



Antriebs- und Fahrwerktechnik



Strategien zur Testfallgenerierung aus UML-Zustandsautomaten

Dipl.-Ing. Carsten Paulus (FKFS), Dipl.-Ing. Michael Wolff (ZF Friedrichshafen AG),
Prof. Dr.-Ing. Hans-Christian Reuss (FKFS)

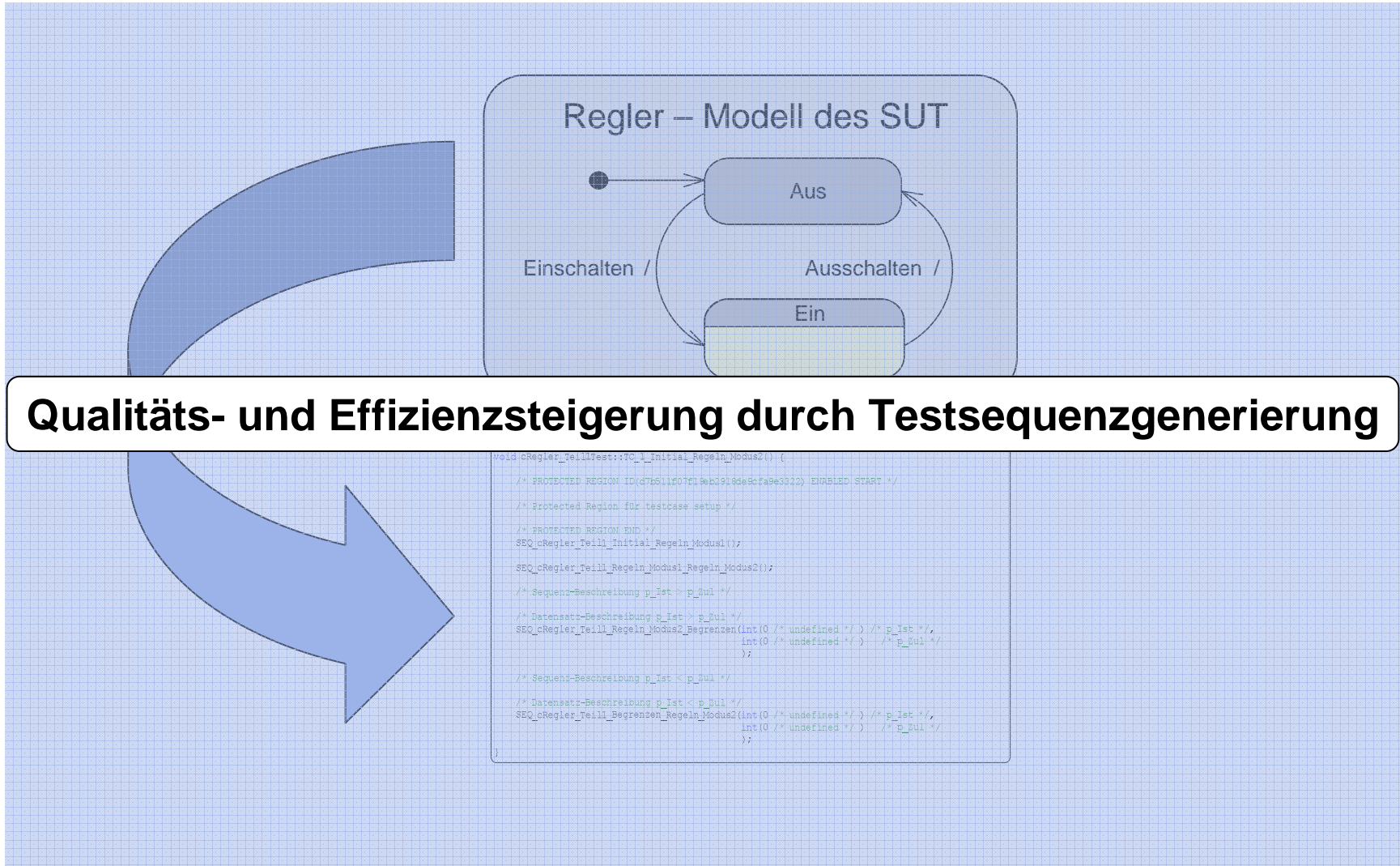




Gliederung



- **Motivation und Ziele**
- **Testprinzip**
- **Modellierung und Testsequenzerstellung**
- **Anwendung und Erfahrung**
- **Zusammenfassung und Ausblick**





Ziele



Automatische Testsequenzgenerierung aus Zustandsautomaten

- Qualitätssteigerung durch systematisches Vorgehen
- Effizienzsteigerung durch Automatisierung

