## 17 Persistence (3)

★ **Persistence in Object-oriented Systems**

- Objects survive the termination of the environment (thread, application execution) in which they were instantiated or used
- Powerful mechanism in OO operating systems for
  - ➤ Data storage
  - ➤ Data transport
- Examples
  - ➤ File systems
  - ➤ Database systems
  - ➤ Persistent communication objects
- Properties of the object-oriented programming model are automatically inherited by all mechanisms which are constructed on its basis






# C.8 Object-oriented Software Development

- Object-oriented software engineering
- Software development phases
  - ➤ Requirement analysis
  - ➤ Object-oriented analysis
  - ➤ Object-oriented design
  - ➤ Implementation
  - ➤ Test
- UML: The Unified Modeling Language — a short sketch
  - ➤ Use case diagram
  - ➤ Class diagram
  - ➤ Interaction diagrams






## 1 Object-oriented Software Engineering

- 1980 - 1990: OO programming languages establish
  - ➤ Smalltalk
  - ➤ C++
  - ➤ Eiffel
  - ➤ LOOPS, Flavors
- from 1990: Propagation of object-oriented software engineering methods
  - ➤ Sally Shlaer & Steve Mellor
  - ➤ Peter Coad & Ed Yourdon
  - ➤ Grady Booch
  - ➤ Jim Rumbaugh et al. (OMT: Object Modeling Technique)
  - ➤ Ivar Jacobson (OOSE, Objectory)
- 1995: Booch, Rumbaugh, and Jacobsen start development of UML






## 2 Why Object-Oriented Software Engineering?

- Optimal usage of the object-oriented paradigm
- Transition to OO programming languages is easy
- ***Paradigm Shift*** ist the problem
  - ➤ advantages of object orientation depend on the way a problem is tackled
  - ➤ otherwise danger of wrong usage of oo concepts
    - ➥ C++
- Object-oriented software engineering methods give guidance to a correct deployment of object-oriented concepts
  - ➤ proper determination of classes
  - ➤ proper usage of inheritance
  - ➤ …

## 2 Why Object-Oriented Software Engineering? (2)

- ▲ Software often too complex to be comprehended altogether
- ➥ Building models to abstract from details of the system
  - ➤ different viewpoints of the system — e. g.
    - – specification view
    - – design view
  - ➤ different levels of abstraction
  - ➤ abstractions for parts of the system
  - ➤ different aspects of the system
    - – requirements
    - – static aspects — structure
    - – dynamic aspects — behavior






## 2 Why Object-Oriented Software Engineering? (3)

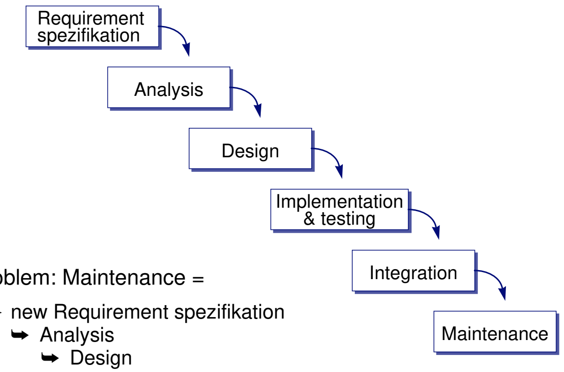
- ▲ Control of the development process
- ■ Uncontrolled development leads to
  - ➤ unsufficient analyses of requirements
    - ➥ development does not meet the principal's requirements
  - ➤ early start with implementation
    - ➥ analyses and design not sufficient
    - ➥ errors in analyses or design are detected late
    - ➥ high costs due to subsequent changes
- ▲ Software engineering methods give a guideline to
  - ➤ handle today's software complexity
  - ➤ develop software economically
  - ➤ make software development a controlled, controlable, and calculable process






## 3 Software-Development Phases

- ■ Traditional: Waterfall approach



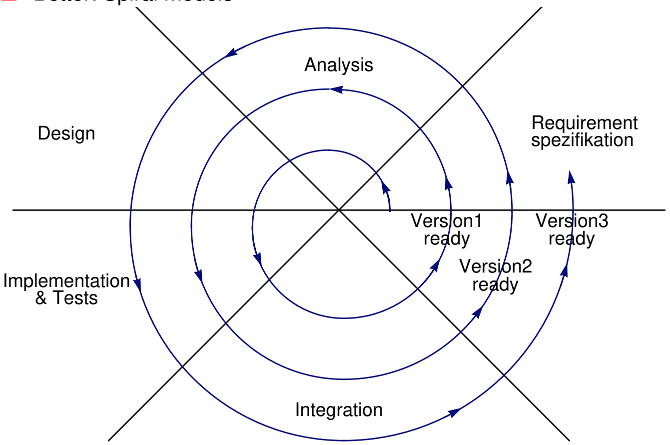

- ■ Problem: Maintenance =
  - ➤ new Requirement spezifikation
    ➥ Analysis
    ➥ Design
    ➥ new Implementation
    ➥ …





