



TEST-UML



FernUniversität  
Gesamthochschule in Hagen



SOFTWARE  
ENGINEERING  
Praktische Informatik III

---

# Testen und UML


---

Dr. Mario Winter

FernUniversität Hagen

TEST-UML

Testen und UML



SOFTWARE  
ENGINEERING  
Praktische Informatik III

---

## Informatik an der FernUni

- Studienmöglichkeiten
  - Integrierter Diplomstudiengang Informatik
  - Modellstudiengang Bachelor of Science in Informatik
  - Zusatzstudiengang Praktische Informatik
  - Vorbereitende Studien für Erweiterungsprüfung zum Lehramt (Sekundarstufe I/II) im Fach Informatik
  - Informatik-Fächer im Diplomstudiengang Mathematik

Alle Kurse der Informatik zur Weiterbildung geeignet  
Im Rahmen des aktuellen Lehrangebots belegbar

Strukturierte Weiterbildung  
Weiterbildungspakete aus thematisch zusammenhängenden Kursen für bestimmte Adressatengruppen  
In zwei bis vier Semestern studierbar

---

TUML

Seite 2

Dr. Mario Winter, FernUniversität Hagen

TEST-UML
Testen und UML

## Überblick

- Warum projektbegleitend Testen?
- Was ist die UML?
- UML: Modell, Sichten, Diagramme
  - Funktionssicht
  - Struktursicht
  - Verhaltenssicht
- UML-Diagramme aus dem Blickwinkel des Testers ...
  - Anwendungsfalldiagramm und Aktivitätsdiagramm
  - Klassendiagramm und Objektdiagramm
  - Sequenzdiagramm und Kollaborationsdiagramm
  - Zustandsdiagramm
  - Komponentendiagramm und Verteilungsdiagramm
- Die Object Constraint Language OCL
- Zusammenfassung

TUML
Seite 3
Dr. Mario Winter, FernUniversität Hagen

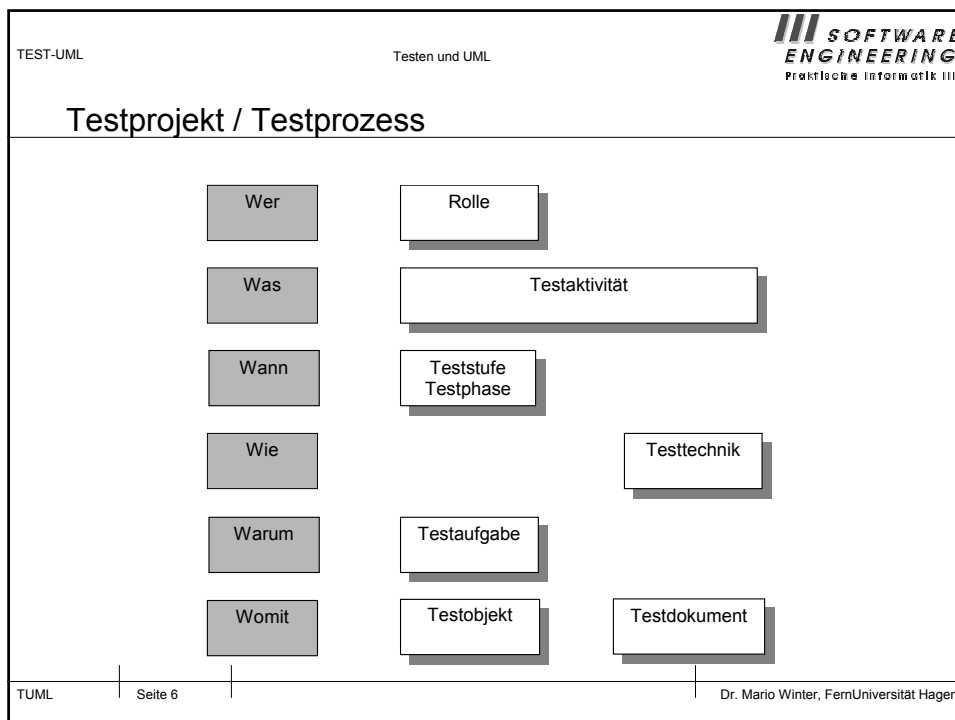
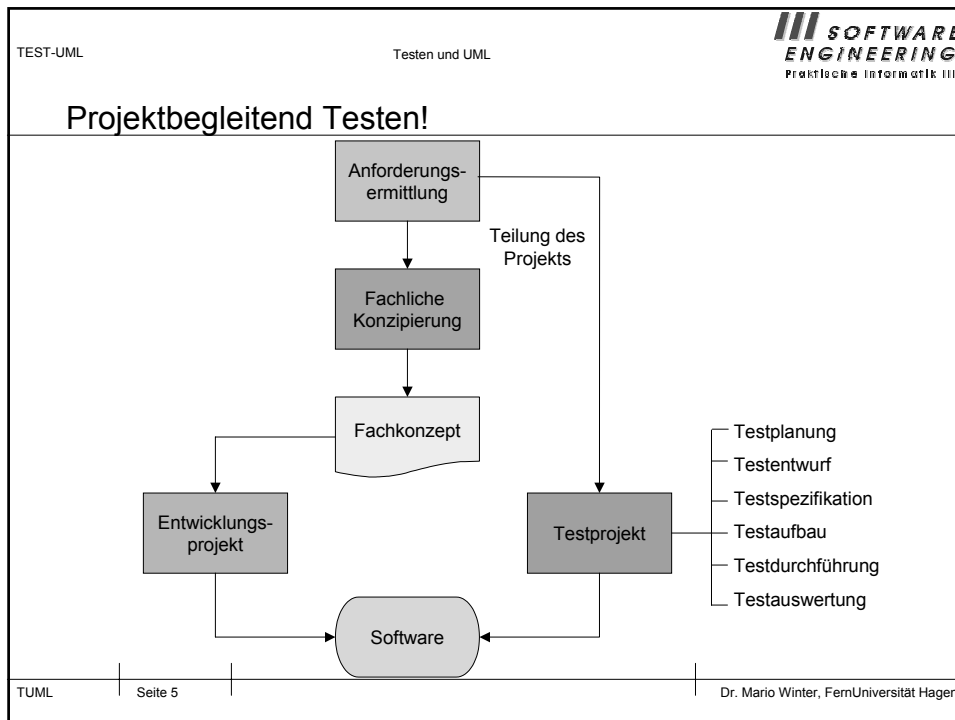
TEST-UML
Testen und UML