Randbedingungen

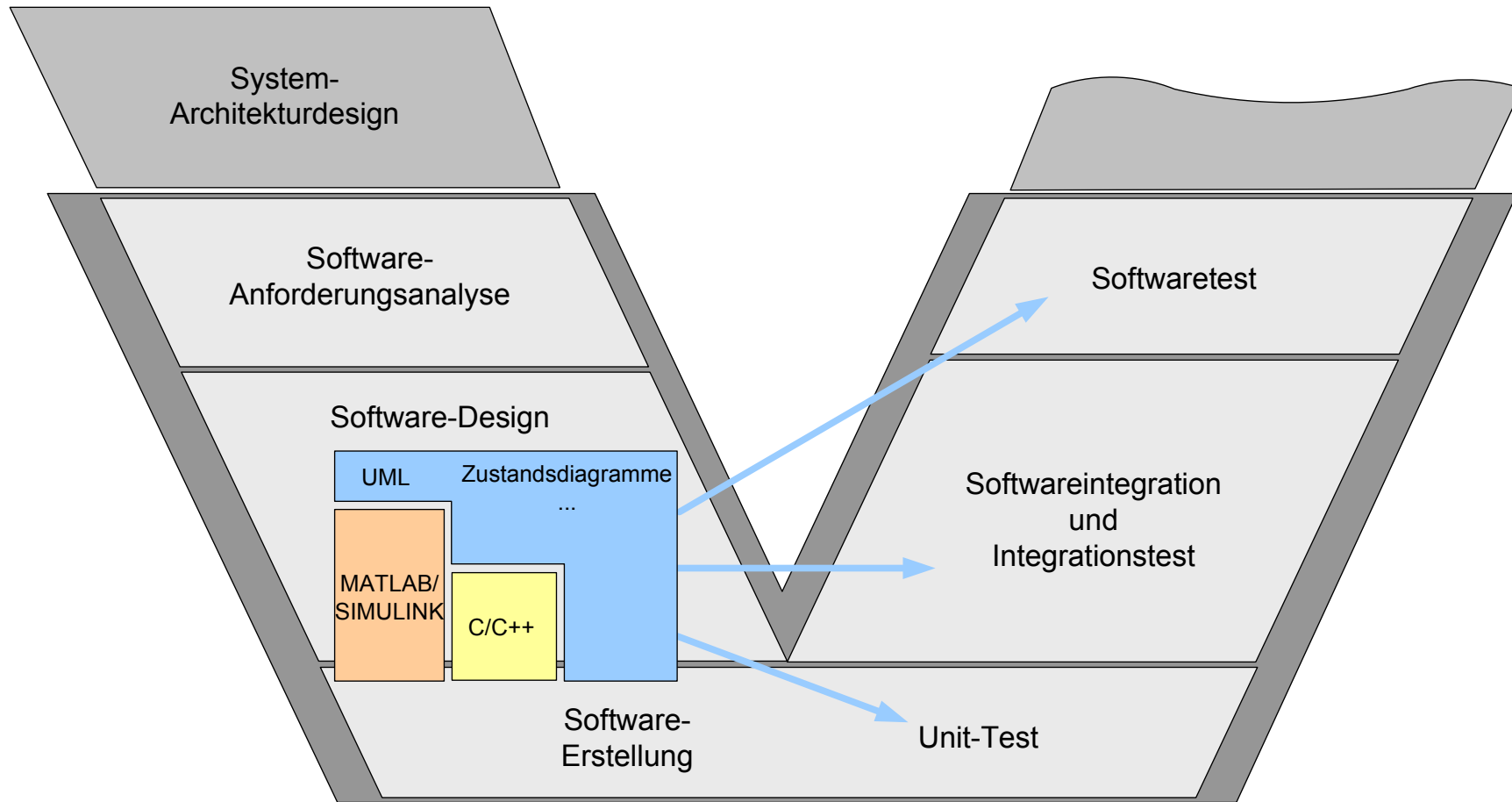
- Hohe Nutzerakzeptanz
- Wiederverwendung von Informationen aus dem Entwicklungsprozess

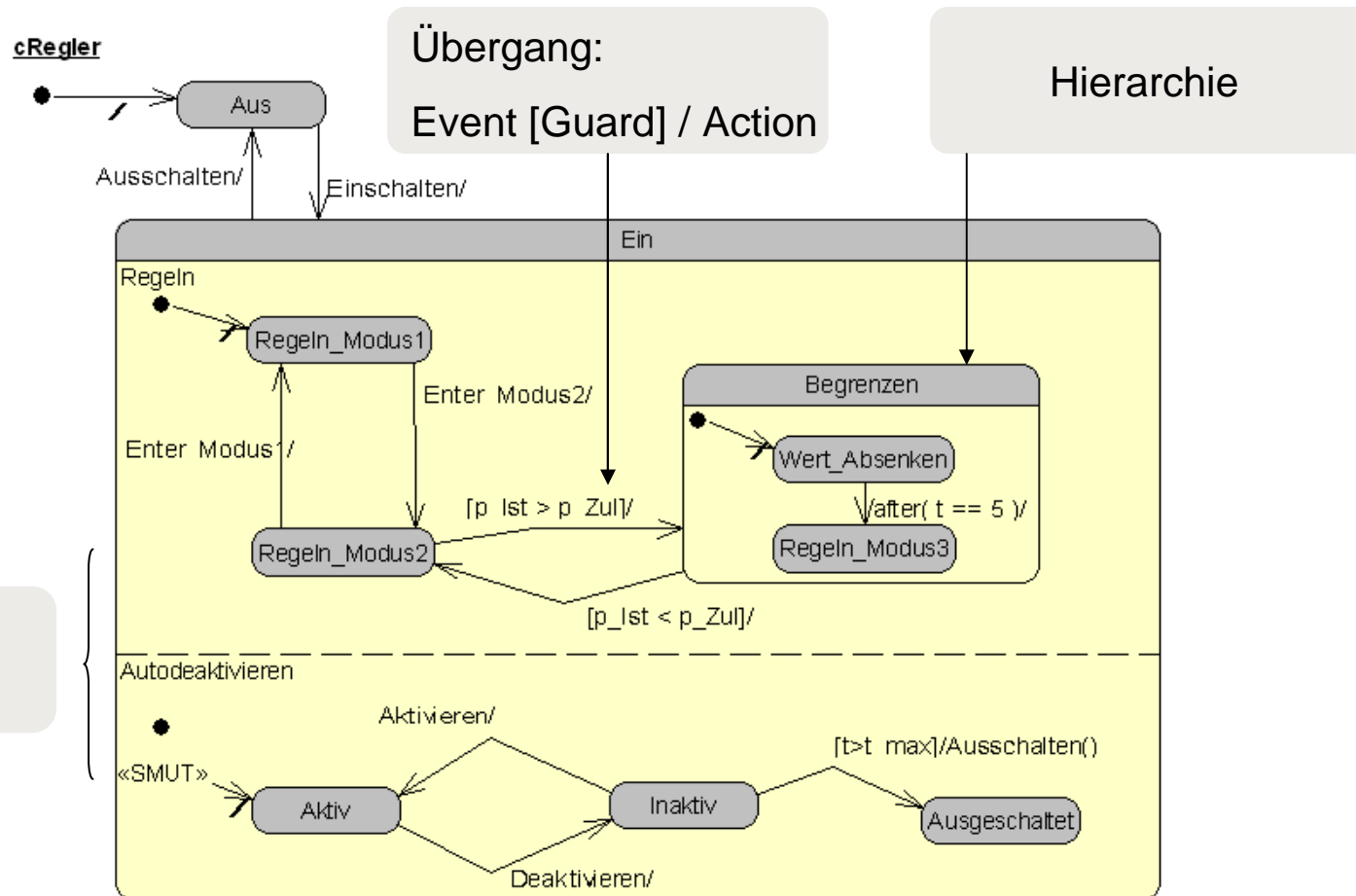
Fragestellungen

- Welche Fehlerarten können gefunden werden?
- Analyse des Testsequenzerstellungsvorgangs
 - ◆ Wie kann die Anzahl der Testsequenzen beeinflusst werden?
 - ◆ Wie können nachvollziehbare Testsequenzen erstellt werden?



