

# Kapitel 1

## Software-Entwicklung und formale Spezifikation

Prof. Dr. Rolf Hennicker

22.04.2010

## Ziele

- ▶ Die Grundprinzipien der Software-Entwicklung verstehen.
- ▶ Die Rolle formaler Methoden in der Software-Entwicklung verstehen.
- ▶ Die Grundprinzipien objektorientierter Systeme und ihrer formalen Spezifikation verstehen.

## 1.1 Einführung

### **Software Engineering**

befasst sich mit den technischen und organisatorischen Aspekten der Entwicklung und Wartung von großen Software-Systemen.

### **State of the Art**

- ▶ Dokumentationen in natürlicher Sprache
- ▶ graphische Modellierungssprachen (z.B. UML)
- ▶ CASE Tools (Computer Aided Software Engineering)
- ▶ formale Methoden (für sicherheitskritische Systeme)

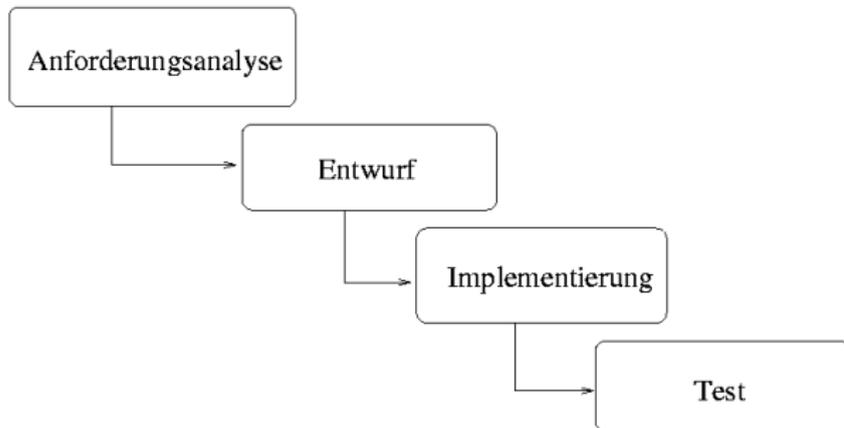
## Formale Methoden

- ▶ basieren auf der Mathematik (Mengenlehre, Algebra, math. Logik)
- ▶ Anwendungen:
  - ▶ formale Semantik von Programmier- und Spezifikationssprachen (Bereichstheorie)
  - ▶ formale Spezifikation (von Eigenschaften) von Programmen (Logik)
  - ▶ Korrektheitsbeweise (z.B. Hoare-Kalkül)
- ▶ Vorteile:
  - ▶ eindeutige Interpretation syntaktischer Konstrukte
  - ▶ unmissverständliche Spezifikation von Systemeigenschaften
  - ▶ Analyse und Verifikation von Eigenschaften (von Spezifikationen, Modellen, Programmen)
  - ▶ Verifikation der Korrektheit von Entwicklungsschritten
- ▶ Schwierigkeiten:
  - ▶ Kenntnis der formalen Notationen und ihrer Bedeutung
  - ▶ zusätzliche Entwicklungskosten

## 1.2 Software-Entwicklungsprozess

### Prozessmodelle

- ▶ Wasserfallmodell



- ▶ iteratives Modell, V-Modell, Spiralmodell, ...

Die formale Software-Entwicklung benutzt zusätzlich *formale Spezifikationen* in den verschiedenen Prozessphasen. Die Korrektheit eines Entwicklungsschrittes kann nur auf der Grundlage formaler Spezifikationen verifiziert werden.

## 1.3 Formale objektorientierte Software-Entwicklung

### **Objektorientiertes System**

besteht aus einer Menge von Objekten, welche durch Nachrichtenaustausch kommunizieren. Wenn ein Objekt eine Nachricht empfängt, so führt es eine Operation (Methode) aus, welche möglicherweise den Zustand des Objekts bzw. des Systems ändert.

