Teststrategien für komplexe Sicherungssysteme

Eine Herausforderung für Wissenschaft und Industrie

Dipl.-Ing. V. Knollmann
DLR

Dr.-Ing. I. Suwalski
Siemens TS RA

Deutsches Zentrum
DLR für Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

Themenüberblick

- **Warum testen?**
- **Der Prüfling**
- **Der Lösungsansatz**
- **Die Simulations- und Testumgebung**
- **Vom Konzept zur Umsetzung**
- **Zusammenfassung und Ausblick**

Warum testen?
Einige Fallbeispiele zur Motivation

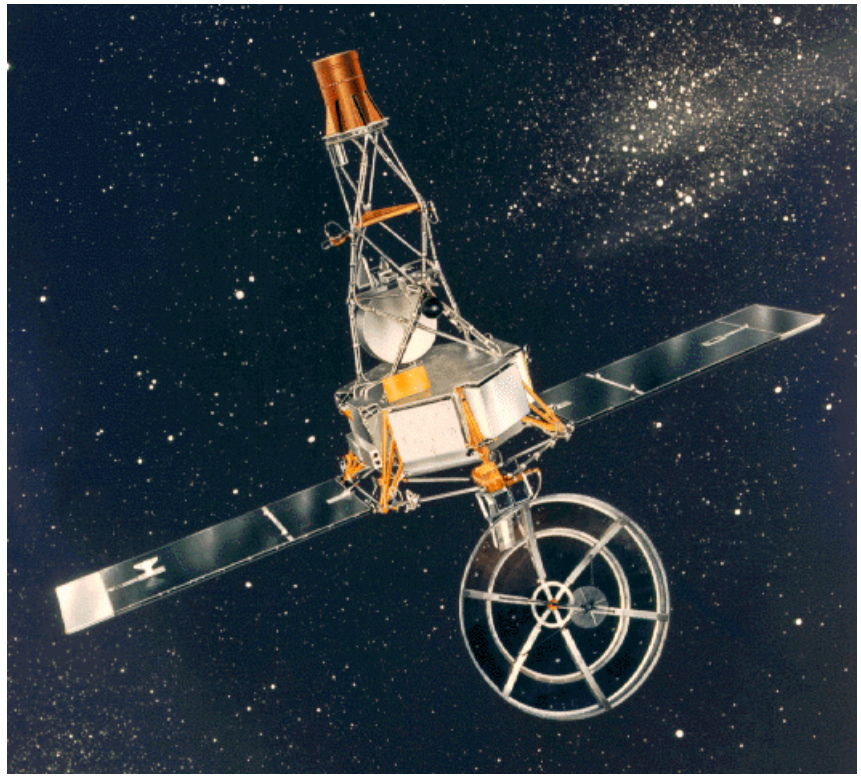
- Die Raumsonde Mariner 1 startete am 22.07.1962 …
- … und explodierte 293 Sekunden später.
- Ursache: Softwarefehler

```
DO 5 K = 1.3
```

statt

```
DO 5 K = 1,3
```

- Schaden: 18,5 Mio. $



Warum testen?
Einige Fallbeispiele zur Motivation

- Die Rakete Ariane 5 explodierte am 04.06.1996 nach 39 Sekunden
- Ursache: Softwarefehler
- Variablenüberlauf durch Weiterverwendung von Software aus Ariane 4
- Kosten: 500 Mio $

Warum testen?
Einige Fallbeispiele zur Motivation

- Der Mars Rover „Spirit“ hatte 20 Ausfalltage bei einer Lebensdauer von 90 Tagen
- Ursache: Softwarefehler
- Speicherüberlauf eines Flash Memory (256 MB)
- Kosten: 90 Mio. $

Warum testen?
Weitere Beispiele für gravierende Softwarefehler

- Stromausfall in den USA (Sommer 2003)
- Hartz-IV-Software
- Verzögerungen bei Einführung des Mautsystems
- Handyabstürze
- Rückrufaktionen bei PKW
- …

Warum testen?

Systemkomplexität und unentdeckte Fehler

Anzahl Codezeilen

Handy (einfach) 70.000	Sicherungstechnik 600.000	Windows 2000 27.000.000

Programmieraufwand (Zeilen pro Tag und Person)

