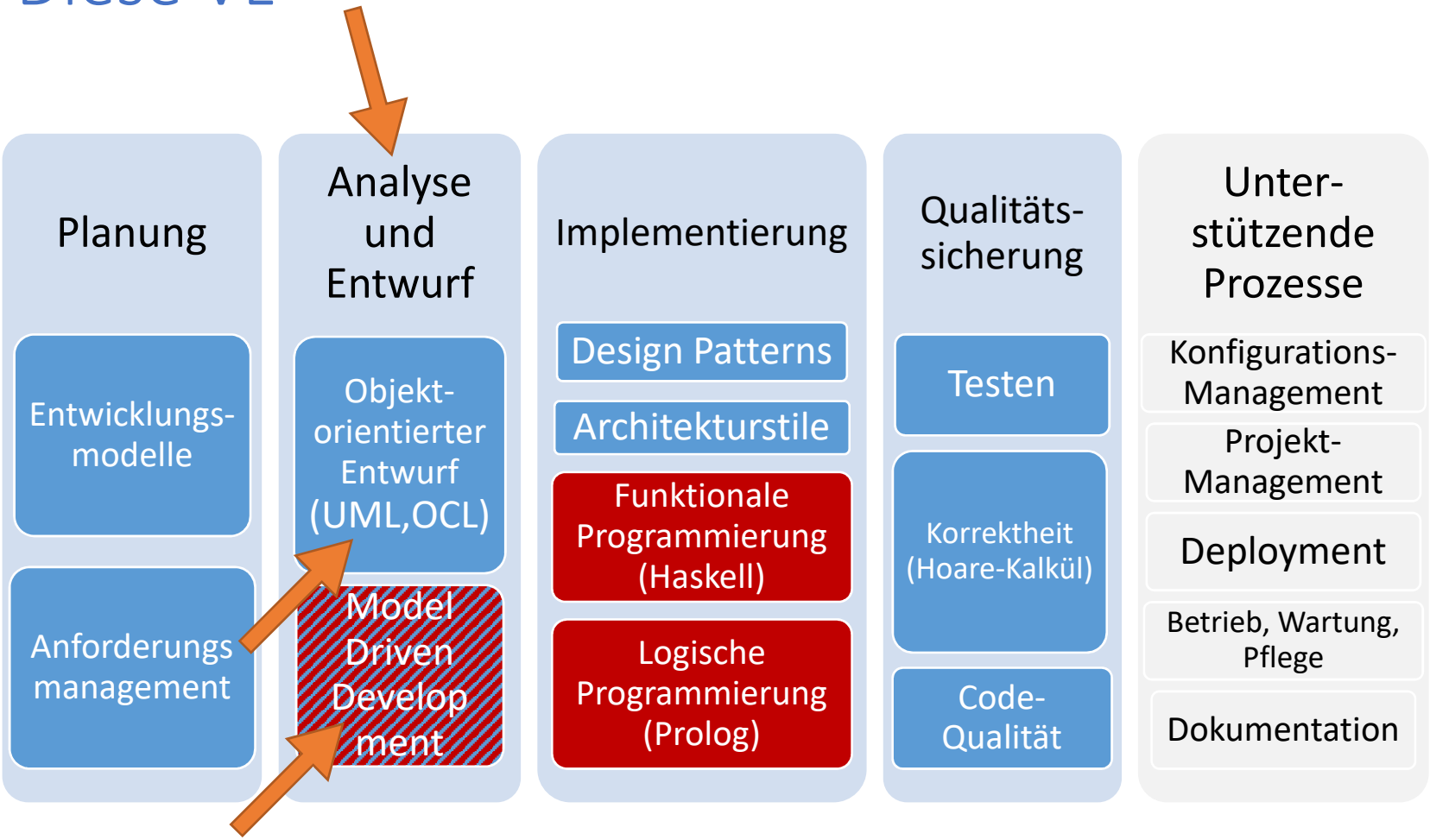
Softwaretechnik und Programmierparadigmen

08 Analyse und Entwurf - Verhalten

Prof. Dr. Sabine Glesner
Software and Embedded Systems Engineering
Technische Universität Berlin

Diese VL


Planung
Entwicklungs-modelle
Anforderungs management
Analyse und Entwurf
Objekt-orientierter Entwurf (UML,OCL)
Model Driven Develop ment
Implementierung
Design Patterns
Architekturstile
Funktionale Programmierung (Haskell)
Logische Programmierung (Prolog)
Qualitäts-sicherung
Testen
Korrektheit (Hoare-Kalkül)
Code-Qualität
Unter-stützende Prozesse
Konfigurations-Management
Projekt-Management
Deployment
Betrieb, Wartung, Pflege
Dokumentation
Softwaretechnik-Anteil
Programmierparadigmen-Anteil

Inhalt

Analyse und Entwurf - Verhalten

- UML Verhaltensdiagramme
- Aktivitätsdiagramm

Model Driven Development (Ausblick)

Inhalt

Analyse und Entwurf - Verhalten

- UML Verhaltensdiagramme
- Aktivitätsdiagramm

Model Driven Development (Ausblick)

Analyse und Entwurf

UML definiert sieben Verhaltensmodelle

- zur Spezifikation des dynamischen Verhaltens des Systems

➢ Was wann und in welcher Reihenfolge mit welchen beteiligten Komponenten (oder Akteuren) passieren soll

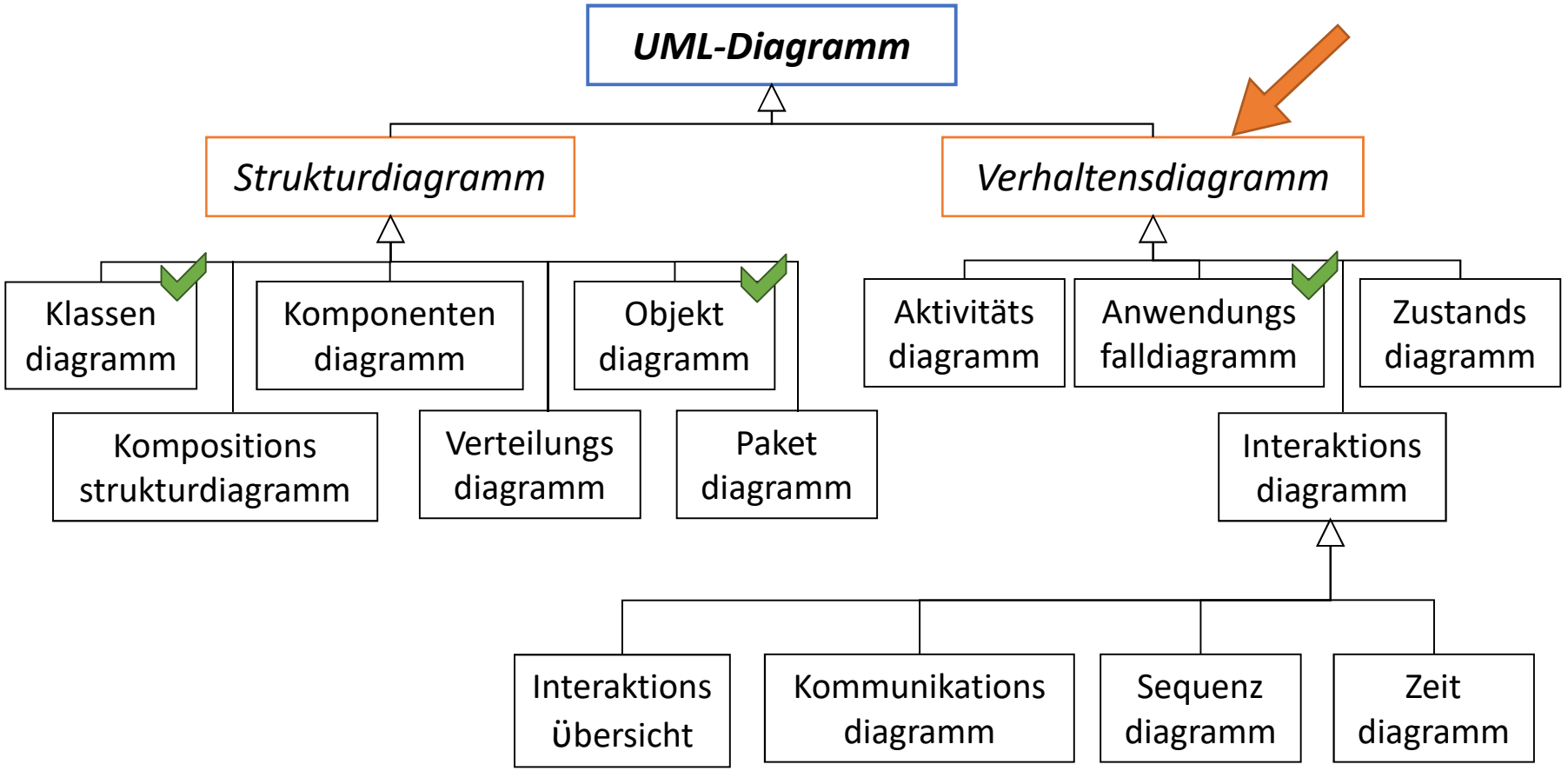
Interaktionsdiagramme visualisieren die Interaktion von Komponenten

- Verständliche Visualisierung von dynamischem Verhalten ist schwierig
- Idee: Vier verschiedene Modelle zur Hervorhebung verschiedener Aspekte