## Warum Testen?

The graph plots three metrics against project phases: Bedarfanalyse, Analyse/Design, Realisierung, Systemeinführung, and Betreuung. The Y-axis represents the 'Prozentualer Anteil' (Percentage) from 0 to 100. 'Testaufwand' (Test effort) is a dotted line that peaks at Realisierung. 'Wahrscheinlichkeit Fehlerentdeckung' (Probability of error detection) is a dashed line that peaks at Realisierung. 'Risiko' (Risk) is a solid line that starts high and decreases as the project progresses.

Projektphase (Zeit)	Testaufwand (%)	Wahrscheinlichkeit Fehlerentdeckung (%)	Risiko (%)
Bedarfanalyse	~5	~5	~90
Analyse/Design	~15	~15	~85
Realisierung	~90	~50	~75
Systemeinführung	~15	~25	~30
Betreuung	~10	~20	~15

TUML
Seite 4
Dr. Mario Winter, FernUniversität Hagen




TEST-UML
Testen und UML


## Teststufen und Testphasen

- Teststufen
- Entwicklertest des Entwurfs
- Entwicklertest der Klassen
- Integrationstest der Klassen
- Funktionstest der Komponenten
- Integrationstest der Komponenten
- Funktionstest des Systems
- Integrationstest des Systems
- Abnahmetest

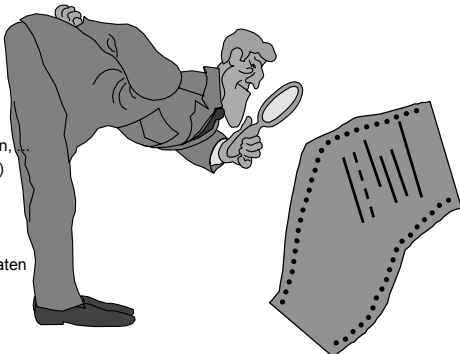
- Testphasen
- Testplanung
- Testentwurf
- Testinfrastrukturaufbau
- Testfallermittlung
- Testdatenermittlung
- Testprozedurerstellung
- Testausführung
- Testauswertung

TUML
Seite 7
Dr. Mario Winter, FernUniversität Hagen

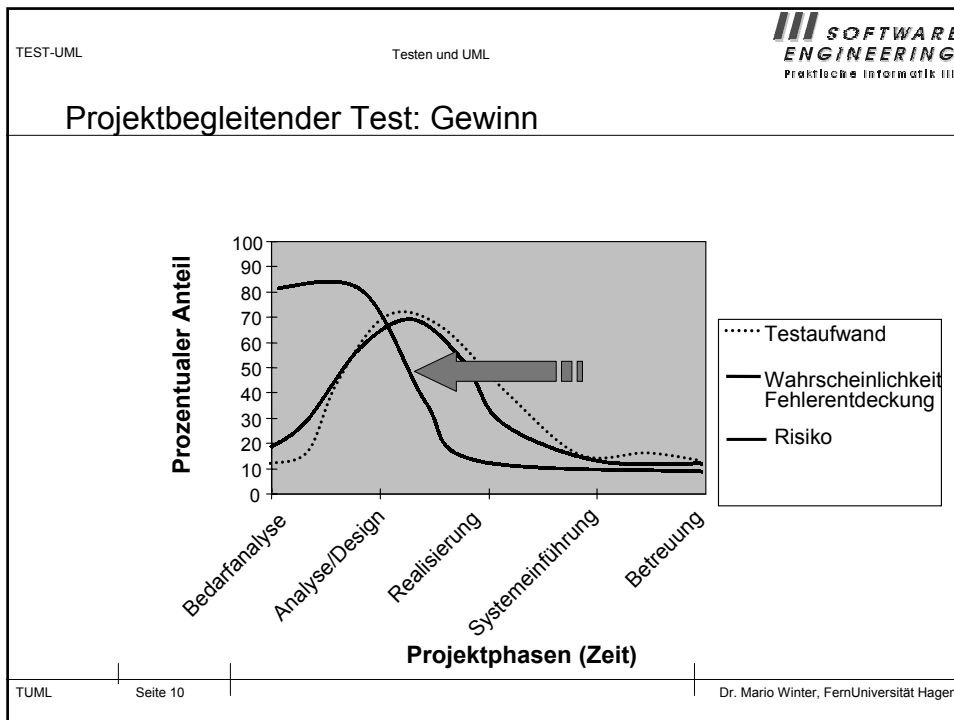
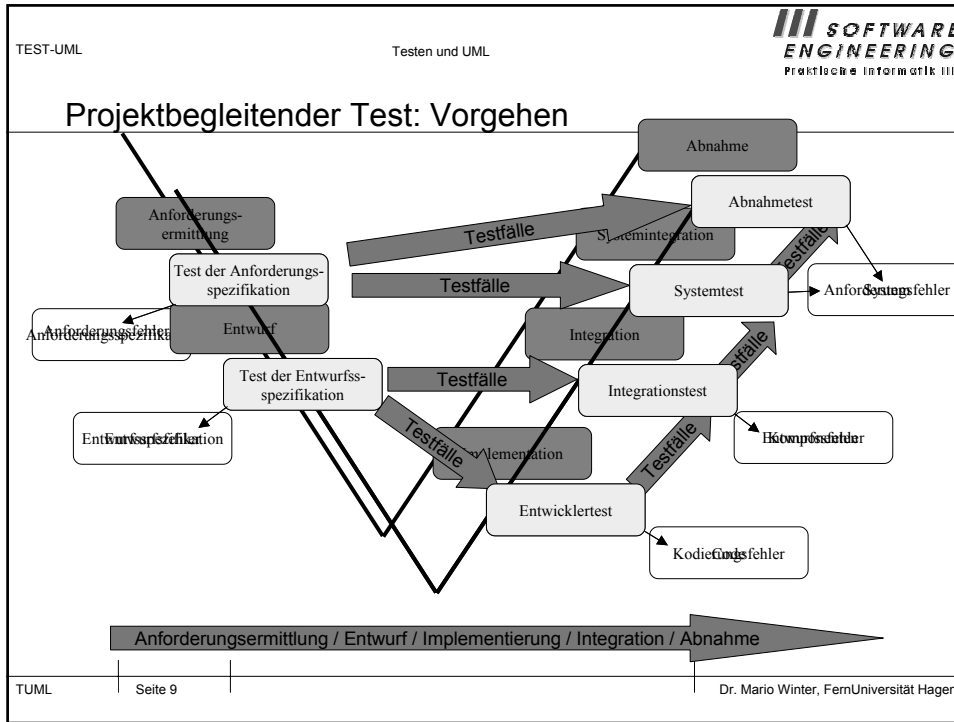
TEST-UML
Testen und UML