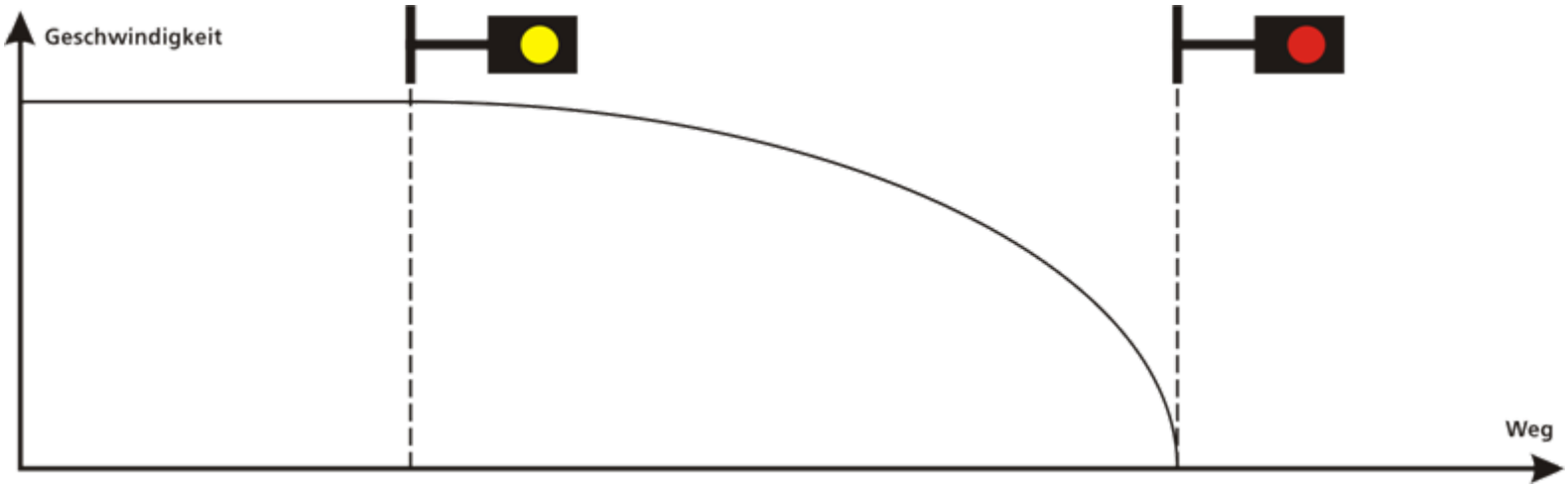
mittlere Prg. 50 Zeilen	kritische SW 1 – 2 Zeilen	große Prg. 10 – 20 Zeilen

Fehler pro 1000 Zeilen

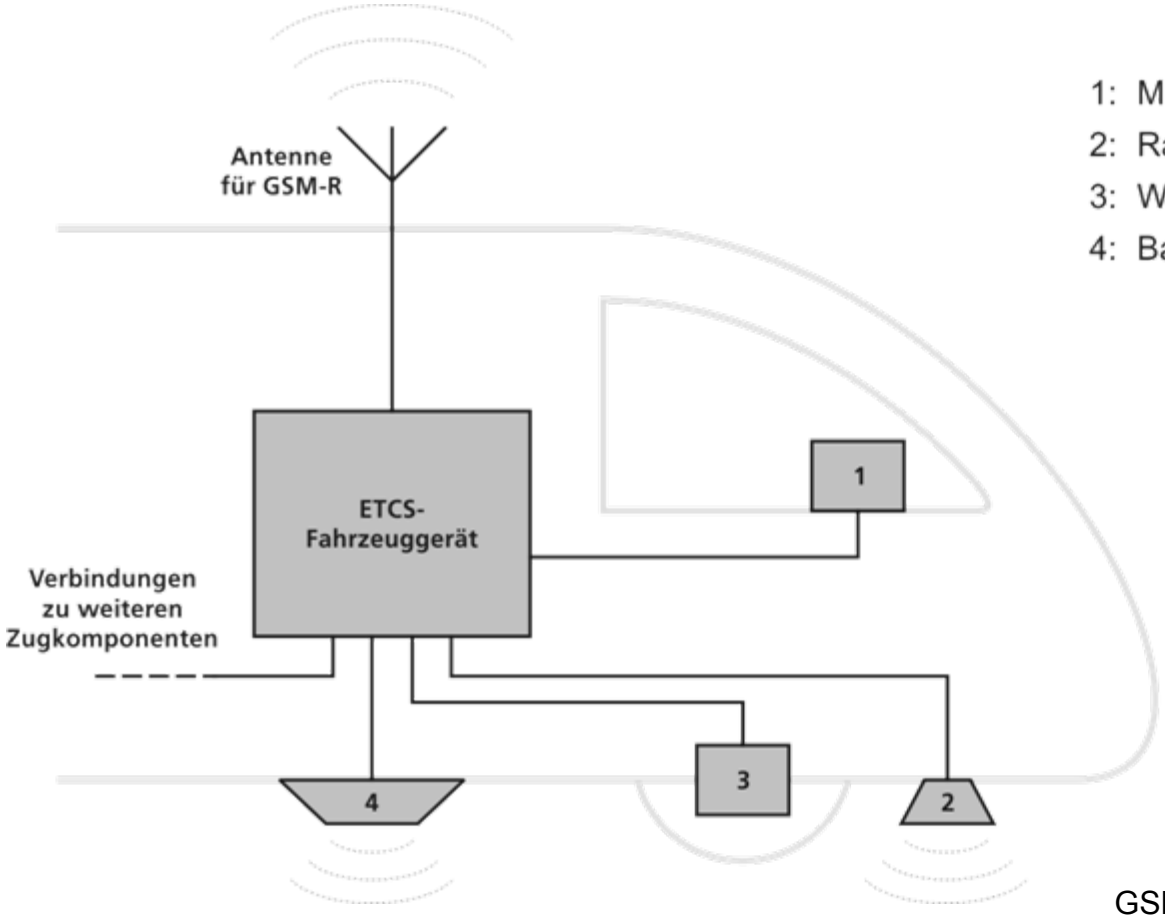
Einfache SW 25 – 30 Fehler	Sicherungstechnik 0,2 ≤ Fehler	Medizinische SW 0,2 Fehler