➢ Anwendungsbereich, Ziele und Gestaltungsmittel der Diagramme überlappen

➢ Auswahl hängt stark von der Anwendung ab

UML Diagrammübersicht



Übersicht Verhaltensdiagramme

Anwendungsfalldiagramm (use case diagram)

- Überblick von Anwendungsfällen und ihren beteiligten Akteuren
- Haben wir im Requirements Engineering kennengelernt

Aktivitätsdiagramm (activity diagram)

- Modelliert Ablauf einer Aktivität im System, z.B. Anwendungsfall, oder organisatorische Prozesse (Workflow)
- Beinhaltet Kontroll- und Datenfluss
- Semantik ähnlich der von Petri-Netzen

Zustandsautomat (state diagram)

- Dynamisches Verhalten als endlicher Automat
- Eignet sich besonders für nebenläufige Aktivitäten und nicht-terminierende Systeme

Übersicht Interaktionsdiagramme

Sequenzdiagramm (sequence diagram)

- Beschreibt Interaktionen im System als Austausch von Nachrichten
- Spezifiziert explizit die zeitliche Ordnung von Ereignissen und die beteiligten Objekte und Klassen

Kommunikationsdiagramm (communication diagram)

- Dient zur Übersicht der Kommunikationspartner und ihrer Verbindungen
- Ähnlich dem Sequenzdiagramm, aber Fokus stärker auf den Verbindungen als der Reihenfolge der Interaktionen

Zeitverlaufsdiagramm (timing diagram)

- Präzise Spezifikation von zeitlichen Abläufen
- Darstellung der Interaktionen in zweidimensionalem Diagramm (x-Achse: Zeit)

Interaktionsübersichtsdiagramm (interaction overview diagram)

- Dient zur Modularisierung von komplexen Interaktionsdiagrammen
- Ähnelt dem Aktivitätsdiagramm, wobei die Knoten ihrerseits andere Interaktionsdiagramme enthalten

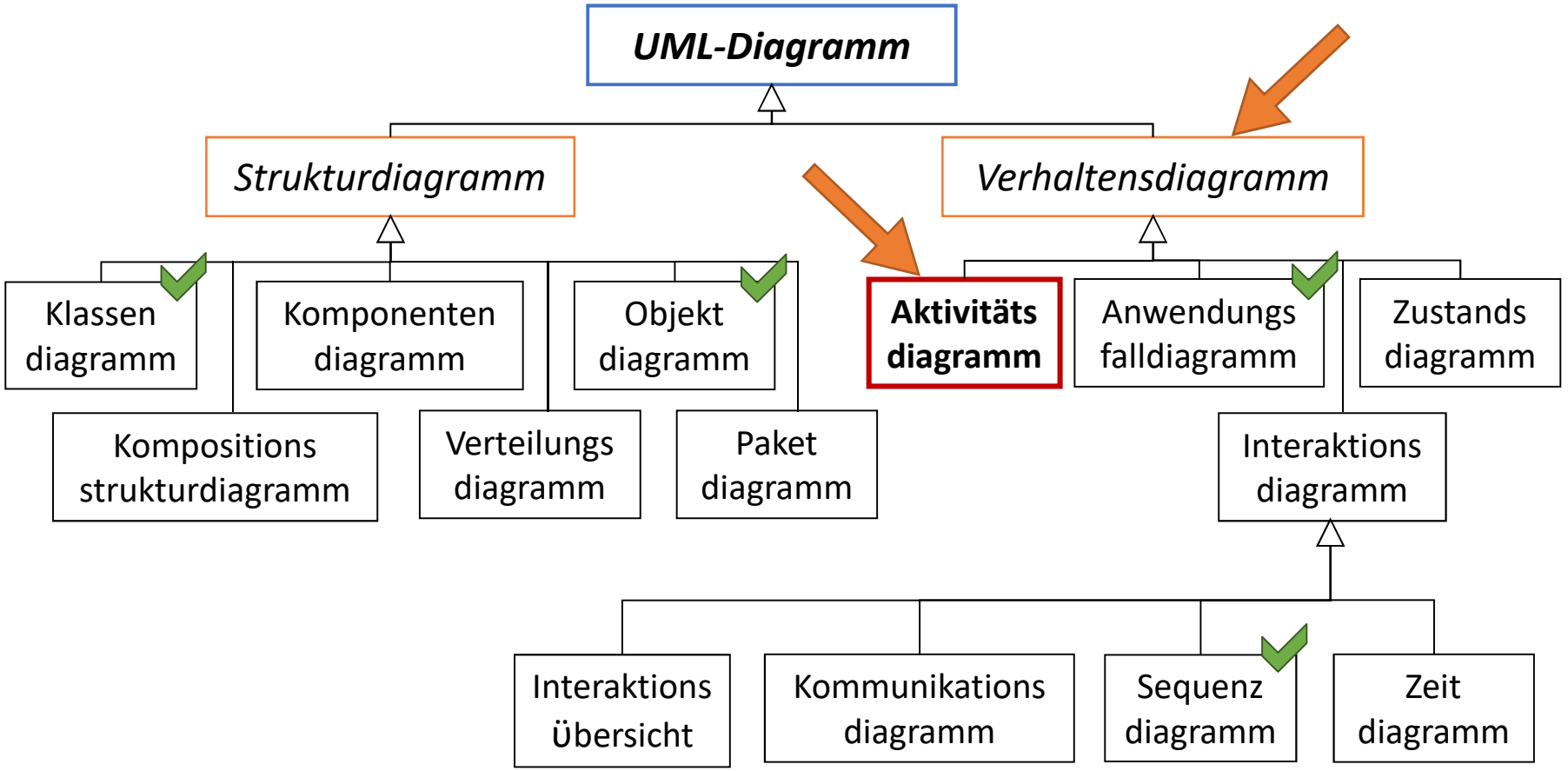
Inhalt

Analyse und Entwurf - Verhalten

- UML Verhaltensdiagramme
- Aktivitätsdiagramm

Model Driven Development (Ausblick)

UML Diagrammübersicht



Aktivitätsdiagramme

- Beschreiben Aktivitäten innerhalb des Systems
  - Dienen der Beschreibung von dynamischem Verhalten
  - Reihenfolge und Ablauf von Aktionen ergibt Aktivität

- Breites Anwendungsspektrum:
  - Darstellung von Kontroll- und Objektfluss
  - Detaillierte Spezifikation von Anweisungen und Operationen (ähnlich einer Programmiersprache)
  - Organisatorische Prozesse (Workflows)
  - Zur Modellierung verschiedener Abstraktionsebenen

- Sinnvoll: Zusammenfassung aller Szenarien eines Anwendungsfalls mit einem Aktivitätsdiagramm

Aktivitätsdiagramme

Aktivität wird dargestellt als **gerichteter Graph**

- Knoten: Aktionen, Objekte, Kontrollknoten
- Kanten: Kontroll-, Datenfluss

Aktionen sind kleinste Ausführungseinheiten, z.B.:

- Aufrufe von Operationen
- Senden und Empfangen von Nachrichten
- Zugriff auf Daten und Objekte

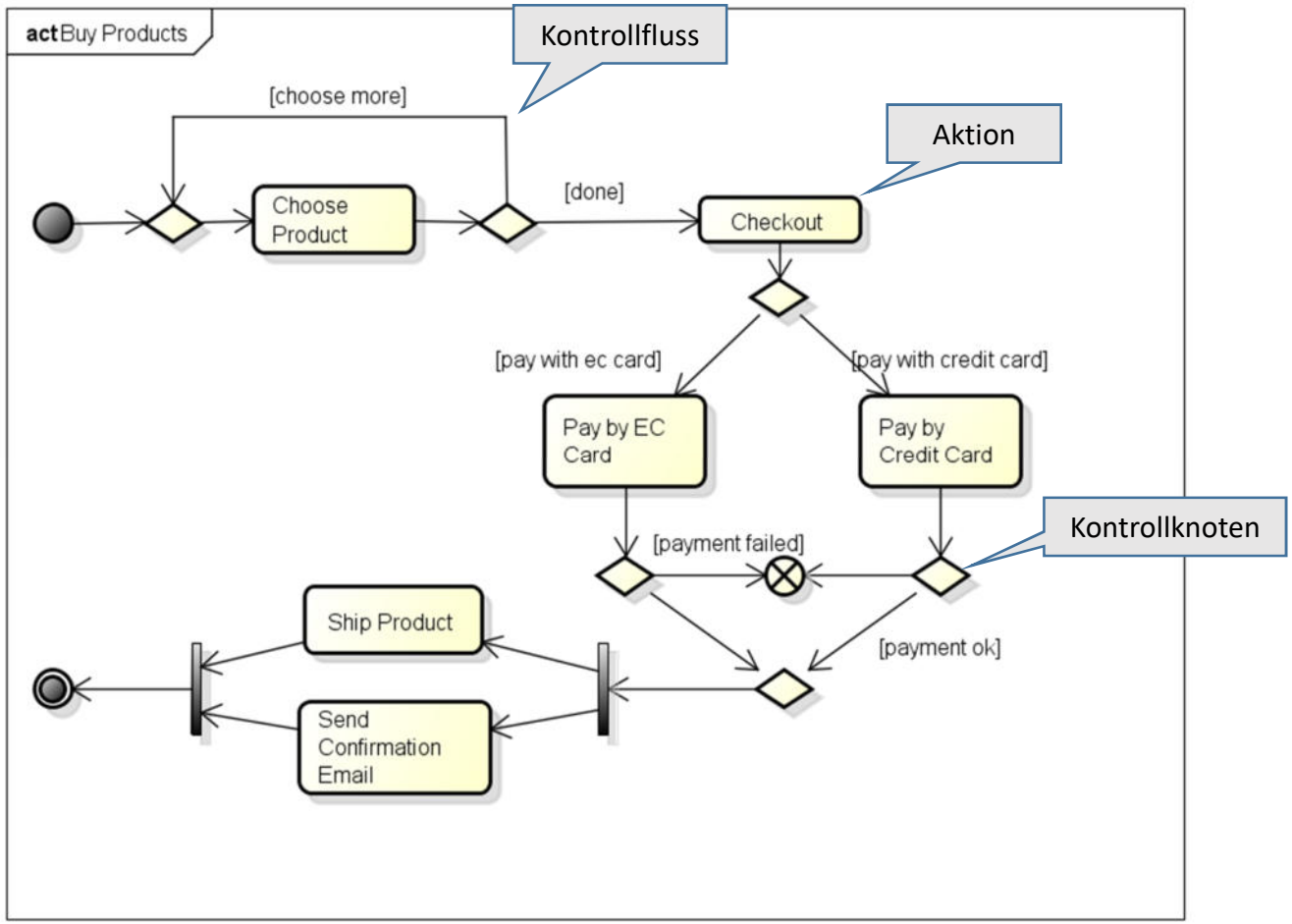
Spezielle **Kontrollknoten** zur Verzweigung des Kontrollflusses:

- Start-, Endknoten
- Entscheidung, Zusammenführung
- Splitting, Synchronisation

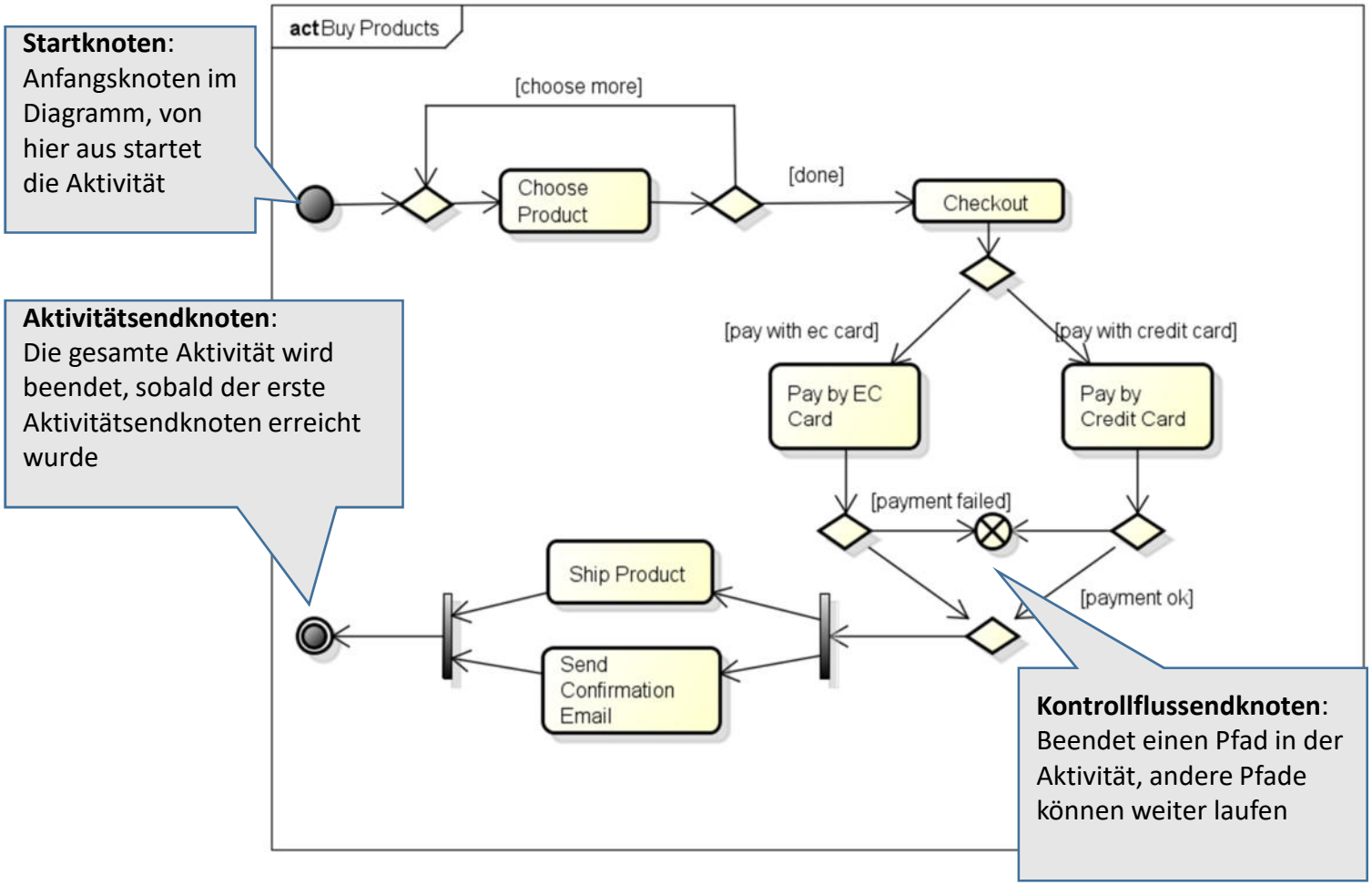
Token-basierte Semantik

➢Knoten wird erreicht (ausgeführt), sobald Vorgänger-Aktion beendet ist

Aktivitätsdiagramme - Beispiel



Start-/Endknoten



Startknoten: Anfangsknoten im Diagramm, von hier aus startet die Aktivität

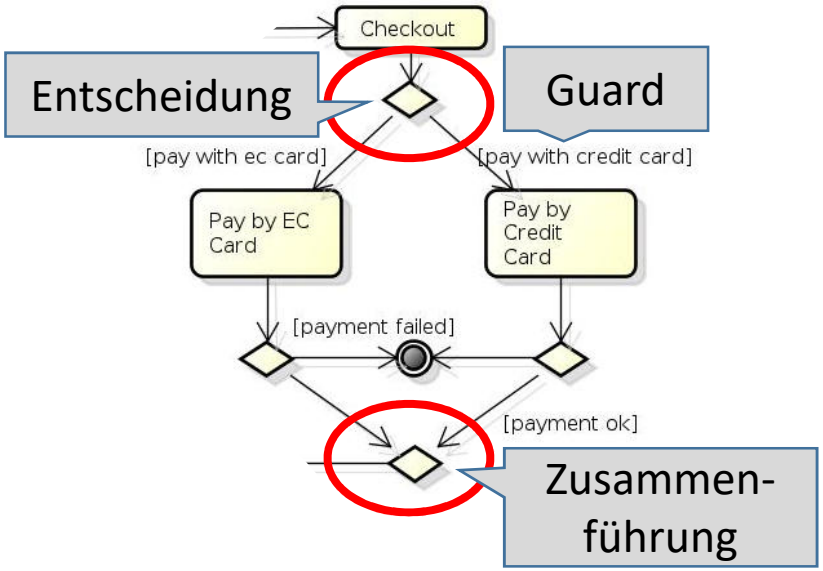
Aktivitätsendknoten: Die gesamte Aktivität wird beendet, sobald der erste Aktivitätsendknoten erreicht wurde

Kontrollflussendknoten: Beendet einen Pfad in der Aktivität, andere Pfade können weiter laufen

Entscheidung/Zusammenführung

Entscheidung

- Nur einer der folgenden Kontrollflüsse wird ausgeführt
- **Disjunkte Guards** notwendig
- Weglassen eines Guards an einem Ausgang gleichbedeutend mit „Else“



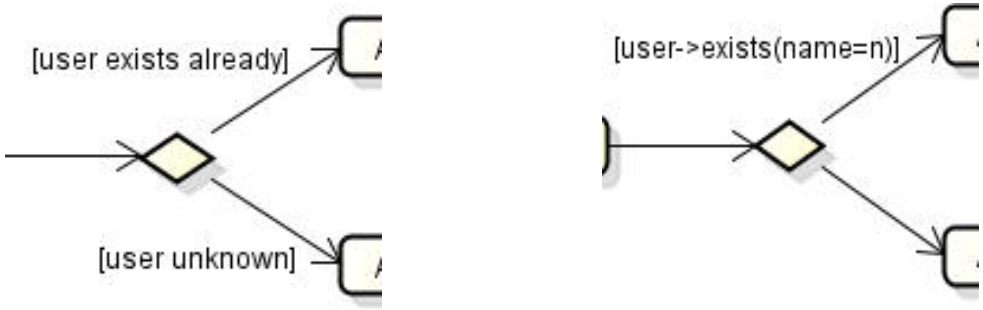
Zusammenführung

- Führt alternative Zweige wieder zusammen

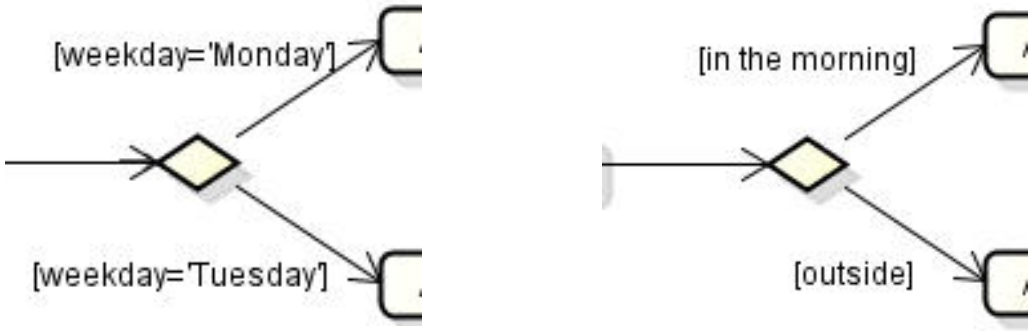
Beliebig viele Ein-/Ausgänge der Verzweigungsknoten möglich

