

## 3.8 The Unified Software Process

- Precursors:
  - Ericsson-Approach (Jacobson) since 1967
  - Objectory (Jacobson) since 1988
- Principles:
  - use-case driven
  - architecture centric
  - iterative and incremental

## **Use-Case Driven**

- Which user-visible processes are implemented by the system?
- Analysis, design, implementation, and testing driven by use-cases

## **Architecture centric**

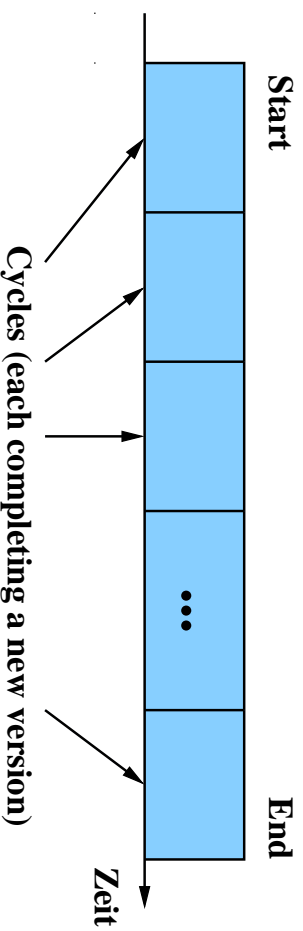
- Architecture developed in parallel to use cases (mutual dependency)

## **Iterative and Incremental**

- eliminate risks first
- checkpoint after each iteration
- on failure of an iteration step, only current extension needs to be reconsidered
- small steps speed up project
- easy stepwise identification of the requirements

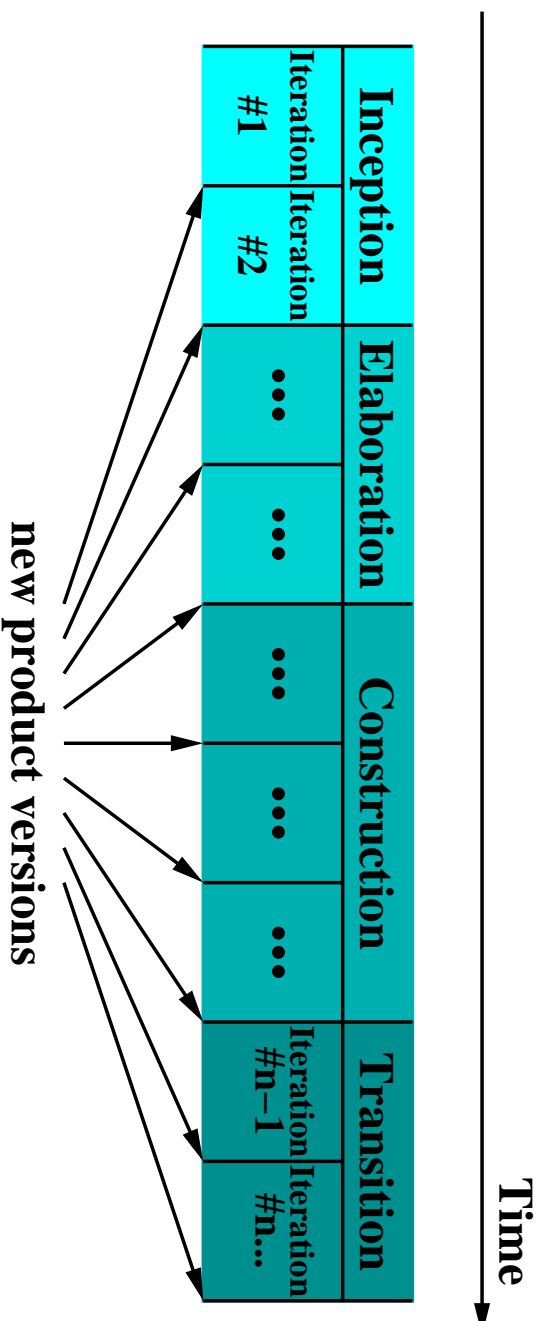
### 3.8.1 Structure of the Unified Process

- sequence of cycles
- after each cycle: product release with code, manuals, UML models, and test cases