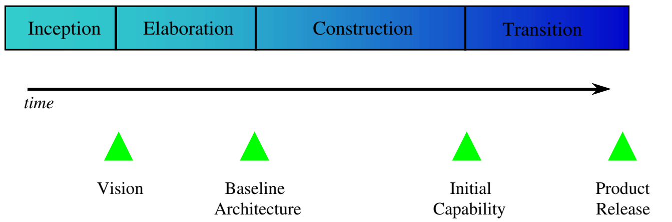

## 3 Software-Development Phases (2)

- ■ Better: Spiral models



# 3 Software-Development Phases (3)

- Further improved: Iterative approach
- Four lifecycle phases



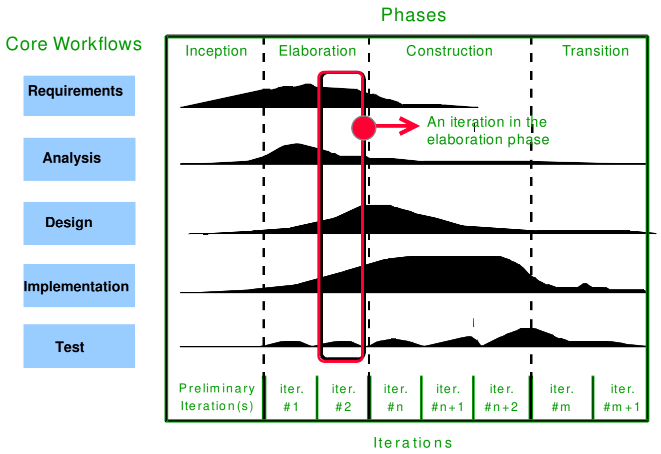

- ➤ Inception: define scope of the system, develop business case
- ➤ Elaboration: plan project, specify features, baseline the architecture
- ➤ Construction: build the product
- ➤ Transition: transition the product to its users






# 3 Software-Development Phases (4)

- Iterations and Workflows









# C.9 Object-oriented Analysis

- *What* shall my system do?
- Basis: Analysis of the “real world”
  - ➤ Components, terms, tasks
  - ➤ Requirements and constraints
- Abstraction from unimportant aspects & implementation details
- Abstraction from implementational details
  - – unimportant for the moment: how do I implement things?
  - + to be considered: aspects of the implementation environment and runtime platform which means do I have?






# C.9 Object-oriented Analysis (2)

## 1 The Process

- Analyze requirements
- Describe use cases
- Determine terms of the problem domain
- Find objects

} Requirements Model

- Organize objects
- Determine first internal structures of the objects
- Describe interactions of objects

} Analysis Model

- Determine operations of objects

## 2 OOA — Requirement Analysis

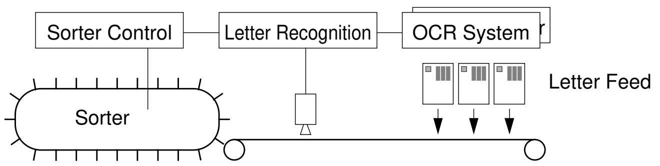
- Focus on problem domain
- Determine goals and tasks
  - ➤ Basis for alternative solutions
  - ➤ Basis for reviews and evalutions
- Customers view
- Performance-, user- and architecture-related requirements
- Informal description






## 3 OOA — Example Requirement Analysis

- Automatic letter sorter




- Problem domain: Letter (ZIP code) recognition
- Task: recognize ZIP on a letter and tell sorter control system number of output tray
- Requirements:
  - recognition may last max. 1 minute
  - separate OCR hosts for character recognition
  - statistics about ZIP frequency and errors






## 4 OOA — Use Cases

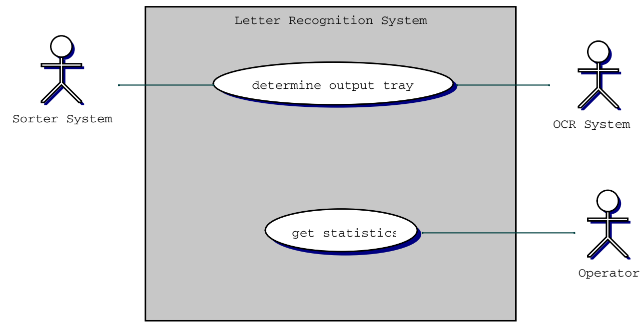
- Describe interactions between "users" and system
- Parties:
  - ➤ Actor: Person or another component of the system **! playing a specific role**
  - ➤ Use case: "Dialog" between actor and system for a specific purpose
- Substantiation of requirement analysis
- First step of modularization of the problem






