Overview for the Lecture

- Three ways to deal with complexity
  - Abstraction and Modeling
  - Decomposition
  - Hierarchy
- Introduction into the UML notation
- First pass on:
  - Use case diagrams
  - Class diagrams
  - Sequence diagrams
  - Statechart diagrams
  - Activity diagrams

Abstraction

- Complex systems are hard to understand
  - The “7 +/- 2 chunks in the brain” phenomena
    - Our short term memory cannot store more than 7+/-2 pieces at the same time -> limitation of the brain
    - TUM Phone Number: 498928918204

Abstraction

- Complex systems are hard to understand
  - The 7 +/- 2 phenomena
    - Our short term memory cannot store more than 7+/-2 pieces at the same time -> limitation of the brain
    - TUM Phone Number: 498928918204
- Chunking:
  - Group collection of objects to reduce complexity
  - 4 chunks:
    - State-code, city-code, TUM-code, Office-Part
Abstraction

• Complex systems are hard to understand
  • The 7 + 2 phenomena
    • Our short term memory cannot store more than 7 + 2 pieces at the same time - -> limitation of the brain
  • TUM Phone Number: 498928918204

• Chunking:
  • Group collection of objects to reduce complexity
    • State-code, city-code, TUM-code, Office-Part

We use Models to describe Software Systems

• Object model: What is the structure of the system?
• Functional model: What are the functions of the system?
• Dynamic model: How does the system react to external events?
• System Model: Object model + functional model + dynamic model

Other models used to describe Software System Development

• Task Model:
  • PERT Chart: What are the dependencies between tasks?
  • Schedule: How can this be done within the time limit?
  • Organization Chart: What are the roles in the project?

• Issues Model:
  • What are the open and closed issues?
  • What blocks me from continuing?
  • What constraints were imposed by the client?
  • What resolutions were made?
  • These lead to action items

Issue-Modeling

Proposal 1: The earth!

Proposal 2: The sun!

Issue: What is the center of the Universe?

Pro: Copernicus says so.

Con: Jupiter’s moons rotate around Jupiter, not around Earth.

Pro: Aristotle says so.

Pro: Change will disturb the people.
**Issue-Modeling**

**Issue:** What is the Center of the Universe?

**Proposal 1:** The earth!

- **Pro:** Aristotle says so.
- **Pro:** Change will disturb the people.

**Proposal 2:** The sun!

- **Pro:** Copernicus says so.

**Proposal 3:** Neither!

- **Pro:** Galaxies are moving away from each other.

**Resolution (1615):** The church decides proposal 1 is right.

**Resolution (1998):** The church declares proposal 1 was wrong.

**Proposal 3:** Neither!

- **Pro:** Galaxies are moving away from each other.

---

**2. Technique to deal with Complexity: Decomposition**

- A technique used to master complexity ("divide and conquer")
- Two major types of decomposition
  - Functional decomposition
  - Object-oriented decomposition
- Functional decomposition
  - The system is decomposed into modules
  - Each module is a major function in the application domain
  - Modules can be decomposed into smaller modules.

**Decomposition (cont’d)**

- Object-oriented decomposition
  - The system is decomposed into classes ("objects")
  - Each class is a major entity in the application domain
  - Classes can be decomposed into smaller classes

Which decomposition is the right one?

---

**Functional Decomposition**

- The functionality is spread all over the system
- Maintainer must understand the whole system to make a single change to the system
- Consequence:
  - Source code is hard to understand
  - Source code is complex and impossible to maintain
  - User interface is often awkward and non-intuitive.
### Functional Decomposition

- The functionality is spread all over the system
- Maintainer must understand the whole system to make a single change to the system
- Consequence:
  - Source code is hard to understand
  - Source code is complex and impossible to maintain
  - User interface is often awkward and non-intuitive

- Example: Microsoft PowerPoint’s Autosshapes
  - How do I change a square into a circle?

