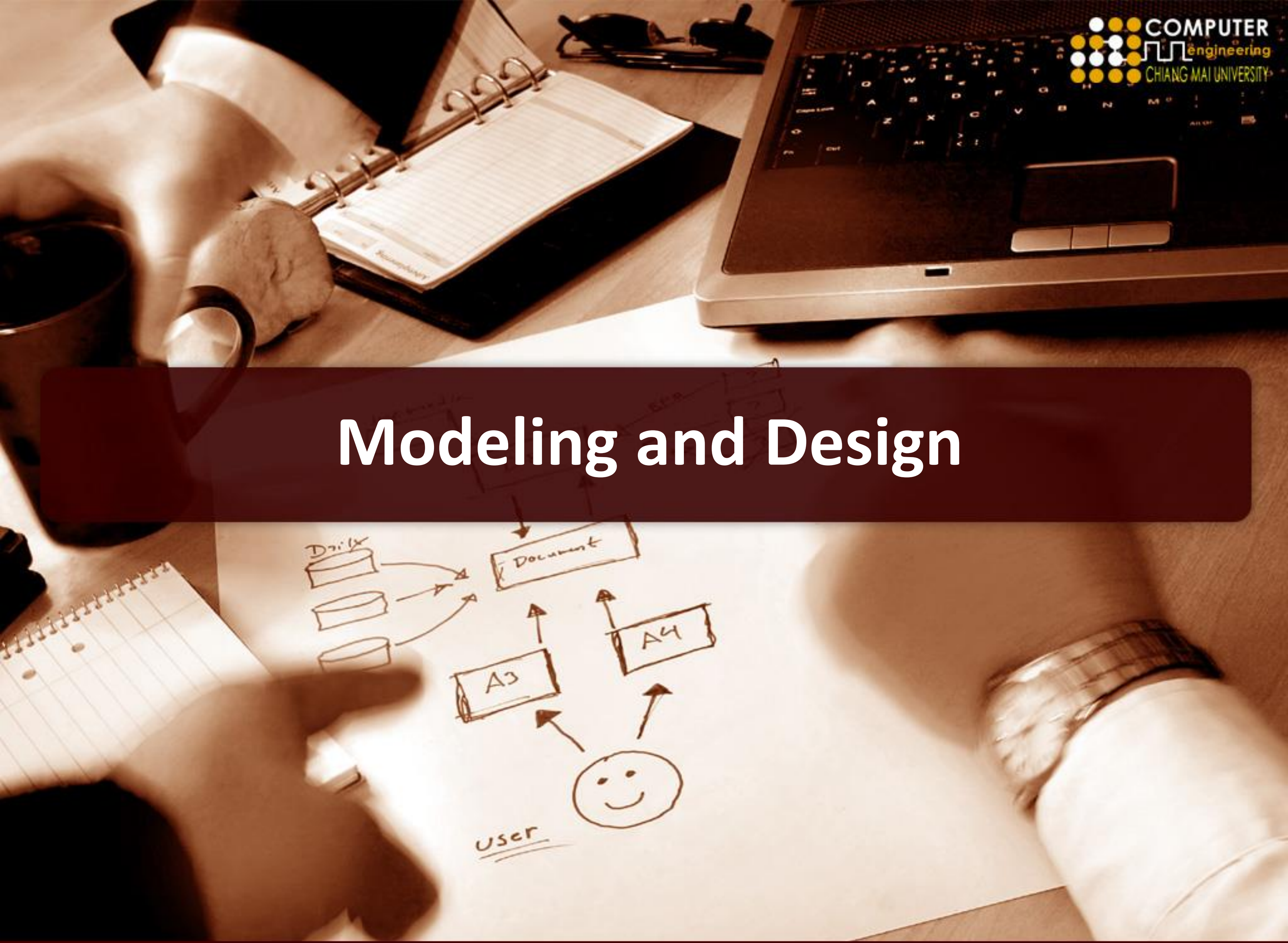


COMPUTER engineering CHIANG MAI UNIVERSITY

Modeling and Design

Class Review

- Software specification is the process of determining system services, its operation constraints and the development constraints. The process is called Requirement Engineering (RE).
- RE consists of four activities: Requirements elicitation and analysis, Requirements specification, Requirements validation and Requirement management.
- Main types of requirements include user and system requirements, and functional and non-functional requirements.

Modeling

- In Requirements Engineering, modeling of the existing system are created to guide the requirements for the new system.
- The models of the new system are also created. Stakeholders see the models as the proposal of the new system. Engineers use it to discuss design proposals.

Design

- The design process begins with architectural design to identify the sub-systems and the framework for sub-system control and communication.
- The output of this design process is a description of the software architecture.

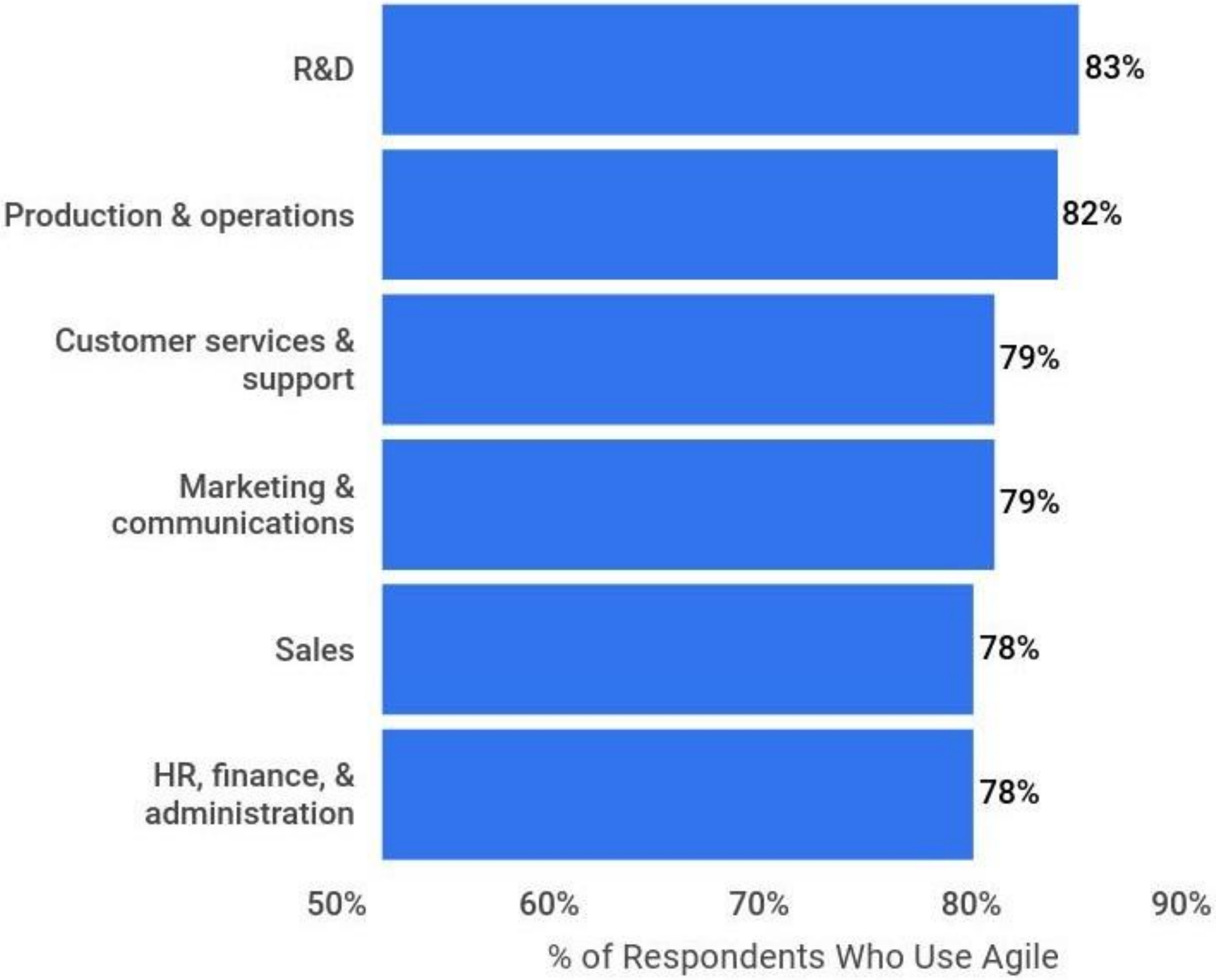
Modeling and Design

- In real life, Requirements Engineering may be overlapped with Designing.

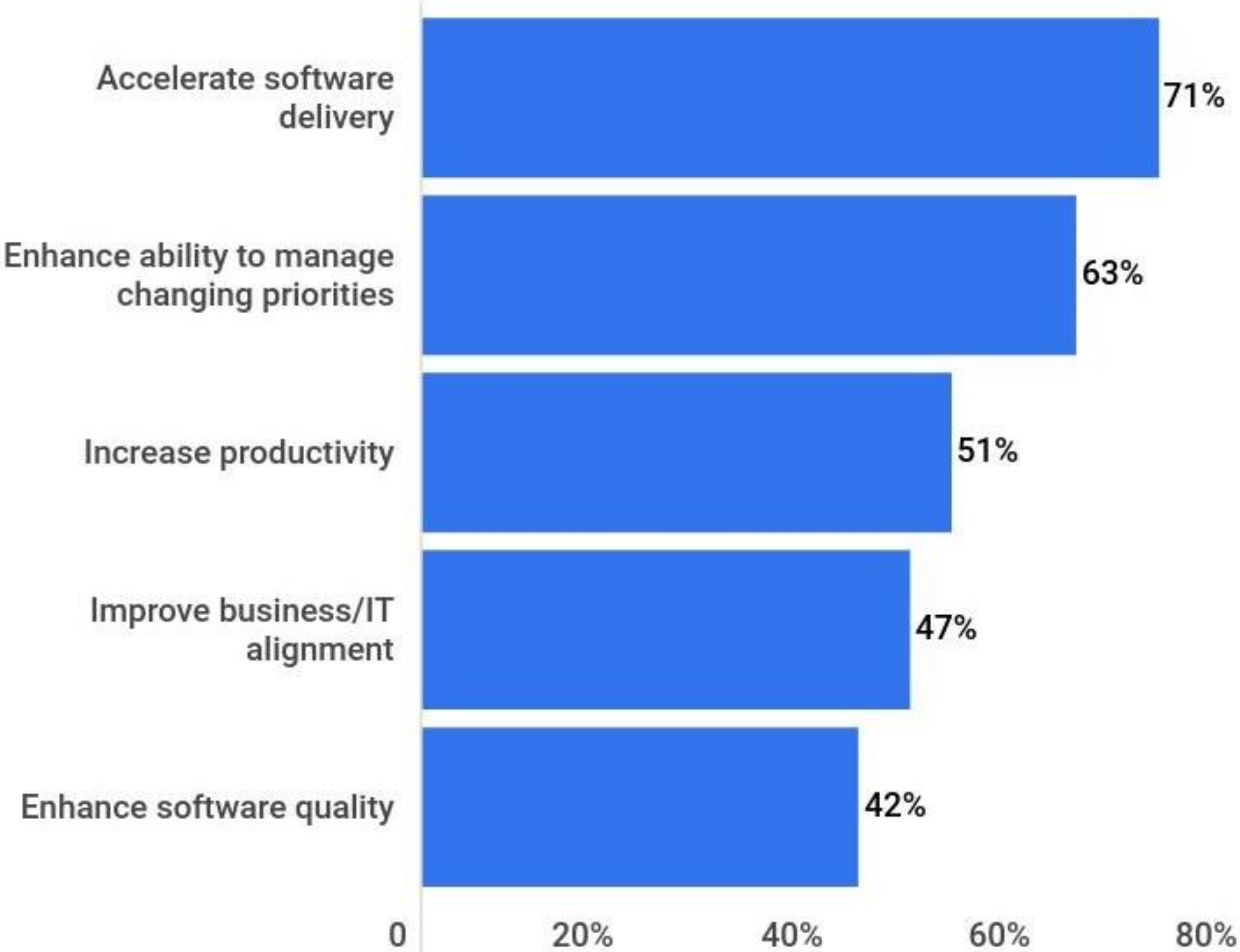
- Agile Modeling (AM), a supplement to other Agile development methodologies, is "a practice-based methodology for effective modeling and documentation of software-based systems" – Simplilearn.com, Wikipedia.org

Agile Statistics as per 2023

- Research Summary – Zippia.com
  - Agile are increasingly used for software management in many companies.
- “At least 71% of U.S. companies are now using Agile.”
- Agile projects have a 64% success rate, whereas waterfall only have a 49%. So, Agile is nearly 1.5X more successful.
- Scrum is considered the most popular Agile framework. 61% of respondents (76 countries) reported that they use it.


R&D
Production & operations
Customer services & support
Marketing & communications
Sales
HR, finance, & administration
83%
82%
79%
79%
78%
78%
50%
60%
70%
80%
90%
% of Respondents Who Use Agile

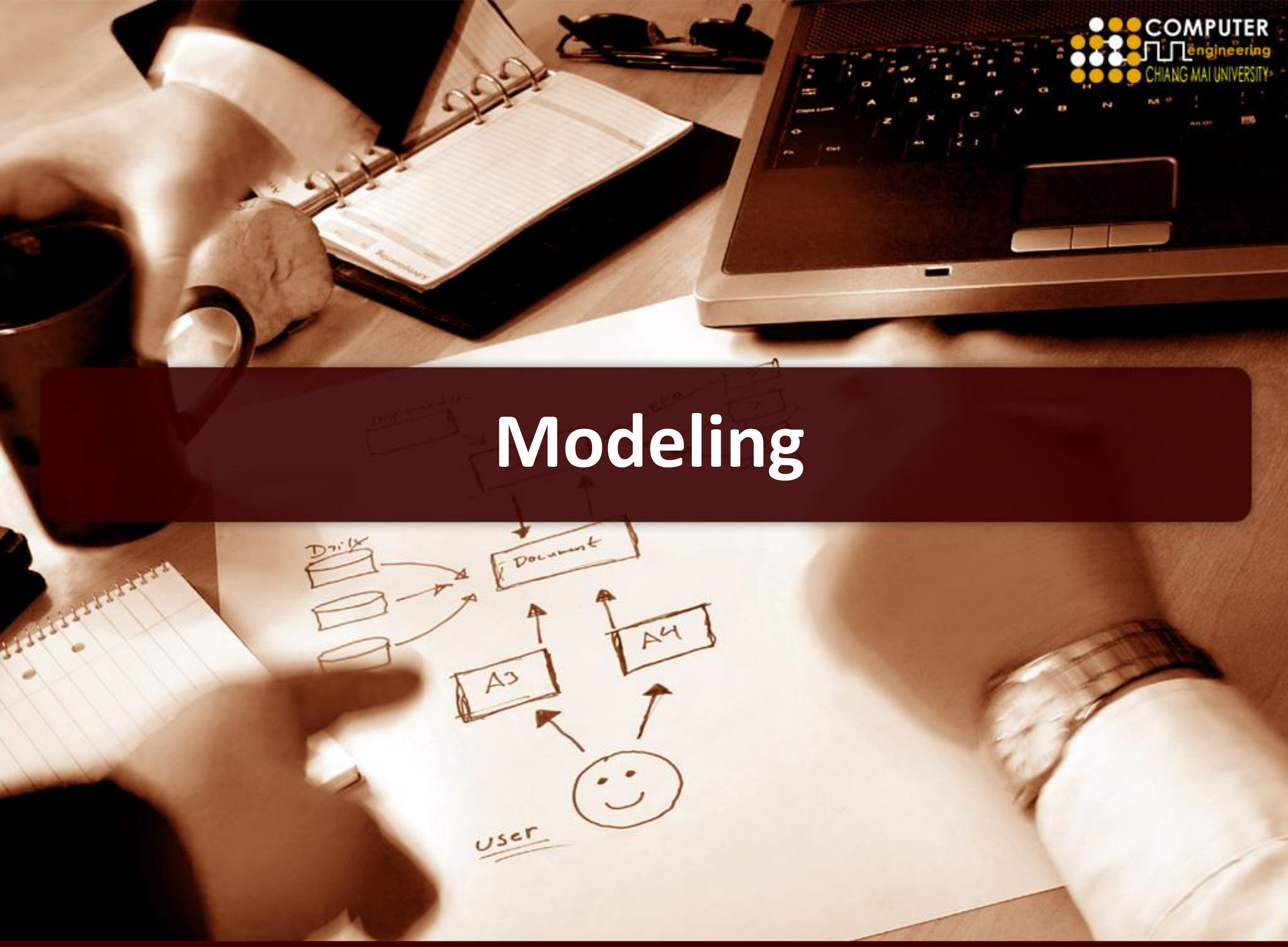
TOP FIVE REASONS FOR ADOPTING AGILE



AM: Core practices

- Document continuously throughout the life-cycle.
- Document as late as possible.
- Executable specifications in the form of executable "customer tests", instead of "static" documentation.
- Single-source information (models, documentation, software) is stored in one place only for "correctness" of versions / information.

Modeling



Modeling

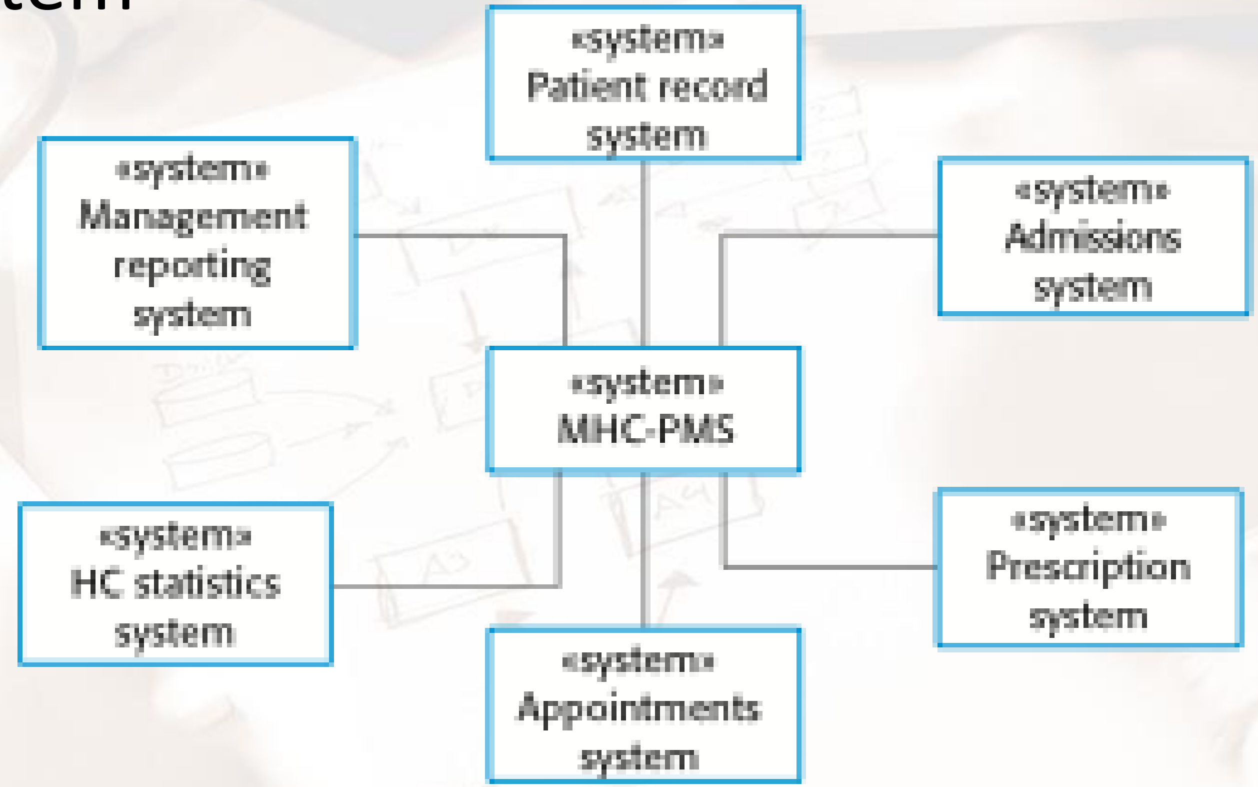
- Modeling is a mean of representing a system using some kind of graphical notation such as the Unified Modeling Language (UML).
- Modelling helps clarify the functionality of the system and ,therefore, is used to communicate with customers.
- Types of Models:
  - Context models
  - Interaction models
  - Structural models
  - Behavioral models

Models

- **Context models**
  - Illustrate system boundaries or the operational context of a system or scope.
- **Interaction models:**
  - User interaction model helps identifying user requirements.
  - Systems interaction model highlights communication issues.
  - Component interaction model helps in understanding a system architecture.

Context Model: MHC PMS

- Mental Healthcare Patient Management System


«system» Patient record system
«system» Management reporting system
«system» Admissions system
«system» MHC-PMS
«system» HC statistics system
«system» Prescription system
«system» Appointments system

Models

- **Structural models**
  - When designing the system architecture, use structureal models to display the organization of a system in terms of the components and their relationships.
- **Behavioral models**
  - Display the dynamic behavior of a system in execution by showing what happens and how a system responses to a stimulus.
- Use case diagrams and Sequence diagrams may be used.

