## Diagram types and application areas

<table>
<thead>
<tr>
<th>Diagram type</th>
<th>Diagram</th>
<th>Phase</th>
<th>Application area</th>
</tr>
</thead>
</table>
| **Use cases**        | Use case diagrams            | • Requirements  
                          |                                       | Business processes, general applications             |
| **Static Models**    | **Class diagrams**           | • Definition  
                          |                                       | Everywhere, the class diagram is the most Important diagram in UML |
|                      | **Instance diagrams**        | • Creation                       |                                                       |
| **Dynamic Models**   | Activity diagrams  
                          | • Requirements  
                          | Shows the information flow and the time flow of the cooperation of the objects | Timeflow structure with few messages |
|                      | Interaction diagrams        | • Definition  
                          |                                                       | Timeflow structure with few classes                  |
|                      | Collaboration diagrams      | • Creation  
                          |                                                       |
|                      | Sequence-diagrams           | • Delivery                        |                                                       |
|                      | State diagrams               | • Requirements  
                          | Representation of the dynamic behavior               |
|                      |                              | • Definition  
                          |                                                       |
|                      |                              | • Creation                        |                                                       |
|                      |                              | • Delivery                         |                                                       |
Class and instance diagrams

- Class and instance diagrams are **static** diagrams.
- **Representation method for the graphic notation and semantics for the modeling of objects, classes, and their relations**
- **Instance diagrams** are used for describing **not only the test cases, but also the scenarios or examples**
- UML does not describe how classes shall be determined.
- No visual differentiation between objects and classes - objects are distinguished from classes only through **underscoring** in their notation.
According to UML notation the class is represented in a rectangle and is comprised of:

- **Class name:** Naming of a class from the language area of application field. As the rule, a main word or an adjective (e.g., „vehicle“ or „civil vehicle“)

- **Attribute:** A data value (e.g., position, size, color,...). The objects of a class have the data value of this class

- **Operation:** A function (e.g., delay, delete, change size,..). Application to objects of a class
Object attribute

- An **object attribute of a class** is the description of a **data element** that is available in each object of the class. The name of the object attribute is given in the description of the class.

- An object carries an individual and unchangeable attribute value for each object attribute.

**Notation:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Object : Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute_1: data type</td>
<td>Attribute_1 = Attributevalue_1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Attribute_n: data type</td>
<td>Attribute_n = Attributevalue_n</td>
</tr>
</tbody>
</table>

**Example:**

<table>
<thead>
<tr>
<th>Team meeting</th>
</tr>
</thead>
<tbody>
<tr>
<td>titel: String</td>
</tr>
<tr>
<td>beginning: Date</td>
</tr>
<tr>
<td>duration: int</td>
</tr>
</tbody>
</table>
3.2. object oriented modelling with UML
3.2.3. Static Models

Object diagram with examples for object attributes

**Team meeting**
- title: String
- beginning: Date
- duration: int

**ar12: team meeting**
- title = "12. Department discussion"
- beginning = 5-10-98 10:00
- duration = 120 min

**ar13: team meeting**
- title = "13. Department discussion"
- beginning = 5-17-98 10:00
- duration = 120 min

**ar14: team meeting**
- title = "exhibition preparation"
- beginning = 5-12-98 9:00
- duration = 180 min

\[ O \quad \text{means "} O \text{ is an instance of } C \text{"} \]

**O C**
A **class attribute** is the description of a **data element that is only available once for all instances of a class**.

According to UML notation the **class attributes** are **underlined**.

**Notation:**

<table>
<thead>
<tr>
<th>Team meeting</th>
</tr>
</thead>
<tbody>
<tr>
<td>title: String</td>
</tr>
<tr>
<td>beginning: Date</td>
</tr>
<tr>
<td>duration: Int</td>
</tr>
<tr>
<td>quantity: Int</td>
</tr>
</tbody>
</table>
### Team meeting

- **theme:** String
- **beginning:** Date
- **duration:** int
- **quantity:** 3

### ar12: Team meeting

- **title:** "12. department discussion"
- **beginning:** 5-10-98 10:00
- **duration:** 120 min

### ar13: team meeting

- **title:** "13. department discussion"
- **beginning:** 5-17-98 10:00
- **duration:** 120 min

### ar14: team meeting

- **title:** "exhibition preparation"
- **beginning:** 5-1-98 9:00
- **duration:** 180 min
An operation of a class is a description of a task which can be performed by every instance of the class. In the description of the class, the name of the operation is noted. The notation of a parameter list is possible but not necessary. Common operations are also called instance operations.