Der Prüfling
Aufgaben eines Zugsicherungssystems

- Sicherung gegen Geschwindigkeitsüberschreitung
- Einhalten der aktuellen Fahrerlaubnis, insbesondere:
  - Berechnung und Überwachung einer kontinuierlichen Bremskurve
  - Kommunikation mit der Streckenseite (Fahrerl., Profile, Position)
  - Stillstands- und Rückrollüberwachung



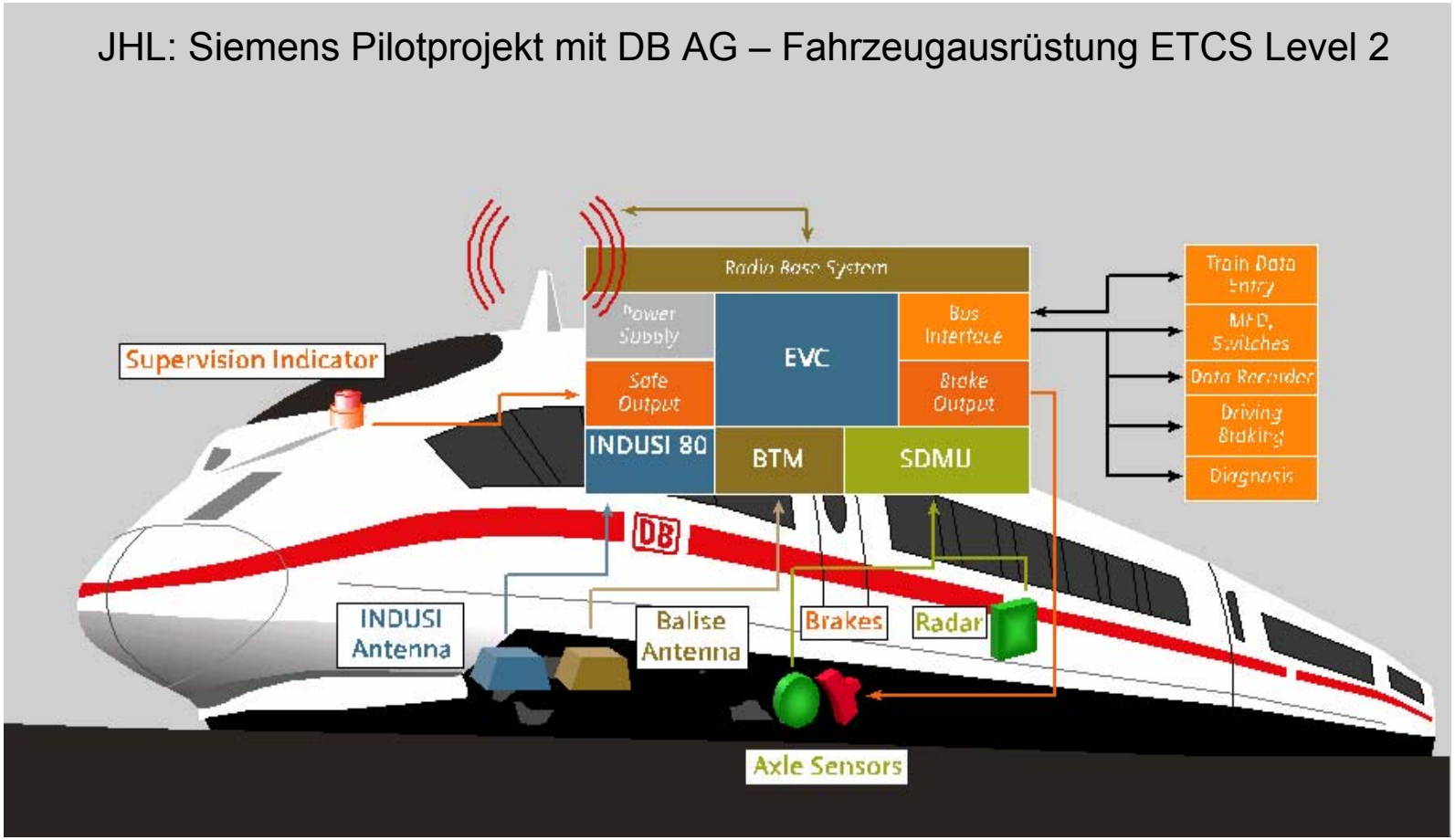
Der Prüfling
Vereinfachtes Blockdiagramm eines ETCS-Systems



1: MMI (Man Machine Interface)
2: Radar
3: Wegimpulsgeber
4: Balisenantenne

GSM-R: GSM for Railways
ETCS: European Train Control System

Der Prüfling
Komponenten eines ETCS-Fahrzeuggerätes



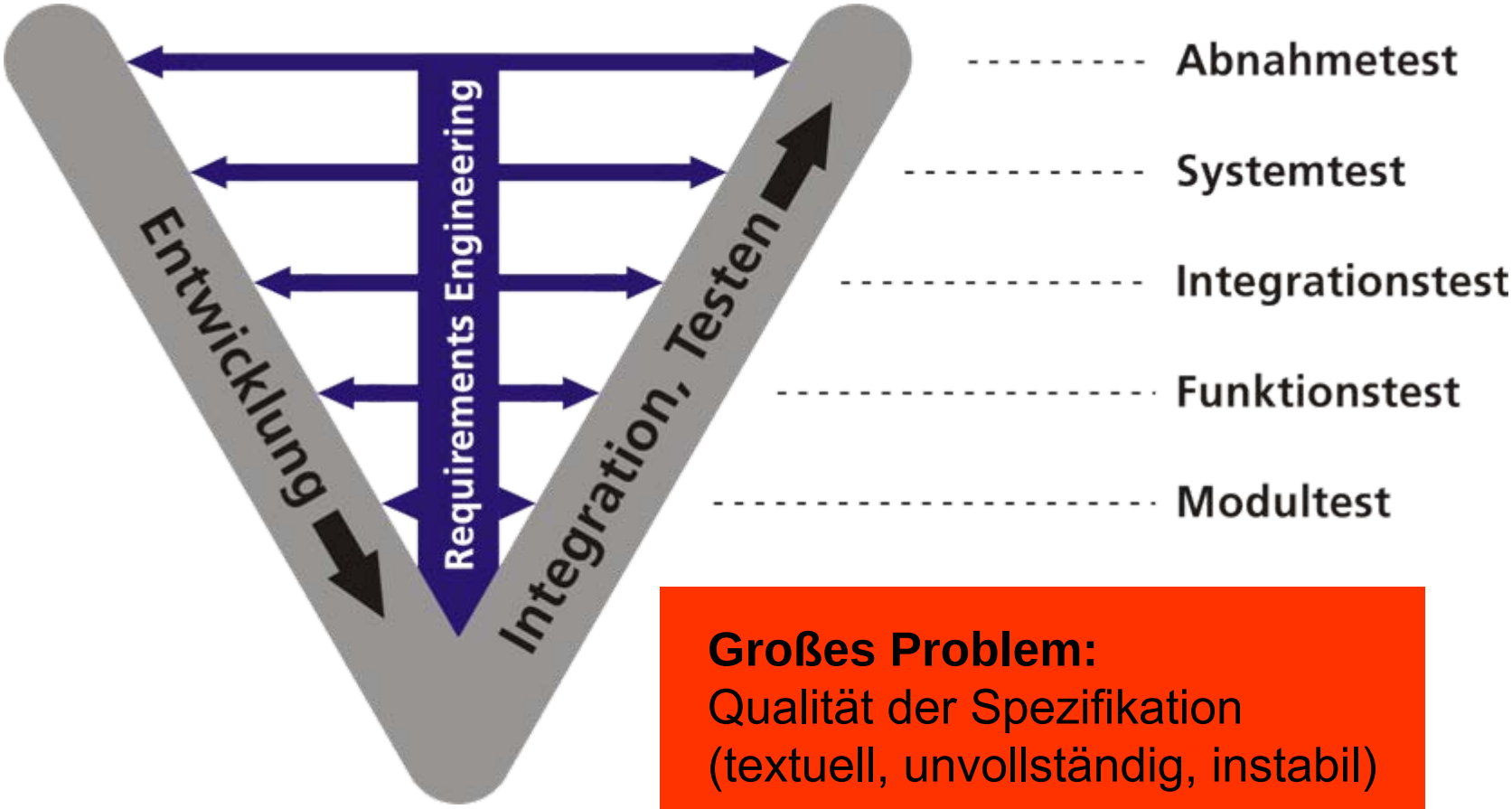
Der Prüfling
Einsatz der ETCS-Fahrzeuggeräte von Siemens