## 4 OOA — Use Cases (2)

- Letter sorter
  - ➤ sorter control requests number of output tray
  - ➤ user retrieves statistics
- UML Diagram



# 5 OOA — Finding Objects

- Identify terms of the problem domain
  - ➤ look for nouns in the terminology of the problem domain

    EXAMPLE
    - ➤ Sorter Control
    - ➤ OCR
    - ➤ Letter
    - ➤ Camera

  - ➤ Strategy:
    - – principal sketches his view
    - – system analyst records the terms

- Goal: Establish a basis for the following work
  - ➤ NOT to define the entire system






# 6 OOA — Organizing Objects

- ➔ Structuring the model ➔ Analysis Model

- different categories of objects
  - ➔ Boundary objects
  - ➔ Entity objects
  - ➔ Control objects






# 6 OOA — Organizing Objects (2)

**Boundary Objects**

- Represent actors of the use cases
- Starting point for activities in the system
- Interfaces of the system to the "outside world"

EXAMPLE
- ➤ Sorter Control
- ➤ Operator Panel
- ➤ OCR System

**Entity Objects**

- Represent the actual system state
  - ➤ are rather long-lived
  - ➤ often persist the execution of a use case

EXAMPLE
- ➤ Statistics
- ➤ Letter
- ➤ Picture






# 6 OOA — Organizing Objects (3)

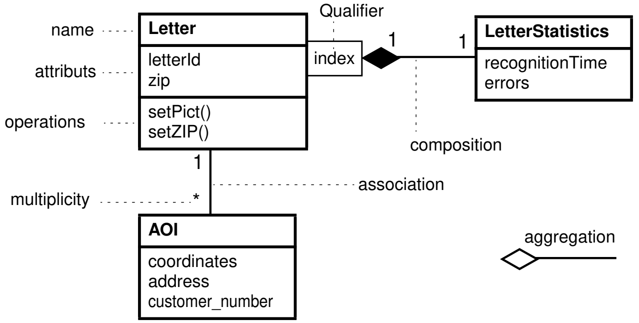
**Control objects**

- Problem: task / activity cannot be assigned to one of the objects
  - ◆ main focus is on the procedure
  - ◆ such procedures often result directly from uses
    - ➤ define something which is responsible for the procedure
      - ➥ create an object for it

EXAMPLE
- ➤ Task: perform ZIP recognition
  - – take picture
  - – look for address area
  - – give job to OCR host
  - – tray number to sorter control
  - ➥ Object *LetterController*

# 6 OOA — Organizing Objects (4) UML Notation: Class Diagrams

- Class with its attributes (variables) and operations (methods)
- Relations between classes
- Example:



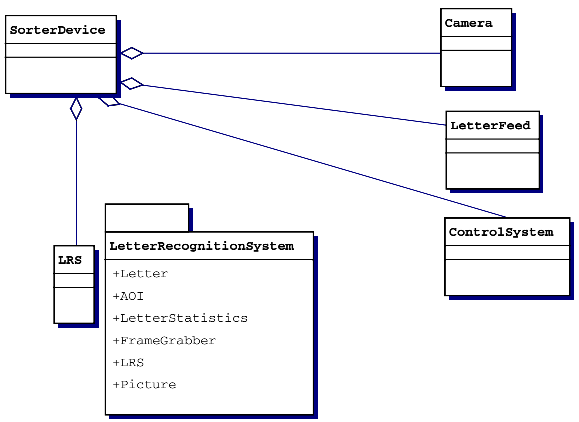






# 6 OOA — Organizing Objects (5)

- Letter Sorter — System overview: UML Diagram



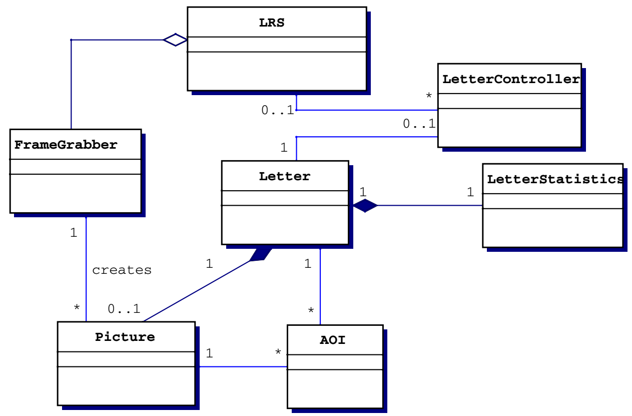






# 6 OOA — Organizing Objects (6)

- Letter Sorter — LetterRecognition: UML Diagram









# 6 OOA — Organizing Objects (7)

- Identify the state of the objects
  - ➤ Attributes (become instancevariables)
  - ➤ Types