Guards

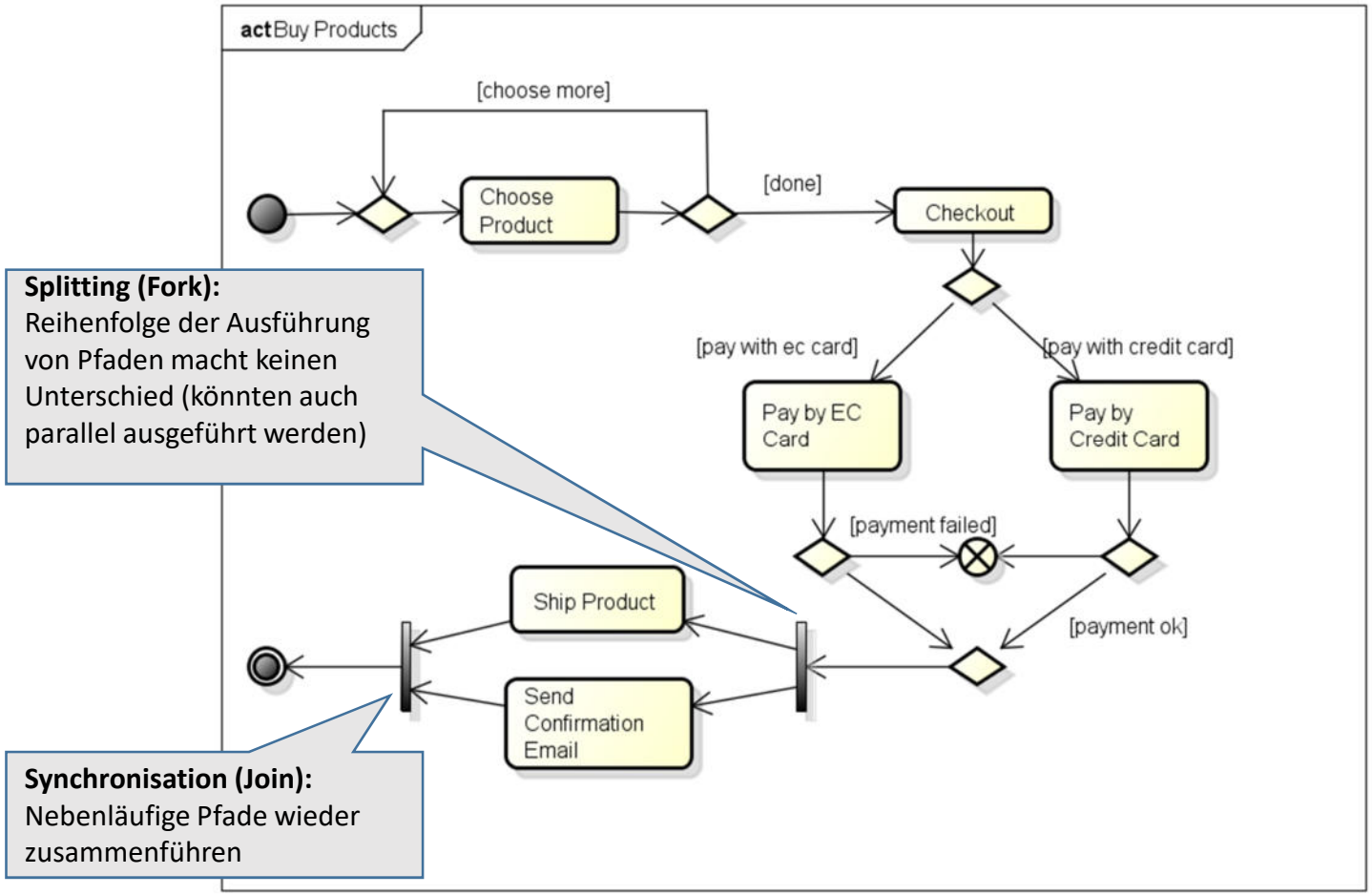
Natürlichsprachlich oder formal (z.B. OCL)



Falsch: Nicht vollständig oder nicht disjunkt



Splitting/Synchronisation

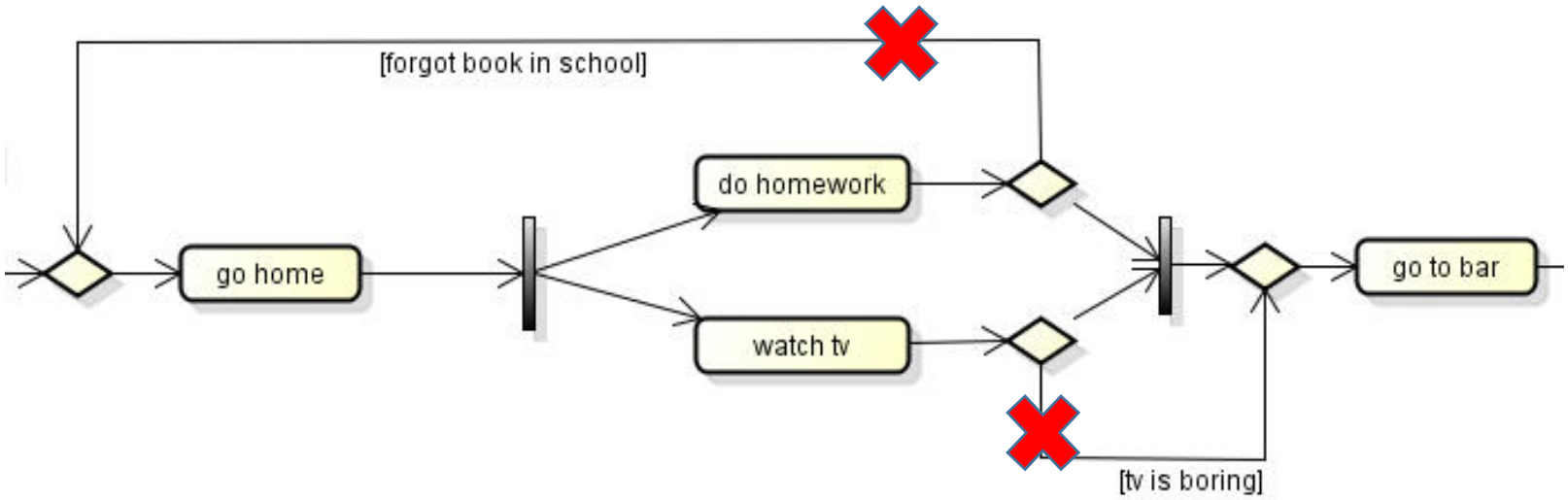


Splitting/Synchronisation

Der Kontrollfluss wird in mehrere Kontrollflüsse aufgeteilt bzw. wieder vereinigt (Tokens vervielfältigt oder vereinigt)

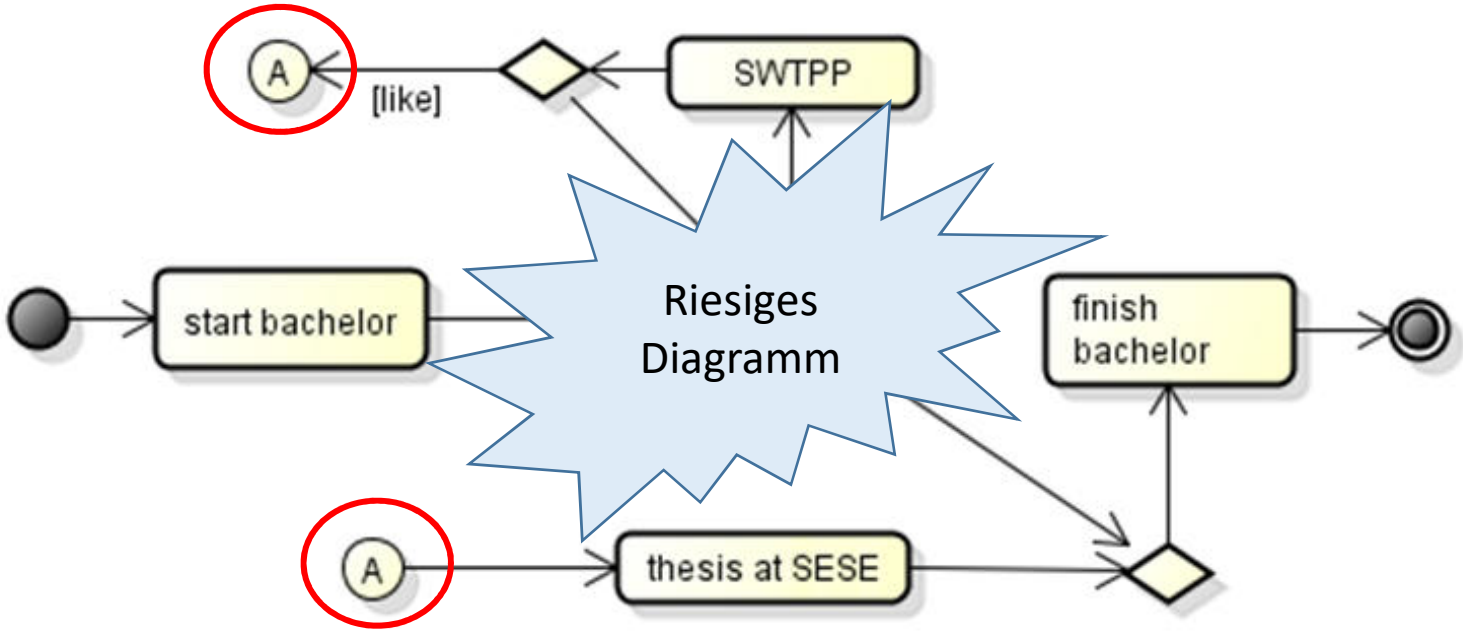
Wichtig: Getrennte Pfade werden i.A. entweder zusammengeführt oder einzeln gestartet / beendet.