AVE S103 – Spanien; Madrid-Barcelona
Siemens Zug; Vmax 350 km/h

Taurus Rh 1116 – Österreich
Siemens Lok; Vmax 250km/h

AVE S102 – Spanien; Madrid-Barcelona
Bombardier Zug; Vmax 330 km/h

Der Prüfling
Derzeitiger Entwicklungsprozess

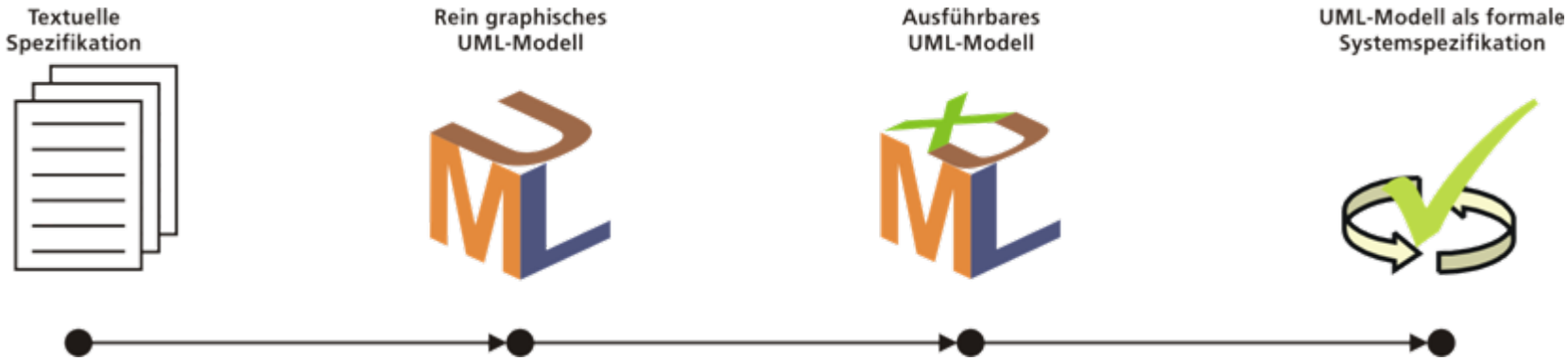


Großes Problem:
Qualität der Spezifikation
(textuell, unvollständig, instabil)

Der Lösungsansatz

Einführung formalen Notationen und Simulationen (1)

- Überführung der informalen Kundenanforderungen nach UML
- Erweiterung des Modells um „Action Semantics“ (-> Ausführbarkeit!)
- Validierung des Modells unter Verwendung der Simulation



UML: Unified Modelling Language
xUML: Executable UML

Der Lösungsansatz
Modellzweck und Modellkomponenten

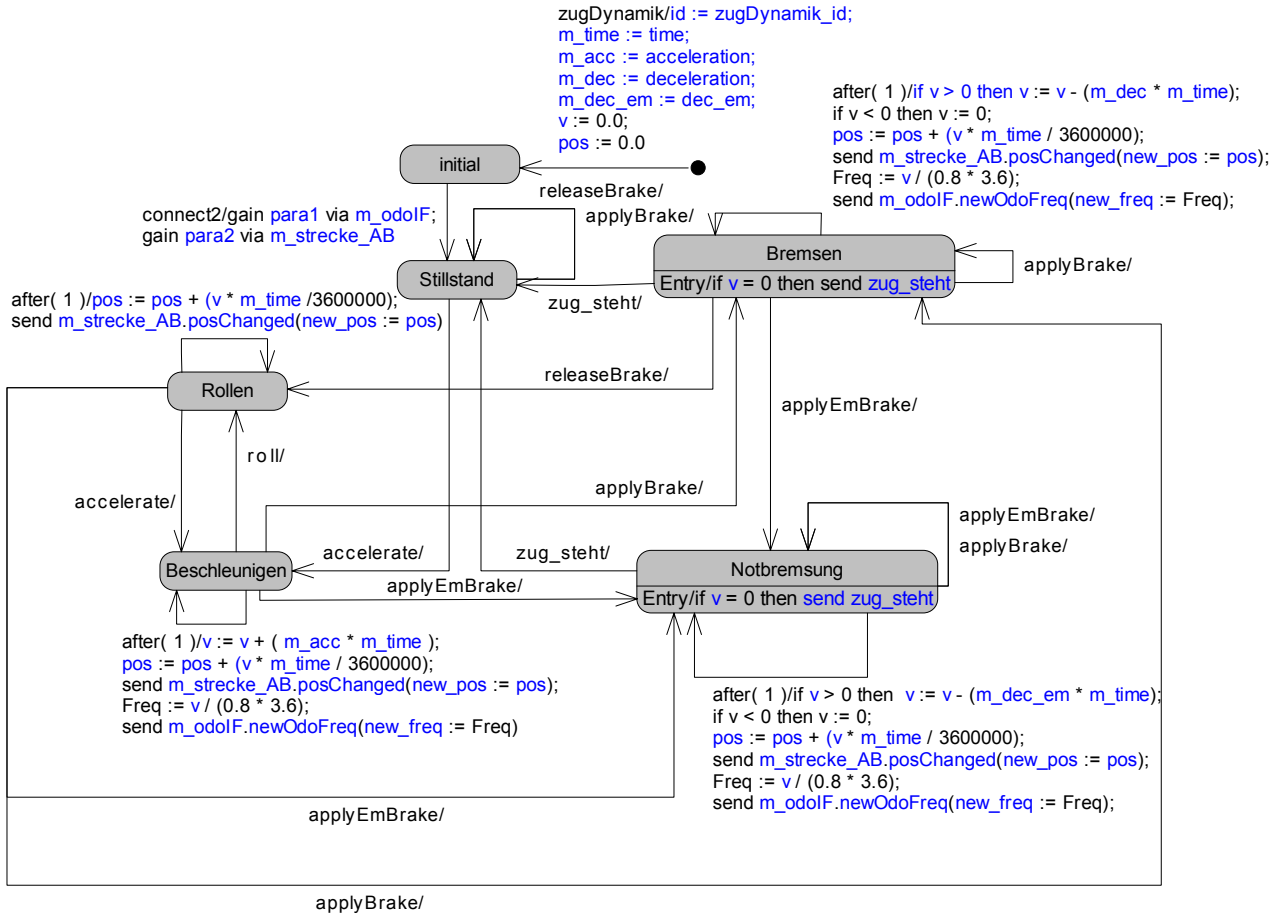
- Das Modell dient der Beschreibung funktionaler Eigenschaften
- Es handelt sich daher um ein Black- oder Gray-Box-Modell
- Komponenten: Struktur (statisch) und Verhalten (dynamisch)
- Erweitertes Verhaltensmodell durch „Action Semantics“