Unified Modelling Language (UML)

Unified Modelling Language (UML)

- UML (Unified Modeling Language) is a standardized general-purpose modeling language in the field of software engineering
- Developed by Object Management Group (OMG)
- UML 2.5 has 14 types of diagrams divided into three categories.
- The latest version is 2.5.1.

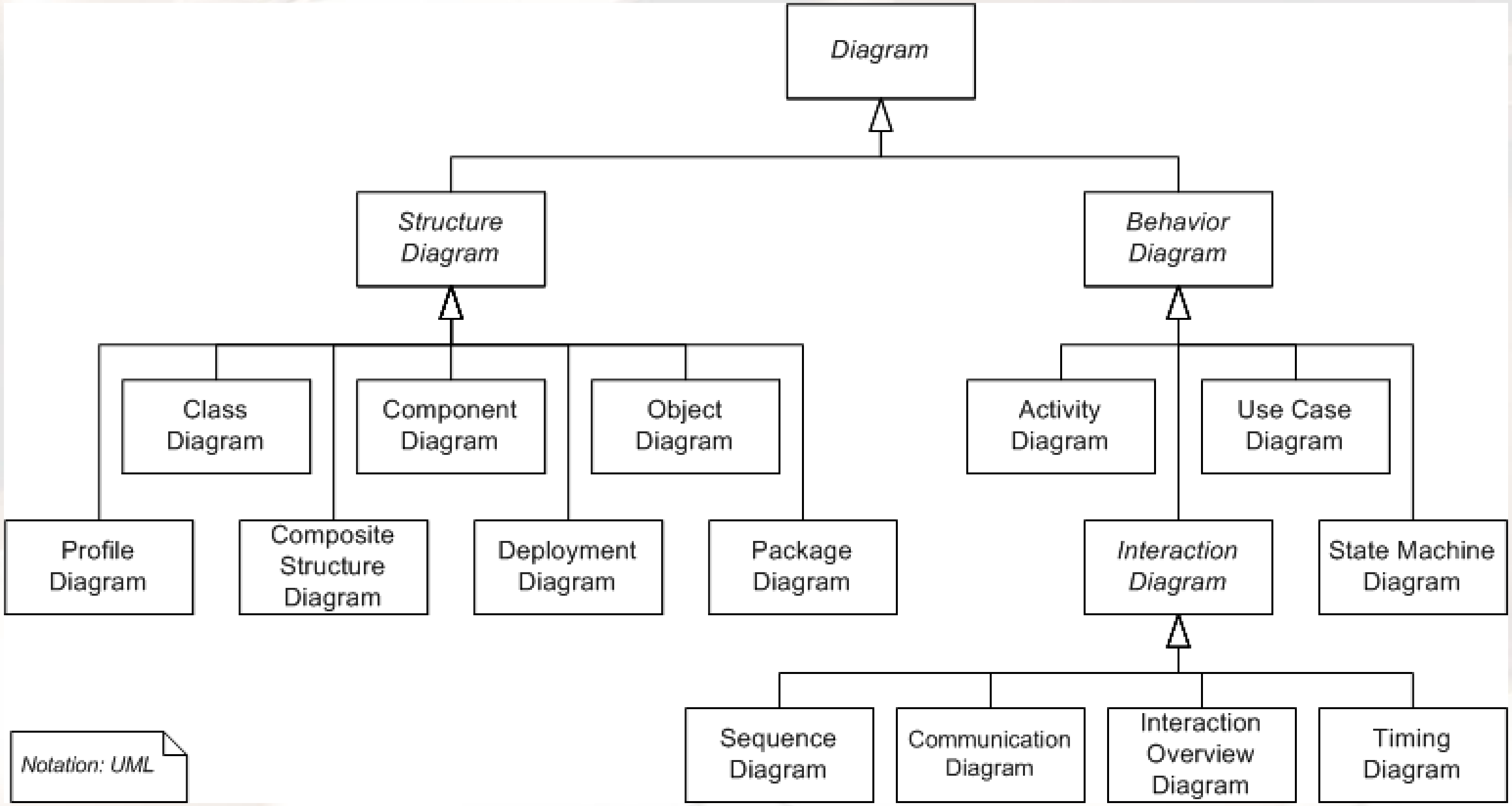
14 UML Diagrams

- Structure Diagrams
  - Class Diagram
  - Component Diagram
  - Composite Structure Diagram
  - Deployment Diagram
  - Object Diagram
  - Package Diagram
  - Profile Diagram*

- Behavior Diagrams
  - Activity Diagram
  - State Machine Diagram
  - Use Case Diagram
- Interaction Diagrams
  - Communication Diagram
  - Interaction Overview Diagram
  - Sequence Diagram
  - Timing Diagram

(**Source**: Wikipedia)

14 UML Diagrams (*Contd.*)



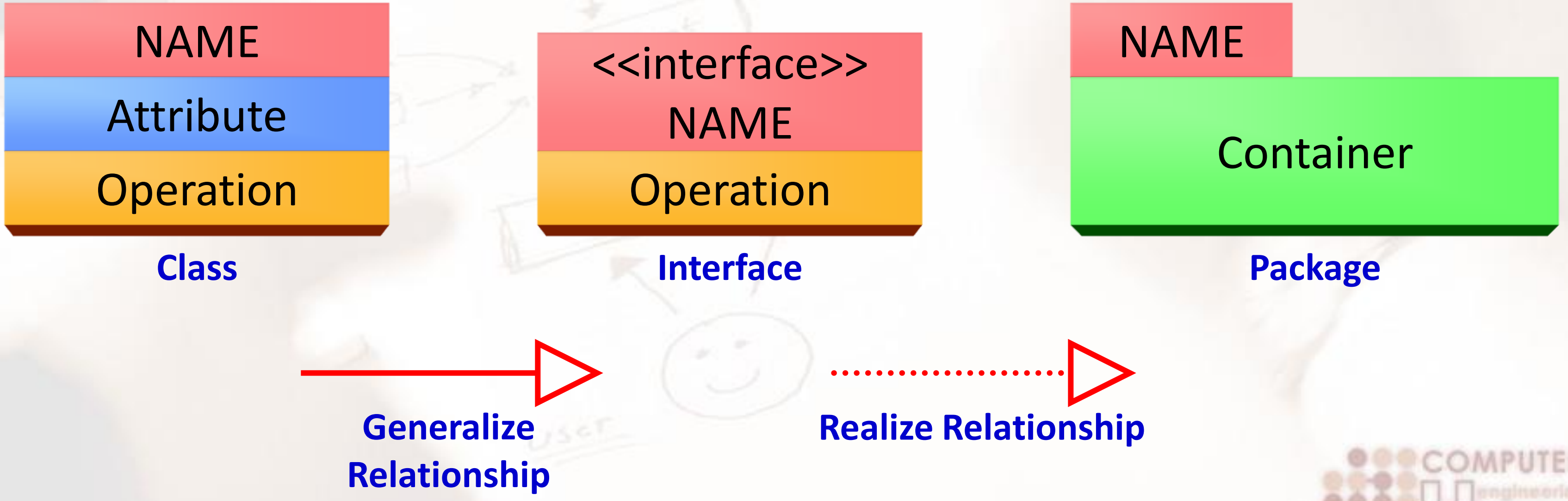
Structure Diagrams

- Emphasizes the static structure of the system using objects, attributes, operations and relationships

Structure Diagrams: Class Diagram

- Shows the system’s classes, their attributes, and the relationships among the classes

- **Application:** General OO programming



NAME

Attribute

Operation

Class

<<interface>>

NAME

Operation

Interface

NAME

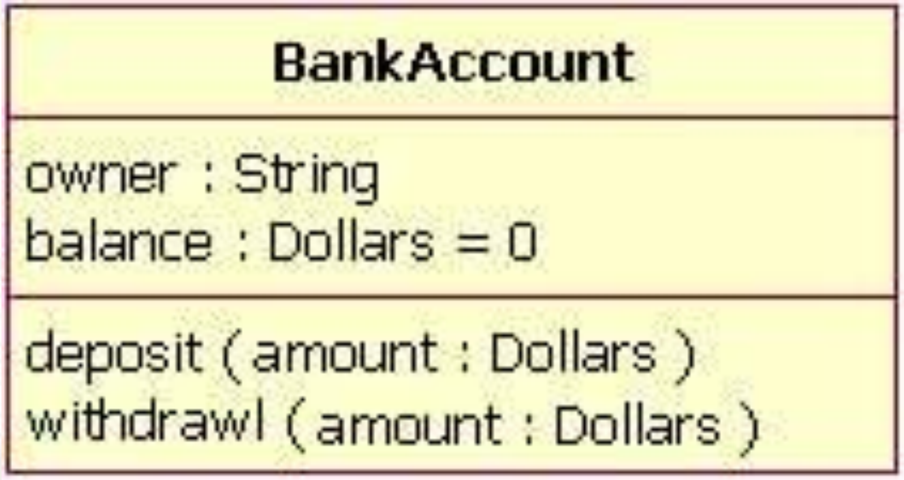
Container

Package

Generalize Relationship

Realize Relationship

Structure Diagrams: Class Diagram

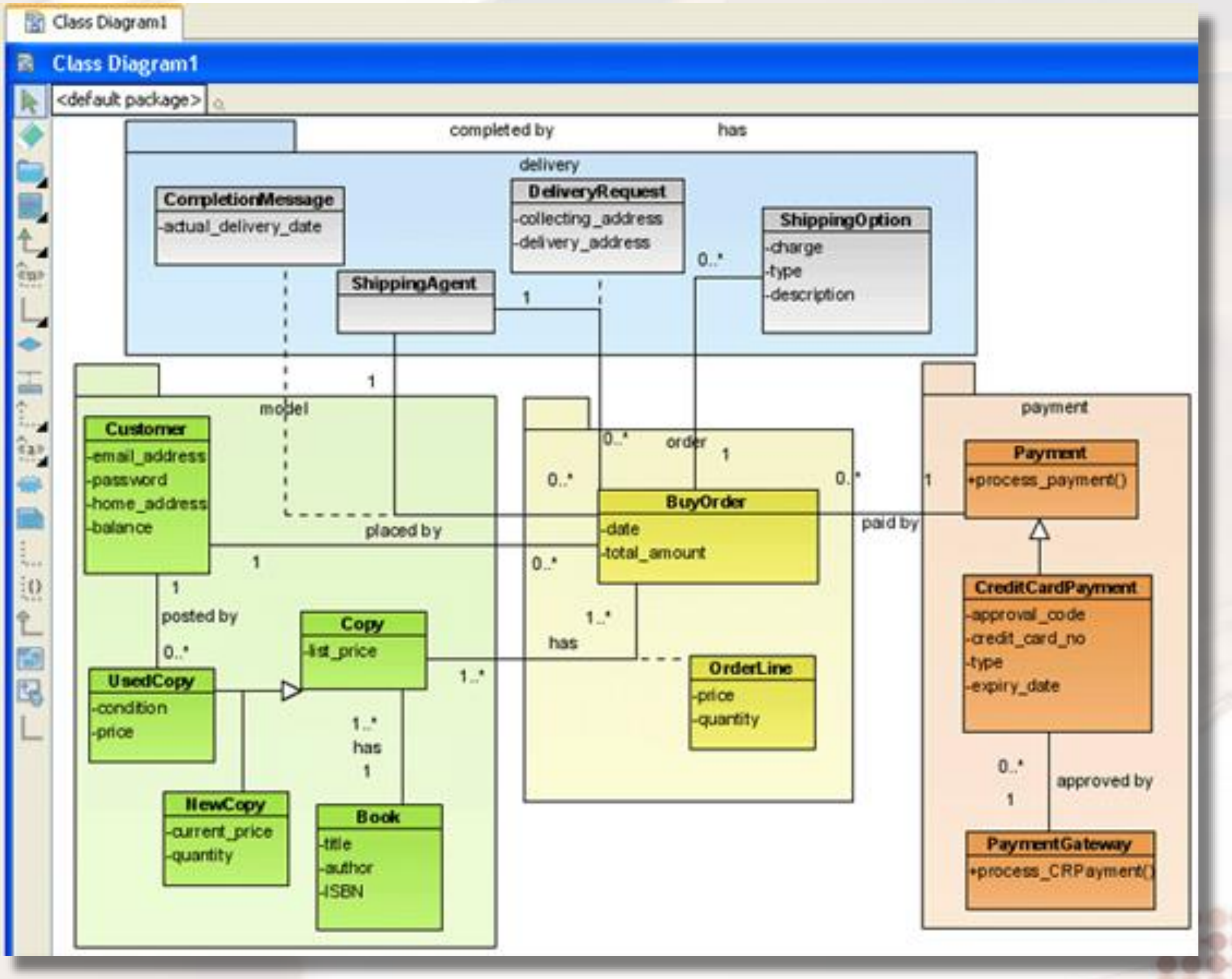


BankAccount

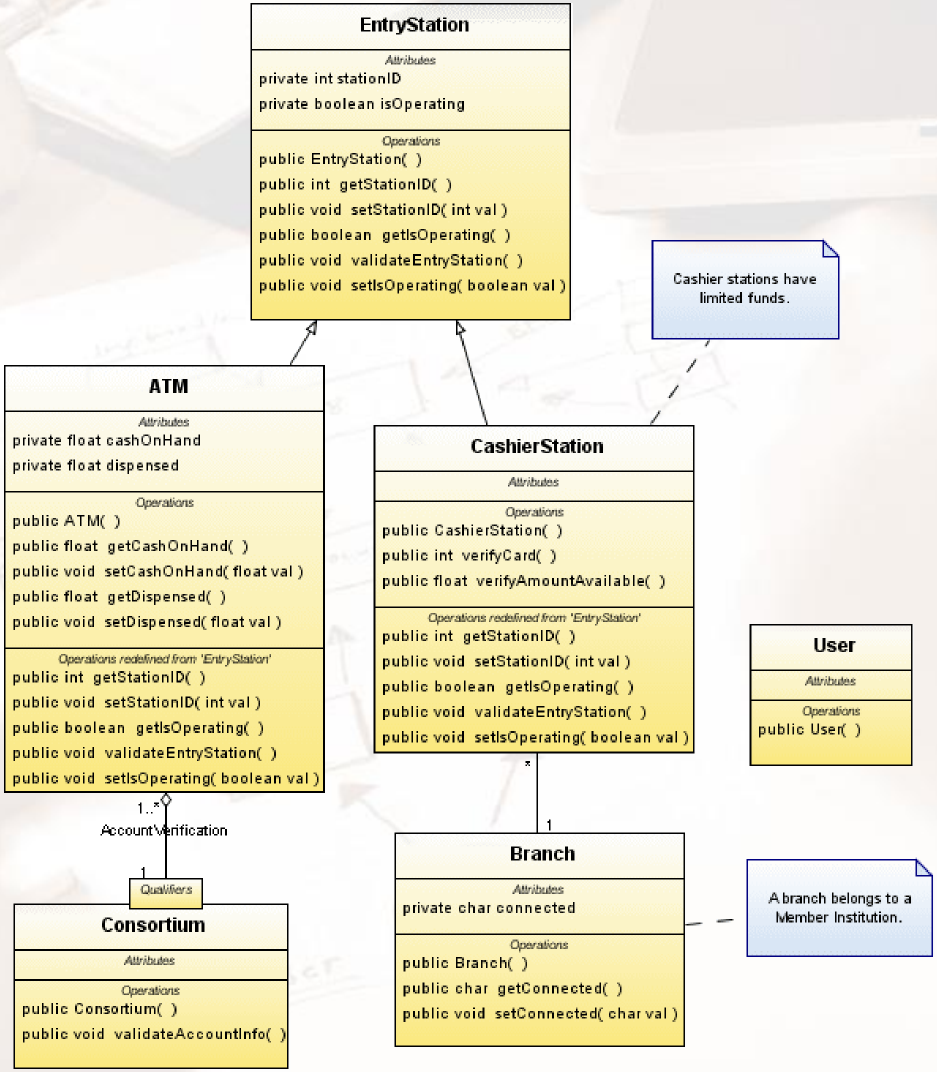
owner : String
balance : Dollars = 0

deposit (amount : Dollars)
withdrawl (amount : Dollars)

Structure Diagrams: Class Diagram



Structure Diagrams: Class Diagram



EntryStation

Attributes
private int stationID
private boolean isOperating

Operations
public EntryStation( )
public int getStationID( )
public void setStationID(int val)
public boolean getIsOperating( )
public void validateEntryStation( )
public void setIsOperating(boolean val)

Cashier stations have limited funds.

ATM

Attributes
private float cashOnHand
private float dispensed

Operations
public ATM( )
public float getCashOnHand( )
public void setCashOnHand(float val)
public float getDispensed( )
public void setDispensed(float val)

Operations redefined from 'EntryStation'
public int getStationID( )
public void setStationID(int val)
public boolean getIsOperating( )
public void validateEntryStation( )
public void setIsOperating(boolean val)

CashierStation

Attributes

Operations
public CashierStation( )
public int verifyCard( )
public float verifyAmountAvailable( )

Operations redefined from 'EntryStation'
public int getStationID( )
public void setStationID(int val)
public boolean getIsOperating( )
public void validateEntryStation( )
public void setIsOperating(boolean val)

User

Attributes

Operations
public User( )

Consortium

Qualifiers

Attributes

Operations
public Consortium( )
public void validateAccountInfo( )

Branch

Attributes
private char connected

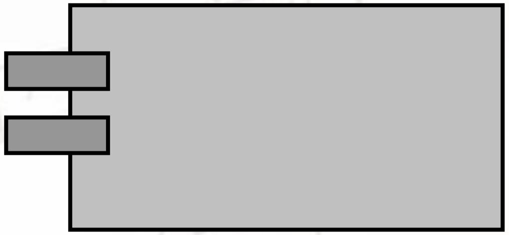
Operations
public Branch( )
public char getConnected( )
public void setConnected(char val)

A branch belongs to a Member Institution.

Structure Diagrams: Component Diagram

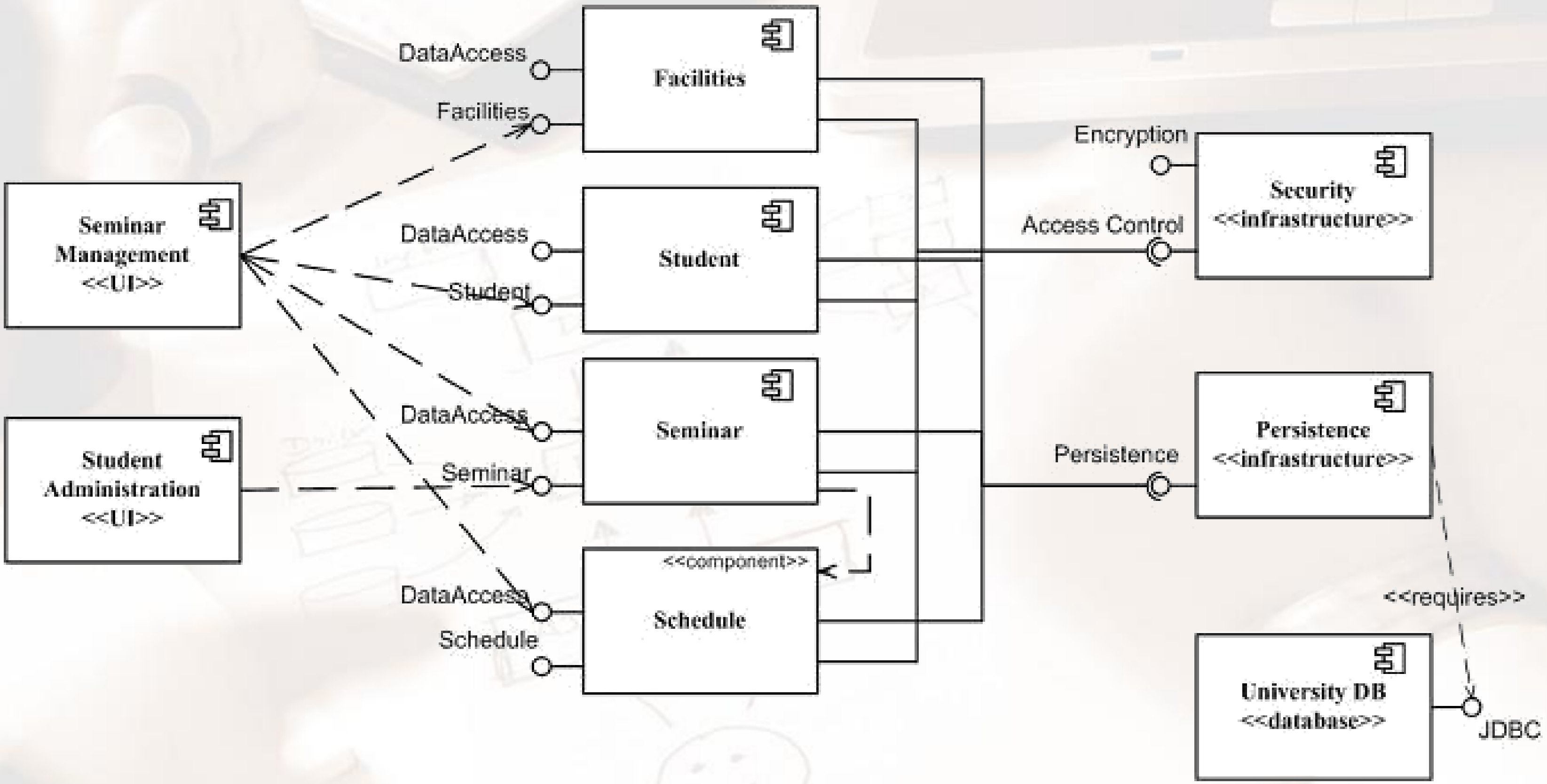
- Depicts how a software system is split up into components
- Shows the dependencies among these components