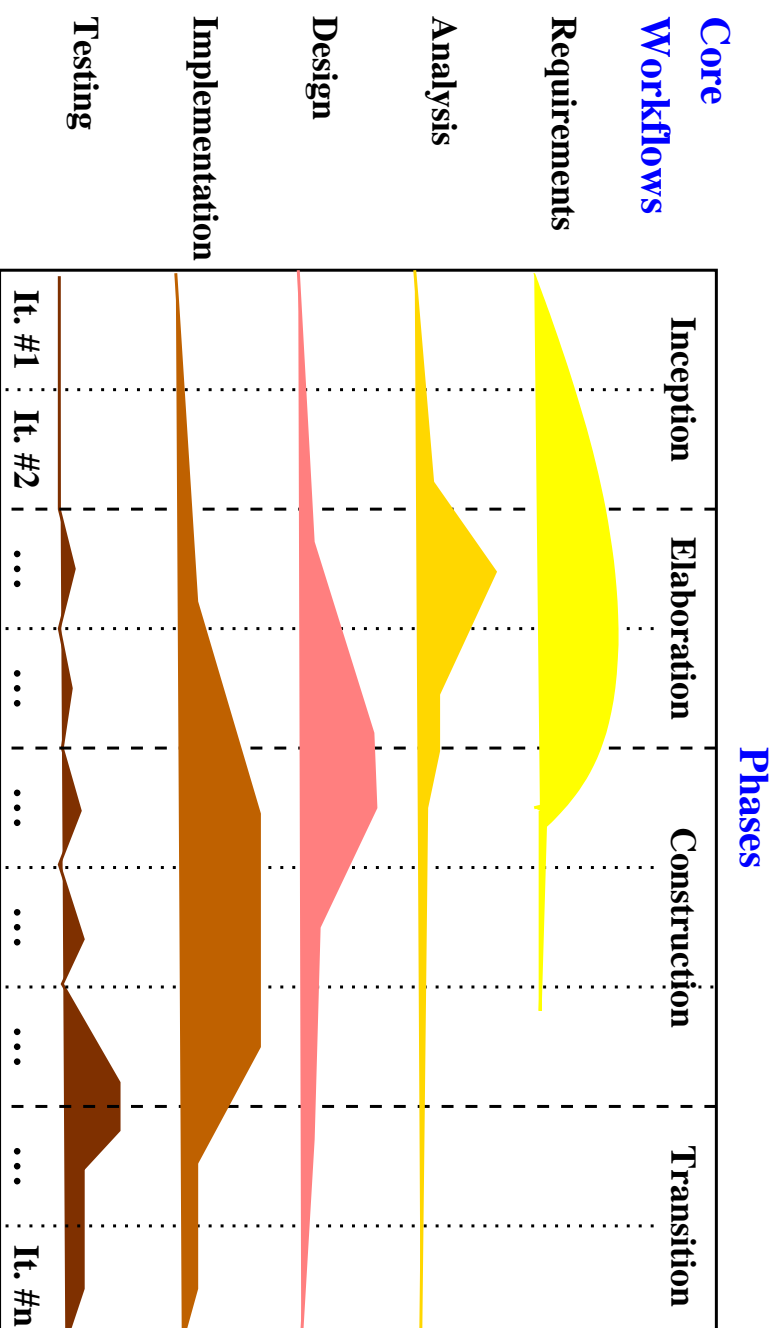


- cycle consists of 4 phases: Inception, Elaboration, Construction, Transition
- each phase consists of iterations

# Cycle



# Main-Workflows and Phases



- each phase ends with a **mile stone**
- each phase processes all workflows (with varying intensity)

## Inception Phase

- **GOAL:** rough vision of the product
- functionality of system from users' perspective most important use cases (**stakeholder needs**)
- preliminary sketch of suitable architecture
- project plan and cost
- identify most important risks (with priorities)
- plan elaboration phase

## Elaboration Phase

- specify (most) use cases in detail
- design architecture
- prototype (proof-of-concept for architecture)
- implement most important use cases
- result: initial architecture
- plan activities and resources for remaining project
- use cases and architecture stable?
- risk management?

## Construction Phase

- implement system
- high resource needs
- small architectural changes
- **GOAL:** system ready for customer (small errors acceptable)

## **Transition Phase**

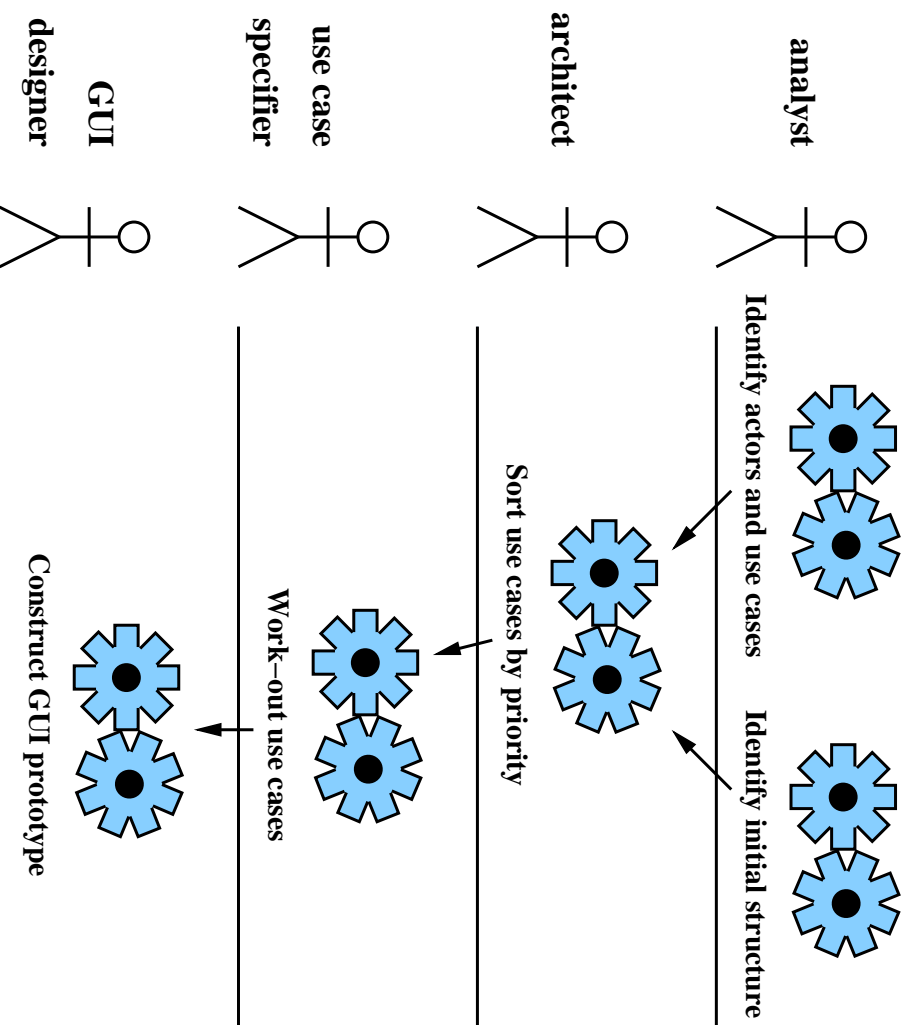
- deliver beta-version to customer
- address problems (immediately or in next release)
- train customer
- hotline