### **Klasse**

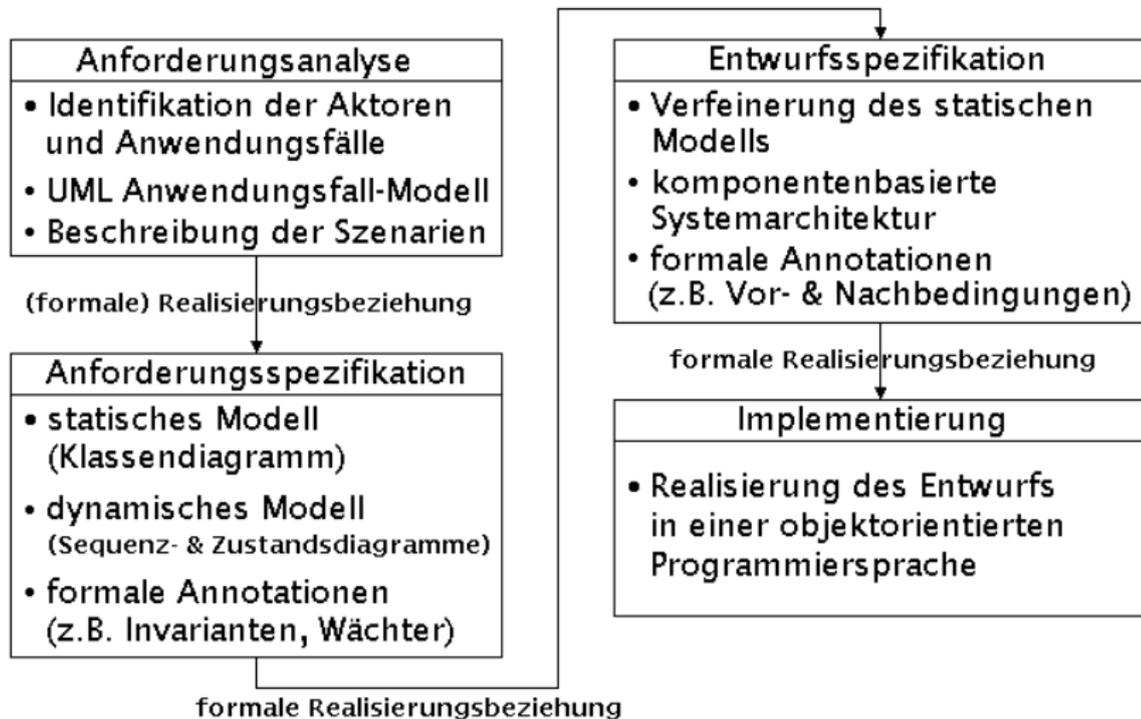
beschreibt eine Menge von Objekten mit gemeinsamen Eigenschaften (Attributen) und Operationen.

### **Objektzustand**

Der augenblickliche Wert der Attribute eines Objekts bestimmt den Zustand des Objekts (zum besteffenden Zeitpunkt).

### **Systemzustand**

Die Zustände aller existierenden Objekte und aller Objektverbindungen (links) bestimmen den aktuellen Systemzustand.



## Beweis der Korrektheit einer Implementierung

1. Implementierung gegenüber der Anforderungsspezifikation verifizieren  
(*post mortem*)  
**ODER**
2. jeden einzelnen Realisierungsschritt verifizieren

### Anmerkungen

- ▶ Testverfahren können nur die Existenz von Fehlern nachweisen.
- ▶ Verifikation kann die Abwesenheit von Fehlern nachweisen.
- ▶ Die Adäquatheit einer (formalen) Spezifikation in Bezug auf die Wünsche des Anwenders kann nicht verifiziert werden.

## OCL: Object Constraint Language

- ▶ ist eine formale Sprache zur Spezifikation (zusätzlicher) Constraints für UML-Modelle
- ▶ wurde ursprünglich von IBM (1995) entwickelt und basiert auf Syntropy
- ▶ sollte einfach zu benutzen sein (ohne tiefere mathematische Kenntnisse)
- ▶ ist eine streng typisierte, deklarative Sprache

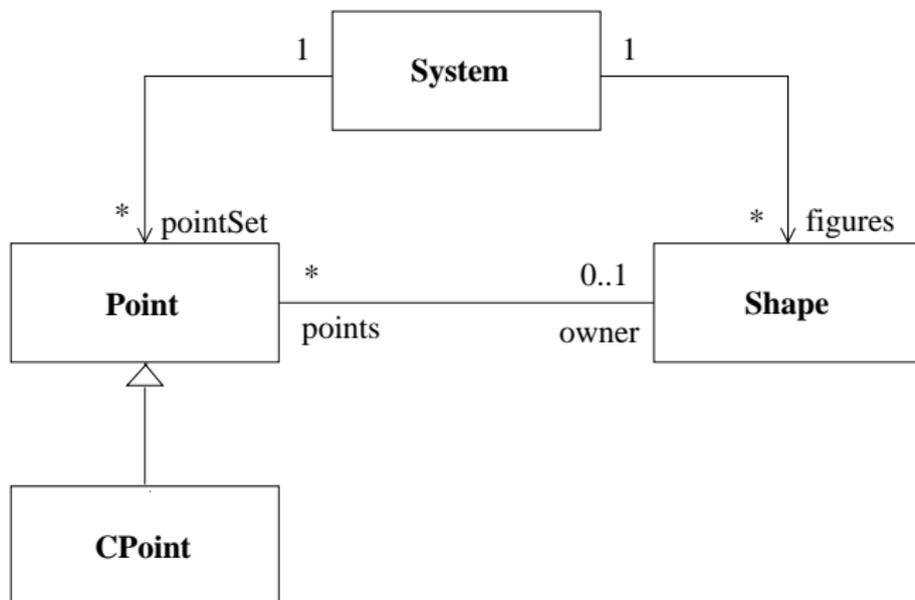
### Anwendungsmöglichkeiten von OCL

- ▶ Spezifikation von Klassen- und Komponenteninvarianten
- ▶ Spezifikation von Vor- und Nachbedingungen von Operationen
- ▶ Formulierung von Wächterbedingungen in Zustandsdiagrammen
- ▶ Formulierung von Prädikaten in Sequenzdiagrammen
- ▶ Definition von Anfragen (Queries)
- ▶ Spezifikation von Constraints für das UML-Metamodell (dh. zur Sprachdefinition)