➢ Verbindungen der nebenläufigen Pfade mit Elementen außerhalb Fork/Join sind meist nicht sinnvoll



Connector

Falls die Verbindungen zu unübersichtlich werden: **Überbrückung mit Connector** (nur Notation)



Datenfluss (Objekte)

Objekte fügen dem Kontrollfluss **Datenabhängigkeiten** hinzu, beinhalten Ergebnis einer Aktion

Darstellung durch Objektknoten (eckig)

- Transportieren Daten von einer Aktion zur nächsten

Alternative Pin-Notation für Objekte

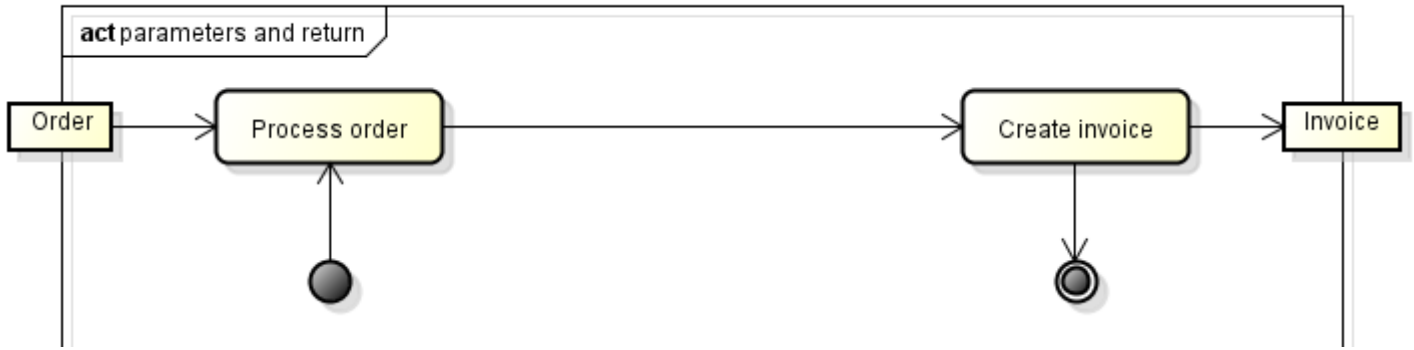
Parameter

Eingabeparameter einer Aktivität

- Wenn kein Startknoten vorhanden: Objektübergabe startet Aktivität
- Mehrere Parameter: Alle sind gleichzeitig verfügbar

Ausgabeparameter einer Aktivität

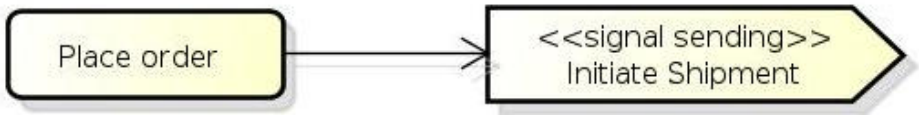
- Wenn kein Endknoten vorhanden: Objektrückgabe beendet Aktivität



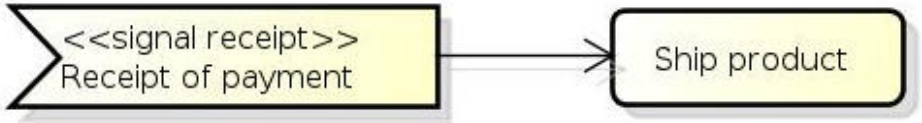
Ereignisse

Sonderform von Aktionen für **Interaktionen außerhalb der Aktivität**, z.B. Nachrichten mit Komponenten außerhalb des Systems

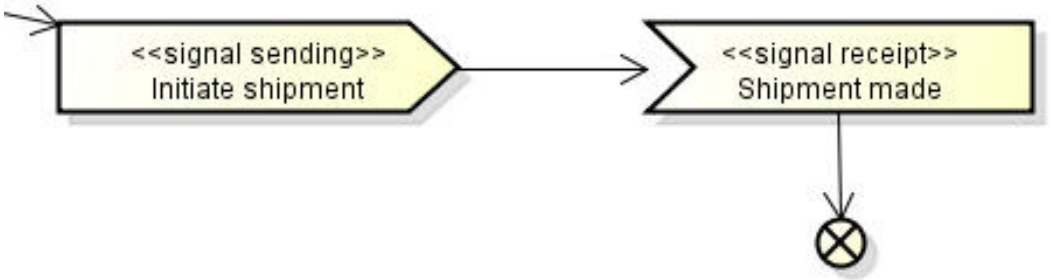
Ausgehende Signale



Eingehende Signale



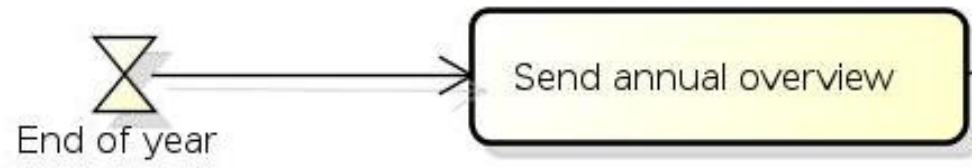
Können auch zusammen verwendet werden um auf externen Input zu warten



Zeitereignisse

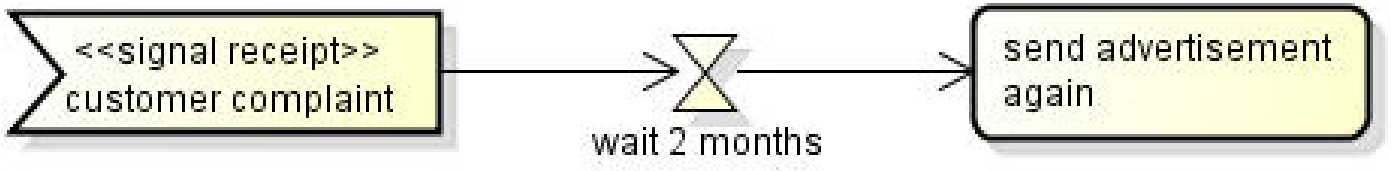
Beschreiben, dass Aktionen **zu bestimmten Zeitpunkten** ausgeführt werden sollen

- Möglich als **Start einer Aktivität**

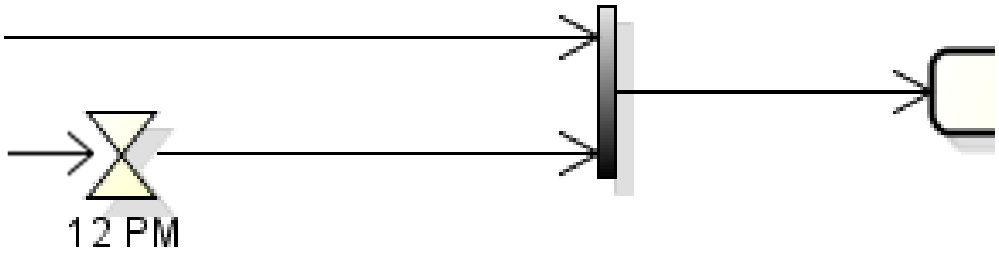


Zeitereignisse

Zeit kann auch explizit **vergehen** (Zeitereignis mit eingehender Kante)