- **Application:** Any systems which can be modularized

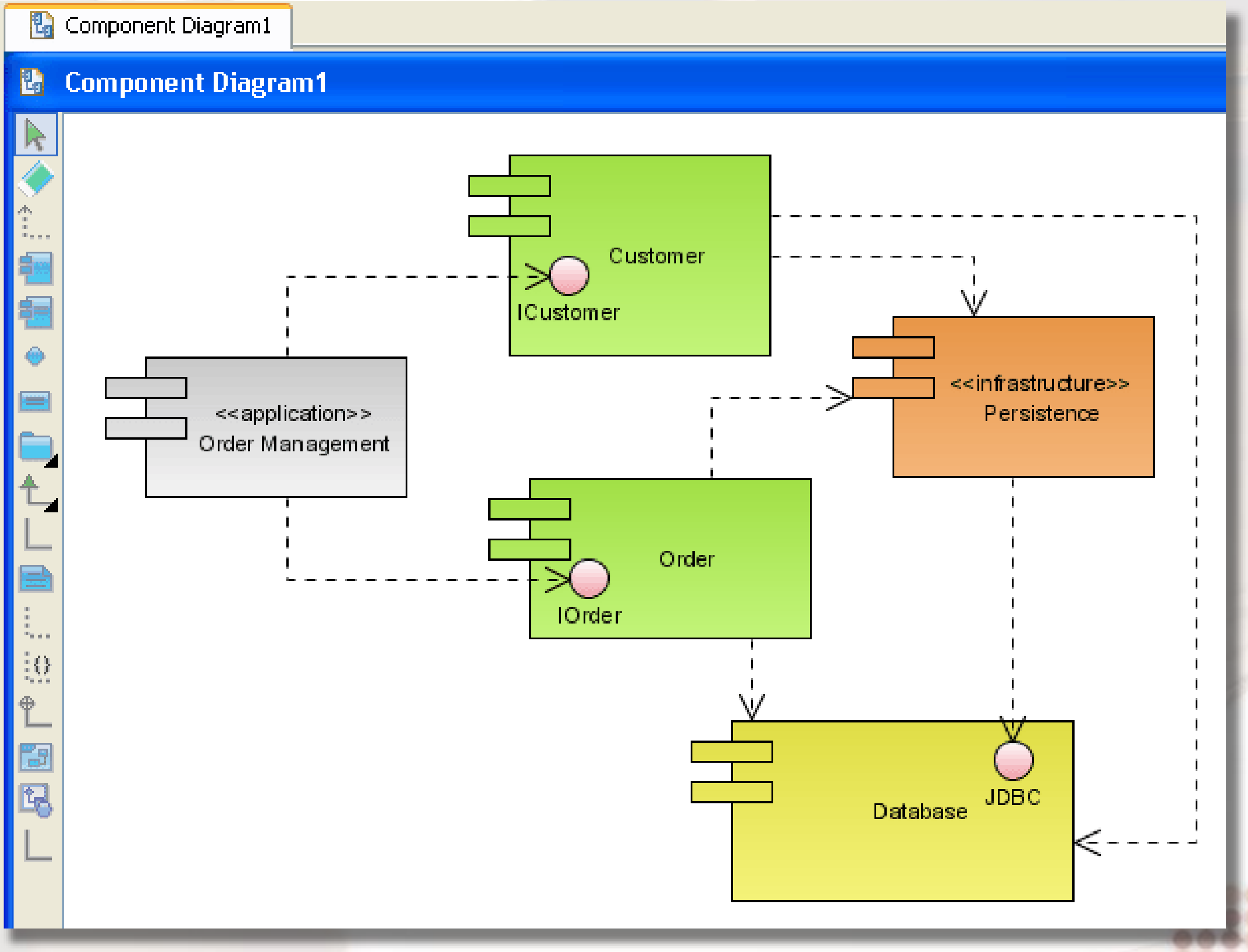


Component

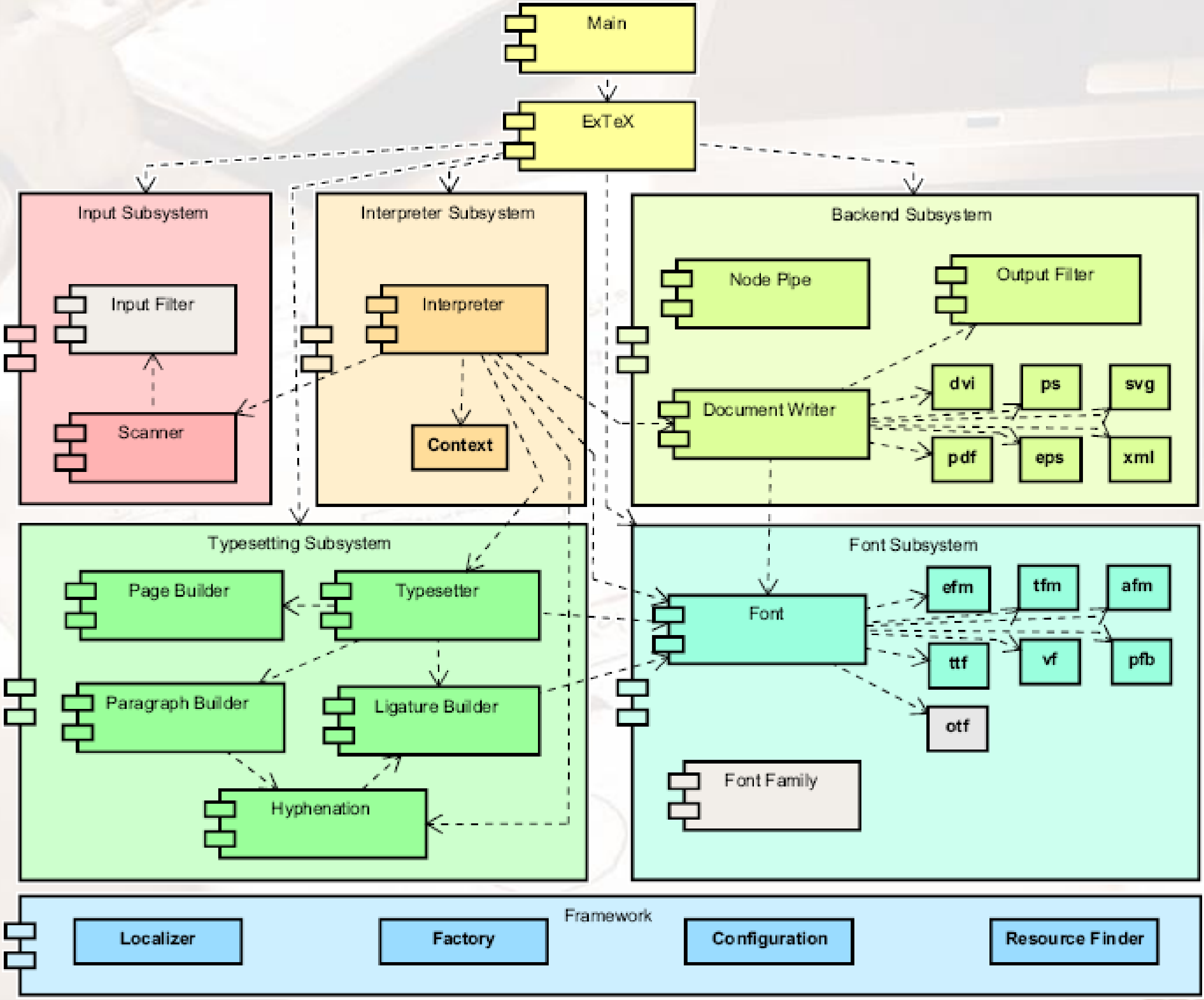
Structure Diagrams: Component Diagram



Structure Diagrams: Component Diagram



Structure Diagrams: Component Diagram

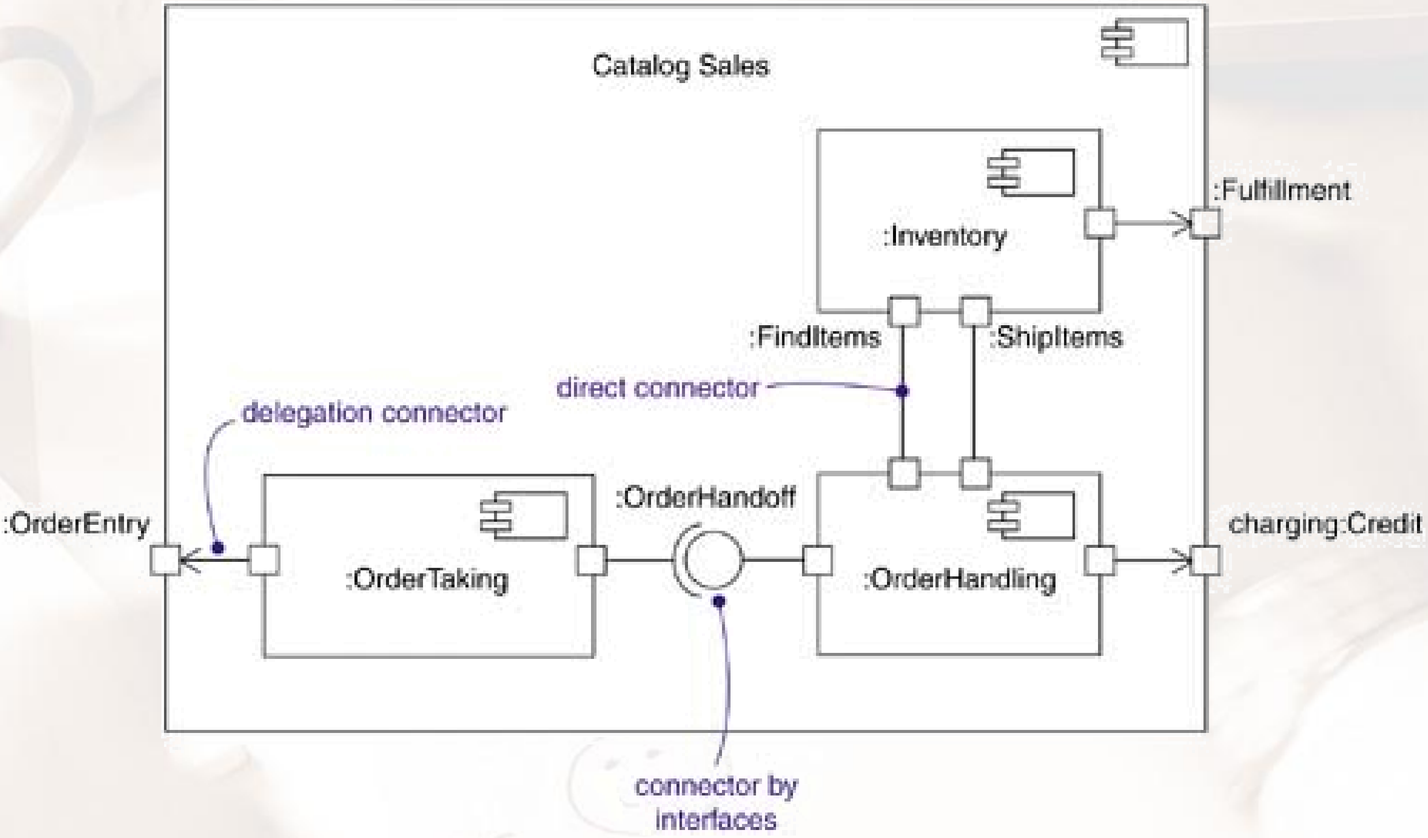


Structure Diagrams: Composite Structure Diagram

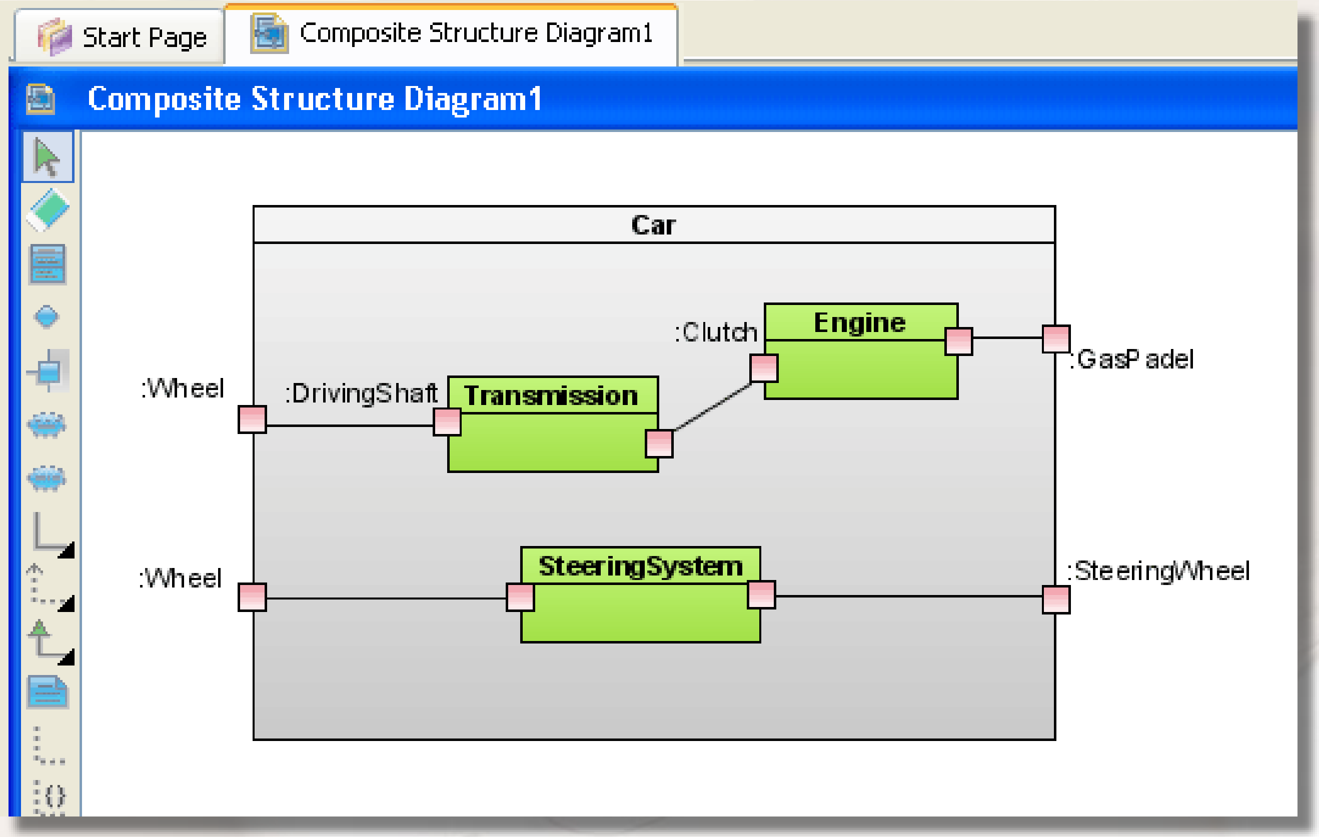
- Describes the internal structure of a class
- Describes the collaborations that this structure makes possible

- **Application:** Any systems which involves modules with internal structure

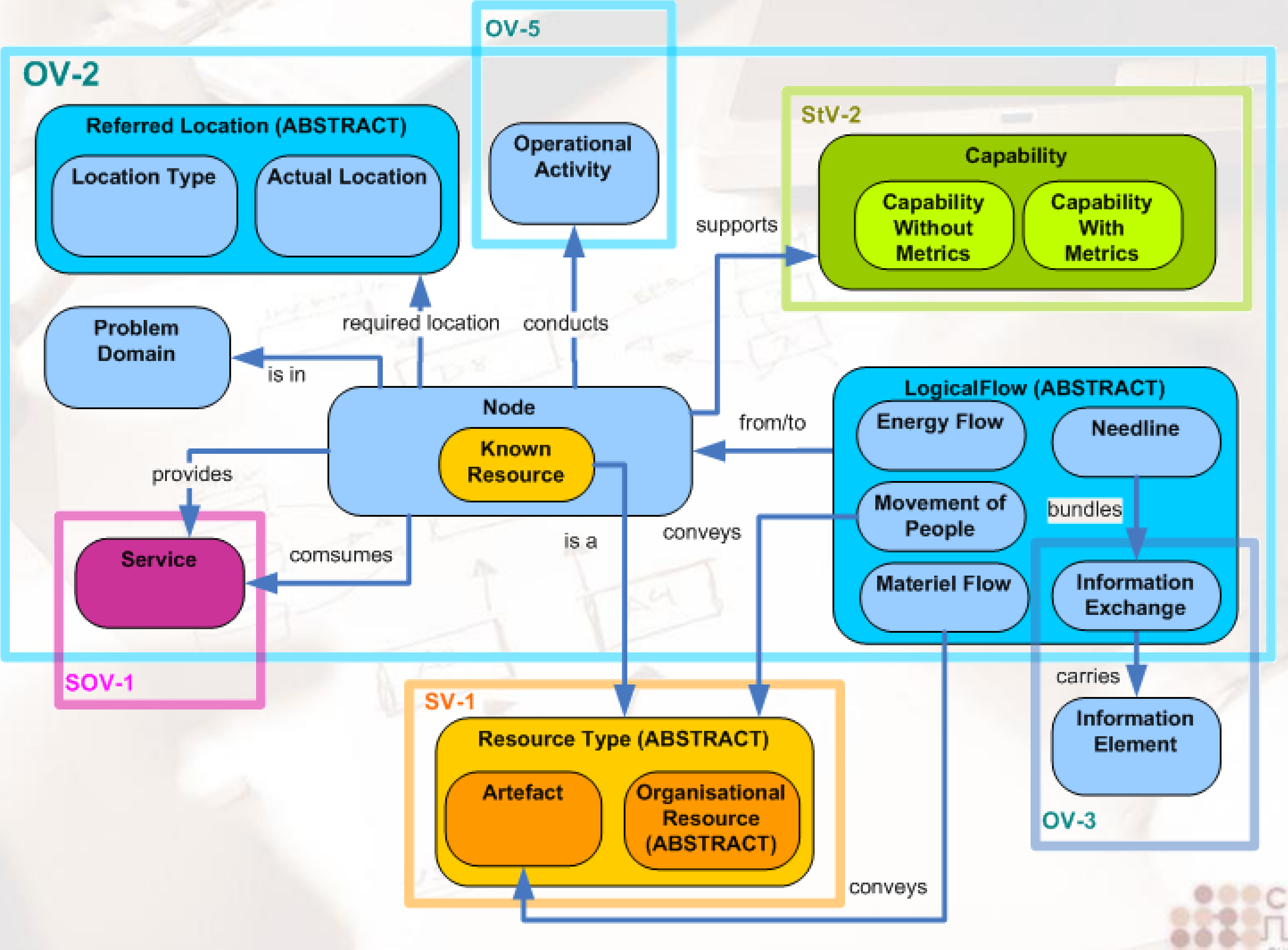
Structure Diagrams: Composite Structure Diagram



