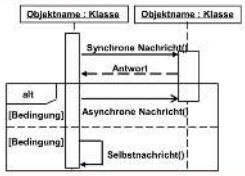
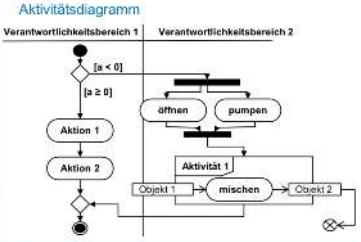
Überblick

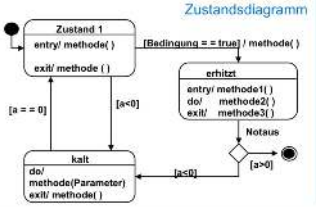
Sequenzdiagramm



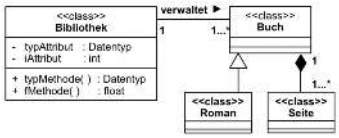
Aktivitätsdiagramm



Zustandsdiagramm



Klassendiagramm



Use-Case-Diagramm:
- formale Anforderungsdefinition aus Nutzersicht (Schnittstellen und Randbedingungen)

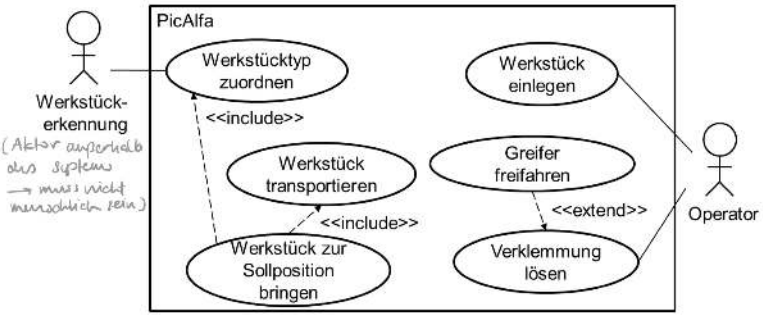
Sequenzdiagramm:
- modelliert Interaktion zwischen Objekten chronologisch
  → Nachrichtenaustausch und Reihenfolge
- detaillierte **Szenarien / Ausschnitte** des Systemverhaltens

Timing-Diagramm
- Präzise zeitliche Spezifikation des Verhaltens von Objekten
  → Entwurf von echtzeitkritischen Systemen

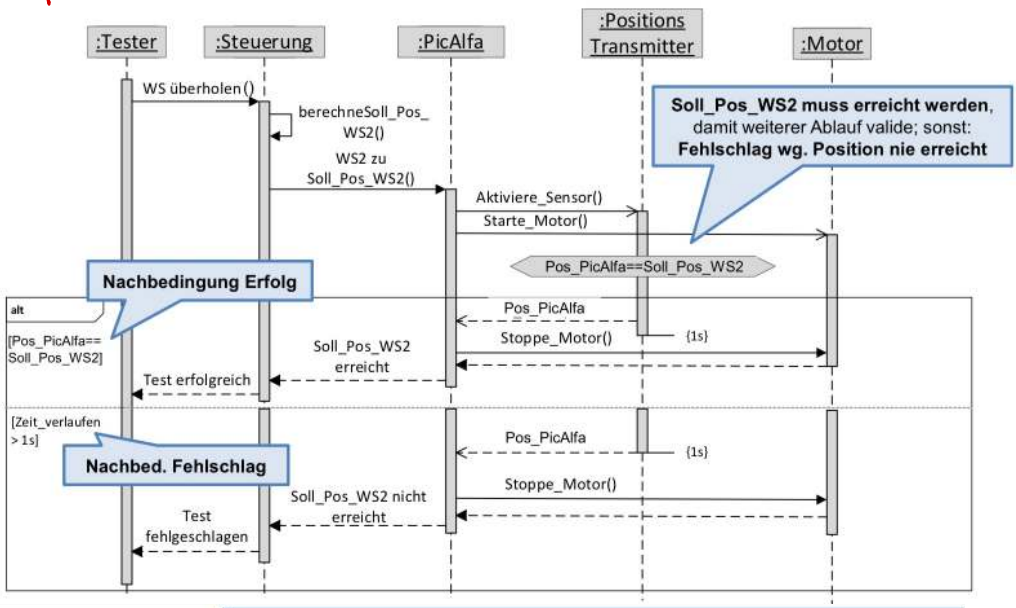
Aktivitätsdiagramm
- Darstellung des (abstrakten, möglichen) Gesamtverhaltens
- Maschinenbau-Sicht

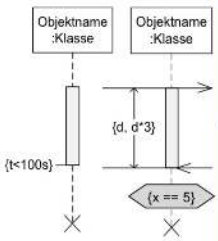
Zustandsdiagramm
- Internes Verhalten / Zustände von Objekten
- ermöglicht **detaillierte Beschreibung komplexer Systeme**
- Informatik- und Elektrotechniksicht

Use Case Diagramm



Sequenzdiagramm





Objekt: Das Objekt welches mit anderen Objekten interagieren möchte.