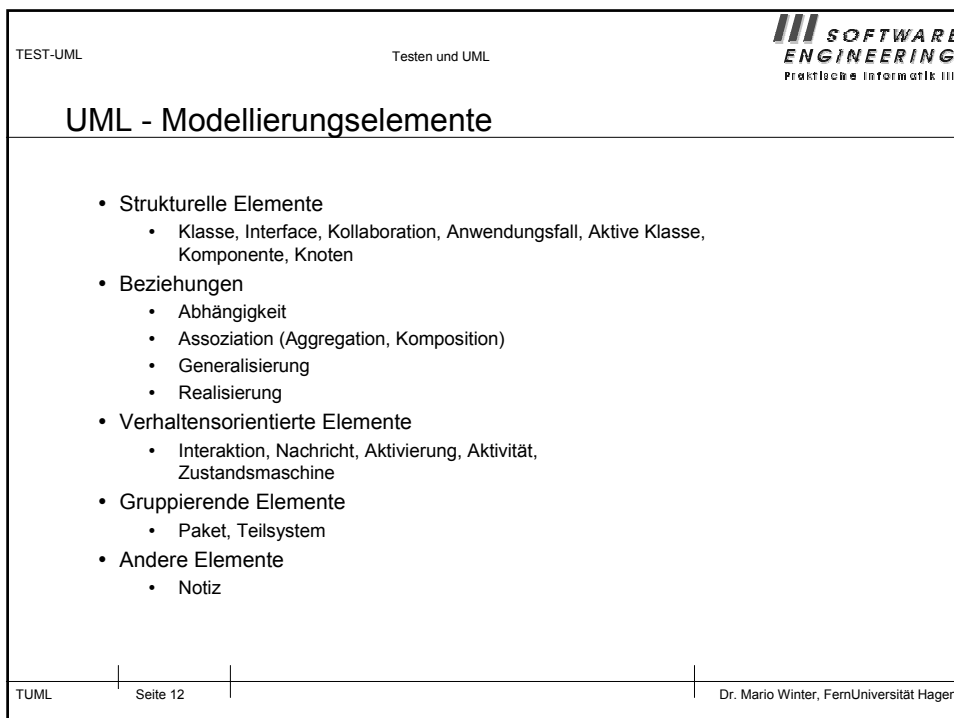
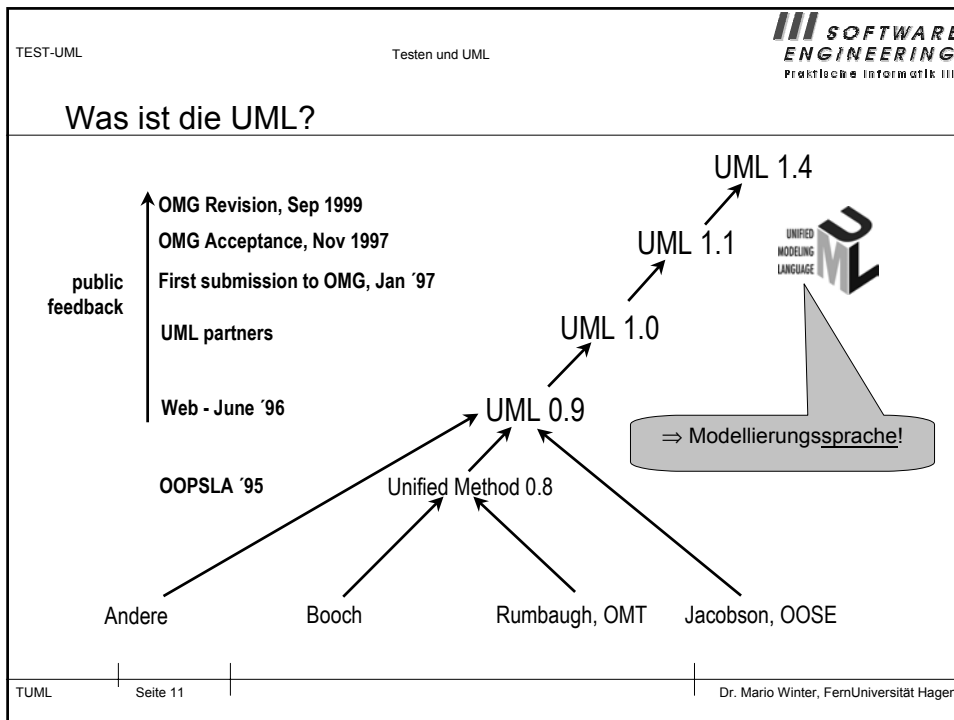
## Testtechniken

