Example for an activity diagram (1)

- **Task**: determine the age of the youngest person in a line.

- **Solution description**:

- **Algorithm**:
  1: Go to the first person.
  2: Ask for its age.
  3: Memorize its age.
  4: As long as not all persons were questioned, repeat steps 4.1 - 4.3:
    4.1: Go to next person.
    4.2: Ask for its age.
    4.3: If its age is smaller than the memorized age, memorize this age.
  5 The youngest person is „memorized age“ years old.
Example for an activity diagram (2)

- Solution with UML (Activity diagram):

1. Start
2. Go to first person
3. Input: Ask for age
4. Memorize age
5. [all persons questioned == false]
6. Go to next person
7. Input: ask for its age
8. Memorize new age
9. [requested age >= memorized age]
10. [requested age < memorized age]
11. Output: youngest person is 'memorized age' years old

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
Software Engineering - Model 1

- Water fall model

```
requirements analysis
  Validation

software development
  Validation

implementation
  Validation

integration and tests
  Validation

usage and maintenance
  Validation
```

• V- Model
  - Military software development

Requirements definition → Use case scenarios → Commissioning

Preliminary design → Test cases → System test

Detailed design → Test cases → Integration test

Module implementation → Test cases → Module test
3.3.1 Introduction

Prototype Model

- Requirements analysis
- Prototyping
- Software development
- 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
- Evaluate solution alternatives
- Overcome risks
- Prototype creation
- Next phase planning
- Development and verification

[Beister 08]
Rational Unified Process (RUP) is an object oriented process model for software development. It is also a commercial product of the Rational Software company - which since 2002 has been a part of IBM.

- RUP uses Unified Modeling Language (UML) as the notation language
- The life-cycle of RUP is divided into 4 principle phases:
  - **Inception** (conceptualization: determine project extent)
  - **Elaboration** (Design: develop convertible architecture)
  - **Construction** (Implementation: fill the architectural skeleton with functionality)
  - **Transition** (Product transfer: transfer application to the user environment)
Rational Unified Process: Dynamic Structure

Distribution of tasks in small steps (Iterations)

Source: IBM - Rational
Phases of the Rational Unified Process: Inception and Elaboration

- **Inception** (Conceptualization)
  - Specification of the vision for the end product
  - Specification of the essential transactions
  - Definition of the project extent
  - Project costs and risks
  - Finish: Life Cycle Objective Milestone

- **Elaboration** (Design)
  - Specification of product characteristics
  - Design of architecture
  - Design of the necessary activities and resources
  - Finish: Life Cycle Architecture Milestone
Phases of the Rational Unified Process: **Construction and Transition**

- **Construction** (Implementation)
  - Generation of the product
  - Development of the architecture
  - Result: manufactured product
  - Finish: Initial Operational Capability Milestone

- **Transition** (Product transfer)
  - Release of the product to the user
  - Test of the quality level
  - Delivery, Training, Customer Support, Maintenance
  - Finish: Release Milestone
The results of the phases are the so-called **Milestones**:

- **Lifecycle objectives milestone**: Vision including a rudimentary use case model, (essential functionalities), tentative architecture, identification of important risks, design of the engineering phase

- **Lifecycle architecture milestone**: Architecture prototype, detailed use case model, design of the construction phase

- **Initial operational capability milestone**: Design models and Beta-Release of the Software

- **Product release milestone**: Release in production quality
The disciplines (interchangeably used with working steps) orient themselves to **special roles within the developing team**, and are administered by certain people or groups.

In detail, these disciplines are:

- **Central work steps**
  - Business Modeling
  - Requirements
  - Analysis & Design
  - Implementation
  - Test
  - Deployment

- **Supported work steps**
  - Configuration & Change Management
  - Project Management
  - Environment
Rational Unified Process

3.3. Methods of OO Analysis and Design

3.3.1 Introduction
Business Modelling

• Define the **core problem**:
  - Which functions should the system offer?
  - How should the operating surface behave?
  - How efficient, safe, … must the system be?