**Notation:**

- **Class**
  - Attribute_1: data type
  - ...  
  - Attribute_n: data type

- Operation_1: return data type
  - ... 
  - Operation_m: return data type

**Example:**

<table>
<thead>
<tr>
<th>Team meeting</th>
</tr>
</thead>
<tbody>
<tr>
<td>title: String</td>
</tr>
<tr>
<td>beginning: Date</td>
</tr>
<tr>
<td>duration: int</td>
</tr>
</tbody>
</table>

| room_scheduling(): void |
| invite(): boolean |
| cancel(): void |
• A **class operation** of a class is a description of a task which can be performed only with knowledge of all current instances of a class.

**Notation:**

```
Class

Attribute_1: data type
...  
Attribute_n: data type

Operation_1: return data type
...  
Operation_m: return data type
```

**example:**

```
Meeting room

room_number: int
capacity: int

reservation(): boolean
approval(): boolean
Searchforfreeroom(): void
```
Associations

• A (binary) **association** between two classes K1 and K2 is a **description of a fact that the instances of both classes are having a functionally important relationship to each other**

• For **every object of the class K1**, there is an **individual, changeable and final amount of objects from the class K2, with which an association is made**. The same is true for objects from K2

**Notation:**

```
Class_1 --- Association --- Class_2
```

**example:**

```
Team meeting --- Guidance --- Team member
```
The **multiplicity** is noted usually on both ends of the connection line. This tells how many objects can stand this relationship. Example: '1', '0..1', '0..*', '5..8', '5,8'

There are differences between cardinality (# of elements) and multiplicity (area of allowed cardinality)

**Notation:**

```
Class_1  Association  Class_2
Mult
```

Multiplicity \( Mult : \)

- \( n \) (exactly \( n \) object of the \( Class_2 \))
- \( n..m \) (\( n \) till \( m \) objects of the \( Class_2 \))

Also valid for \( n \) and \( m \) are 0 and * (arbitrary 0)

**example:**

```
Team meeting  guidance  Team member
*  1
```

```
*  participation  2..*
```
• A special case of an association is the aggregation. Aggregation means an asymmetrical relationship between non-equal partners, e.g., not a peer-to-peer, but a master-slave-relationship

• In aggregation the „whole“ can exist without the „parts“

• In composition the „whole“ exists only when the single „parts“ exist. (ex: computer components of a computer)

Notation:
Example:

Aggregation

Composition
An inheritance relationship from class K1 to class K2 is a description of the fact that all objects O of the class K2, in addition to the described characteristics in class K2, have all characteristics of the class K1.

'characteristics' are here:
- The list of attributes and participation in associations,
- The list of operations and
- Possibly the associated type and parameter information

Notation:

Class_1

Colloquial:
Each Class_2 is one Class_1.
Class_2 is special from Class_1.
Inheritance (2)

- Helps to model the similarities and differences between classes
- **Upper class** (basis class) contains the shared attributes and operations
- Each **subclass** (derived) adds additional individual attributes and operations.
- A subclass inherits all attributes, operations, and relationships of the upper class
- A subclass can re-write and re-define the implementation of an operation.
- Each instance of a subclass is at the same time an instance of the upper class
Inheritance (3)

Example:

**Date**
- title: String
- beginning: Date
- duration [0..1]: int
- delayed(): void

**Personal date**
- place: String
- time_to_get_there: int
- authorization(): boolean

**Team meeting**
- themes [0..*]: String
- room_confirmation(): boolean
- invite(): void
- cancel(): void
• The same message leads to activation of different semantically similar methods depending on the receiver (Dynamic Bond).

```
Date
  ...
  delayed(): void
  (new: Date): Boolean

Personal date
delayed(): void

persDate: Personal date

delayed (new=7-20-98 17:00)

Team meeting
delayed(): void

teamMet: Team meeting
```
Polymorphism: Overwriting of Operations

What is defined in K can also represent all objects in classes K, K1 and K2.

What is defined in K is valid only for objects in class K and K2.

K1 receives a different meaning (override).

