



Introduction to Software System Analysis and Design

Part 1



About the Course

- **Course Materials**

- My over 20 years of IT technology strategy and business application software development experiences
- Business application project management experiences
- Software industry best practices

- **Classroom Style**

- Lecture
- Discussion
- Practice
- Questions and answers

- **Exams**

- Test (30% of total grade)
- Design project (70% of total grade)



Topics Covered in This Course

- **Software Development Lifecycle**
 - Development Methodology
 - Software Engineering Goals and Roles

- **Project Development Planning**
 - Project scope
 - Project management

- **System Analysis**
 - Requirement Gathering
 - Use Case Modeling
 - Structural Analysis
 - Behavior Analysis

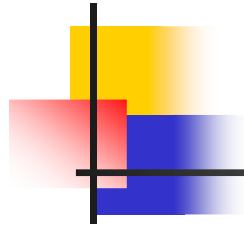
- **System Design Approach**
 - System Architecture Design
 - User Interface Design
 - Business Logic Design
 - Persistence Data Design

- **Implementation**
 - Programming construction
 - Testing



Course Schedule

Time	6/29 (Monday)	6/30 (Tuesday)	7/1 (Wednesday)	7/2 (Thursday)	7/3 (Friday)	7/6 (Monday)
8:00 am – 8:45 am	Self Introduction Part 1 Introduction - About the course	Part 5 System Analysis - Analysis Process - Business and technology system	Part 8 Analysis Modeling - Concepts - Techniques - UML	Classroom Test Part 11 System Design Approach - Design and development strategy	Part 16 Data Persistence Design - Data Persistence and Database Classroom Workshop - Student Practice	Final Project Report Student Project Presentation (Student course grading)
8:55 am – 9:40 am	Part 1 Introduction - Software system development lifecycle	Part 5 System Analysis - Current System - Business process automation Business Process reengineering	Part 9 Structural Analysis - Conceptual system structures	Part 12 System Architecture - System architecture Layers - Architecture decisions	Design and Development Reviews - Basic system design review	(Same as above)
9:50 am – 10:35 am	Part 2 Development methodology - Information Engineering - Object-oriented - Component-based	Part 6 Requirement Gathering - Get requirements - Joint Application Design	Part 10 Behavior Analysis - System interactions for business data and processing -	Part 13 Application Design - Concepts - Technique - Applying concepts	Part 17 Implementation and Testing - Programming Testing	(Same as above)
1:00 pm – 1:45 pm	Part 3 Software engineering Goals and Roles - Roles in business system - Roles in technology system	Part 7 Use Case Modeling - Concepts - Technique	Classroom workshop - Student Practice	Part 14 User Interface Design - User interface design process - Design Consideration	Part 18 - Final Course Review	(Same as above)
1:55 pm – 2:40 pm	Part 4 Project Development Planning - Project Initiation Project Management	Part 7 Use Case Modeling - Applying concepts Classroom Workshop - Student Practice	Analysis Reviews - Basic system analysis review	Part 15 Business Logic Design - Application logic - Component dependency	Student Project Assignment - Complete during 6/28-6/29 - Submit Final Report by Monday Morning 6/30	(Same as above)



Grading

Test	<u>30</u>
Final Project	<u>70</u>
Total	<u>100</u>

Grading

Pass

86-100	A
75-85	B
60-74	C

Not Pass

< 60	F
------	---



Course Objectives

- What will you learn from this course?
 - Concepts about system architecture
 - Concepts about project management
 - System analysis and design concepts
 - System analysis and design techniques

- What is the value to you?
 - Gain general knowledge about software engineering
 - Learn about system analysis and design skill
 - Improve key design communication skill
 - Increase your system thinking skill
 - More importantly, prepare yourself with knowledge and ability for the future



Main Learning Objectives

We will take a top-down approach for this course with the following subjects as the main learning objectives:

- **Software system analysis and design**
 - Software analysis and design concepts
 - Analysis and design artifacts

- **Software system development lifecycle**
 - Fundamental analysis and design process
 - Software system architect basic skills

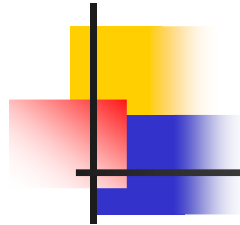


What is “System Analysis and Design”?

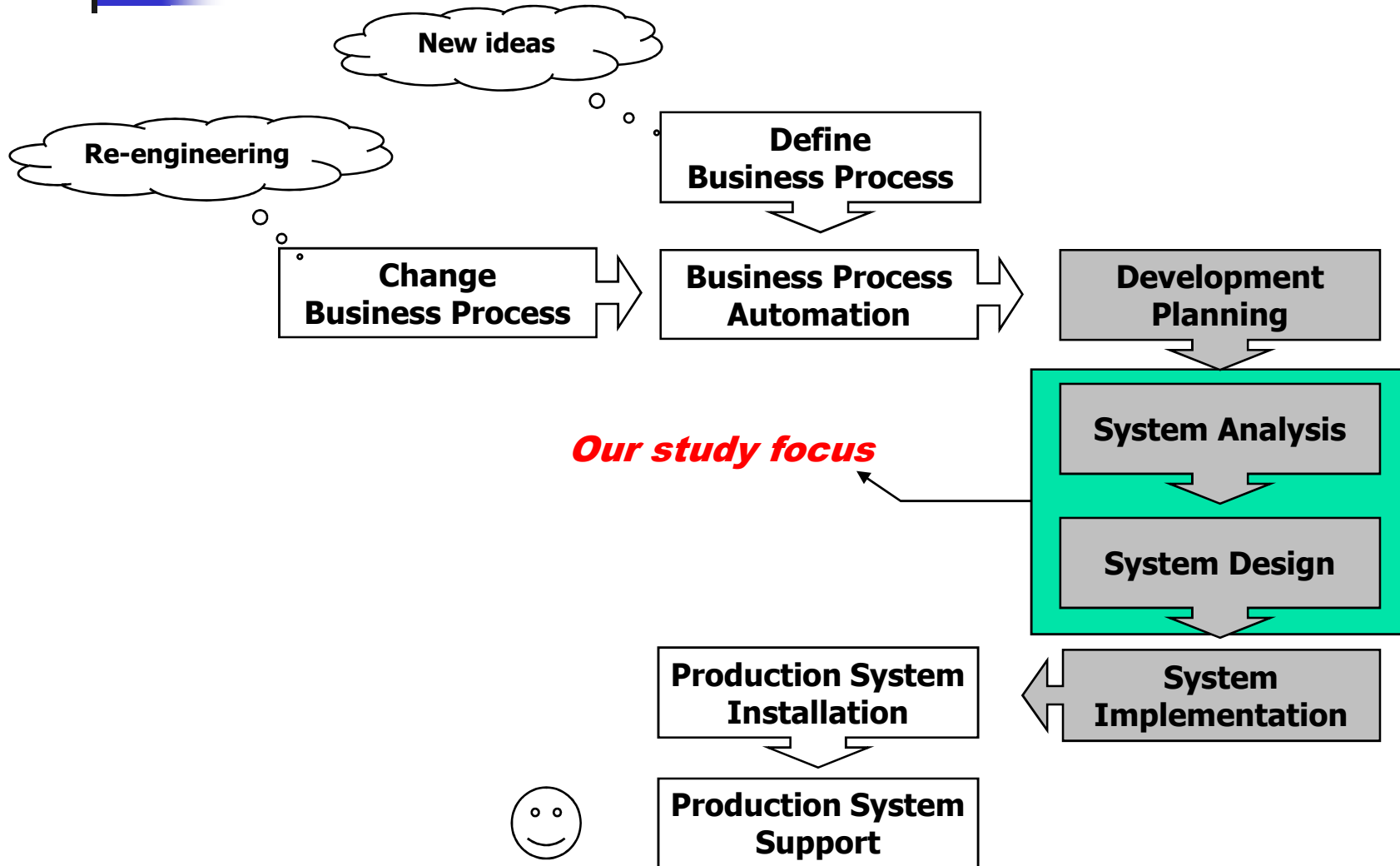
- Technology solutions for business
 - Understand the business needs
 - Enable business with technology solutions

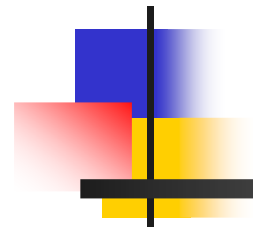
- Software engineering
 - Follow a systematic development process
 - Communicate artifacts of system transformation

- Technology automation creation
 - Define system architecture
 - Create design models



Conceptual View of System Development Process





System Development Methodologies

Part 2



What is Software System Development Methodology?

A Software System Development Methodology should:

- describe an approach for software system development.
- guide development teams using common steps.
- define consistent artifacts (e.g. design models).
- support a system development lifecycle.
- support a team development environment.
- be supported by a standardized set of tools.

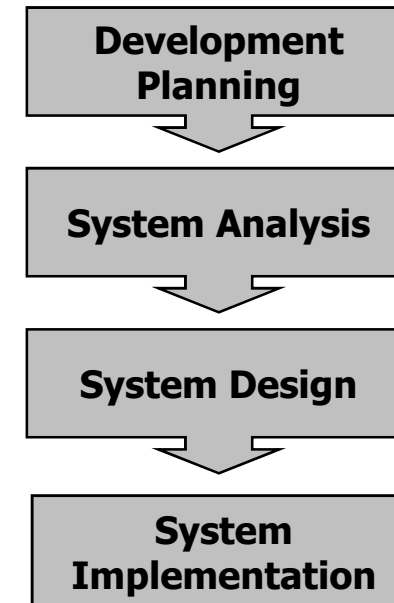


Different Software Development Methodologies

- Information Engineering Development Methodology
 - Water-fall methodology
 - Modular structure
 - Procedure processing
- Object-oriented Development Methodology
 - Iterative methodology
 - Class structure
 - Event driven and messaging
- Component-based Development Methodology
 - Component integration
 - Component reuse

Information Engineering Development Methodology

- **Pros:**
 - Simple process
 - Fixed requirements
 - Clear phase changes
- **Cons:**
 - Less flexible for changes
 - High degree module dependencies
 - High degree data dependencies



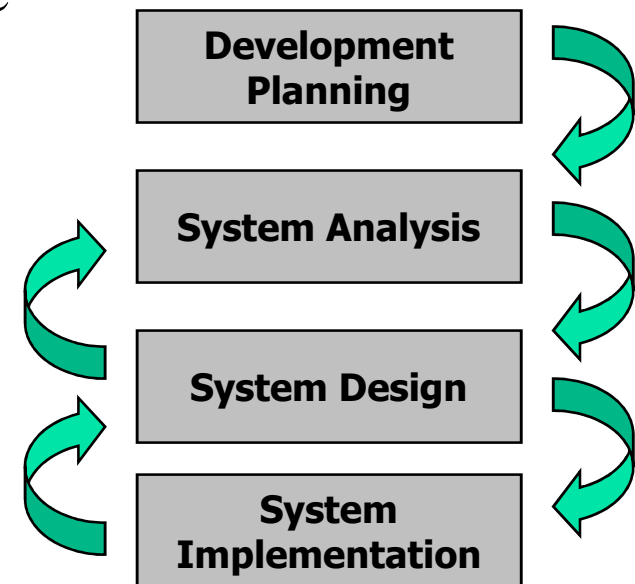
Object-Oriented Development Methodology

■ Pros:

- Iterative analysis and design process
- Flexible design changes with incremental requirement changes
- Promote functional and data independence
- High degree of development productivity
- Object reuse

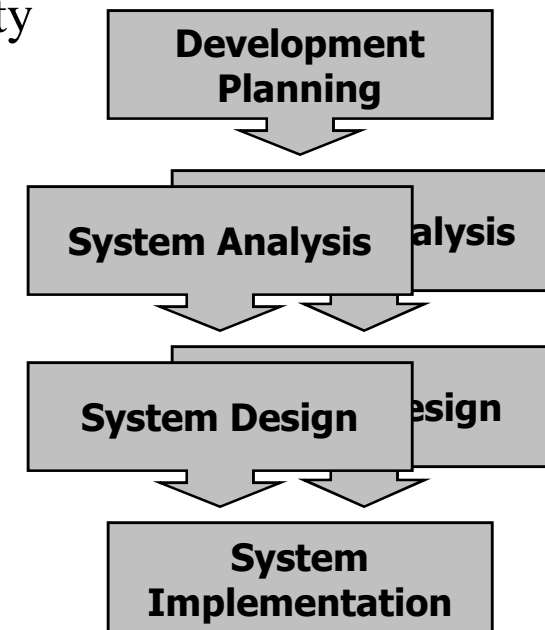
■ Cons:

- Difficulty of managing project increases
- Software complexity increases



Component-based Development Methodology

- **Pros:**
 - Flexible design changes with incremental requirement changes
 - Promote functional and data independence
 - High degree of development productivity
 - Requirements managed independently
 - Clear development boundaries
 - Parallel development
 - Component reuse
- **Cons:**
 - Complex component development management process





System Development Lifecycle – Planning

Phase	Step	Technique	Artifacts
<u>Planning</u> <i>(Why build the system?)</i>	Identifying Business Value	System request and dependencies	Business request and requirements
	Analyze feasibility	Technical feasibility Economic feasibility Organizational feasibility	Feasibility study
	Develop work plan	Task Identification Time estimation	Work plan
	Staff the project	Creating a staffing plan Creating a project charter (goals and plan)	Staffing plan Project plan
	Control and direct project (throughout the project)	Refine estimates Track tasks Coordinate project Manage scope Mitigate risk	GANTT chart CASE tool Standards list Project binder(s) Risk assessment



System Development Lifecycle - Analysis

Phase	Step	Technique	Artifacts
<u>Analysis</u> <i>(Who, what, when, where will the system be?)</i>	System analysis	Problem analysis Benchmarking Reengineering	Analysis report
	Information gathering	Interviews Questionnaires	Requirement information
	Use case modeling	Use cases Use case models Activity diagram	Functional models (business automation)
	Structural modeling (static)	Class diagrams	System structural models
	Behavioral modeling (dynamic)	Sequence diagram Collaboration diagram Statechart diagram	System interaction models



System Development Lifecycle - Design

Phase	Step	Technique	Artifacts
<u>Design</u> <i>(How will the system work?)</i>	System design	Custom development Package development Outsourcing	Design strategy
	Network design	Hardware design Network design	Technical architecture Infrastructure design
	Interface design	Interface structure Input design Output design	Interface design
	Database design (including any files)	Data schema Data storage	Physical data structure/table Data storage design
	Object design (refined physical design)	Class diagrams Sequence diagrams	Program structure chart Program specifications



System Development Lifecycle - Implementation

Phase	Step	Technique	Artifacts
<u>Implementation</u> <i>(System delivery)</i>	Construction	Programming Testing	Programs Test plan
	Installation	Direct Online Creating Training plan	Configured system Training plan
	Support	Support strategy Post-implementation review	Support/service plan



Software Engineering Goals and Roles

Part 3



Software Engineering Goals

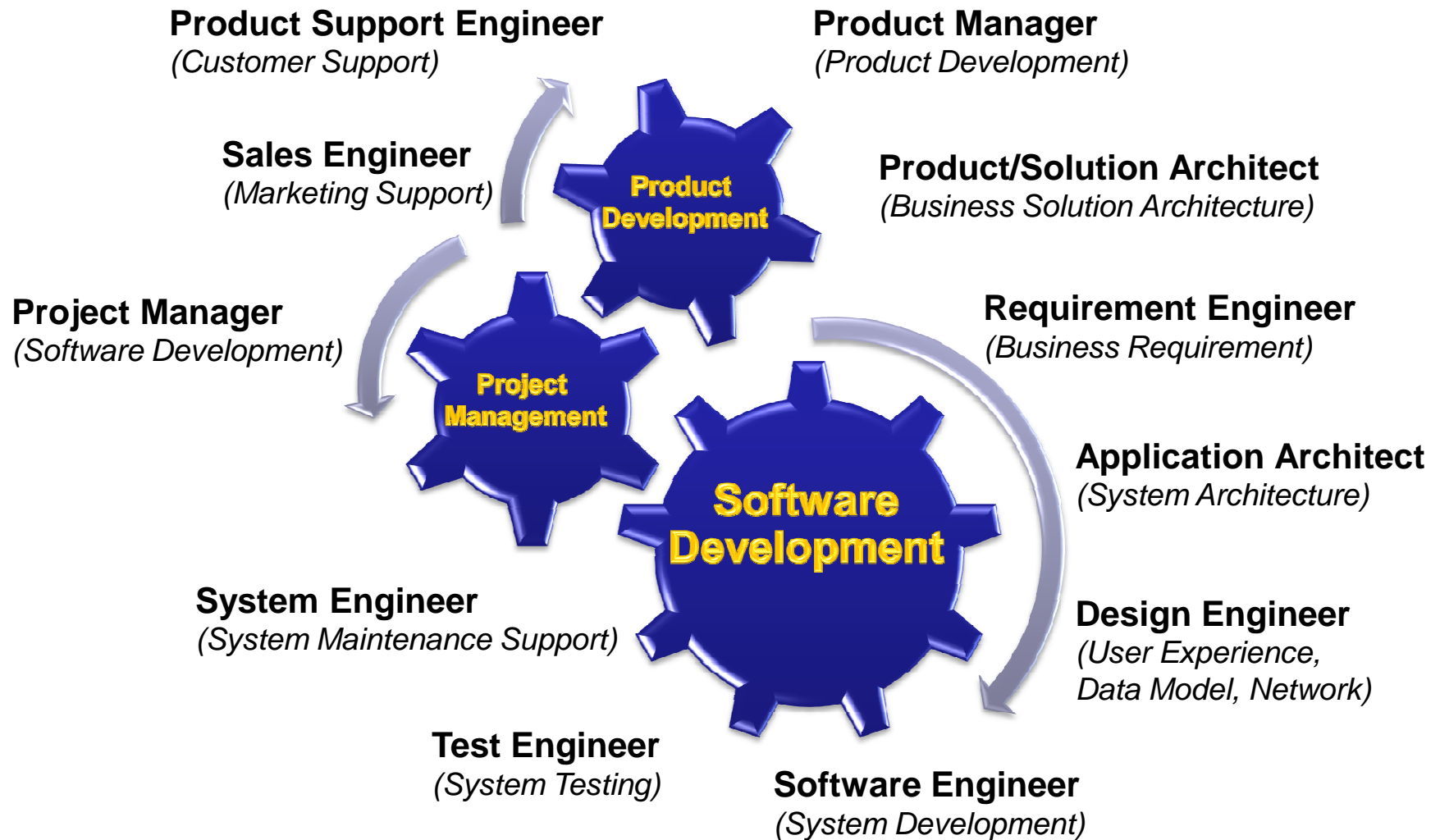
- **Productivity**
 - To easily develop and modify system designs with efficient and effective development methodology
- **Extendibility**
 - To minimize the dependencies between data and functions to increase flexibility of making modifications
 - To design modular functions for easy of adding new business functions
- **Reusability**
 - To reuse data and functions for other business opportunities



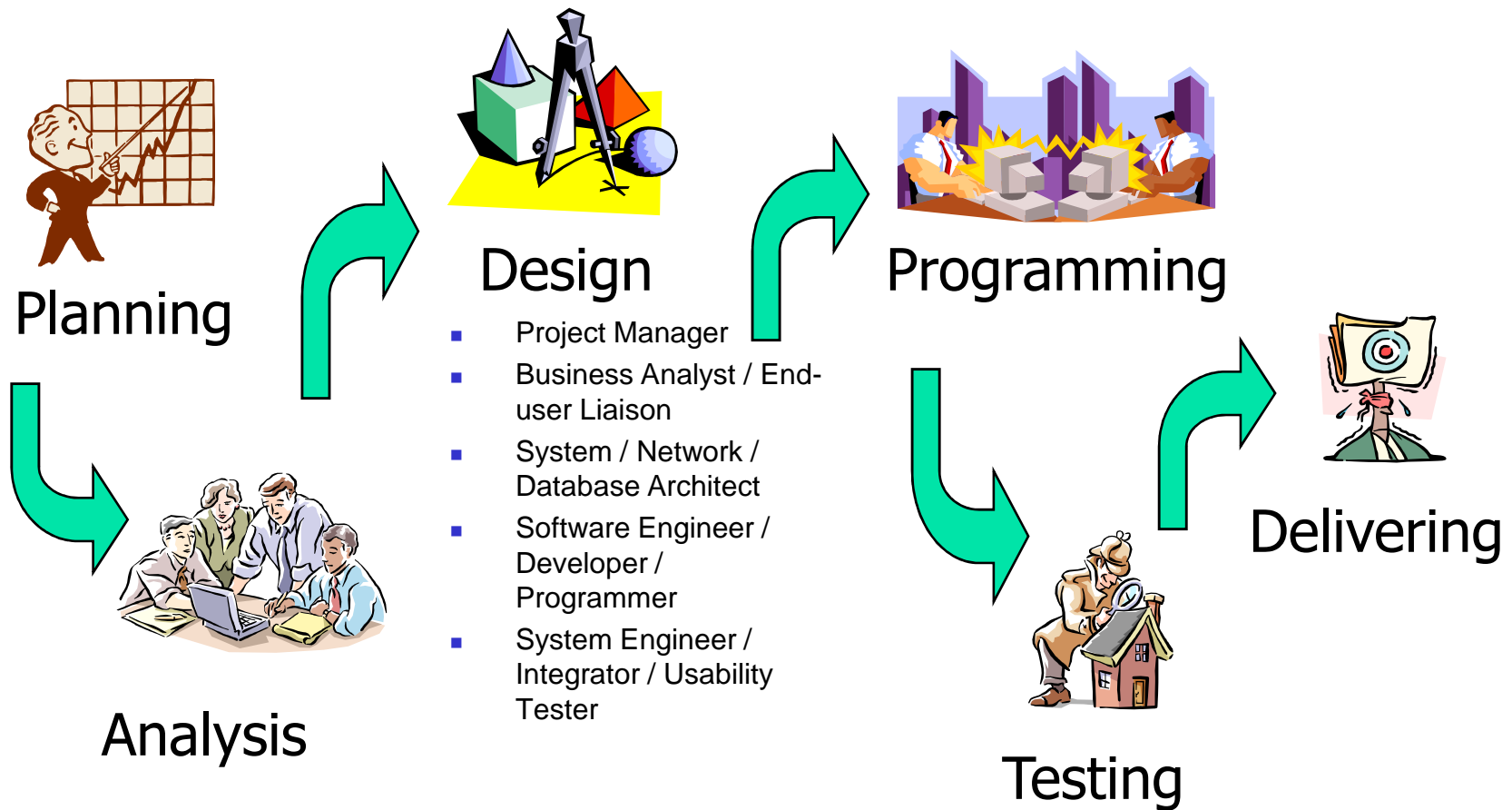
Software Engineering Key Design Considerations

