten
- Reihenfolge des Arbeitsablaufs
- Festlegung der Teilergebnisse
- Fertigstellungskriterien
- Verantwortlichkeit und Kompetenzen
- Anzuwendende Standards, Methoden Richtlinien und Werkzeuge

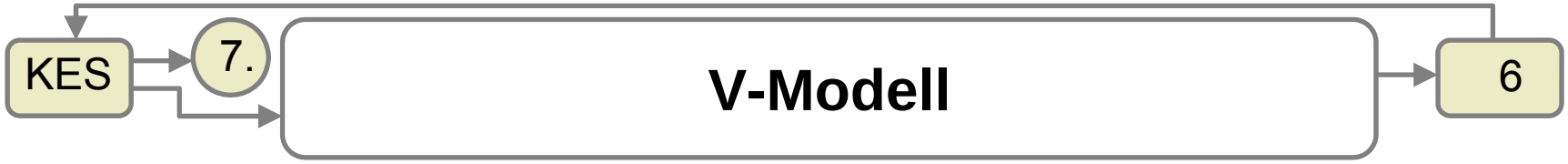
→ siehe 1. Teil der Vorlesung

- **Auswahl von Vorgehensmodellen**
  - Wasserfallmodell
  - V-Modell
  - V-Modell XT
  - Spiralmodell
  - Rapid Prototyping
  - Rational Unified Process (RUP)

Wasserfallmodell

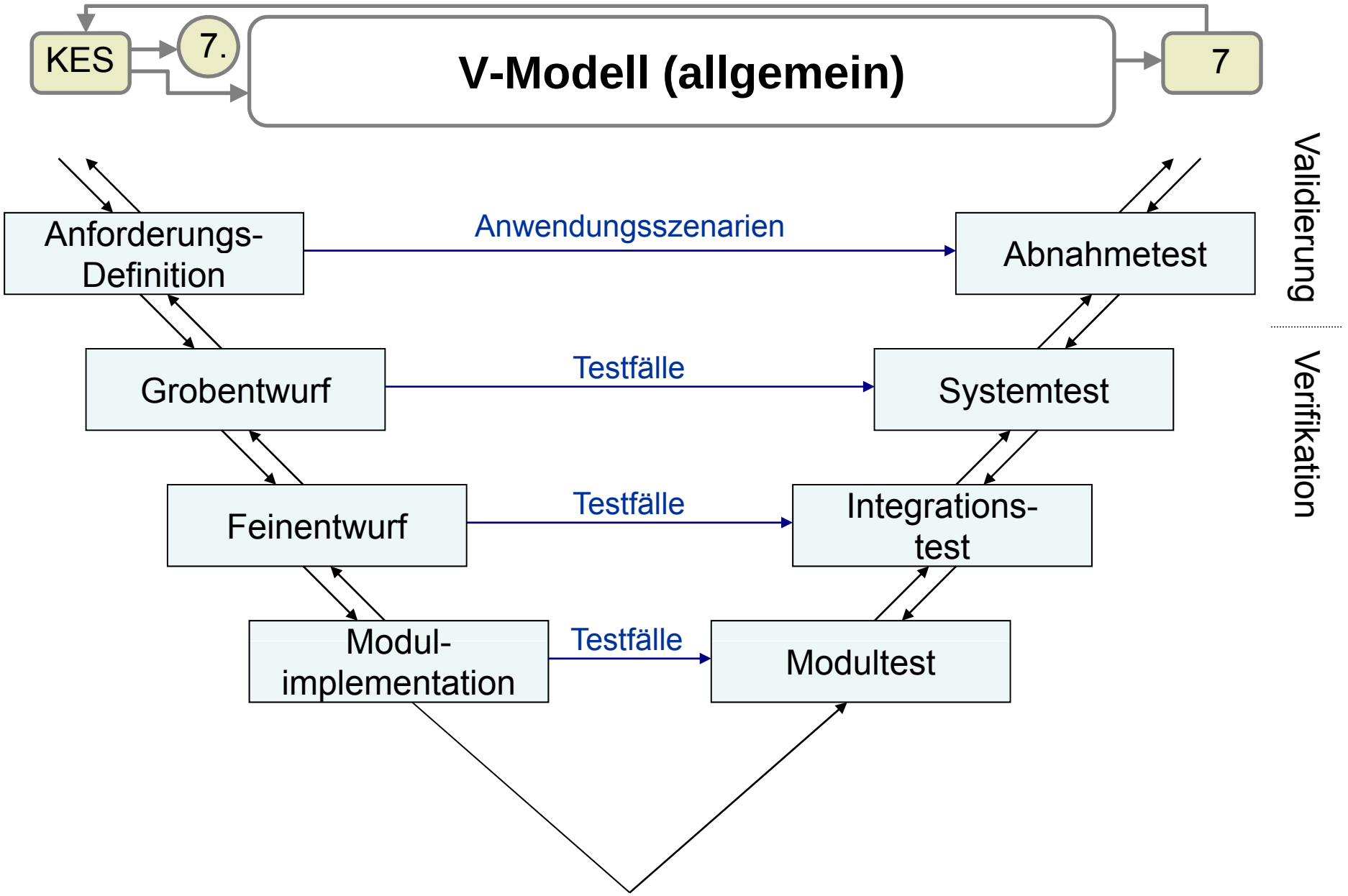
- Lineares Vorgehensmodell
- Vordefinierte Start- und Endpunkte je Phase
- Sequentieller Ablauf
- Ende jeder Phase: fertiggestelltes Dokument
- Erweitertes Modell (Bild) ergänzt iterative Elemente durch die Möglichkeit des Rücksprungs zu vorhergehenden Phasen
- Top Down Verfahren





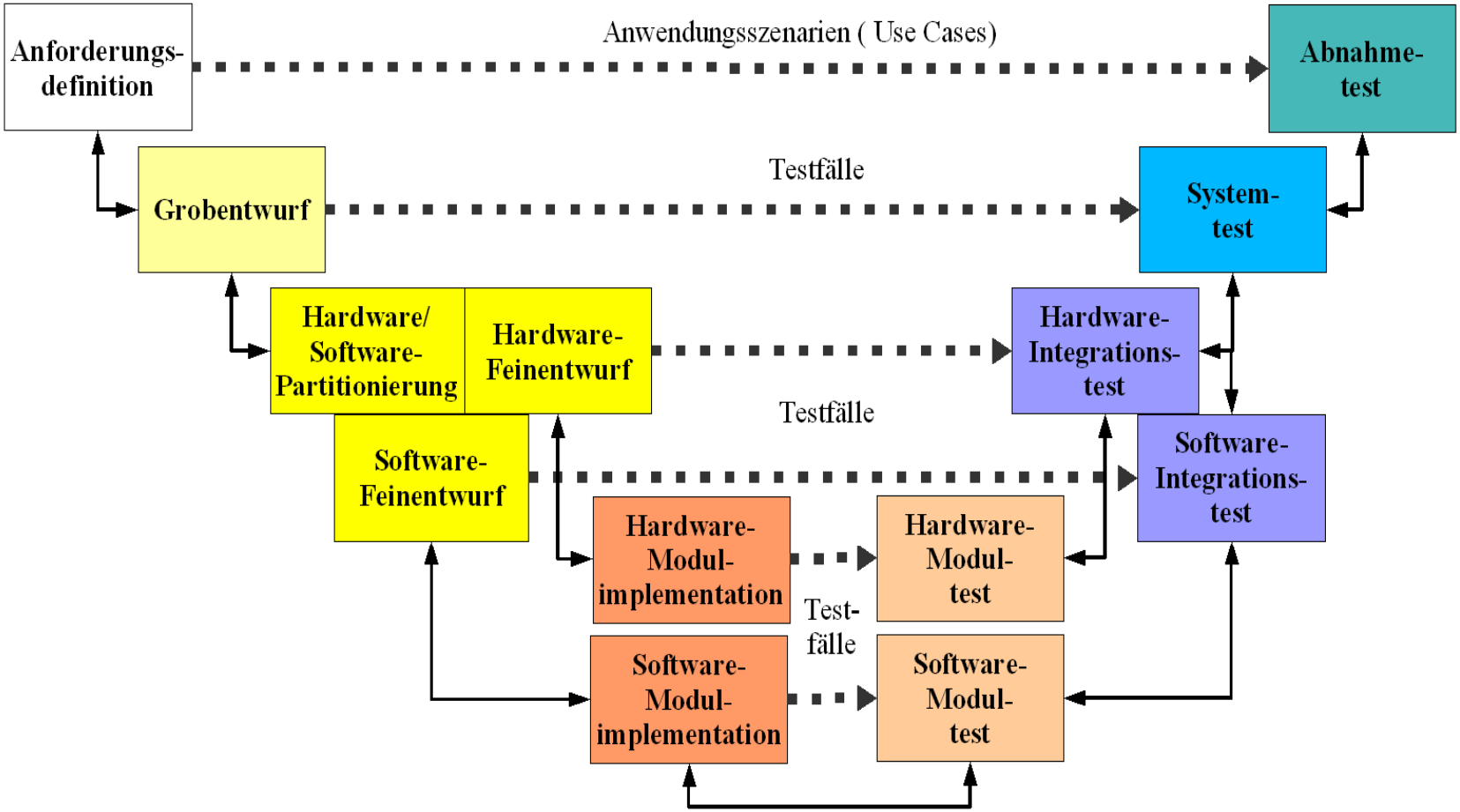
V-Modell

- Weiterentwicklung des Wasserfallmodells
- Explizite Qualitätssicherung
- Verifikation („Wird ein korrektes Produkt entwickelt“) und Validation („Wird das richtige Produkt entwickelt“) Bestandteile des Modells

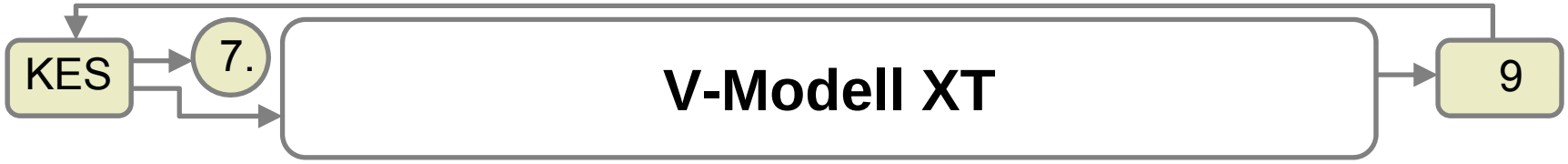


Quelle: Balzert, H: Lehrbuch der Softwaretechnik

V-Modell (Erweitert für eingebettete Systeme)



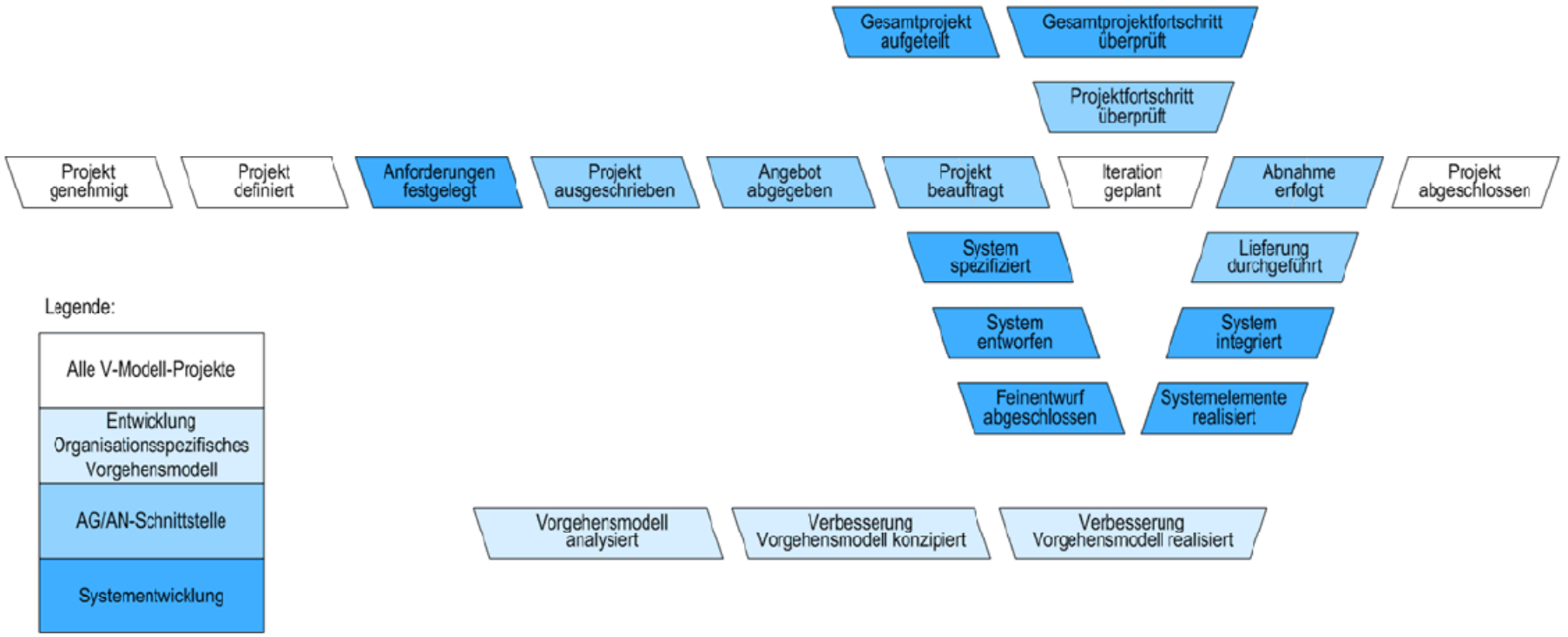
Quelle: Müller, M.; Implementation eines UML Entwurfs; Studienarbeit; TU Ilmenau, 2005



V-Modell XT

- Modell ist an die jeweiligen Bedürfnisse anpassbar
- Einbindung des Auftraggebers
- Starke Modularisierung mit Vorgehensbausteinen
- Keine Vorschriften über zeitliche Abfolge der einzelnen Verfahrensbausteine
- Produkte im Mittelpunkt, nicht Aktivitäten
- Verpflichtende Vorgehensbausteine (V-Modell Kern):
  - Projektmanagement
  - Qualitätssicherung
  - Konfigurationsmanagement
  - Problem- und Änderungsmanagement
- Weitere optionale Vorgehensbausteine nach Projekttyp