### Changing a Square into a Circle

#### Functional Decomposition: Autoshape

- Draw
  - Change
    - Change Rectangle
    - Change Oval
    - Change Circle
  - Draw Rectangle
  - Draw Oval
  - Draw Circle

#### Object-Oriented View

- Autoshape
  - Draw()
  - Change()
Class Identification

- **Basic assumptions:**
  - We can find the classes for a new software system: Greenfield Engineering
  - We can identify the classes in an existing system: Reengineering
  - We can create a class-based interface to an existing system: Interface Engineering

Class Identification (cont’d)

- **Why can we do this?**
  - Philosophy, science, experimental evidence
- **What are the limitations?**
  - Depending on the purpose of the system, different objects might be found
- **Crucial**
  - Identify the purpose of a system

3. Hierarchy

- So far we got abstractions
  - This leads us to classes and objects
  - “Chunks”
- Another way to deal with complexity is to provide relationships between these chunks
  - One of the most important relationships is hierarchy
  - Two special hierarchies
    - “Part-of” hierarchy
    - “Is-kind-of” hierarchy

Part-of Hierarchy (Aggregation)

- Computer
  - I/O Devices
  - CPU
  - Memory
    - Cache
    - ALU
    - Program Counter

Is-Kind-of Hierarchy (Taxonomy)

- Cell
  - Muscle Cell
  - Blood Cell
    - Erythrocyte
    - Megacaryocyte
  - Nerve Cell
    - Axon
    - Dendrite
    - Synapse
Where are we now?

- Three ways to deal with complexity:
  - Abstraction, Decomposition, Hierarchy
- Object-oriented decomposition is good
  - Unfortunately, depending on the purpose of the system, different objects can be found
- How can we do it right?
  - Start with a description of the functionality of a system
  - Then proceed to a description of its structure
  - Ordering of development activities
  - Software lifecycle

Models must be falsifiable

- Karl Popper (Objective Knowledge):
  - There is no absolute truth when trying to understand reality
  - One can only build theories, that are "true" until somebody finds a counter example
- Falsification: The act of disproving a theory or hypothesis
  - The truth of a "theory" is never certain. We must use phrases like:
    - "by our best judgment", "using state-of-the-art knowledge"
  - In software engineering any model is a theory:
    - We build models and try to find counter examples by:
      - Requirements validation, user interface testing, review of the design, source code testing, system testing, etc.
  - Testing: The act of disproving a model.

Concepts and Phenomena

- Phenomenon
  - An object in the world of a domain as you perceive it
    - Examples: This lecture at 6:05, my black watch
- Concept
  - Describes the common properties of phenomena
  - Example: All lectures on software engineering
  - Example: All black watches
  - A Concept is a 3-tuple:
    - Name: The name distinguishes the concept from other concepts
    - Purpose: Properties that determine if a phenomenon is a member of a concept
    - Members: The set of phenomena which are part of the concept.

Abstract Data Types & Classes

- Abstract data type
  - A type whose implementation is hidden from the rest of the system
- Class:
  - An abstraction in the context of object-oriented languages
  - A class encapsulates state and behavior
    - Example: Watch

Type and Instance

- Type:
  - In the context of programming languages
  - Example:
    - Name: int
    - Purpose: integral number
    - Members: 0, 1, 2, 3, …
  - Instance:
    - Member of a specific type
    - The type of a variable represents all possible instances of the variable
  - The following relationships are similar:
    - Type <=> Variable
    - Concept <=> Phenomenon
    - Class <=> Object
**Systems**

- A **system** is an organized set of communicating parts
  - **Natural system**: A system whose ultimate purpose is not known
  - **Engineered system**: A system which is designed and built by engineers for a specific purpose
- The parts of the system can be considered as systems again
  - In this case we call them **subsystems**

**Examples of natural systems:**
- Universe, earth, ocean

**Examples of engineered systems:**
- Airplane, watch, GPS

**Examples of subsystems:**
- Jet engine, battery, satellite.

---

**Systems, Models and Views**

- A **model** is an abstraction describing a system or a subsystem
- A **view** depicts selected aspects of a model
- A **notation** is a set of graphical or textual rules for depicting models and views:
  - formal notations, "napkin designs"

**System: Airplane**

**Models:**
- Flight simulator
- Scale model

**Views:**
- Blueprint of the airplane components
- Electrical wiring diagram
- Fuel system
- Sound wave created by airplane

---

**Systems, Models and Views ("Napkin" Notation)**

**Implementation results in source code (Java):**
```java
public class StockExchange {
    public m_Company = new Vector();
    
    public m_StockExchange (
        public m_Company = new Vector();
    );
}
```

---

**Model-Driven Development**

1. Build a platform-independent model of an applications functionality and behavior
   a) Describe model in modeling notation (UML)
   b) Convert model into platform-specific model
2. Generate executable from platform-specific model

