

# Objektorientierte Analyse

## Software Engineering in der Praxis

David Föhrweiser   Marc Spisländer

Lehrstuhl für Software Engineering  
Friedrich-Alexander-Universität Erlangen-Nürnberg

1 Inhalt

2 Überblick

3 Werkzeuge

# Zweck der Objektorientierten Analyse

## Was soll das System leisten (nicht *wie*)

- Erkundung der Natur des gegebenen Problems
- Systemgrenzen, Außenschnittstellen identifizieren
- Hilfe bei der “Zerlegung” des Systems in Teile
- Abläufe, (grobe) Kontrollflüsse, Nebenläufigkeit
- Schaffung eines Ausgangspunktes für das Design

Lernprozess, kein Design!

# Zweck der Objektorientierten Analyse

## Was soll das System leisten (nicht *wie*)

- Erkundung der Natur des gegebenen Problems
- Systemgrenzen, Außenschnittstellen identifizieren
- Hilfe bei der “Zerlegung” des Systems in Teile
- Abläufe, (grobe) Kontrollflüsse, Nebenläufigkeit
- Schaffung eines Ausgangspunktes für das Design

Lernprozess, kein Design!

# Zweck der Objektorientierten Analyse

## Was soll das System leisten (nicht *wie*)

- Erkundung der Natur des gegebenen Problems
- Systemgrenzen, Außenschnittstellen identifizieren
- Hilfe bei der “Zerlegung” des Systems in Teile
- Abläufe, (grobe) Kontrollflüsse, Nebenläufigkeit
- Schaffung eines Ausgangspunktes für das Design

Lernprozess, kein Design!

# Zweck der Objektorientierten Analyse

## Was soll das System leisten (nicht *wie*)

- Erkundung der Natur des gegebenen Problems
- Systemgrenzen, Außenschnittstellen identifizieren
- Hilfe bei der “Zerlegung” des Systems in Teile
- Abläufe, (grobe) Kontrollflüsse, Nebenläufigkeit
- Schaffung eines Ausgangspunktes für das Design

Lernprozess, kein Design!

# Zweck der Objektorientierten Analyse

## Was soll das System leisten (nicht *wie*)

- Erkundung der Natur des gegebenen Problems
- Systemgrenzen, Außenschnittstellen identifizieren
- Hilfe bei der “Zerlegung” des Systems in Teile
- Abläufe, (grobe) Kontrollflüsse, Nebenläufigkeit
- Schaffung eines Ausgangspunktes für das Design

Lernprozess, kein Design!

# Zweck der Objektorientierten Analyse

## Was soll das System leisten (nicht *wie*)

- Erkundung der Natur des gegebenen Problems
- Systemgrenzen, Außenschnittstellen identifizieren
- Hilfe bei der “Zerlegung” des Systems in Teile
- Abläufe, (grobe) Kontrollflüsse, Nebenläufigkeit
- Schaffung eines Ausgangspunktes für das Design

Lernprozess, kein Design!



# Zweck der Objektorientierten Analyse

**Was** soll das System leisten (**nicht wie**)

- Erkundung der Natur des gegebenen Problems
- Systemgrenzen, Außenschnittstellen identifizieren
- Hilfe bei der “Zerlegung” des Systems in Teile
- Abläufe, (grobe) Kontrollflüsse, Nebenläufigkeit
- Schaffung eines Ausgangspunktes für das Design

**Lernprozess, kein Design!**

# Use-Case Analyse

## UML Anwendungsfall-Diagramme

- Fokus auf das äußere funktionale Verhalten ("Feature")
- Nutzer, Akteure
- Anwendungsfälle
- Systemgrenzen
- Beziehungen zwischen Akteuren und Anwendungsfällen

Systematische Erhebung der Ereignisse, die die Systemgrenzen passieren

# Use-Case Analyse

## UML Anwendungsfall-Diagramme

- Fokus auf das äußere funktionale Verhalten (“Feature”)
- Nutzer, Akteure
- Anwendungsfälle
- Systemgrenzen
- Beziehungen zwischen Akteuren und Anwendungsfällen