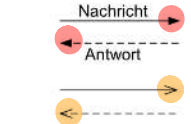
Lifeline: Die Lifeline ist die Lebenslinie des Objektes

Ausführung: Ist die Zeit die ein Objekt instanziiert wurde und Interaktionen ausführt.

Zeiteinschränkungen: Gibt Zeitanforderungen an das System an

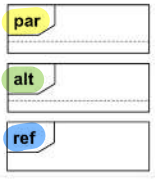
Komponente: ist modularer Systemteil mit transparenter Kapselung seines Inhalts. Sie besteht aus Elementen mit klar definierter Funktionalität und kann eine eigenständige Anwendung sein und lässt sich nur über Schnittstellen beschreiben

State-Invariant: Gibt eine boolesche Bedingung an (z.B. x==5). Sie muss erfüllt sein, damit der restliche Ablauf des Sequenzdiagramms noch valide ist.



Synchrone Kommunikation *(geschlossene Pfeilspitze)*
- Der Sender erwartet eine Antwort. Sowohl Sender als auch Empfänger können bis zum Erhalt der Antwort keine Prozesse ausführen

Asynchrone Kommunikation *(offene Pfeilspitze)*
- Der Sender erwartet keine Antwort. Das Senden und Empfangen von Daten ist zeitlich versetzt und blockiert keine Prozesse



Parallele Abarbeitung
- Ermöglicht die Modellierung von parallel ablaufenden Fragmenten

Alternative Abarbeitung (if)
- Ermöglicht die Modellierung von alternativen ablaufenden Fragmenten

Interaktionsverweis
- Beschreibt einen Teilablauf, definiert durch ein anderes Sequenzdiagramm („Black Box")

Aktivitätsdiagramm

Eine **Aktivität** beschreibt Abläufe, die aus mehreren elementaren Aktionen bestehen.
<<precondition>>: Aktivität nur gestartet, wenn die Vorbedingung erfüllt ist.
<<postcondition>>: Nach Beenden der Aktivität, muss Nachbedingung erfüllt werden.

Durch **Aktionsknoten** werden die elementaren Ablaufschritte oder unterlagerte Aktivitäten beschrieben. Es ist das elementarste Modellelement zur Beschreibung von Verhalten.

Alternativ können **Objektknoten** durch Rechtecke dargestellt werden. Sie können als ein- oder ausgehende Parameter in Aktivitäten verwendet werden.

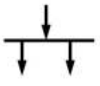
Aktivitätskanten sind gerichtete Beziehung zwischen zwei Knoten einer Aktivität. Es wird zwischen Kontroll- und Objektfluss unterschieden. Beide werden grafisch gleich notiert. Beim Objektfluss führt der Pfeil von oder zu einem Objektknoten.

Ein **Startknoten** ist ein Startpunkt eines Ablaufes in einem Aktivitätsdiagramm.

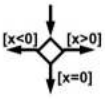
Ein **Endknoten** beendet den beschriebenen Ablauf, d.h. alle Aktionen und Kontrollflüsse.

Ein **Ablaufende** beendet einen einzelnen Objekt- und Kontrollfluss.

Eine **Synchronisation** ist ein Kontrollknoten, bei dem auf alle eingehenden Kontrollflüsse gewartet wird, bevor der Kontrollfluss fortgesetzt wird (**UND-Verknüpfung**).