Structure Diagrams: Composite Structure Diagram



Structure Diagrams: Composite Structure Diagram

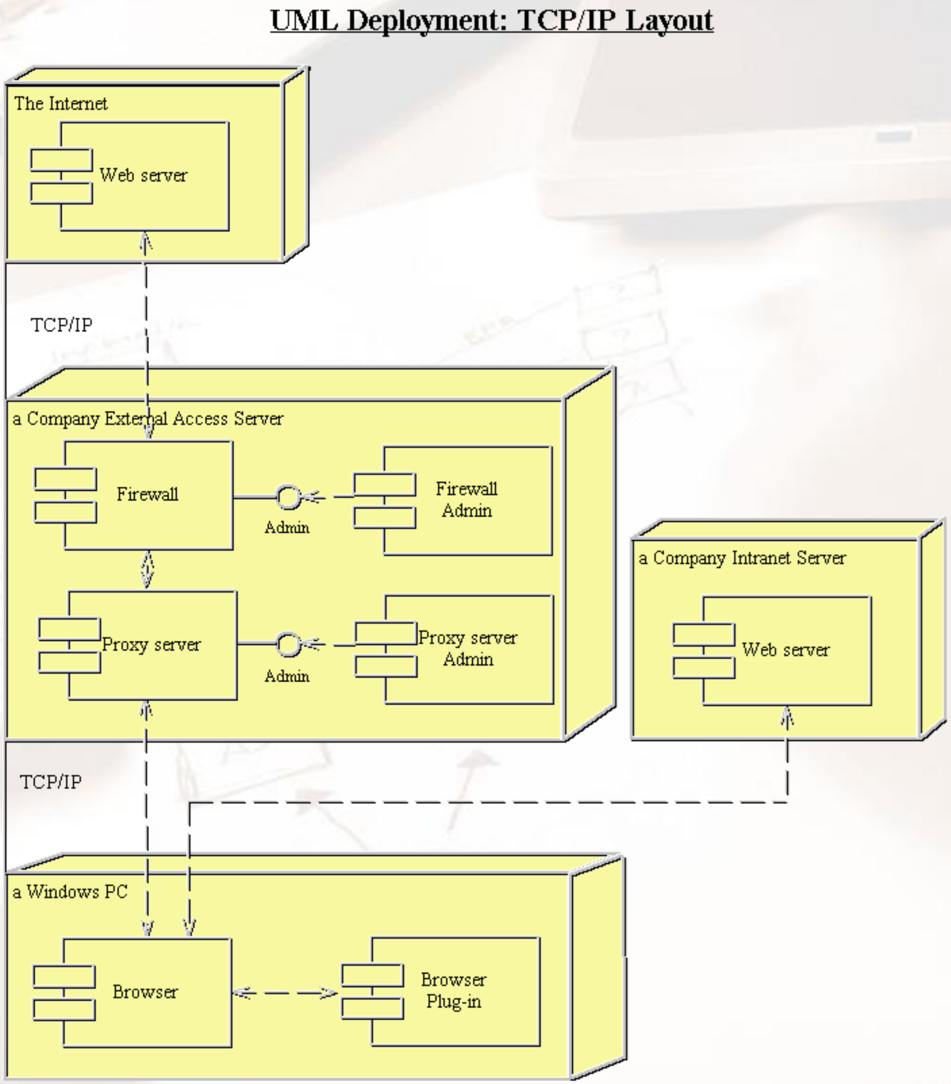


Structure Diagrams: Deployment Diagram

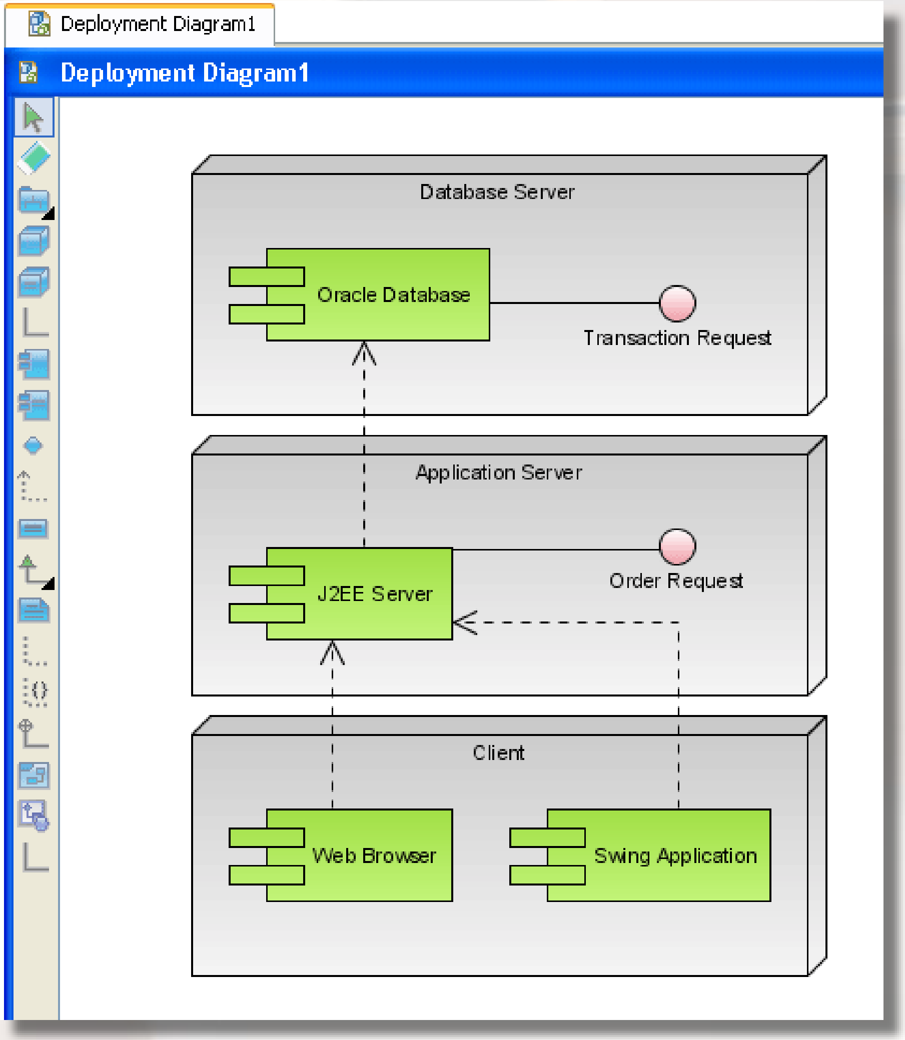
- Serves to model the hardware used in system implementations
- Describes the execution environments and artifacts deployed on the hardware

- **Application:** Hardware dependent systems

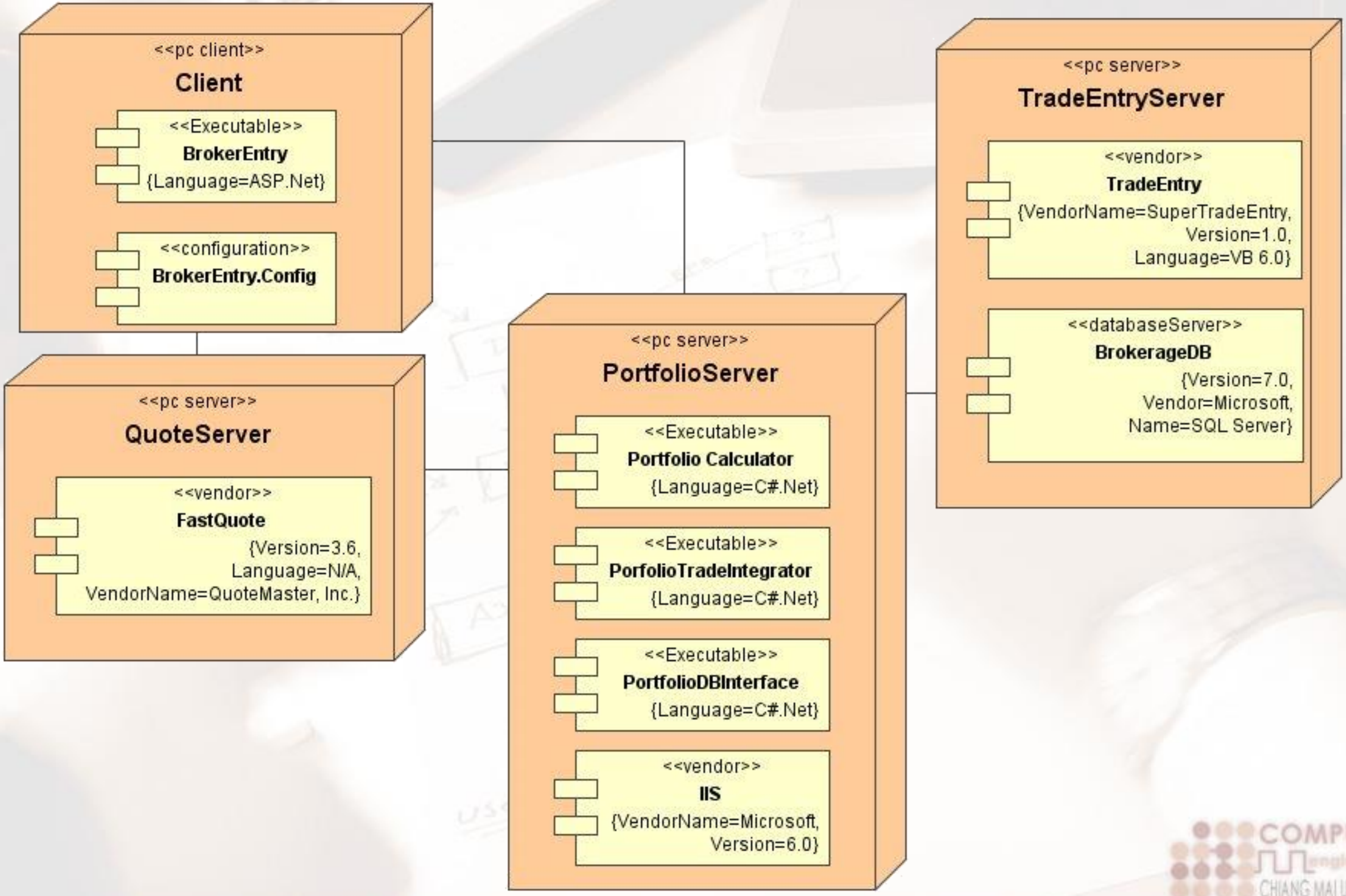
Structure Diagrams: Deployment Diagram



Structure Diagrams: Deployment Diagram



Structure Diagrams: Deployment Diagram

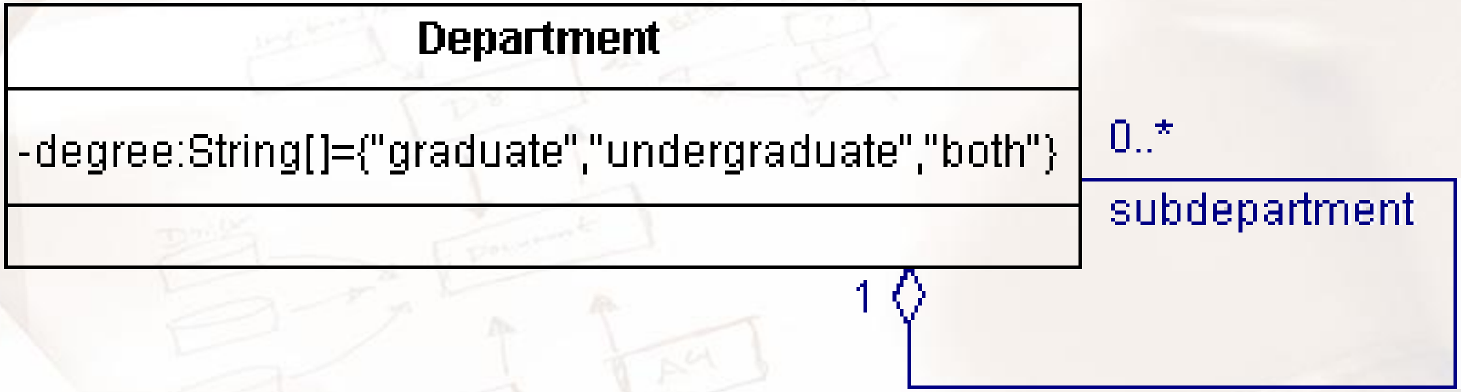


Structure Diagrams: Object Diagram

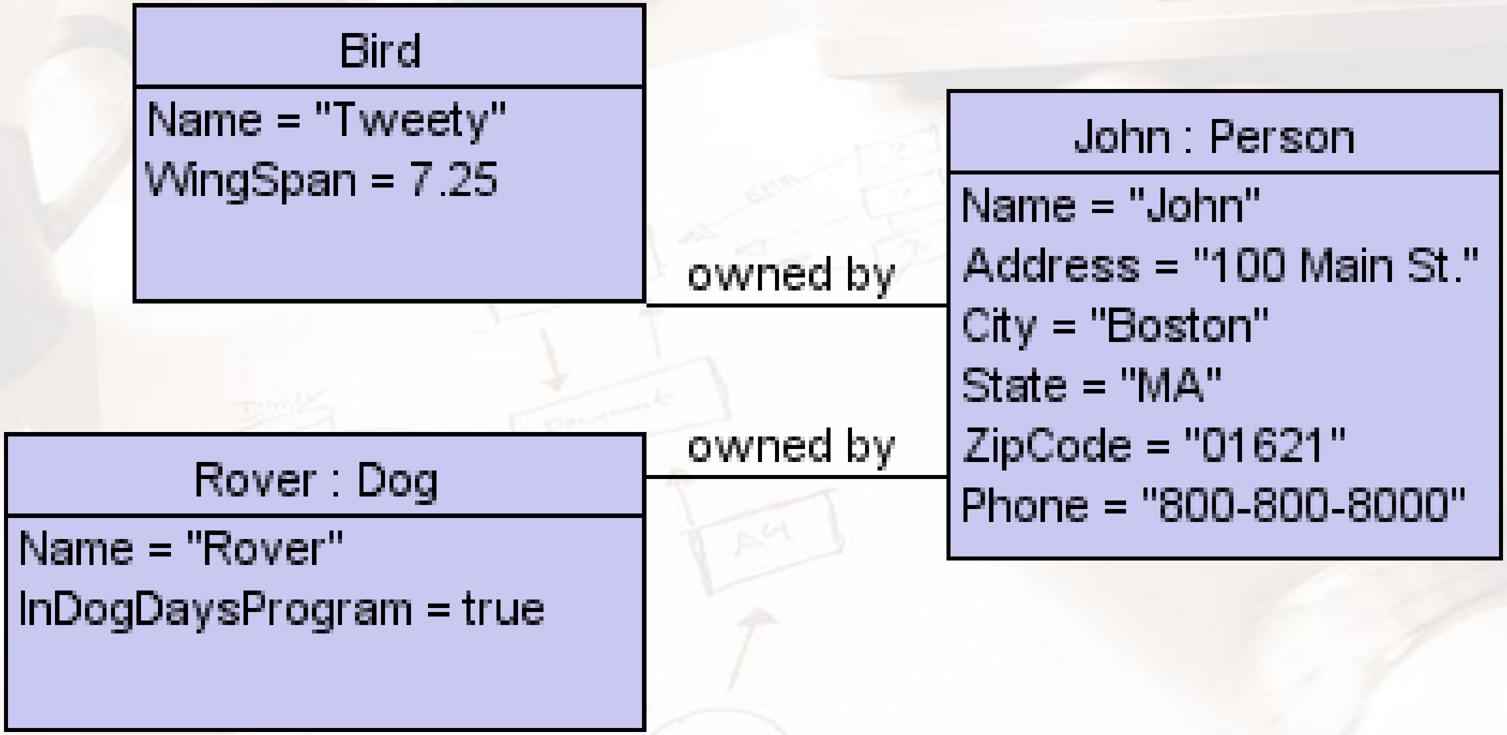
- Shows a complete or partial view of the structure of a modeled system at a specific time

- **Application:** A system which needs to show internal data structure

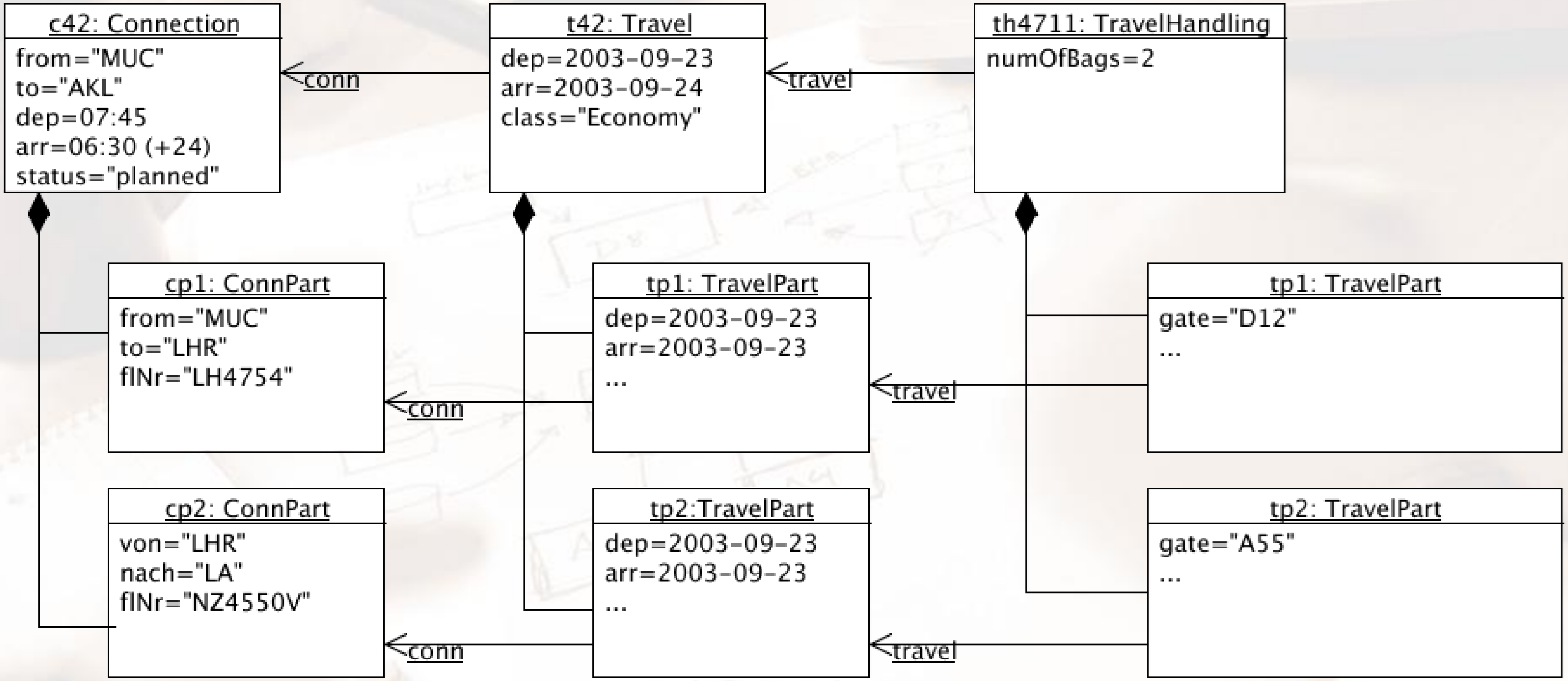
Structure Diagrams: Object Diagram



Structure Diagrams: Object Diagram



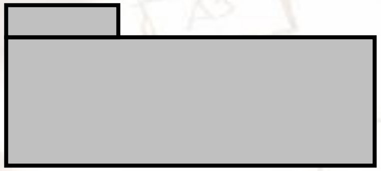
Structure Diagrams: Object Diagram



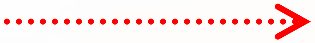
Structure Diagrams: Package Diagram

- Groups objects into packages in order to simplify the system
- Shows the dependencies among these groupings