## Models of the Unified Process

- use-case model
- analysis model
- design model
- deployment model
- implementation model
- test model
- based on UML
- models yield consistent views
  - diagrams of a model refine diagrams of superior model  
(trace dependency)
  - traceability of refinement alleviates understanding

# Software Development Process Model

- workflows, workers, activities, artifacts



## Adaptation of the Unified Process

- depending on environment (organization, application area, life cycle, programming language, tool, ...)
- UP specified using OO → activities, diagrams, etc can be specialized
- one process per organization
- components and workers exchangeable

## **Tool Support**

- automating the process
- tools are part of the software development process
- required to keep models current and consistent

## 3.8.2 UP is Use-Case driven

- each activity determined by use cases
- advantage: systematic and intuitive incorporation of functional requirements

### Use-Case Model:

- from use-case diagrams

### Analysis Model:

- derive classes from use case descriptions
- collaboration and sequence diagrams from use cases
- class diagram with **conceptual classes**
- identify collaborating classes on a per use-case basis
- assign responsibilities to classes

## **Design Model**

- architecture based on central use cases
- hierarchy of components and subsystems with interfaces
- determines structure of the implementation
- associations, generalization, dependencies throughout the hierarchy

## **Implementation Model**

- implementation of the design classes
- use cases determine the sequence

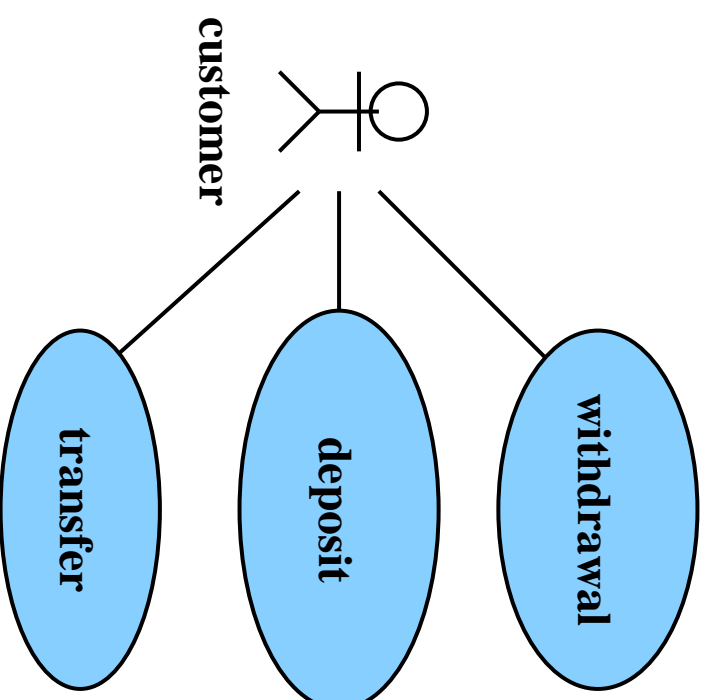
## **Test Model**

- early development of black-box test cases possible (from use cases)
- sequence of integration determines by use cases
- glass-box testing from use cases

## Identify Use Cases

- **Actors:** human users, external hard- and software system
- one person can take on different roles (actors)
- many persons can adopt the same role
- use cases come about by viewing the use of the system from different perspectives
- use case (often) describes sequence of actions
- join use cases with similar paths (variants)
- specify non-functional requirements in use cases (runtime, accuracy, ...)