**Advantages:**
- Code is generated from model ("mostly")
- Portability and interoperability
- Model Driven Architecture effort:
  - [http://www.omg.org/mda](http://www.omg.org/mda)
  - OMG: Object Management Group

---

**Model-driven Software Development**

**Reality:** A stock exchange lists many companies.
Each company is identified by a ticker symbol.

**Analysis results in analysis object model (UML Class Diagram):**
```
StockExchange
|      * Company
|      |        * List
|      |            * TickerSymbol
```

**Implementation results in source code (Java):**
```java
public class StockExchange {
    public m_Company = new Vector();
    
    public m_StockExchange {
        public m_Company {
            public int m_tickerSymbol;
        }
    }
}
```
Application vs Solution Domain

- **Application Domain (Analysis):**
  - The environment in which the system is operating
- **Solution Domain (Design, Implementation):**
  - The technologies used to build the system
- Both domains contain abstractions that we can use for the construction of the system model.

Object-oriented Modeling

<table>
<thead>
<tr>
<th>Application Domain (Phenomena)</th>
<th>Solution Domain (Phenomena)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Model (Concepts/Analysis)</td>
<td>System Model (Concepts/Design)</td>
</tr>
<tr>
<td>Aircraft, TrafficController, FlightPlan, Airport, FlightPlanDatabase, MapDisplay, SummaryDisplay, TrafficControl</td>
<td>UML Package</td>
</tr>
</tbody>
</table>

What is UML?

- **UML** (Unified Modeling Language)
  - Nonproprietary standard for modeling software systems, OMG
  - Convergence of notations used in object-oriented methods
    - OMT (James Rumbaugh and colleagues)
    - Booch (Grady Booch)
    -OOSE (Ivar Jacobson)
  - Current Version: UML 2.2
- Commercial tools: Rational (IBM), Together (Borland), Visual Architect (business processes, BCD)
- Open Source tools: ArgoUML, StarUML, Umbrello
- Commercial and Opensource: PoseidonUML (Gentleware)

UML: First Pass

- **Use case diagrams**
  - Describe the functional behavior of the system as seen by the user. What a system does rather than how.
- **Class diagrams**
  - Describe the static structure of the system: Objects, attributes, associations.
- **Sequence diagrams**
  - Describe the dynamic behavior between objects of the system.
- **Statechart diagrams**
  - Describe the dynamic behavior of an individual object
- **Activity diagrams**
  - Describe the dynamic behavior of a system, in particular the workflow.

UML Core Conventions

- All UML Diagrams denote graphs of nodes and edges
  - Nodes are entities and drawn as rectangles or ovals
    - Rectangles denote classes or instances
    - Ovals denote functions
  - Names of Classes are not underlined
  - Names of Instances are underlined
    - myWatch:SimpleWatch:Firefighter
  - An edge between two nodes denotes a relationship between the corresponding entities
Use case diagrams represent the functionality of the system from user’s point of view.

Class diagrams represent the structure of the system.

Sequence diagrams represent the behavior of a system as messages (“interactions”) between different objects.

Statechart diagrams represent behavior of a single object with interesting dynamic behavior.
Other UML Notations

UML provides many other notations, for example:
- Deployment diagrams for modeling configurations
  - Useful for testing and for release management
- We introduce these and other notations as we go along in the lectures
  - OCL: A language for constraining UML models

What should be done first? Coding or Modeling?

- It all depends....
- Forward Engineering
  - Creation of code from a model
  - Start with modeling
- Reverse Engineering
  - Creation of a model from existing code
  - Interface or reengineering projects
- Roundtrip Engineering
  - Move constantly between forward and reverse engineering
  - Reengineering projects
  - Useful when requirements, technology and schedule are changing frequently.

UML Basic Notation Summary

- UML provides a wide variety of notations for modeling many aspects of software systems
- Today we concentrated on a few notations:
  - Functional model: Use case diagram
  - Object model: Class diagram
  - Dynamic model: Sequence diagrams, statechart

Additional References

- Martin Fowler
- Grady Booch, James Rumbaugh, Ivar Jacobson
- Commercial UML tools
  - Rational Rose XDE for Java
  - Together (Eclipse, MS Visual Studio, JBuilder)
- Open Source UML tools
  - [http://java-source.net/open-source/uml-modeling](http://java-source.net/open-source/uml-modeling)
  - ArgoUML, UMLet, Violet, ...