- Statisch
  - Reviews, Inspektionen
  - Walk Throughs
- Dynamisch
  - Funktional
    - Grenzwerte, Äquivalenzklassen, ...
    - Schnittstellenbasiert (Verträge)
    - Szenarienbasiert
  - Zustandsbasiert
    - Über Instanzvariablen
    - Über explizite Zustandsautomaten
  - Strukturell
    - Kontrollfluss
    - Daten/Objektfluss
    - Botschaftsfluss



TUML
Seite 8
Dr. Mario Winter, FernUniversität Hagen





TEST-UML Testen und UML **SOFTWARE ENGINEERING**  
Praktische Informatik III

## UML - Erweiterungsmechanismen

- Stereotype
- Tagged values
- Constraints

The diagram shows a class `ActionQueue` with the following elements:

- Stereotype:** `«container»` (indicated by a bracket on the left).
- Tagged Value:** `{version = 3.2}` (indicated by a dot on the right).
- Operations:** `add(a : Action)`, `remove(n : Integer)`, `length() : Integer`, and `reorder()`.
- Constraint:** `{add runs in O(1) time}` (indicated by a dot on the right).
- Other Stereotypes:** `«query»` and `«helper functions»` are also shown.

TUML Seite 13 Dr. Mario Winter, FernUniversität Hagen

TEST-UML Testen und UML **SOFTWARE ENGINEERING**  
Praktische Informatik III

## UML: Modell, Sichten und Diagramme

The diagram illustrates the relationship between a central **Modell** (represented by a cylinder) and various UML diagrams, categorized into four views:

- Funktionssicht (Functional View):**
  - Aktivitäts Diagramm (Activity Diagram)
  - Anwendungsfall Diagramm (Use Case Diagram)
  - Klassen Diagramm (Class Diagram)
- Log. Struktursicht (Logical Structural View):**
  - Objekt Diagramm (Object Diagram)
  - Komponenten Diagramm (Component Diagram)
- Verhaltenssicht (Behavioral View):**
  - Sequenz Diagramm (Sequence Diagram)
  - Kollaborations Diagramm (Collaboration Diagram)
  - Zustands Diagramm (State Diagram)
- Phys. Struktursicht (Physical Structural View):**
  - Deployment Diagramm (Deployment Diagram)

TUML Seite 14 Dr. Mario Winter, FernUniversität Hagen

TEST-UML Testen und UML

**SOFTWARE ENGINEERING**  
Praktische Informatik III

## Visuelle Modellierung mit UML Diagrammen

**Anwendungsfall-  
diagramm**

**Aktivitäts-  
diagramm**

**Klassendiagramm**

**Komponenten-  
diagramm**

**Sequenz-  
diagramm**

**Zustands-  
diagramm**

**Deployment-  
diagramm**

**Kollaborations-  
diagramm**

TUML Dr. Mario Winter, FernUniversität Hagen

TEST-UML Testen und UML

**SOFTWARE ENGINEERING**  
Praktische Informatik III

## Anwendungsfalldiagramm

- Von außen beobachtbare Systemfunktionalität

TUML Seite 16 Dr. Mario Winter, FernUniversität Hagen



TEST-UML
Testen und UML

## Anwendungsfalldiagramm

- Von außen beobachtbare Systemfunktionalität
- Anforderungsermittlung
- Zweck
  - Spezifiziert den Kontext eines Systems
  - Erfasst die funktionalen Anforderungen an das System
  - Validierung der Systemarchitektur
  - Treibt die Implementation und dient zur Testfallgenerierung
- Erstellt von Analytikern und Domänenexperten
- Testaspekte
  - Grundlage des System- und Abnahmetests
  - Sollte nicht-funktionale Anforderungen für Performanz- und Lasttests umfassen
  - Strukturierung der "Testware"

TUML
Seite 17
Dr. Mario Winter, FernUniversität Hagen

TEST-UML
Testen und UML

## Aktivitätsdiagramm

- Dynamisches Verhalten (Aktionsorientiert)