V-Modell XT

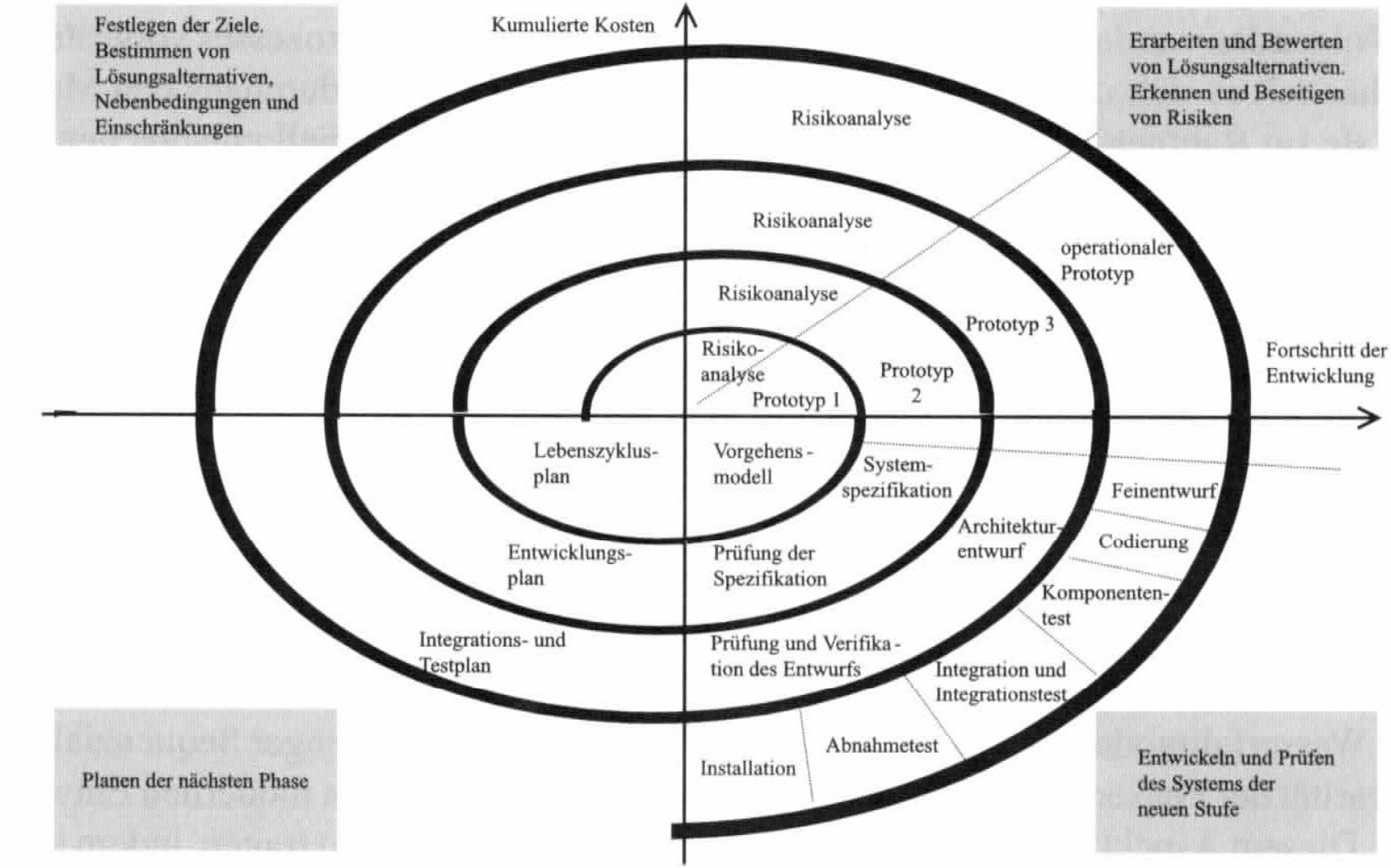


Quelle: Dokumentation V-Modell XT, 2004, http://www.v-modell-xt.de/

Spiralmodell

- Iteratives Verfahrensmodell
- Wiederholung von immer gleich aufgebaute Zyklen
  - Festlegen der Zielen
  - Erarbeiten und Bewerten von Lösungsalternativen
  - Entwickeln und Prüfen des Systems
  - Planen der nächsten Phase

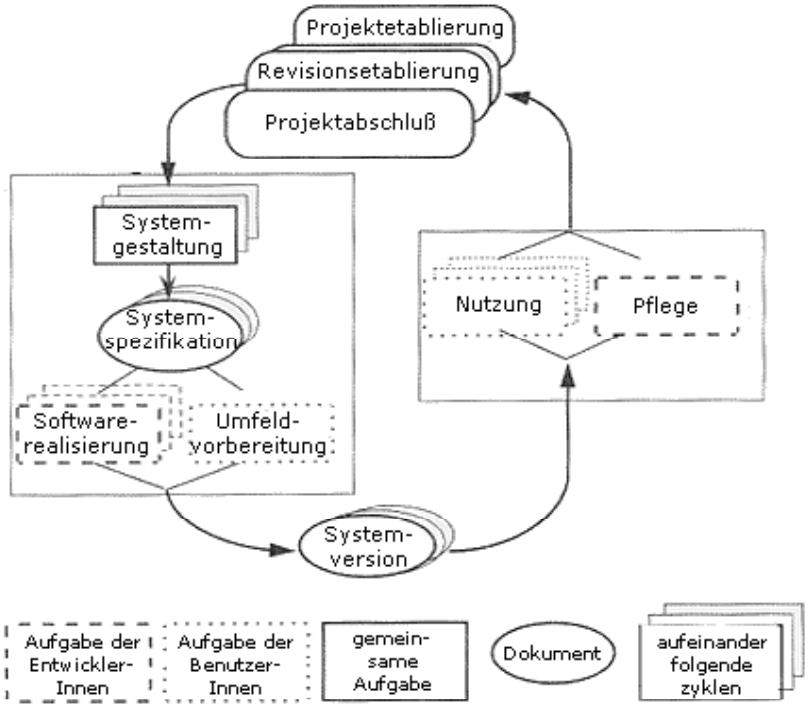
Spiralmodell



Quelle: A. Silkora, R. Drescher: Software-Engineering und Hardware-Design, Hanser, 2002

Rapid Prototyping

- Erstellung eines Prototyps und zyklische Erweiterung / Überarbeitung, bis Projektziel erreicht ist
- Besonders geeignet, wenn die Anforderung bei Projektbeginn nicht genau beschreibbar sind



Quelle: C. Floyd u.a., in: Informatik-Spektrum Heft 1, Feb. 1997.

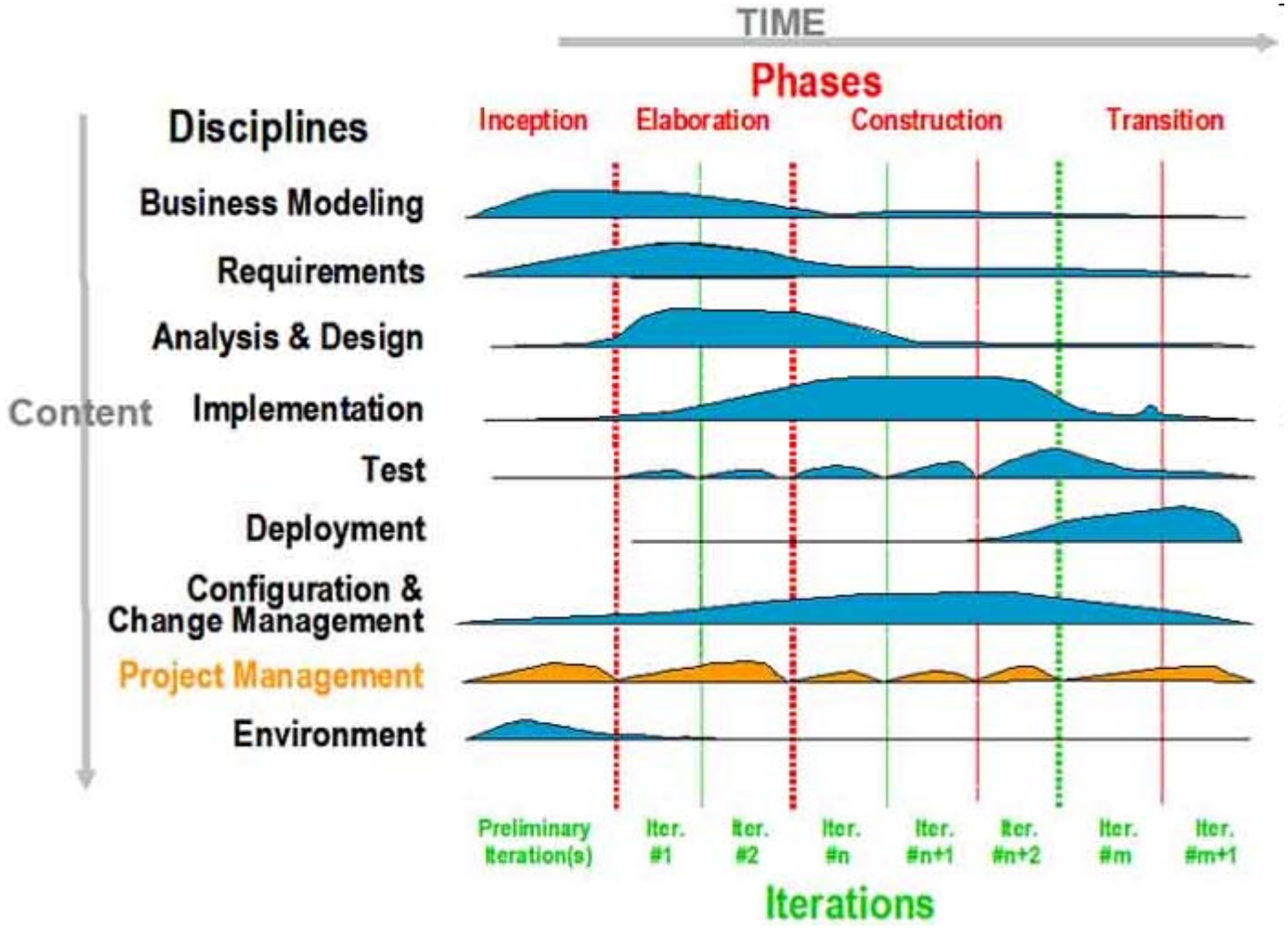
Rational Unified Process (RUP)

- Objektorientiertes Vorgehensmodell zur Softwareentwicklung
- Kommerzielles Produkt von Rational Software, heute IBM
- Benutzt UML als Notationssprache
- Metamodell für Vorgehensmodelle zur Softwareentwicklung
- Basiert auf folgenden Prinzipien
  - Anwendungsfällen
  - Architektur im Zentrum der Planung
  - Inkrementellem und iterativen Vorgehen

Rational Unified Process (RUP)

- grundlegende Arbeitsschritte
  - Geschäftsprozessmodellierung
  - Anforderungsanalyse
  - Analyse & Design
  - Implementierung
  - Test
  - Auslieferung
  - Weitere unterstützende Arbeitsschritte
- 4 Phasen, in welchen diese Schritte angewendet werden
  - Konzeptionsphase
  - Entwurfsphase
  - Konstruktionsphase
  - Übergabephase

Rational Unified Process (RUP)



Quelle: IBM

Softwaretechnische Aspekte

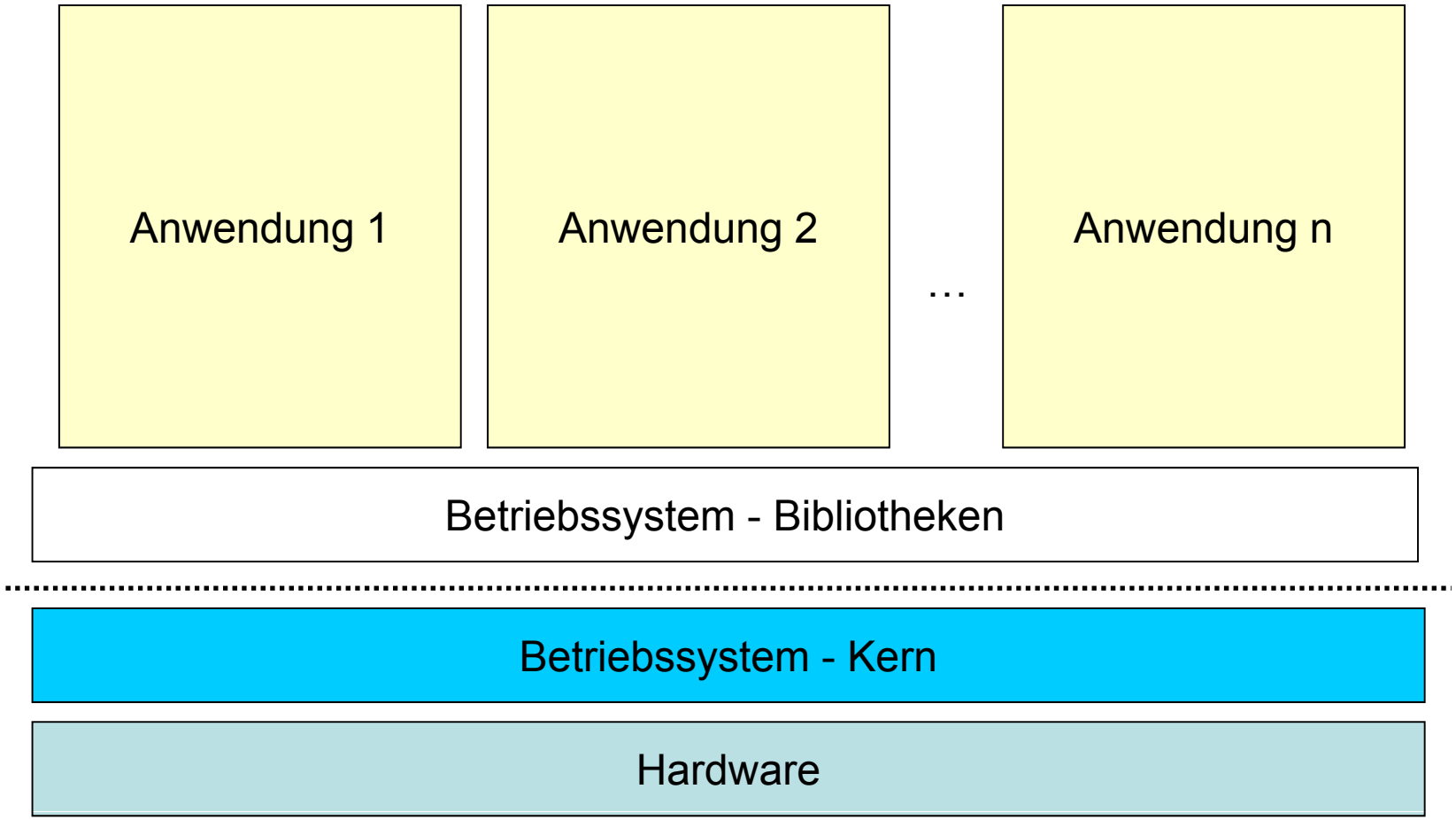
1. Vorgehensmodelle
2. Echtzeit Programmierung
3. Testen
    1. Allgemeines
    2. Remote Debugging
    3. In Circuit Emulation
    4. Hardware in the loop / Software in the loop
    5. JTAG
4. Autosar