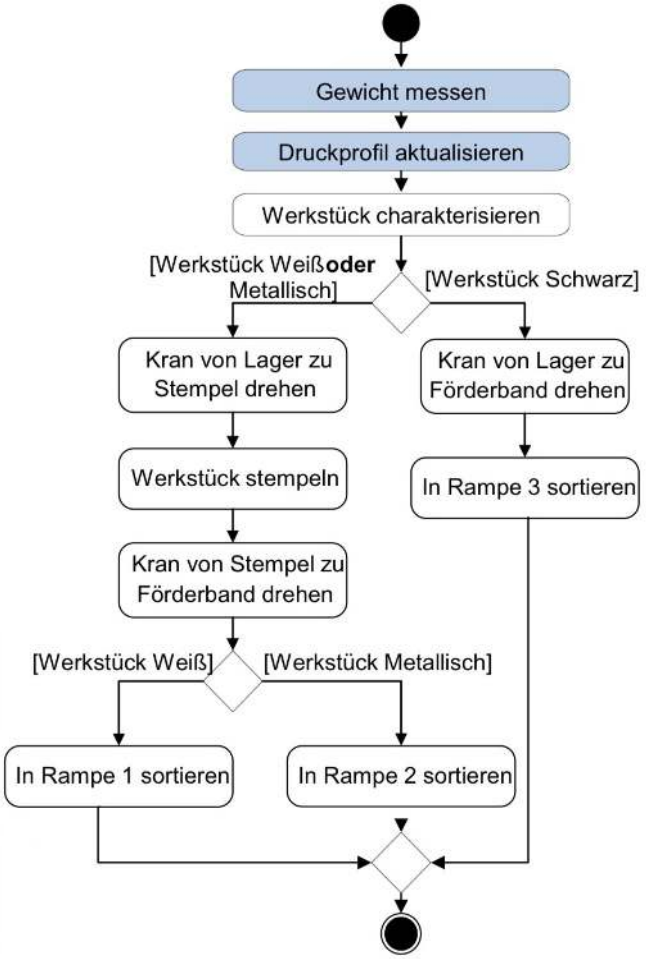


Eine **Teilung** ist ein Schritt im Ablauf, an dem ein eingehender Kontrollfluss ohne Bedingung sofort in mehrere ausgehende nebenläufige Kontrollflüsse geteilt wird.

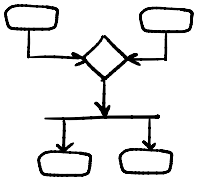


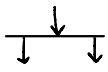
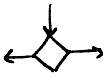
Eine **Entscheidung** ist ein Kontrollknoten mit einem oder mehr ausgehenden Kontrollflüssen, an dem aufgrund von Bedingungen entschieden wird, welcher Kontrollfluss fortgesetzt werden soll.

Eine **Zusammenführung** ist ein Kontrollknoten, bei dem jeder von mehreren eingehenden Kontrollflüssen sofort zum ausgehenden Kontrollfluss führt (**ODER-Verknüpfung**).



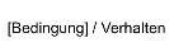
Anm:



!  und  müssen wieder zusammengeführt werden

Zustandsdiagramm

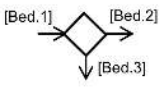
Der **Zustand** (state) repräsentiert eine Menge von Wertekombinationen des zugehörigen Elements. Er hat einen Namen und gegebenenfalls ein internes Verhalten, das aufgrund definierter Ereignisse ausgeführt wird.



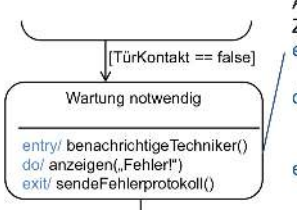
Die **Transition** (transition) spezifiziert einen Zustandsübergang. Sie ist eine gerichtete Beziehung zwischen zwei Zuständen und definiert einen Auslöser und eine Bedingung, die zum Zustandsübergang führen, sowie ein Verhalten, dass während des Übergangs ausgeführt wird.

Der **Startzustand** (initial state) ist eine ausgehende Transition, die auf den initialen Zustand zeigt.

Endzustand (final state)



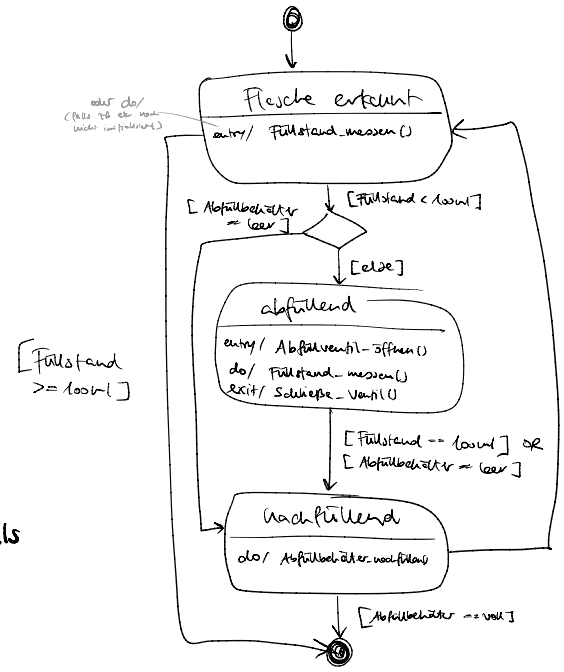
Entscheidungs- /Verzweigungspunkt (junction point) ermöglicht Transitionen miteinander zu Verknüpfen bzw. zu Verzweigen