- **Application:** A system which emphasizes on functionality

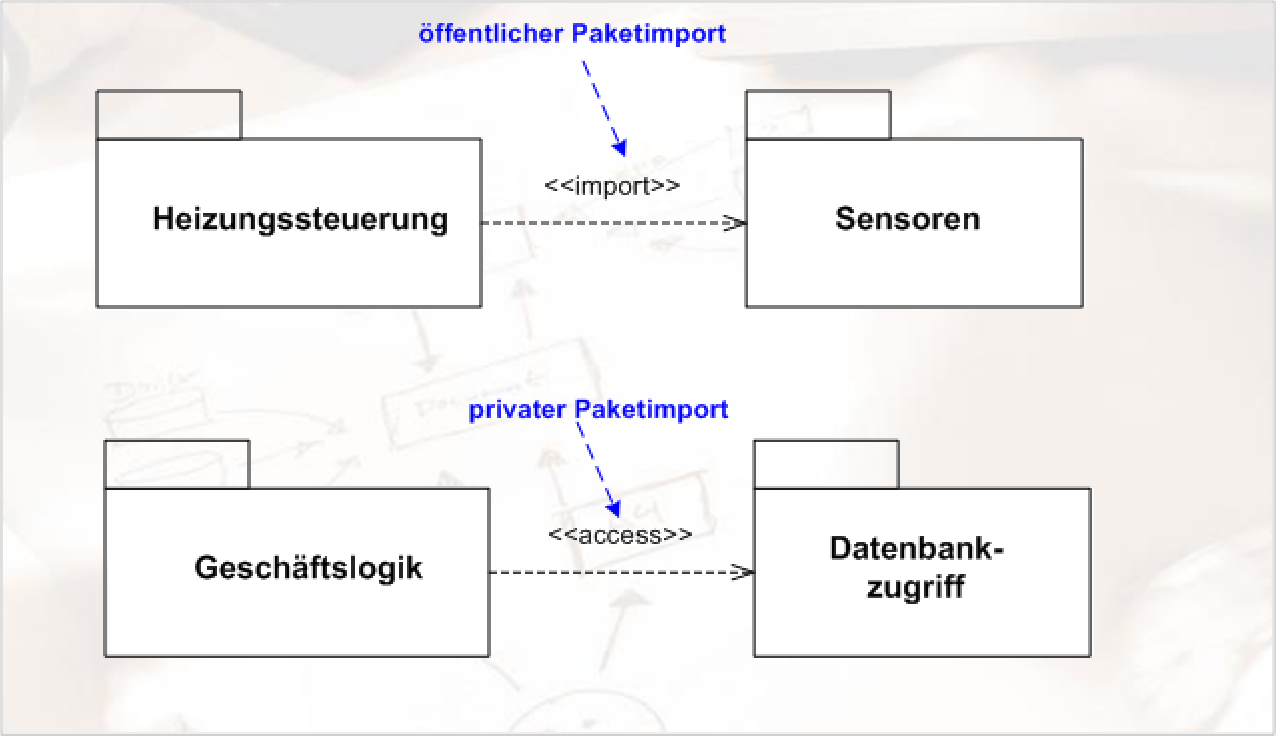


Package

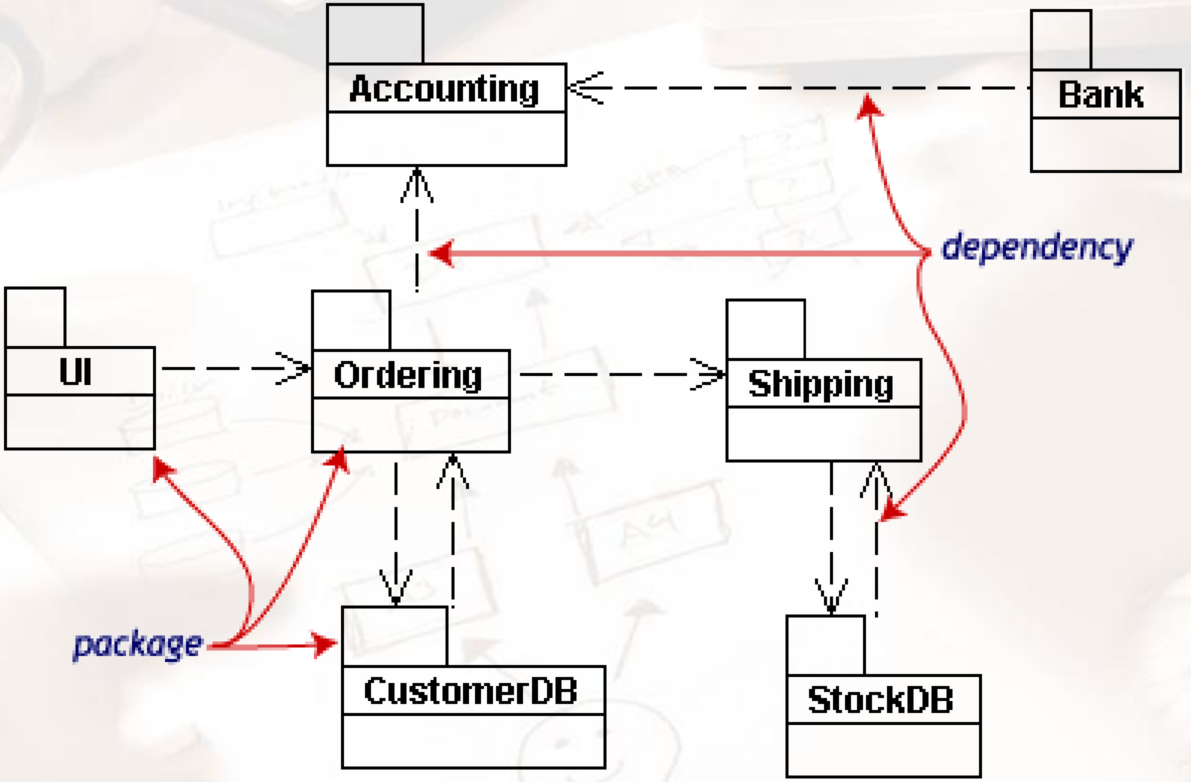


Dependencies

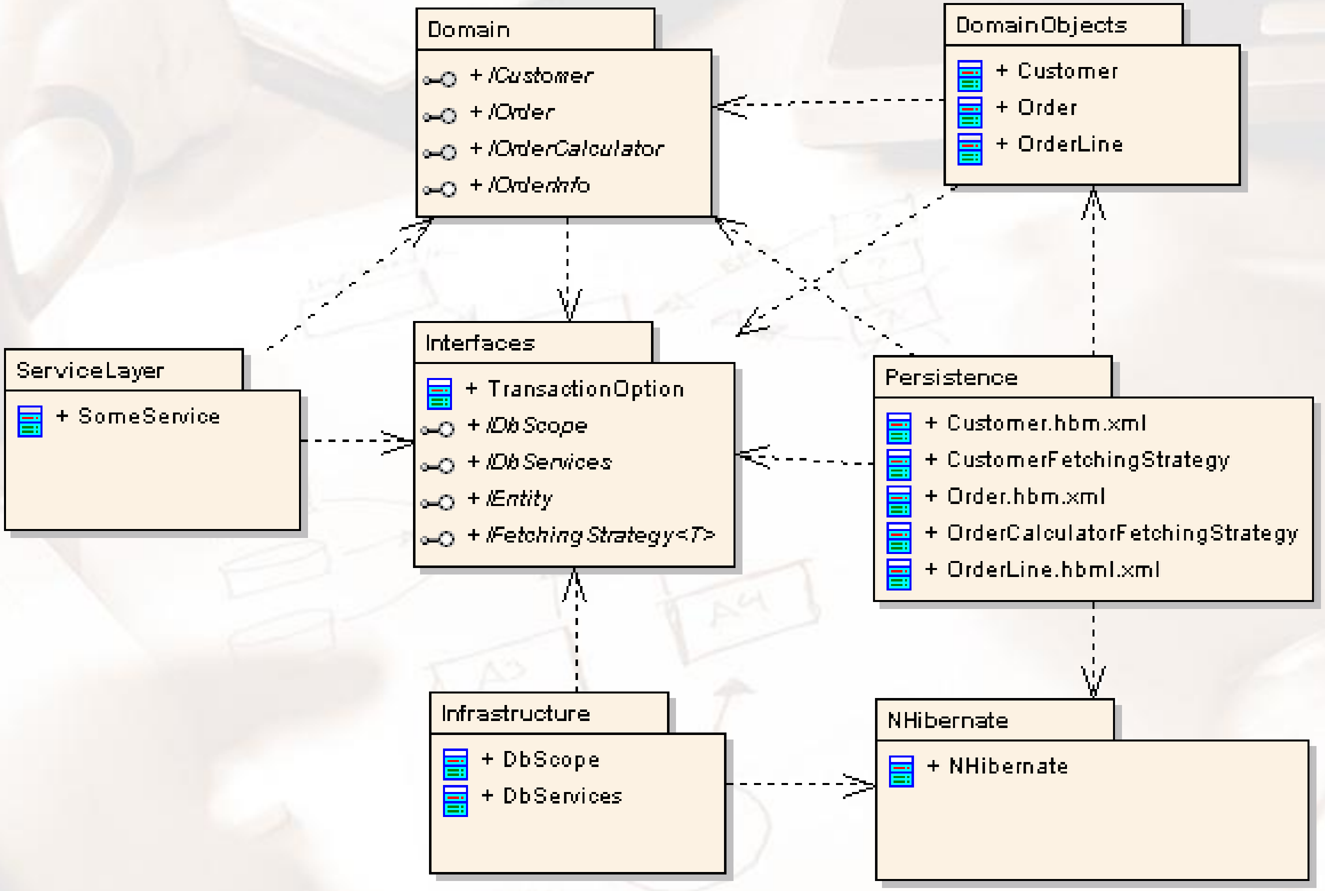
Structure Diagrams: Package Diagram



Structure Diagrams: Package Diagram



Structure Diagrams: Package Diagram

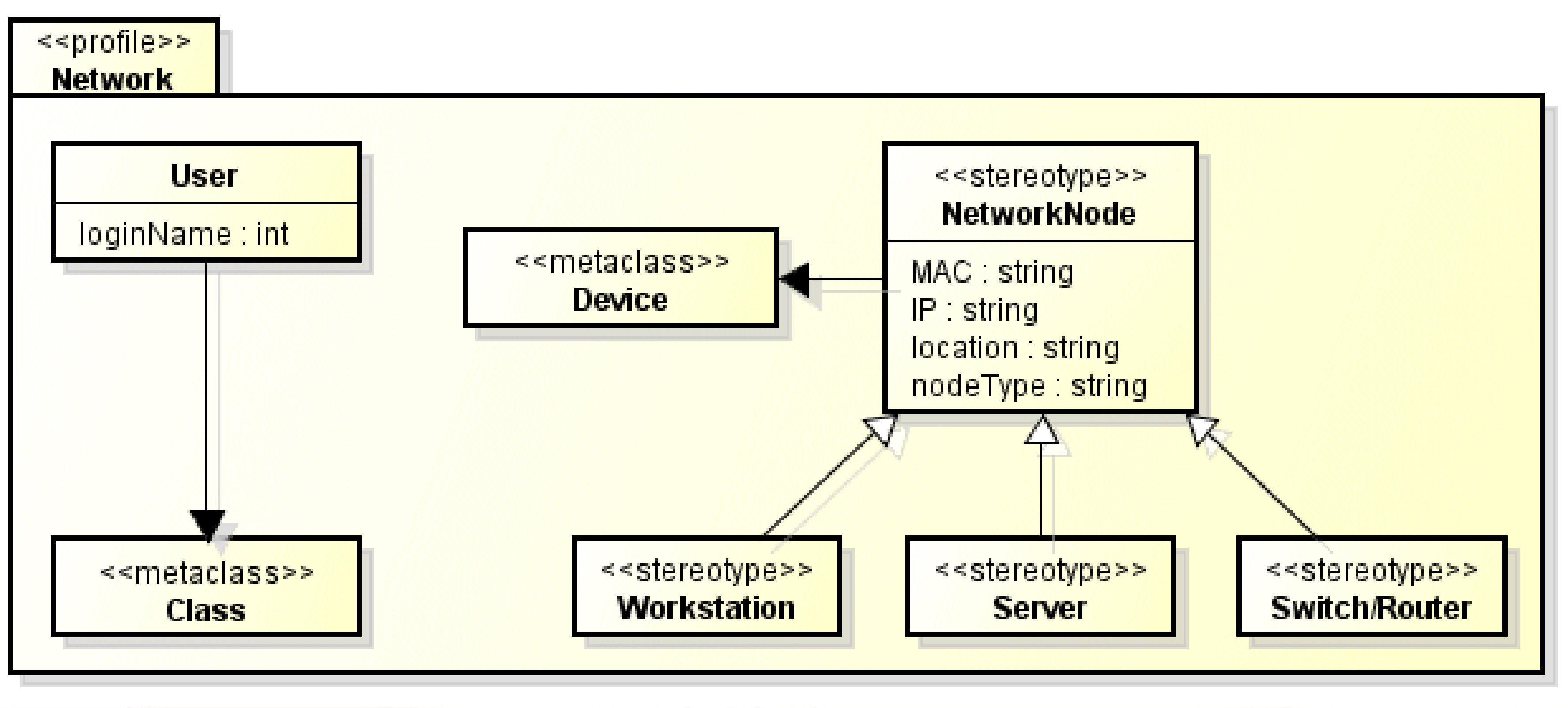


Structure Diagrams: Profile Diagram

- A kind of package that extends a reference metamodel by defining limited extensions to the reference metamodel with the purpose of adapting it to a specific platform or domain.
- Show custom stereotypes.
- **Application:** A system working on different platforms.

Structure Diagrams: Profile Diagram

- Image courtesy of technologyUK.net.

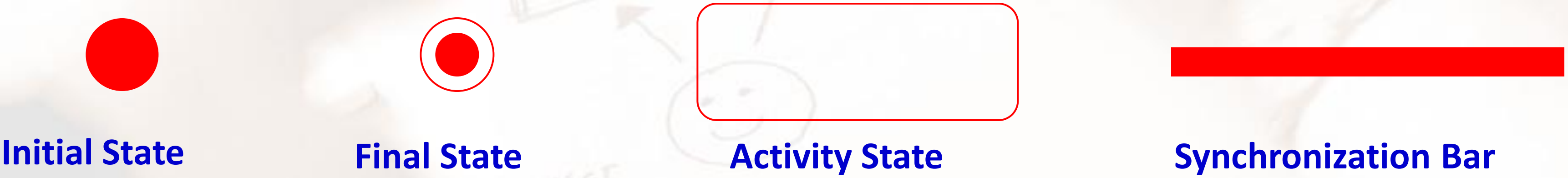


Behavior Diagrams

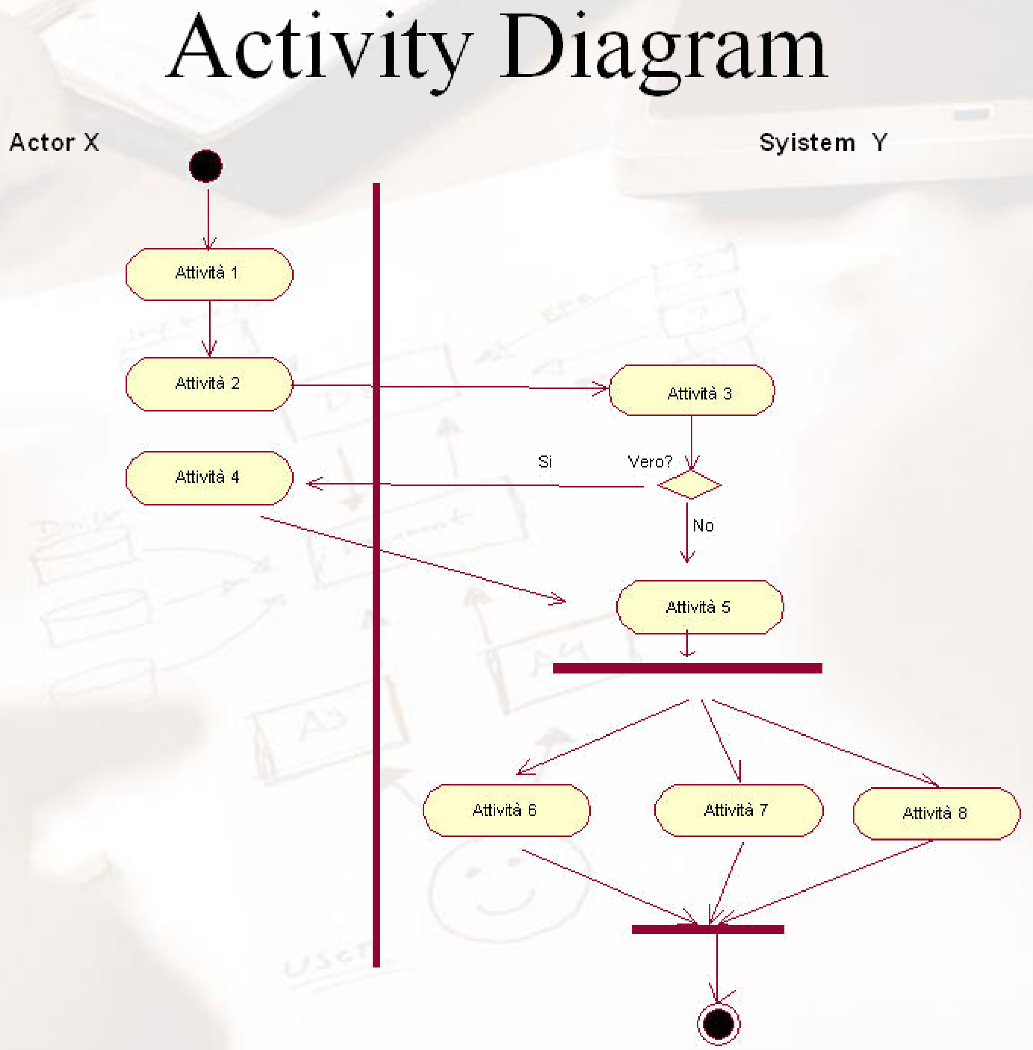
- Emphasizes what must happen in the system being modeled

Behavior Diagrams: Activity Diagram

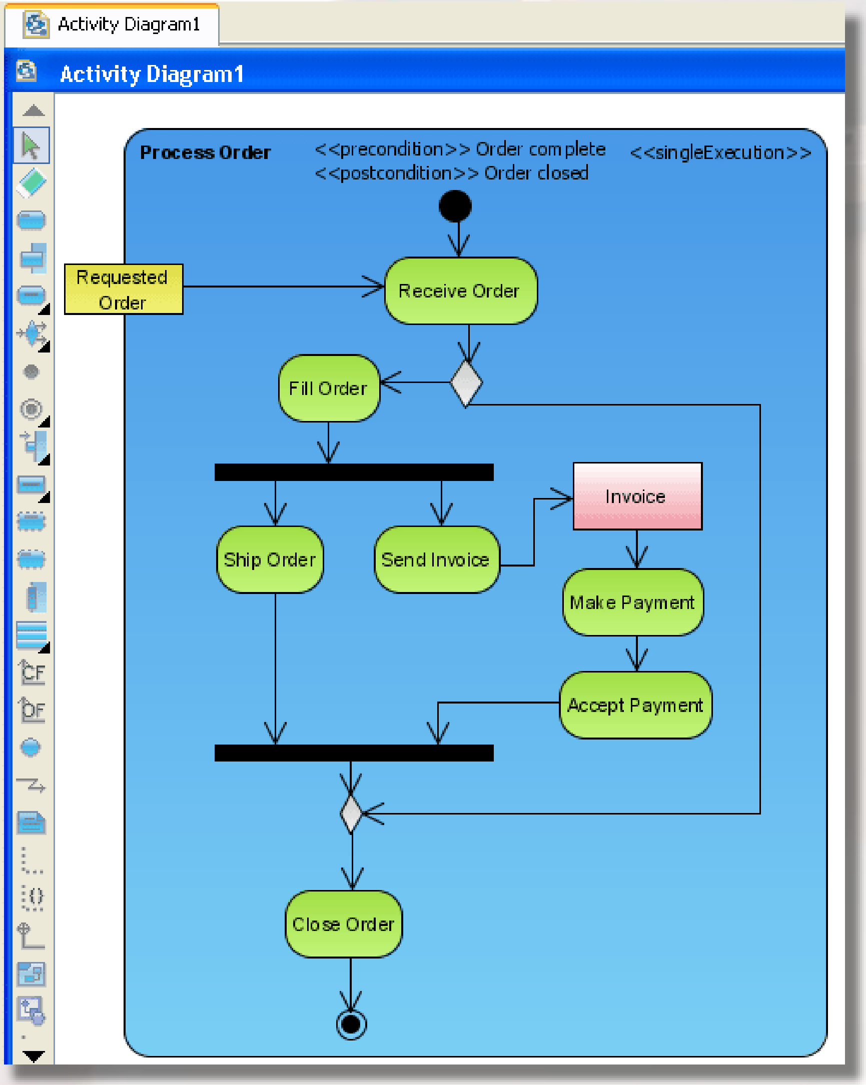
- Represents the business and operational step-by-step workflows of components in a system
- An activity diagram shows the overall flow of control
- **Application:** Flowing processes