Echtzeitprogrammierung

„Echtzeitbetrieb ist ein **Betrieb eines Rechnersystems**, bei dem Programme zur Verarbeitung anfallender Daten **ständig** derart betriebsbereit sind, dass die Verarbeitungs-ergebnisse **innerhalb einer vorgegebenen Zeitspanne** verfügbar sind.“ (DIN 44300)

→ siehe 1. Teil der Vorlesung

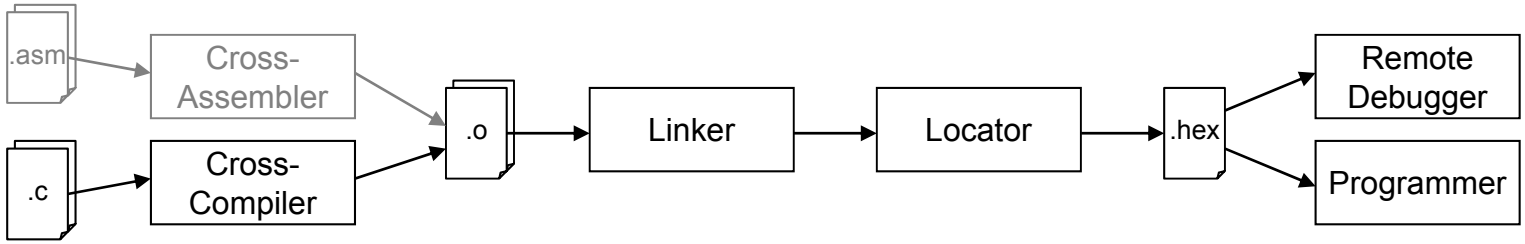
Softwarearchitektur



- Embedded C / C++
- Embedded Java / Realtime Java
- Assembler

Embedded C / C++

- Eingeschränkter Sprachumfang
  - z.B. keine Fließkommaverarbeitung, keine Mehrfachvererbung, keine Namespaces
  - Sprachumfang ist abhängig von Plattform und Compiler
- Für die Zielplattform optimierte Standardbibliotheken
  - z.B. keine Datei Ein-/Ausgabe, dafür aber direkter Zugriff auf Funktionen der vorhandenen Hardware
- Übersetzungsvorgang
  - Cross-Compiler übersetzt Quelltext in Binärcode für die Zielarchitektur
  - Locator passt Speicheradressen für Programm- und Datenbereiche entsprechend der vorhanden ROM / RAM Adressen an



```
void led_out(char ziffer, char dot, int stelle) {
/* Ziffer: 0-15 (A-F),'-','<Space>'
   Dot: >0:
   Stelle: 0-7 */

    /* 7 – Segment Repräsentation der Ziffern 0-9, a-f, -) */
    unsigned char ztab[18] = {0x3F,0x06,0x5b,0x4f,0x66,0x6d,0x7d,
            0x07,0x7F,0x6F,0x77,0x7c,0x39,0x5e,0x79,0x71,0x40,0};
    /* Adressen der 7-Segment Anzeigen */
    unsigned char digit[8]={0x20, 0x10, 0x40, 0x30, 0x60, 0x50,
                  0x80, 0x70};
    unsigned char out;
    if (ziffer=='-') ziffer=16;
    if (ziffer==' ') ziffer=17;
    if (ziffer>17) ziffer=17;
    out = ztab[ziffer];
    if (dot) out |= 0x80; // Punkt einschalten, falls erforderlich
    DP1L = 0xFF; // Richtung Datenbusport: Ausgabe
    P1H = digit[stelle]; // Adresse auf Adressbus ausgeben
    P1L = out; // Wert auf Datenbus ausgeben
    P1H = 0; // Adressbus zurücksetzen
}
```

RealTime Java

- Erweiterung von Java um Echtzeit–Funktionen
- Spezielle Thread - Klassen für Echtzeitaufgaben, Ausführung wahlweise mit (RealtimeThread) oder ohne Garbage Collection (NoHeapRealtimeThread)
- Prioritäten-basiertes Scheduling
- Spezielle Speicherverwaltungsklassen (ScopedMemory, ImmortalMemory), welche nicht vom Garbage Collector verwaltet werden
- Klassen zur Synchronisierung von Threads, priorisierte Warteschlangen für Ereignisse
- Asynchrone Ereignisbehandlung für externe Ereignisse
- Hochauflösende Timer
- Direkter Zugriff auf den physikalischen Speicher

Embedded Java / J2ME

- Auf eingebettete Systeme zugeschnittene, optimierte Variante der Java Laufzeitumgebung
- Verschiedene Sprachumfänge, z.B. Embedded Java (basiert auf Java SE) oder Java Micro Edition, J2ME
- Hohe Zuverlässigkeit, Stabilität und Verfügbarkeit des Systems
- Wiederverwendbarkeit und Portabilität von Komponenten

- Beispiele
  - MIDP (Mobiltelefone)
  - MHP (DVB Receiver)
  - jControl, sunSPOT (embedded Java Hardware)
  - nanoVM (VM für Atmel AVR Controller)

Assembler

- Direkte Umsetzung in Maschinensprache
- Ausnutzung der kompletten Bandbreite des Prozessors
- Direkte Programmierung angeschlossener Peripherie
- Programme erfahrener Assemblerprogrammierer meist schneller und kleiner als Programme, welche mit einer Hochsprache wie C oder Java erstellt wurden
- Bestimmung des genauen Zeitverhaltens möglich
- Sprache ist prozessorspezifisch, keine direkte Übertragung auf andere Plattformen möglich
- Programmquelltexte sind im Vergleich zu Hochsprachen wesentlich länger und komplizierter zu verstehen

Assembler


```
        MOV DX, 0B0H ; 1. Ausgabeadresse in DX
        MOV EDI,10
        MOV ESI,OFFSET ziff ;Adresse von ziff ins Register bringen
m2:     MOV AL,[ESI+EDI-1]
        OUT DX,AL
        INC DX ; nächste Displaystelle
        DEC EDI ; nächste Ziffer
        JNZ m2 ; Solange wie EDI nicht 0 zurück zu m2

ziff    DB 3FH,03H,6DH,67H,53H,76H,7EH,23H,7FH,77H
```

Softwaretechnische Aspekte

1. Vorgehensmodelle
2. Echtzeit Programmierung
3. Testen
    1. Allgemeines
    2. Remote Debugging
    3. In Circuit Emulation
    4. Hardware in the loop / Software in the loop
    5. JTAG
4. Autosar

Ziele

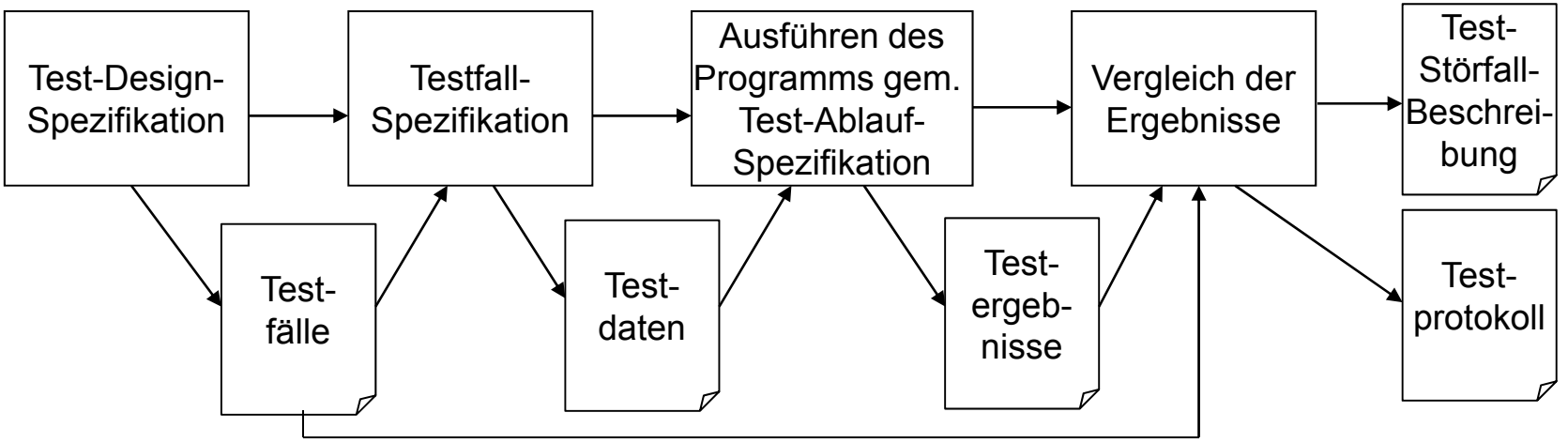
- Testen der Software in kontrollierter Umgebung um
  - Nachzuweisen, das die Software wie spezifiziert funktioniert (Verifikation, *Entwickeln wir das System korrekt ?*)
  - Fehler zu finden
  - zu Prüfen, das die Spezifikation den Anforderungen entspricht (Validierung, *Entwickeln wir das korrekte System ?*)

Vorgehensweise

Vorgehensweise nach IEEE 826

- Übersicht
  1. Testplan
- Test-Spezifikation
  1. Test-Design-Spezifikation
  2. Testfall-Spezifikation
  3. Test-Ablauf-Spezifikation
  4. Testfall-Übertragungsbeschreibung
- Test-Beschreibung
  1. Test-Protokoll
  2. Test-Störfall-Beschreibung
  3. Test-Zusammenfassung

Vorgehensweise



Klassifizierung

- Klassifizierung von Tests nach Informationsstand
  - White Box Test
    Die interne Funktionsweise des Testobjektes ist bekannt und wird zur Erzeugung der Testfälle genutzt
  - Black Box Test
    Die interne Funktionsweise des Testobjektes ist nicht bekannt. Die Testfälle werden auf Basis der definierten Anforderungen und Schnittstellen entworfen
  - Grey Box Test
    Die Testfälle werden vor dem zu testenden System erstellt

Teststufen

- Komponententest (Modultest, Unit-Test)
  - Test auf der tiefsten Ebene, einzelne Komponenten (Module, Klassen, Funktionen) werden auf Funktionalität getestet (White Box Test).
- Integrationstest
  - Testet die Zusammenarbeit von verschiedenen Komponenten (Black Box Test)
  - Bei eingebetteten Systemen Test zusätzlich Test der Zusammenarbeit von Hardware- und Softwarekomponenten
- Systemtest
  - Test des Gesamtsystems gegen die Anforderungen (funktionale und nicht-funktionale).
- Abnahmetest (Verfahrenstest, Akzeptanztest)
  - Test des Gesamtsystems durch den Endbenutzer in dessen Systemumgebung

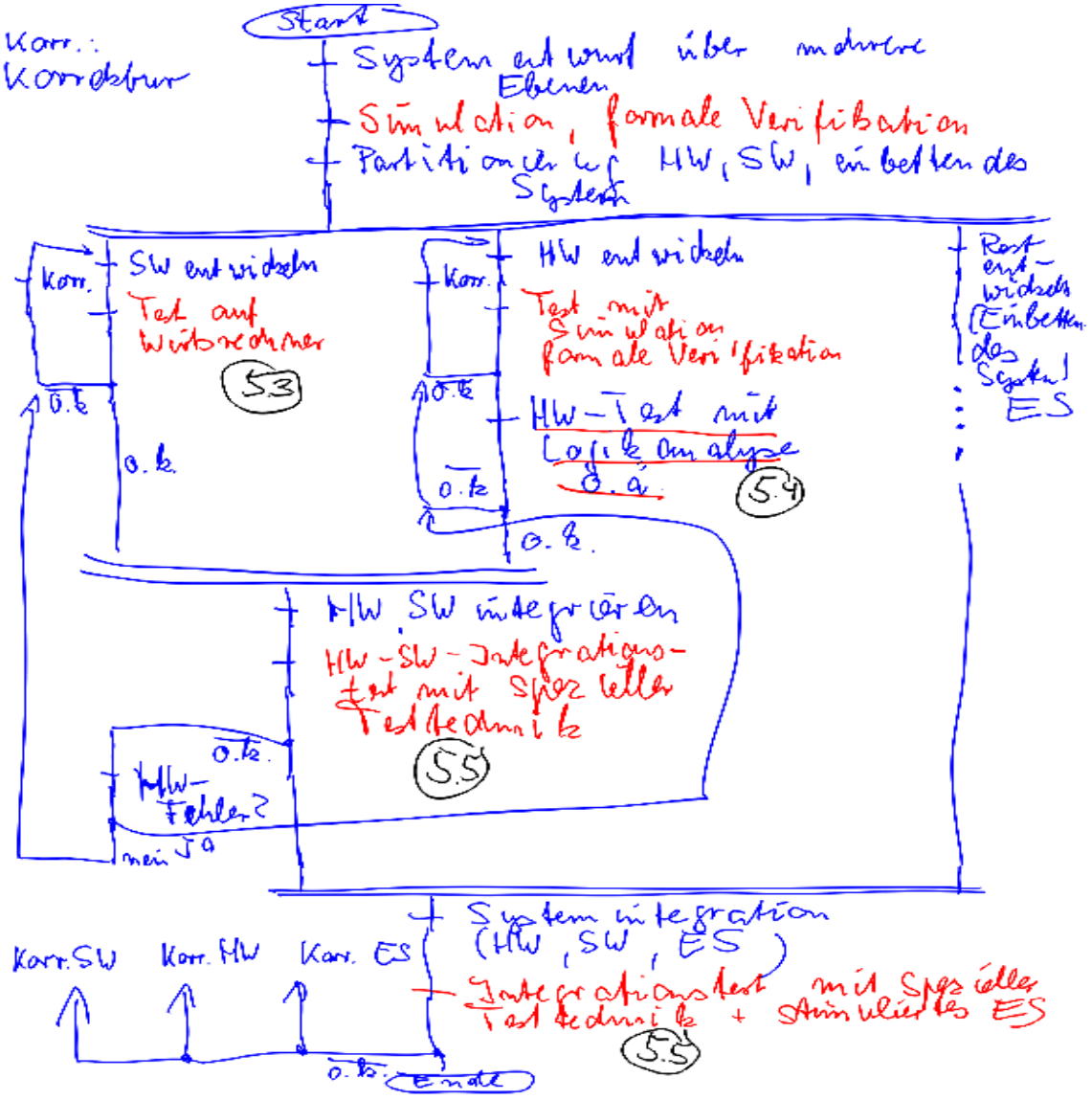
Automatisierte Unit Tests

- xUnit Bibliotheken sind für viele Programmiersprachen verfügbar
- Stellen ein Framework bereit, um wiederholbare Testfälle automatisiert abzuarbeiten
  - Definition von Testfällen
  - Überprüfen von Testbedingungen
  - Gruppierung von Testfällen (Suite)
- Beispiel (Java: JUnit 4)

```
import org.junit.Test;
import static org.junit.Assert.*;

@Test public void simpleAdd() {
      Money m12EUR= new Money(12, "EUR");
      Money m14EUR= new Money(14, "EUR");
      Money expected= new Money(26, "EUR");
      Money result= m12EUR.add(m14EUR);
      assertTrue(expected.equals(result));
 }
```

Einordnung von Inbetriebnahme und Test in den Entwicklungsablauf



Integrationstest für Embedded Systems

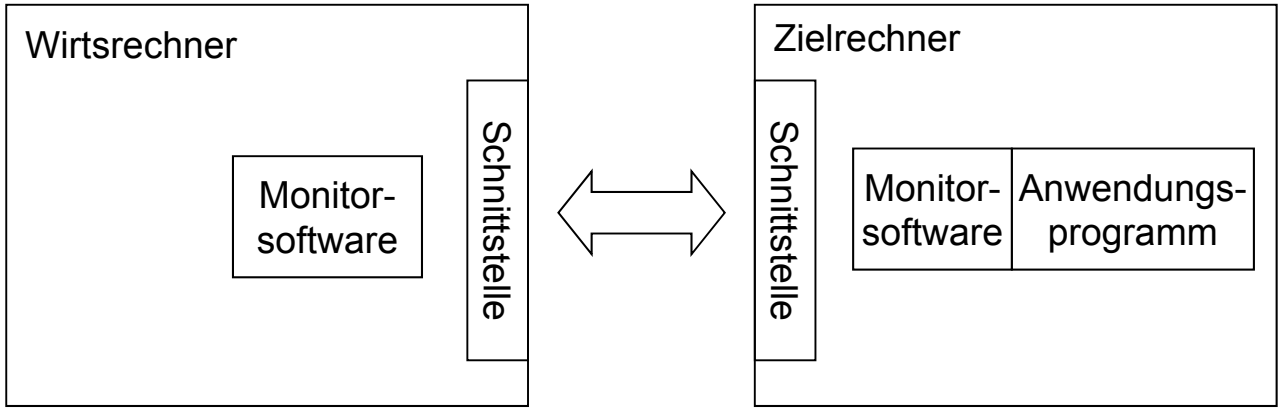
- Ausgangspunkt
  - Software ist (zumindest teilweise) entworfen und separat getestet
  - Rechnerkern und Zusatzhardware sind entworfen und getestet
- Testmöglichkeiten
  - Laden (oder ROM programmieren) der Software und Test über normale Bedienhandlung -> Fehlerursachen sind so nicht erkennbar
  - Überwachen und Steuern der Zielhardware während des Tests der Software über Zusatzsoftware / Hardware, um Fehlerursachen finden zu können

Softwaretechnische Aspekte

1. Vorgehensmodelle
2. Echtzeit Programmierung
3. Testen
    1. Allgemeines
    2. Remote Debugging
    3. In Circuit Emulation
    4. Hardware in the loop / Software in the loop
    5. JTAG
4. Autosar

Remote Debugging

- Software wird auf der Zielhardware getestet
- Ein Monitorprogramm überwacht und steuert die Zielhardware
- Monitorprogramm lädt Programme, zeigt Prozessorzustände und Speicherinhalte, steuert Programmabarbeitung
- Anforderungen: zusätzliche Schnittstelle, Speicher für Monitor, ladbarer Programmspeicher
- Nur bedingt echtzeitfähig



Typische Monitorfunktionen:

- Laden von Programmen
- Anzeige Prozessorzustand (Register, Befehlszähler u.ä)
- Anzeige Speicherinhalte
- Gesteuerte Programmabarbeitung (Einzelbefehle, Befehle bis Haltepunkt, …)
- Erzeugen von Programmtraces (tatsächlicher Ablauf der Befehle)

Softwaretechnische Aspekte

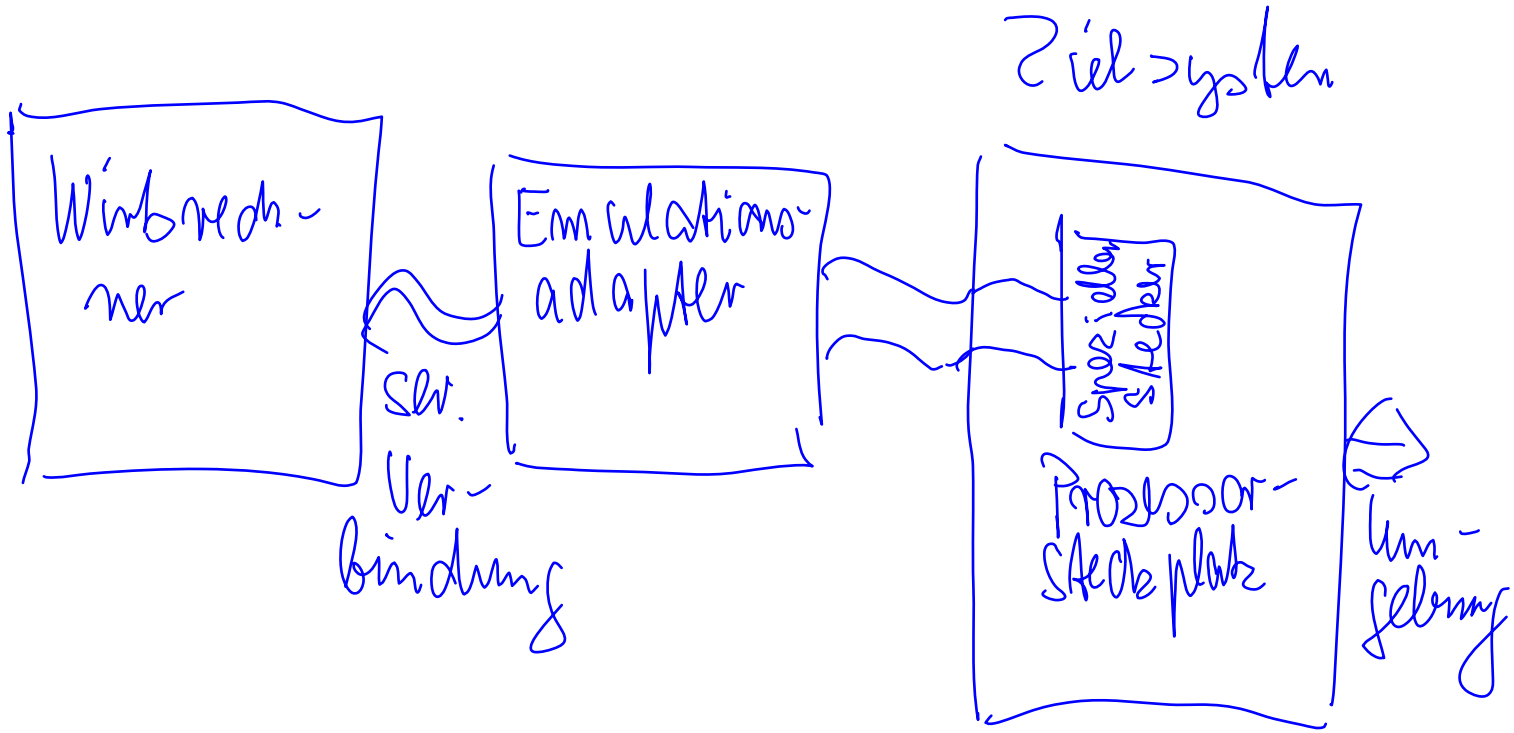
1. Vorgehensmodelle
2. Echtzeit Programmierung
3. Testen
    1. Allgemeines
    2. Remote Debugging
    3. In Circuit Emulation
    4. Hardware in the loop / Software in the loop
    5. JTAG
4. Autosar

In Circuit Emulation

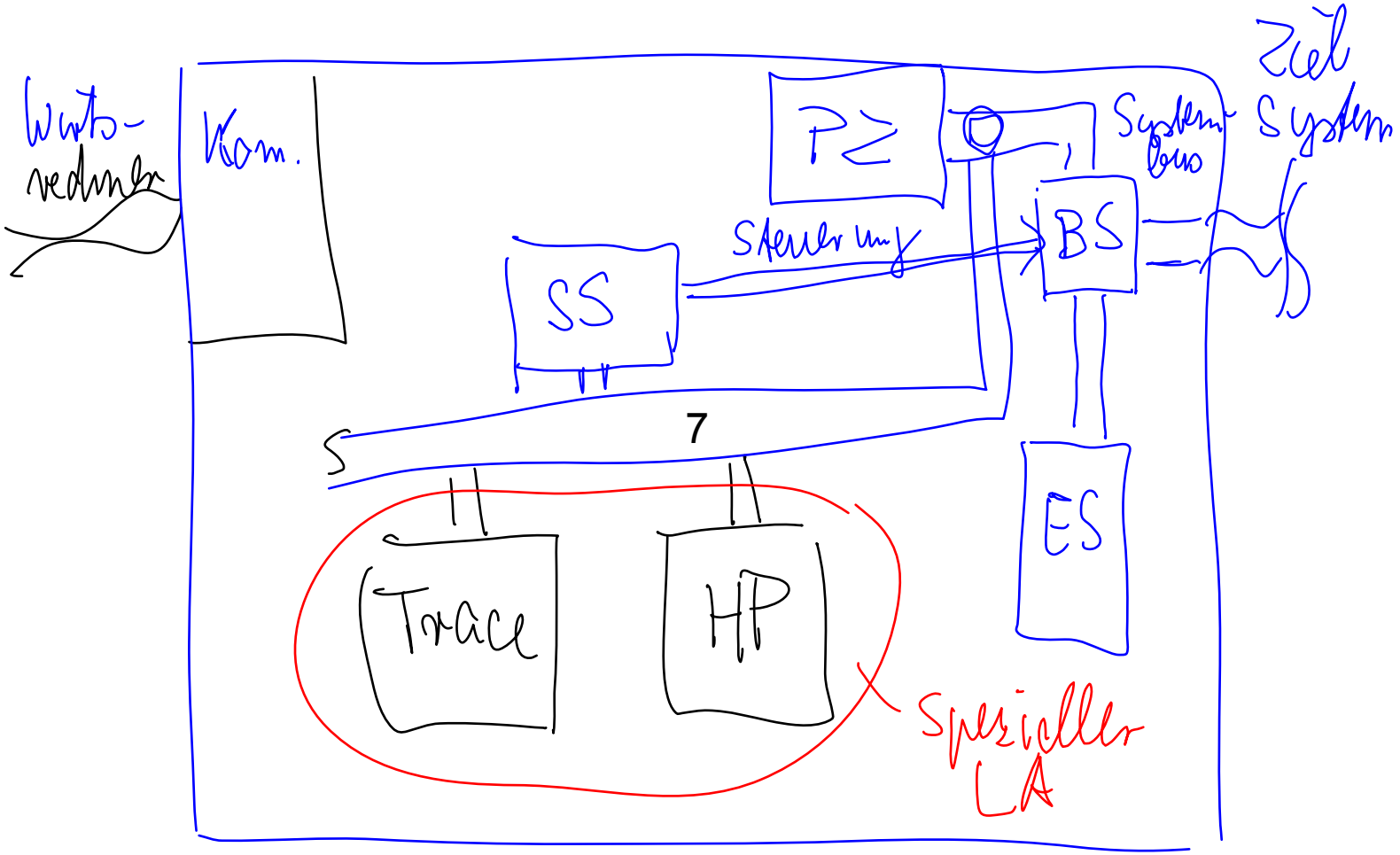
- Emulator ersetzt den Prozessor / Controller in der Zielhardware
- Emulator ist spezielle Version des Prozessors mit zusätzlichenherausgeführten Signalen („Bond-Out Chip“) oder wird komplett nachgebildet
- Gezieltes Überwachen von Ein- und Ausgängen
  - Aufzeichnung der Ein- und Ausgangssignale im Tracebuffer für spätere Analyse in Echtzeit
- Debuggingfunktionen (Anzeige und Aufzeichnung der Register- und Speicherbelegungen (in Echtzeit), Breakpoints, schrittweise Abarbeitung, etc.)

→ siehe Vorlesung Rechnerentwurf

Emulationsadapter



Struktur des Emulationsadapters



PZ: Prozessor vom Typ des Zielsystemprozessors (bildet diesen nach)

BS: Busschalter: schaltet in Abhängigkeit von der Seitensteuerung den Prozessorbus auf das Zielsystem oder den ES

ES: Emulationsspeicher Bildet den bzw. Teile des Zielsystemspeichers nach (Adressbereichs-abhängig, z.B. ROM des Zielsystems als ladbarer RAM im ES)

SS: Schaltet nach vorgegebenen Adressbereichen den Bus vom PZ auf das Zielsystem bzw. den ES (im Zeitraster von Maschinenzyklen)

zB. Eingabebefehl nach Register

(Befehle stehen im ES, EA ist im Zielsystem)

Befehl lesen -> Busschalter auf ES

Operand lesen (Eingabe) -> Busschalter auf Zielsystem

HP: Haltepunktlogik überwacht alle Bussignale in Echtzeit, erzeugt Triggersignal in Abhängigkeit der überwachten Signale,

dieses kann zum Anhalten des Programmes genutzt werden.

(entspricht der Triggerlogik eines Logikanalysators)

Trace: Speicher, zyklisch, zum Aufzeichen der Bussignale in Maschinenzyklen: dient zur Auswertung des Programmverlaufs vor dem Haltepunkt (Vorgeschichte), evtl. mit gewissem Anteil danach (Nachgeschichte) (entspricht dem Signalspeicher im Logikanalysator)

Kommunikationslogik: realisiert die Kommunikation mit dem Wirtsrechner

Wirtsrechner benötigt spezielles Dialogprogramm zur Bedienung des Emulationsadapteres zu Testzwecken (spezieller Debugger)

Funktionen ähnlich Remote Debugging + Zusatzfunktionen

Vorteile:

echtzeitfähig

Zugriff auf EA im Zielsystem

keine Zusatzhard- und Software im Zielsystem

Test auch mit noch unvollständiger bzw.nicht voll funktionsfähiger Hardware möglich, auch zum Harwaretest geeignet

Nachteile:

teuer (30000 … 50000 EUR)

viele Funktionsgruppen sind Zielprozessorabhängig -> andrer Prozessor stark veränderter oder anderer Emulationsadapter

evtl. Adaption des Prozessorsteckplatzes konstruktiv schwierig

geht nur Prozessoren, die von außen (über den Systembus) vollständig beobachtbar sind

Bsp. Prozessor hat pipeline, die spekulative Sprungvorhersage benutzt.

evtl. werden nicht alle über den Bus transportierten Befehle ausgeführt.