TUML
Seite 18
Dr. Mario Winter, FernUniversität Hagen

TEST-UML
Testen und UML

## Aktivitätsdiagramm

- Enthält das dynamische Verhalten (Aktionsorientiert)
- Zweck
  - Modellierung von Geschäftsprozessen (business workflows)
  - Modellierung des Ablaufs von Anwendungsfällen
  - Modellierung des Ablaufs von Operationen
- Erstellt von Analytikern und Entwicklern
- Testaspekte
  - System- und Abnahmetest
  - Grundlage für konkrete Testszenarien

TUML
Seite 19
Dr. Mario Winter, FernUniversität Hagen

TEST-UML
Testen und UML

## Ermittlung von Testfällen aus Aktivitätsdiagrammen

- Testszenarien so auswählen, dass die geforderte Überdeckung des Aktivitätsdiagramms erzielt wird
- Anzahl der zu testenden Szenarien bezüglich der Überdeckung aller Knoten (Aktionen) und Kanten (Übergänge) im Aktivitätsdiagramm minimieren
- Überdeckungsmetriken zur Generierung weiterer Testszenarien verwenden.

TUML
Seite 20
Dr. Mario Winter, FernUniversität Hagen

TEST-UML Testen und UML

**SOFTWARE ENGINEERING**  
Praktische Informatik III

## Klassendiagramm

- Vokabular des Systems
  - Domänendinge
  - Sachverhalte
  - Beziehungen

**Class**

- name
- Shape
- attributes
- + origin : Point
- + move(p : Point)
- + resize(s : Scale)
- + display()
- # invalidateRegion()
- signature
- operations
- Responsibilities
- manage shape
- state
- handle basic shape transformations
- extra compartment
- visibility

**Interface**

- objects are underlined abstract elements are in italics
- name
- IApplication

**Class**

- name
- Department
- name : Name
- multiplicity
- Location
- constraint
- role
- member
- 1..\*
- 1
- manager
- Person
- name : Name
- employeeID : Integer
- title : String
- attributes
- operations
- getPhoto(p: Photo)
- getSoundBite()
- getContactInformation()
- getPersonalRecords()
- dependency
- PersonnelRecord
- taxID
- employmentHistory
- salary
- interface
- ISecureInformator
- Office
- name
- address : String
- voice : Number
- generalization
- Headquarters
- Company
- aggregation
- 1

TUML Seite 21 Dr. Mario Winter, FernUniversität Hagen

TEST-UML Testen und UML

**SOFTWARE ENGINEERING**  
Praktische Informatik III

## Klassendiagramm

- Enthält das Vokabular des Systems
- Aufgebaut und verfeinert während der Entwicklung
- Zweck
  - Bezeichnet und modelliert wichtige Domänendinge
  - Spezifiziert Zusammenhänge
  - Spezifiziert das logische Datenbankschema
- Erstellt von Analytikern, Architekten und Entwicklern
- Testaspekte
  - Entwicklertest und Integrationstest
  - Strukturelle Tests bzgl. erlaubter Objektkonstellationen
  - Ermittlung einer Integrations-Strategie

TUML Seite 22 Dr. Mario Winter, FernUniversität Hagen

TEST-UML
Testen und UML

## Ermittlung von Testfällen aus Assoziationen

- Testfallmatrix, in der für eine Assoziation/Aggregation/Komposition die zu testenden Anzahlen verbundenen Instanzen angegeben werden

	Konto	Min. - 1=0	Min.=Max.= typisch = 1	Max. + 1 = 2
Buchung				
Min (0)		N/A	OK	N/A
Typisch (z.B. 2000)		Fail	OK	Fail
Max. (z.B. 2^16)		-	OK	-
Max +1		-	Fail	-