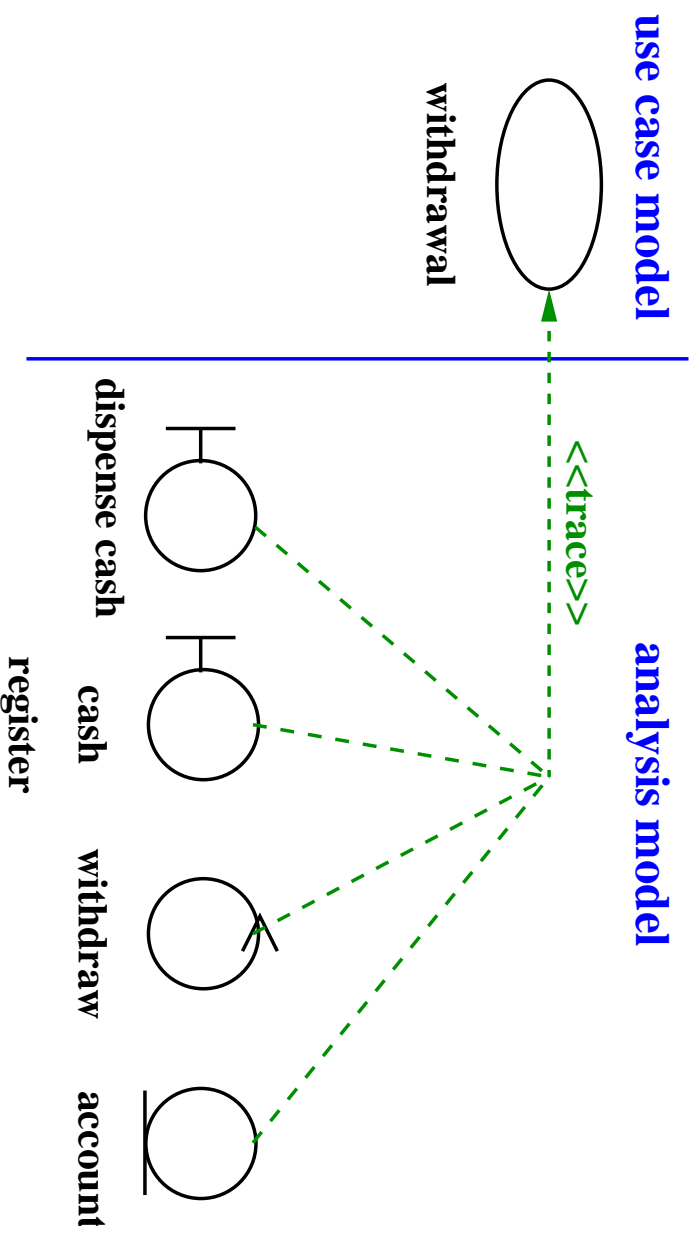
## Example: Use Case Diagram for telling machine



## From Use Case Diagram to Analysis Model

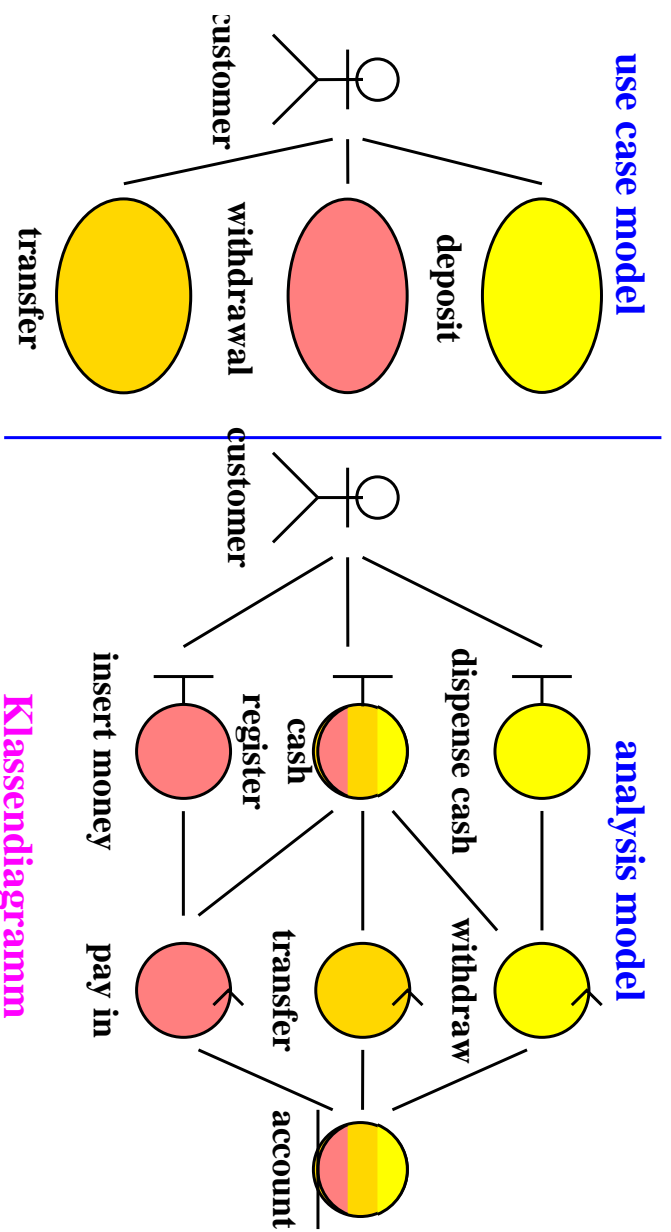
- determine (conceptual) classes from use case descriptions
- Stereotypes
  - **control class**
    - \* coordination and control of other objects
    - \* transactions
    - \* model behavior specific to one or several use cases
  - **boundary class**
    - \* communication between environments (actors) and inner workings
  - **entity class**
    - \* information stored by the system and associated behavior
    - \* generic, reused, often persistent characteristics
    - \* entity objects participate in several use cases (and survive)