Systematische Erhebung der Ereignisse, die die Systemgrenzen passieren

# Use-Case Analyse

## UML Anwendungsfall-Diagramme

- Fokus auf das äußere funktionale Verhalten (“Feature”)
- Nutzer, Akteure
- Anwendungsfälle
- Systemgrenzen
- Beziehungen zwischen Akteuren und Anwendungsfällen

Systematische Erhebung der Ereignisse, die die Systemgrenzen passieren

# Use-Case Analyse

## UML Anwendungsfall-Diagramme

- Fokus auf das äußere funktionale Verhalten (“Feature”)
- Nutzer, Akteure
- Anwendungsfälle
- Systemgrenzen
- Beziehungen zwischen Akteuren und Anwendungsfällen

Systematische Erhebung der Ereignisse, die die Systemgrenzen passieren

# Use-Case Analyse

## UML Anwendungsfall-Diagramme

- Fokus auf das äußere funktionale Verhalten (“Feature”)
- Nutzer, Akteure
- Anwendungsfälle
- Systemgrenzen
- Beziehungen zwischen Akteuren und Anwendungsfällen

Systematische Erhebung der Ereignisse, die die Systemgrenzen passieren

# Use-Case Analyse

## UML Anwendungsfall-Diagramme

- Fokus auf das äußere funktionale Verhalten (“Feature”)
- Nutzer, Akteure
- Anwendungsfälle
- Systemgrenzen
- Beziehungen zwischen Akteuren und Anwendungsfällen

Systematische Erhebung der Ereignisse, die die Systemgrenzen passieren

# Use-Case Analyse

## UML Anwendungsfall-Diagramme

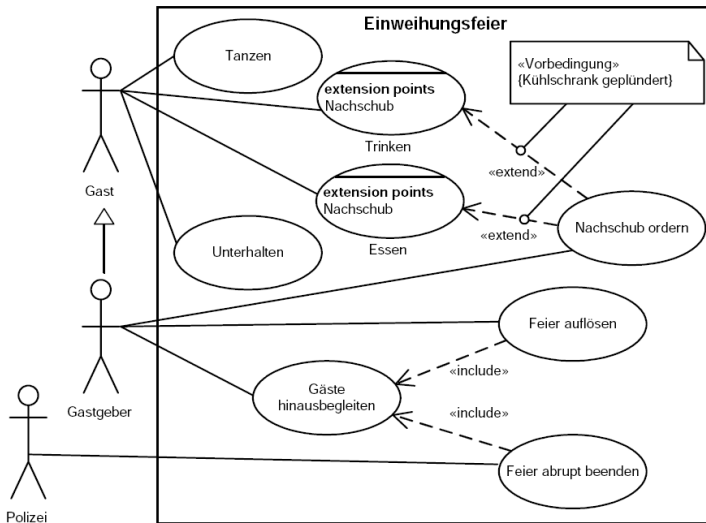
- Fokus auf das äußere funktionale Verhalten (“Feature”)
- Nutzer, Akteure
- Anwendungsfälle
- Systemgrenzen
- Beziehungen zwischen Akteuren und Anwendungsfällen

Systematische Erhebung der Ereignisse, die die Systemgrenzen passieren



# Use-Case-Diagramm

## Beispiel



# Analyse der automatischen Produktionszelle

## Use-Case Diagramme?

- Betrachte Nachbarsysteme als Nutzer
- Zeichne UseCase-Diagramme für die Systemteile

*Oder wähle einen geeigneteren Diagrammtyp aus!*

# Analyse der automatischen Produktionszelle

## Use-Case Diagramme?

- Betrachte Nachbarsysteme als Nutzer
- Zeichne UseCase-Diagramme für die Systemteile

*Oder wähle einen geeigneteren Diagrammtyp aus!*

# Analyse der automatischen Produktionszelle

## Use-Case Diagramme?

- Betrachte Nachbarsysteme als Nutzer
- Zeichne UseCase-Diagramme für die Systemteile

*Oder wähle einen geeigneteren Diagrammtyp aus!*

# Analyse der automatischen Produktionszelle

## Use-Case Diagramme?

- Betrachte Nachbarsysteme als Nutzer
- Zeichne UseCase-Diagramme für die Systemteile