Einchipcontroller haben typisch den Speicher- und EA-Bus nicht als Anschlüsse.

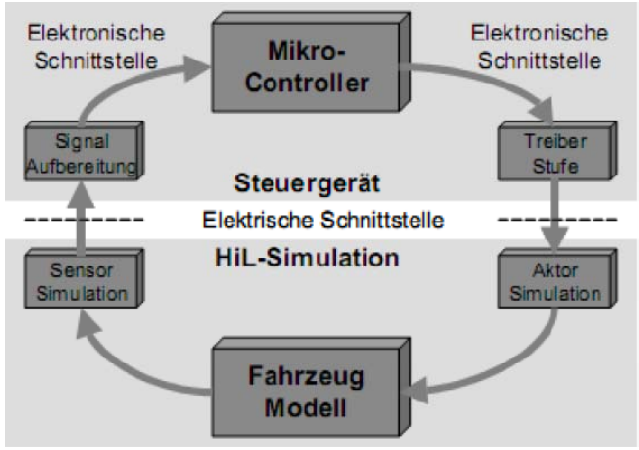
Ein Ausweg: Spezielle Einchipcontrollertypen mit herausgeführten Bussignalen.

Softwaretechnische Aspekte

1. Vorgehensmodelle
2. Echtzeit Programmierung
3. Testen
    1. Allgemeines
    2. Remote Debugging
    3. In Circuit Emulation
    4. Hardware in the loop / Software in the loop
    5. JTAG
4. Autosar

Hardware in the loop

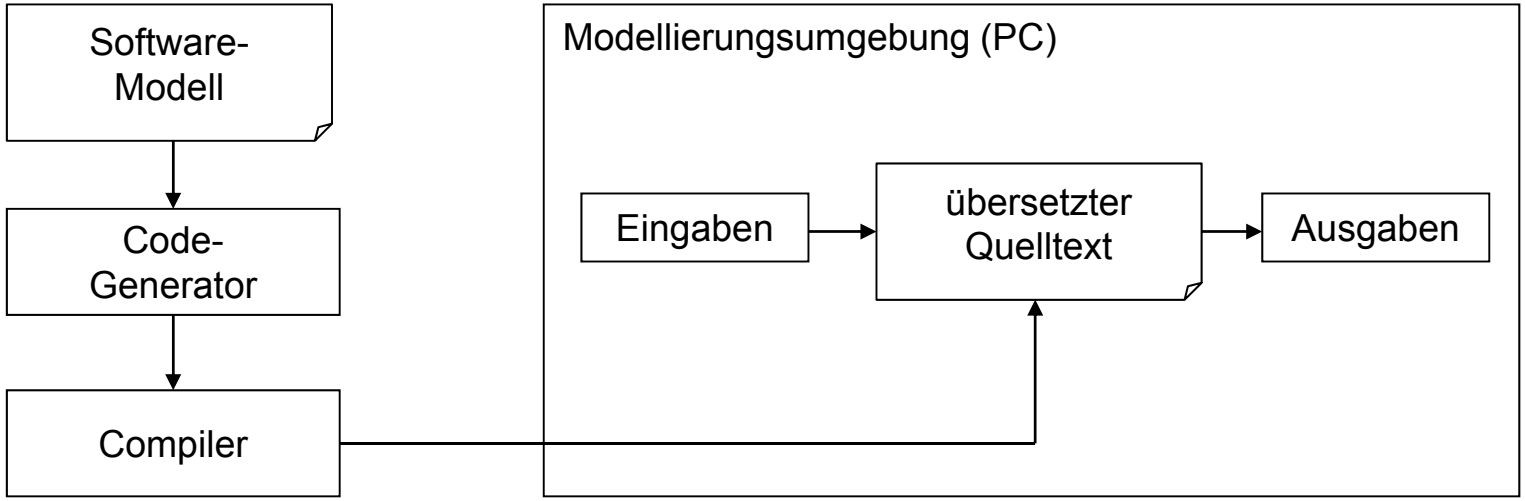
- Nachbildung der realen Umgebung des Systems durch HiL-Simulator zum Test des Steuergerätes
- HiL-Simulator
  - Modellbasierte Simulation der Umgebung
  - Erzeugt Sensordaten als Eingabe
  - Simuliert die Reaktionen auf Ausgaben
  - Erlaubt wiederkehrende Abläufe zu simulieren
- Simulation in Echtzeit
- Beispiel: Testen von Steuergeräten im Automobilbereich



Quelle: Dissertation „Modellbasierter Hardware-in-the-Loop Test von eingebetteten elektronischen Systemen“ , Dipl.-Ing. Bernhard Spitzer, Mannheim, 2001

Software in the loop

- Kein Einsatz besonderer Hardware
- Umgebung wird analog zu „Hardware in the loop“ simuliert
- Das erstellte Modell der Software wird in ausführbaren Code umgewandelt und auf dem Entwicklungsrechner statt auf der eigentlichen Zielplattform ausgeführt



Softwaretechnische Aspekte

1. Vorgehensmodelle
2. Echtzeit Programmierung
3. Testen
    1. Allgemeines
    2. Remote Debugging
    3. In Circuit Emulation
    4. Hardware in the loop
    5. JTAG
4. **Autosar**

Grundidee: Einbau von Testhardware während des Entwurfs mit in den Serienprozessor (vergleichsweise wenig Logik)

- Zusätzliches serielles Interface zur Bedienung dieser Testharware: JTAG: spezielles serielles Interface (5 Signale) für Testzwecke

Vorteile:

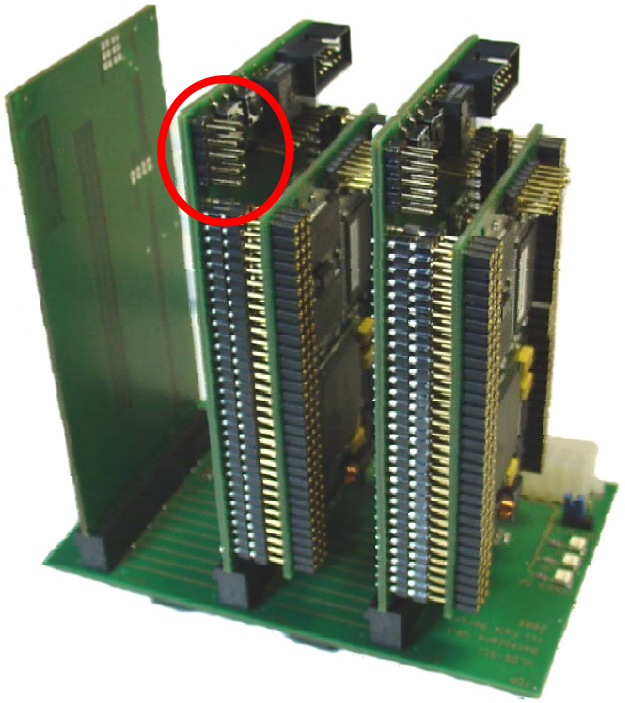
- Echtzeittest möglich
- keine Zusatzhardware
- Kosten etwas geringer als vollständiger Emulationsadapter
- Test im Zielsystem möglich
- Ankopplung konstruktiv unproblematisch

Nachteile:

- Funktionsumfang etwas geringer als Emulationsadapter
- Für von außen nicht vollständig beobachtbare Prozessoren meist die einzige Möglichkeit.

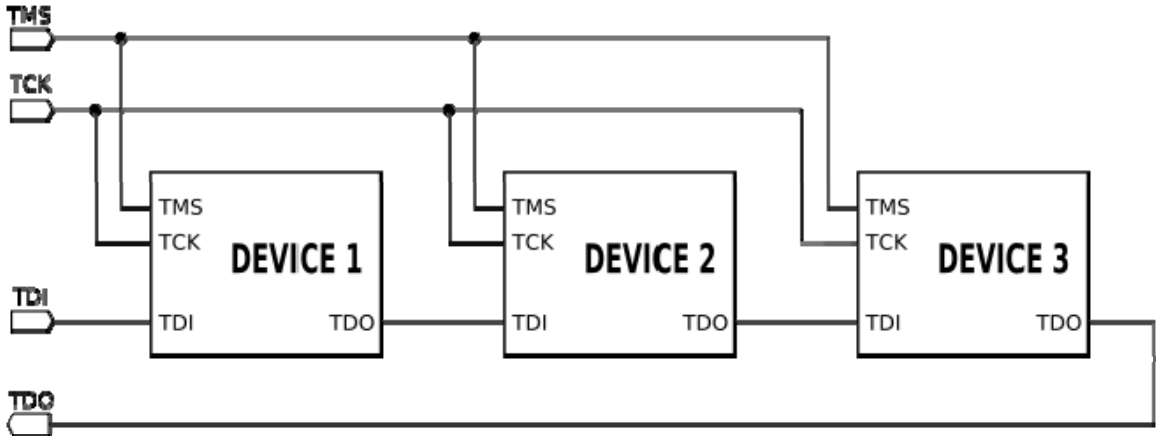
JTAG

- „Joint Test Action Group“, IEEE Standard 1149.1
- Beobachtung der Eingänge und internen Zustände
- Setzen definierter Ausgangswerte
- Mehrere IC können an eine Schnittstelle angeschlossen werden
- Herstellerspezifische Erweiterungen, z.B. für Remote Debugging

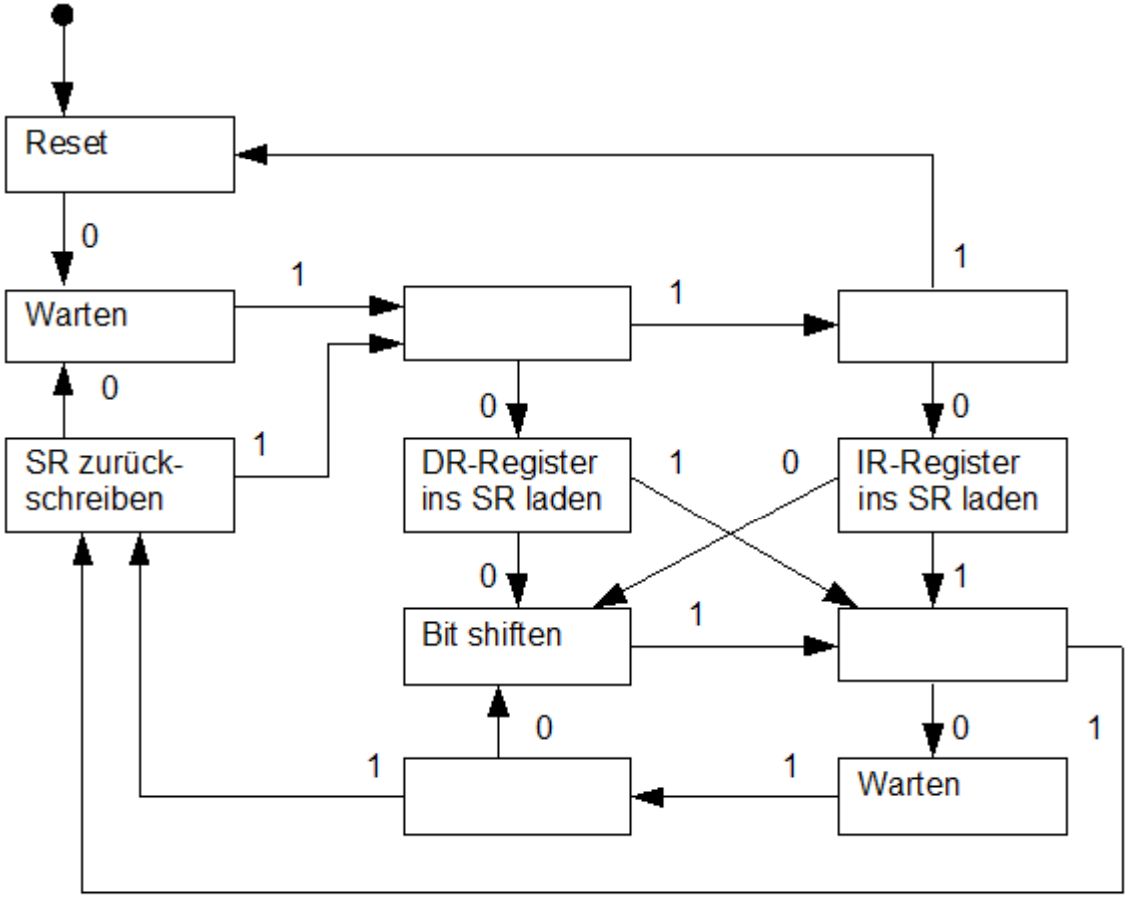


JTAG

- Aufbau als Kette aus Schieberegistern, serielles Lesen & Schreiben
- Signale
  - Test Data In (TDI)
  - Test Data Out (TDO)
  - Test Mode Select (TMS)
  - Test Clock (TCK)
  - Test Reset (TRST), optional



Auswahl des Registers über das TMS Signal



Legende
SR – Schieberegister
DR – Datenregister
IR – Instruction Register

JTAG

- Register (herstellerspezifisch)
  - Instruction Register (IR)
    Auswahl des Datenregisters, Ausführen herstellerspezifischer Befehle
  - Datenregister (DR)
    - Boundary Scan Register
      Repräsentiert die PINS des IC
    - IDCODE Register
      Enthält Herstellercode und Produkttyp zur Identifizierung des IC
    - Bypass Register
      Dummy-Register, wird verwendet, wenn bei mehreren ICs in der Schiebekette nur ein IC gelesen werden soll
    - Sonstige Register
      Weitere herstellerspezifische Datenregister, die beispielsweise interne Zustände enthalten und zum Debugging verwendet werden

Softwaretechnische Aspekte

1. Vorgehensmodelle
2. Echtzeit Programmierung
3. Testen
    1. Allgemeines
    2. Remote Debugging
    3. In Circuit Emulation
    4. Hardware in the loop / Software in the loop
    5. JTAG
4. Autosar

Autosar

AUTOSAR (AUTomotive Open System ARchitecture) is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers.

http://www.autosar.org/

Autosar: key features (1)

Modularity and configurability

- Definition of a modular software architecture for automotive electronic control units
- Consideration of HW dependent and HW independent SW modules
- Integration of SW modules provided by different suppliers to increase the functional reuse
- Transferability of functional SW-modules within a particular E/E-system at least at the final software linking process
- Resource optimized configuration of the SW infrastructure of each ECU depending on the function deployment
- Scalability of the E/E-system across the entire range of vehicle product lines

Autosar: key features (2)

Standardized interfaces

- Standardization of different APIs to separate the AUTOSAR SW layers
- Facilitate encapsulation of functional SW-components
- Definition of the data types of the SW-components to be AUTOSAR compliant
- Identify basic SW modules of the SW infrastructure and standardize their interfaces

Autosar: key features (3)

Runtime Environment

- AUTOSAR runtime environment to provide inter- and intra-ECU communication across all nodes of a vehicle network
- Runtime environment is located between the functional SW-components and the basic SW-modules
- All entities connected to the AUTOSAR RTE must comply with the AUTOSAR specification
- Enables the easy integration of customer specific functional SW-modules

- To achieve the technical goals modularity, scalability, transferability and re-usability of functions AUTOSAR will provide a common software infrastructure for automotive systems of all vehicle domains based on standardized interfaces for the different layers shown in the top right image.