# Example: cash dispenser



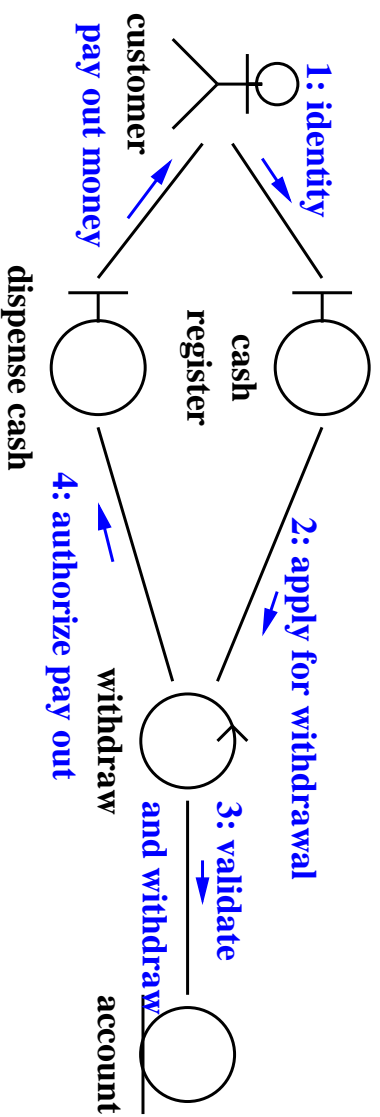
analysis classes for use case "withdrawal"