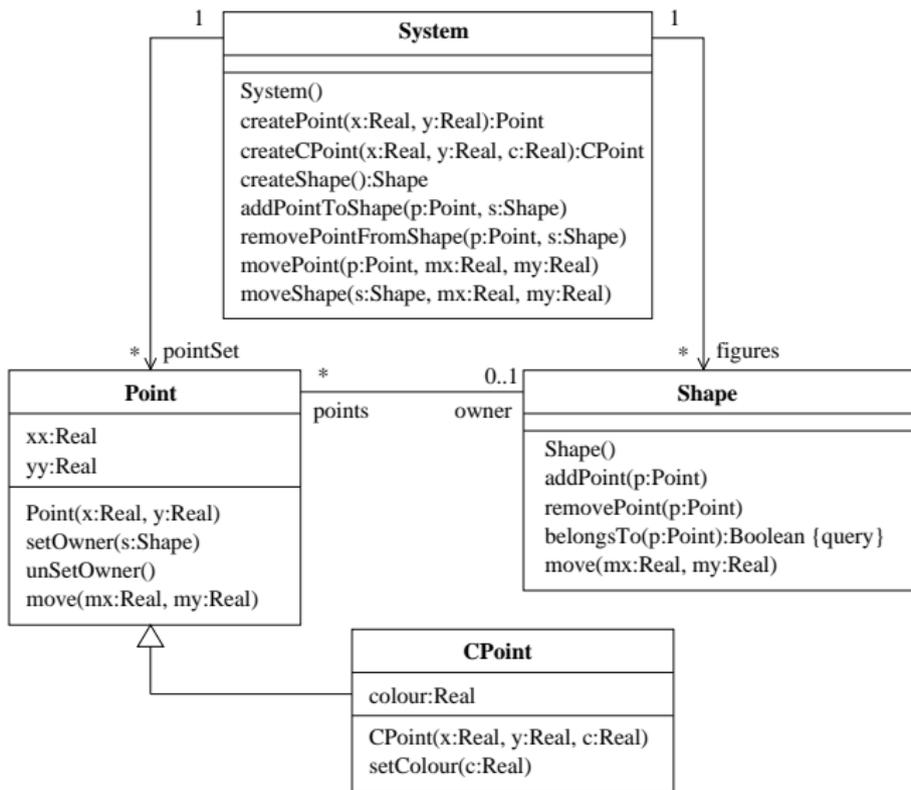
## 1.4 Beispiel: Punkte und Formen

### Anforderungen

- ▶ Zu entwickeln ist ein System zur Darstellung und Manipulation von Punkten (points) und geometrischen Formen (shapes)
- ▶ Eine geometrische Form besteht aus einer Menge von Punkten
- ▶ Jeder Punkt gehört zu höchstens einer geometrischen Form
- ▶ Ein farbiger Punkt (coloured point) ist ein Punkt mit einer Farbe
- ▶ Punkte und Formen werden von einer Systemklasse verwaltet
- ▶ Ein Punkt wird durch seine Koordinaten bestimmt
- ▶ Ein Punkt kann zu einer geometrischen Form hinzugefügt (add) oder aus ihr entfernt werden (remove)
- ▶ Für jeden Punkt kann geprüft werden, ob er zu einer bestimmten Form gehört oder nicht (belongsTo)
- ▶ Punkte und geometrische Formen können bewegt werden (move)



# Statisches Modell für Punkte und Formen



## OCL Constraints für Punkte und Formen

```
context System
```

```
  inv: pointSet -> forAll(p|  
    p.owner <> null implies  
    p.owner.points -> includes(p)) and  
  figures -> forAll(s|  
    s.points -> forAll(p|  
    p <> null implies p.owner = s))
```

```
context System::addPointToShape(p:Point, s:Shape)
```

```
  pre: p <> null and s <> null and  
    pointSet -> includes(p) and figures -> includes(s) and  
    p.owner = null and s.points -> excludes(p)  
  post: p.owner = s and  
    s.points = s.points@pre -> including(p)
```

```
context Point::move(mx:Real, my:Real)
```

```
  post: xx = xx@pre + mx and yy = yy@pre + my
```

## 1.5 Zusammenfassung

- ▶ Software-Entwicklung benutzt Prozessmodelle
- ▶ Formale Methoden können in allen Phasen der Software-Entwicklung eingesetzt werden
- ▶ Beweise der Korrektheit einer Implementierung setzen das Vorhandensein einer formalen Spezifikation voraus
- ▶ Ein objektorientiertes System besteht aus einer Menge von kommunizierenden Objekten
- ▶ OCL ist eine formale, deklarative Sprache zur Spezifikation von Eigenschaften objektorientierter Systeme (z.B. Invarianten, Vor- und Nachbedingungen)