What is defined in K is valid only for objects in class K (and is therefore redundant when K is abstract).
Example of a class diagram: job processing (1)

- Relationship between order and customer
  - An order comes from a single customer
  - A customer can give different orders over time

- Relationship between order and order position
  - Each order has different order positions that refer to a single product

- Relationship between customer and corporate or private customer
  - The class „customer“ has 2 subclasses „corporate customer“ and „private customer“ --> in general

- Relationship between company, customer and employee
  - Multiplicity 0..1 means either there is one or there isn‘t
Example of a class diagram: Job processing(2)

3.2. object oriented modelling with UML

3.2.3. Static Models

- **Class**:
  - **Employee**
    - **Attributes**: name, address, credit-worthiness()
    - **Operations**: credit card nr
  - **Order**
    - **Attributes**: amount : Integer, price : money, isdelivered : Boolean
    - **Operations**: execute(), terminate()
  - **Corporate Customer**
    - **Attributes**: company name, credit-worthiness, Credit line, remind(): void
  - **Private Customer**
    - **Attributes**: credit card nr
  - **Product**
    - **Attributes**: amount : String, price : money, isdelivered : Boolean
    - **Operations**: execute(), terminate()

- **Associations**:
  - **Order-position**
    - **Attributes**: amount : Integer, price : money, isdelivered : Boolean
    - **Operations**: execute(), terminate()
## Diagram types and application areas

<table>
<thead>
<tr>
<th>Diagram type</th>
<th>Diagram</th>
<th>Phase</th>
<th>Application area</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use cases</strong></td>
<td>Use case diagrams</td>
<td>• Requirements • Definition • Creation • Delivery</td>
<td>Business processes, general applications</td>
</tr>
<tr>
<td><strong>Static Models</strong></td>
<td>Class diagrams • Instance diagrams</td>
<td>• Definition • Creation</td>
<td>Everywhere, the class diagram is the most Important diagram in UML</td>
</tr>
<tr>
<td><strong>Dynamic Models</strong></td>
<td>Activity diagrams • Interaction diagrams</td>
<td>• Requirements • Definition • Creation • Delivery</td>
<td>Shows the information flow and the time flow of the cooperation of the objects</td>
</tr>
<tr>
<td></td>
<td>Sequence-diagrams</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>State diagrams</td>
<td>• Requirements • Definition • Creation • Delivery</td>
<td>Representation of the dynamic behavior</td>
</tr>
</tbody>
</table>
Dynamic models

- **Dynamic Modeling** investigates:
  - When does the system do what?
    --> event, reaction
  - What does the system do?
    --> actions, activities
  - Which time-dependent behavior does the system show?
    --> conditions

- The **description tools of dynamic models** are:
  - State diagrams
  - Activity diagrams
  - Interaction diagrams (Sequence or Collaboration Diagrams)
## Diagram Types and Application Areas

<table>
<thead>
<tr>
<th>Diagram Type</th>
<th>Diagrams</th>
<th>Phase</th>
<th>Application Area</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use cases</strong></td>
<td>Use case diagrams</td>
<td>Requirements, Definition, Creation, Delivery</td>
<td>Business processes, general applications</td>
</tr>
<tr>
<td><strong>Static Models</strong></td>
<td>Class diagrams, Instance diagrams</td>
<td>Definition, Creation</td>
<td>Everywhere, the class diagram is the most Important diagram in UML</td>
</tr>
<tr>
<td><strong>Dynamic Models</strong></td>
<td>Activity diagrams, Interaction diagrams, Collaboration diagrams, Sequence-diagrams</td>
<td>Requirements, Definition, Creation, Delivery</td>
<td>Shows the information flow and the time flow of the cooperation of the objects, Timeflow structure with few messages, Timeflow structure with few classes</td>
</tr>
<tr>
<td></td>
<td>State diagrams</td>
<td>Requirements, Definition, Creation, Delivery</td>
<td>Representation of the dynamic behavior</td>
</tr>
</tbody>
</table>
State models

- A state is a characteristic of an object which exists for a limited time. The characteristics defined in the static modeling are valid for the entire life cycle of the object.

- An event is an occurrence of a negligible duration which has influence on the system in view. An event is described through its name (e.g. the event class) and possibly other parameters.

- Examples for events:
  - Creating or deleting an object
  - Sending a message to an object
  - Achieving or termination of a duration (time-out)

- Condition models describe the change of state of an object by influence of different events.
• An object can take on variegated states in its lifetime. With the help of a state diagram, you can see these states, as well as functions, that lead to the changes in the state of an object.