Softwaredesign – Formale Basis für die Testspezifikation





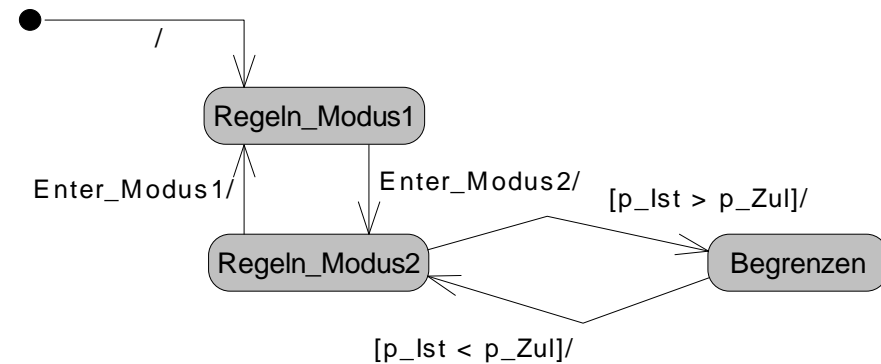


Testsequenzerstellung



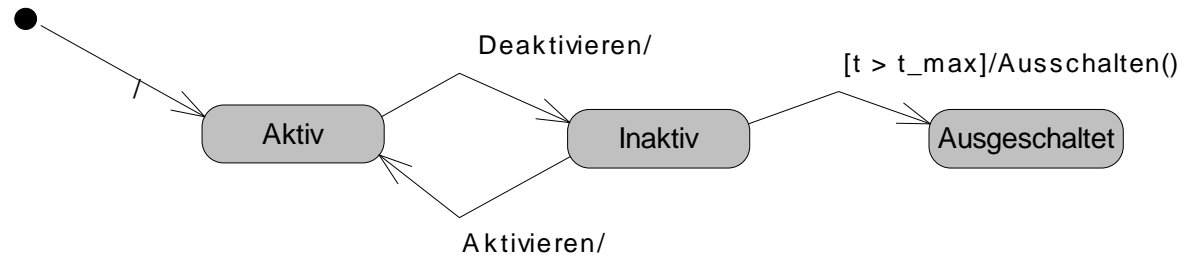
Strukturelle Überdeckung

- **All-States:** Jeder Zustand wird mindestens einmal besucht
 - ◆ Anzahl der Testschritte = $2 \cdot n$
- **All-Transitions:** Jeder Übergang wird mindestens einmal durchlaufen
 - ◆ Anzahl der Testschritte* = $k \cdot n$
- **N+:** Ausgehend von jedem Zustand werden alle Übergangs-Bedingungen aktiviert
 - ◆ Anzahl der Testschritte* = $k^2 \cdot n / 2$



k: Anzahl der Übergänge
n: Anzahl der Zustände

*: Faustformel für die Anzahl der Testschritte nach [Bin99]



Falscher Übergang: Die Implementierung durchläuft einen nicht aus der Spezifikation stimulierten Übergang

Falsche Aktion: Die an einen Übergang geknüpfte Aktion ist fehlerhaft

Nicht spezifizierter Übergang: Bei dem Aufruf eines Events wird in der Implementierung ein Übergang stimuliert, welcher in dem Modell nicht vorgesehen war

Nicht spezifizierter Zustand: Bei dem Aufruf eines Events wird in der Implementierung ein Zustand erreicht, welcher in dem Modell nicht vorgesehen war

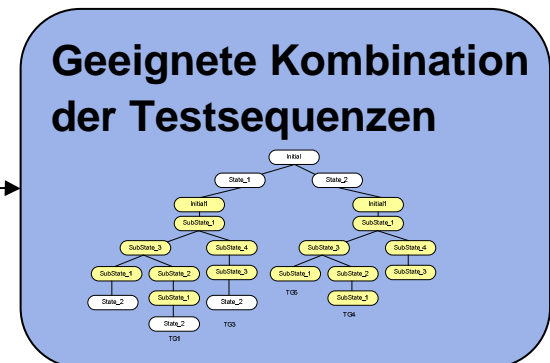
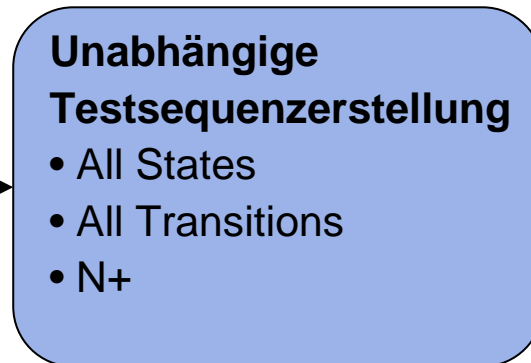
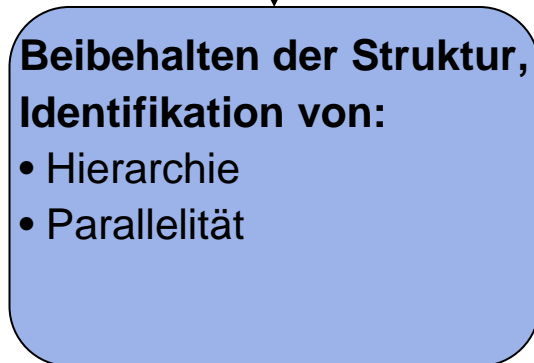
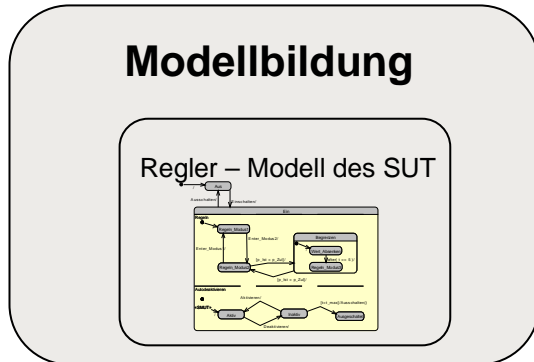