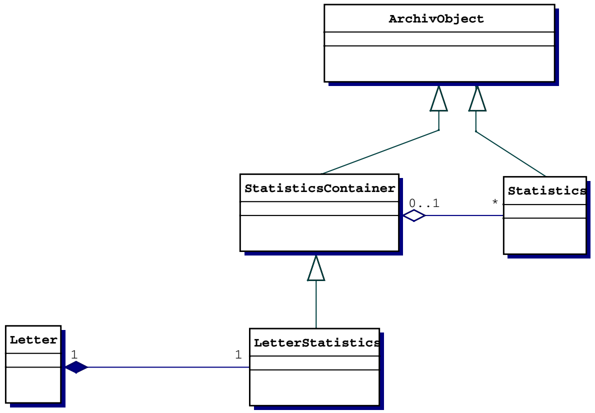
EXAMPLE
➤ Letter:
– Letter ID
– ZIP Code
– Output Tray

- Identify bahavior
  - ➤ Operations / methods
  - ➤ Interaction between objects
- Look for similarities / common things
  - ➤ Basis for inheritance hirarchy
- Look for dependencies between objects
  - ➤ Aggregation
  - ➤ Composition

EXAMPLE
➤ Letter "has" a:
– Picture
– Statistics object

# 6 OOA — Organizing Objects (8)

- Letter Sorter — Statistics: UML Diagram with inheritance relations



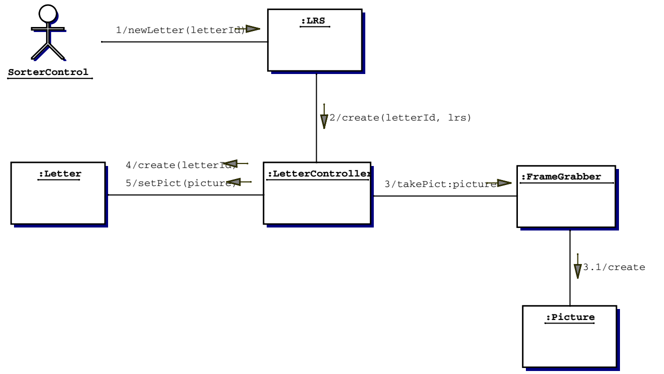






# 7 OOA — Describe Interactions (2)

- Letter Sorter — UML Collaboration Diagram









# 7 OOA — Describe Interactions

- Execution of use cases

EXAMPLE
- ➤ Processing a letter
- ➤ Retrieving statistics

**Determine Operations**

- Essential methods of the objects

EXAMPLE
- ➤ Sorter Control:
  - Throw letter into tray






# 8 OOA — Refine Structure

- Structure use cases
  - ➤ Identify common sqeuences
- Document objects in more detail
  - ➤ Attributes
  - ➤ Operations
  - ➤ Describe roles and responsibilities

## 9 OOA

**Where ends analysis and where begins design?**

- Analysis often extends over 50% of a cycle!
- Design starts after 20 - 30%
- Transition is very fluent
  - ➤ when implementaional aspects cannot be detained any longer






# C.10 Object-oriented Design

- Transformation of the analysis model to an implementable model
- Add aspects of the implementation environment
- Make structural and strategic decisions
  - ➤ Where do we need separate threads
  - ➤ Distribution of the application to several hosts
  - ➤ Which sort of inter-process communication
  - ➤ Database interfaces
  - ➤ Error handling
  - ➤ Garbage collection
- Further refinement of object interaction and interfaces






## 1 Phases

- Class design
  - ➤ Find software classes for the classes of the analysis model
  - ➤ Partition analysis classes
  - ➤ Remove unnecessary analysis classes
  - ➤ Add new classes (e. g. lists or hash tables for management purposes)
  - ! Keep object boundaries
    Analyses ➔ Design ➔ Implementation (Traceablity)
- System design
  - ➤ Non problem-related aspects
    (distribution, concurrency, resource requirements, system interfaces, ...)
- Program design
  - ➤ Programming language
  - ➤ Error handling (Exceptions, return-values)
  - ➤ Performance aspects






# C.11 OOA / OOD - Conclusion

- OOA
  WHAT shall my system do — not HOW
  - ➤ Requirement analysis
  - ➤ Finding and structuring objects
  - ➤ Analyse interactions
- OOD
  HOW shall my system work
  - ➤ Integrate aspects of the execution environment
  - ➤ make strategic decisions
  - ➤ refine object model to an implementable model