Struktur:
- Klassendiagramme
- Strukturdiagramme
- Objektdiagramme

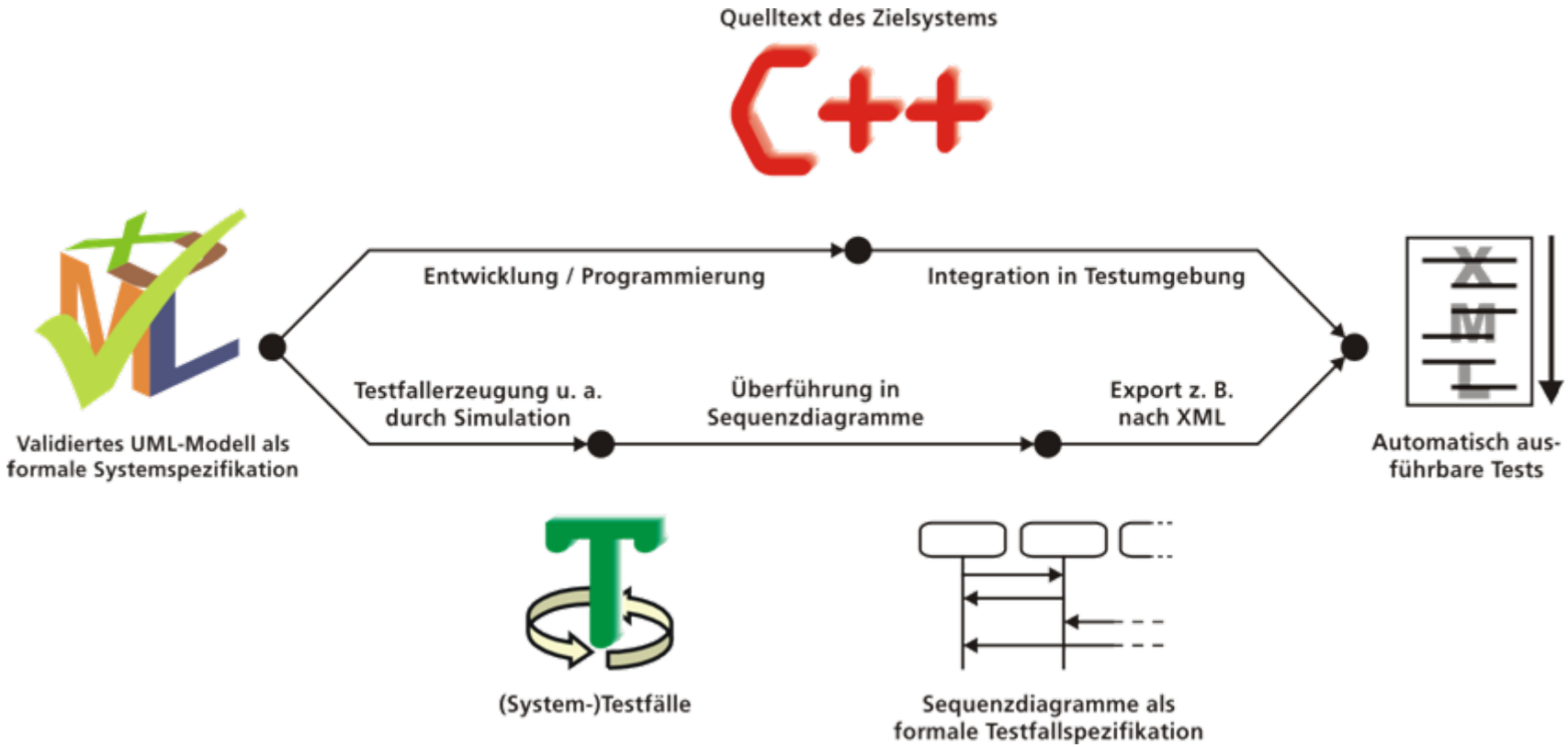
Verhalten:
- Zustandsdiagramme
- Sequenzdiagramme
- Kollaborationsdiagramme

Der Lösungsansatz

Beispiel eines Zustandsdiagramms mit Action Semantics

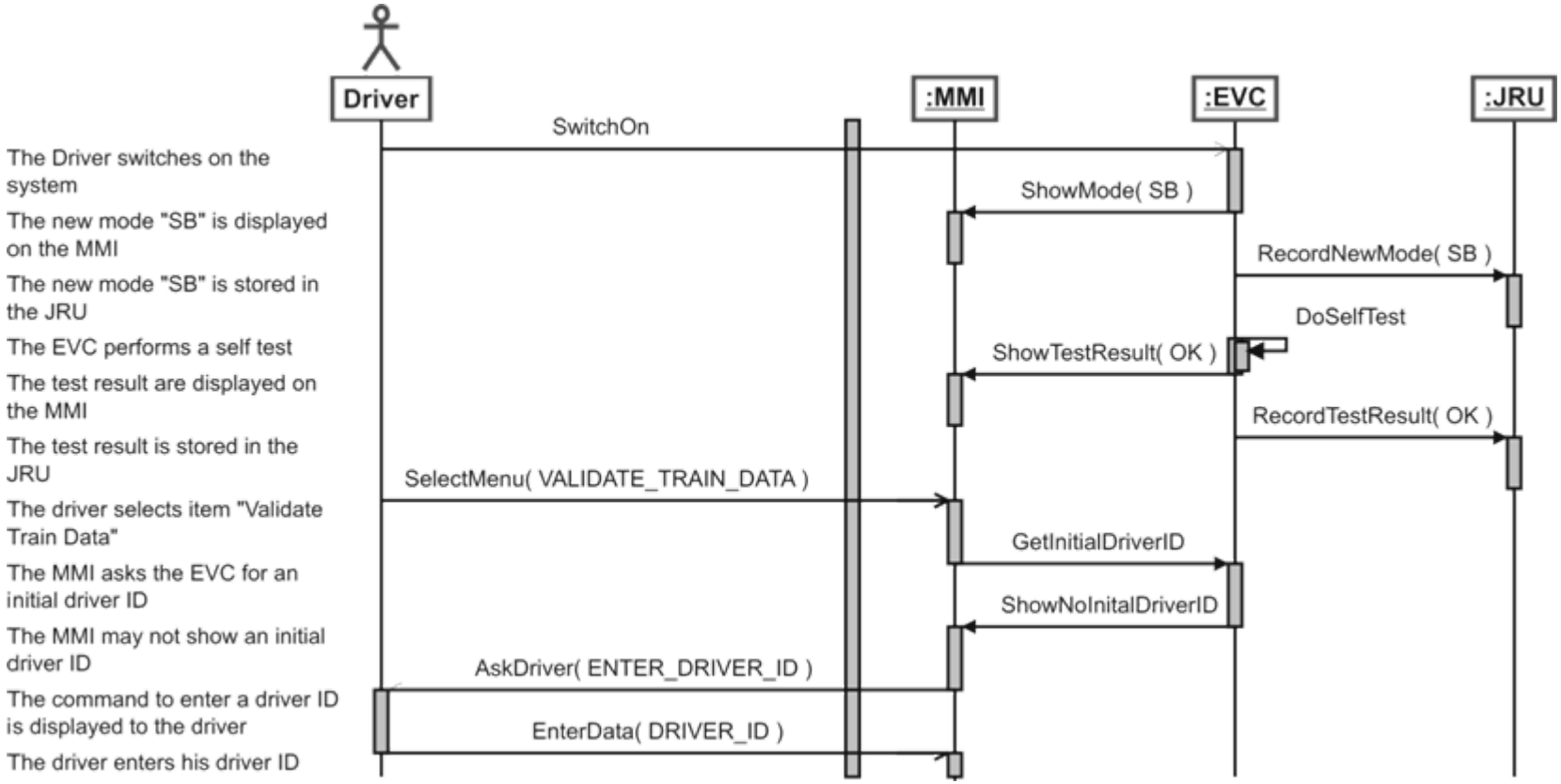


Der Lösungsansatz
Einführung formalen Notationen und Simulationen (2)



Der Lösungsansatz

Beispiel eines Sequenzdiagramms (UNISIG-Testseq.)