# Example: cash dispenser



association of analysis classes to use cases

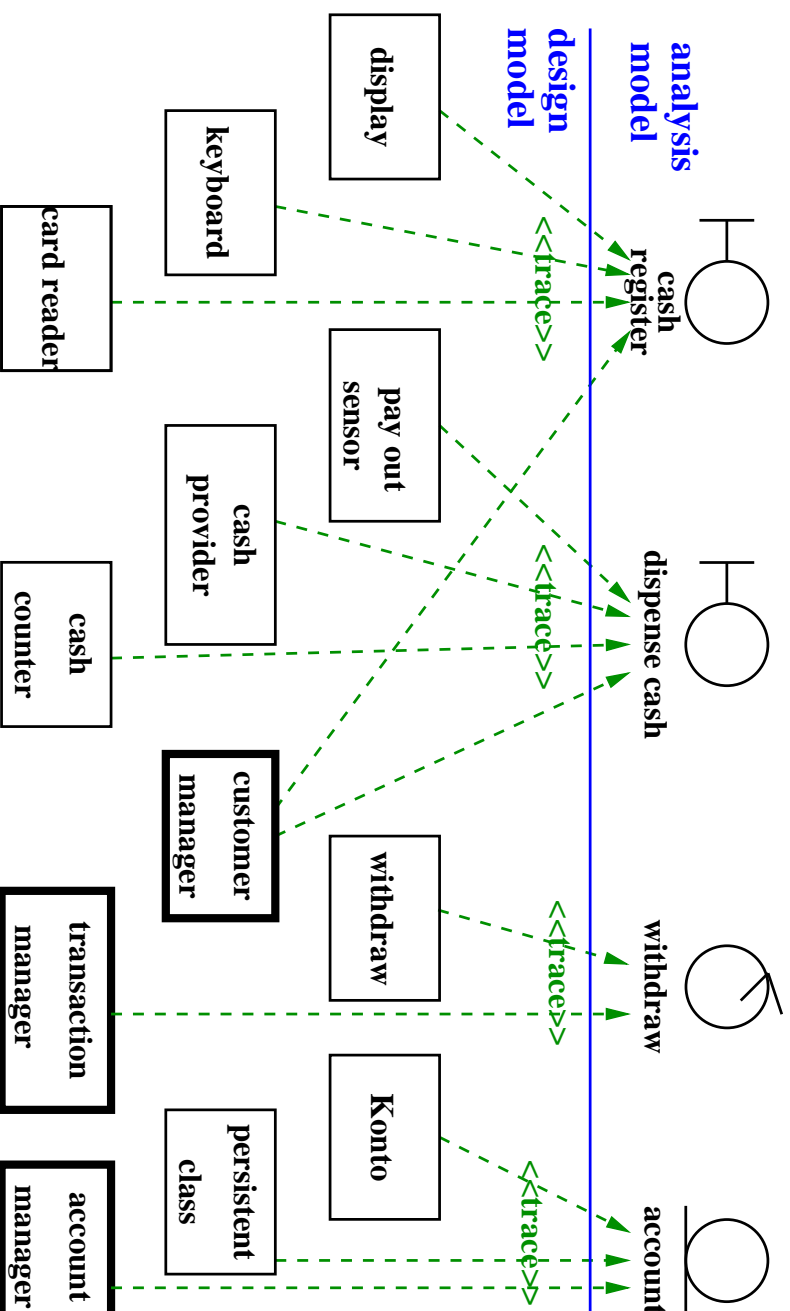
## Collaboration Diagram



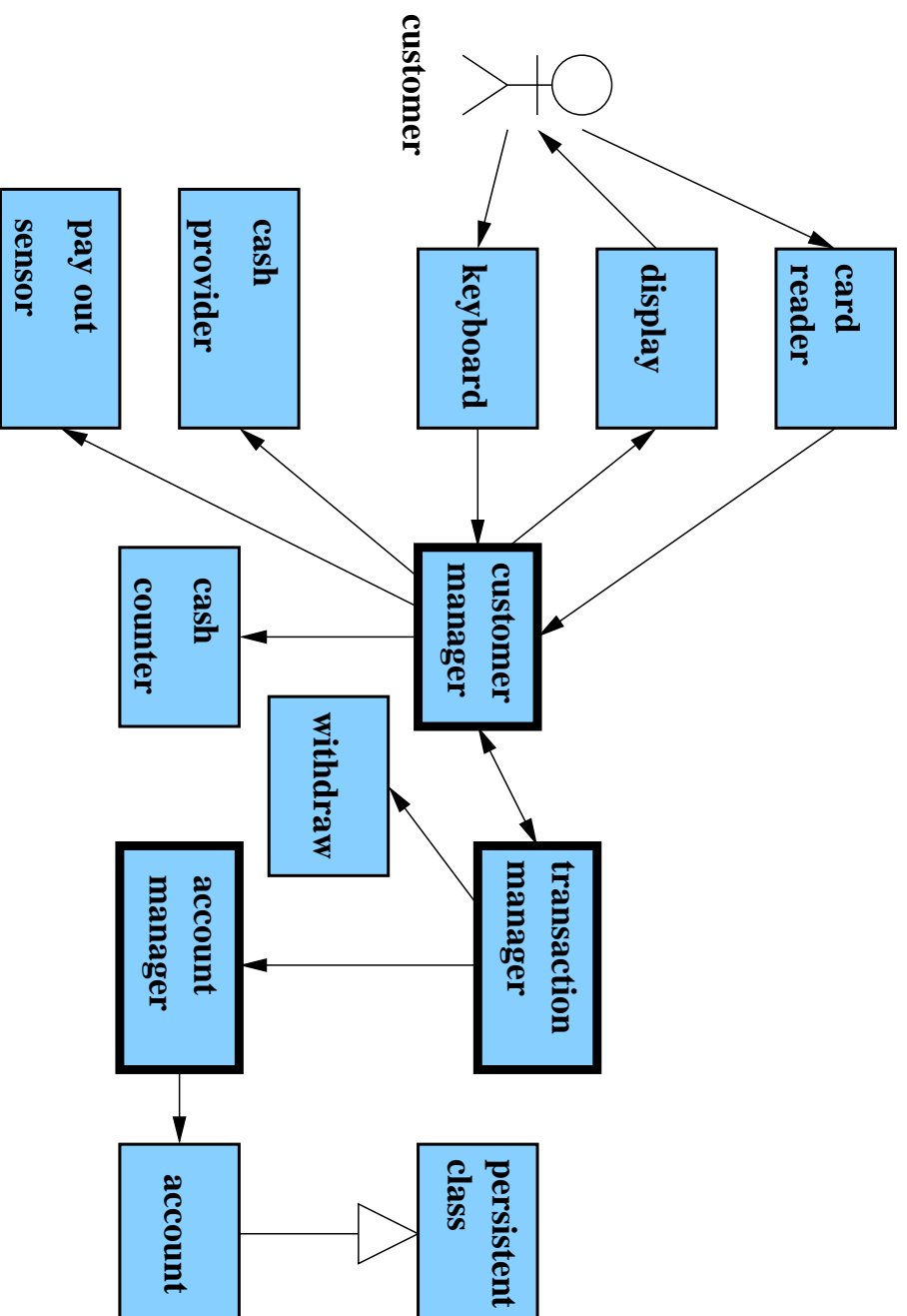
- alternatively: sequence diagram
- messages correspond to method invocation (cf. class diagram)