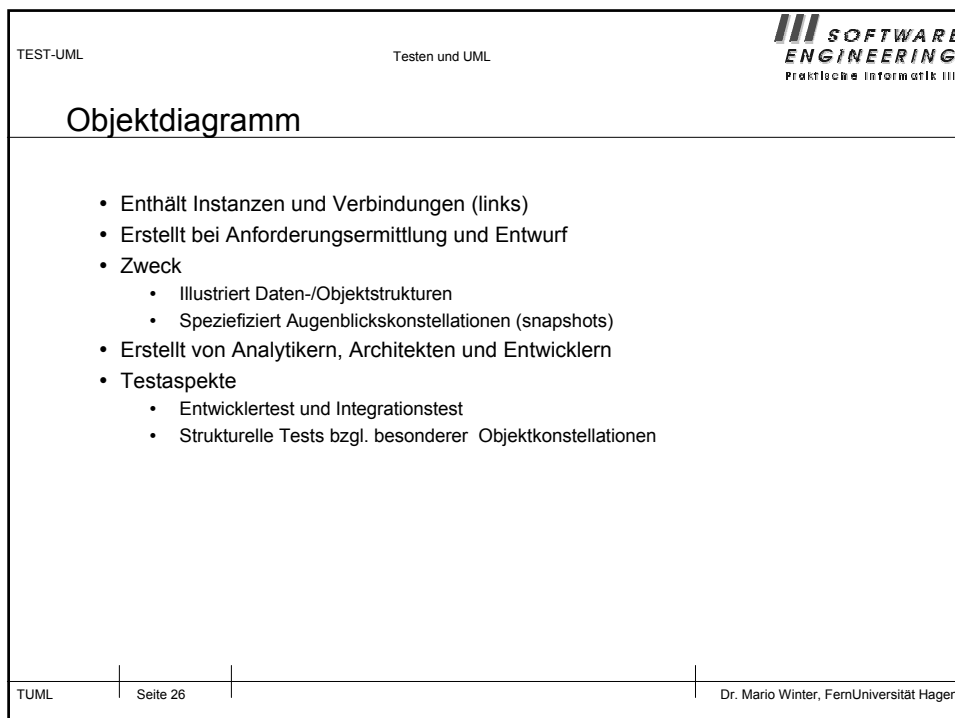
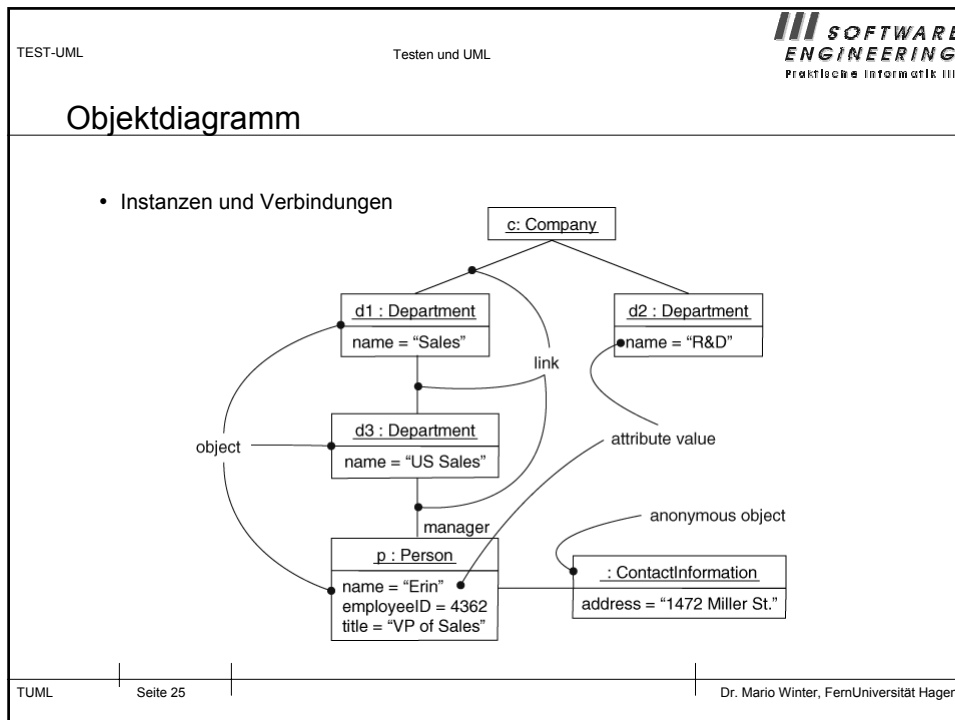
TUML
Seite 23
Dr. Mario Winter, FernUniversität Hagen

TEST-UML
Testen und UML


## Ermittlung einer Integrationsstrategie

1. Fokussierung auf die zu integrierenden Komponenten
  - Klassendiagramm, Interaktions- sowie Komponentendiagramme betrachten
  - Menge zu integrierender Komponenten eingrenzen
  - Nutzungs- und ggf. Generalisierungsbeziehungen ermitteln
  - Ergebnis: „Komponentengraph“
2. Benutzungshierarchie bottom-up, Generalisierungshierarchie top-down

TUML
Seite 24
Dr. Mario Winter, FernUniversität Hagen



TEST-UML
Testen und UML



Praktische Informatik III

## Interaktionsdiagramme

**Sequenzdiagramm**


- Ablauforientierte Modellierung dynamischen Verhaltens
- Zweck
  - Modellierung von Operationsabläufen (Kontrollfluss)
  - Illustration typischer Szenarien

**Kollaborationsdiagramm**

- Nachrichtenorientierte Modellierung dynamischen Verhaltens
- Zweck
  - Modellierung von Operationsabläufen (Kontrollfluss)
  - Illustration der Koordinierung in Objektstrukturen
- Erstellt von Analytikern, Entwicklern und Testern
- Testaspekte
  - Entwicklertest und Integrationstest
  - Ablauforientierter Test

TUML
Seite 27
Dr. Mario Winter, FernUniversität Hagen

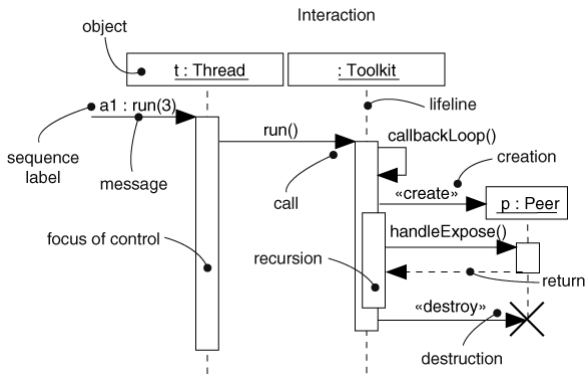
TEST-UML
Testen und UML



Praktische Informatik III

## Sequenzdiagramm

- Dynamisches Verhalten (Ablauforientiert)



The diagram illustrates the following sequence of events:

- Object:** t: Thread
- Message:** a1: run(3) (sequence label) is sent to t: Thread.
- Call:** t: Thread calls run() on : Toolkit.
- Creation:** : Toolkit creates p: Peer (labeled «create»).
- Message:** : Toolkit calls callbackLoop() on t: Thread.
- Message:** : Toolkit calls handleExpose() on p: Peer.
- Return:** p: Peer returns to : Toolkit.
- Recursion:** : Toolkit calls handleExpose() on itself (recursion).
- Message:** : Toolkit calls handleExpose() on t: Thread.
- Message:** t: Thread returns to : Toolkit.
- Destruction:** : Toolkit destroys p: Peer (labeled «destroy»).