- **Modularity**
  Modularity of automotive software elements will enable tailoring of software according to the individual requirements of electronic control units and their tasks.
- **Scalability**
  Scalability of functions will ensure the adaptability of common software modules to different vehicle platforms to prohibit proliferation of software with similar functionality.

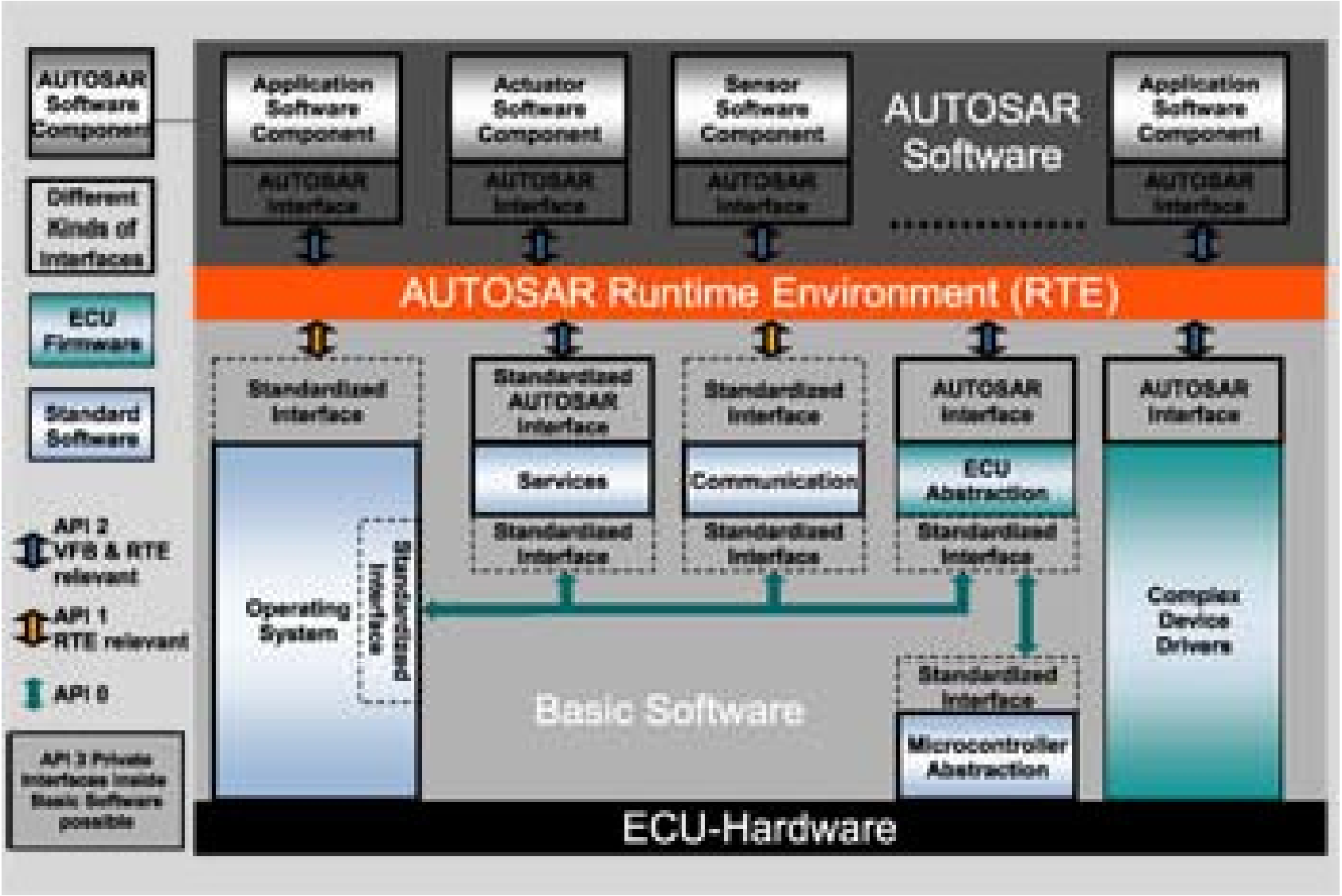
- **Transferability**
  Transferability of functions will optimize the use of resources available throughout a vehicle's electronic architecture.

- **Re-usability**
  Re-usability of functions will help to improve product quality and reliability and to reinforce corporate brand image across product lines.

- **Standardized interfaces**
  Standardization of functional interfaces across manufacturers and suppliers and standardization of the interfaces between the different SW-Layers is seen as a basis for achieving the technical goals of AUTOSAR.

- The AUTOSAR Software (the layer above AUTOSAR Runtime Environment) consists of AUTOSAR Software Components that are mapped on the ECU. All interaction between AUTOSAR Software Components and Atomic Software Components is routed through the AUTOSAR Runtime Environment. The AUTOSAR Interface assures the connectivity of software elements surrounding the AUTOSAR Runtime Environment.

Structure of the software for an ECU



Software Components

- The AUTOSAR Software Components encapsulate an application which runs on the AUTOSAR infrastructure. The AUTOSAR Software Components have well-defined interfaces, which are described and standardized within AUTOSAR.

Software Components

- Software Component Description

  For the interfaces as well as other aspects needed for the integration of the AUTOSAR Software Components, AUTOSAR provides a standard description format, i.e. the Software Component Description.

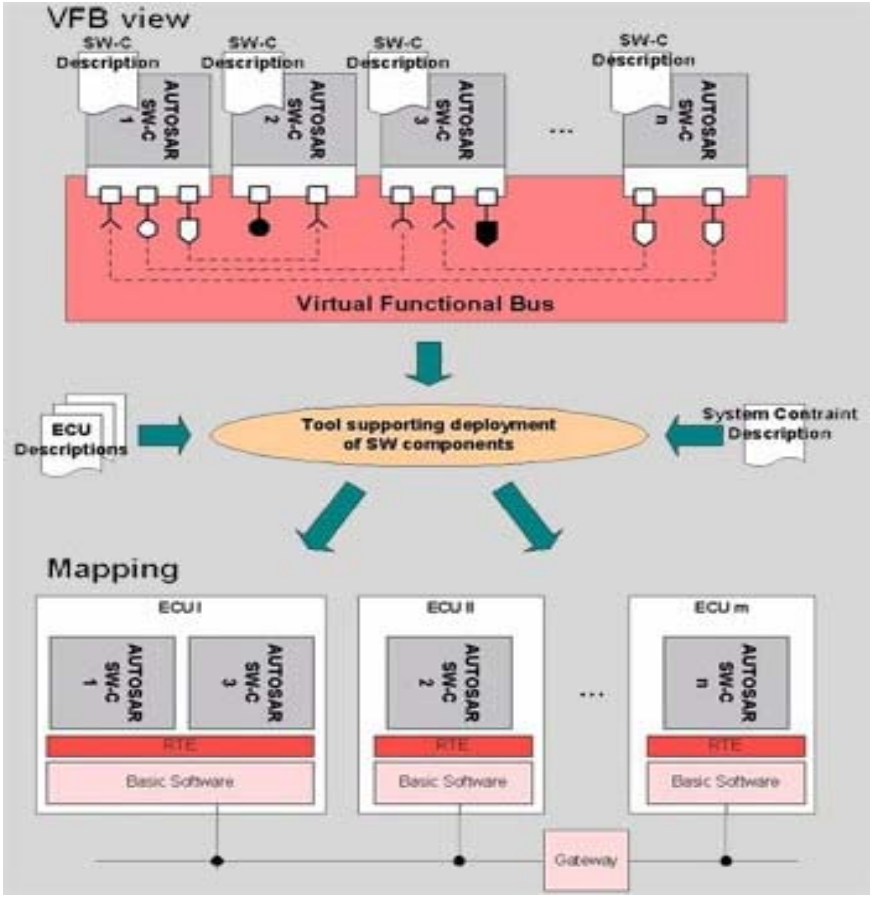
Virtual Functional Bus (VFB)

- Virtual Functional Bus (VFB)
  The VFB is the sum of all communication mechanisms and essential interfaces to the basic software provided by AUTOSAR on an abstract, i.e. technology independent, level. When the connections between AUTOSAR Software Components for a concrete system are defined, the VFB will allow a virtual integration of them in an early development phase.

- System Constraint and ECU Descriptions

  In order to integrate AUTOSAR Software Components into a network of ECUs, AUTOSAR provides description formats for the complete system as well as for the resources and configuration of the single ECUs. These descriptions are kept independent of the Software Component Descriptions.

Mapping on ECUs



Following the AUTOSAR Methodology, the E/E architecture is derived from the formal description of software and hardware components.

Fengler, Zimmermann 03-2009

Mapping on ECUs

- Mapping on ECUs

  AUTOSAR defines the methodology and tool support needed to bring the information of the various description elements together in order to build a concrete system of ECUs. This includes especially the configuration and generation of the Runtime Environment and the Basic Software on each ECU.

- Runtime Environment (RTE)
  From the viewpoint of the AUTOSAR Software Component, the RTE implements the VFB functionality on a specific ECU. The RTE can however delegate this task to the Basic Software as far as possible.

- Basic Software
  The Basic Software provides the infrastructural functionality on an ECU.

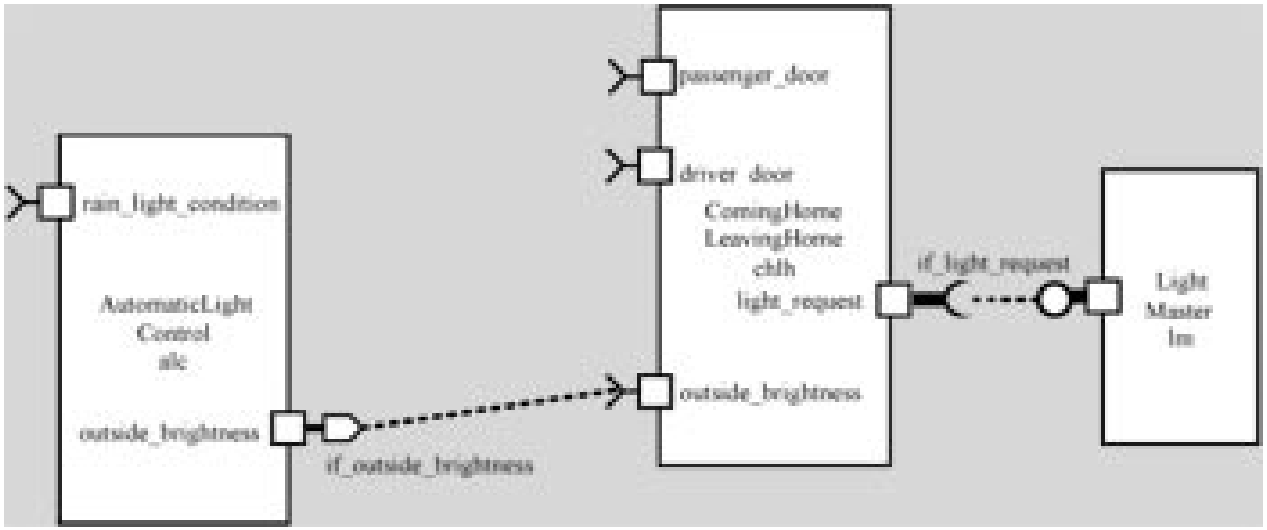
Software Component

- The AUTOSAR Software Component implementation is independent from the infrastructure
- A fundamental design concept of AUTOSAR is the separation between:
- application and
- infrastructure

An application in AUTOSAR consists of interconnected "AUTOSAR Software Components".

Software Components which are interconnected

73



The image shows an application consisting of three AUTOSAR Software Components which are interconnected by several "connectors".

- Each AUTOSAR Software Component encapsulates part of the functionality of the application. AUTOSAR does not prescribe how large the AUTOSAR Software Components are. Depending on the requirements of the application domain an AUTOSAR Software Component might be a small, reusable piece of functionality (such as a filter) or a larger block encapsulating an entire automotive functionality.

- However, the AUTOSAR Software Component is a so-called "Atomic Software Component". It cannot be distributed over several AUTOSAR ECUs. Consequently, each instance of an AUTOSAR Software Component that should be present in a vehicle is assigned to one ECU.

Software Component Description

The AUTOSAR Software Component Description contains the following information:

- The operations and data elements that the software component provides and requires.
- The requirements which the software component has on the infrastructure.
- The resources needed by the software component (memory, CPU-time, etc.).
- Information regarding the specific implementation of the software component.

The structure and format of this AUTOSAR Software Component description is called the "software component template".

Software Componemt Implementation

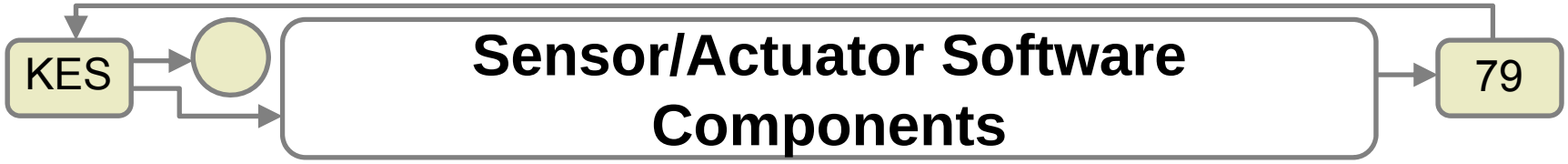
The AUTOSAR Software Componemt Implementation is independent from the infrastructure

An implementation of an AUTOSAR Software Component fundamentally is independent from: The type of microcontroller of the ECU on which the AUTOSAR Software Component is mapped.