• A state diagram describes a hypothetical machine that at any point in time is in different states. It is comprised of:
  - A beginning state
  - Numerous possible final states
  - Numerous possible end results
  - Numerous number of transitions that describe the transition of the object to the next state
The states are visualized in rounded rectangles bound with arrows. Arrows show the transitions. Starting condition is a filled in circle, the end condition is an empty circle with a smaller filled circle in the middle.
Example of a State Diagram

1. **Elevator ready**
   - Required floor chosen
   - **Elevator active**
     - entry: Show required floor
     - do: evaluate required floor
     - do: determine direction of ride
     - do: wait 10 seconds
     - Current Load<Max
     - **Prepare ride**
       - do: close doors
       - Doors closed/begin ride
     - **Elevator rides**
       - do: ride one floor further
       - do: evaluate current floor
       - Required floor reached
       - /Start braking
     - **Prepare to exit**
       - entry: show current floor
       - do: open doors

2. **Verify ride goal**
   - do: evaluate required floor
   - Other floor chosen

3. **End**
<table>
<thead>
<tr>
<th>Diagram type</th>
<th>Diagram</th>
<th>Phase</th>
<th>Application area</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use cases</strong></td>
<td><strong>Use case diagrams</strong></td>
<td>• Requirements</td>
<td>Business processes, general applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Definition</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Creation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Delivery</td>
<td></td>
</tr>
<tr>
<td><strong>Static Models</strong></td>
<td><strong>Class diagrams</strong></td>
<td>• Definition</td>
<td>Everywhere, the class diagram is the most Important diagram in UML</td>
</tr>
<tr>
<td></td>
<td><strong>Instance diagrams</strong></td>
<td>• Creation</td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic Models</strong></td>
<td><strong>Activity diagrams</strong></td>
<td>• Requirements</td>
<td>Shows the information flow and the time flow of the cooperation of the objects</td>
</tr>
<tr>
<td></td>
<td><strong>Interaction diagrams</strong></td>
<td>• Definition</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Collaboration diagrams</strong></td>
<td>• Creation</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Sequence-diagrams</strong></td>
<td>• Delivery</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>State diagrams</strong></td>
<td>• Requirements</td>
<td>Representation of the dynamic behavior</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Definition</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Creation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Delivery</td>
<td></td>
</tr>
</tbody>
</table>
• **Activity diagrams** describe the **action possibilities that are comprised out of single activities** (steps)

• The course of a use case is frequently described by an activity diagram - it is suited for the modeling of all activities within a system

• The **activity diagram is comprised of**:
  - **Activities**: modeling elements, that represent the **execution of a set of operations**
  - **Transitions**: shows the **transition from one activity to the next**. Can be tied to other conditions
**Activity diagrams and their graphical notation**

- **Beginning condition**
- **Transition to new activity**
- **End condition**
- **Object condition:** Link of activities with conditions
- **Decision:** select a transition with condition 1
- **Synchronization of parallel activities**
Activity diagrams and their graphical notation

Lane markers:
Simplify the responsibility
## Diagram types and application areas

<table>
<thead>
<tr>
<th>Diagram type</th>
<th>Diagrams</th>
<th>Phase</th>
<th>Application area</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use cases</strong></td>
<td>Use case diagrams</td>
<td>• Requirements</td>
<td>Business processes, general applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Definition</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Creation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Delivery</td>
<td></td>
</tr>
<tr>
<td><strong>Static Models</strong></td>
<td>Class diagrams</td>
<td>• Definition</td>
<td>Everywhere, the class diagram is the most important diagram in UML</td>
</tr>
<tr>
<td></td>
<td>Instance diagrams</td>
<td>• Creation</td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic Models</strong></td>
<td>Activity diagrams</td>
<td>• Requirements</td>
<td>Shows the information flow and the time flow of the cooperation of the objects</td>
</tr>
<tr>
<td></td>
<td>Interaction diagrams</td>
<td>• Definition</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Creation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sequence-diagrams</td>
<td>• Delivery</td>
<td></td>
</tr>
<tr>
<td></td>
<td>State diagrams</td>
<td>• Requirements</td>
<td>Representation of the dynamic behavior</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Definition</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Creation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Delivery</td>
<td></td>
</tr>
</tbody>
</table>
Interaction diagrams

- Interaction diagrams represent a scenario graphically.
- Each of these interaction diagrams contains at least
  - **Life cycle data** (production, initializing, use, deletion) of objects
  - **Communication pathways** (links)
  - Reaction of receiving objects
  - Temporal order of messages
  - **Messages** along the links

- Benefits of interaction diagrams
  - Simple presentation allows quick review and reading of details

- Presentation through either
  - **Sequence diagrams** with emphasis on temporal progress or
  - **Collaboration diagrams** with emphasis on the collaboration of objects
Sequence diagrams are the most important interaction diagrams and show the timely run of a series of messages (method calls) between certain objects in a time-limited situation.

The vertical time axis is therefore important. It is represented as a dashed life line under each object.

The calls can be construed as method calls as synchronous or asynchronous messages, or also as their own methods along a life line. According to the desired degree of detail, call parameters and return values can also be given.
Sequence diagrams and their graphical notation

3.2. Object oriented modelling with UML
3.2.4. Dynamic Models

Object

Life line

Messages

Creation of new objects

Asynchronous messages

Answer

Destruction of an object

New()

Message1() (return)

Message2()

Message3() (return)

:Actor

:Object1

new()

:Object2
Collaboration diagrams

- A **collaboration diagram** shows **which objects can work together by a specified system activity** - similar to a sequence diagram
- The **timely progression of the calls** can be given here through a **prefixed order of numbers**
- Collaboration diagrams are used prior to this to demonstrate which objects, if any, can work together in a specific situation
- The **time axis** plays a **secondary role**.
Collaboration Diagrams and their graphical Notation

3.2. object oriented modelling with UML

3.2.4. Dynamic Models

Collaboration Diagrams

- **Actor**: A figure representing an external entity
- **Object1**, **Object2**, **Object3**: Figures representing objects in the system
- **Start message()**: A message sent to the object
- **Conditions**: [Condition1], [Condition2]

**Messages**:
- **1.1: message()**
- **1.2: answer: = message()**

**Lines**:
- **Associations line**: Connects objects and conditions
- **Messages, comprised of timed numbering**

**Start messages contains no numbering**

**Answer**
Example of a sequence and collaboration diagram (1)

- The assignment acquisition window sends a message „prepare“ to the assignment
- The assignment then sends the „prepare“ to every assignment position of the assignment
- Every assignment position checks the indicated article in stock
  - When this examination delivers „true“, the assignment position deletes the corresponding quantity of articles in stock from the warehouse
  - Otherwise, the quantity of the articles in stock falls under the boundaries of the order, and the article in stock requires a new delivery
Example of a sequence and collaboration diagram (2)

3.2. object oriented modeling with UML

3.2.4. Dynamic Models

Information Management in Engineering

Prof. Dr. Dr.-Ing. Jivka Ovtcharova – CSE-Lecture – Ch. 3 - WS 08/09 - Slide 42
Example of a sequence and collaboration diagram (3)

1. prepare()
2. *[for each assignment position]: prepare()
3. available := check()
4. remove()
5. repeat order := repeated order Necessary()
6. new
7. new

An Assignment Window
An Assignment
An Assignment Position
A Deliverable Item
A Repeat Order Item

Object
Message
Sequence Number
Self-Delegation

3.2. object oriented modelling with UML
3.2.4. Dynamic Models
Each message processing corresponds to a transition in the state diagram.
Correlation: Class and State Diagrams

2.2. Objektorientierte Modellierung mit UML

2.2.4. Dynamische Modelle

- Operations change the attribute values of the instance
- This means a transition into another state in the state diagram

Class Diagram

<table>
<thead>
<tr>
<th>Kl</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
</tr>
<tr>
<td>a2</td>
</tr>
</tbody>
</table>

/ z: State

| op1 |
| op2 |
| op3 |

State Diagram

- op1
- op3
- opX
3.2. object oriented modelling with UML

3.2.4. Dynamic Models

Class diagram

Classes

Instances

Sequence diagram

KI1

KI2

ob1:KI1

ob2:KI1

ob3:KI2
Outline

Lecture Content

1. Preface

2. Basics

3. Object orientation
   3.1. Introduction and basic concepts
      3.1.1. Introduction
      3.1.2. Objects and classes
      3.1.3. Attributes of object orientation
   3.2. Object-oriented modelling with UML
      3.2.1. Intro
      3.2.2. Use Case models
      3.2.3. Static models
      3.2.4. Dynamic models
      3.2.5. Summary
   3.3. Methods of the OO Analysis and Design
Weaknesses of UML