TUML
Seite 28
Dr. Mario Winter, FernUniversität Hagen

TEST-UML Testen und UML **SOFTWARE ENGINEERING** Praktische Informatik III

## Kollaborationsdiagramm

- Dynamisches Verhalten (Nachrichtenorientiert)

```

sequenceDiagram
    participant Client as c: Client
    participant Transaction as Transaction
    participant ODBDProxy as p: ODBDProxy

    Client->>Transaction: link
    activate Transaction
    Transaction->>Transaction: «local» object
    deactivate Transaction
    Client->>Transaction: 1: «create»
    Client->>Transaction: 2: setActions(a, d, o)
    Client->>Transaction: 3: «destroy»
    deactivate Transaction
    Transaction->>ODBDProxy: message «global»
    activate ODBDProxy
    ODBDProxy->>ODBDProxy: 2.1: setValues(d, 3.4)
    ODBDProxy->>ODBDProxy: 2.2: setValues(a, "CO")
    deactivate ODBDProxy
    
```

collaboration diagram

TUML Seite 29 Dr. Mario Winter, FernUniversität Hagen

TEST-UML Testen und UML **SOFTWARE ENGINEERING** Praktische Informatik III

## Ermittlung von Testfällen aus Sequenzdiagrammen

1. Fokussierung auf zusammengehörnde Interaktionsdiagramme
2. Ermittlung von Testfällen aus Kontrollflussinformation
3. Testdatenermittlung aus Bedingungen

```

sequenceDiagram
    participant Bestellung as :Bestellung
    participant Posten as :Posten
    participant Produkt as :Produkt
    participant Nachbestell as :Nachbestellposten

    Bestellung->>Bestellung: erzeugen()
    activate Bestellung
    Bestellung->>Posten: erzeugen()
    activate Posten
    Posten->>Posten: preis()
    Posten->>Produkt: gibVerkaufspreis()
    activate Produkt
    Produkt->>Produkt: prüfeLagermenge()
    deactivate Produkt
    Posten->>Bestellung: MengeOK = prüfeLagermenge()
    deactivate Posten
    Bestellung->>Bestellung: [NOT MengeOK] bestelle()
    activate Bestellung
    Bestellung->>Nachbestell: erzeugen()
    activate Nachbestell
    deactivate Nachbestell
    deactivate Bestellung
    
```

TUML Seite 30 Dr. Mario Winter, FernUniversität Hagen

TEST-UML Testen und UML **SOFTWARE ENGINEERING**  
Praktische Informatik III

## Zustandsdiagramm (Statechart)

- Dynamisches Verhalten (Ereignisorientiert)

The diagram illustrates a State Machine with the following components:

- States:** Idle (initial state), Working (nested state), Connecting, and Connected (final state).
- Transitions:**
  - Idle to Idle: `keepAlive / check()` (internal transition)
  - Idle to Working: `onHook` (transition)
  - Working to Connecting: `ready(3) [signalOK]` (transition with guard)
  - Connecting to Connected: (unlabeled transition)
  - Connected to Idle: `offHook / reclaimConnection()` (transition)
- Other Elements:**
  - `off`: Transition from Idle to a final state.
  - `event`: Trigger for the `offHook / reclaimConnection()` transition.
  - `action`: Associated with the `offHook / reclaimConnection()` transition.

TUML Seite 31 Dr. Mario Winter, FernUniversität Hagen

TEST-UML Testen und UML **SOFTWARE ENGINEERING**  
Praktische Informatik III

## Zustandsdiagramm (Statechart)

- Enthält das dynamische Verhalten (Ereignisorientiert)
- Zweck
  - Modellierung des Objektverhaltens (object lifecycle)
  - Modellierung reaktiver Objekte (Benutzungsschnittstellen, Geräte, etc.)
- Erstellt von Analytikern, Entwicklern und Testern
- Testaspekte
  - Entwicklertest und Funktionstest (Systemtest)
  - Test bzgl. des konformen Verhaltens
  - Robustheitstest

TUML Seite 32 Dr. Mario Winter, FernUniversität Hagen



TEST-UML Testen und UML **SOFTWARE ENGINEERING** Praktische Informatik III

## Ermittlung von Testfällen aus Zustandsdiagrammen

1. Fokussierung auf das Zustandsdiagramm
2. Ableiten des Übergangsbaumes für den Zustands-Konformanztest
3. Erweitern des Übergangsbaumes für den Zustands-Robustheitstest
4. Generieren der Botschaftssequenzen und Ergänzen der Botschaftsparameter

TUML Seite 33 Dr. Mario Winter, FernUniversität Hagen

TEST-UML Testen und UML **SOFTWARE ENGINEERING** Praktische Informatik III

## Komponentendiagramm

- Physikalische Struktur der Implementation

TUML Seite 34 Dr. Mario Winter, FernUniversität Hagen

TEST-UML
Testen und UML

## Komponentendiagramm

- Enthält die physikalische Struktur der Implementation
- Teil der Architekturspezifikation
- Zweck
  - Organisation des Quellcodes
  - Erzeugung eines ausführbaren Releases (build)
  - Spezifikation der physikalischen Datenbank
- Erstellt von Entwicklern
- Testaspekte
  - Installationstest, Systemtest, Abnahmetest
  - Organisation der Testware
  - Ermittlung einer Integrationsstrategie