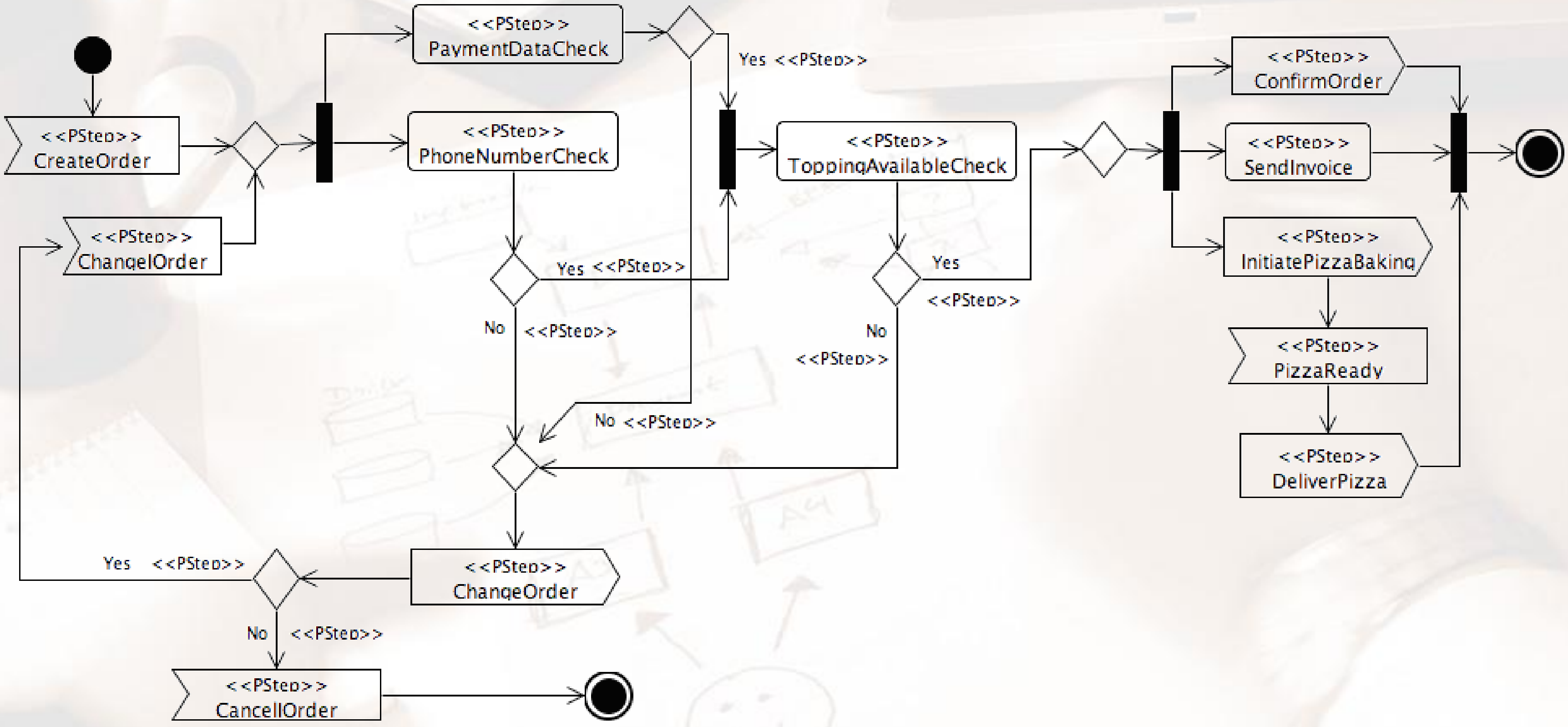
Behavior Diagrams: Activity Diagram



Behavior Diagrams: Activity Diagram



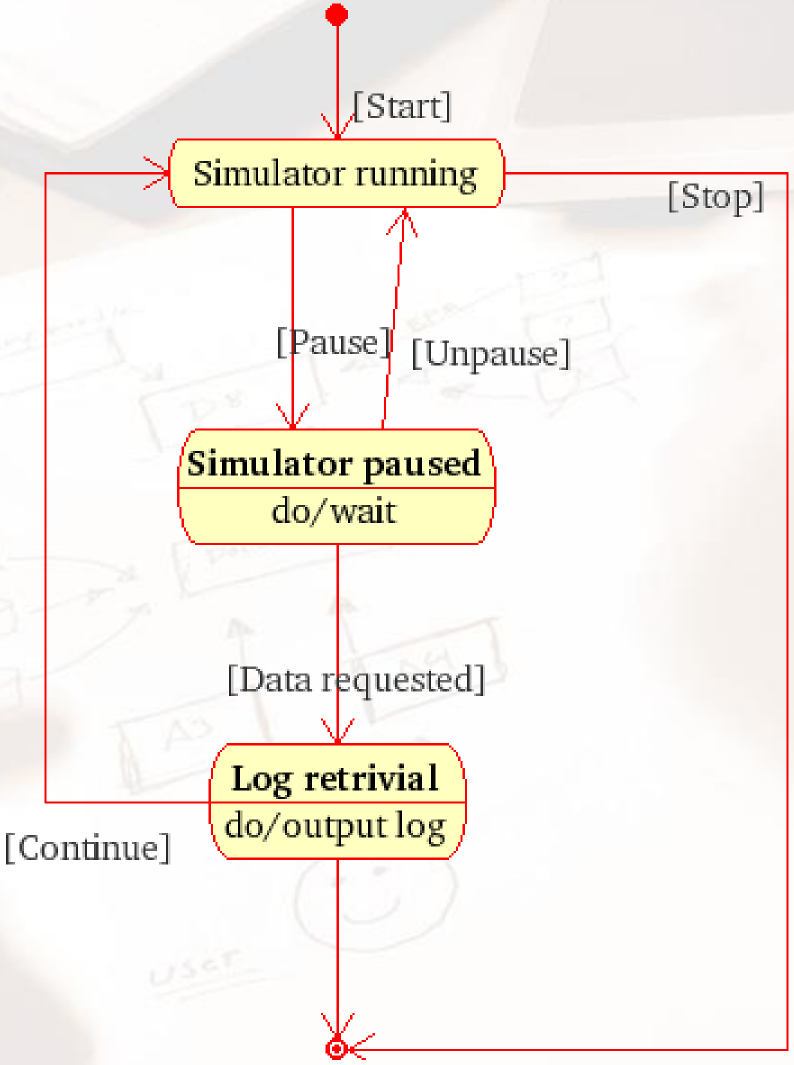
Behavior Diagrams: Activity Diagram



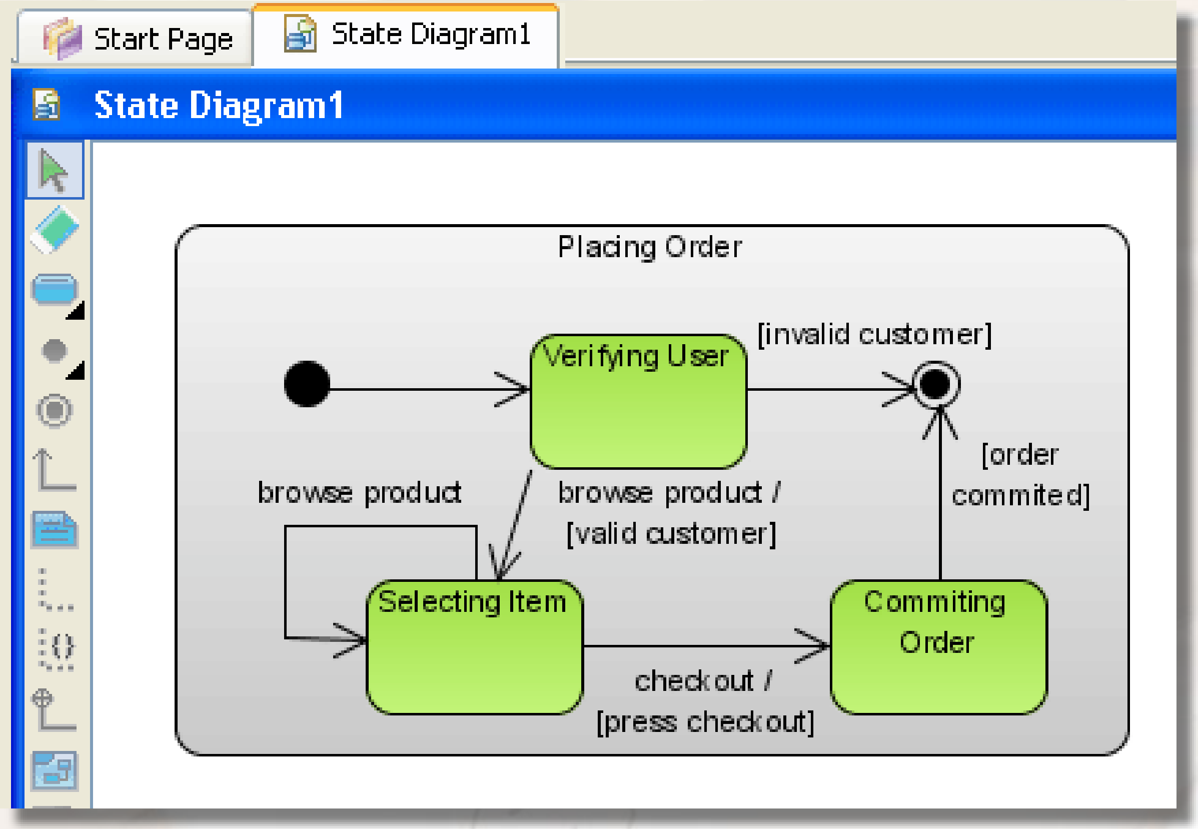
Behavior Diagrams: State Machine Diagram

- Describe various states which certain classes response to certain events
- It is normally used for explaining classes with high complexity
- **Application:** Flowing processes with different choices of interactions

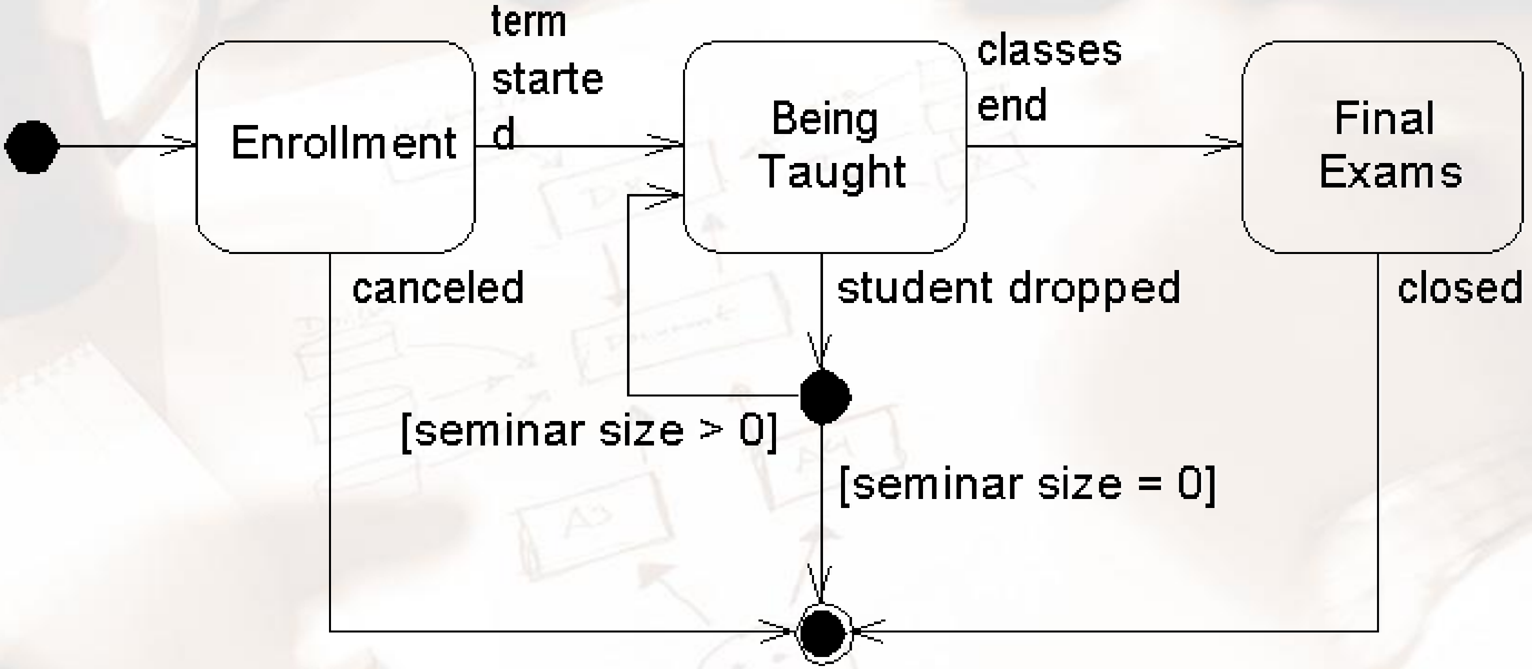
Behavior Diagrams: State Machine Diagram



Behavior Diagrams: State Machine Diagram



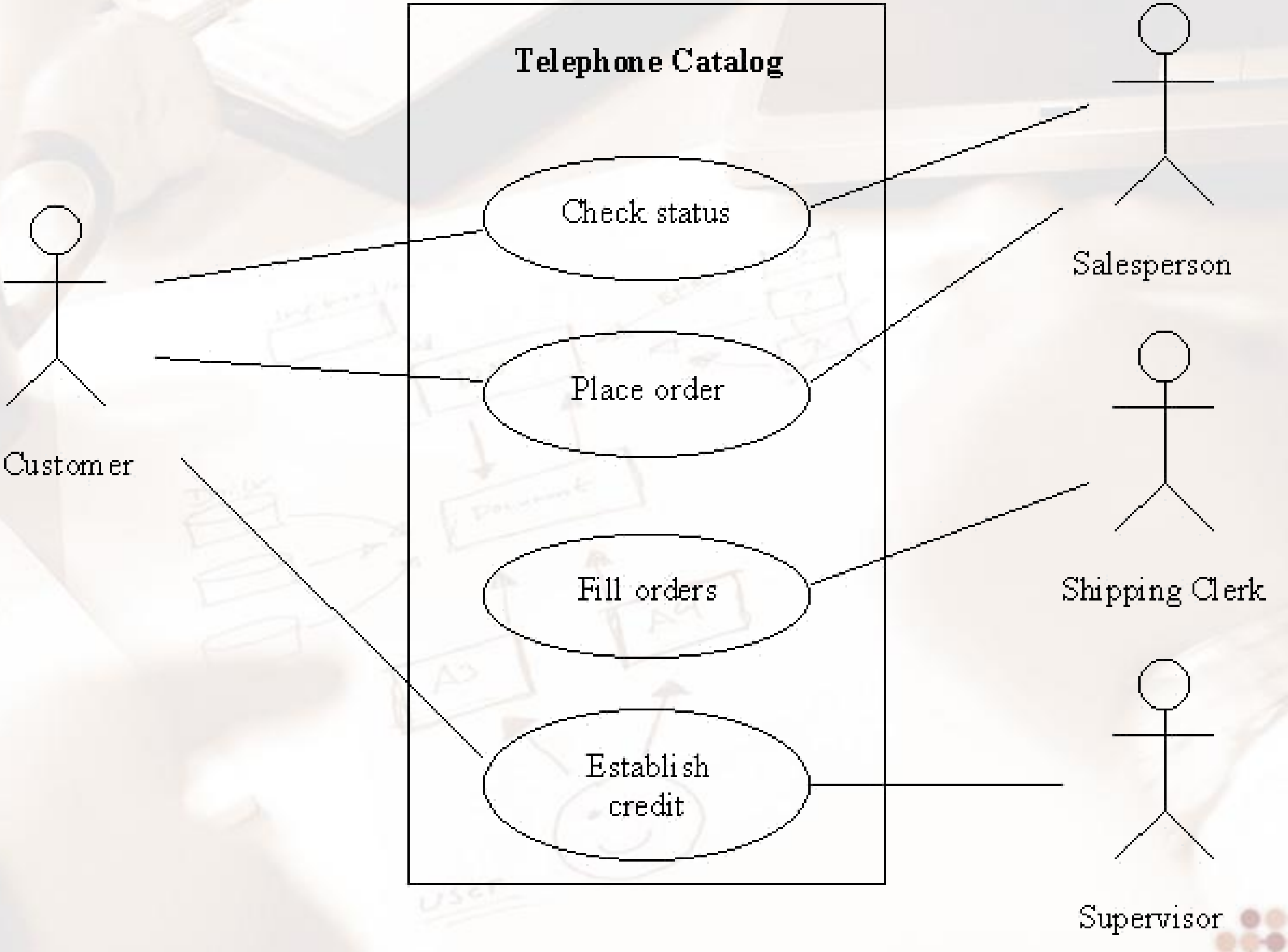
Behavior Diagrams: State Machine Diagram



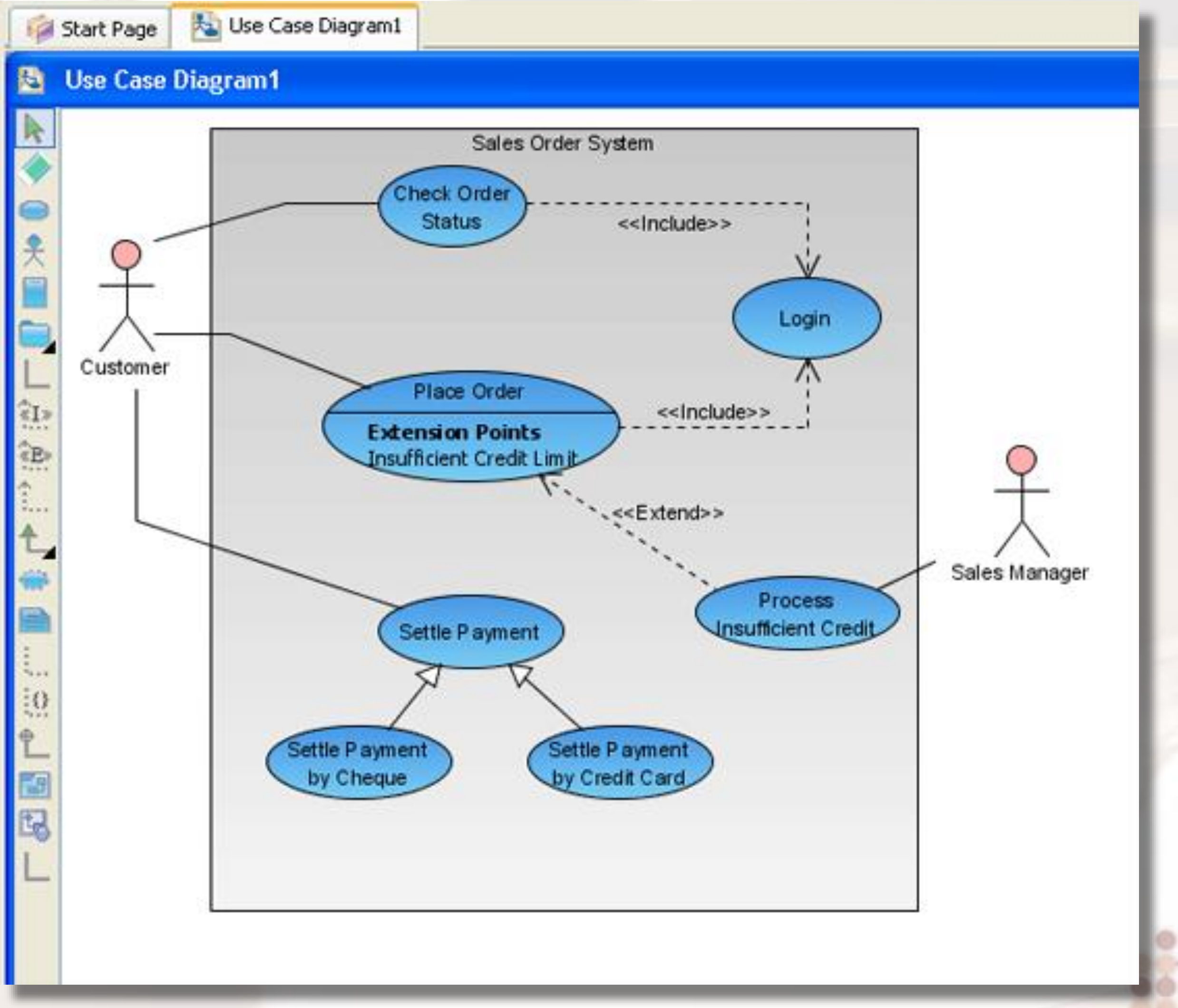
Behavior Diagrams: Use Case Diagram

- Describe what a system does from the standpoint of an external observer
- The emphasis is on what a system does rather than how
- **Application:** Determining features, communicating with clients, generating test cases