Aktionen bzw. Ereignisse im Zustandsdiagramm:
entry/-Aktion: wird ausgeführt, sobald der **Zustand betreten** wird
do/-Aktion: wird ausgeführt, solange der **Zustand aktiv** oder bis er abgeschlossen ist.
exit/-Aktion: wird ausgeführt, sobald der **Zustand verlassen** wird

Transitionen im Zustandsdiagramm:
Trigger[Bedingung]/Aktion

(können jeweils mehrere Funktionen enthalten)



Klassendiagramm

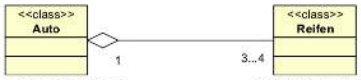
Assoziation
(eine *nicht existenzabhängige Verbindung* zwischen Klassen)

→ Pointer

Kardinalitäten: Kunde kann beliebig viele Konten haben, jedes Konto gehört genau einem Kunden

Aggregation
(eine *nicht existenzabhängige Hierarchieverbindung* zwischen Klassen)

→ Attribut der Klasse (kann Pointer sein)



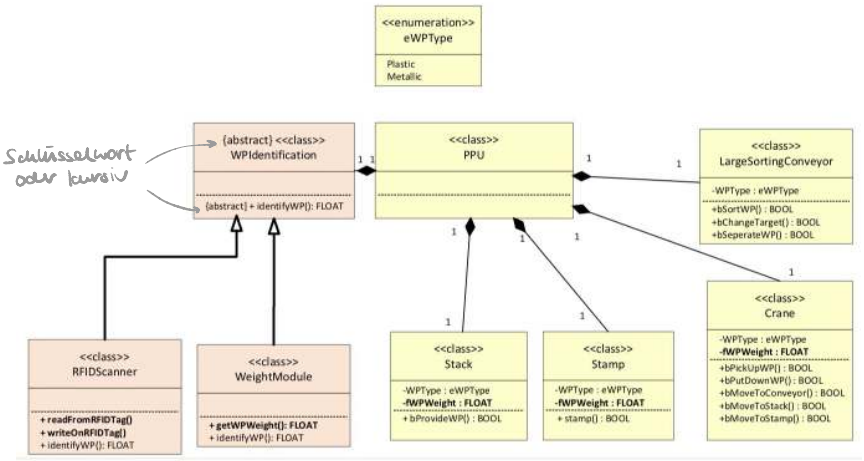
Aggregierende Klasse *Aggregierte Klasse*

Besteht-aus-Beziehung: Ein Auto besteht aus Reifen, Reifen können abmontiert und an andere Autos montiert werden

Komposition
(eine *existenzabhängige Hierarchieverbindung* zwischen Klassen)

→ Klasse in der Klasse definiert

Wenn das Auto zerstört wird, wird auch die Karosserie zerstört. Karosserie gehört immer nur zu einem Auto



! Herausforderung: alle Modelle müssen konsistent sein

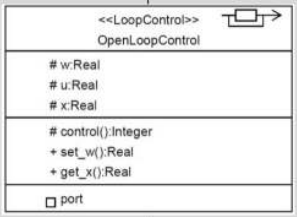
Stereotypen = Spezifikation / Erweiterung vorhandener UML-Meta-Elemente

UML-Profil: Metamodell für UML → Beschreibt neue Modellierungselemente, Constraints → spezifiziert, wie UML Diagramme zB. in Automotive aussehen müssen → einheitlich

Meta-Modell	**Modell**
• Grundlage für Modelle	• Basierend auf Metamodell
• Festlegung der Konstrukte (Objekte, Beziehungen, Constraints…) und Struktur der Modellierungssprache	• Repräsentation des realen Systems im betrachteten Bereich, d.h. konkreter Sachverhalt
• vgl. Deutsche Sprache: Welche Wörter nutzbar (z.B. „Software“, „entwickeln“, „ist“, „mit“, „Modell“ und „einfach“)?	• vgl. Deutsche Sprache: „Software entwickeln mit Modell ist einfach“

→ **Meta-Modell** und **Modell** stehen in einer Klasse-Instanz-Beziehung zueinander
→ **Modell** ist Instanz von **Meta-Modell** (instance-of Beziehung)

Stereotypen
- Spezifiziert Anpassung/ Erweiterung vorhandener UML-Meta-Elemente
  - z.B. Erweiterung von UML Klassen, Methoden, Attributen, Assoziationen
- **Stereotypen** können nach ihrer Definition wie Standard-UML-Meta-Elemente verwendet werden



Constraints
- Definition von zusätzlichen Einschränkungen an das UML-Profil, die im UML-Meta-Modell nicht formuliert werden können
- Können über Vererbung an weitere Stereotypen übergeben werden

Auf die Outputs des Reglers Zugriff nur über einen definierten Port.