Auswahl einer Methode

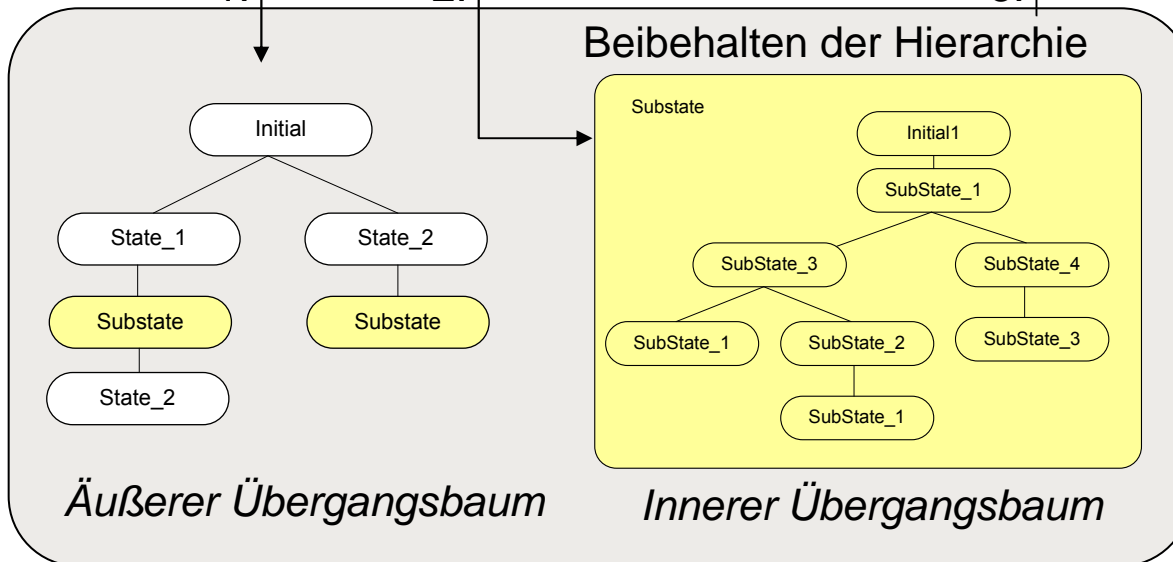
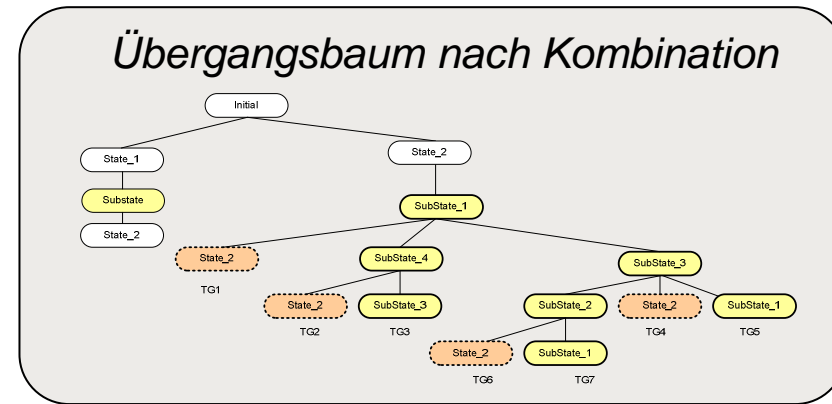
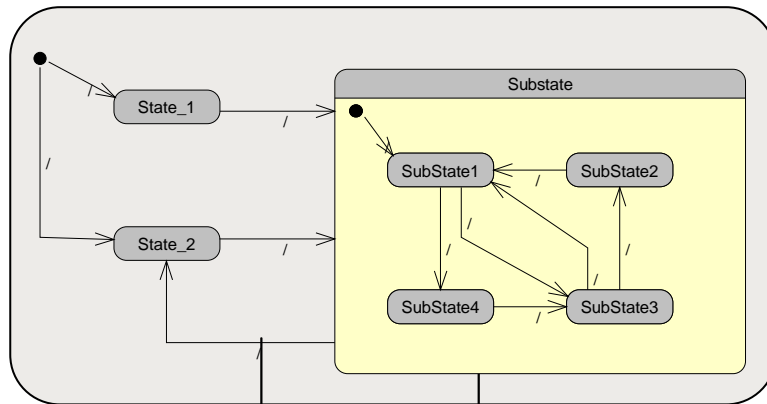


	All-States	All-Transitions	N+
Falsche Transition	Nein	Ja	Ja
Falsche Aktion	Nein	Ja	Ja
Nicht spezifizierter Übergang	Nein	Nein	Ja
Nicht spezifizierter Zustand	Nein	Nein	Ja
Faustformel Testschrittzahl	$2 \cdot n$	$k \cdot n$	$k^2 \cdot n/2$

- Strategien, welche eine Bedingungsüberdeckung des Modells berücksichtigen, erhöhen die Testschrittzahl weiter
- Die angegebenen Formeln gehen von einem flachen Zustandsautomaten aus. Hierarchie und Parallelität erhöhen die Testschrittzahl ebenfalls