Behavior Diagrams: Use Case Diagram

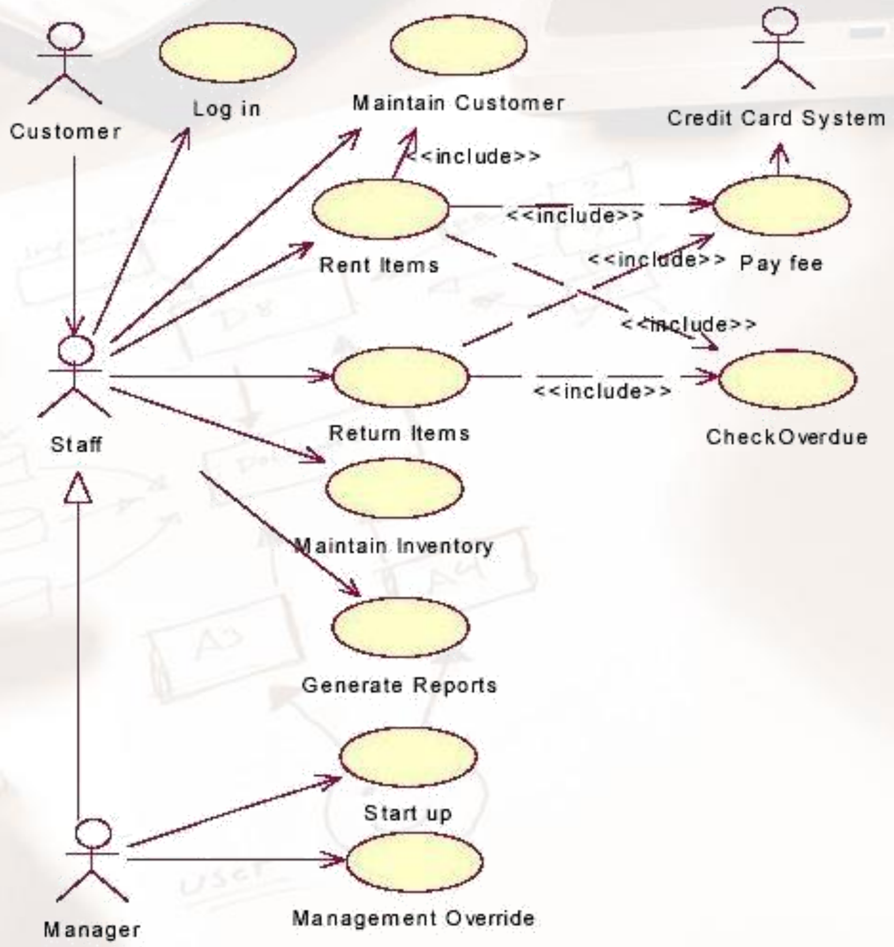


Behavior Diagrams: Use Case Diagram



Behavior Diagrams: Use Case Diagram

Video Rental Store Use Case Diagram



Interaction Diagrams

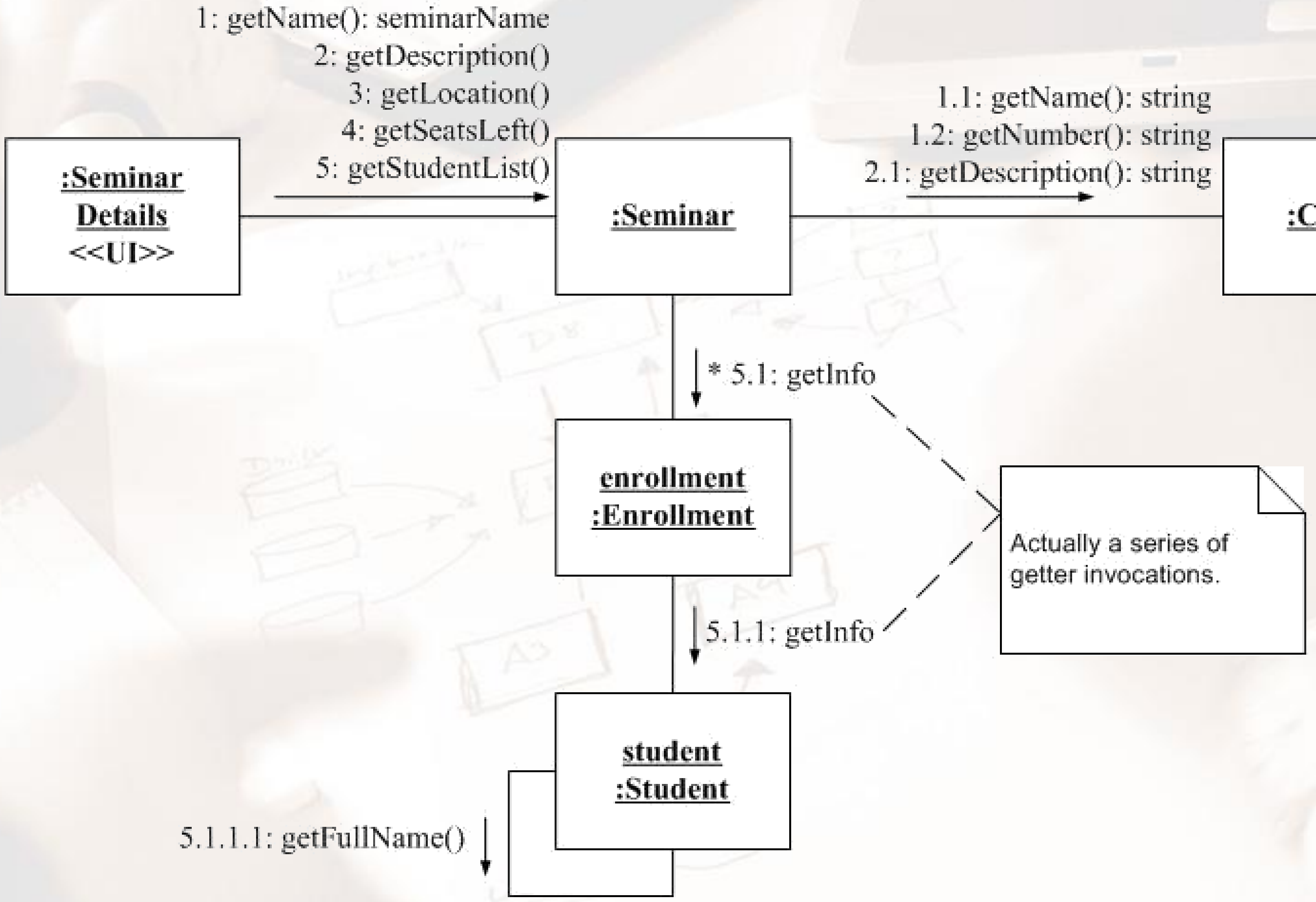
- A subset of behavior diagrams
- Emphasize the flow of control and data among the things in the system being modeled

Interaction Diagrams: Communication Diagram

- Shows the message flow between objects in an OO application
- Shows the basic associations (relationships) between classes

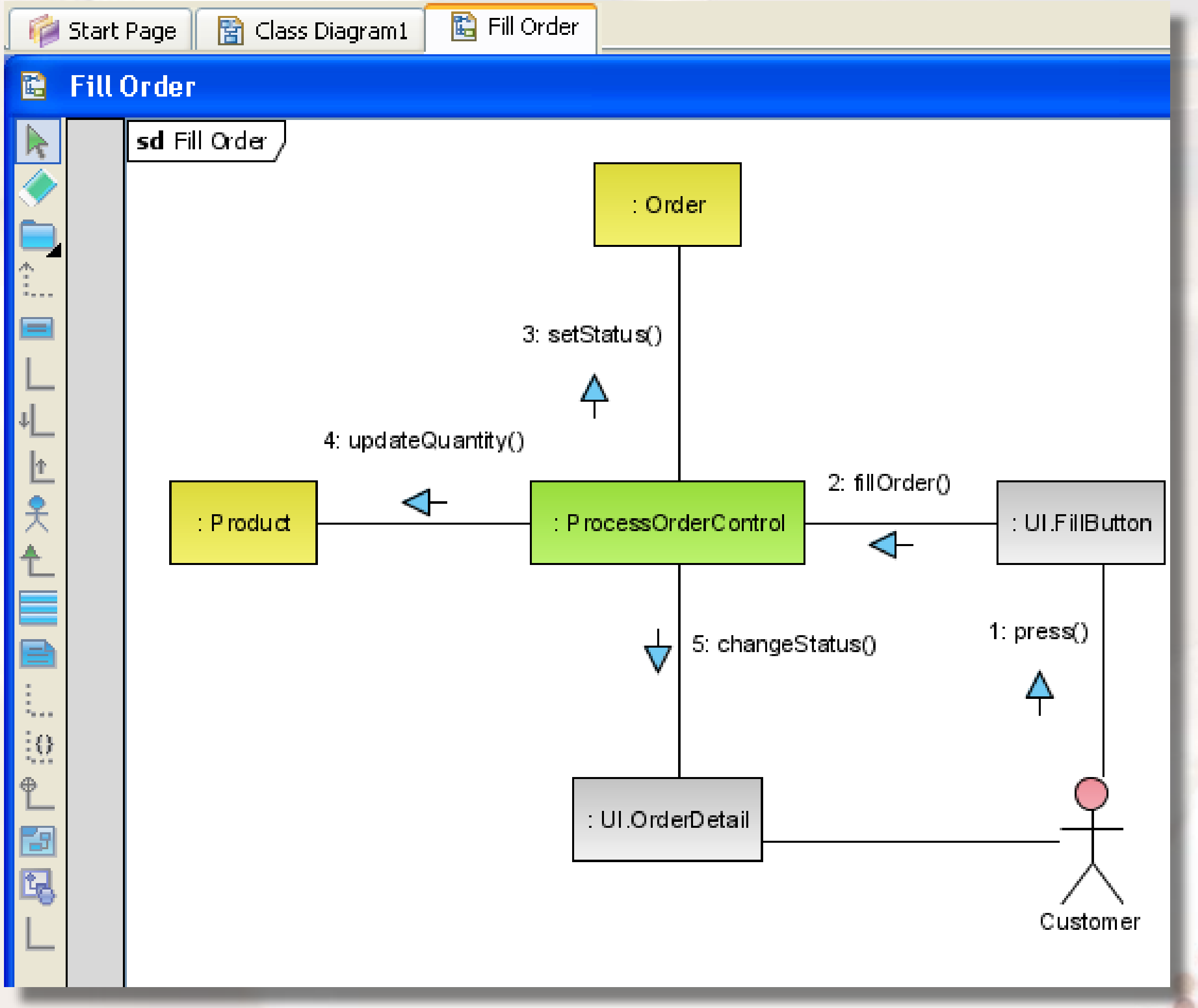
- **Application:** General systems with interactive processes

Interaction Diagrams: Communication Diagram

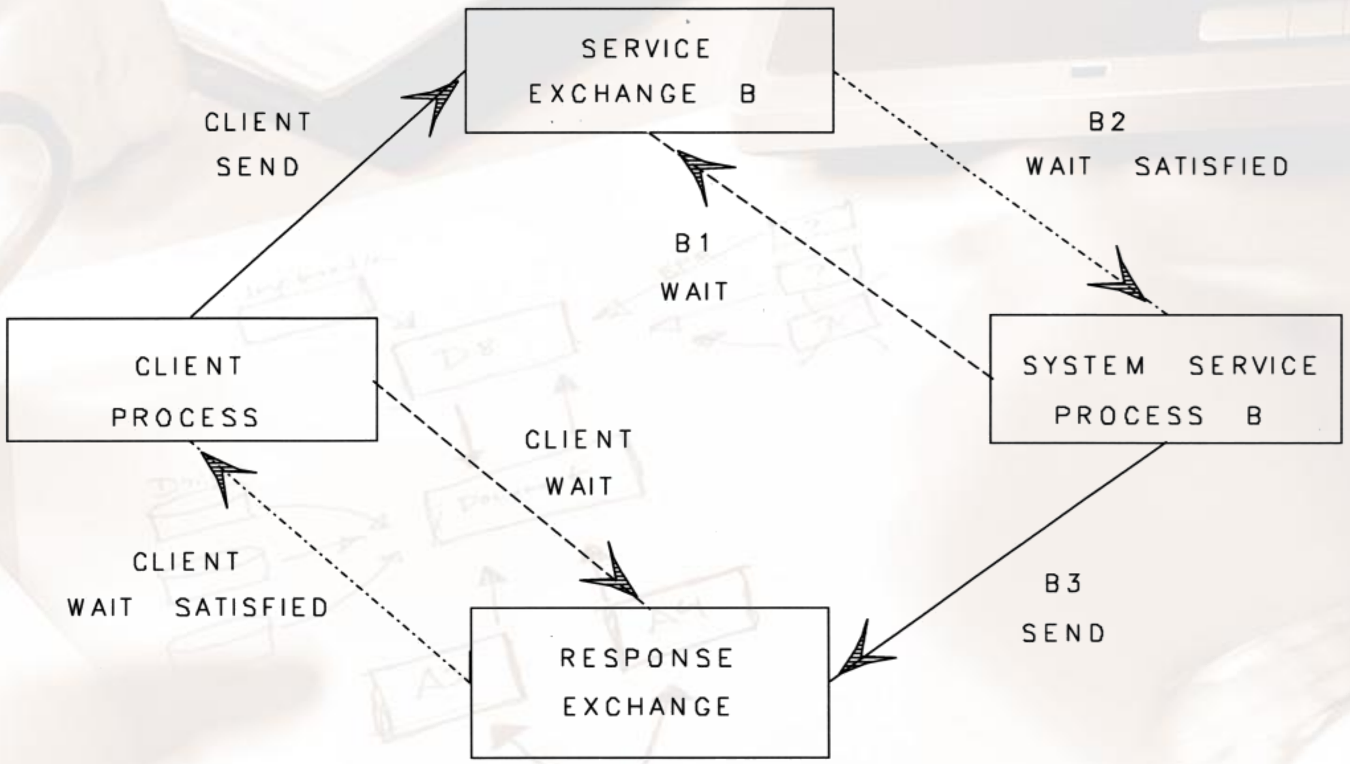


Actually a series of getter invocations.

Interaction Diagrams: Communication Diagram



Interaction Diagrams: Communication Diagram



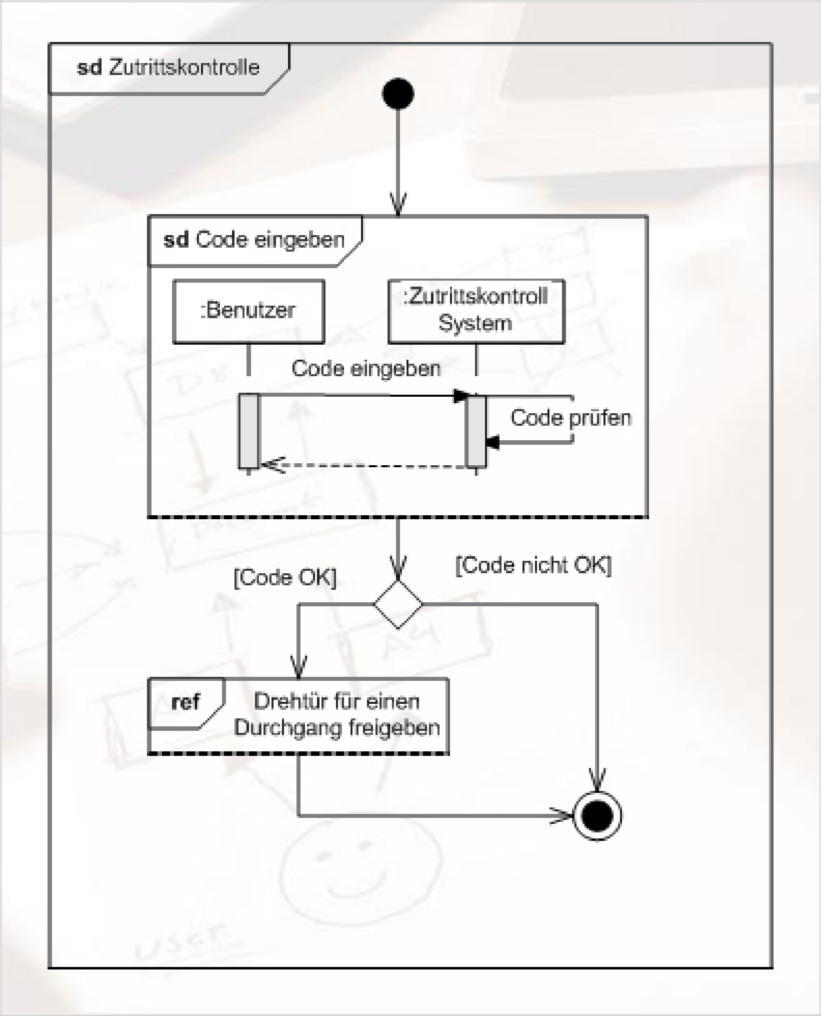
CLIENT AND SYSTEM SERVICE IPC

Interaction Diagrams: Interaction Overview Diagram

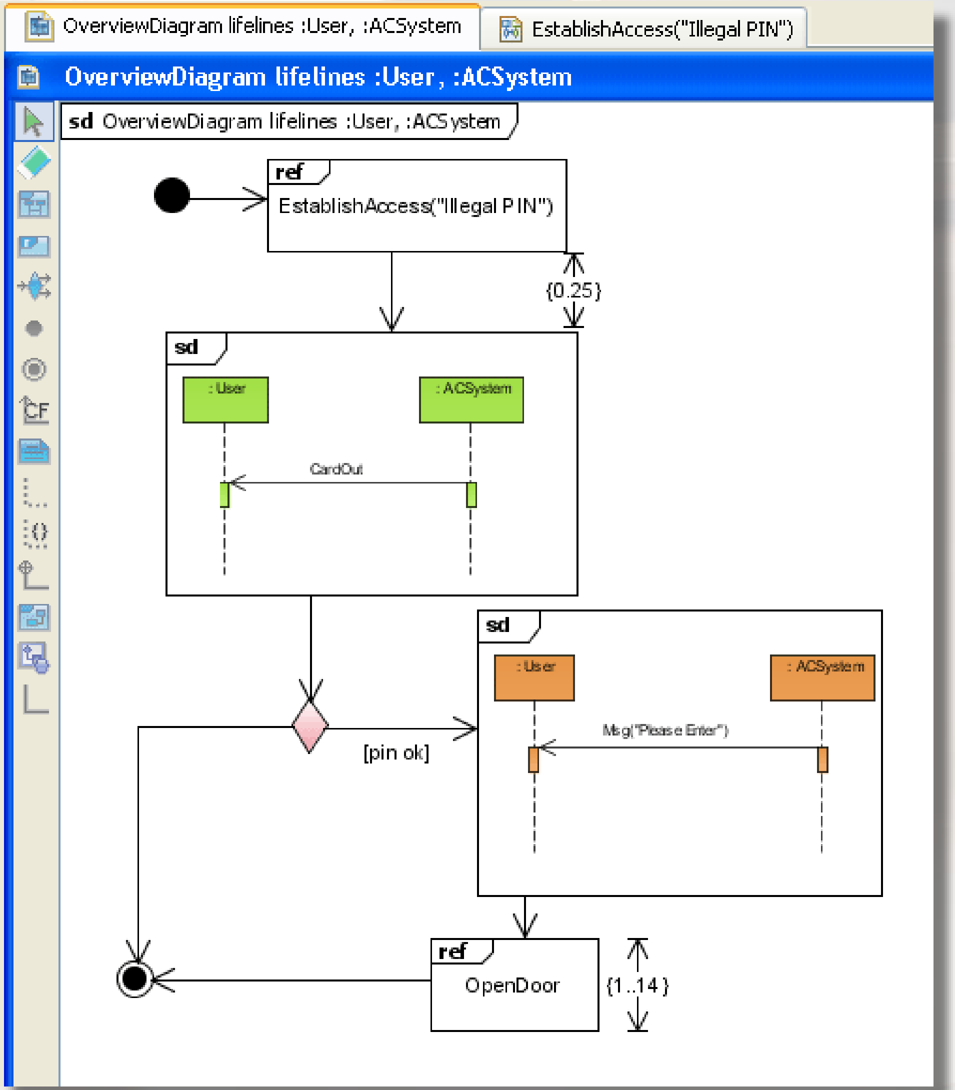
- Is an activity diagram in which overviews control flows
- The nodes within the diagram are framed

- **Application:** System components with flowing interactions

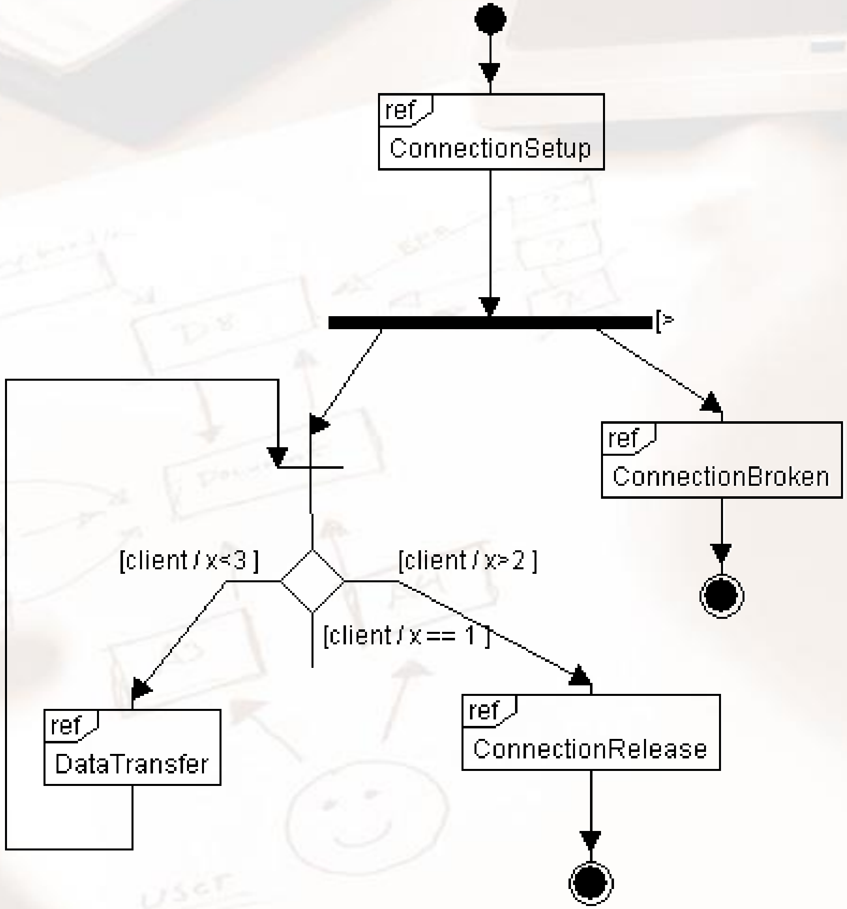
Interaction Diagrams: Interaction Overview Diagram



Interaction Diagrams: Interaction Overview Diagram


OverviewDiagram lifelines :User, :ACSystem
EstablishAccess("Illegal PIN")
OverviewDiagram lifelines :User, :ACSystem
sd OverviewDiagram lifelines :User, :ACSystem
ref
EstablishAccess("Illegal PIN")
{0.25}
sd
: User
: ACSystem
CardOut
sd
: User
: ACSystem
Msg("Please Enter")
[pin ok]
ref
OpenDoor
{1..14}