• Define the **relevant environment**:
  - type, number of users
  - available hardware, previous software

• Estimate **feasibility**
  - Technical, personal capacity
  - Costs (and uses)
Requirements

- Purpose of the system
- Desired functions
- Correct / false input, corresponding reactions
- Configuration of the operating surface
- Efficiency - goals
- Document requirements
- Protection / Safety aspects
- Standards to use
- Time plan, expenditure estimate, risk estimate (approx.)

3.3. Methods of OO Analysis and Design

3.3.1 Introduction
- Rough system structure
  - Analyze the components
  - Define teamwork
  - Test design vs. requirements

- Describing the components
  - Purpose and role of a component
  - Service offered from one component

- Relationships between components
  - Which components can use the other ones
  - Forming of design decisions
Implementation and tests

- Implement the components
  - Choose data structures
  - Choose algorithms
  - Formulate in the programming language

- Document the components
  - How do the components accomplish their tasks?
  - Establish implementation alternatives

- Test components (vs. design)
  - Arrange the test environment, gather test data
  - Perform a run through of the test
  - Verification

- Test externally programmed components
Deployment

• **Application**
  - Install the system for the customer
  - accommodate
  - Inspection by the customer
  - Transfer of the system
  - Train the User

• **Maintenance**
  - Correct errors
  - Modify: functionalities adapt/ better
### Application of UML in RUP - Disciplines

<table>
<thead>
<tr>
<th>Diagram type</th>
<th>Diagram</th>
<th>Disciplines</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use case</strong></td>
<td>Use case diagram</td>
<td>Business modeling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Requirements</td>
</tr>
<tr>
<td><strong>Static model</strong></td>
<td>Class diagram</td>
<td>Analysis &amp; Design</td>
</tr>
<tr>
<td></td>
<td>Instance diagram</td>
<td>Implementation</td>
</tr>
<tr>
<td><strong>Dynamic model</strong></td>
<td>Collaboration diagrams</td>
<td>Analysis &amp; Design</td>
</tr>
<tr>
<td></td>
<td>Sequence diagrams</td>
<td>Requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Analysis &amp; Design</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Implementation</td>
</tr>
<tr>
<td></td>
<td>State diagrams</td>
<td>Requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Analysis &amp; Design</td>
</tr>
<tr>
<td></td>
<td>Activity diagrams</td>
<td>Requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Analysis &amp; Design</td>
</tr>
</tbody>
</table>
Recommended Literature


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


• According to definition 1.1 from lecture 1, computer science is the science that concerns itself with the structure, effectiveness, construction principles, and applications of information processing systems as well as their usage.

• In order to complete these tasks, computer science developed a formalism, which allows verified characteristics and exceptions to structures, effectiveness, and principles.

• This formalism and the possibility for their verification are prerequisites for a systematic data processing, which can be automatized through a computer.
• **A data structure** in computer science is a **specific form to manage data and link the structures** with one another, so that it’s possible to access the data and especially manipulate it in an applicable manner.

• Data structures are always linked with certain operations to allow these manipulations.

• Application examples are
  - Data pool systems
  - Networks
  - Lists of assembly parts
  - Telephone book
  - Pictures

Source: [www.witaf.at](http://www.witaf.at)
Source: [irc.rz.uni-karlsruhe.de](http://irc.rz.uni-karlsruhe.de)
Source: [GWDG](http://www.gwdg.de)

Source: [www.witaf.at](http://www.witaf.at)

The different types of data structures are applicable for different specific tasks according to their characteristics, linked operations, and implementation.

Decision criteria are, for example
- Amount of data to be managed
- Data fluctuation
- Necessary access possibilities
- Run time behavior of the data access
- Alterability of the data structure
A graph is a formation of vertices that are connected through edges.

Graphs are visually described through images and through mathematical structures. This general definition allows the description of complex associations and characteristics.
**Definition 3.1:** $G = (V, E)$ is a **graph** if and only if

1. $V$ is heap (vertices)
2. $E \subseteq \{ \{v_1, v_2\} \mid v_1, v_2 \in V \}$ (edges)

**example**

$G_1 :$

\[
\begin{array}{c}
V = \{v_1, v_2, v_3\} \\
E = \{\{v_1, v_2\}, \{v_2, v_3\}\}
\end{array}
\]

**Nomenclature**

In the graph $G_2 = (\{v_1\}, \{v_1\})$

the edge $e_1 = \{v_1\}$ is a **loop**.
Definition 3.2: Let $G = (V,E)$ and $G' = (V',E')$. 