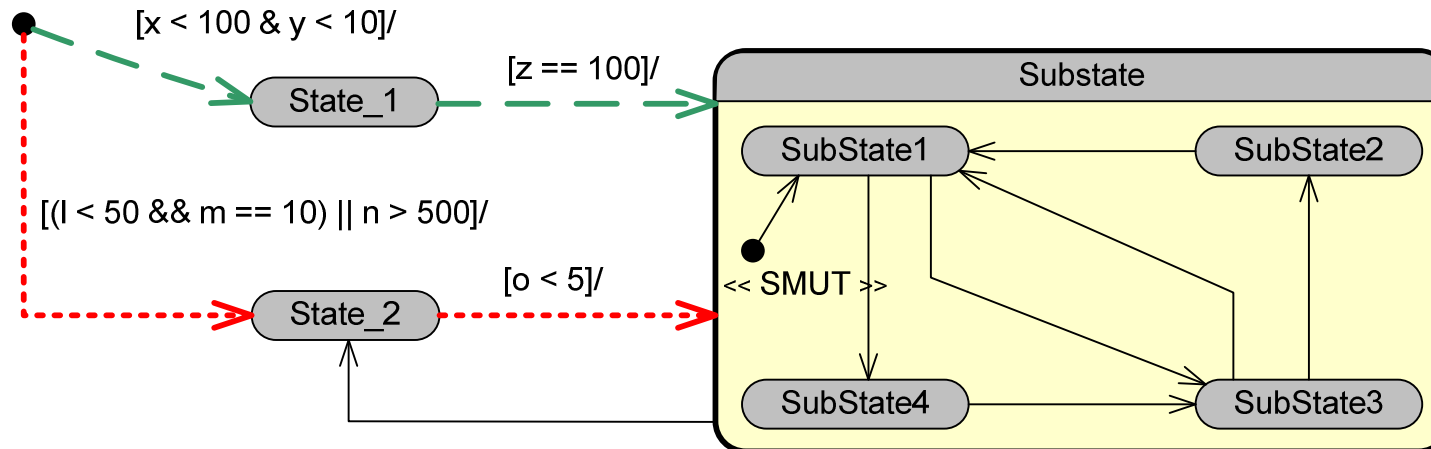
- ➔ Beibehalten der Struktur, um die Testsequenzen übersichtlich zu halten
- ➔ Beeinflussung der Testsequenzanzahl
- ➔ Gezieltes Testen von Modellteilen



- ➔ Die Hierarchie wird beibehalten
- ➔ Die vollständige Testsequenz wird inkrementell zusammengesetzt



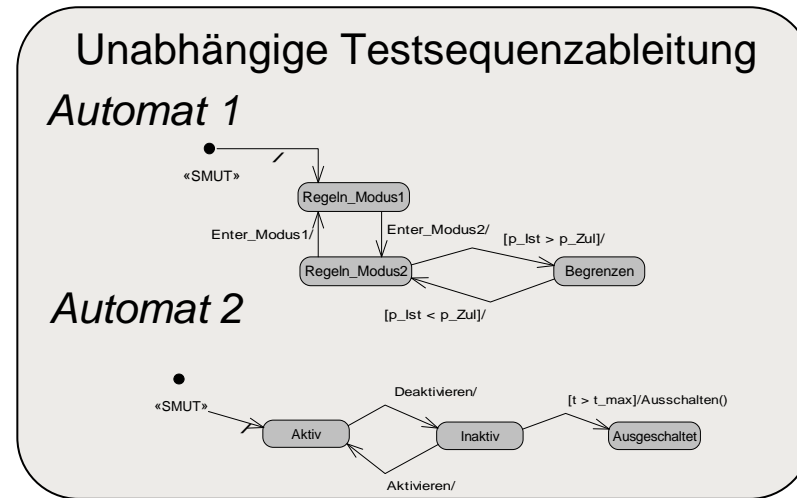
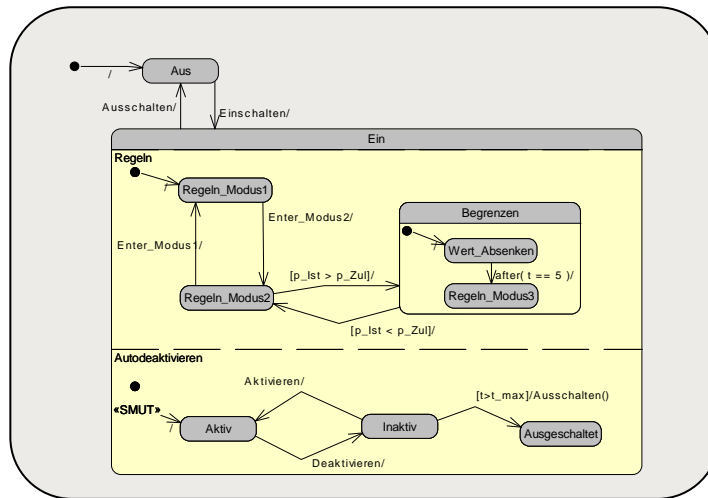
Testen des inneren Übergangsbaumes



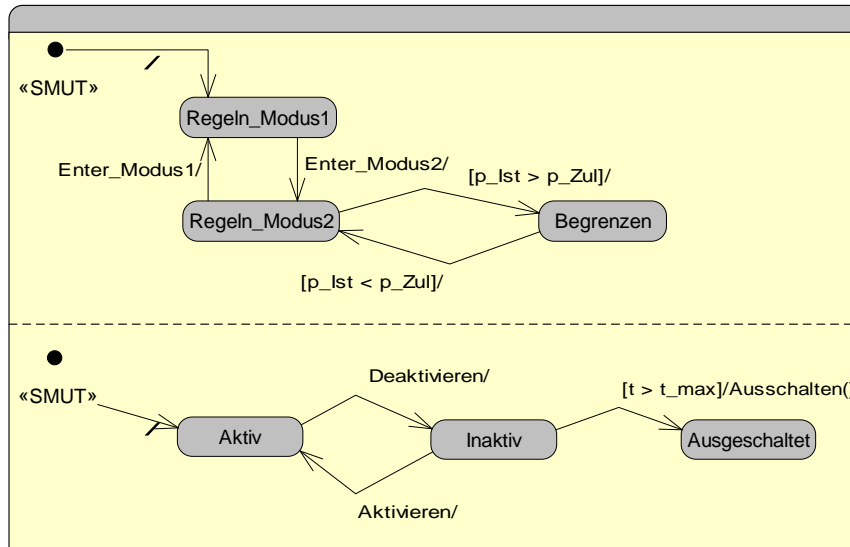
- Um die Testsequenzen für den inneren Übergangsbaum auszuführen, muss der Rest des Zustandsautomaten durchlaufen werden (State_1 oder State_2) – Initialisierung
- Mit dem Dijkstra Algorithmus wird der kürzeste Initialisierungspfad gewählt.
Parameter:
 - ◆ Anzahl der Zustände
 - ◆ Anzahl der Transitionen
 - ◆ Anzahl der zu bedatenden Argumente