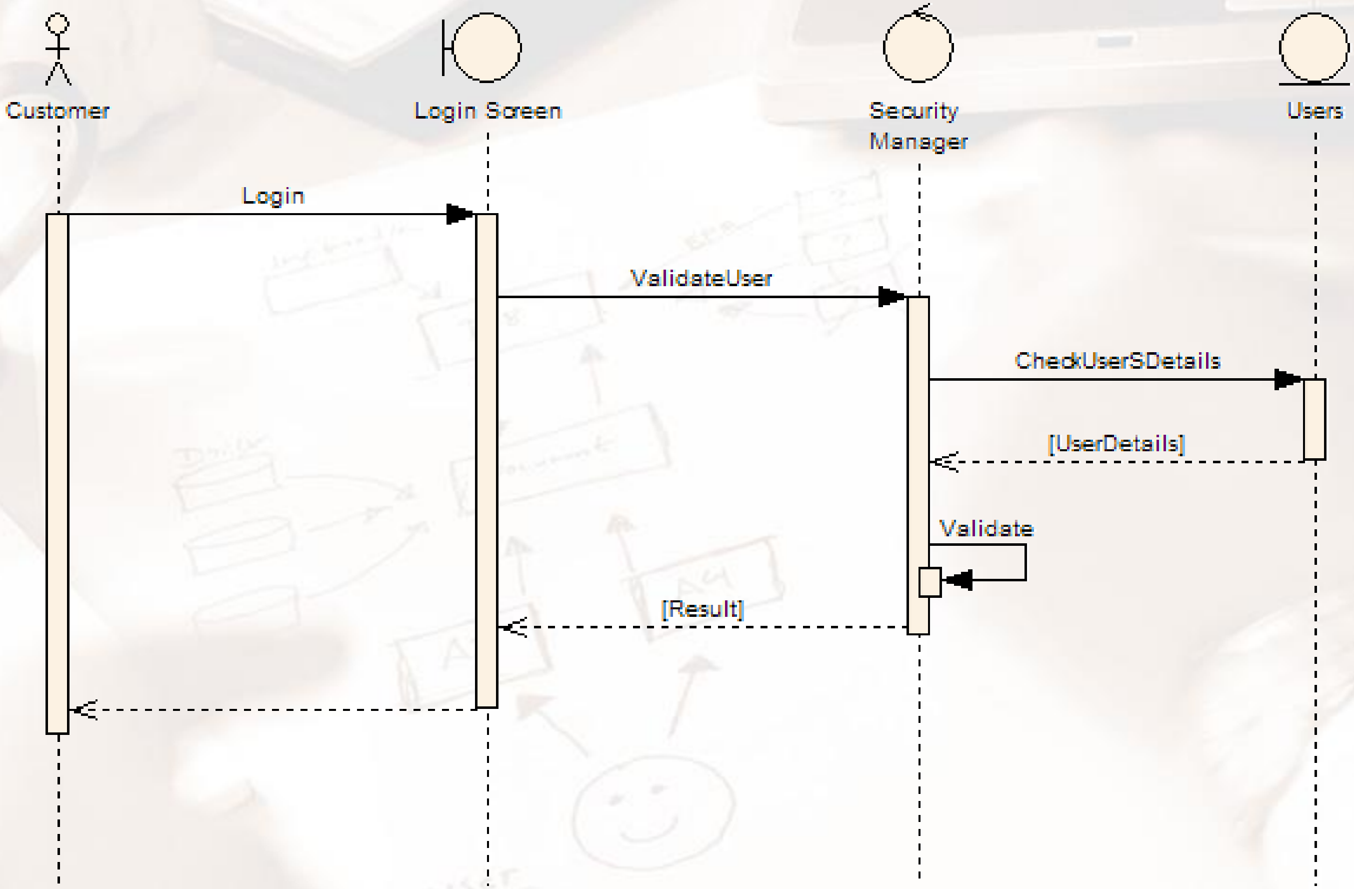
Interaction Diagrams: Interaction Overview Diagram



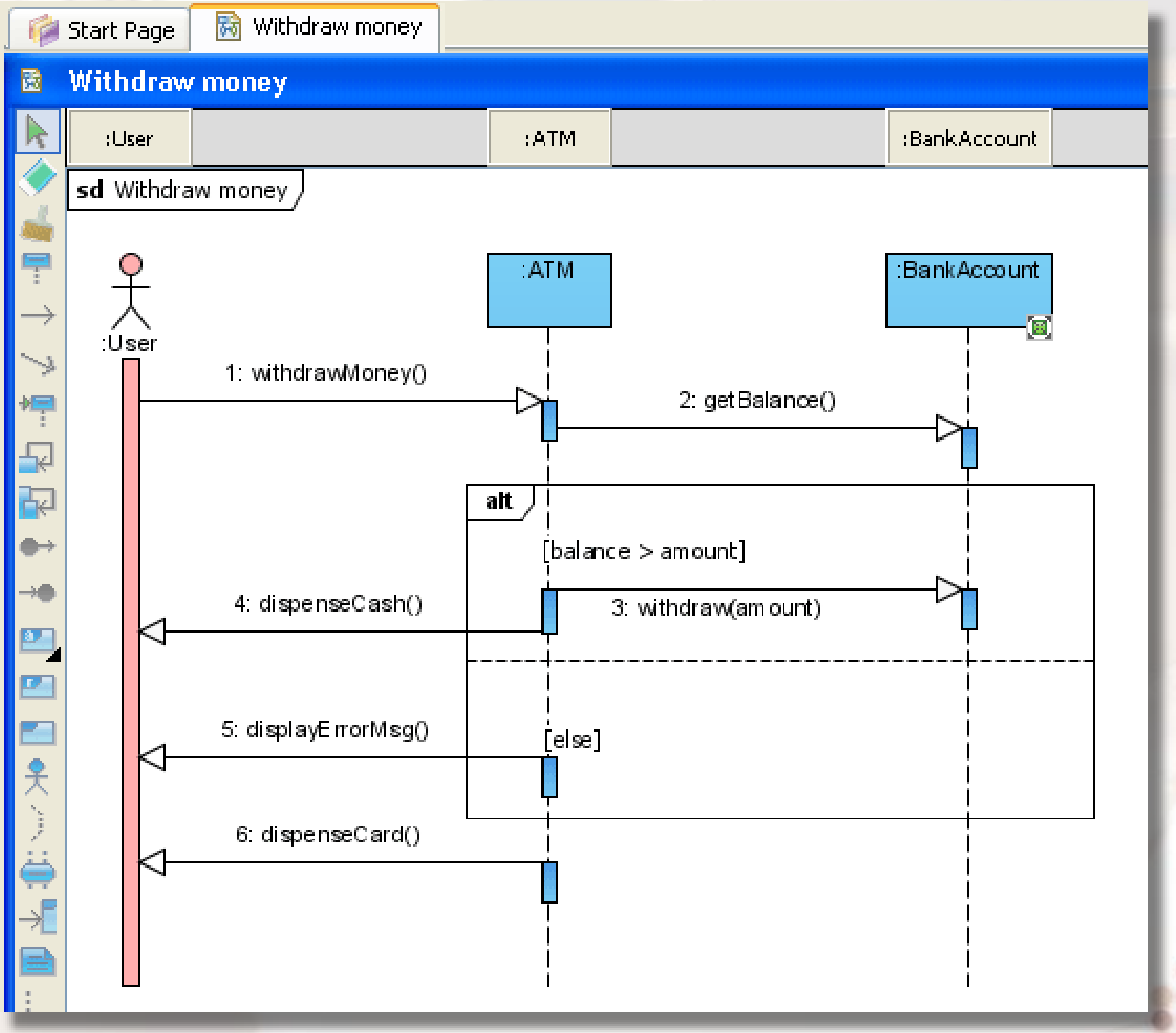
Interaction Diagrams: Sequence Diagram

- Shows the sequence of the system, ordered by objects and time
- Indicates the lifespan of objects relative to those messages
- **Application:** Systems with tentative sequences

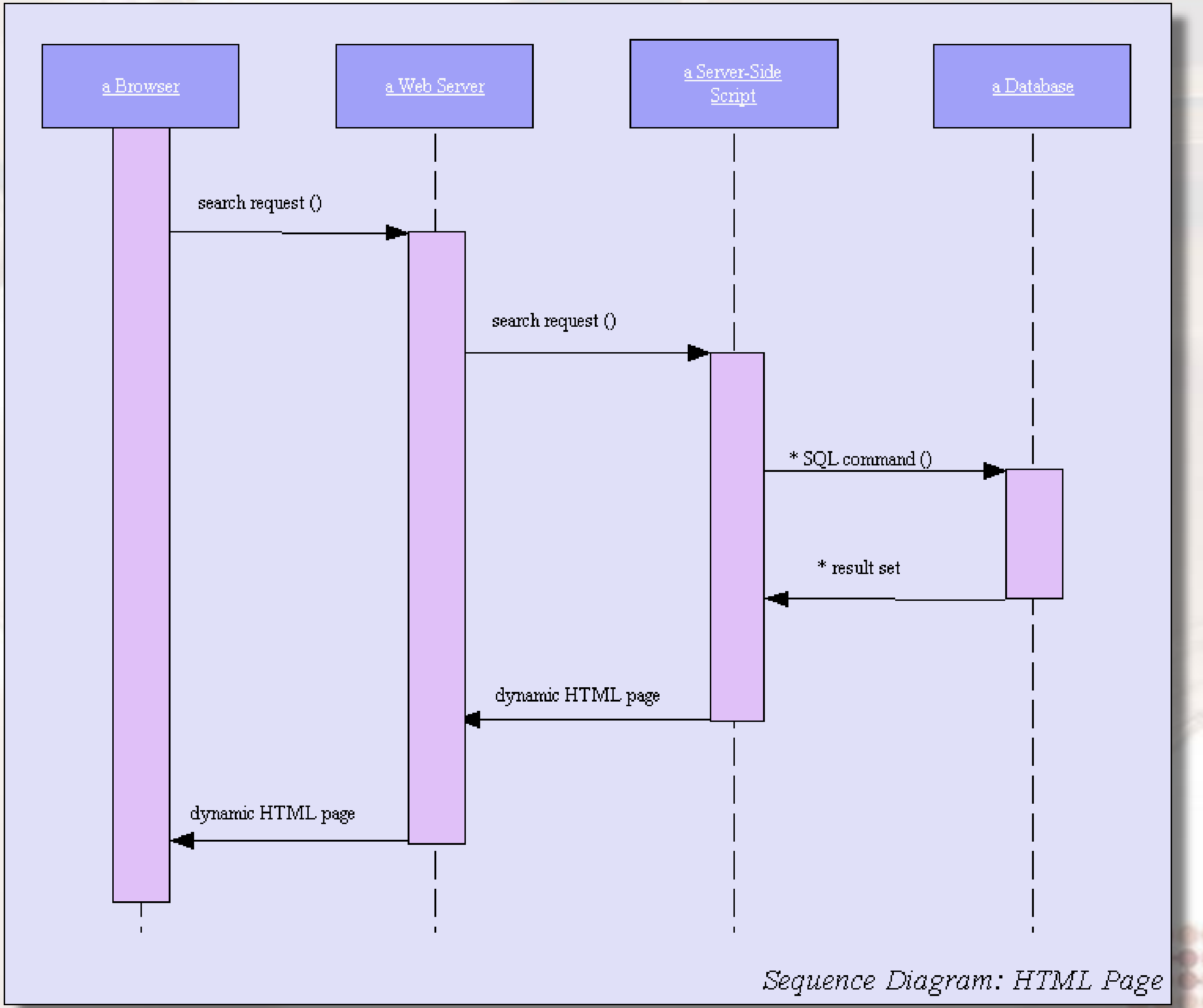
Interaction Diagrams: Sequence Diagram



Interaction Diagrams: Sequence Diagram



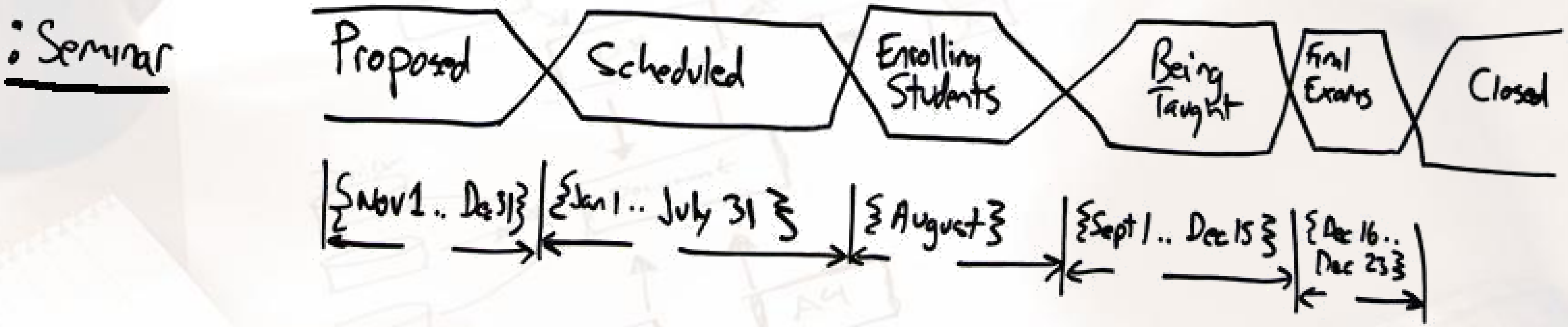
Interaction Diagrams: Sequence Diagram



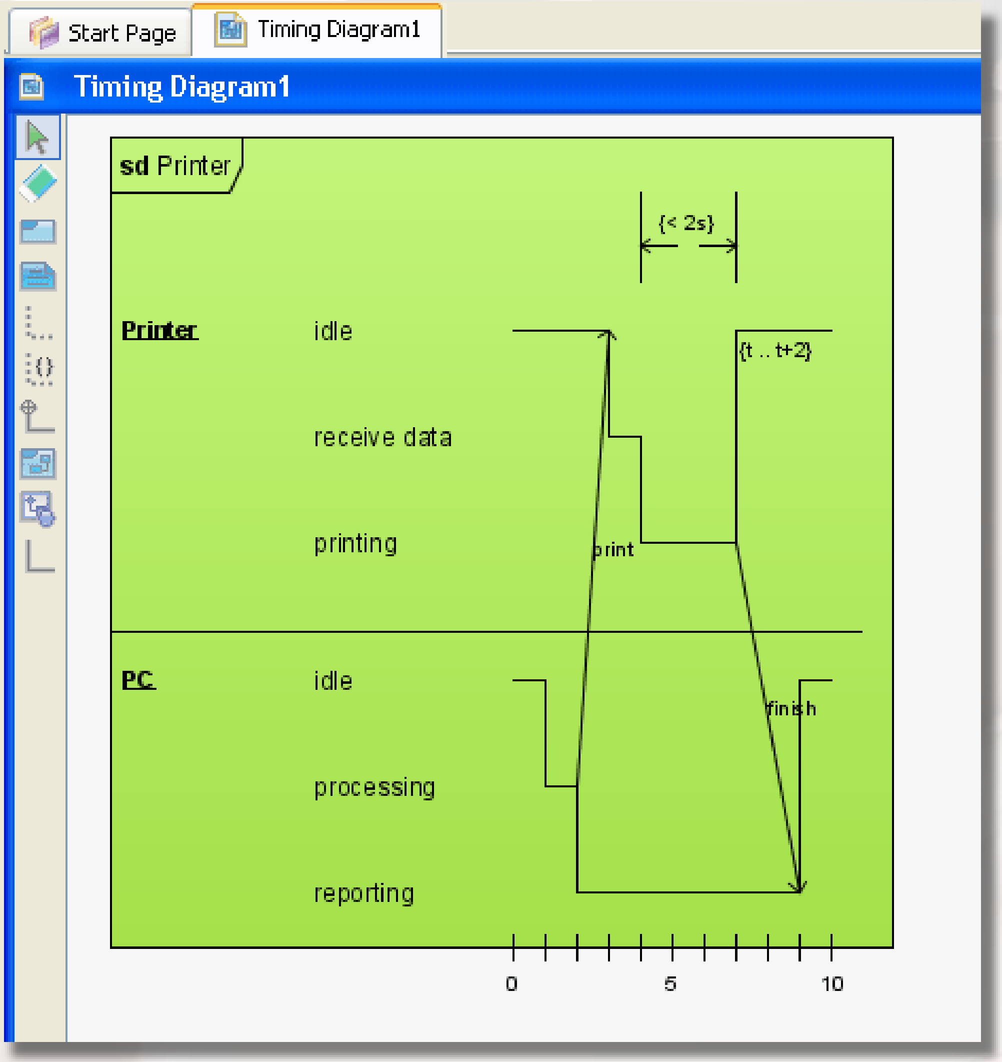
Interaction Diagrams: Timing Diagram

- Focuses on timing constraints
- **Application:** Processes which are time-oriented

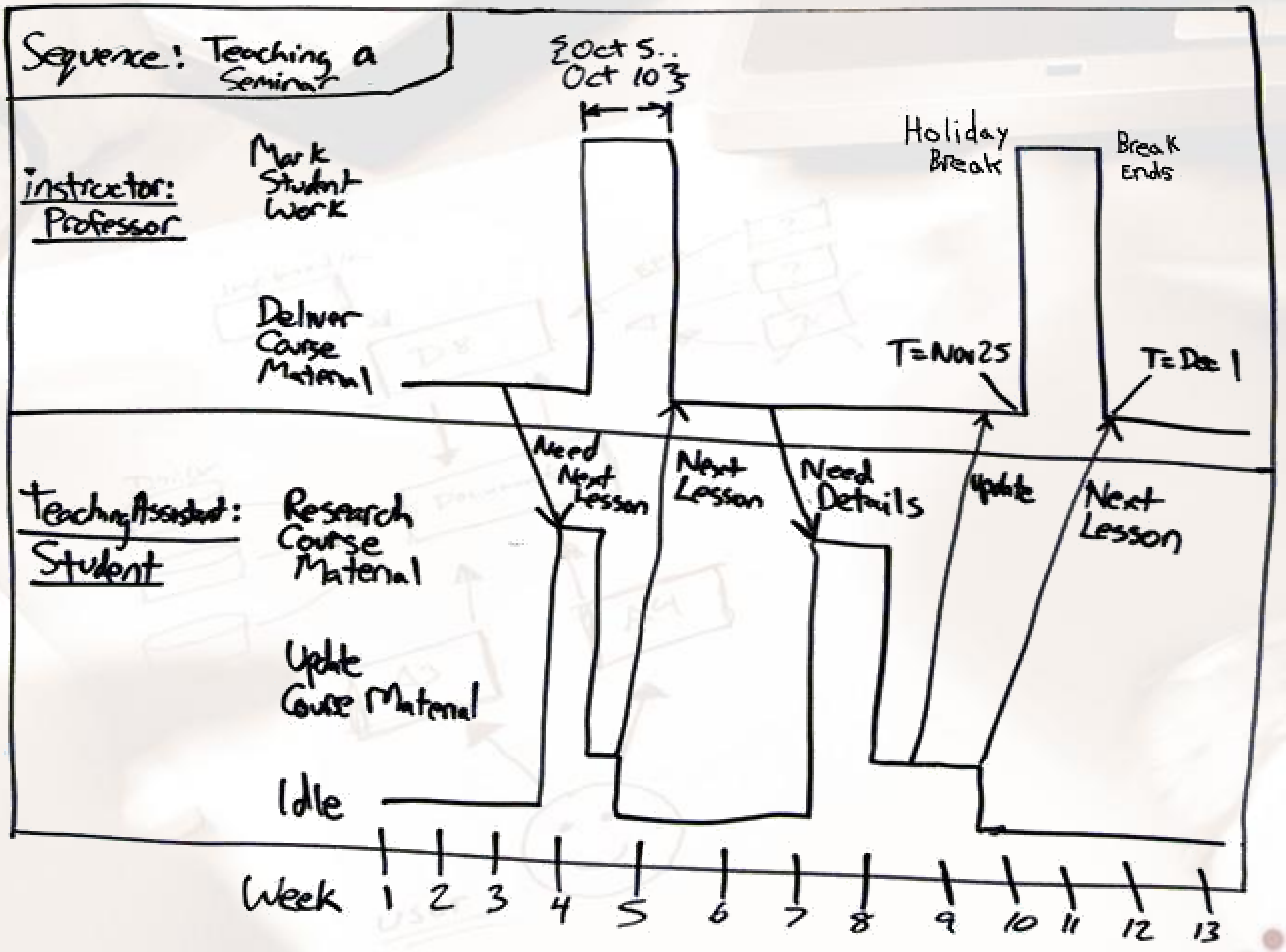
Interaction Diagrams: Timing Diagram



Interaction Diagrams: Timing Diagram



Interaction Diagrams: Timing Diagram



Summary

- Modeling is the process of representing the system, mostly in graphical notations, for understanding, designing and communication.
- Architectural design is the process of identifying sub-systems and its framework for controlling and communication.
- Software architecture is the output of the Architectural design process.

End of Lecture Questions

1. What is the purpose of each model below?
   - context model
   - structure model
   - Behavioral model
   - Interaction model
2. Which one of these should be included in a software requirements specification? Why?
   - Use case
   - State machine
   - Sequence
   - Timing

Activity

- Each group project creates the following model and post as a reply to the corresponding activity on Mango.
  - A use case model
  - A another UML diagram your group considers necessary to understand system's requirements and should accompany your SRS.