- responsibility of class determined from its roles

## From Analysis Model to Design Model

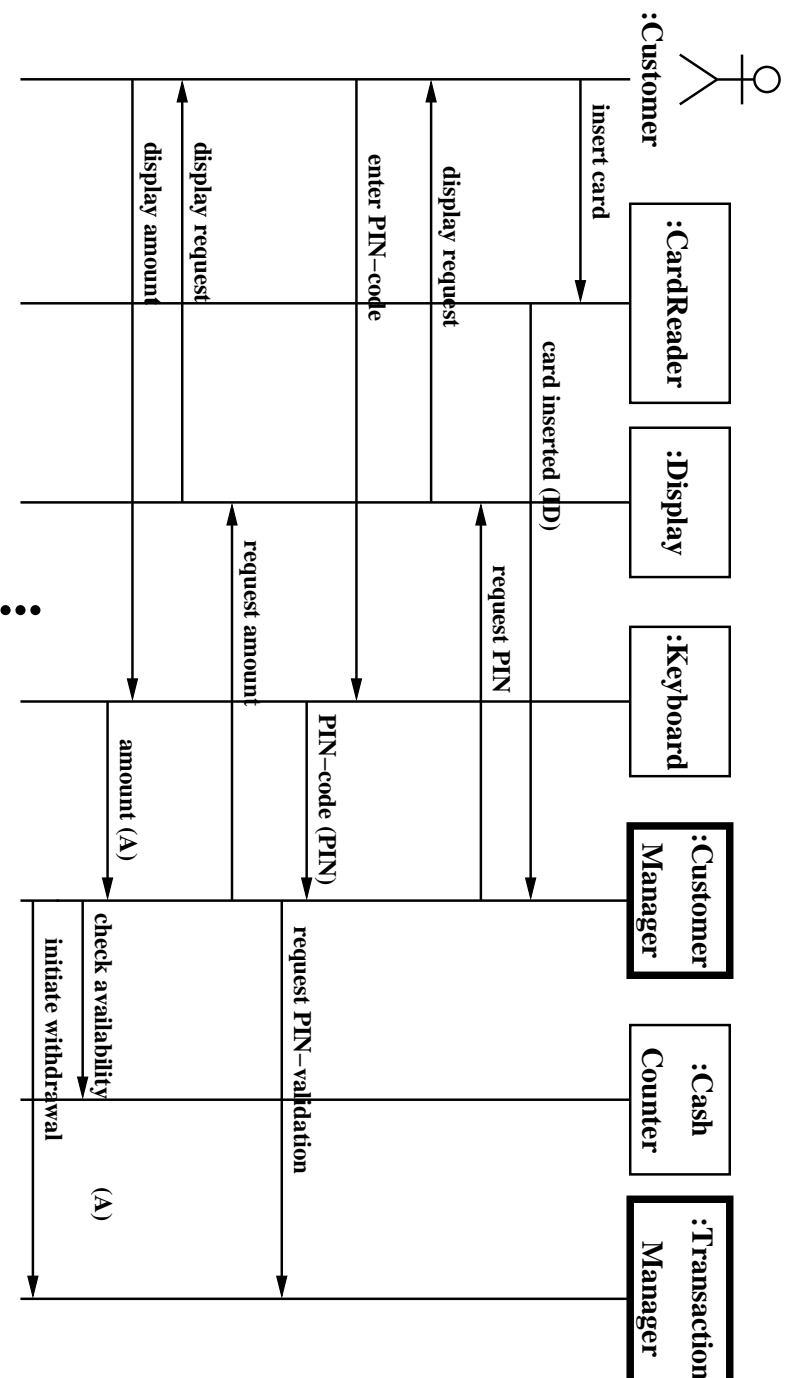
- refinement including middleware, GUI, DBMS, legacy systems
- conceptual classes → physical classes (often more than one)
- structure remains the same