- Semantics presently described only textually, only informal, *ambiguities possible*;
- Methodology of utilization of UML insufficiently researched (note: multiplicity of methodologies possible, use case specific)
- Little support for real-time applications
- Support of *semantics-containing refining*
Typical Problems with Graphical Modeling

- Relationships between diagrams:
  - Overlap can lead to inconsistencies,
  - Separation can lead to incompletion
- Various interpretation of diagrams by developers/clients
- Step-by-step refining of diagrams cannot be viewed. The addition of details often does not preserve the semantics
- Scalability of graphic notation often problematic, hierarchical draft or other structuring necessary
- Tabular representation sometimes clearer
• Notation for the description of the structure of object oriented programs
• Based on UML, entire program fragments can be generated
  - MDA (Model Driven Architecture)
• Different layers describing the same circumstances
• Dynamic and static diagram types, describing both the structure and the behavior
• CASE Tools (Computer Added Software Engineering) can generate source code from UML diagrams and vice versa.
  - Code $\rightarrow$ UML = Reverse Engineering
  - Code $\rightarrow$ UML $\rightarrow$ Code = Roundtrip Engineering
• Overview on CASE / UML Tools: http://jeckle.de/umltools.html
Recommended Literature

- Dr. Christian Prehofer: Vorlesung Objektorientierung, TU München, SS 2001
- Dr. Uwe Aßmann, Vorlesung Informatik II, Institut für Programmstrukturen und Datenorganisation, Universität Karlsruhe, SS 2000
- Martin Fowler / Kendall Scott: UML konzentriert, Addison-Wesley, 2te Auflage, 2000
- G. Krüger: Handbuch der Java-Programmierung, Addison-Wesley, 2000
- http://www.rational.com/uml/
- http://www.wikipedia.de
Recommended Literature


[HiKa99]  M. Hitz, G. Kappel: „UML@Work“, dpunkt.verlag, 1999


Recommended Literature


3.3.2. Example: Modeling a M&S System
Why Software-Engineering?

- Ariane 5: Failure of its first flight on 4th of June 1996:
  Rocket blows itself up 36 sec after start
- Source of failure: software
- Parts of the software were taken from Ariane 4
- Ariane 5 can accelerate faster: variable overflow
- Steering system breaks down, status message to navigation system
- Navigation system interprets this as flight data
- Because of „serious deviation from course“ the rocket blows itself up

[Source: Wikipedia]
What is Software Engineering?

- The area of computer science that deals with the **methods and tools for an engineering-type design and implementation of software**
- Software engineering is performed today **almost exclusively with help of special software tools** (so called CASE-Tools)
- **Special methods of software engineering are dependent on the basic principles of program development**, such as **structured or object oriented programming**
- The **process of software engineering contains all of the phases**, from problem specification up to the final use of the program or program systems
• Emerged from the software crisis (NATO conference 1968)
  - Software costs > hardware
  - Lack of transparency in the software development process
  - Engineering view on the software development process

• High quota of failed software projects (CHAOS study)
  - Study on success/failure of software projects
  - Only about 17% of all software projects are successful regarding
    ▪ Costs
    ▪ Timeframe
    ▪ Requirements
• Water fall model
3.3. Methods of OO Analysis and Design

3.3.1 Introduction

- V-Model
  - Military software development

Diagram:
- Requirements definition
- Use case scenarios
- Commissioning
- Preliminary design
- Test cases
- System test
- Detailed design
- Test cases
- Integration test
- Module implementation
- Test cases
- Module test
- System test
Software Engineering - Model 3

3.3. Methods of OO Analysis and Design

3.3.1 Introduction

• Prototype Model

- Requirements analysis
- Prototyping
- Software development
- Prototyping
- Implementation
- Integration and tests
- Usage and maintenance
3.3. Methods of OO Analysis and Design

3.3.1 Introduction

- Determine objectives
- Solution variants
- Boundary conditions

- Next phase planning

- Life-Cycle-Plan
- Development plan
- Integration and test plan
- Implementation

- Requirements

- Design
- Test
- Installation

- Evaluation of solution alternatives
- Overcome risks
- Prototype creation

- Development and verification

[Beister 08]