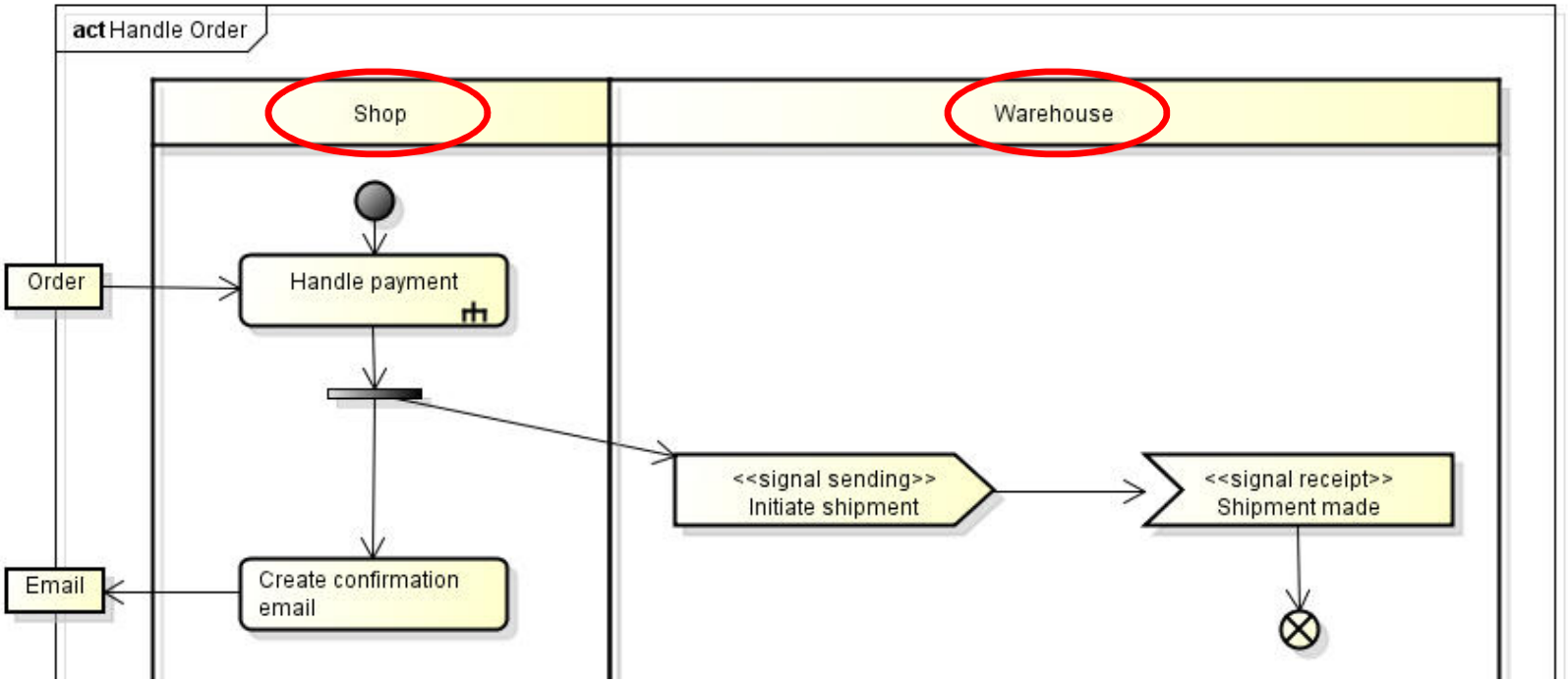
Verhalten kann außerdem mit Zeitereignissen **synchronisiert** werden



Aktivitätsbereich

Gliederung in z.B. organisatorische Einheiten oder Standorte

- Können auch externes Verhalten in einer Aktivität darstellen

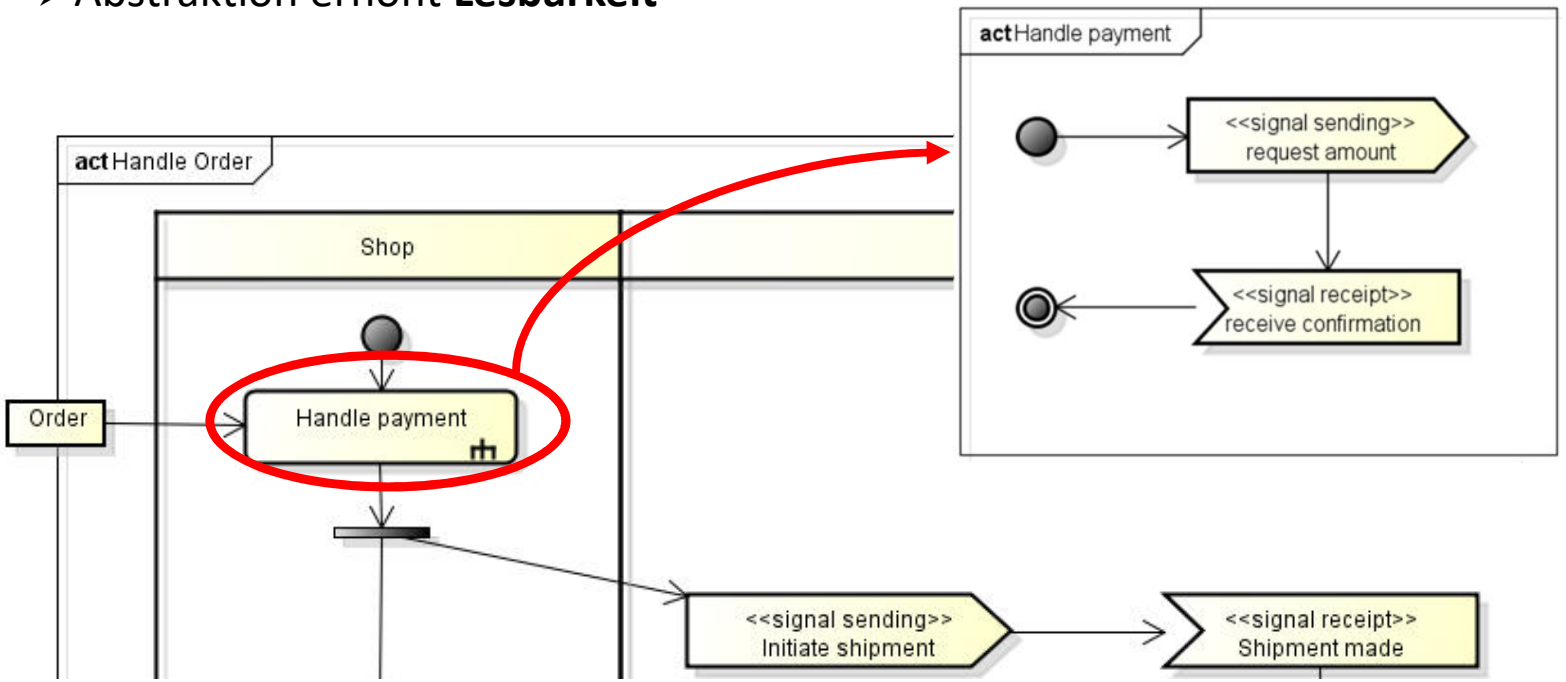


Aktivitätsaufruf

Darstellungsform für **komplexeres Verhalten** innerhalb einer Aktion

- Kann durch **separates Aktivitätsdiagramm** modelliert sein

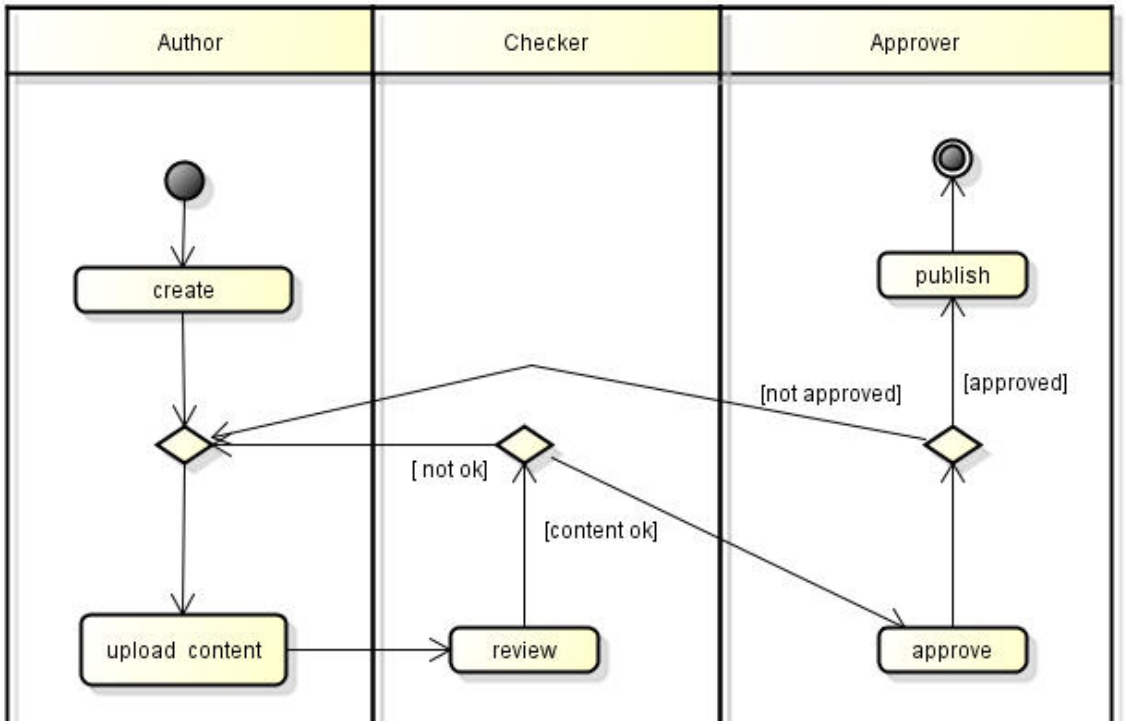
➢Abstraktion erhöht **Lesbarkeit**



Workflow

Aktivitätsdiagramm kann beliebig zur Modellierung von anderen **Abläufen** im System verwendet werden

- Beispiel für typischen Dokument-Workflow in einem Content Management System:



Inhalt

Analyse und Entwurf - Verhalten

- Grundlagen
- Aktivitätsdiagramm

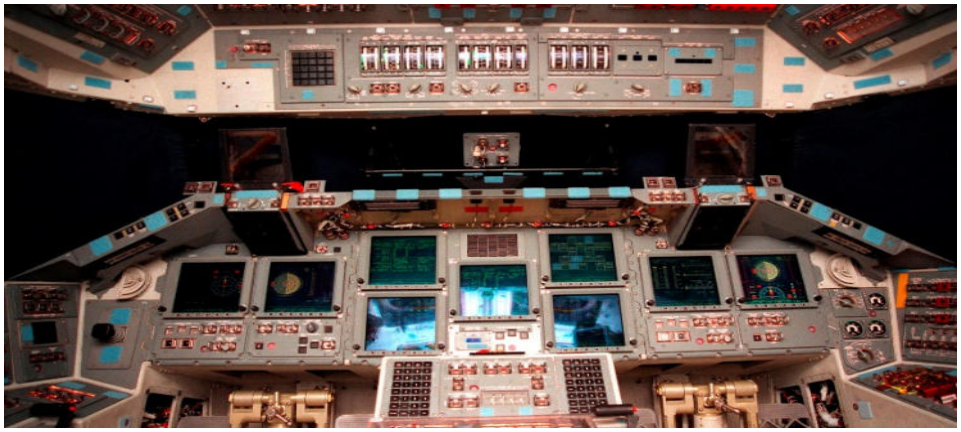
Model Driven Development (Ausblick)

Model Driven Development

Nun sind wir in der Lage unser System zu modellieren.

➢Wäre es nun nicht schön an dieser Stelle auch mit der Entwicklung fertig zu sein?

Leider sind die Anforderungen oft sehr kompliziert…



➢Wäre es nicht gut, wenn ein Domänen-Experte in der Lage wäre, das Modell selbst zu erstellen?

Bild: http://www.nasa.gov/centers/langley/news/releases/2000/GCTHUMBNAILS.html

Motivation

Softwareentwicklung ist kein Selbstzweck, sondern dient der **Lösung domänenspezifischer Probleme**

- Softwaresysteme sind **eingebettet** in einen domänenspezifischen Kontext

An der klassischen Softwareentwicklung sind unter anderem **zwei Gruppen** wesentlich beteiligt

- Sachkundige **Experten** (Domänenspezifisches Wissen)
- **Informatiker** (Methodisches Wissen)

Die **Verständigung** ist oft mühsam und anfällig für **Missverständnisse**

Table I

Standish project benchmarks over the years

Year	Successful (%)	Challenged (%)	Failed (%)
1994	16	53	31
1996	27	33	40
1998	26	46	28
2000	28	49	23
2004	29	53	18
2006	35	46	19
2009	32	44	24

Model-Driven Engineering

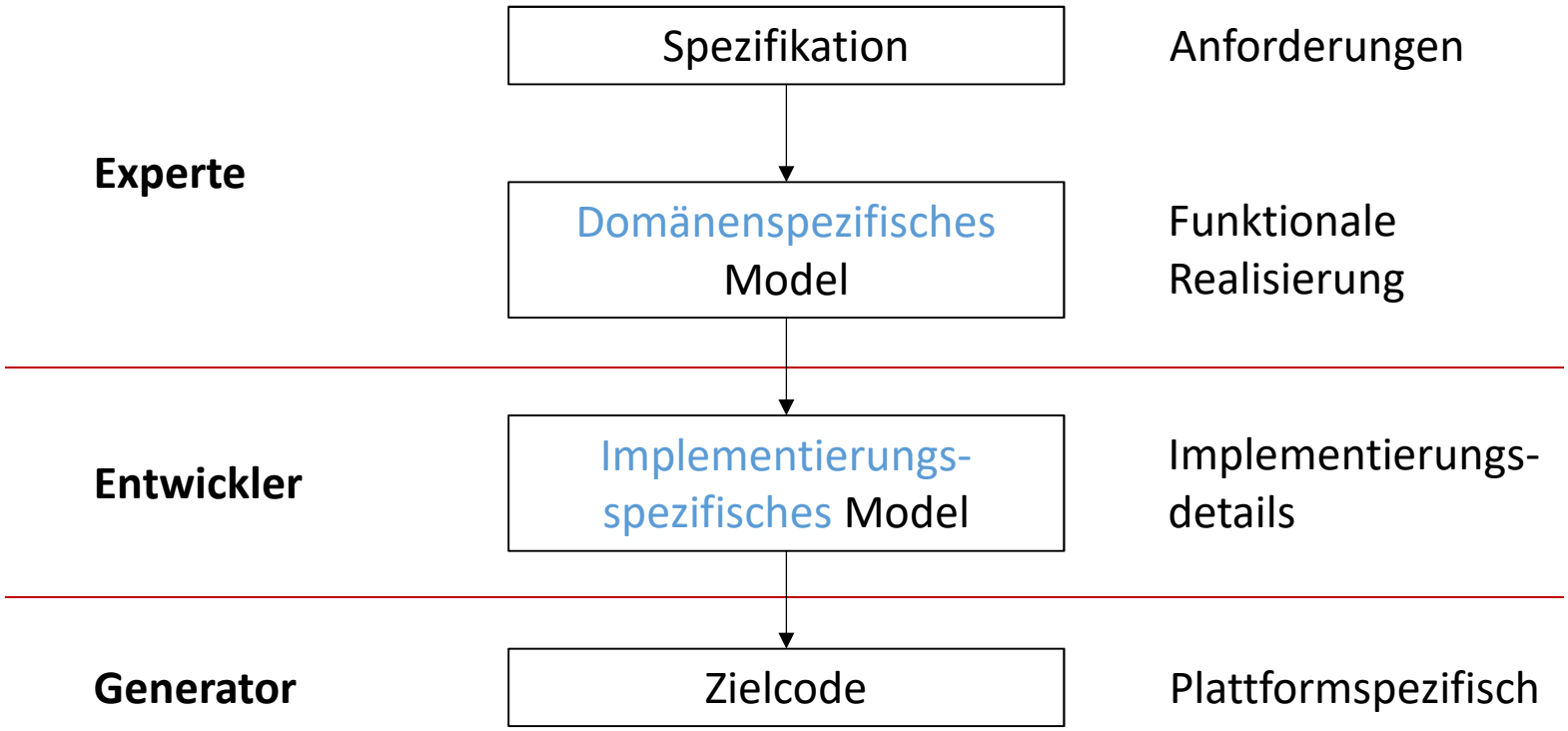
Bei der **modellgetriebenen Softwareentwicklung** steht die Erstellung eines **abstrakten Modells** im Vordergrund

- Erlaubt Entwicklung auf hohem Abstraktionsniveau
- Reduziert die Wahrscheinlichkeit von technischen Fehlern
- Erhöht die Wiederverwendbarkeit und verringert Redundanz
- Entwurfs- und Implementierungsprozess wird beschleunigt
- Die Modelle sind plattformunabhängig (frei von Ausführungsdetails)
- Domänenspezifische Modelle können direkt von Experten erstellt werden

Das endgültige Produkt kann aus dem Modell **abgeleitet** werden

- Manuell: Durch klassische Implementierung
- **Automatisch**: Durch Code-Generatoren

Model-Driven Development (MDD)



Von UML zu Code

UML bietet viele **Abstraktionsmöglichkeiten** und Gestaltungsmittel ohne eindeutige Semantik

- ➢Keine direkt Erstellung von **ausführbaren** Systemen möglich
- ➢Aber: Tools bieten die Möglichkeit **Skelette** zu erzeugen

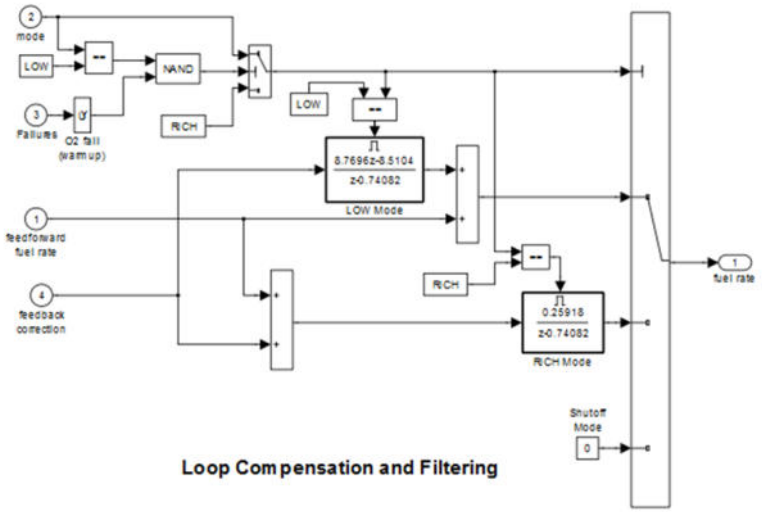
xUML (Executable UML)

- **Klassendiagramme** beschreiben die **domänenspezifische Struktur**
- **Zustandsmodelle** beschreiben den **Lebenszyklus** für jede Klasse
- **OCL** oder eine **UML Action Language** beschreiben das **Verhalten**
- Zusätzliche **Ausführungssemantik** und Zeitverhalten ermöglichen die automatische Generierung von **plattformspezifischen Modellen**

Modellierungssprachen

MATLAB **Simulink**

- **Datenflussorientierte** Sprache
- Funktionalität wird durch Blöcke modelliert, Datenfluss durch Verbindungen
- Ermöglicht physikalische Modellierung durch kontinuierliche Blöcke
- Breites Angebot **komplexer Blöcke** von verschiedenen Herstellern
- Gut für die Auslegung von **Regelungssystemen** geeignet

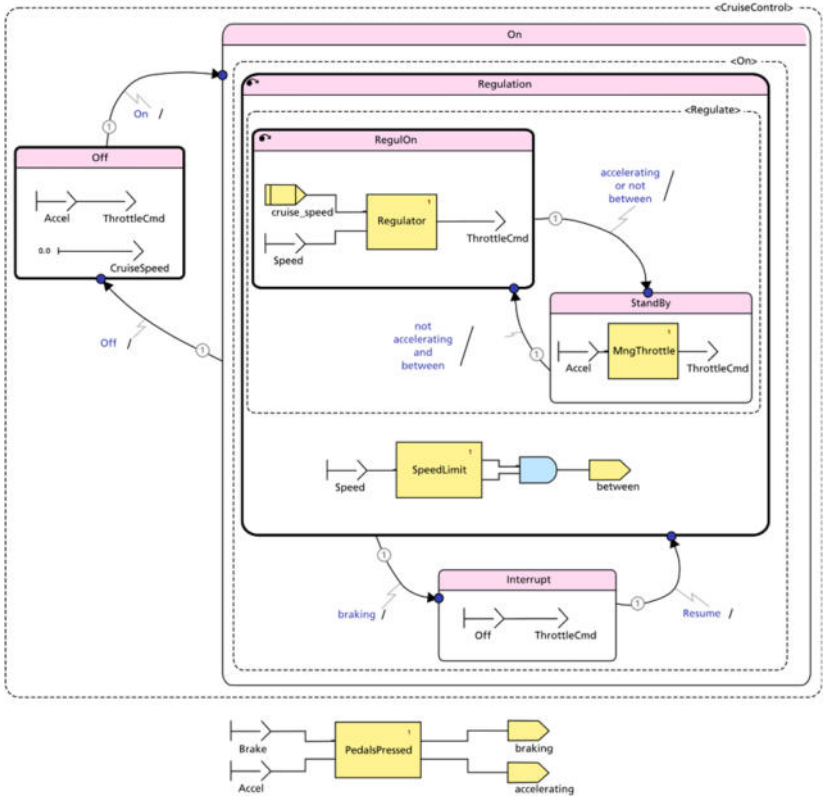


[Fuelsys.mdl, Simulink Demos, The Mathworks]

Modellierungssprachen

SCADE (Esterel Technologies)

- Basiert auf **LUSTRE** (synchron, datenflussorientiert)
- Erlaubt beliebige Mischung von **Datenfluss-** und **Zustands-modellierung** in einem Diagramm
- Import von UML- und Simulink-Modellen möglich
- Formale **Verifikation** der Modelle möglich
- Nach DO-178B für die Entwicklung von Avionik-Software zertifiziert

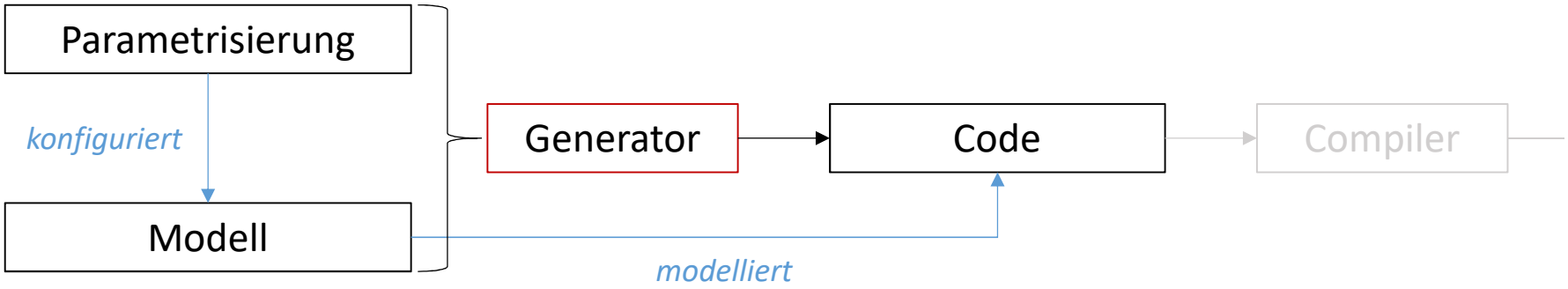


https://de.wikipedia.org/wiki/SCADE

Generative Programmierung

In der **generativen Programmierung** wird Programmcode **automatisch** durch einen Generator **erzeugt**

- Eine parametrisiert **Eingabespezifikation** (Modell) wird **konfiguriert** und in eine konkretisierte **Ausgabespezifikation** (Code) **übersetzt**
- Das geschieht entweder **statisch** (Entwickler) oder **dynamisch** (Kunde)
- Ermöglicht die gleichzeitige Entwicklung mehrerer **Programmvarianten**



Von Simulink zu Code

Simulink-Modelle können durch verschiedene Generatoren in **unterschiedliche Zielsprachen** übersetzt werden

- **Simulink Coder** erzeugt **parametrisierten Zielcode**, der in verschiedene Zielsprachen übersetzt werden kann
- **Embedded Coder** erzeugt Zielcode, der **Festkomma-Arithmetik** unterstützt, falls die Zielplattform nicht über Gleitkommaeinheiten (FPUs) verfügt
- **HDL-Coder** erzeugt VHDL-Code der zur Synthese von **Hardware** (z.B. FPGAs) verwendet werden kann
- **PLC Coder** erzeugt IEC 61131-3 Code für die **Automatisierungstechnik** (Kraftwerke, Fabriken, …)

Modellbasiertes Testen

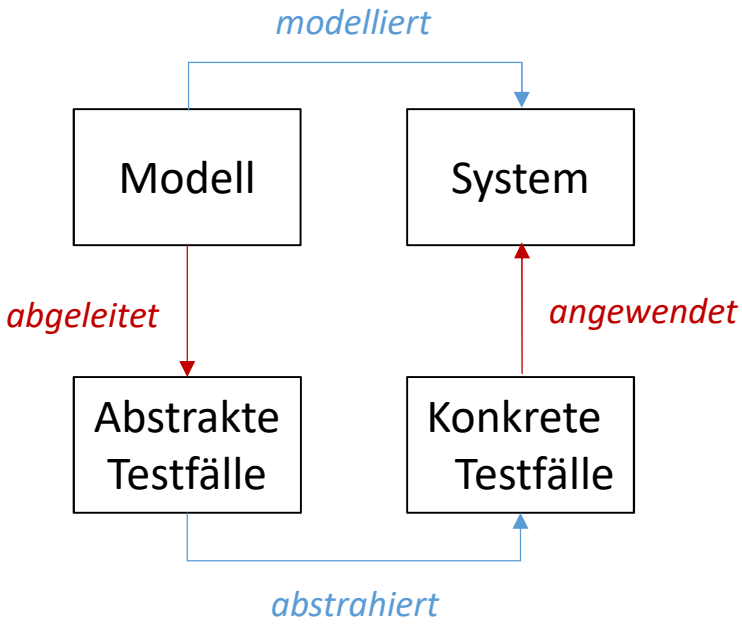
Auch **Testfälle** können direkt aus Modellen **abgeleitet** werden

- Die Modelle dienen dann als **Spezifikation** für das **System**

Erstellung und **Anwendung** von Testfällen kann automatisiert werden

- Personenunabhängige Qualität der Testfälle
- Höhere Transparenz und Wiederverwendbarkeit

Achtung: Modell und Testfallgenerator müssen validiert/verifiziert werden!



Domänenspezifische Sprachen

General-Purpose Languages, GPLs

C, C++, C#, Python, Java, Haskell…

- Meist **Turing-Mächtig**
- Domänenspezifische Probleme müssen erst in generelle Form **überführt** werden
- Resultierende Programme enthalten meist **wenig** direkt zugängliche **Information** über die Domäne

Domain-Specific Languages, DSLs

SQL, VHDL, MATLAB, EN 61131-3 …

- Auf Domäne **zugeschnitten**
- **Direkte Beschreibung** domänenspezifischer Probleme
- Resultierende Programme sind oft **frei von „boilerplate“-Code**
- Für Endnutzer **leichter** zu erlernen

Lernziele

- ❑ Was ist das Ziel der Verhaltensmodellierung?
- ❑ Welche Diagramme bietet UML dafür an?
- ❑ Was stellen Interaktionsdiagramme dar und warum gibt es mehrere?
- ❑ Wofür können Interaktionsdiagramme eingesetzt werden?
- ❑ Was stellen Aktivitätsdiagramme dar?
- ❑ Wofür können Aktivitätsdiagramme eingesetzt werden?
- ❑ Was sind Aktionen?
- ❑ Welche Kontrollknoten gibt es und was bedeuten sie?
- ❑ Was ist der Unterschied zwischen Aktivitäts- und Kontrollflussendknoten?
- ❑ Warum müssen die Guards von Verzweigungen disjunkt und vollständig sein?
- ❑ In welcher Reihenfolge werden Pfade ausgeführt, die durch Splitting aufgeteilt wurden?
- ❑ Welche Ereignisse gibt es und was kann man damit ausdrücken?
- ❑ Was ist das Ziel von modellgetriebener Entwicklung (MDD)?