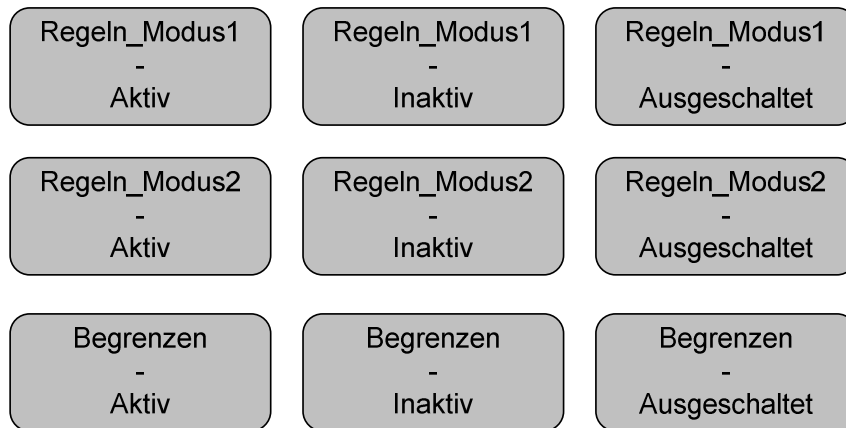
Metastrategie zum Umgang mit Parallelität



- ➔ Für alle Regionen werden unabhängig Testsequenzen erstellt
- ➔ Es kann anwendungsbasiert entschieden werden, wie die Testsequenzen miteinander kombiniert werden
- ➔ Die Art der Kombinatorik entscheidet über die Anzahl der Testfälle



↓ Konfigurationen des Produktautomaten



Getrennte Betrachtung

Zustände: $n_1 = 3$
 Transitionen: $k_1 = 5$

Zustände: $n_2 = 3$
 Transitionen: $k_3 = 4$

Summe Testschritte

All Transition Methode
 Anzahl Testschritte ($k \cdot n$)

15

12

27

Produktautomat

Zustände: n_p
 Transitionen: k_p

Summe Testschritte

All Transition Methode
 Anzahl Testschritte ($k \cdot n$)

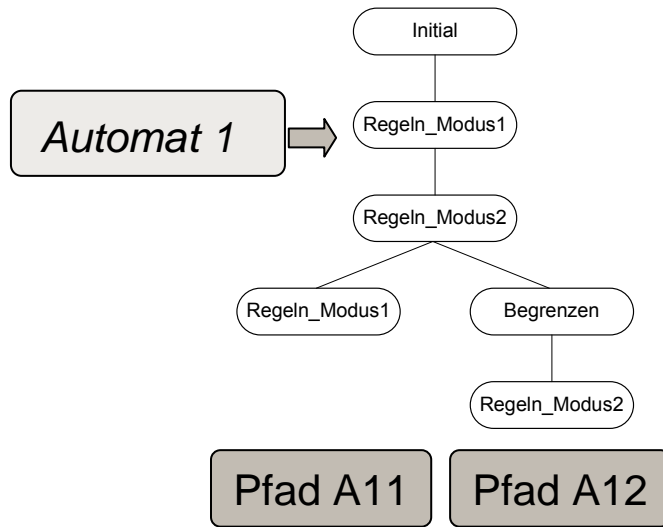
$n_1 \cdot n_2$
 $k_1 \cdot n_2 + k_2 \cdot n_1 + k_1 \cdot k_2$

$n_1 \cdot n_2 \cdot (k_1 \cdot n_2 + k_2 \cdot n_1 + k_1 \cdot k_2)$
 $9 \cdot (15 + 12 + 20) = \mathbf{513}$

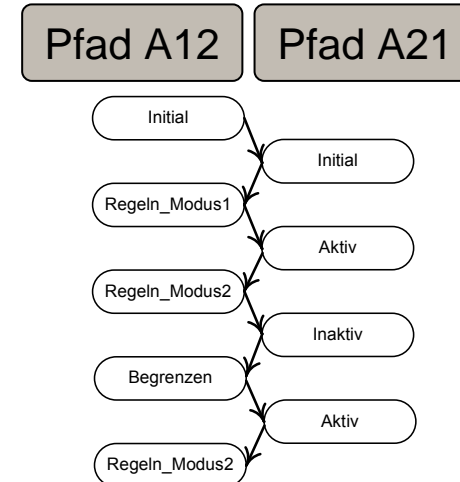
→ 19 Mal so viele Testschritte!



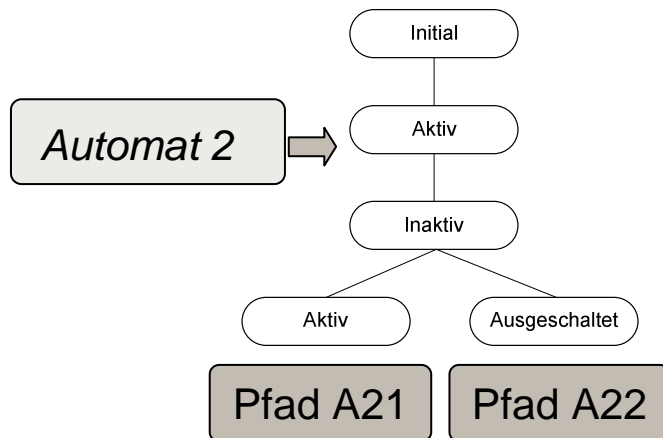
Ausführungsmöglichkeiten der unabhängig erstellten Pfade



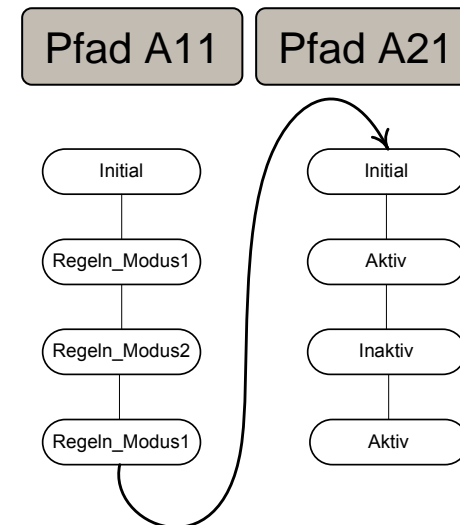
Implementierungsabhängige Entscheidung:



Verzahnte Ausführung

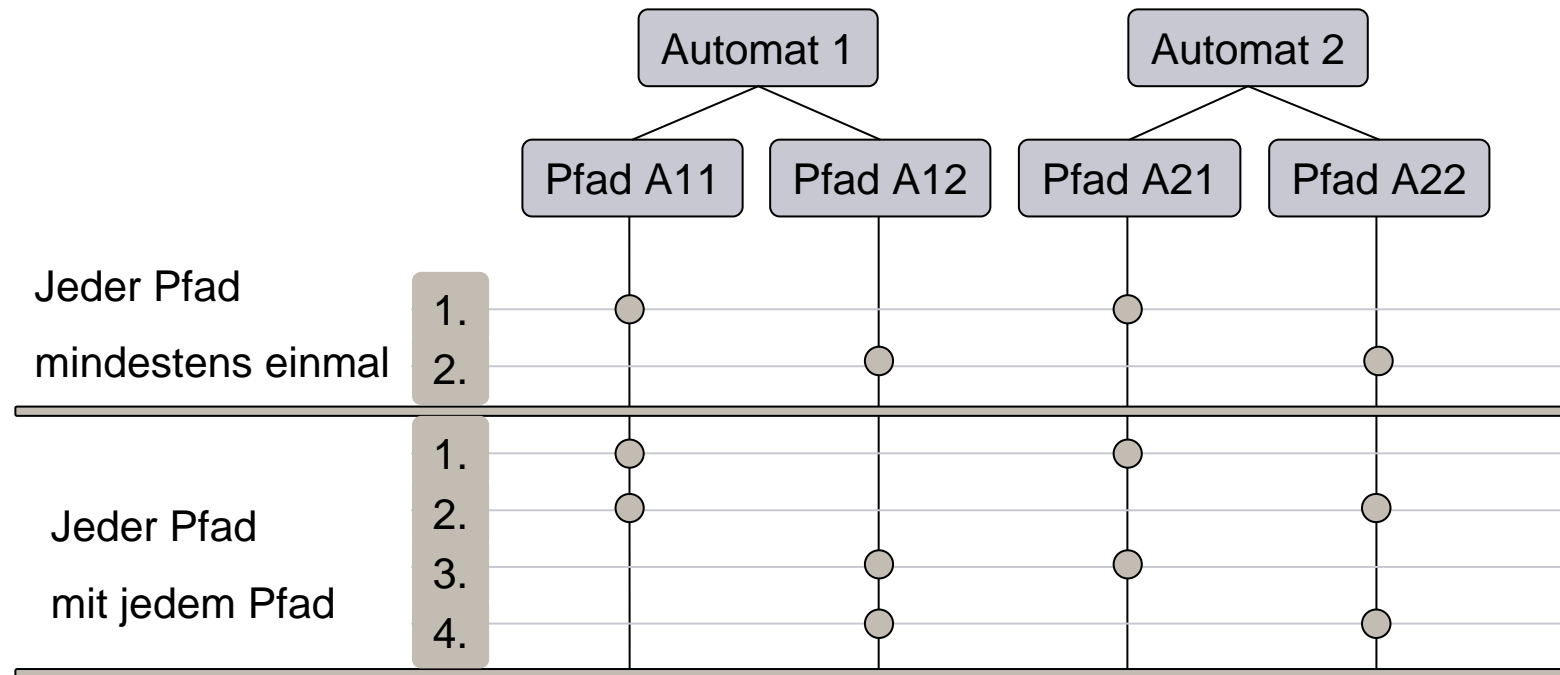


Sequenzielle Ausführung



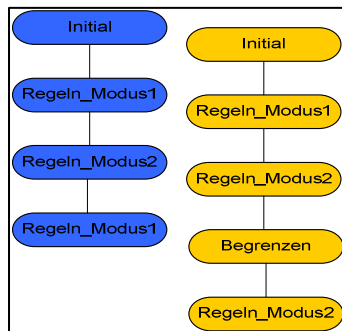


Kombinationsmöglichkeiten der Tetsequenzen

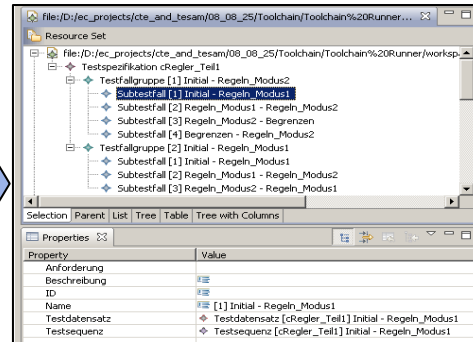




Testsequenzen



Testfall- beschreibungssprache



Ausführbarer Testcode Unit-Test (C++)

```
void cDruckReglerTest::TC_1_ZUSTAND_STARTUP_ZUSTAND_AUS() {
    /* PROTECTED REGION ID(cd8cefbacda2bcb4a4357a53215b4a) ENABLED START */
    /* Protected Region für testcase setup */

    /* PROTECTED REGION END */
    SEQ_cDruckRegler_ZUSTAND_STARTUP_ZUSTAND_AUS();

    /* Sequenz-Beschreibung Datenfeld.t_Einsprung(HoleTemperatur()) <= 0 */

    /* Datensatz-Beschreibung Datenfeld.t_Einsprung(HoleTemperatur()) <= 0 */
    SEQ_cDruckRegler_ZUSTAND_AUS_ZUSTAND_EIN_REGELND(si16(20000) /* p_Steuer_Soll */,
                                                    si16(0) /* p_Regel_Soll */
                                                    );
    SEQ_cDruckRegler_ZUSTAND_EIN_REGELND_ZUSTAND_AUS();
}
```

- ➔ Die Kontrollflussüberdeckung des zu testenden Codes kann gemessen werden und stellt ein Maß für die Güte der Testsequenzen dar
- ➔ Auswahl der All-Transitions Methode
 - ◆ Nicht spezifizierte Zustände und Übergänge werden durch die Kontrollflussmessung identifiziert
 - ◆ Die Anzahl der Testfälle bleibt überschaubar
- ➔ Abhängigkeit zwischen Modell und Testsequenzen ist transparent



Erfahrung



- Die Automatisierung wurde gegen eine manuelle Referenzimplementierung getestet. Der manuell erstellte Testcode besteht aus ca. 4000 Zeilen. Die Werkzeugkette generiert ca. 2000 Zeilen Testcode
- In dem Projekt konnte mit einem Modell bereits eine Bedingungsüberdeckung des zu testenden Codes von 50 % nachgewiesen werden
- Fehlende Überdeckung geht nicht zu Lasten der Modellierung oder der Methode. Das zustandsbasierte Verhalten des zu testenden Codes macht in dieser Anwendung nur ca. 70 % aus
- Die transparenten strukturellen Testsequenzen können manuell ergänzt werden
 - ◆ Funktioniert ein Pfad bei mehrmaligem Durchlauf?
 - ◆ Überprüfen, einer nicht im Modell spezifizierten Bedingung
 - ◆ Unterschiedliches Zeitverhalten: Ereignisse mit verschiedenen Zeitabständen



Zusammenfassung

- Qualitätssteigerung durch transparent nachvollziehbare strukturelle Testsequenzen
- Effizienzsteigerung durch automatische Testcodegenerierung
- Analyse des Testsequenzerstellungsprozesses
- Identifikation von Faktoren zur Steuerung der Testsequenzanzahl und ihrer Transparenz
 - ◆ Auswahlkriterien für eine Testsequenzerstellungsmethode
 - ◆ Metastrategien zum Umgang mit Parallelität und Hierarchie

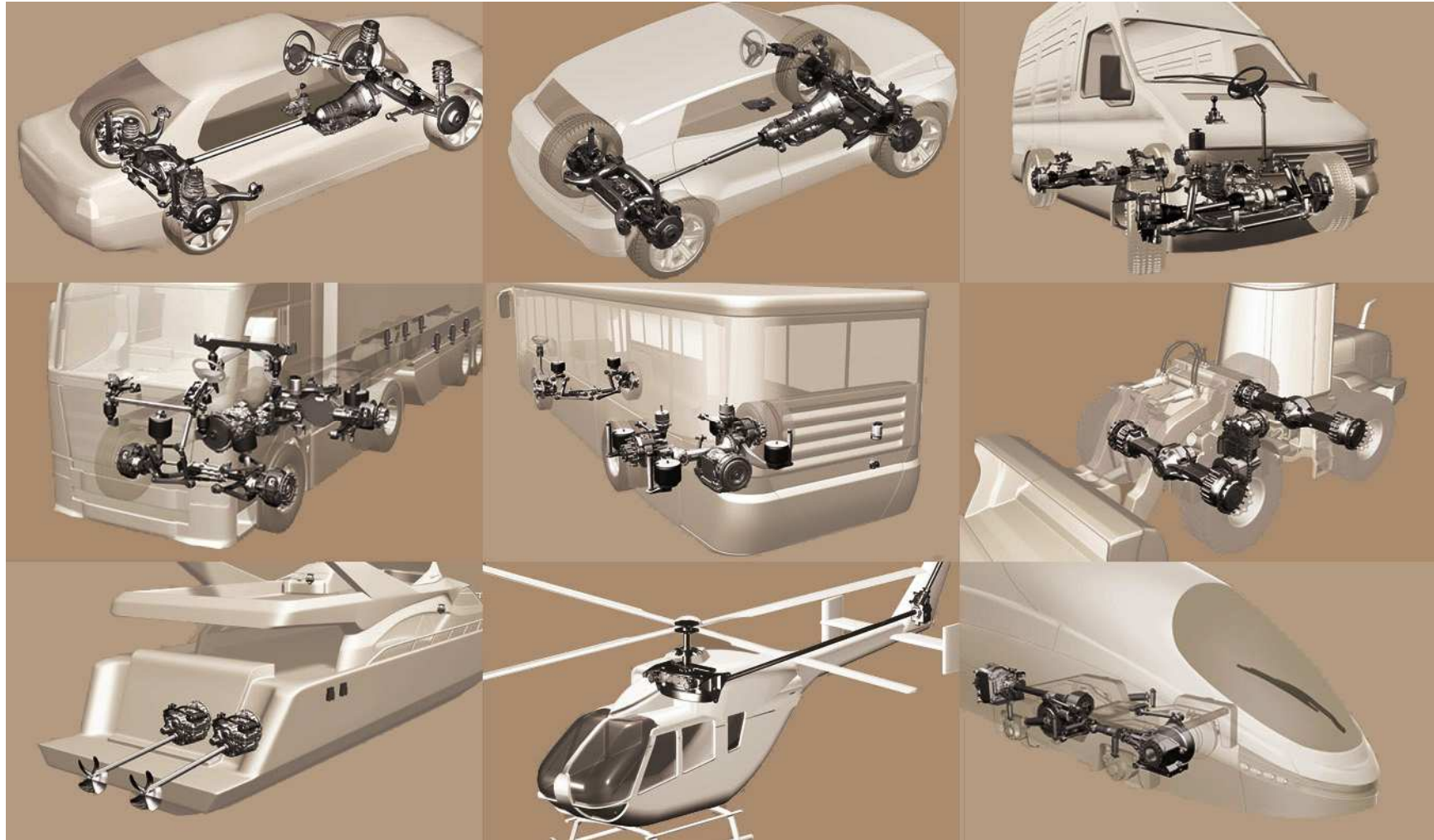
Ausblick

- Einsatz der Methoden in anderen Testphasen
- Verbreitung der Methoden in den Projekten



Vielen Dank für Ihre Aufmerksamkeit!

Gerne stehe ich für Fragen zur Verfügung





Literatur



- [Bin99] Robert V. Binder: Testing Object-Oriented Systems, Models, Patterns and Tools. Addison Wesley, 1999.