*Oder wähle einen geeigneteren Diagrammtyp aus!*

# Aktivitätsdiagramme

## Fokus auf Abläufe, die im System stattfinden

- Geschäftsprozesse, Systemprozesse, Methoden einer Klasse
- Eignet sich für Analyse *und* Design
- Aktivität := Menge potentieller Abläufe
- Kontrollelemente: Verzweigung, Vereinigung, Fork, Join
- Elemente: Aktionen, Kanten, Objektknoten, [Pin-Notation ... ]
- Kontrolltoken, Datentoken
- uvm.

Inspiziert von Petri-Netzen

# Aktivitätsdiagramme

Fokus auf Abläufe, die im System stattfinden

- Geschäftsprozesse, Systemprozesse, Methoden einer Klasse
- Eignet sich für *Analyse und Design*
- Aktivität := Menge potentieller Abläufe
- Kontrollelemente: Verzweigung, Vereinigung, Fork, Join
- Elemente: Aktionen, Kanten, Objektknoten, [Pin-Notation ... ]
- Kontrolltoken, Datentoken
- uvm.

Inspiziert von Petri-Netzen

# Aktivitätsdiagramme

Fokus auf Abläufe, die im System stattfinden

- Geschäftsprozesse, Systemprozesse, Methoden einer Klasse
- Eignet sich für *Analyse und Design*
- Aktivität := Menge potentieller Abläufe
- Kontrollelemente: Verzweigung, Vereinigung, Fork, Join
- Elemente: Aktionen, Kanten, Objektknoten, [Pin-Notation ... ]
- Kontrolltoken, Datentoken
- uvm.

Inspiziert von Petri-Netzen



# Aktivitätsdiagramme

Fokus auf Abläufe, die im System stattfinden

- Geschäftsprozesse, Systemprozesse, Methoden einer Klasse
- Eignet sich für Analyse *und* Design
- Aktivität := Menge potentieller Abläufe
- Kontrollelemente: Verzweigung, Vereinigung, Fork, Join
- Elemente: Aktionen, Kanten, Objektknoten, [Pin-Notation ... ]
- Kontrolltoken, Datentoken
- uvm.

Inspiriert von Petri-Netzen

# Aktivitätsdiagramme

Fokus auf Abläufe, die im System stattfinden

- Geschäftsprozesse, Systemprozesse, Methoden einer Klasse
- Eignet sich für Analyse *und* Design
- Aktivität := Menge potentieller Abläufe
- Kontrollelemente: Verzweigung, Vereinigung, Fork, Join
- Elemente: Aktionen, Kanten, Objektknoten, [Pin-Notation ... ]
- Kontrolltoken, Datentoken
- uvm.

Inspiziert von Petri-Netzen

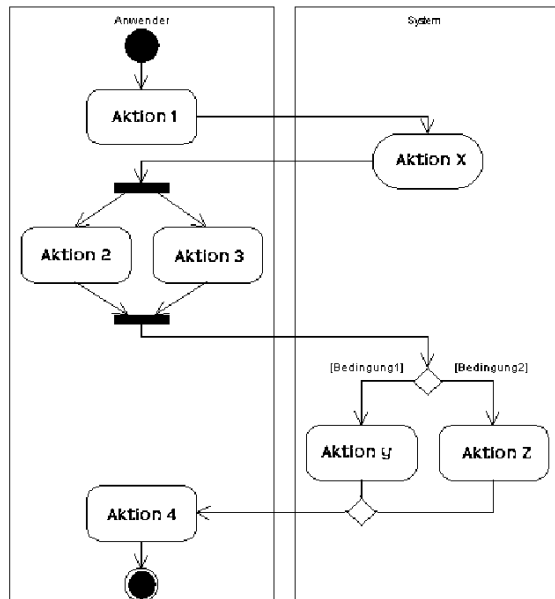
# Aktivitätsdiagramme

Fokus auf Abläufe, die im System stattfinden

- Geschäftsprozesse, Systemprozesse, Methoden einer Klasse
- Eignet sich für Analyse *und* Design
- Aktivität := Menge potentieller Abläufe
- Kontrollelemente: Verzweigung, Vereinigung, Fork, Join
- Elemente: Aktionen, Kanten, Objektknoten, [Pin-Notation ... ]
- Kontrolltoken, Datentoken
- uvm.