TUML
Seite 35
Dr. Mario Winter, FernUniversität Hagen


TEST-UML
Testen und UML

## Deploymentdiagramm

- Hardwaretopologie

TUML
Seite 36
Dr. Mario Winter, FernUniversität Hagen

TEST-UML	Testen und UML	
<h2>Deploymentdiagramm</h2>		
<ul style="list-style-type: none"><li>• Enthält die Topologie der Hardware</li><li>• Teil der Architekturspezifikation</li><li>• Zweck<ul style="list-style-type: none"><li>• Spezifiziert die Verteilung der Komponenten</li><li>• Identifikation von Flaschenhälsen (performance bottlenecks)</li></ul></li><li>• Estellt von Architekten, Netzwerkspezialisten, and Systemingenieuren</li><li>• Testaspekte<ul style="list-style-type: none"><li>• Installationstest, Systemtest</li><li>• Sicherheits-, Performanz- und Lasttest</li></ul></li></ul>		
TUML	Seite 37	Dr. Mario Winter, FernUniversität Hagen

TEST-UML	Testen und UML	
<h2>Object Constraint Language OCL</h2>		
<ul style="list-style-type: none"><li>• Deklarative Spezifikationsprache</li><li>• Modellbasierte Spezifikationen<ul style="list-style-type: none"><li>• Atomare Datentypen</li><li>• Logische Ausdrücke</li><li>• Mengentheoretische Operationen</li></ul></li><li>• Zweck<ul style="list-style-type: none"><li>• Zusicherungen für Modellelemente</li><li>• Klassen- und Operationsspezifikationen</li></ul></li><li>• Estellt von Analytikern und Entwicklern</li><li>• Testaspekte<ul style="list-style-type: none"><li>• Präzise Spezifikation von Testfällen und Testdaten</li><li>• Funktionale Tests</li></ul></li></ul>		
TUML	Seite 38	Dr. Mario Winter, FernUniversität Hagen

TEST-UML
Testen und UML

## Testfallermittlung aus OCL-Operationsspezifikationen

Context ExtendedStack	A	B	C	D	E	F
invariant@ self.size() >= 0 AND	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
invariant@ self.size() <= self.MAXSIZE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
ExtendedStack() pre@ true	TRUE					
ExtendedStack() post@ self.size() = 0 AND	TRUE					
ExtendedStack() post@ self.MAXSIZE = maxSize	TRUE					
push() pre@ self.size() < self.MAXSIZE	TRUE	TRUE				
push() post@ self.size()=self.size()@pre+1	-	TRUE				
top() pre@ self.size() > 0	FALSE	TRUE	TRUE			
top() post@ return != null	-	-	TRUE			
pop() pre@ self.size() > 0	FALSE	TRUE	TRUE	TRUE		
pop() @post self.size() = self.size()@pre - 1	-	-	-	TRUE		
all() pre@ true	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
all() post@ self.size() > 0 AND	-	-	-	-	FALSE	TRUE
all() post@ return.size() = self.size() OR	-	-	-	-	dc	TRUE
all() post@ not self.size() > 0 AND	-	-	-	-	TRUE	FALSE
all() post@ return = null	-	-	-	-	TRUE	dc


TUML
Seite 39
Dr. Mario Winter, FernUniversität Hagen

TEST-UML
Testen und UML

## Testen und UML

- "Lingua Franca" für Entwickler und Tester
- Hilfe in allen Teststufen
  - ++ Entwicklertest
  - + Funktionstest
  - + Integrationstest
  - + System- und Abnahmetest
  - - Sicherheits-, Performanz und Lasttest
- Unterstützt herkömmliche Testtechniken
  - Funktionstest
  - Ablauftest
  - Strukturtest
- OCL ermöglicht präzise Spezifikationen

TUML
Seite 40
Dr. Mario Winter, FernUniversität Hagen

TEST-UML	Testen und UML	 <b>SOFTWARE ENGINEERING</b> Praktische Informatik III
<h2>Literatur und weitere Information</h2>		
<ul style="list-style-type: none"><li>• Bücher zum objektorientierten Test<ul style="list-style-type: none"><li>• Robert Binder: Testing Object-Oriented Systems. Addison Wesley, Upper Saddle River, 1999</li><li>• John McGregor, David Sykes: A Practical Guide to Testing Object-Oriented Software. Addison Wesley, Upper Saddle River, 2001</li><li>• Noack, J.: Techniken der objektorientierten Softwareentwicklung. Springer, Berlin, 2001</li><li>• Harry Sneed und Mario Winter: Testen objektorientierter Software – Das Praxishandbuch. Hanser, München, 2001</li></ul></li><li>• Seminare<ul style="list-style-type: none"><li>• Plenum Institut: <a href="http://pin.plenum.de/pIN_Floater.htm">http://pin.plenum.de/pIN_Floater.htm</a></li><li>• GI-DIA: <a href="http://www.dia-bonn.de/">http://www.dia-bonn.de/</a></li></ul></li><li>• Organisationen<ul style="list-style-type: none"><li>• GI-FG 2.1.7 TAV AK Test objektorientierter Programme (TOOP) <a href="http://www.informatik.fernuni-hagen.de/import/pi3/GI">http://www.informatik.fernuni-hagen.de/import/pi3/GI</a><ul style="list-style-type: none"><li>• 320+ Annotierte Referenzen zu OOT</li><li>• Fragebogen OO-Testwerkzeuge (OO-CAST)</li><li>• OO-Testbarkeit</li><li>• OO-Reviews</li></ul></li></ul></li></ul>		
TUML	Seite 41	Dr. Mario Winter, FernUniversität Hagen