$G'$ is a **subgraph** of $G$ if and only if 

$E' \subseteq E$ and $V' \subseteq V$ \ ($\iff G' \subseteq G$)

**example**

$G_3 :$

$V = \{v_1, v_2, v_3, v_4\}$

$E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_2\}\}$

$G_3' :$

$V' = \{v_1, v_2\}$

$E' = \{\{v_1, v_2\}, \{v_2\}\}$
Definition 3.3: A path $\pi$ in $G$ is a sequence of $\pi = (v_0, v_1, \ldots, v_n)$ with
\[ \forall 0 \leq i < n : \{v_i, v_{i+1}\} \in E \]

Example

$G_4 :$

Nomenclature

$n = |\pi| = \text{Length of } \pi$

$\text{Start}(\pi) = v_0$

$\text{End}(\pi) = v_n$

The total of all paths $\pi$ in $G$ is $\text{Path}(G)$
Definition 3.4: The path $\pi$ in $G$ is a **circle** if and only if
\[ \text{Start}(\pi) = \text{End}(\pi) \]

example
\[ G_4 : \]

\[ \pi = (v_2, v_4, v_3, v_2) \] is a circle
\[ \pi = (v_1, v_4, v_2) \] is NOT a circle

Nomenclature

A path $\pi$ is a **simple edge** circle, when each edge on the path $\pi$ is contained only once in the path.
**Definition 3.5:** the branching factor $d$ for a graph $G$ and a vertex $v$ is the number of edges connected with the vertex $v$.

$$d(G,v) = \#(v' \in V \mid \{v,v'\} \in E)$$

The branching factor $d$ for a graph $G$ gives the maximal branching factor for all vertices.

$$d(G) = \text{Max}(d(G,v))$$

**example**

$G_5$:

- $d(G_5,v_3) = 2$
- $d(G_5,v_4) = 1$
- $d(G_5) = 5$
A graph $G$ was previously described by the amounts $E$ and $V$ and can also be described through the input of a **weight function $g$**.

Let $G = (V, E)$ and $g : V \times V \rightarrow \mathbb{R}$ be the weight function.

Therefore:

\[
g(v_1, v_2) = \infty \Rightarrow \{v_1, v_2\} \notin E
\]

\[
g(v_1, v_2) \neq \infty : \text{weight of the edge } \{v_1, v_2\} \in E
\]

That means that an edge is an element of the graph when the weight is smaller than $\infty$.

**Example**

\[
G_6:
\]

\[
\begin{align*}
G_6 : & \quad g(v_1, v_2) = 2 \\
& \quad g(v_2, v_3) = \infty \\
& \quad g(v_1, v_3) = 5
\end{align*}
\]
Definition 3.6: A directed graph $G$ is a pair $G = (V,R)$ with

1. $V$ is a heap (vertices)
2. $R \subseteq V \times V$ (directed edges)

(The order of the vertices in an edge gives the direction)

example

$G_6:$

$V = \{v_1,v_2,v_3,v_4\}$

$R = \{\{v_2,v_1\}, \{v_3,v_1\}, \{v_1,v_4\}, \{v_3,v_4\}, \{v_2,v_2\}\}$
4. Data Structures
4.2 Graphs

Example of a Graph (1)

KVV – bus and street car map
How do I get from the main train station to the university (path)?

Start(\(\pi\)) = *Hauptbahnhof*

End(\(\pi\)) = *Kronenplatz / KIT Campus Süd*

path \(\pi\) = \{*Hauptbahnhof, Poststraße, Augartenstraße, Kongresszentrum, Ettlinger Tor, Marktplatz, Kronenplatz / KIT Campus Süd*\}

How much time do I need for this path (edge quantifier)?

\[ \text{weight}(G_\pi) = \sum_{e \in E} g(e) = 12 \]  
(Sum of all edge weights)

\[ G_\pi = (V,E) \subseteq KVV \]  
\((G_\pi\) is the subgraph of the line graph)