Inspiriert von Petri-Netzen

# Aktivitätsdiagramm: Kontrollfluss mit "Swim-Lanes"



# Aktivitätsdiagramme

Fokus auf Abläufe, die im System stattfinden

- Geschäftsprozesse, Systemprozesse, Methoden einer Klasse
- Eignet sich für Analyse *und* Design
- Aktivität := Menge potentieller Abläufe
- Kontrollelemente: Verzweigung, Vereinigung, Fork, Join
- Elemente: Aktionen, Kanten, Objektknoten, [Pin-Notation ... ]
- Kontrolltoken, Datentoken
- uvm.

Inspiziert von Petri-Netzen

# Aktivitätsdiagramme

Fokus auf Abläufe, die im System stattfinden

- Geschäftsprozesse, Systemprozesse, Methoden einer Klasse
- Eignet sich für Analyse *und* Design
- Aktivität := Menge potentieller Abläufe
- Kontrollelemente: Verzweigung, Vereinigung, Fork, Join
- Elemente: Aktionen, Kanten, Objektknoten, [Pin-Notation ... ]
- Kontrolltoken, Datentoken
- uvm.

Inspiziert von Petri-Netzen

# Aktivitätsdiagramme

Fokus auf Abläufe, die im System stattfinden

- Geschäftsprozesse, Systemprozesse, Methoden einer Klasse
- Eignet sich für Analyse *und* Design
- Aktivität := Menge potentieller Abläufe
- Kontrollelemente: Verzweigung, Vereinigung, Fork, Join
- Elemente: Aktionen, Kanten, Objektknoten, [Pin-Notation ... ]
- Kontrolltoken, Datentoken
- uvm.

Inspiziert von Petri-Netzen

# UML-Zustandsautomaten

- **Zustände: Name, Verhalten**
- Übergänge: Trigger [Guards] / Effect
- Entscheidungen
- Pseudozustände, Historien, Hierarchien, Regionen ...



# UML-Zustandsautomaten

- Zustände: Name, Verhalten
- Übergänge: Trigger [Guards] / Effect
- Entscheidungen
- Pseudozustände, Historien, Hierarchien, Regionen ...

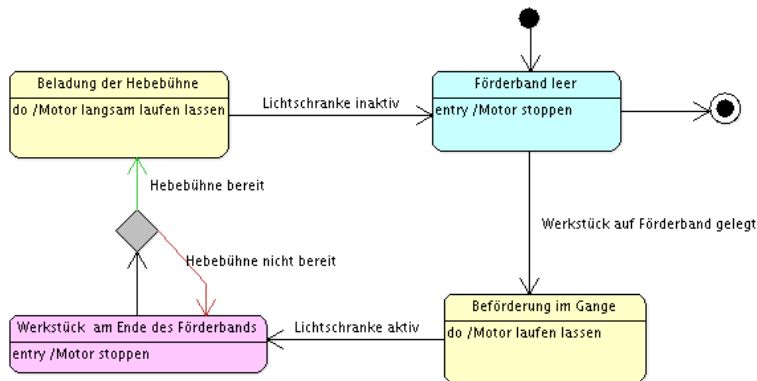
# UML-Zustandsautomaten

- Zustände: Name, Verhalten
- Übergänge: Trigger [Guards] / Effect
- Entscheidungen
- Pseudozustände, Historien, Hierarchien, Regionen ...

# UML-Zustandsautomaten

- Zustände: Name, Verhalten
- Übergänge: Trigger [Guards] / Effect
- Entscheidungen
- Pseudozustände, Historien, Hierarchien, Regionen ...

# Zustandsautomat



# Werkzeuge (Auswahl)

- Topcased
- Fujaba
- StarUML
- Rational Software Architect
- Borland Together für Eclipse
- Magic Draw
- MID Innovator