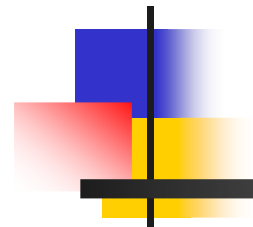
- **Modularity**
 - by data encapsulated with processing functions (i.e. methods)
- **Extendibility**
 - by object independence and inheritance
- **Testability**
 - by high degree of object independence Component integration
- **Maintainability**
 - by high degree of modularity
- **Reusability**
 - by reuse of objects, components, and object-oriented frameworks

Lifecycles and Roles of System Development



Team Work with Software Development Process





Project Development Planning

Part 4



Project Development Planning

- **Project Initiation**
 - Identify business value
 - Feasibility analysis

- **Project Management**
 - Develop project plan
 - Staff the project
 - Manage the project



Identify Business Value of using Technology

- Project sponsors
 - Business owners
 - Customers

- Business needs
 - New business capabilities
 - Business re-engineering

- Functionality
 - Business functions
 - Business processes



Identify Business Value of using Technology (continued)

- Expected value
 - Financial gains
 - Increasing market shares
 - Customer satisfactions

- Special issues/constrains
 - Business limitations
 - Regulation limitations
 - Competitions in the marketplace
 - Market supply/demand limitations



Feasibility Analysis

- Technical Feasibility
 - i.e., can we build the system?*
 - Familiarity with business application
 - Familiarity with technology to be used
 - Manageable project size



Feasibility Analysis (continued)

- Economical feasibility
i.e., is it worth to build the system?
 - Cash flow
 - Pay for the development resources (people, hardware, software, etc.)
 - Total Cost (TC) of years
 - Development costs
 - Operating costs
 - Total Benefits (TB) of years
 - Reduced cost of labor reduction and efficiency
 - Additional revenues generated
 - Total Net Benefit (TNB) = $TB - TC$
 - Return on Investment = TNB/TC , (*ROI is % value*)
 - Net Present Value = $TNB / (1 + \text{interests})^{\text{years}}$
 - Intangible costs and benefits
 - Market presences
 - Reputation
 - Brand recognition



Example

	Year 1	Year 2	Year 3	Total
New Development Cost	\$2500 (software and labor)			\$2500
New System Support Cost (new development)		\$4000 (people and system)	\$3500 (people and system)	\$7500
New Business (benefits)		\$200	\$300	\$500
Current System Support Cost (no change)		\$5000	\$5000	\$10000

- $TNB = TB - TC$
= Cost Saving + new business – initial cost of development
= $[(\$10000 - \$7500) + \$500] - \$2500 = \$500$
- $ROI = 500/2500 = 20\%$
- $Net\ Present\ Value = \$500 / (1+4\%)^3 = \444.5 (value of today)



Feasibility Analysis (continued)

- Organizational Feasibility
i.e., who will support the system?
 - Project champion (e.g. key business or technology management)
 - Senior management (e.g. company's leadership management)
 - Users (e.g. customers)
 - Other stakeholders (e.g. internal and external business partners)

Develop Project Plan

■ Identifying tasks

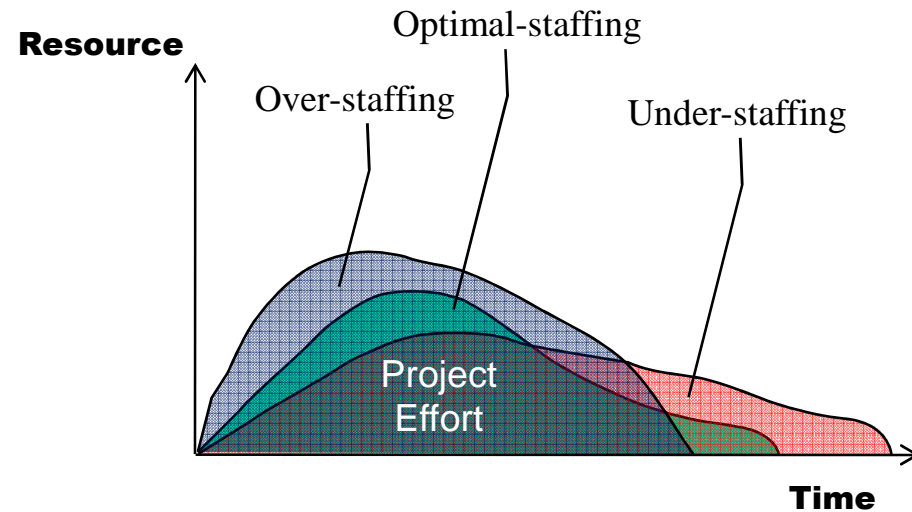
- Activities
- Deliverables
- Task hours
- Assignments

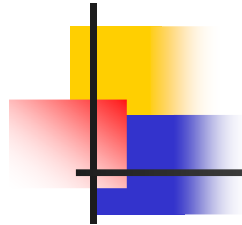
■ Time estimate

- Project milestones
- Resource planning (i.e., when/how many people on project)

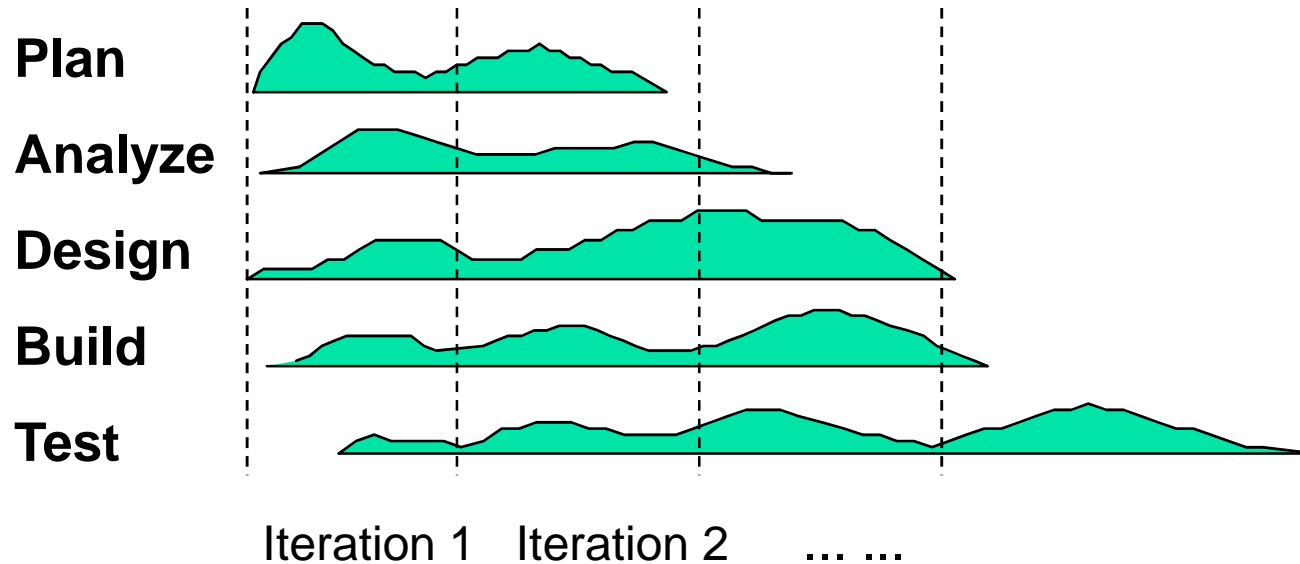
■ Create an overall project plan

- Project effort staffed with appropriate resources:
 - Resource (number of staffs) matched Time (milestones)
- Overhead increased situations (project costs increased)
 - Under staffing (project time will be longer, overhead increased)
 - Overstaffing (project productivity reduced, overhead increased)

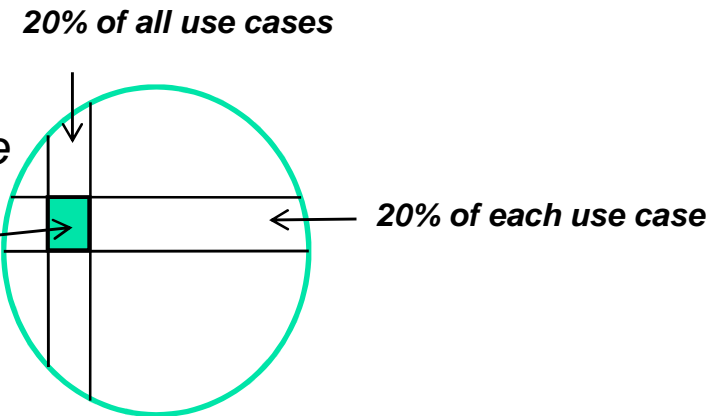




Iterative and Incremental Process



*Project should start with 20% of each use case
out of 20% of all use cases
= 4% total development effort*





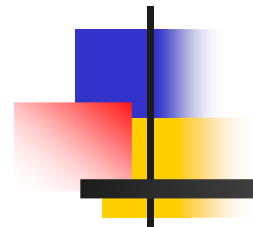
Staff the Project

- Staffing plan
 - Team members (what are the roles and the needs)
 - Skill requirements (what expertise the project team needs)
 - Project team structure (who plays what role)
 - Number resources according to the project plan schedule



Project Management Activities

- Manage the scope
 - What is (or is not) to be delivered
- Refine the estimates
 - When will tasks be done
- Track the tasks
 - Who is doing what
- Coordinate the project
 - Who is responsible for what to happen and when
- Mitigate the risks
 - How can contingency plans reduce the failure risk of the project



System Analysis

Part 5



About System Analysis

- Purpose
- Analysis Process
- Business System and Technology System
- Business process automation
- Business reengineering



What is the Purpose of Analysis?

- Capture the business requirements with models
- Transform the business needs into technology implications
- Understand what functions to be built for the business
- Model the business information for future design solutions



Analysis Process

- Understand as-is system (current environment)
 - Understand the current system
 - Capture the current business and technology environment
- Identify improvement opportunities (changes needed)
 - What are problems that should be solved
 - What are the priorities
 - What are the cost-effective opportunities
- Develop to-be system (future environment)
 - Revise the as-is system
 - Modify existing and add new processes
 - Modify existing and add new data
 - Model and recommend the to-be system

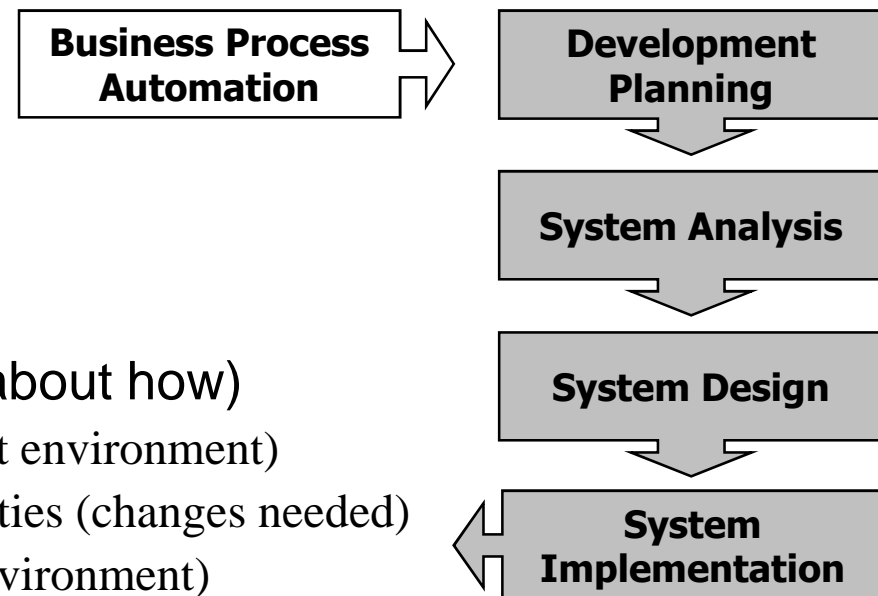


Business System and Technology System

- What is a business system?
 - A depiction of a real world business including:
Business process, Business data, Business organization,
Business operation rules, Business policies, etc.
- What is a technology system?
 - A depiction of a technology system (solution) for a business system including:
System process, System data, System structure, etc.
- Technology systems support the business systems and automate the business processes.

Business Process Automation

- Automate the business with technology (why to change)
 - Change from manual to automation – faster (e.g. on-line banking vs. paper)
 - Use technology to improve business tasks – more accurate
- Analysis methods to determine an automation area (where to change)
 - Duration analysis
 - Activity-based costing
 - Informal benchmarking
 - Formal benchmarking
- Follow the analysis process (about how)
 - Understand as-is system (current environment)
 - Identify improvement opportunities (changes needed)
 - Develop to-be system (future environment)





Business Reengineering

- Change current business process
 - Increasing business value
 - Create business opportunities

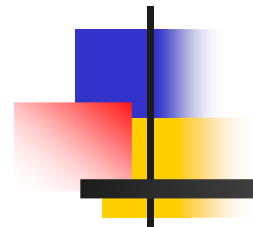
- Apply the analysis process to
 - Outcome analysis – change value produced for customers
 - Breaking assumptions – change reasoning/rules
 - Technical analysis – leverage technology
 - Activity elimination – reduce steps in the process
 - Proxy benchmarking – borrow ideas from different industry

Developing an Analysis Report

- Justify the cost for the benefit of business value



Analysis Objective	Business Process Automation	Business Process Improvement	Business Process Reengineering
<i>Potential Business Value of Return</i>	Low-Moderate	Moderate	High
<i>Project Cost</i>	Low	Low-Moderate	High
<i>Breadth of Analysis Needed</i>	Narrow	Narrow-Moderate	Very broad
<i>Risk of Change</i>	Low-Moderate	Low-Moderate	Very high



Requirement Gathering

Part 6



Business Requirement Gathering

Selecting appropriate techniques to gather business requirements:

- Document Analysis
 - Based on the exiting documentations
- Questionnaires
 - Information in writing from individual(s)
- Interviews
 - Information from meeting of individual(s)
- Joint Application Design (JAD)
 - Information from meeting of group(s)



Document Analysis

- Research existing documentation
 - Business process and data
 - Business management

- Synthesizes the information
 - Findings
 - Identify gaps

- Make recommendations as requirement
 - Changes needs
 - Suggestions



Questionnaires

- Selecting Participants
 - Business representatives
 - Subject matter experts

- Design the Questionnaires
 - Categorize the questionnaires
 - List important items
 - Straight forward and unbiased
 - Set them at a right level without ambiguity

- Administrating the Questionnaire
 - Get participants to complete questions
 - Explain why the questionnaire is conducted
 - Clarify any questions

- Follow-up
 - Get back to the participants to make sure work completions



Interviews

- Selecting Interviewees
 - Name
 - Position
 - Purpose of interview
 - Meeting

- Designing Questions
 - Close-ended questions – Specific needs
 - Open-ended questions – Uncover needs
 - Probing questions – Confirm the needs' certainty or clarity
 - Level of questions:
 - High-level: very general
 - Medium-level: moderately specific
 - Low-level: very specific



Interviews (continued)

- Preparing for the interview
 - Plan - topic
 - Questions – purpose
 - Open-ended questions are easier to prepare than the close-ended
 - Possible answers – confirmation
- Conducting the interview
 - Build trust
 - Professional
 - Unbiased
 - Document accurately
- Follow-up
 - Written interview report
 - Confirm interview points
 - Clarify additional questions



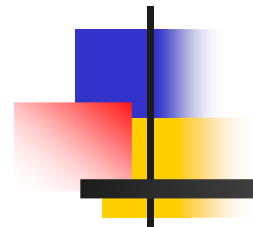
Joint Application Design (JAD)

- **Selecting Participants**
 - Who should join the JAD session
- **Designing the Session**
 - What topics to go over and get results
- **Preparing for the Session**
 - What materials to prepare before the session
- **Conducting the Session**
 - Facilitate the discussion and engage with the participants
- **Follow-Up**
 - Any additional information to get from the participants identified at the session to complete the tasks



Selecting Appropriate Techniques

Techniques	Document Analysis	Questionnaires	Interviews	JAD
Gathered information	As-is	As-is, Improvement	As-is, Improvement, To-be	As-is, Improvement, To-be
Depth of information	<i>Low</i>	<i>Medium</i>	<i>High</i>	<i>High</i>
Breadth of information	<i>Medium</i> (depends on availability)	<i>High</i>	<i>Low</i>	<i>Medium</i>
Integration of information	<i>Low</i>	<i>Low</i>	<i>Low</i>	<i>High</i>
User involvement	<i>Low</i>	<i>Low</i>	<i>Medium</i>	<i>High</i>
Cost	<i>Low</i>	<i>Low</i>	<i>Medium</i>	<i>Medium-High</i>



Use Case Modeling

Part 7



Modeling

- What does modeling mean?
 - General purposes
- What is to model?
 - Subject contents
- Use case modeling
 - Concepts
 - Structures
 - Development



What does Modeling Mean?

Modeling is to depict the real world information with artifact (representations) so that ...

- We capture meaningful information in the scope of interests
i.e. Useful model with relevant information
e.g. Business processes, technology systems
- We know how to utilize the information for developing solutions
i.e. Meaningful model with purpose
e.g. Designs for building system
- We can communicate and share it with development team
i.e. Understandable model with communicable notation
e.g. Graphical and textual descriptions for users to understand



What is to Model?

- Model what is in the context
e.g. business rules, technology constrains
- Model information at a right level
e.g. analysis level, design level
- Model complex information with different level of abstractions
e.g. subsystems, packages



Use Case Modeling

What is Use Case?

“A use case is a sequence of transactions in a system whose task is to yield a result of measurable value to an individual actor of the system.”

Ivar Jacobson

In other words

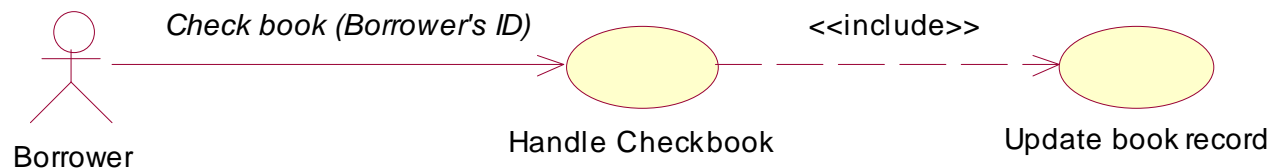
A use case describes a sequence of “system” activities from its user’s perspective.

Therefore

- Use cases describe how a system will be used
- Use cases specify what business requirements are
- Use cases define system’s functionality and scope
- Use cases provide specifications of a system capability
- Use cases create basis for system testing

Use Case Modeling Concepts

- What is use case for?
Define what is to be performed by the “system”.
- Who develops use case?
Jointly developed by the end-users and the “system” development team.



Library System



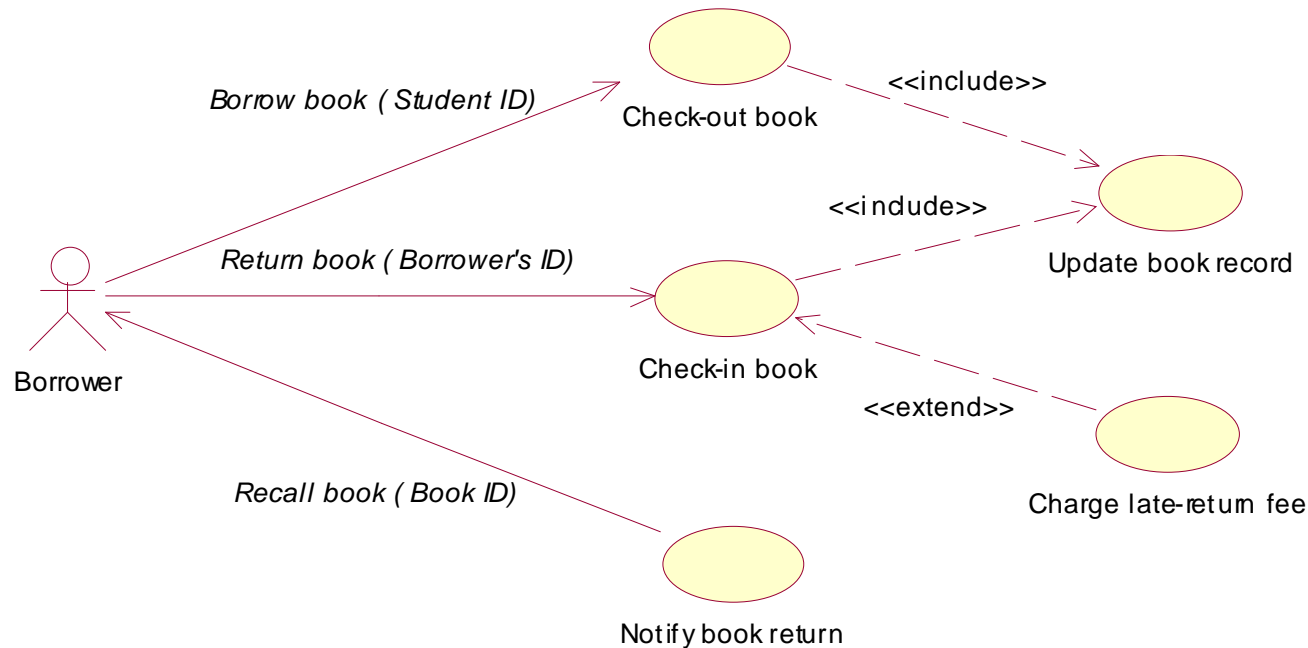
Use Case Model Structure

- Use Case Model Contains:
 - Use case diagram (graphical representation)
 - A graphical depictions of how a system is used in a particular context with the involved actors.
 - Actor
 - Role in the context of use case
 - Initiator or receiver of service
 - External Systems
 - Use case (textual description)
 - Use case event
 - Use case relationship
 - Communications: *between actors and use cases*
 - Includes: *between a use case and a shared use case by another use case*
 - Extends: *between a core use case and an extended use case (extension or alternate course of the core use case)*

Use Case Model Structure (continued)

■ Use Case Modeling Level

- Main use cases
- Detail use cases



Library System



More about Use Case

- Use case is an analysis technique, which can be used for different contexts of modeling “system” and at different levels (conceptual or logical). This system can be application, component, etc.
- Use case gives the abstraction of the system from the perspective of its “actor” (i.e., business users, applications, or components interfacing the system).
- Use case describes the process (activities) performed by the system. It is generally triggered by actor(s) along with some pre-conditions or post-conditions. Alternate steps maybe described if certain conditions are met. This may include any exception handlings.
- Use case diagram is a collection of actors, use cases visually depicting their relationships (e.g., actors trig use cases or use cases associate with other use cases).



Develop Use Case Model

- Identify the Main Use Case
 - Find the system' boundaries
 - List the primary actors
 - List the goals of the primary actors
 - Identify and write the main use cases
 - Review the main use cases
- Expand the Main Use Case to Detail Use Cases
 - Choose main use cases to expand (for further analysis)
 - Fill in details of the chosen use cases (becomes detail use cases)
 - Write the normal flow of the events of the use case
 - Complex or long flow can be sub-flows
 - Identify alternate or exceptional flows
 - Write the use case in terms of “who does what to whom” to provide clarity
 - Write specific condition, rules and any technical requirements as part of the activities



Develop Use Case Model (continued)

- Confirm the main use case
 - Review the main use cases to make any adjustments
 - Use packages to manage complex business functional requirements
- Create use case diagram along the way
 - Determine the system boundary
 - Place the use cases on the diagram
 - Place the actors on the diagram
 - Draw the associations
 - Use case events
 - Use case relationships



How to write a use case?

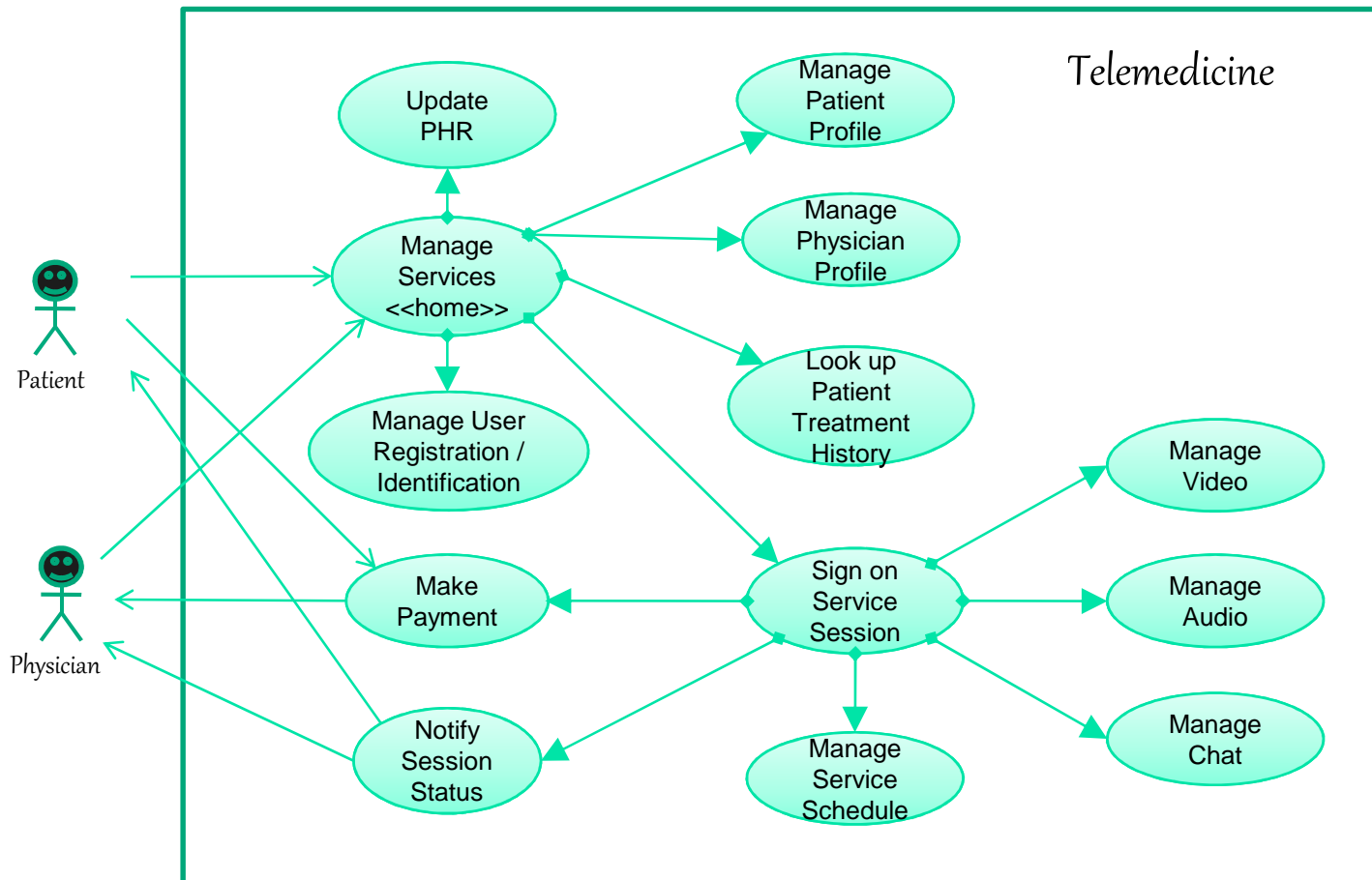
- Identify (discover) an appropriate level of use cases to write
 - Main use cases: *Abstract or generalized sequence*
 - Detailed/Expanded use cases: *Specific sequence*

- Start with an actor
 - A person who initiates the event
 - A system which triggers the event

- End with a completion of an activity thread performed by the system

Use Case Model Example – Use Case Diagram

Telemedicine System Use Case Diagram





Use Case Model Example – Use Case Description

Telemedicine System Use Case Descriptions

Manage Services <<Home>>

This is the home page for users to browse the general information about Telemedicine, which has links to the activities that patients and physicians can perform after having registered and sign on successfully. The intent is to have as much flexibility as possible in principle. The activities may be performed by the patients and physicians are independent until they are in a session. The following links will be shown on the home page to access medical service or medical data. They are described by each use case respectively:

- [Register](#)
- [Sign on Service Session](#)
- [Manage Patient Profile](#)
- [Manage Physician Profile](#)
- [Update Patient Treatment History](#)
- [Update Patient Health Record](#)

Manage User Registration / Identification

This use case manage users' registration and sign-on authentication:

- Register user
and/or
- Authenticate users at the beginning of accessing the services and data.



Use Case Model Example – Use Case Description

Telemedicine System Use Case Descriptions

Sign on Service Session

After successful sign-on verification, a patient will start a session for service. There are two types of sessions:

- On-demand: (request on the fly and waiting for a physician online to response)
- Scheduled (pre-schedule time slot with a physician signed up for, see use case [Manage Service Schedule](#))
- Start a session with a service request that will lead to [Make Payment](#) before a service can be granted.
- Once a successful payment is made, a session then can be started, including a set of physical conditions will be entered by the patient.
- In a case of On-demand:

Patient will be put on a waiting list until a doctor is sign on/response to the request.

The waiting list may be sorted by time, treatment need or by name. (go to step 3)

In a case of scheduled:

See use case [Manage Service Schedule](#)

Scheduled time can be modified any time hereafter for change.

•A session gets started when a doctor who has signed in and respond to a service request. When both the patient and the doctor are ready for the session with options available (they are described in each use cases respectively):

- 1.[Chat](#)
- 2.[Audio](#)
- 3.[Video](#)

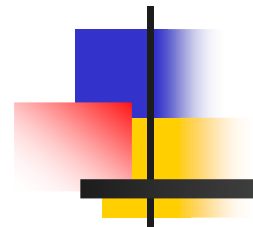
•A generic note of SOAP is created for the physician to capture the information pertain this service:

- Subject - the patient name
- Objective – Service request purpose
- Assessment - the record of diagnoses including text, image
- Plan – treatment plan and medication prescribed.
- The note will be saved in the system for future reference. Patient will be notified for information.

<<Option>>

•A physician may schedule a follow up of treatment services by opening the scheduler for future appointments, which put both physician and patient to a specific date/time through [Manage Service Schedule](#).

•Use case ends with session closed.

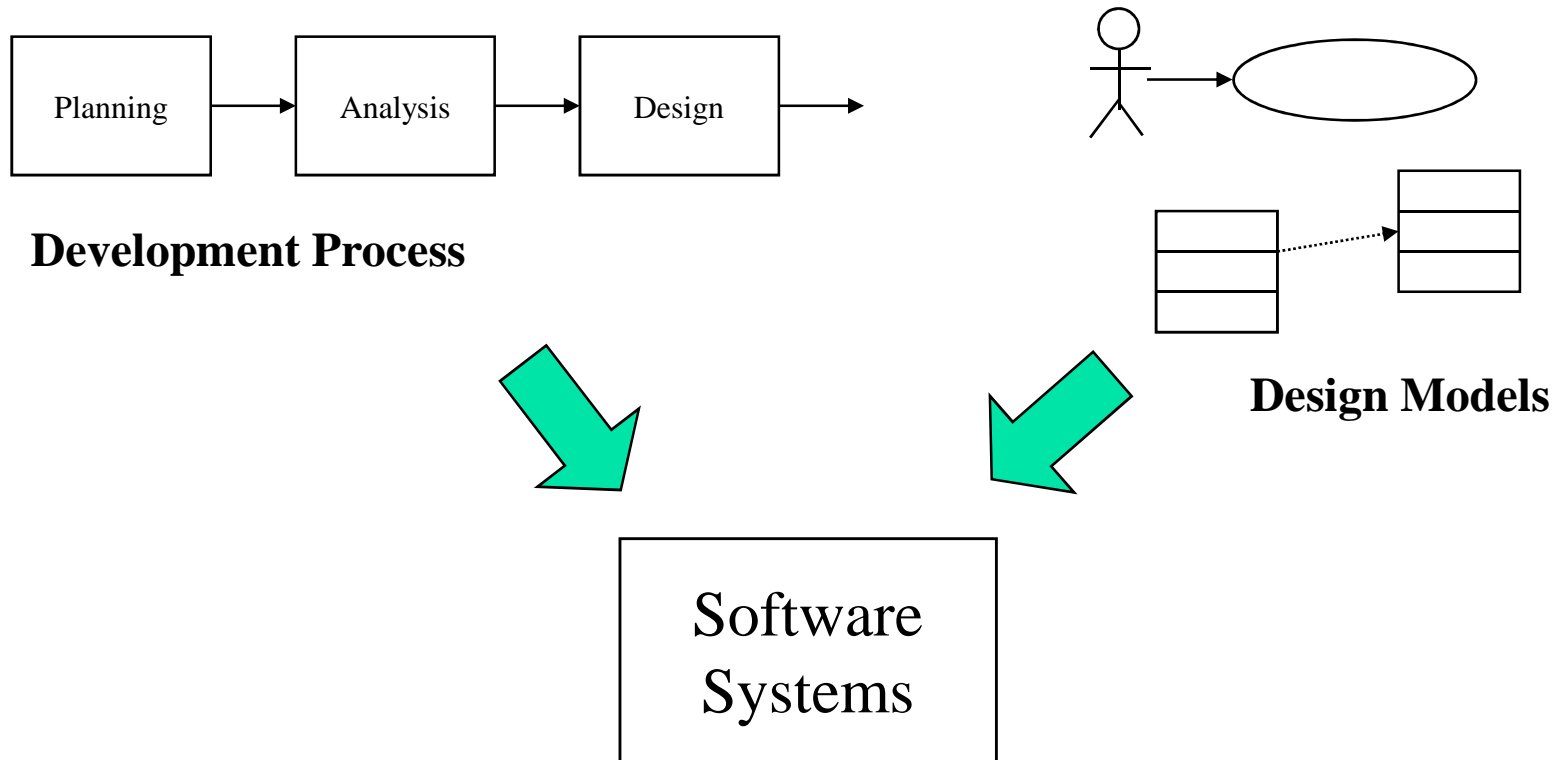


System Modeling with UML

Part 8

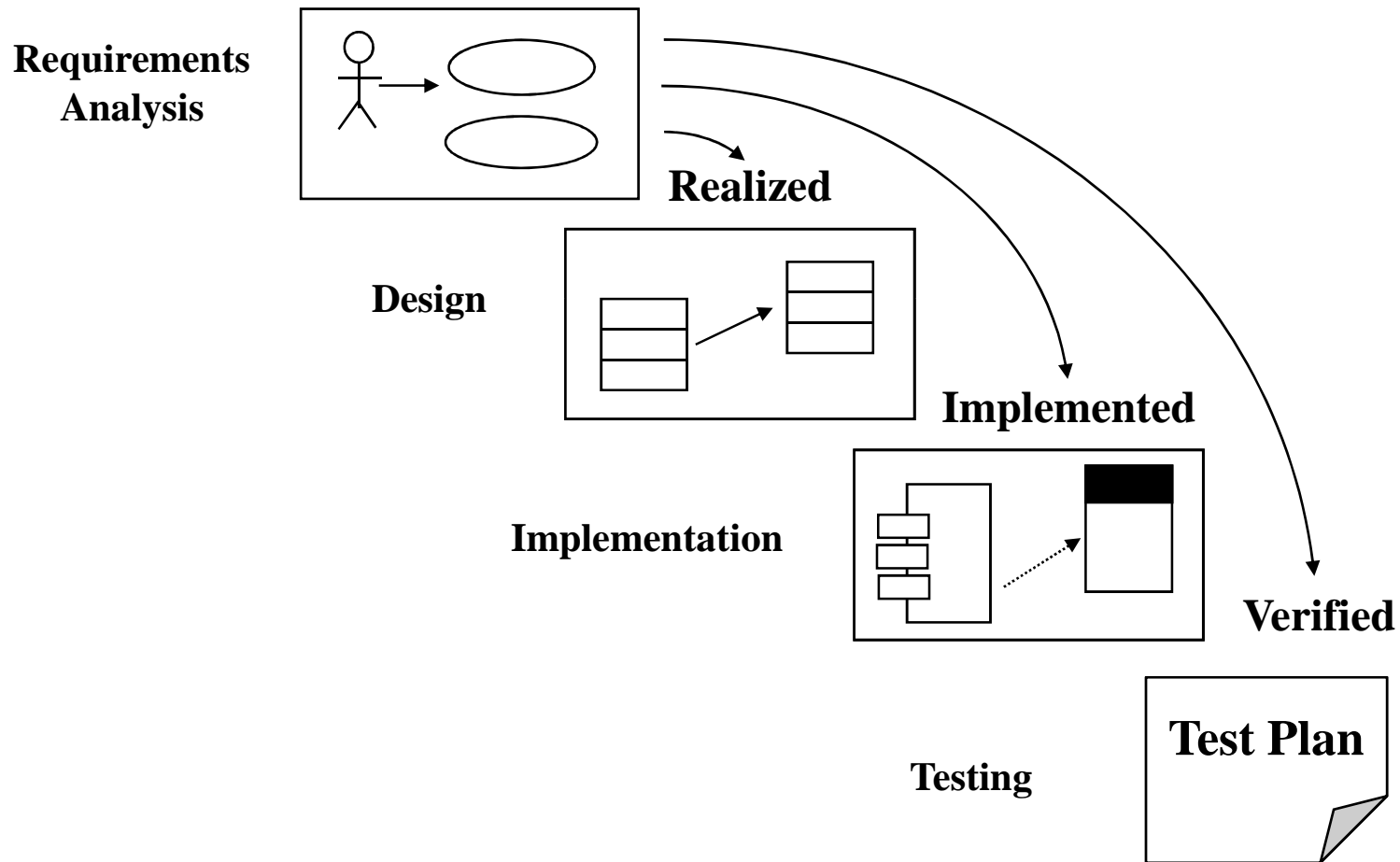
How do We Specify Software Systems?

- Need a software development process
- Need design models to describe the systems



What do We Use to Model Software Systems?

Unified Modeling Language (UML)





UML Offers Features for System Modeling

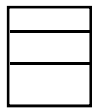
- **Model Elements**
 - Representations for system structures and interactions
- **Relationships**
 - Representations for system static and dynamic relationships
- **Diagrams**
 - Graphical models of static and dynamic information about the system
- **Common Mechanisms**
 - Annotations and user defined information
- **Architecture Views**
 - Models for different system perspectives



Model Elements in UML



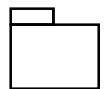
- **Use Case** - A way in which an external actor uses a system



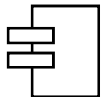
- **Class** - The definition of objects that share a common structure and common behavior



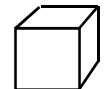
- **State** - A stage of lifecycle in which an object instance is in



- **Package** - A logical grouping of model elements



- **Component** - Physical implementation of a software unit



- **Node** - A hardware processor on which the software units execute



Relationships in UML

- Association - A semantic connection between two classes/instances of objects
- ▷ Generalization - A relationship between an element and the elements that specialize it
- Interface - Mechanism offered by an element for others to access its functions
- > Dependency - An element is need by another
- ◇ Aggregation - An elements is part of another element
- > Navigability - An unidirectional association
- 0..* Multiplicity - Specification of an (+) integer range of allowable cardinalities



Diagrams in UML

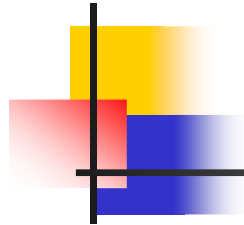
- **Use Case Diagram (UML 1.x and 2.x)**
shows the external view of how a system is used
- **Composite Diagram (UML 2.x)**
shows the system structural parts, ports and interfaces
- **Class Diagram (UML 1.x and 2.x)**
shows the class hierarchy and static relationships
- **Communication Diagram (UML 2.x)**
shows interaction among objects organized in spatial network
- **Sequence Diagram (UML 1.x and 2.x)**
shows interaction among objects organized in temporal order
- **Statechart Diagram (UML 1.x and 2.x)**
shows states and conditions that cause state changes of an object
- **Activity Diagram (UML 1.x and 2.x)**
shows activity flows or algorithms and controls of use cases or objects
- **Component Diagram (UML 1.x and 2.x)**
shows the implementation components in a system and their dependencies
- **Deployment Diagram (UML 1.x and 2.x)**
shows a system placement of nodes, networking and process distribution

Get more information about UML: www.omg.org/uml



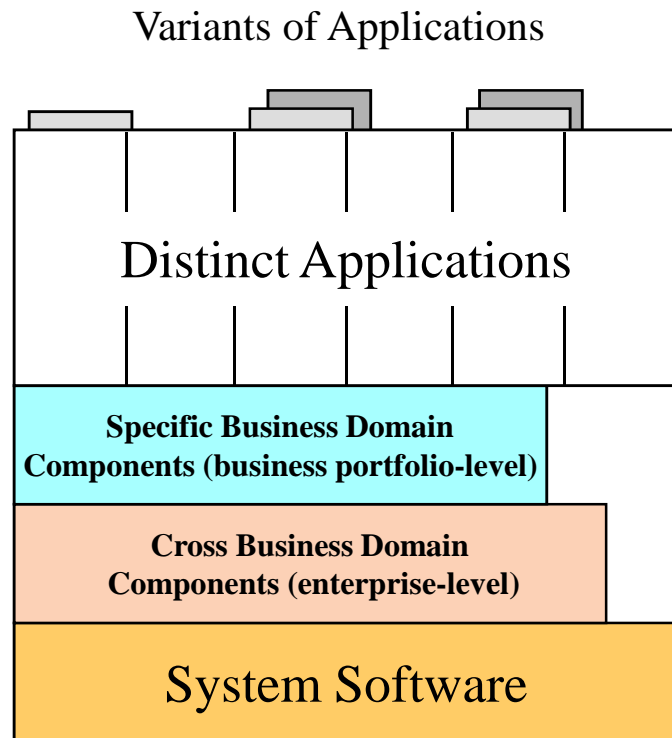
Common Mechanisms in UML

- **Specifications**
A textual description of additional details about an element
- **Adornments**
Details rendered as graphical attachment to basic symbols
- **Notes**
Arbitrary text comments attached to elements
- **Constraints**
Textual specifications of or conditions imposed on relationships
- **Stereotypes**
Creation of a new kind of model elements adapted from an existing model element
- **Properties**
Tag-value pairs to attach arbitrary information to model elements

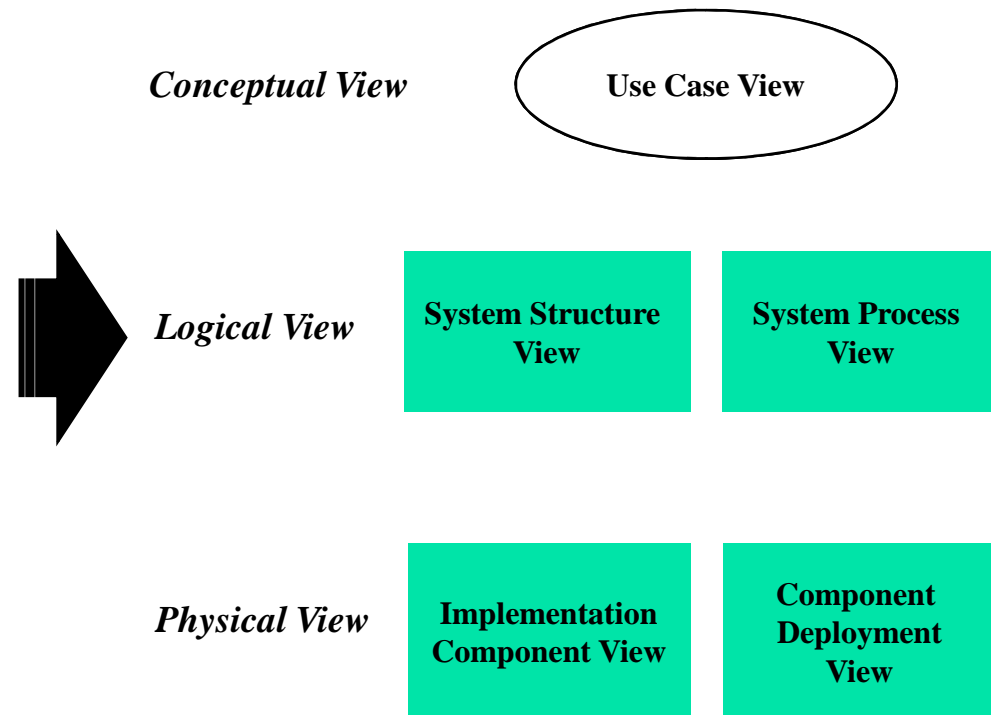


Use UML to Model Applications Analysis and Designs

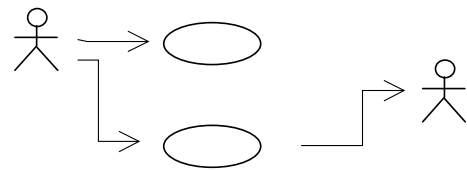
Conceptual Representations



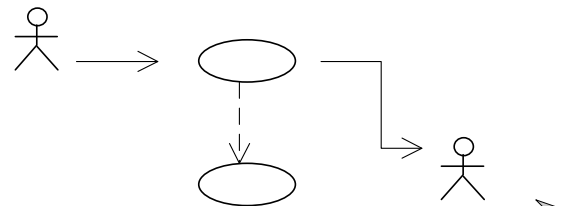
UML Representations



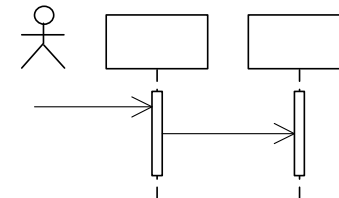
UML Can Model Business and Technology Systems



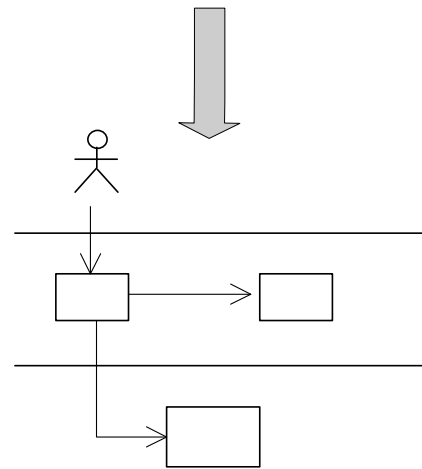
Business Event



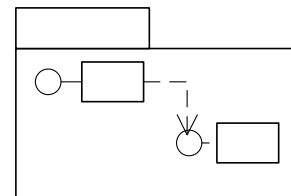
Application Capability



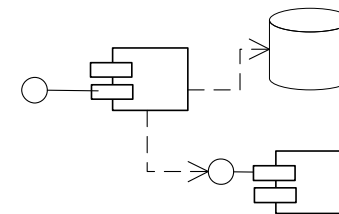
Application Flow



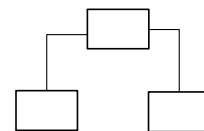
Business Process



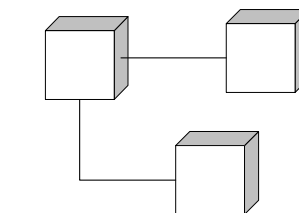
Component Structure



System Component



Data Model



Technology Deployment



Structural Modeling

Part 9



Create System Components using Objects

- **Business Component**
 - Logically grouped business functions with objects
 - For business applications
 - Objects (integrated together) interacting to complete a set of business processing

- **System Component**
 - Non-business specific
 - Technical utilities to support lower-level system functions
 - Built or supplied by technology vendors for supporting business component developments

Object Oriented View

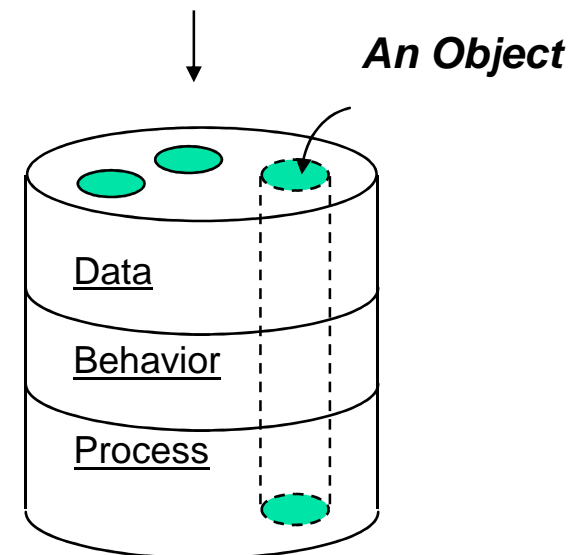
■ View the world by objects

- Analyze information by objects
- Relate objects by associations
- Communicate to objects by messages

■ Build systems with objects

- User interface
- Business logic
- Information brokers
- Data storage

Object Oriented View





Objects are

- Defined by classes with object-oriented approach
 - Contains information about itself attributes and/or states.
 - Has behaviors internally (private methods) and interface (public methods) available for other objects.
- Supported by object-oriented programming languages, JAVA, C++, C#, etc.
- Implemented as software program units of component or applications



Object Oriented Concepts

- **Object**
 - Packaged with data and behavior
- **Class**
 - Object classifications
- **Encapsulation**
 - Information hiding
- **Inheritance**
 - Information generalization and specialization
- **Polymorphism** (multiple forms)
 - Objects (and all subclassed objects) can be referenced and methods can be involved through dynamic bindings.

e.g. in OO Programming: SHAPE has <<subclass>> SQUIRE, TRIANGLE, CIRCLE

for i := 1 to 10

```
{aShape := SHAPE[i]; // SHAPE[i] can be SQUIRE or TRIANGLE  
shapeArea[i] := aShape.computeArea} // SQUIRE.computeArea or TRIANGLE.computeArea
```



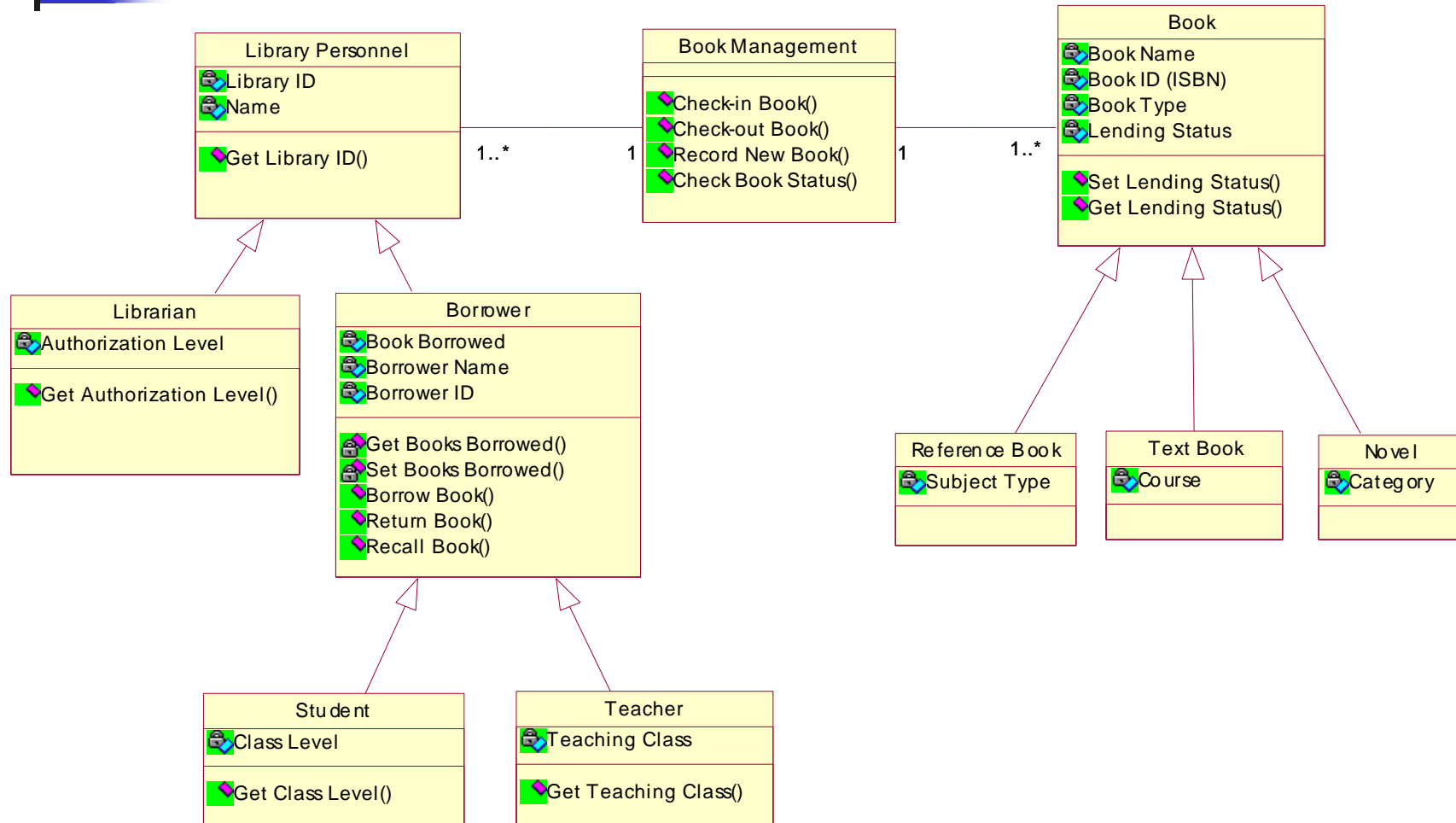
Class Diagram for Structure Modeling

- Class
 - Attributes - *Data*
 - Operations - *Function*

- Associations
 - Static – Cardinality, Containment, Inheritance
 - Dynamic – Messaging (more discussed in Behavior Modeling)

- Class Diagrams (not just for objects)
 - System-level
 - Component-level
 - Object-level

Class Diagram – Library Example



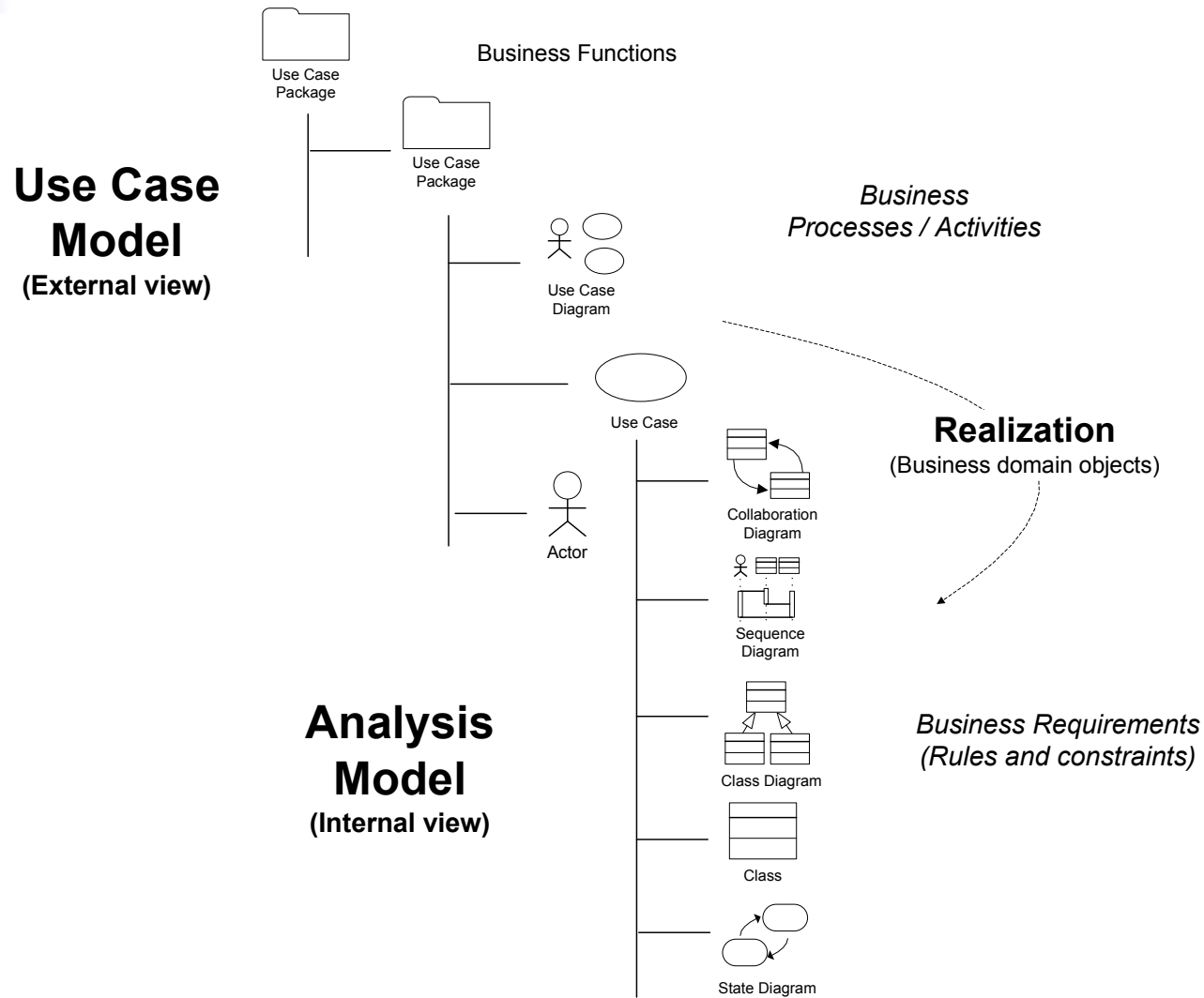
Library System Structural Model



Behavioral Modeling

Part 10

Develop System Analysis Models





Model the System Behaviors

- Use Case View (system external view)
 - What business needs
 - What business conditions
 - What business functions
- Interaction Diagram (system internal view)
 - How the need should be satisfied
 - How the conditions are applied
 - How the functions are carried out

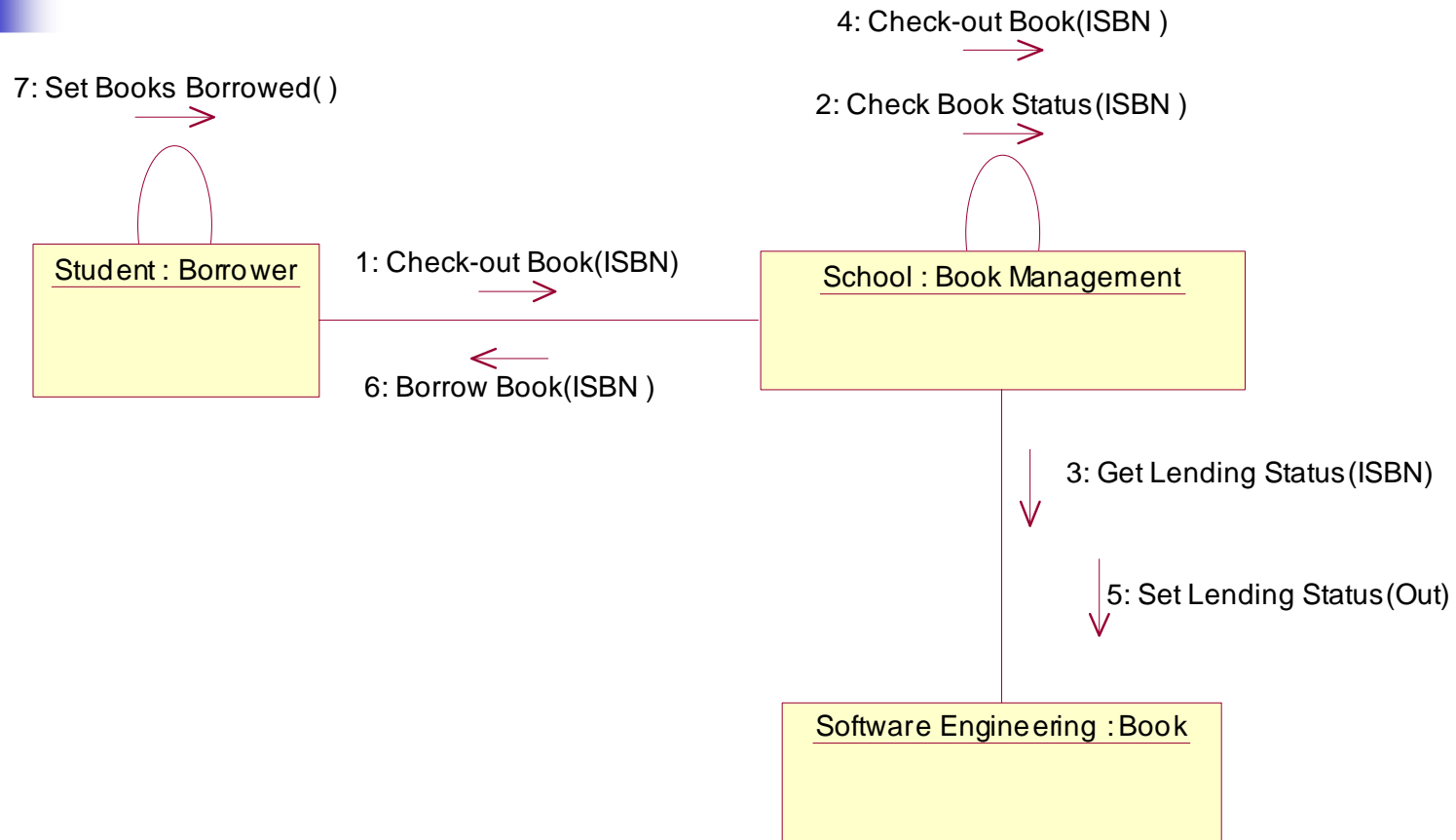


System Interactions

- Collaboration Diagram
(Communication Diagram in UML 2.0)
 - For functional analysis
 - Complexity
 - Distribution

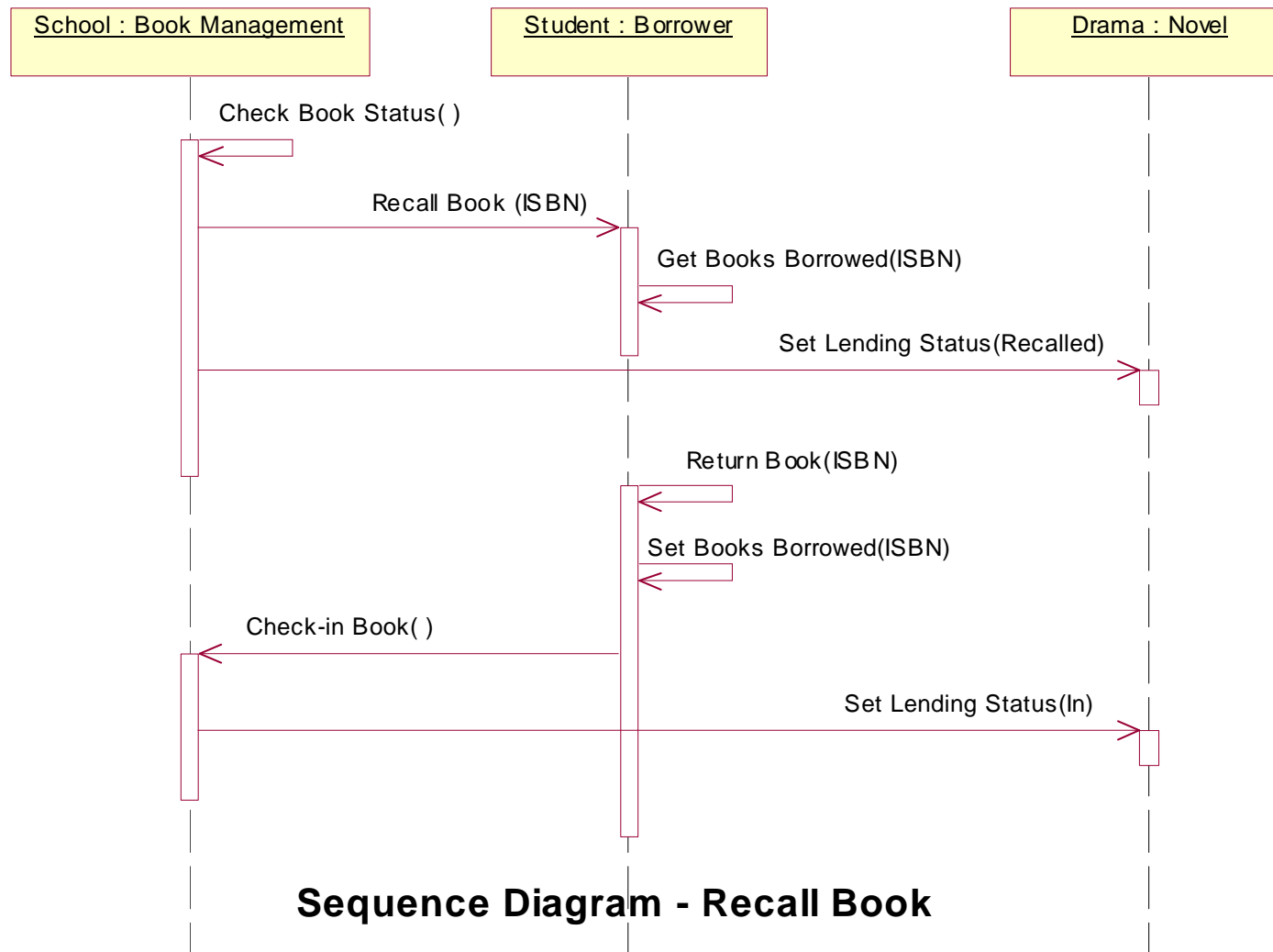
- Sequence Diagram
 - For sequencing analysis
 - Operation order
 - System availability
 - Data availability

Using Collaboration Diagram (Communication Diagram in UML 2.0)



Collaboration Diagram - Check-out Book

Using Sequence Diagram



Sequence Diagram - Recall Book

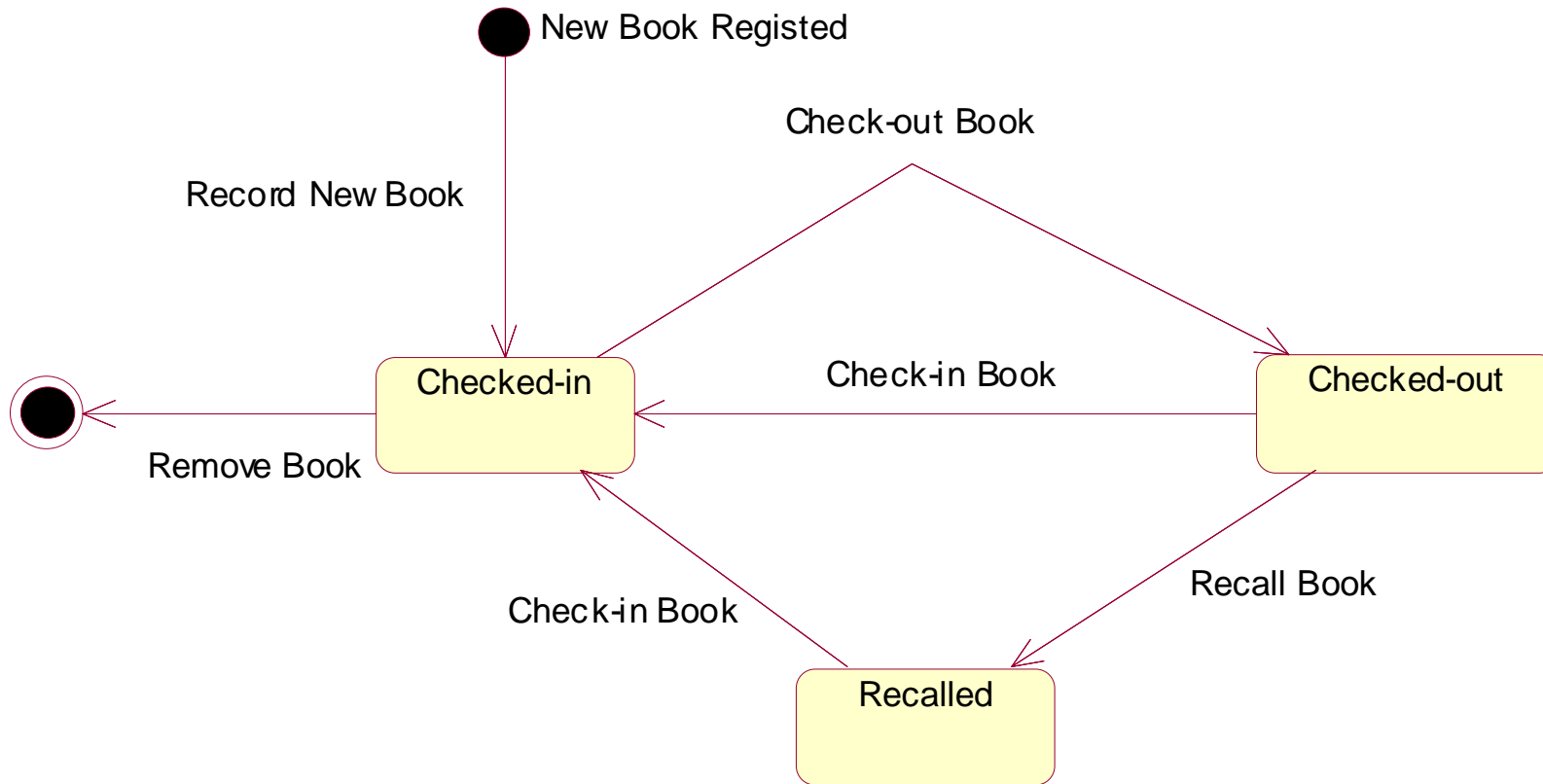


Model System, Component, Object States

- What to model?
 - Initial state
 - Trigger event
 - Every new state
 - All possible end-states

- When to model?
 - States represent different business conditions
 - States represent different business processing steps
 - States represent different business values

Statechart Diagram



Statechart Diagram - Book Lending Status

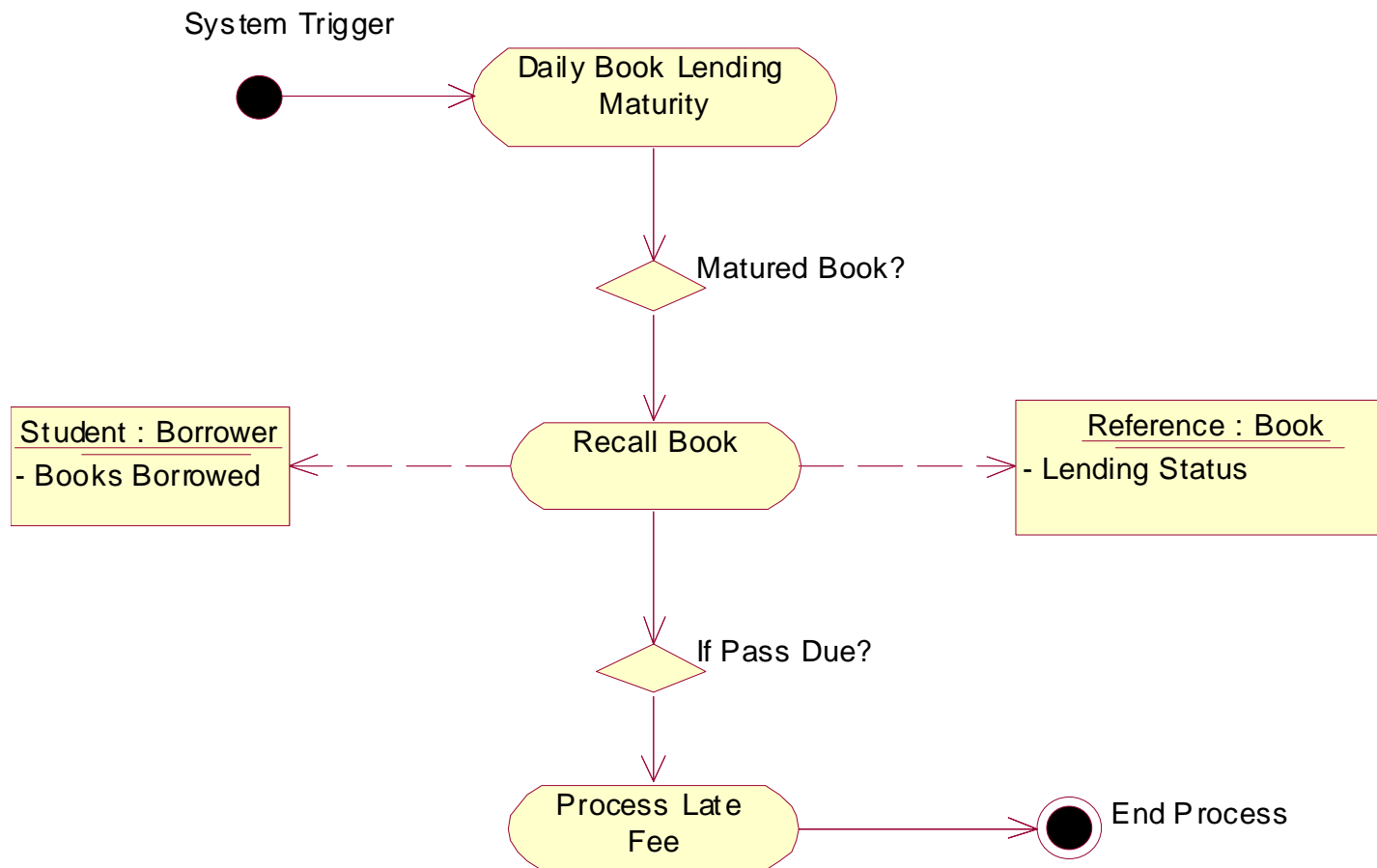


Model System, Component, Object Activities

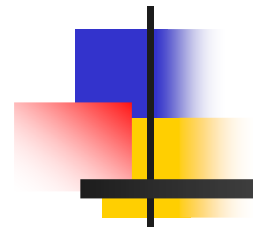
- What to model?
 - Trigger event
 - All activities
 - Conditional/un-conditional steps
 - Messaging to other systems, components, objects

- When to model?
 - Clarify business activities
 - Clarify business processing steps
 - Clarify system, component, object control flow

Activity Diagram



Activity Diagram - Manage Book Circulation



System Design Approach

Part 11



From Analysis to Design

- **Development Strategies**
 - Custom development
 - Packaged software
 - Outsourcing
- **Transform Analysis Models into Design Models**
 - Factoring
 - Partitions and collaborations
- **Structure Application Components**
 - Layers
 - Packaging



Application Development Strategy

- **Custom Development**
 - In house software development teams
 - Develop software products
 - Develop special purpose applications

- **Packaged Software**
 - Off-shelf commercial software products
 - General purpose software products for customization
 - Business components for development
 - Non-business components for development
 - Application frameworks for development

- **Outsourcing**
 - Software vendors
 - In-house consulting and development



Selecting Development Strategy

Select a development strategy that meets the business needs and the time table

Determining Factors	Choose Custom (in-house) Development if ...	Select Package Solution if ...	Look for Out-sourcing Development, if ...
<i>Business Need</i>	unique need	Common need	Not core technology for the business
<i>In-house Experience</i>	Existing business and technical experience	Existing business experience	Both business and technical experience do not exist
<i>Project Skill And Management</i>	Ability to build	Not strategic skill	Strategic outsource direction
<i>Time Table</i>	Flexible solution	Quick solution	Flexible skill available for development, but may not as quick



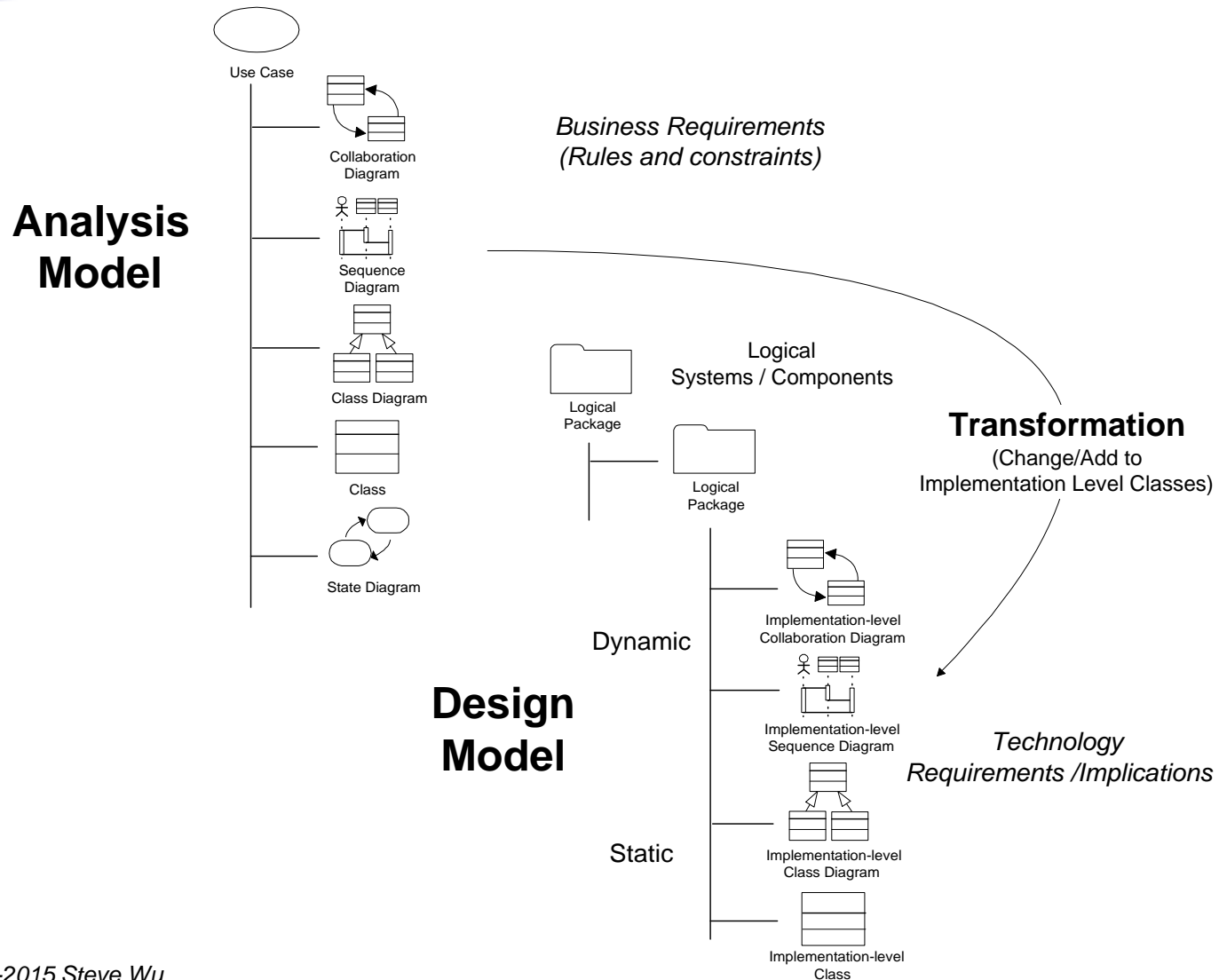
Transform Analysis Models into Design Models

For custom software development:

- Factoring – generalization
 - Abstract functional behaviors
 - Abstract common data relationships
 - Group coherent functions and data

- Partitions – reduce complexity
 - Components
 - Subsystems

Create System Design Models





Structure Your Application

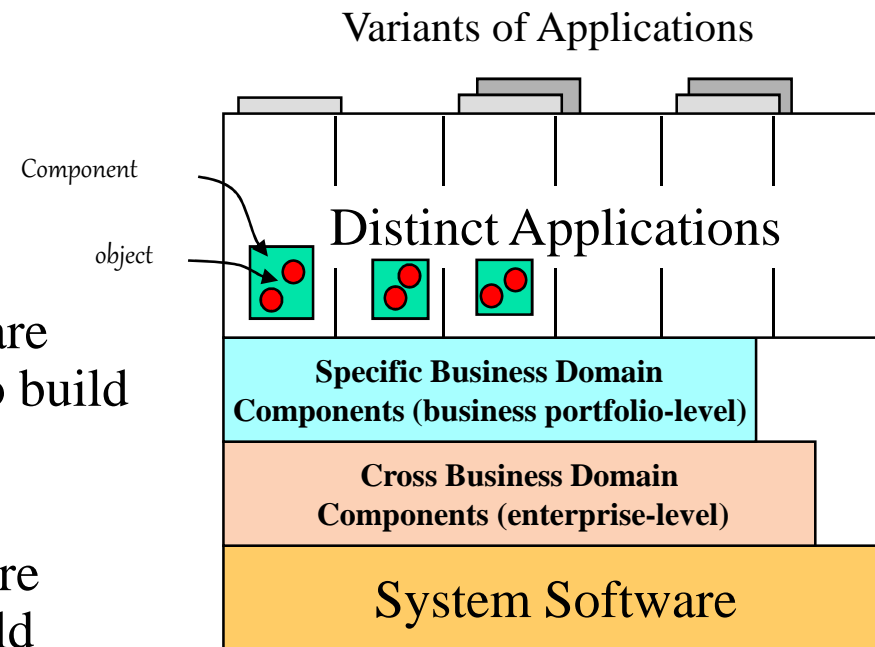
- **Architecture Layers (top to bottom)**
 - Business application specific
 - Non-business application specific
 - Business domain components
 - Non-business domain components
 - System components

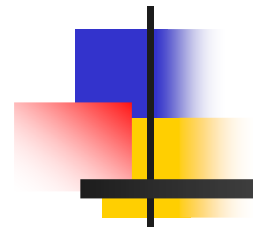
- **Application Tiers (front to back)**
 - User interface – presentation
 - Business functional logic
 - Back-end business data management

- **Package Application Components**
 - Cohesive business functions in scope
 - Implementation of coherent processes and data
 - Runtime component distribution on platforms

Design Applications

- Applications
 - Each Line of business
 - Business domain
- Components
 - High granular units of software
 - High cost-effective, harder to build
- Objects
 - Low granular units of software
 - Good reusability, easy to build





System Architecture

Part 12



System Architecture and Infrastructure

- System Architecture Designs
- Making an Architecture Choice
- Design System with Quality and Reusability
- Infrastructure Design
- Application Deployment Model



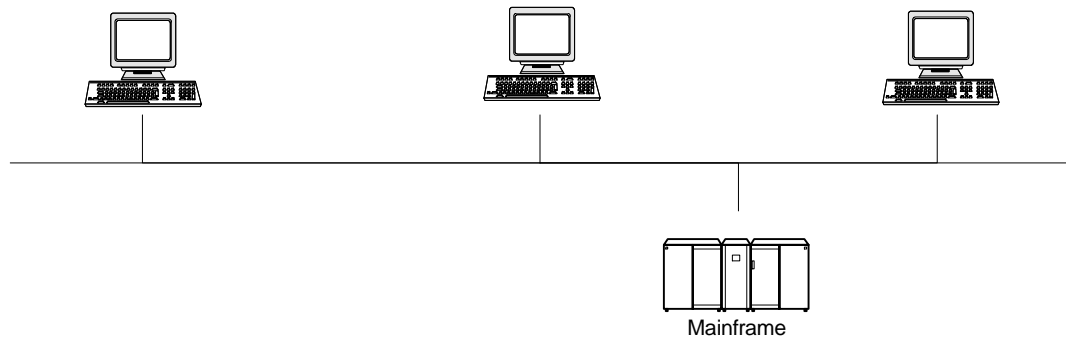
System Architecture Designs

- Server-based
- Client-based
- Client-server
- Distributed multi-tiered



Server-based

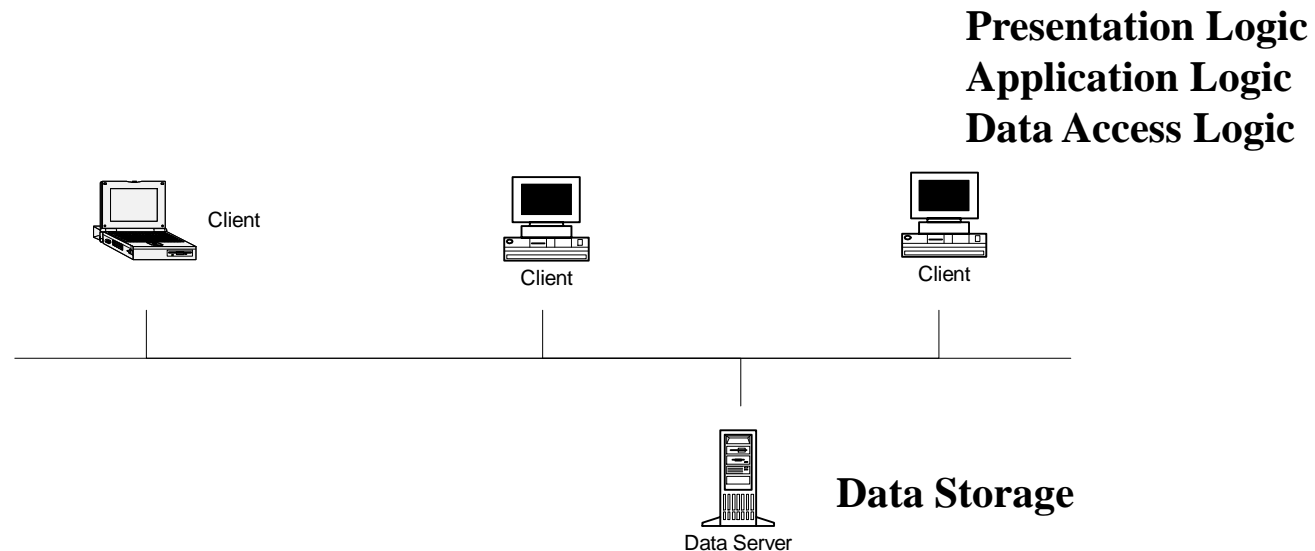
- Easy to manage
- Less flexible and versatile



Presentation Logic
Application Logic
Data Access Logic
Data Storage

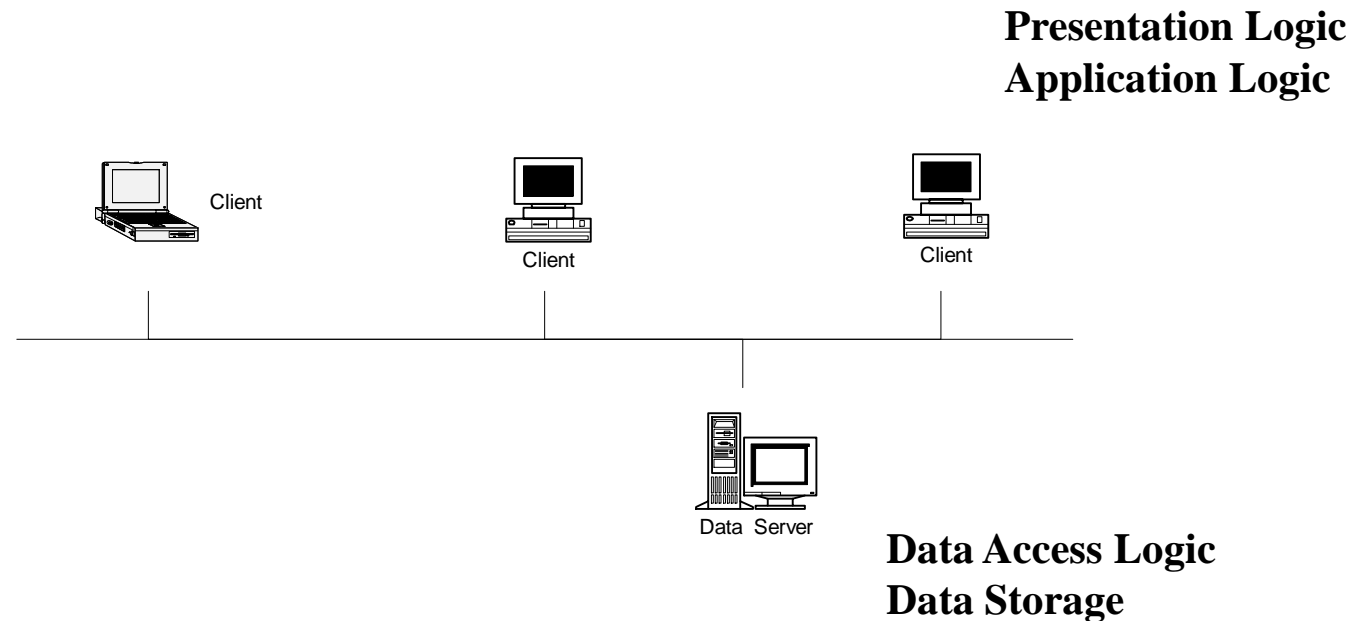
Client-based

- Versatile user experiences
- Difficult to change and upgrade
- Higher cost



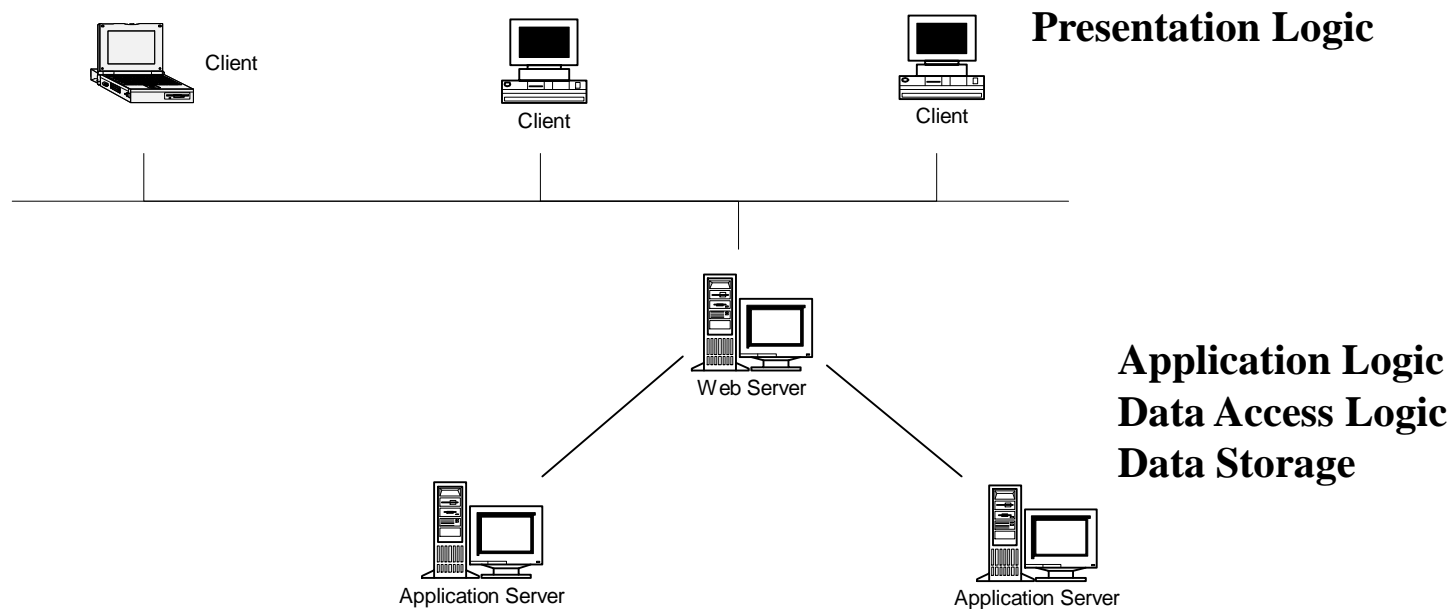
Client-server

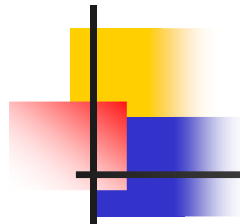
- Versatile user experiences
- Scalable for data accessing
- Harder upgrade
- Higher cost



Distributed multi-tiered

- Versatile user experiences
- Easy to manage and upgrade
- Scalable for application and data processing
- Low cost





Making Architecture Choice

Web-enablement are all possible in these architecture configurations

Measurement	Server-based (Mainframe)	Client-based (Desktop)	Client(rich)- server (2-tiers)	Thin/Rich- Client (N-tiers)
Cost of Infrastructure	Very High	Medium	Low	Medium/High
Cost of Development	Medium	Low	High	Medium/High
Ease of Development	Low	High	Low-Medium	Medium
Interface Capabilities	Low	High	High	High
Control and Security	High	Low	Medium	Low
Scalability	Low	Medium	Medium	High



Design System with Quality and Reusability

- Granularity - Modula and functional decoupling
- Interface - Consistent and interoperable
- Security - Protect the business
- Scalability – able to handle large business data volume and users
- Performance - Efficient and high throughput
- Extensibility - Easy to modify and add new functionality
- Reliability - High degree of fault-tolerance
- Testability - Less programmable errors
- Portability - Develop once use everywhere

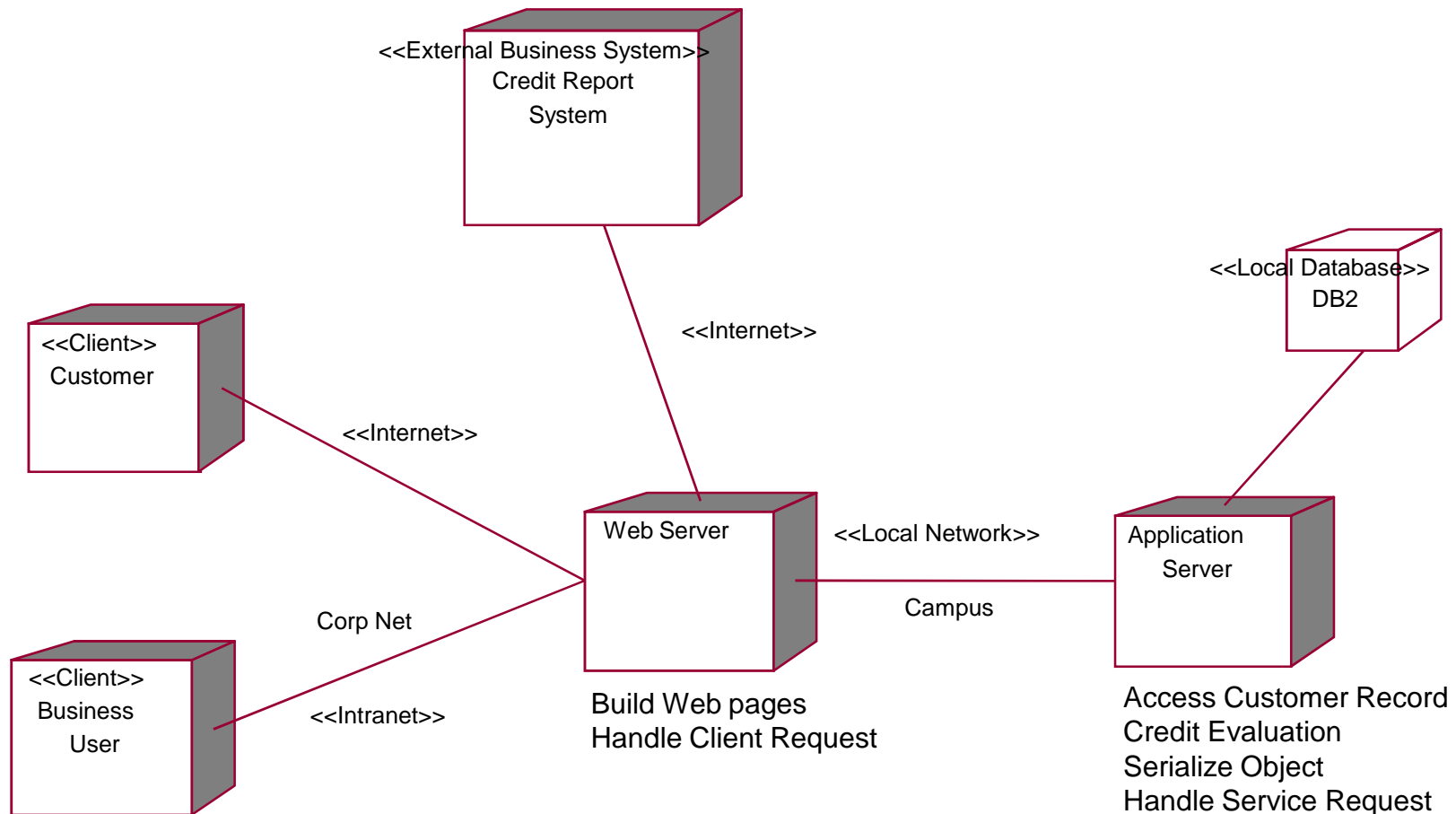


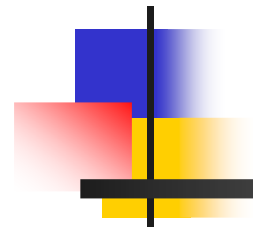
Infrastructure Design

- Technology Deployment Model
 - System topology (locations)
 - System connectivity (LAN, WAN, Wireless)
 - System process distributions (business functions)

- Hardware and Software Specification (requirements)
 - Choices of hardware
 - Platforms/OS (operating system)
 - System operational environment
 - User/client environment

Application Deployment Model (UML)





Application Design

Part 13

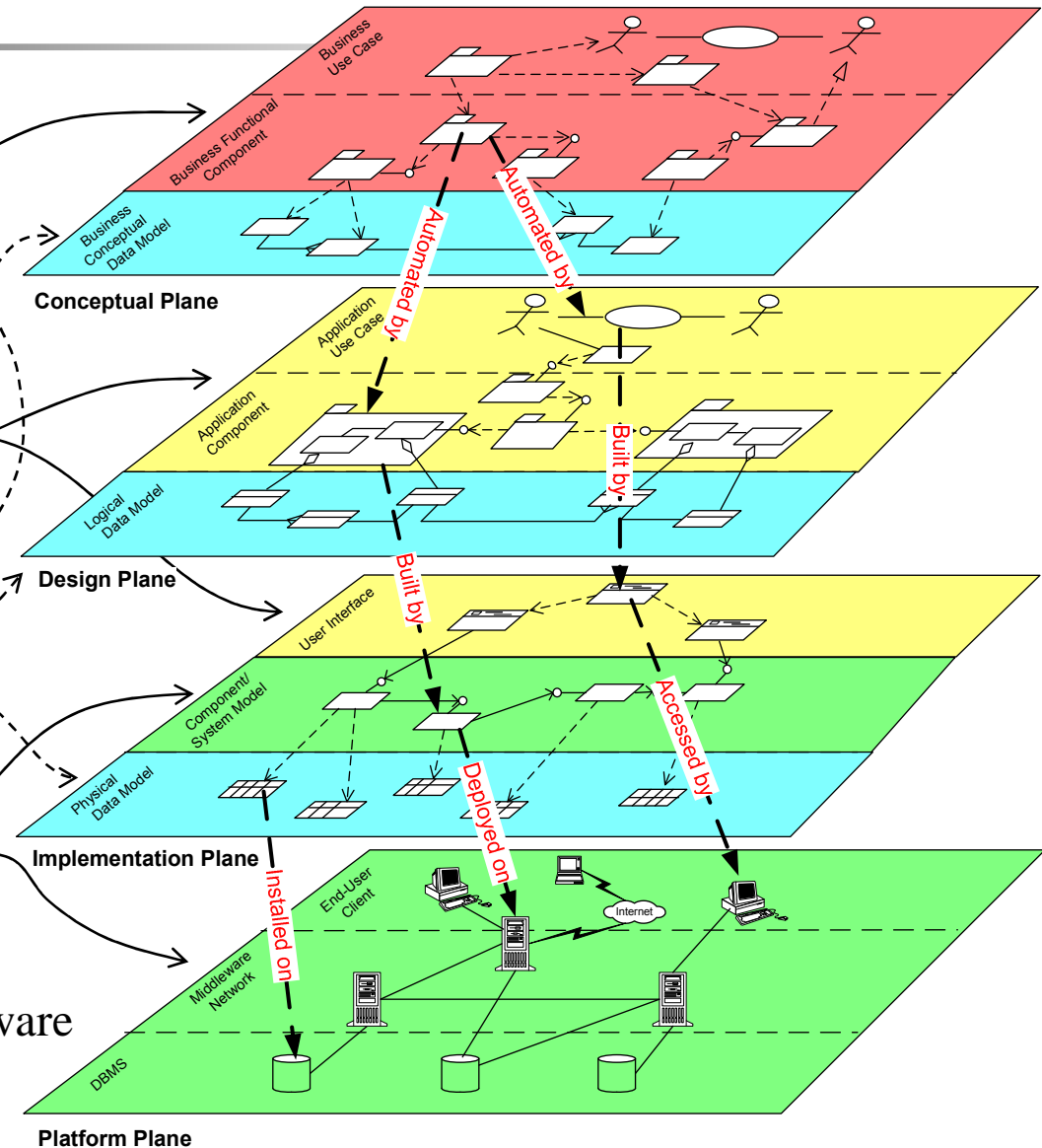


Guiding Your Designs

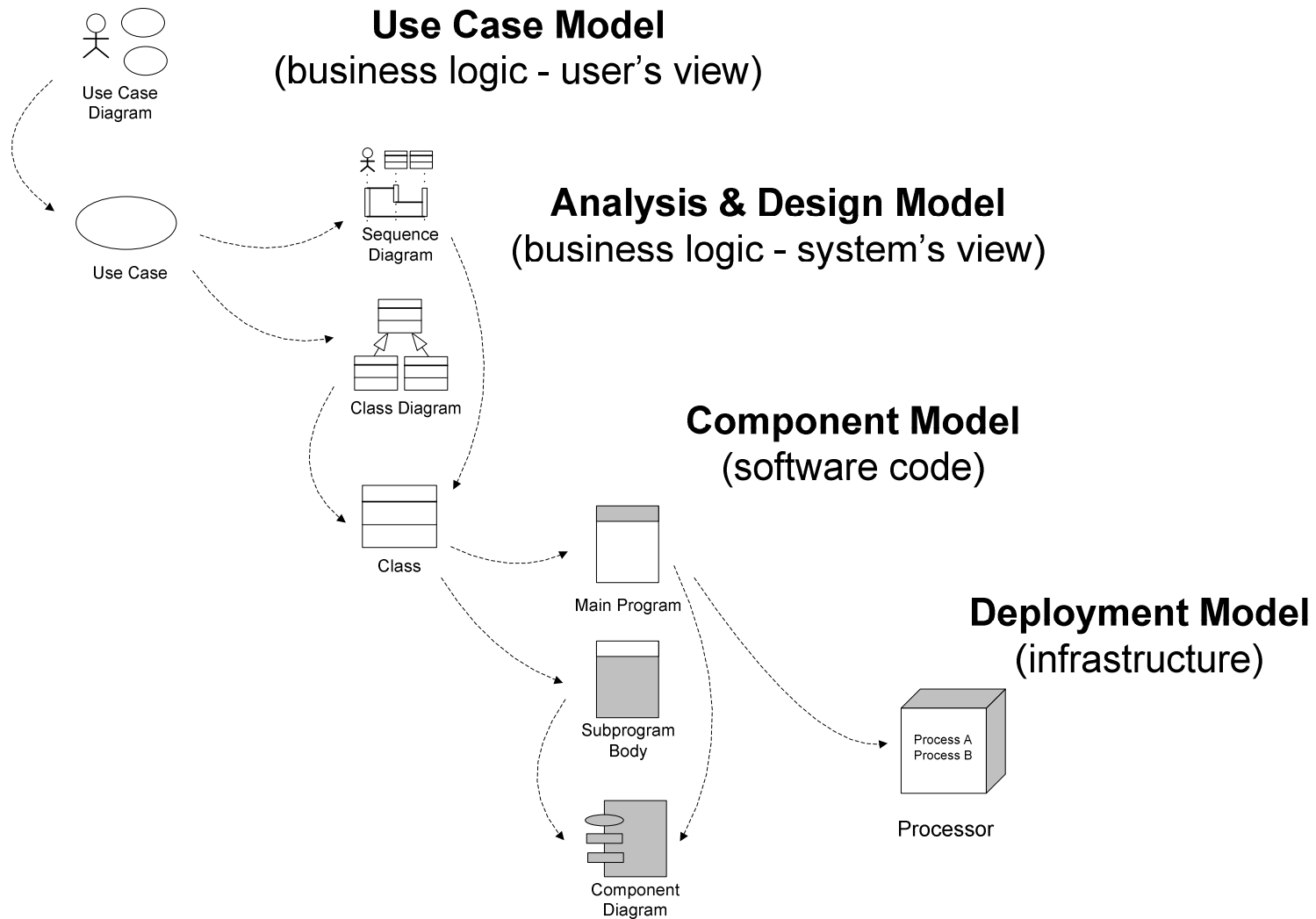
- Architectural Planes
- Purposes of Design Models
- Design Principles
- Design Best Practice
- Design Heuristic
- Design Techniques and Considerations

Architectural Planes

- **Business Architecture**
 - Business processes
 - Business requirements
- **Application Architecture**
 - Application analyses
 - Application designs
- **Data Architecture**
 - Business data
 - Physical designs
- **Technology Architecture**
 - Infrastructure
 - System software and hardware



System Model Transformation





Purposes of Design Models

- To show how a system is designed to work
 - Transformed the business activities captured from the analysis object model.
- To show what a system needs to have
 - Implementation-level classes and relationships
 - Decomposed and added transient classes to support the implementation.
- To meet needs of all business use cases
 - All the design models satisfies the use case requirements
 - Transform all analysis object models into design object models



Design Principles

- A class should have a key abstraction
 - Group coherent functions, not just a collections of data and functions
- Hide (encapsulate) all data within its class
 - Use object interfaces to get data
- Keep classes independent as much as possible
 - For potential reuse and easier to extend
- Distribute system functions among the component evenly
 - Balanced functional distribution among the component structure



Design Best Practice

- Keep related data and behavior in one place
- Factor the commonality of data, behavior, and/or interface as much as possible in an inheritance hierarchy
- Use public interfaces of a class (or a component) for accessing data
- Implement a minimal public interface that all classes understand.
- Make roles as attributes of a class instead of each role as different classes, if they have common responsibilities/behaviors (e.g. *father* and *mother* is attribute value of *parent*)
- An class should minimize the number of class it collaborates with for loose-coupling
- A container class should know what it contains, and the contained classes should not have to know who contains it (so that the class is reusable elsewhere)
- The derived-class should not know anything about its supper class (so that it is independent for reuse)



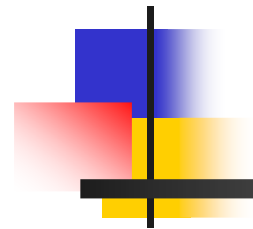
Design Heuristic – Best Practices

- Do not use global (constant/variable) data. Instead, use objects variables and methods) as bookkeeping information for other objects.
- Do not create global class in your system that the most of classes have relationships with it, or are depend on it.
- Do not access or change the data of an object without going through its public interfaces.
- A class with many <<accessor>> methods in its public interface that should not be a class or should be more than one class.
- Application business logic (i.e., functions) should be decoupled with the user interface.
- Physical design criteria should not corrupt the logical designs.



Design Techniques and Considerations

- Take the analysis models and create a system design models:
 - Architecture Layers
 - Design Principles
 - Design Best Practice
- Consider the following when define the system architecture:
 - Current environment
 - System to be interfaced
 - Future business needs
 - Possibility of system expansion
- Use collaboration/sequence diagrams to model the system behaviors.
- Use class diagrams to model the relationships among the objects discovered.
- Use statechart diagrams to model an object class behavior as needed.

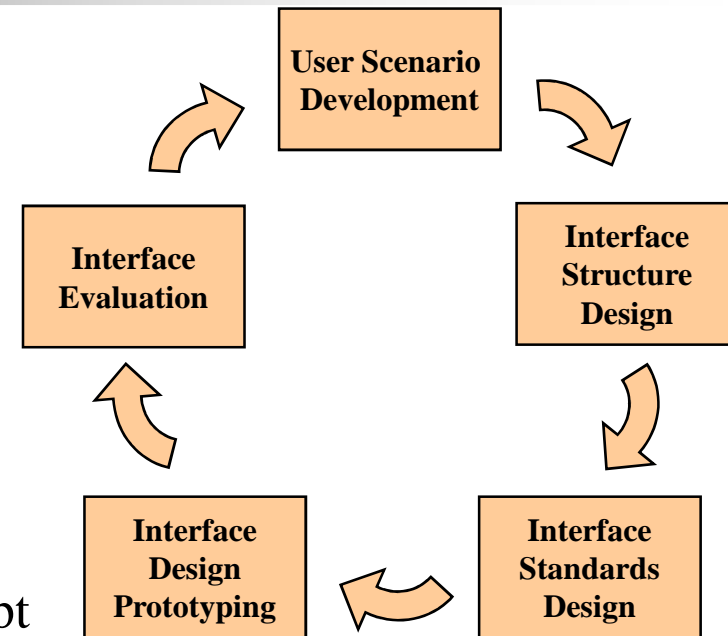


User Interface Design

Part 14

User Interface Design Process

- User Scenario Development
 - Outline steps of user's workflow
- Interface Structure Design
 - Navigation (links) of interface flows
- Interface Standards Design
 - Interface metaphor – real world concept
 - Interface objects – views, forms, pages
 - Interface actions – navigation commands, operations
 - Interface icons – visual placement
 - Interface templates – appearance of page and screens or forms





User Interface Design Process (continued)

- Interface Design Prototyping
 - Storyboard
 - Web-page prototype
 - Language prototype

- Interface Evaluation
 - Heuristic – compare basic design principles and best practice
 - Walk-through – present to users for feedback
 - Interactive – experimental evaluation
 - Formal usability testing – real environment of user's evaluation



User Interface Design Considerations

- **Layout**
 - Areas for commands, navigation, inputs, outputs, status
- **Content Awareness**
 - Informative for users to know where they are in the system
- **Appealing**
 - Functional and inviting with good use of color space and fonts
- **User Experience**
 - Easy to learn and easy to use for users
- **Consistency**
 - Consistent design for clarity and predictable functional settings
- **Minimize Effort**
 - Simple to use the system with simple user interface



User Interface Navigation Design

- **Basic Principles**
 - Prevent user mistakes
 - Simplify recover from mistakes
 - Use consistent grammar order
- **Types of Navigation Controls**
 - Languages (commands)
 - Menus (lists)
 - Direct Manipulations (control widgets)
- **Messages**
 - Responses feedback
 - Guidance instructions (lists)
- **Navigation Design Documentation**



Input Design

- Basic Principles
 - On-line processing vs. batch processing
 - Capture data at the source
 - Minimize the keystrokes
- Types of Inputs
 - Text
 - Numbers
 - Selection - check box, radio button, list box, drop-down, combo box, slider
 - Graphs
- Input Validation
 - Completion
 - Format
 - Range
 - Digit
 - Consistency



Output Design

- **Basic Principles**
 - Understand report usage
 - Manage information load
 - Minimize bias

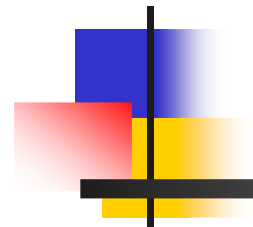
- **Types of Outputs**
 - Detail reports
 - Summary report
 - Document
 - Graphs

- **Media**
 - Online (voice, video)
 - Paper (image, text)
 - Electronic (control/message notification)



Best Practice for User Interface Design

- Separate the user interface objects from domain business objects so they can change independently.
- Avoid sequence of modal dialog windows at anytime if possible as they tend to “lock-up” the windows while users may need to access other information.
- Design user interfaces with users in mind - how it is used and how convenient it is, etc. Minimize user learning curve.
- Considering business rules and work flows to make design trade-off between “flat” vs. “deep” windows, web pages, or menus.
- Prototype user interface and validate the usability with the business users as earlier as possible.
- Balance of interactivity and frequent updates cost



Business Logic Design

Part 15



Business Application Logic


- Design Criteria
 - Object coupling
 - Class cohesion
 - Method coherence
 - Design goals

- Design Activities
 - Design specifications
 - Identifying opportunities for reuse
 - Restructuring the design
 - Optimizing the design

- Design Application Components with Objects
 - Design models for business logic



Object Coupling (between objects)

<i>Level of Coupling</i>	<i>Type of Object Relationship</i>
loose	No direct coupling between objects
	Passing reference to another object
	Passing value to another object
	Passing control data to another object
	Common or global referenced by both objects
	Tight



Class Cohesion

A class should has consistent information

<i>Cohesion</i>	<i>If a class has ...</i>
Good cohesion	Single business concept (attributes related to the business)
↓	Mixed business concepts (different business attributes)
	Mixed business domain (different business areas)
Not good cohesion	Unrelated types of business (different concepts)



Method Coherence

Methods of a class should be related

<i>Coherence</i>	<i>If methods of a class have ...</i>
Good Coherence	only a single function
↓	connected/related functions
	functions using same attributes
	directly related functions
	support indirectly related functions
	supports other related functions of different objects
Not Good Coherence	has unrelated functions



Design Goals

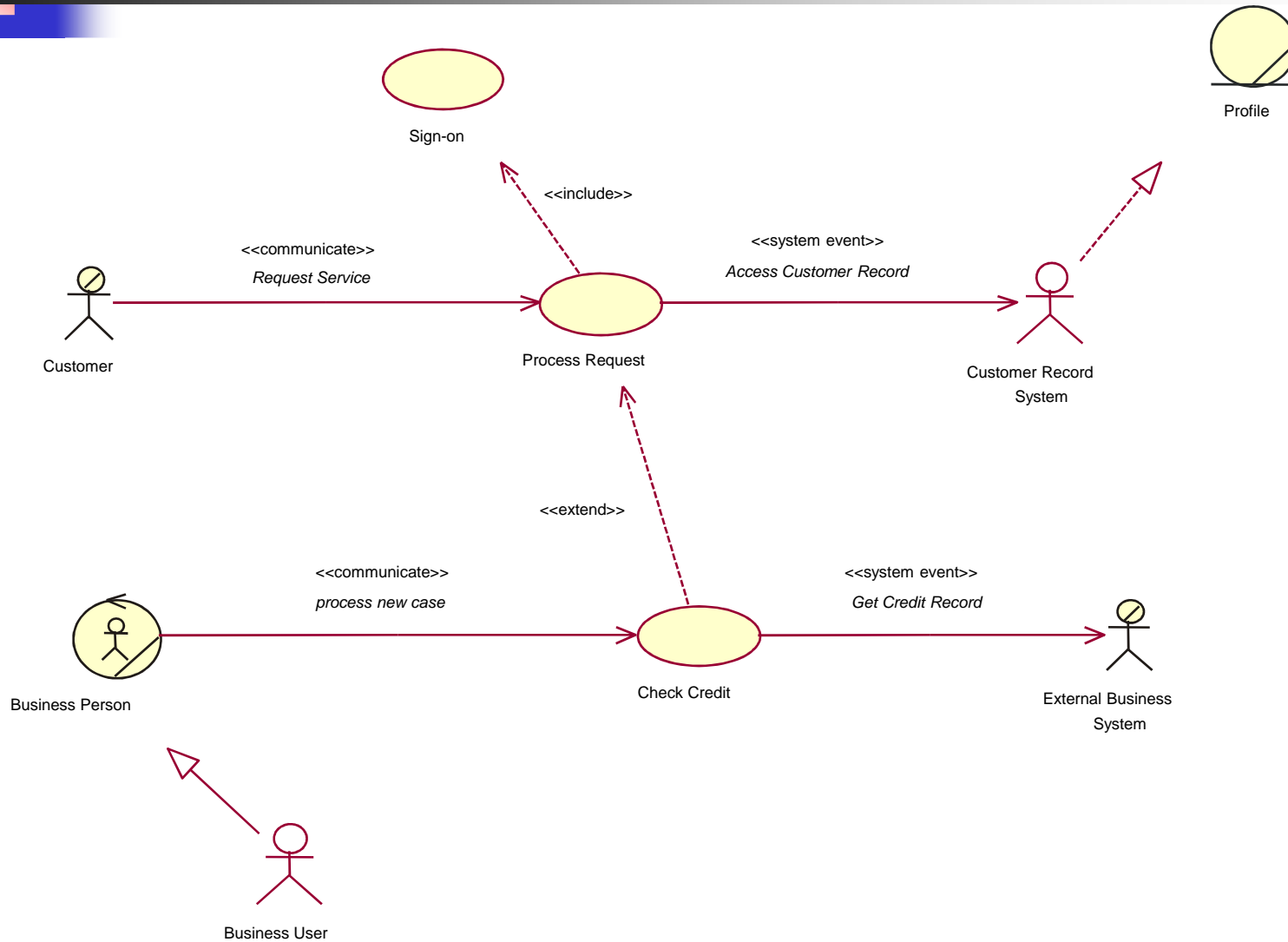
- Maximize the functional coherence within an encapsulation boundary (e.g. within a component or an object)
- Minimize the functional coupling between the encapsulation boundaries (e.g. between two components or objects)



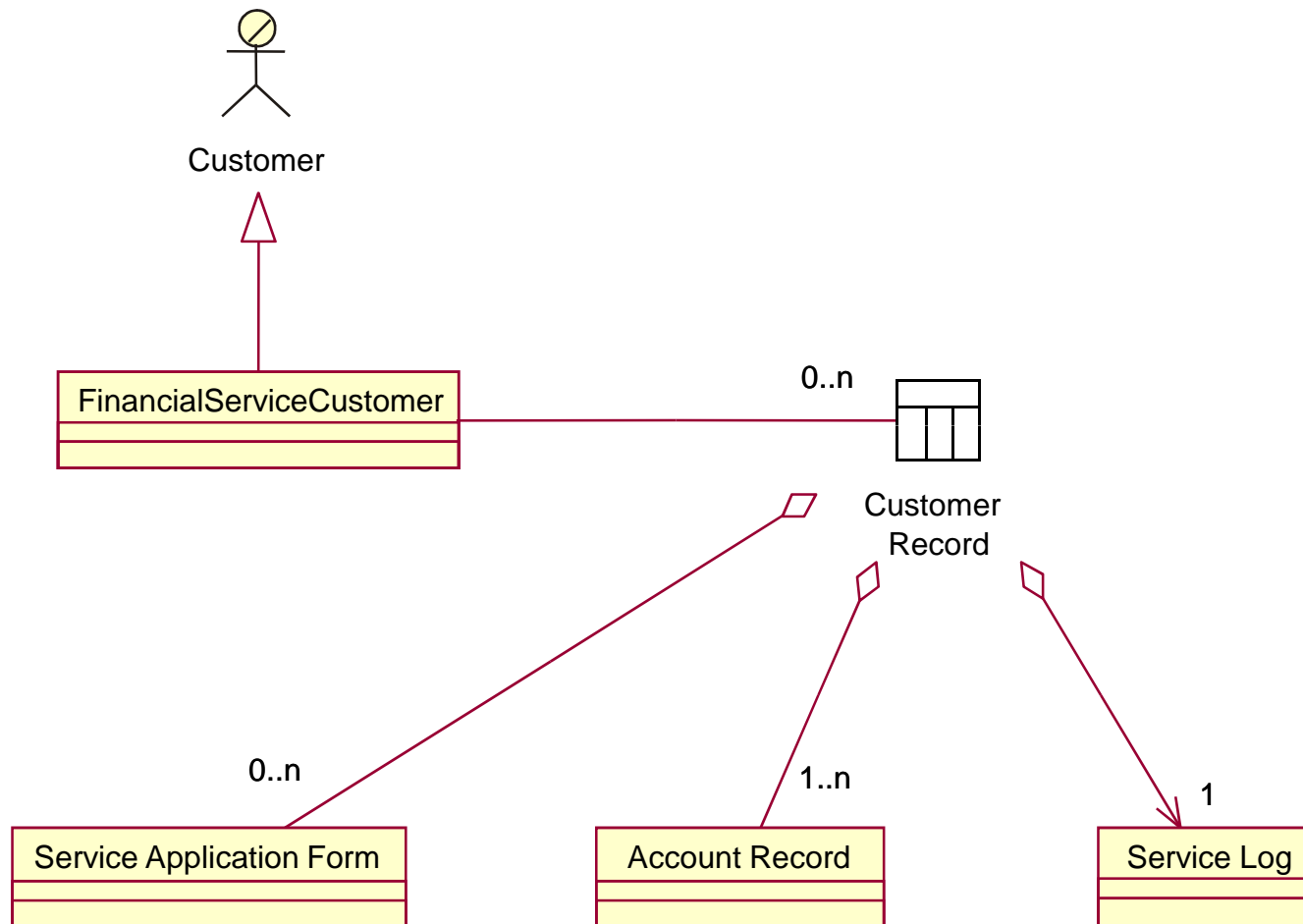
Design Application Components with Objects

- Understand Business Requirements
- Design Business Class Structure
- Design Business Application Logic Flow
- Design Statechart of Business Process Rules
- Design Components of Business Application

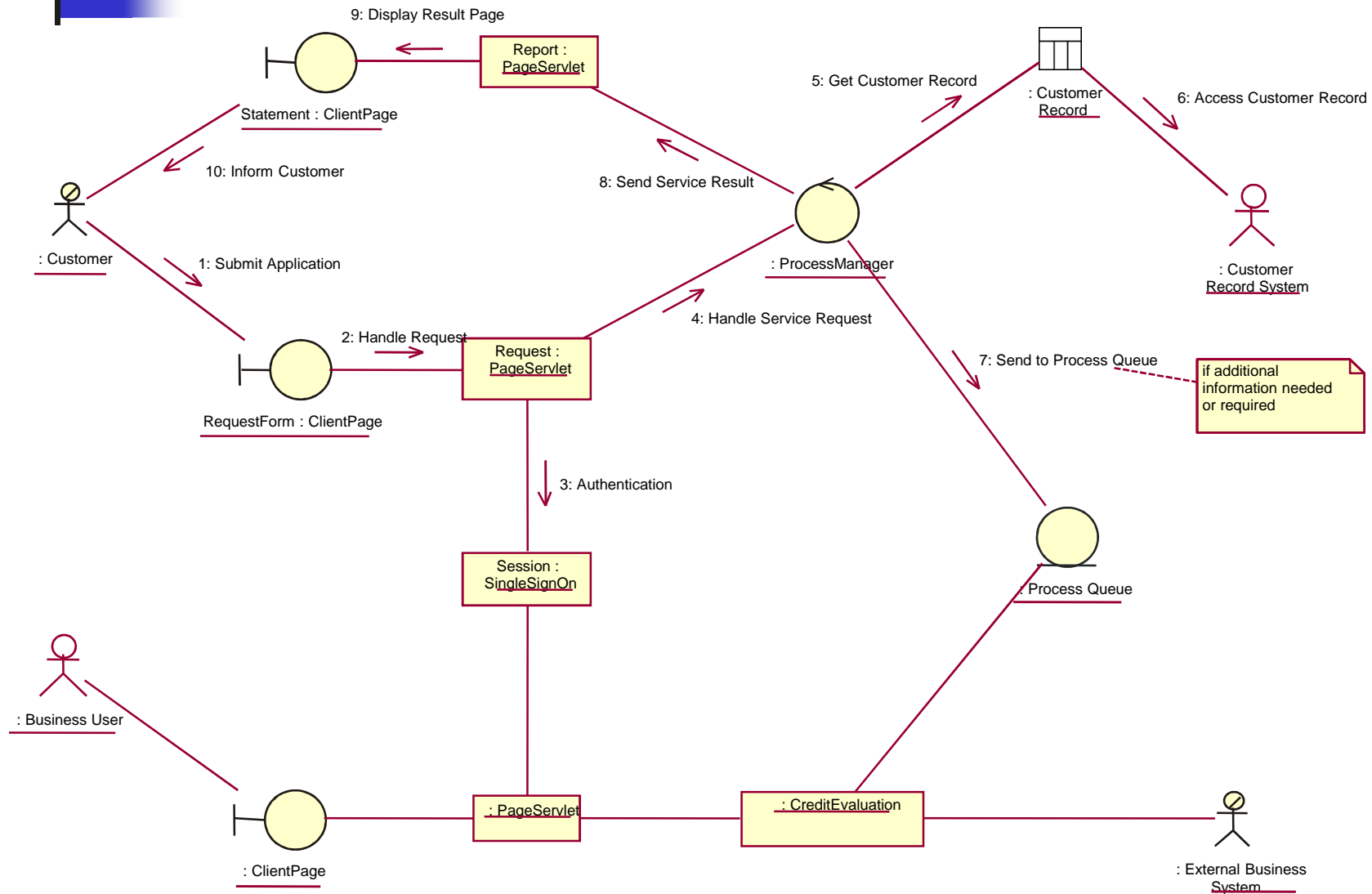
Understand Business Requirements



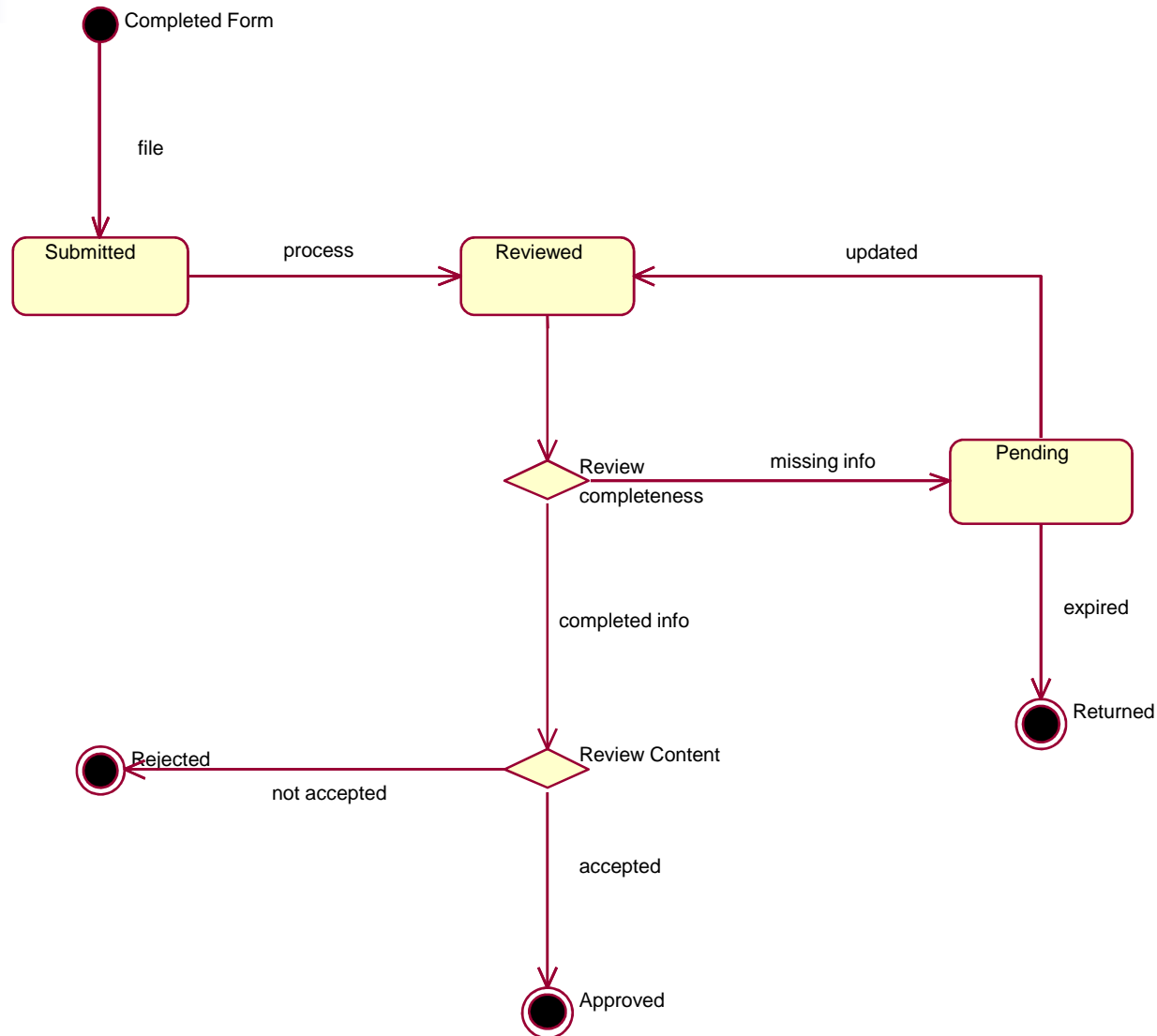
Design Business Class Structure



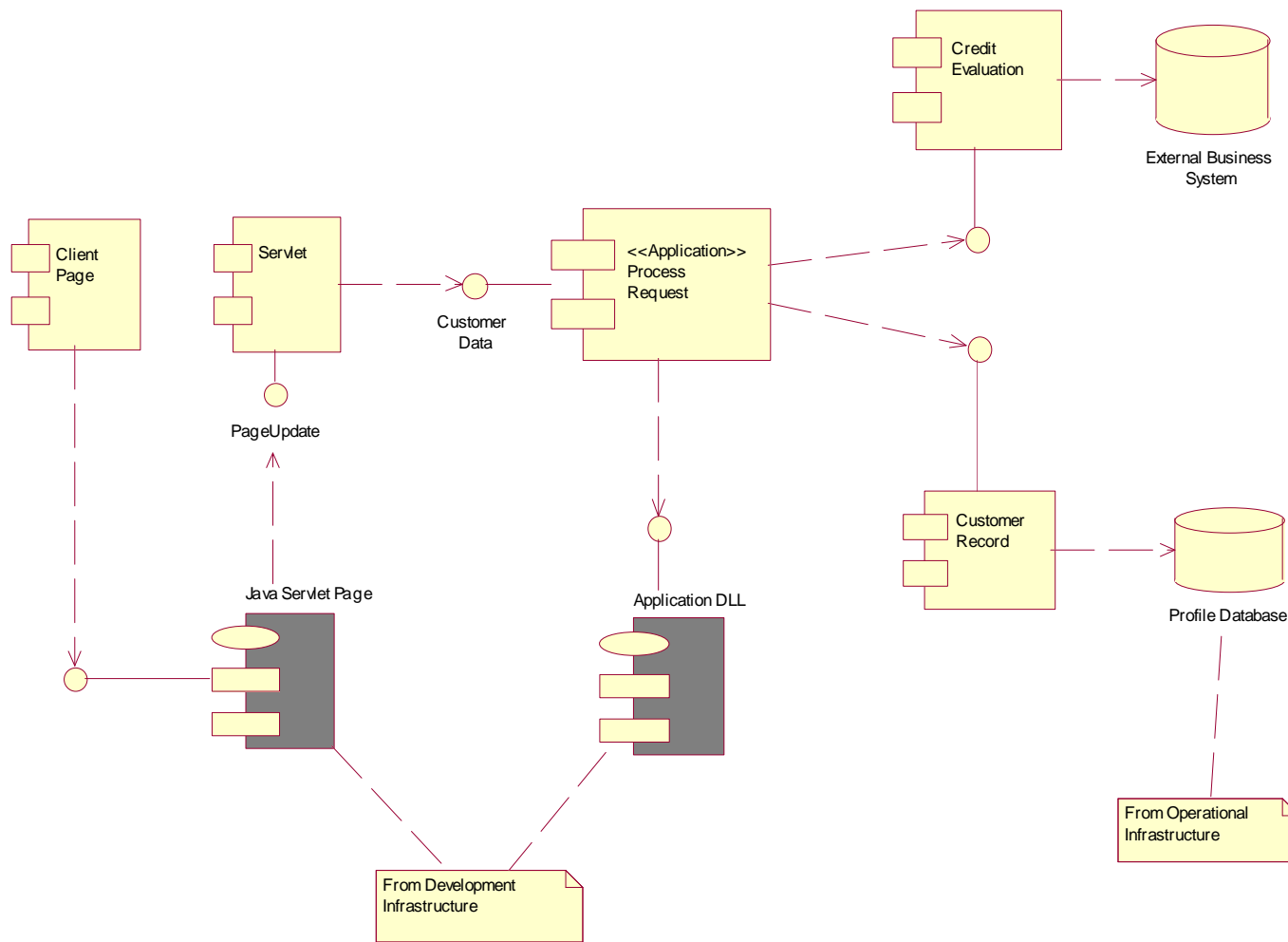
Design Business Application Logic Flow

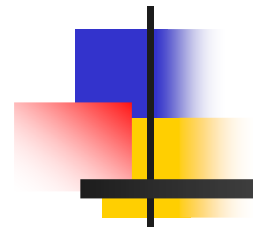


Design Statechart of Business Process Rules



Design Components of Business Application





Data Persistence Design

Part 16



Data Persistence

- Data Persistence Formats
- Mapping Objects to an Relational Format
- Optimizing RDBMS-based Object Storage
- Design Considerations of Data Persistency



Data Persistence Formats

- Sequential and Random Access Files
- Relational Databases
- Object-Relational Databases
- Object-Oriented Databases
- Selecting an Data Persistence Format

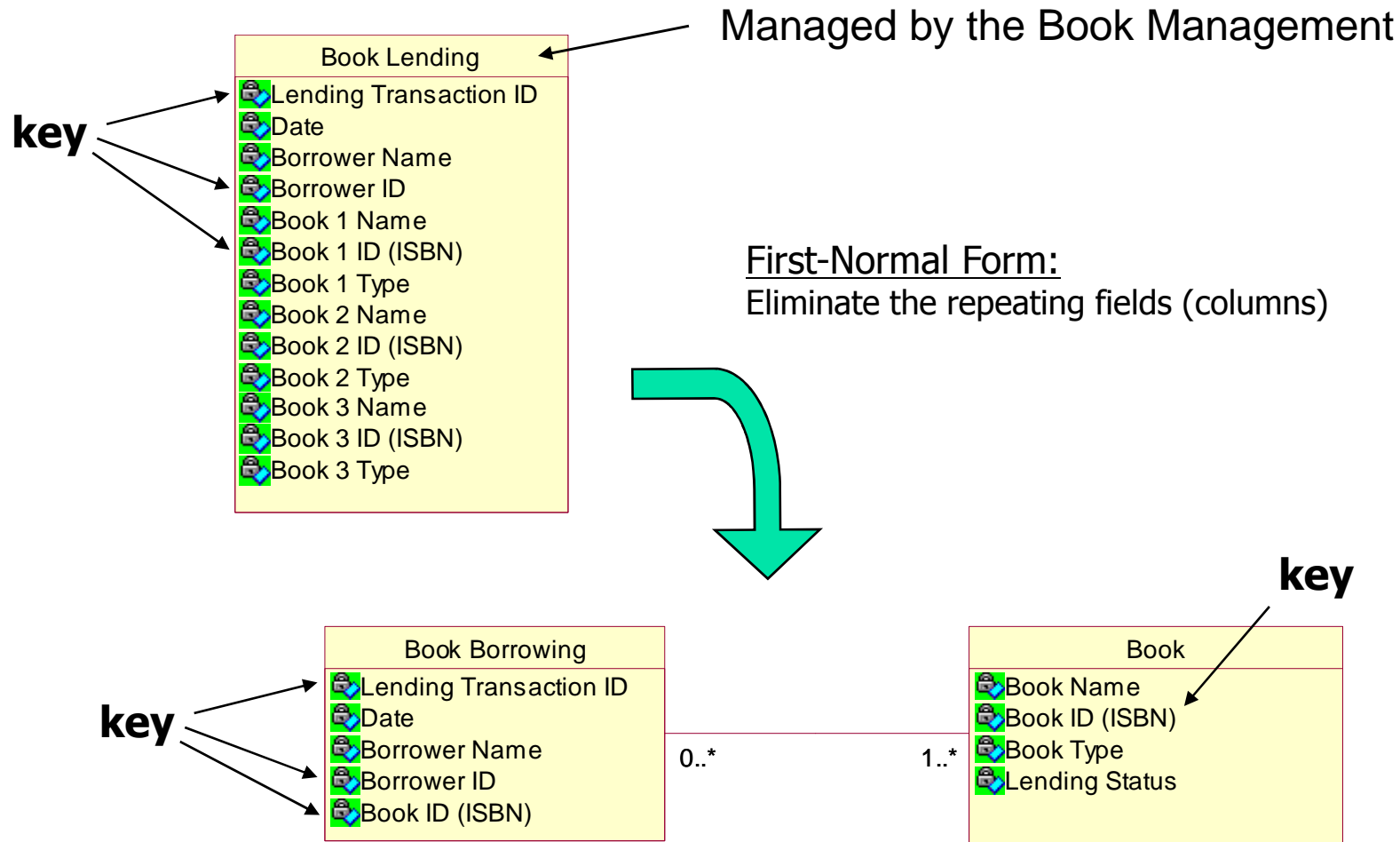


Mapping Objects to Relational Format

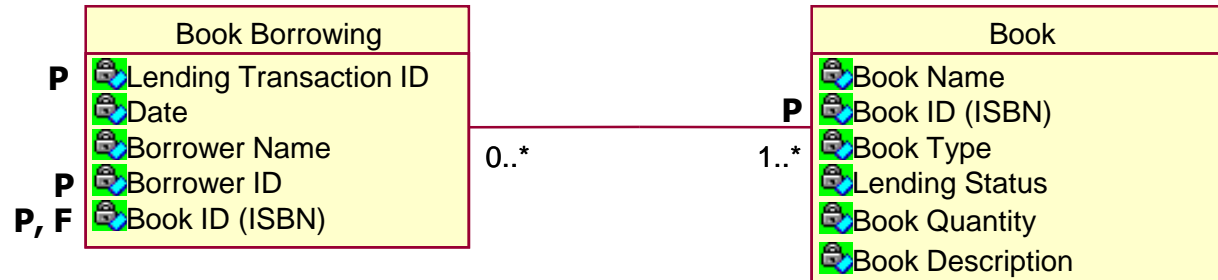
- Create Persistent Object Models
 - Extracted business data from the class hierarchy for data persistency to a relational database
- Mapping Table

Object	Entity (Relational)
Class	Entity table
Object	Entity (Row)
Attribute	Column
Object identity attribute	Primary key (Entity key column)
Methods	Stored procedures
Association	Foreign key
Collections (whole-parts)	Association tables

Optimizing RDBMS-based Object Storage



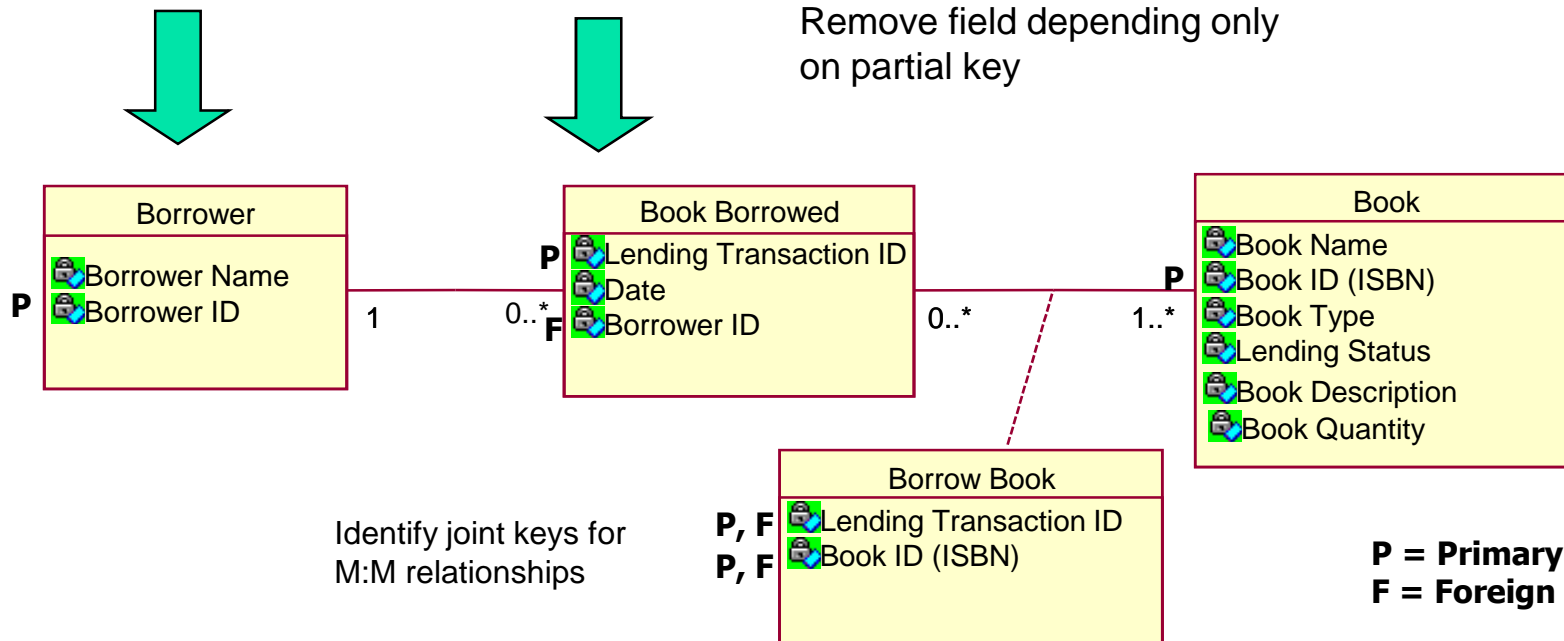
Optimizing RDBMS-based Object Storage



Borrower Name only needs
Borrower ID

Date only needs
Lending Transaction ID

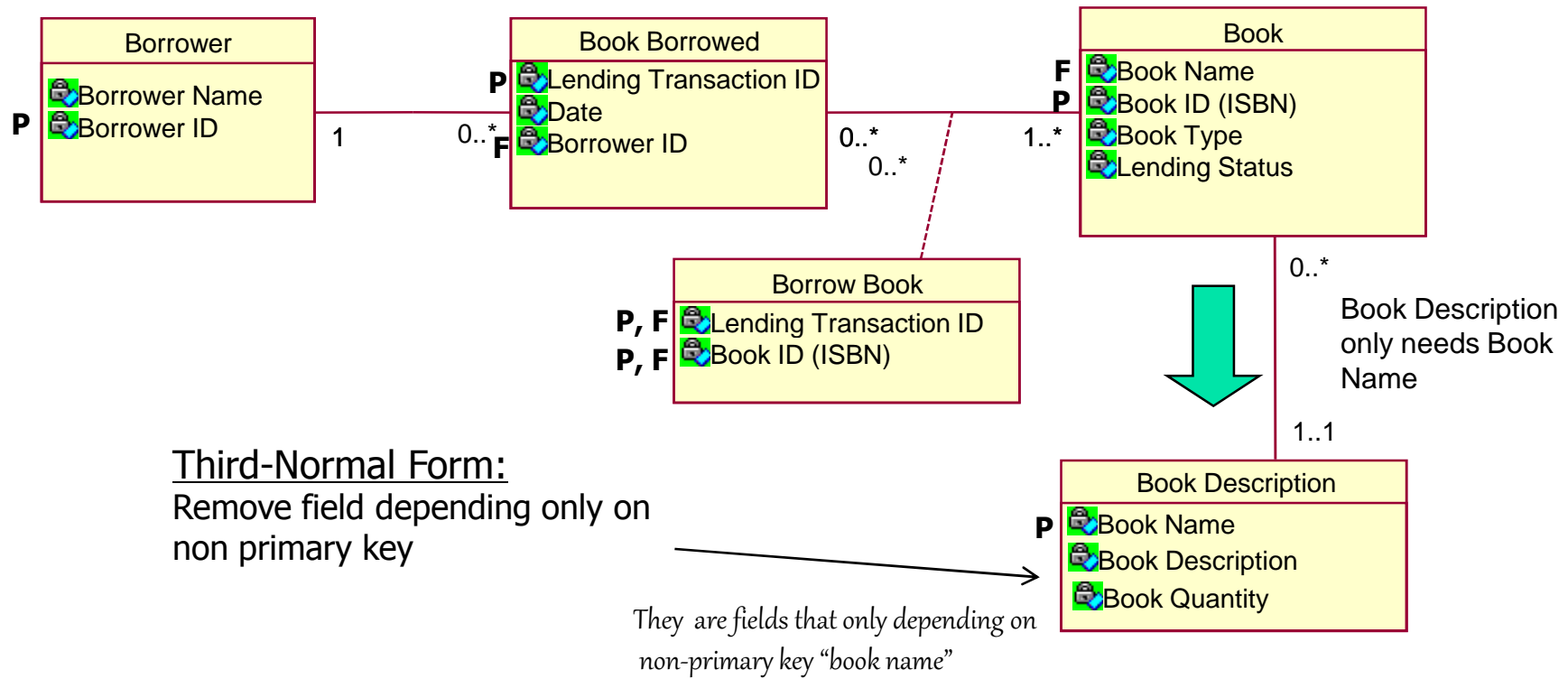
Second-Normal Form:
Remove field depending only
on partial key



Identify joint keys for
M:M relationships

P = Primary Key
F = Foreign Key

Optimizing RDBMS-based Object Storage



P = Primary Key
F = Foreign Key



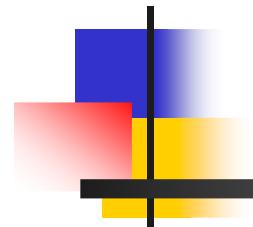
Optimizing Data Access Speed

- De-normalization
 - Reversing the normalization for performance
- Clustering
 - Adding data fields (columns) to other tables for faster data retrieval (e.g. Borrower Name are in both Borrower and Book Borrowed)
- Indexing
 - An indexing table has pre-sorted index based on a type for faster search (e.g. by “Text-book”, “Reference-book” and “novel” pointing at each data record as groups)



Design Considerations of Data Persistency

- Store data in local memory for high access-frequency data that requires less update needs.
- Do not store data in local memory for low access-frequency data that may require frequent update needs.
- Make a design trade-off when data are high access-frequency and frequent update needs. The bottom line is the balance of time vs. space.



Implementation and Testing

Part 17



Manage Implementation

- Resource Assignments
 - GUI development
 - Specialized business function
 - Foundation/Shared function
 - Data accessing/persistence
 - Networking/infrastructure communication

- Manage the schedule
 - Development selections
 - Development dependencies
 - Change management
 - Incremental testing



Avoid Mistakes

- Using “Bleeding Edge” Technology
 - Inexperience
- Using Low-cost Personnel
 - Inexperience
 - Low quality
- Lack of Code Control
 - No change management
 - Lack of development code standards
- Inadequate Testing
 - No rigger test
 - No early test cost later major testing difficulties



Test Planning

- Unit/Component Testing
- Integration Testing
- System Testing
- Acceptance Testing



Unit/Component Testing

- White-box Testing
 - Code
 - Error handling

- Black-box Testing
 - Methods
 - Interfaces
 - Error test



Integration Testing

- User Interface Testing
 - User interface functions
 - Error conditions
- Use Case Testing
 - Use cases
 - Extended use cases
 - Special conditions
- Interaction Testing
 - Business logic
 - Business transaction (data processing)
- System Interfacing Testing
 - Data exchange



System Testing

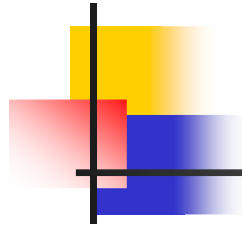
- Requirement Testing
 - Business requirements
 - Business process status/conditions
- Usability Testing
 - Easy of use
 - Performance
 - Error reporting
- Security Testing
 - Authorization
 - Authentication
- Reliability Testing
 - System Monitoring
 - Disaster recover
- Performance Testing
 - Stress
 - Scalability
- Documentation Testing
 - Accuracy of HELP, procedure, tutorials



Acceptance Testing

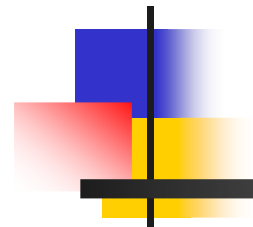
- Beta Testing (pre-release)
 - User environments
 - User data

- Alpha Testing (post-release)
 - Conducted by users
 - Acceptance test



Example of Test Plan Format

Use Case Name	Test Type	Test Conditions Settings / Field Values	Test Pattern (steps)	Expected Results	Actual Results	Pass / Not Pass	Comments



Final Course Review

Part 18



Review of System Analysis and Design

- Capture the business function process
 - Create use cases
 - Use activity models for detail process for each use case as needed
 - Identify requirements

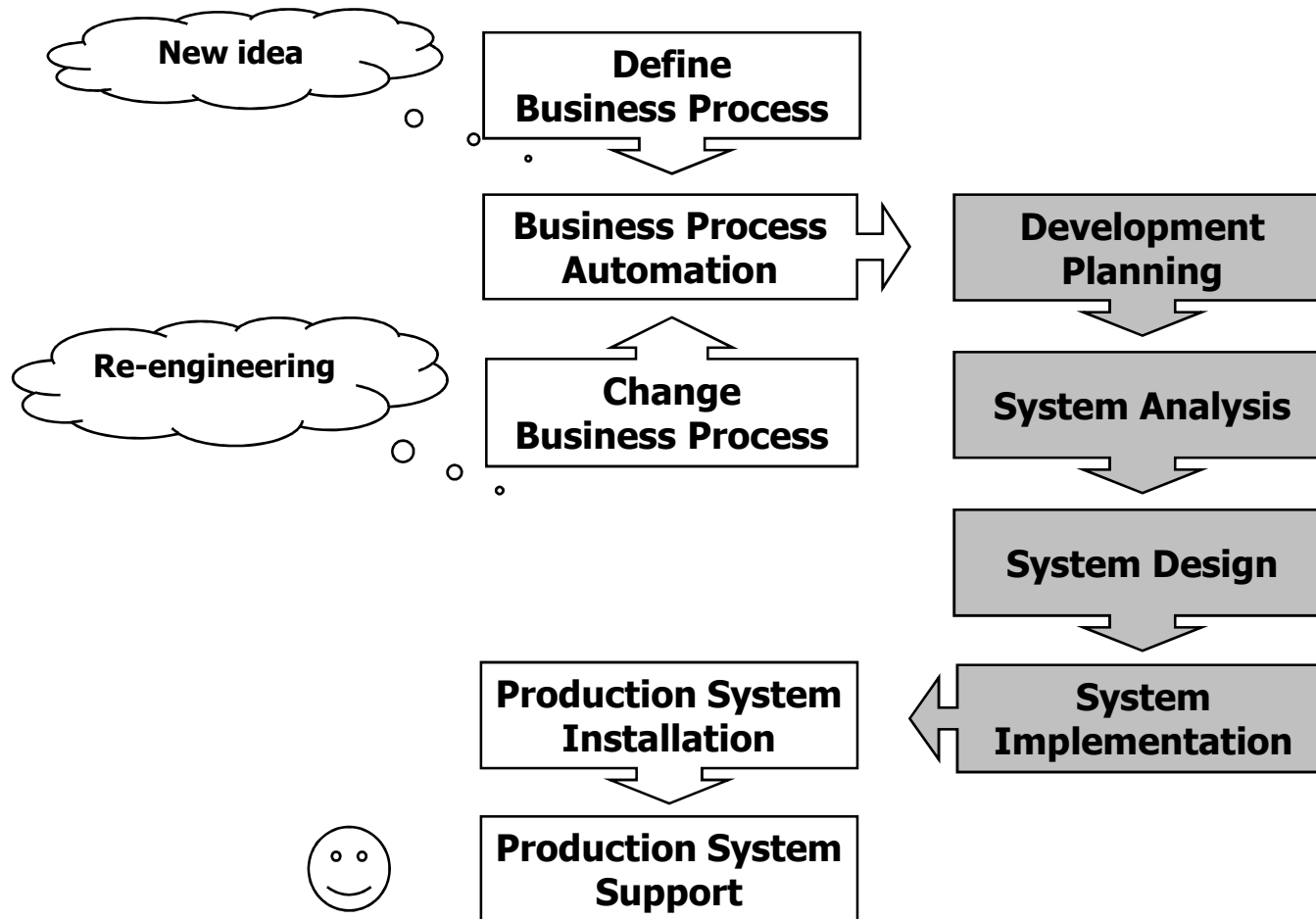
- Analyze the system components to realize the business needs
 - Class structures to support the business data and functions
 - Model their interactions
 - Model their internal behaviors



Review of System Analysis and Design (continued)

- Create system design models
 - Transform the Analysis-level models as base for creation of design
 - Decompose the high-level classes
 - Add design structures to support business objects
 - Add interfaces with underlying development framework, library, system component, utilities
 - Design specific components using class diagrams, interaction diagrams, state diagrams:
 - User interfaces
 - Business components/objects
 - Interfaces with other applications or components
 - Interfaces to the back-end data accessing capabilities
 - Define deployment model:
 - User environments
 - Business process distributions
 - Operational environment, network, security
 - Hardware specifications

Review of System Development Process





Key Learning from This Course

You have learned

- **Software system analysis and design**
 - Software analysis and design concepts
 - Analysis and design artifacts

- **Software system development lifecycle**
 - Fundamental analysis and design process
 - Software system architect basic skills



Thank you ...

**Wish you all
have great accomplishment
in your academic studies
and
feature success !**



Student Practice

Appendix



Student Practice – *Create Use Case Model*

- Add a use case of “Renew book”
- Write use case descriptions for “Check-out Book”
- Students present models



Student Practice – *Create Analysis Model*

- Create a collaboration diagram for “Check-in book”
- Add new class structures to support
 - “Magazine”
 - “Staffs” of the school
- Student presents models



Student Practice - *Create Design Models*

- Create application flow for “check-in book” and “check-out book”:
 - User interface
 - System application logic flow (UI, flow control, data access)
 - Create database table
 - Show deployment diagram
- Students present models