- The type of ECU on which the AUTOSAR Software Component is mapped.
- The AUTOSAR infrastructure takes care of providing the software component with a standardized view on the ECU hardware (such as the ECU input/output periphery)

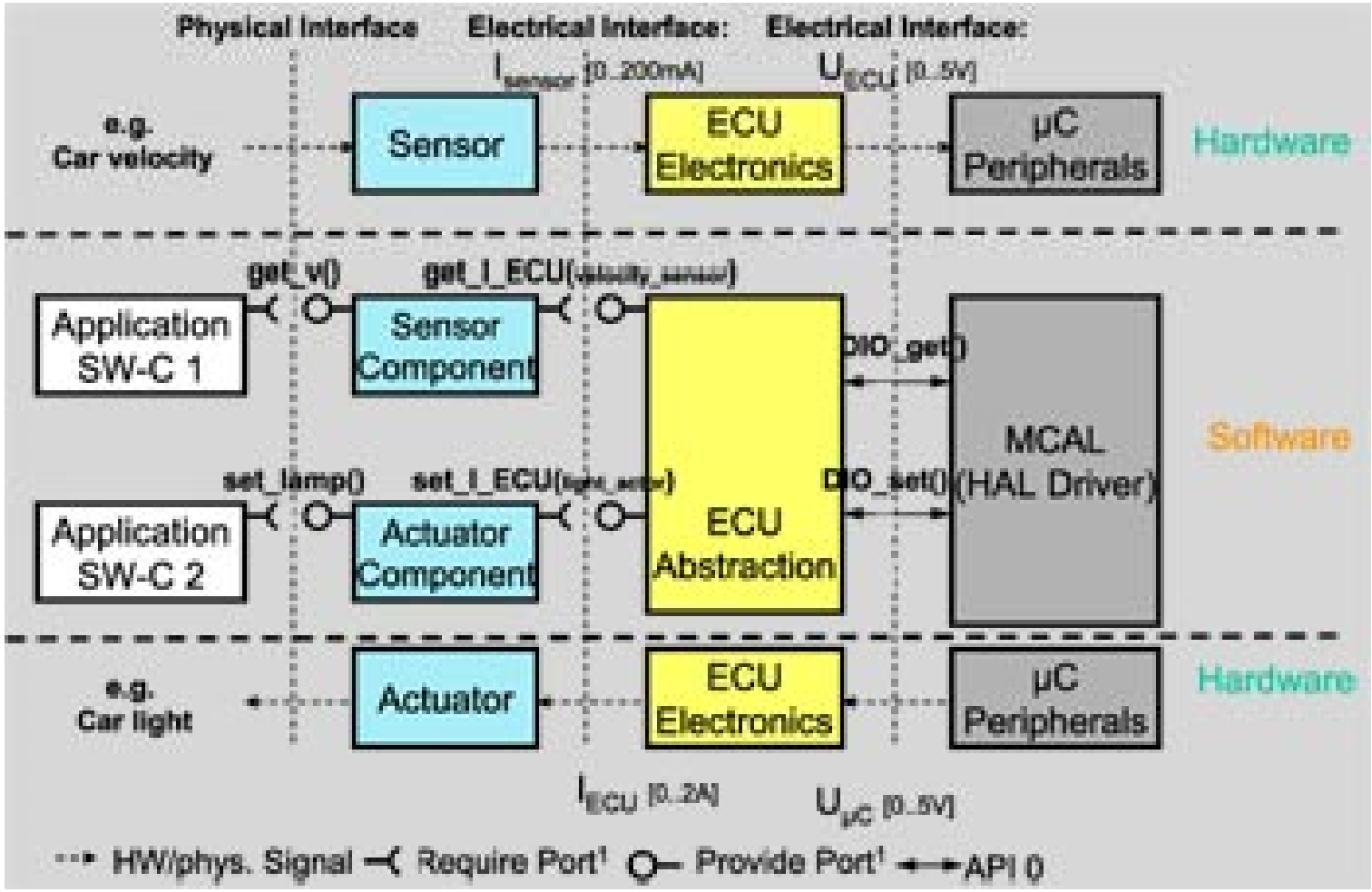
Software Componemt Implementation

- The location of the other AUTOSAR Software Components with which the software component interacts. The software component description precisely describes the data or services that the component provides or requires. The component does not need to know if these services or data elements are provided from components on the same ECU or are coming from components that run on a different ECU. The implementation of the software component therefore also is network technology independent.
- The number of times a software component is instantiated in a system or within one ECU.



- Sensor/Actuator Software Components are special AUTOSAR Software Components which encapsulate the dependencies of the application on specific sensors or actuators.

Typical Conversion Process



Typical Conversion Process

The image illustrates the typical conversion process from physical signals to software signals (e.g. car velocity) and back (e.g. car light). The AUTOSAR infrastructure takes care of hiding the specifics of the microcontroller and the ECU electronics.

Sensor/Actuator Software Components

The AUTOSAR infrastructure does not, however, hide the specifics of sensors and actuators. The dependencies on a specific sensor and/or actuator are dealt with in a "Sensor/Actuator Software Component", which is a special kind of "AUTOSAR Software Component." A Sensor/Actuator Software Component is independent of the ECU on which it is mapped, but it is dependent on the specific sensor and/or actuator for which it is designed. For example, a Sensor Component typically inputs a software representation of the electrical signal present at an input pin of the ECU (e.g. a current produced by the sensor) and outputs the physical quantity measured by the sensor (e.g. the current car speed). Typically, because of performance and timing issues, such components will need to run on the ECU to which the sensor/actuator is physically connected.

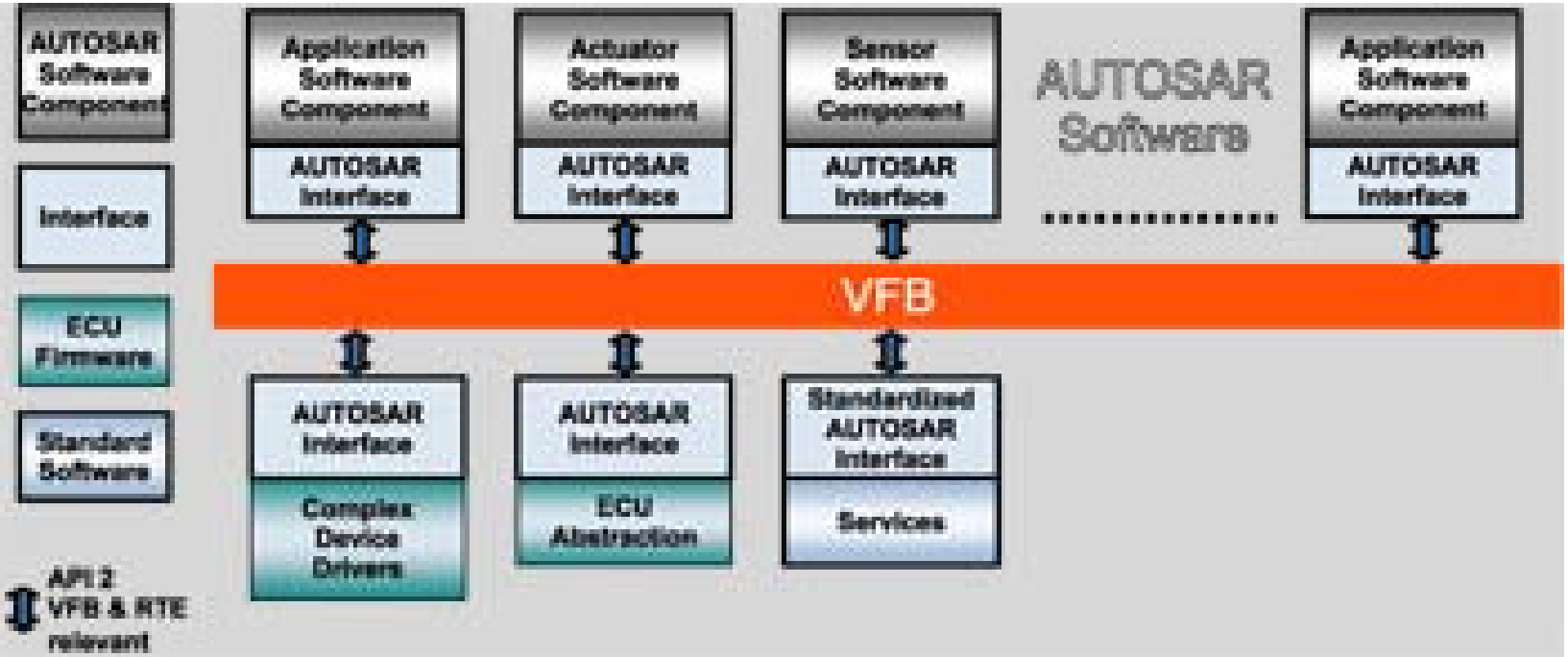
Virtual Functional Bus (VFB)

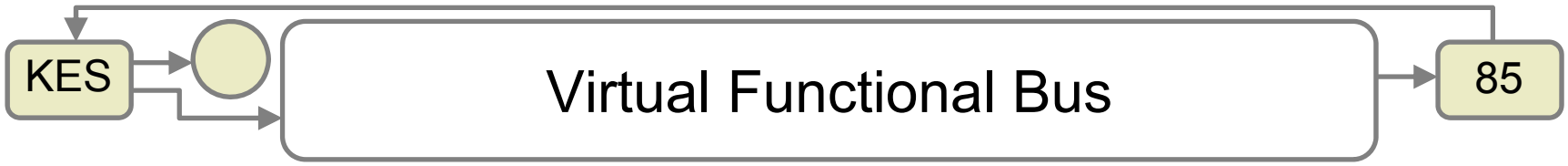
The Virtual Functional Bus (VFB)
In order to fulfill the goal of relocatability, AUTOSAR Software Components are implemented independently from the underlying hardware. The independence is achieved by providing the virtual functional bus as a means for a virtual hardware and mapping independent system integration. This enables a virtual integration of AUTOSAR Software Components so that parts of the integration process of automotive software can be done in much earlier design phases compared to today's development processes.

KES

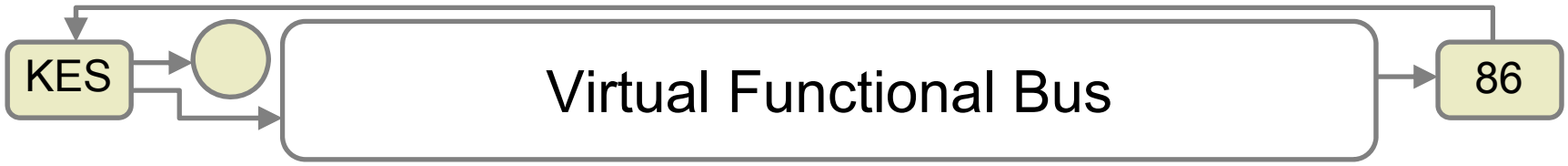
7.

Virtual Functional Bus

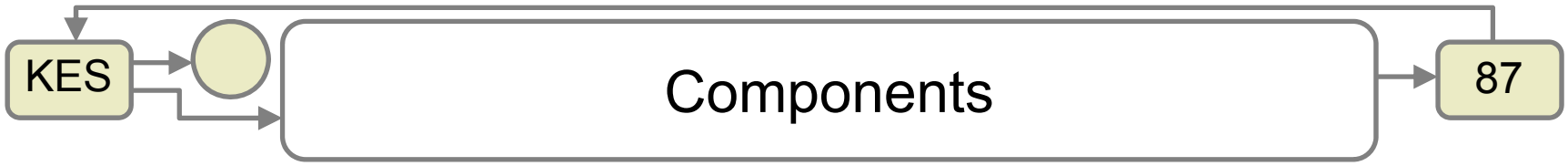




Representations of Atomic Software Components and AUTOSAR Services connected to the Virtual Functional Bus



The virtual functional bus is the abstraction of the AUTOSAR Software Components interconnections of the entire vehicle. The communication between different software components and between software components and its environment (e.g. hardware driver, OS, services, etc.) can be specified independently of any underlying hardware. The functionality of the VFB is provided by well-defined communication patterns. Services and communication protocols are implemented by Basic Software. An AUTOSAR Service offers extended functionality to the users of the VFB just as a standard library adds extended functionality for the users of a programming language.



In order to reuse this extended functionality for all AUTOSAR Software Components, it is essential that the interfaces to AUTOSAR Services are standardized. From the VFB view, ports of AUTOSAR Software Components, Complex Device Drivers, the ECU Abstraction and AUTOSAR Services can be connected. Complex Device Drivers, the ECU Abstraction and AUTOSAR Services are part of the Basic Software. Whereas the AUTOSAR Service Interfaces are standardized, the Complex Device Drivers and the ECU Abstraction are ECU specific.

Components

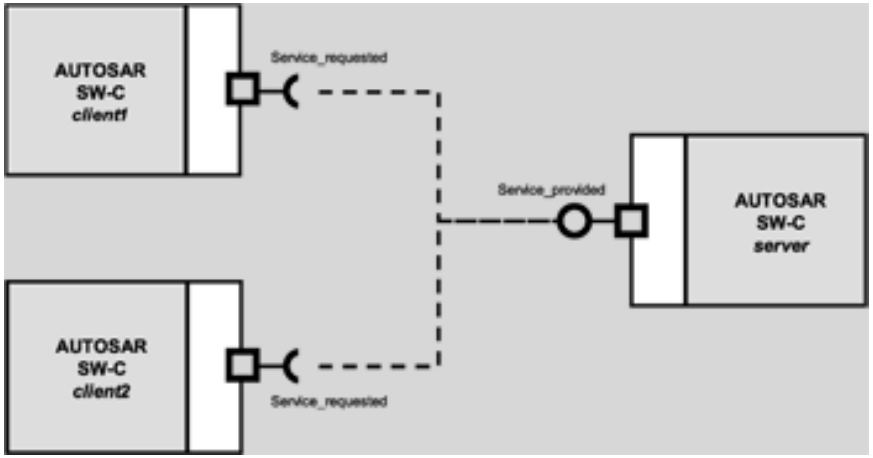
The central structural element in AUTOSAR is the component. A component has well-defined ports, through which it interacts with other components. A port always belongs to exactly one component. The AUTOSAR Interface concept defines the services or data that are provided on or required by a port of a component. The AUTOSAR Interface can either be a Client-Server Interface (defining a set of operations that can be invoked) or a Sender-Receiver Interface, which allows the usage of data-oriented communication mechanisms over the VFB.

A port is either a PPort or an RPort. A PPort provides an AUTOSAR Interface while an RPort requires one.

When a PPort of a component provides an interface, the component to which the port belongs provides an implementation of the operations defined in the Client-Server Interface respectively generates the data described in a data-oriented Sender-Receiver Interface.

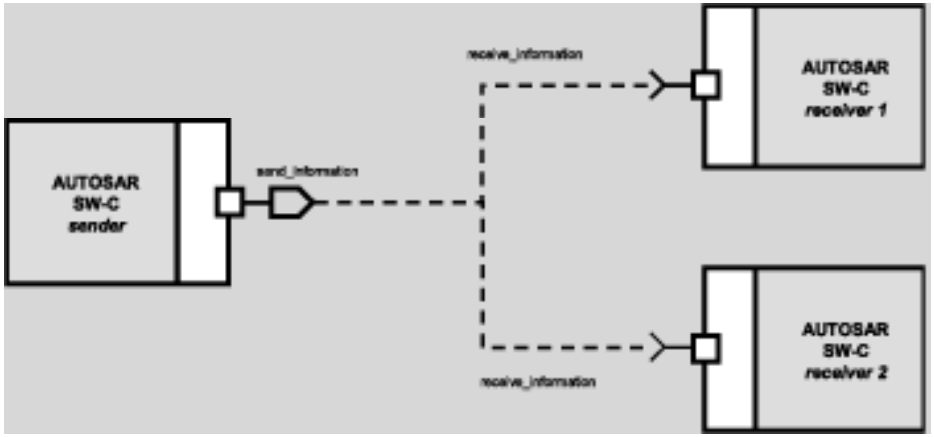
When an RPort of a component requires an AUTOSAR Interface, the component can either invoke the operations when the interface is a Client-Server Interface or alternatively read the data elements described in the Sender-Receiver Interface.

Communication



Client-server communication in the VFB view.

Communication



Data distribution by asynchronous non-blocking communication in the VFB view.

Communication

Client-Server Communication

A widely used communication pattern in distributed systems is the client-server pattern, in which the server is a provider of a service and the client is a user of a service.

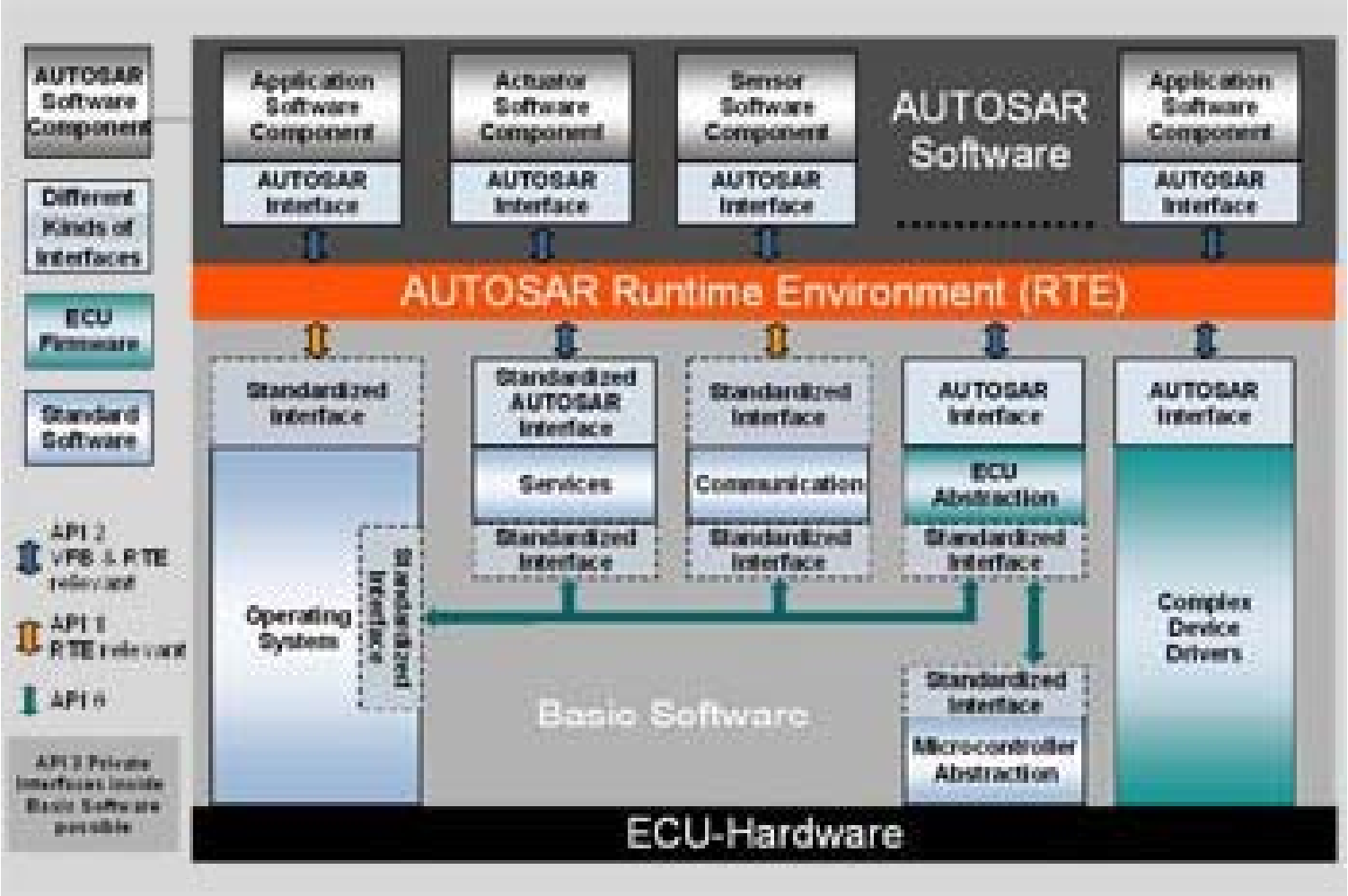
The client initiates the communication, requesting that the server performs a service, transferring a parameter set if necessary. The server waits for incoming communication requests from a client, performs the requested service, and dispatches a response to the client?s request. The direction of initiation is used to categorize whether an AUTOSAR Software Component is a client or a server. A single component can be both a client and a server, depending on the software realization.

The client can be blocked (synchronous communication) or non-blocked (asynchronous communication), respectively, after the service request is initiated until the response of the server is received. The image gives an example how client-server communication for a composition of three software components and two connections is modeled in the VFB view.

Communication

Sender-Receiver Communication
The sender-receiver pattern gives solution to the asynchronous distribution of information, where a sender distributes information to one or several receivers. The sender is not blocked (asynchronous communication) and neither expects nor gets a response from the receivers (data or control flow), i.e. the sender just provides the information and the receivers decides autonomously when and how to use this information. It is the responsibility of the communication infrastructure to distribute the information.
The sender component does not know the identity or the number of receivers to support transferability and exchange of AUTOSAR Software Components. The image illustrates an example how sender-receiver communication is modeled in the AUTOSAR VFB view.

Schematic view of AUTOSAR ECU software architecture.



Fengler, Zimmermann 03-2009

The AUTOSAR Software (the layer above AUTOSAR Runtime Environment) consists of AUTOSAR Software Components that are mapped on the ECU. All interaction between AUTOSAR Software Components and Atomic Software Components is routed through the AUTOSAR Runtime Environment. The AUTOSAR Interface assures the connectivity of software elements surrounding the AUTOSAR Runtime Environment.

AUTOSAR Runtime Environment

At system design level, (i.e. when drafting a logical view of the entire system irrespective of hardware) the AUTOSAR Runtime Environment (RTE) acts as a communication center for inter- and intra-ECU information exchange. The RTE provides a communication abstraction to AUTOSAR Software Components attached to it by providing the same interface and services whether inter-ECU communication channels are used (such as CAN, LIN, FlexRay, MOST, etc.) or communication stays intra-ECU.
As the communication requirements of the software components running on top of the RTE are application dependent, the RTE needs to be tailored, partly by ECU-specific generation and partly by configuration. Thus, the resulting RTE will differ between one ECU and another.

Basic Software

Basic Software is the standardized software layer, which provides services to the AUTOSAR Software Components and is necessary to run the functional part of the software. It does not fulfill any functional job itself and is situated below the AUTOSAR Runtime Environment. The Basic Software contains standardized and ECU specific components. The earlier include:

Services

System services such as diagnostic protocols; NVRAM, flash and memory management

Communication

Communication Framework (e.g. CAN, LIN, FlexRay...), I/O management, Network management

Basic Software

Operating System

As AUTOSAR aims at an architecture that is common for all vehicle domains it will specify the requirements for an AUTOSAR Operating System. The following requirements shall be seen as examples of such:

the OS is configured and scaled statically
is amenable to reasoning of real-time performance
provides a priority-based scheduling
provides protective functions at run-time
runs on low-end controllers and without external resources

AUTOSAR allows the inclusion of proprietary OSs in Basic Software components. To make the interfaces of these components AUTOSAR compliant, the proprietary OS must be abstracted to an AUTOSAR OS. The standard OSEK OS (ISO 17356-3) is used as the basis for the AUTOSAR OS.

Basic Software

Microcontroller Abstraction Microcontroller Abstraction

Access to the hardware is routed through the Microcontroller Abstraction layer (MCAL) to avoid direct access to microcontroller registers from higher-level software.

MCAL is a hardware specific layer that ensures a standard interface to the components of the Basic Software. It manages the microcontroller peripherals and provides the components of the Basic Software with microcontroller independent values. MCAL implements notification mechanisms to support the distribution of commands, responses and information to different processes. Among others it can include: Digital I/O (DIO)

Analog/Digital Converter (ADC)

Pulse Width (De)Modulator (PWM, PWD)

EEPROM (EEP)

Flash (FLS)

Capture Compare Unit (CCU)

Watchdog Timer (WDT)

Serial Peripheral Interface (SPI)

I^2C Bus (IIC)

Basic Software

ECU specific components are:

ECU Abstraction

The ECU Abstraction provides a software interface to the electrical values of any specific ECU in order to decouple higher-level software from all underlying hardware dependencies.

Complex Device Driver (CDD)

The CDD allows a direct access to the hardware in particular for resource critical applications.

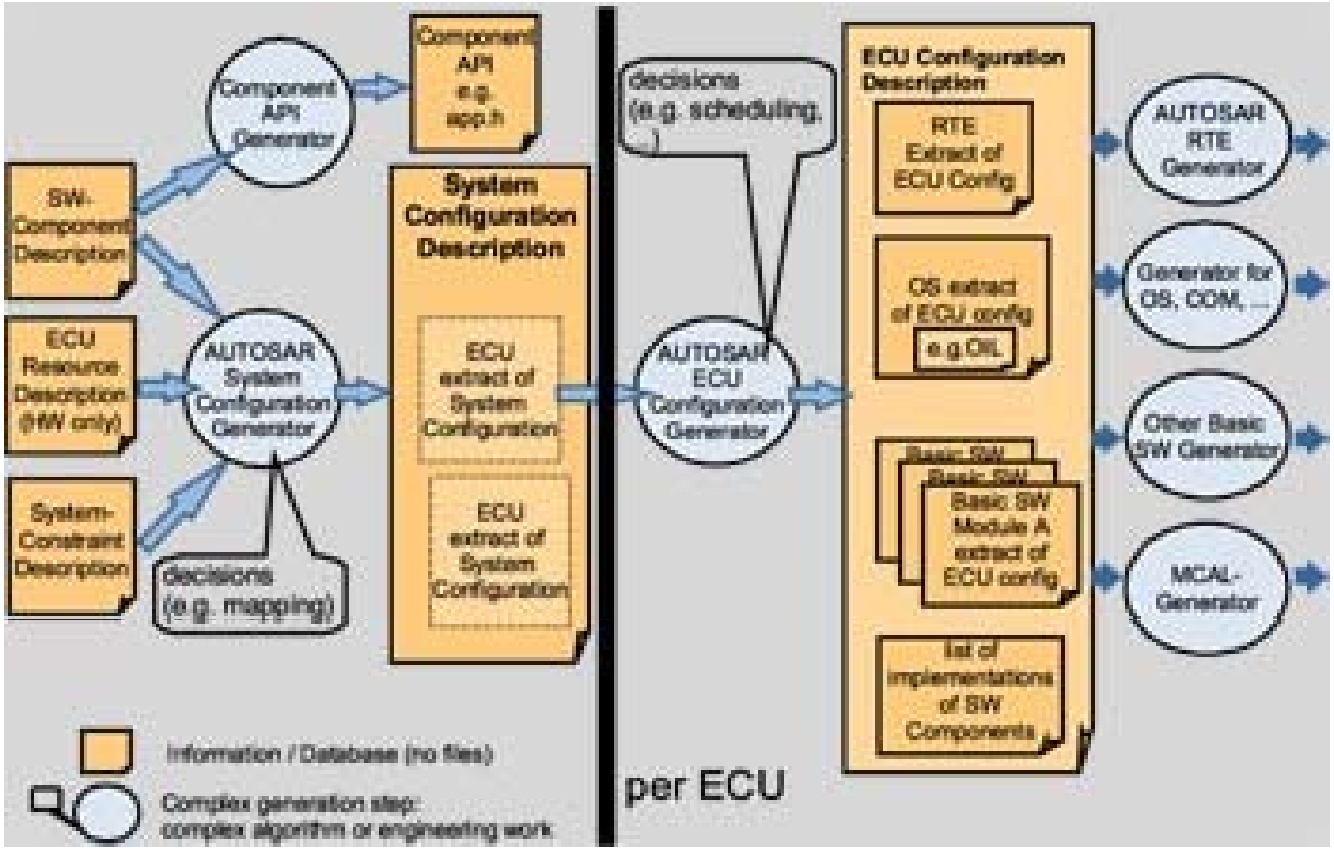
In the image three different types of interfaces are shown, e,g.-"AUTOSAR Interface", "Standardized AUTOSAR Interface" and "Standardized Interface". Please note that the boxes define the classification of the interfaces for the different modules, i.e. the interface boxes in the image shall not be regarded as separate modules or layers. The semantics of these classifications are as follows:

Classification of interfaces

- **AUTOSAR Interface**
  An AUTOSAR Interface describes the data and services required or provided by a component and is specified and implemented according to the AUTOSAR Interface Definition Language. An AUTOSAR Interface is partly standardized within AUTOSAR, e.g. it may include OEM specific aspects. The use of "AUTOSAR Interfaces" allows software components to be distributed among several ECUs. The RTEs on the ECUs will take care of making the distribution transparent to the software components.

- **Standardized AUTOSAR Interface**
  A Standardized AUTOSAR Interface is an AUTOSAR Interface standardized within the AUTOSAR project. Standardized Interface
- A software interface is called **Standardized Interface** if a concrete standardized API exists (e.g. OSEK COM Interface).

The AUTOSAR Methodology



The AUTOSAR Methodology

- Figure 1 shows the design steps to build a system with the AUTOSAR technology. Please note that this is an information flow, not an illustration of files. Terms of descriptions: System Configuration Description: includes all system information and the information that must be agreed between different ECUs
- System Configuration Extractor: extracts the information from the System Configuration Description needed for a specific ECU

- ECU extract:
  is the information from the System Configuration Description needed for a specific ECU ECU Configuration
- Description:
  all information that is local to a specific ECU the runnable software can be built from this information and the code of the software component