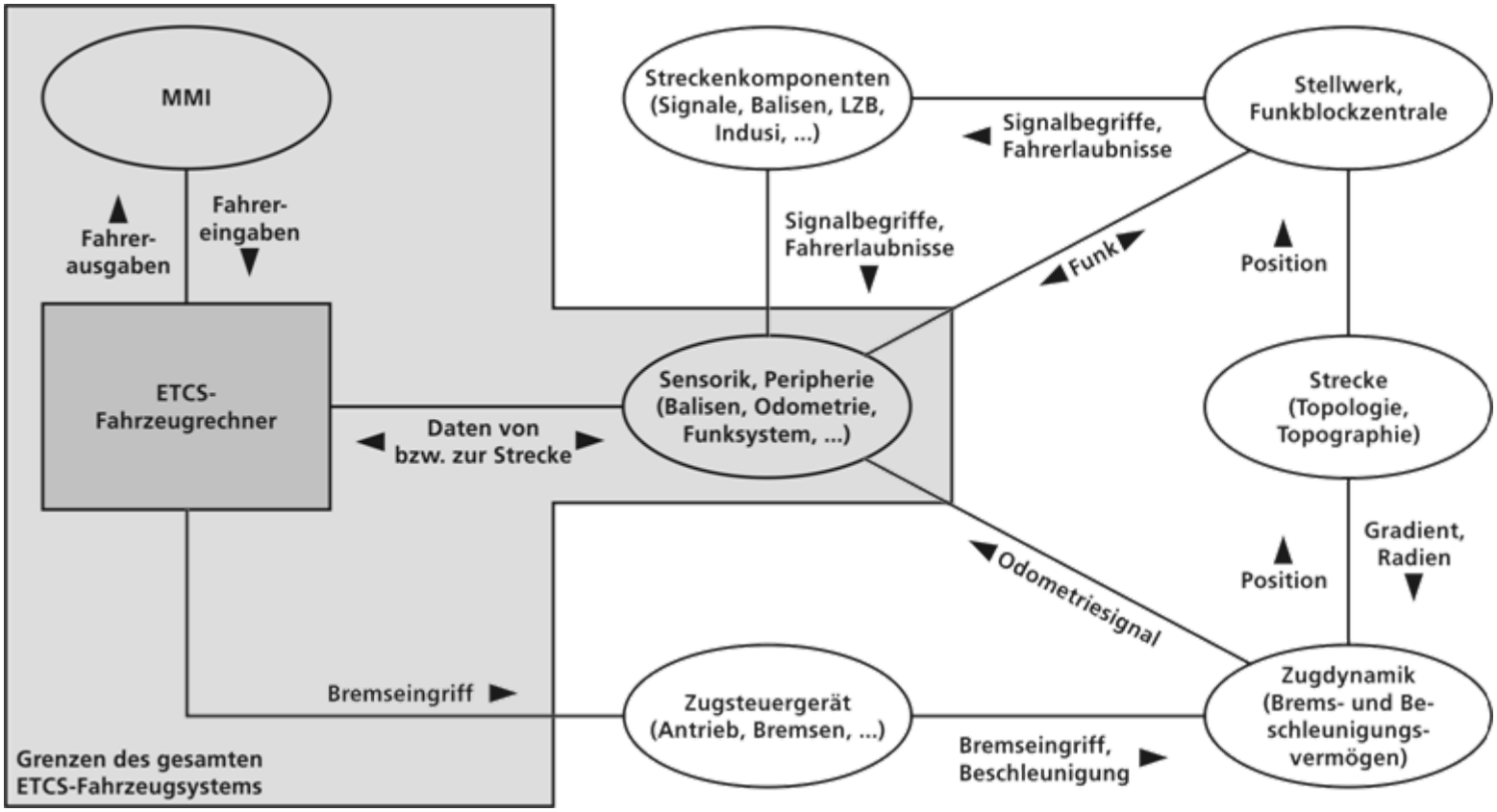


Die Simulations- und Testumgebung
Umgebungssimulation und Schnittstellen

- Ziel: Überprüfung der funktionalen Eigenschaften
- Schnittstellen von Modell und Prüfling müssen korrekt bedient werden
- Simulation:
  - Anreicherung mit einfachem Umgebungsmodell
  - Alternativ: Model-in-the-loop oder Software-in-the-Loop
- Tests:
  - Schnittstelle zwischen Ablaufsteuerung und Prüfling notwendig
  - daher: Testanlage mit Umgebungssimulation und Signalerzeugung
  - Hardware-in-the-loop

Die Simulations- und Testumgebung

ETCS-Fahrzeugsystem und umgebende Komponenten



Vom Konzept zur Umsetzung
Derzeitiger Arbeitsstand

- Erstellung eines einfachen UML-Fahrzeugmodells fast abgeschlossen
- Das Modell ist ausführbar (inkl. einfacher Umgebungssimulation)
- Bedienhandlungen können aufgezeichnet werden (=> Sequenzen)
- Parallel: Implementierung mit anderen Tools zwecks Vergleichbarkeit
  - LabView (Datenflussorientiert)
  - Rhapsody (Zustandsautomaten)
  - Petrinetze (ähnlich Zustandsautomat)
  - evtl. weitere Beschreibungsmittel / Methoden
- XML-Spezifikation für Testfälle wird gerade erarbeitet

Vom Konzept zur Umsetzung
Ausblick und geplante Aktivitäten

- Definition einer Reihe von Testfällen als Sequenzdiagramm
- Export von Sequenzdiagrammen nach XML
- Export von Bediensequenzen (Simulation) in Sequenzdiagramme
- Untersuchungen zur Skalierbarkeit
  - Einfluss der Modellgröße und –komplexität
  - Zeitverhalten
  - Aufwand
- Co-Simulation mit anderen Umgebungen (HiL-Labor / Simulationen)

HiL: Hardware-in-the-Loop

Zusammenfassung
xUML als Grundlage für methodisches Testen

- Komplexe, sicherheitsrelevante Software verlangt intensive Tests
- Die Qualität der Spezifikation ist entscheidend für effektive Tests
- UML kann die textuelle Systemspezifikation ersetzen
- UML-Modelle werden durch „Action Semantics“ ausführbar gemacht
- Testfälle werden aus der Simulation „aufgezeichnet“ (Bedienhandlungen)
- Testfälle werden als Sequenzdiagramme beschrieben
- Die Sequenzdiagramme dienen via Export zur autom. Testausführung
- Das Konzept wird gemeinsam vom DLR und Siemens TS RA erarbeitet und erprobt