# Class Diagram in Design Model



# Sequence Diagram in Design Model

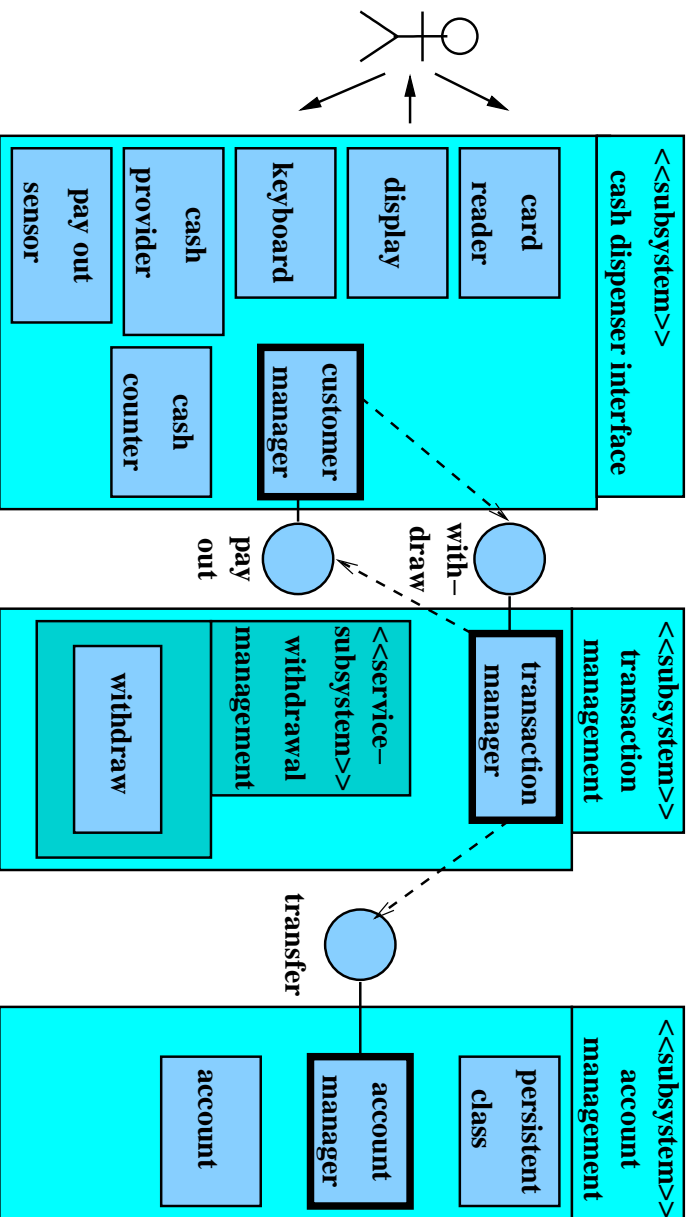


collaboration of design classes

## Subsystems

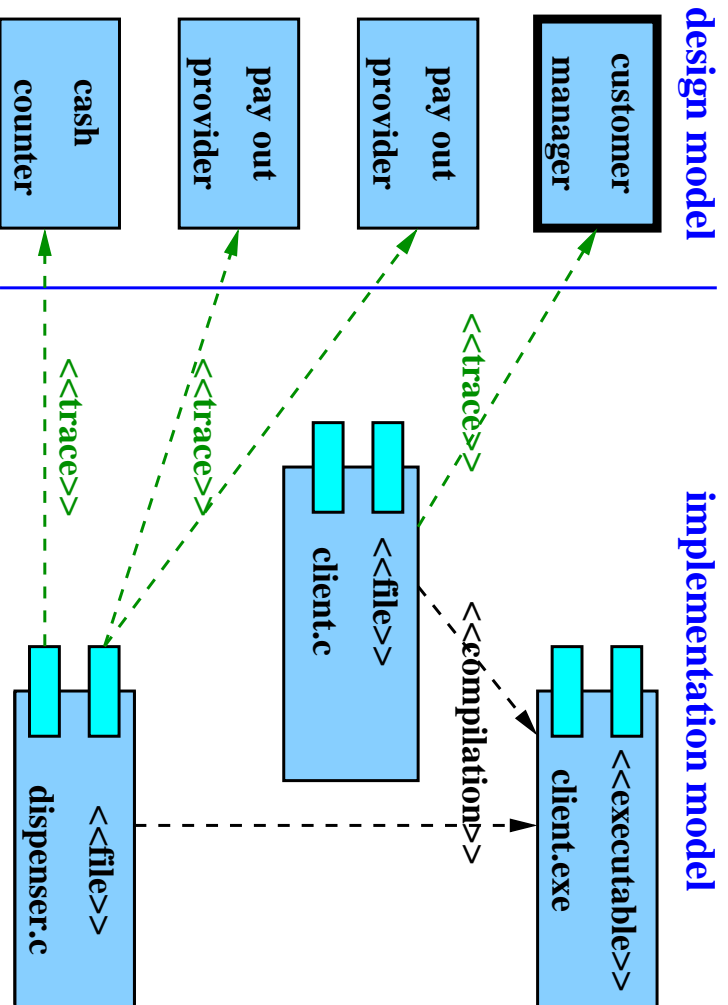
- combine semantically related classes to **subsystem**
- approach
  - bottom-up: starting from classes
  - top-down:
    - \* first: decomposition into unspecific subsystems
    - \* then: distribute classes
- **deployment model** determines assignment of components to computing nodes (may depend on configuration)

# Example Decomposition in Subsystems

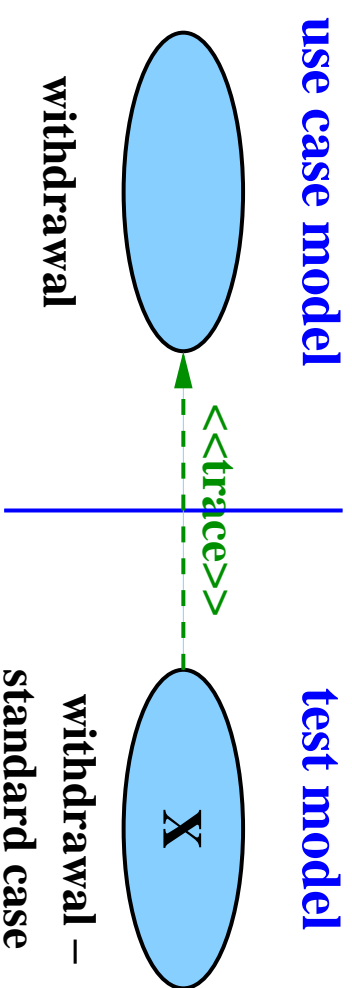


## From Design Model to Implementation Model

- implement all classes in programming language
- implementation model consists of executable components, databases, files, ...



## Testing of Use Cases



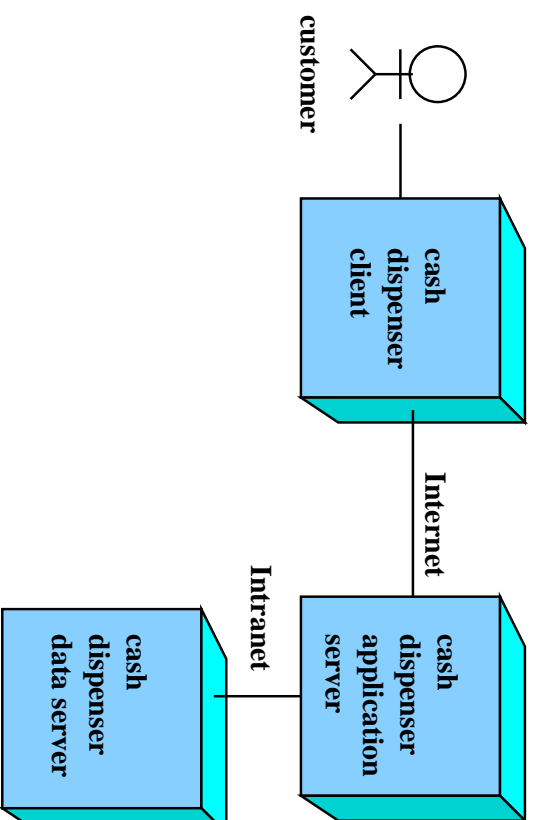
black-box text cases can be generated after completion of use case model

### 3.8.3 UP is architecture centric

- development **not** solely driven by use cases:  
start with rough architectural sketch, **independent of application**
  - architectural patterns (client-server, broker, 3-tier, n-tier)
  - layers
  - Middleware
  - legacy systems
  - DBMS
  - GUI system
- tailor to application
- mainly in elaboration phase

- next: application specific part of development starting from stakeholder needs
  - adapt architecture to application
- finally: consideration of remaining use cases from resulting stable architecture
- use cases biased by requirements and architecture
- negotiate with customer: modified use cases may lead to cheaper implementation
- basic architecture constant throughout the life of the system

# Deployment Model of Basic Architecture



## Assignment of Active Classes to Computing Nodes:

