

# SOFTWARE ENGINEERING

Chapter 3

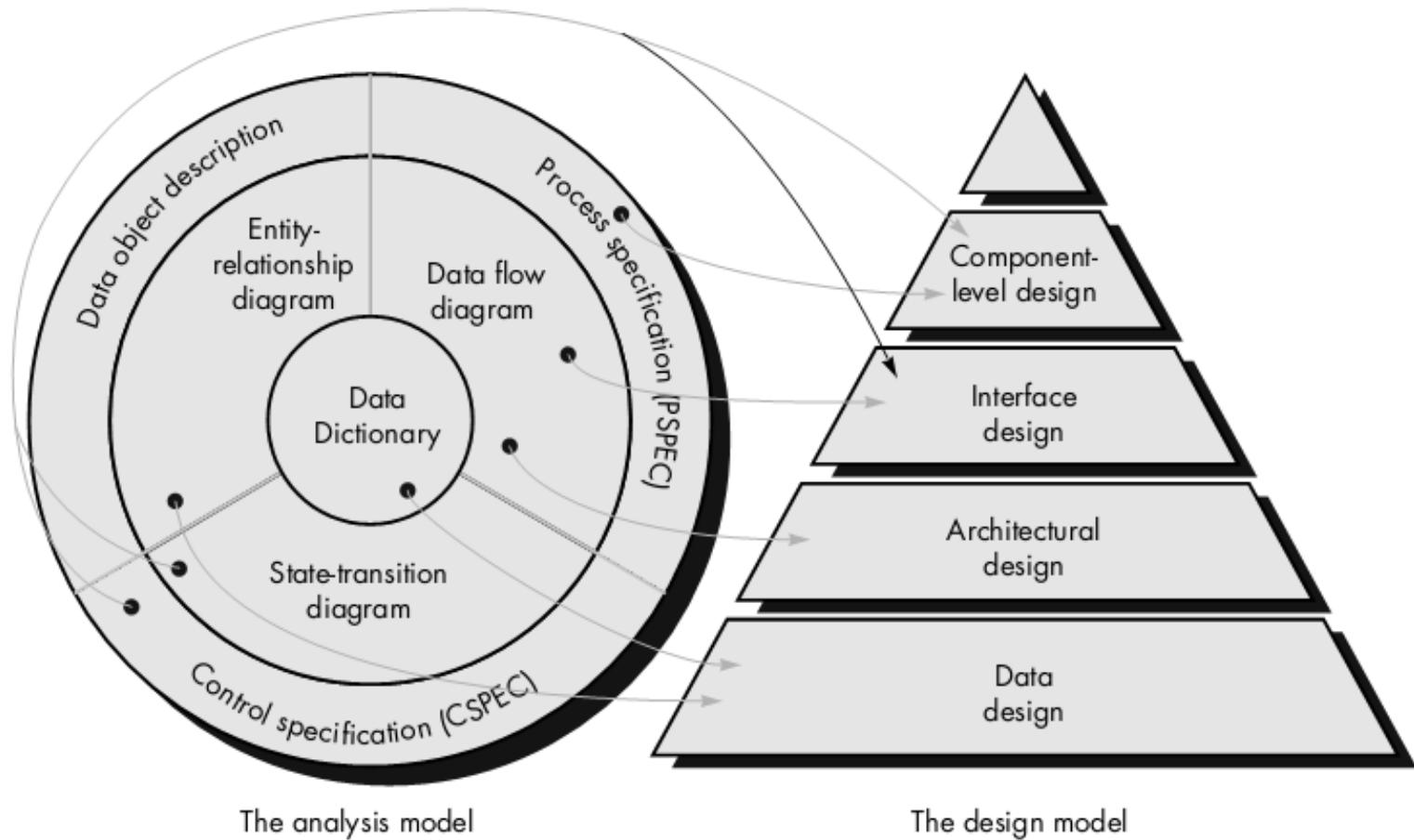
Software design

# Outline

- Overview of software design
- Architectural design
- Database design
- Interface design

# **Overview of software design**

# Overview



# Design Activities

- *Architectural Design:* Defines the overall structure of the system, its main components (subsystems or modules), their relationships, and how they are implemented.
- *Database Design :* Design the data structure for the system and how it is represented in the database.
- *Interface Design:* Define interfaces between system components.
- *Component selection and design:* Look for reusable components. If not available will design these components.

# Principles of design

- Design should not be confined to a narrow view
  - Should be selected from different solutions
- Do not redo existing designs (solutions)
  - Need to reuse as much of existing designs as possible

# Principles of design

- Design representation must be consistent and highly integrated
  - Must agree on how to perform and communicate because design work can be done by many people
- Design should have a flexible structure (easily change if needed)
  - Must be modularized

# Principles of design

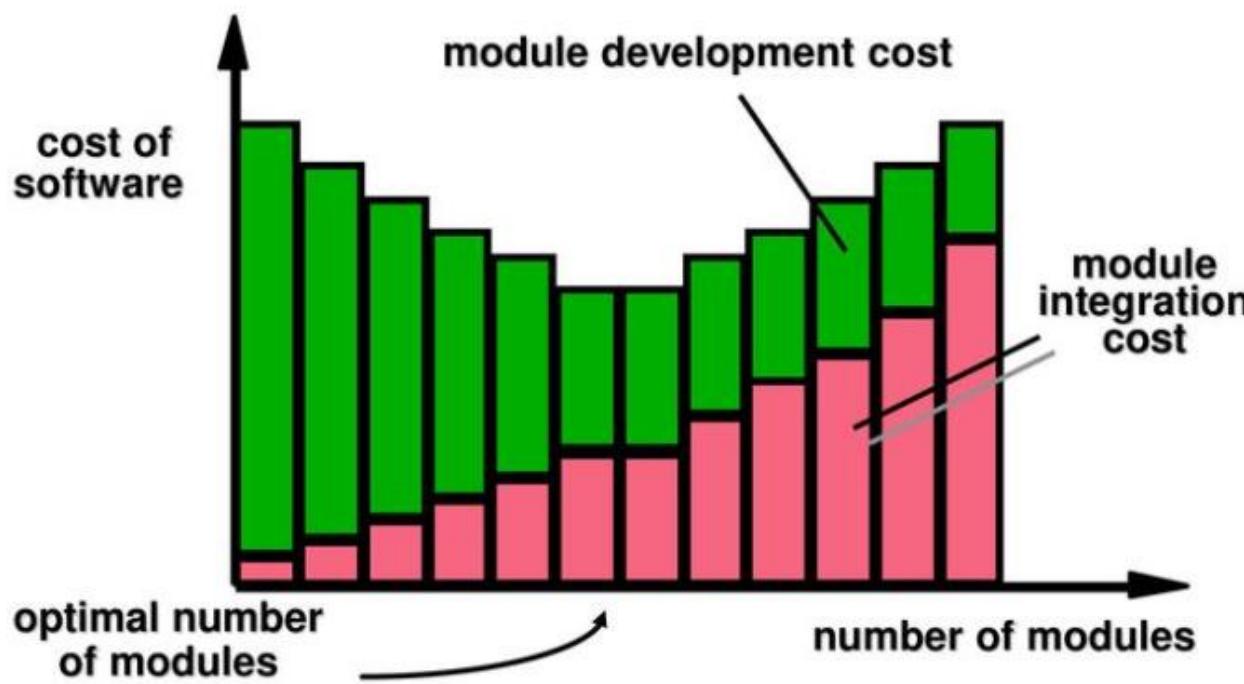
- Design needs to be evaluated while it's being created
  - Through measurements: cohesion, coupling
- Design needs to be validated and re-checked to avoid systematic errors
  - Finding errors: Missing functions, unclear functions, conflicts with requirements specification.

# Modularization

- Principle: divide and conquer
  - Complexity:  $C(p1+p2) > C(p1) + C(p2)$
  - Execution effort:  $E(p1+p2) > E(p1) + E(p2)$
- Benefits:
  - Reduced complexity, easy to upgrade, modify
  - Increased efficiency because of the possibility of parallel development.
  - Increased reusability

# Modularization

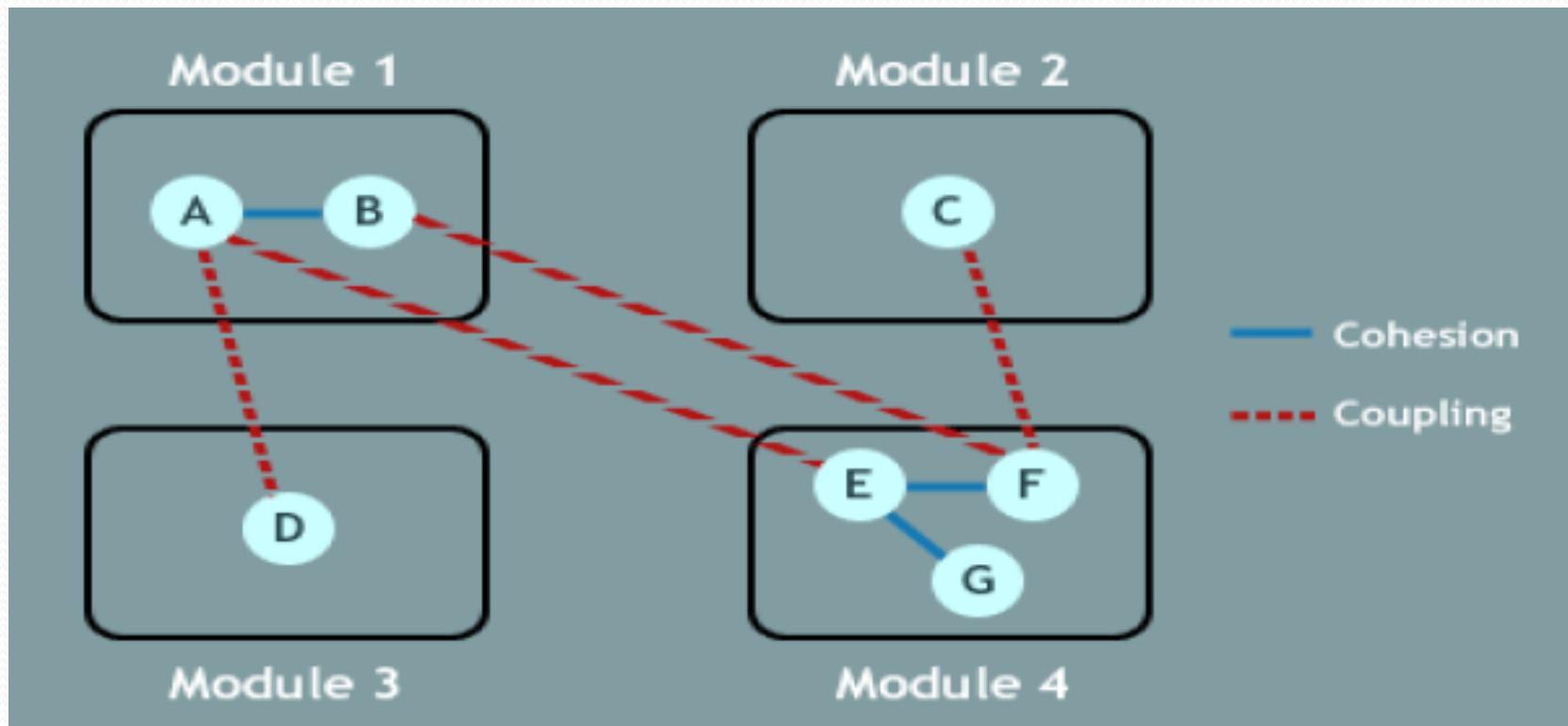
- The number of modules is determined based on the concept of functional independence



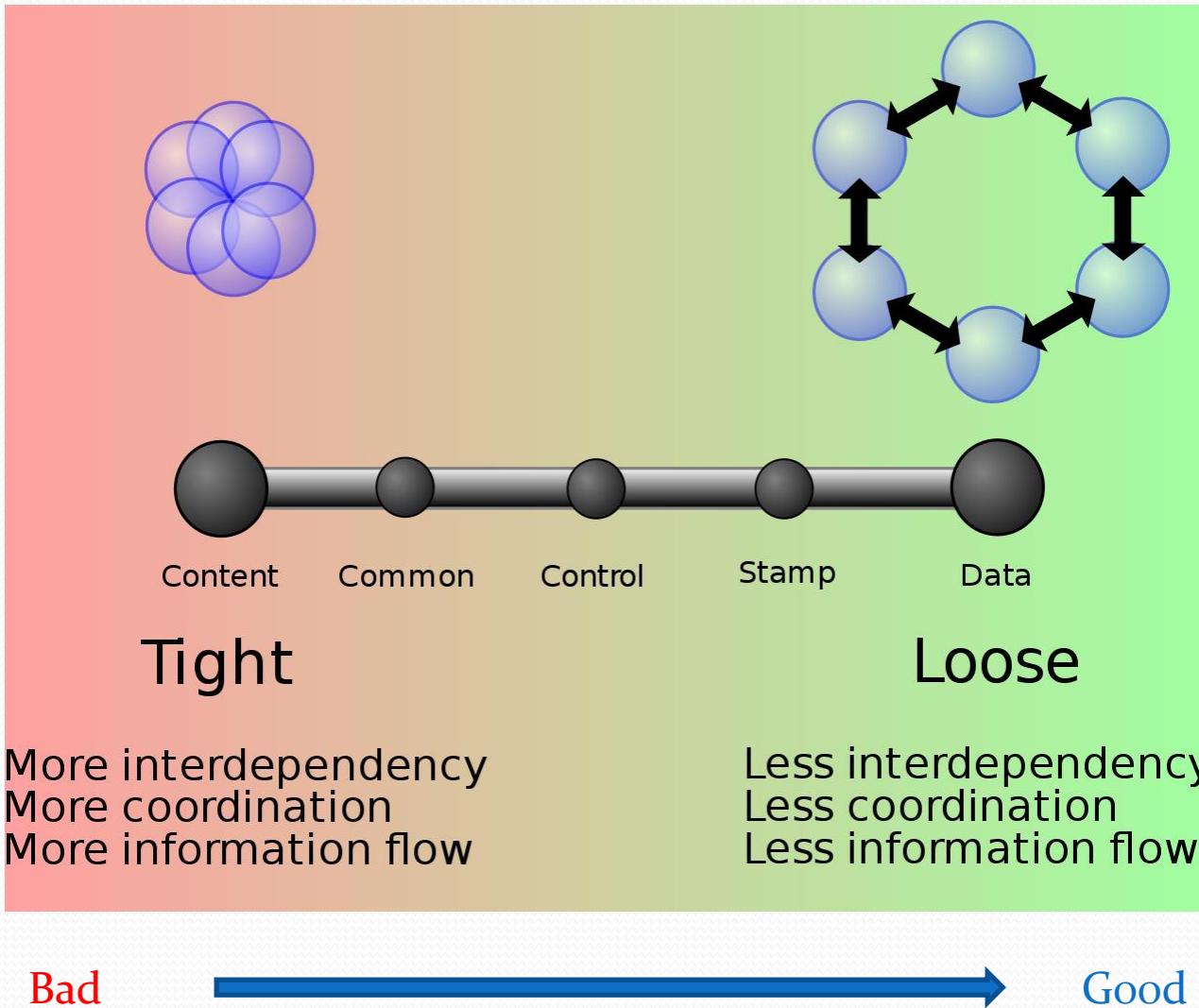
# Design quality metric

**Coupling** : The degree of interdependence between modules.

**Cohesion**: The degree of relevance of the components in the module.

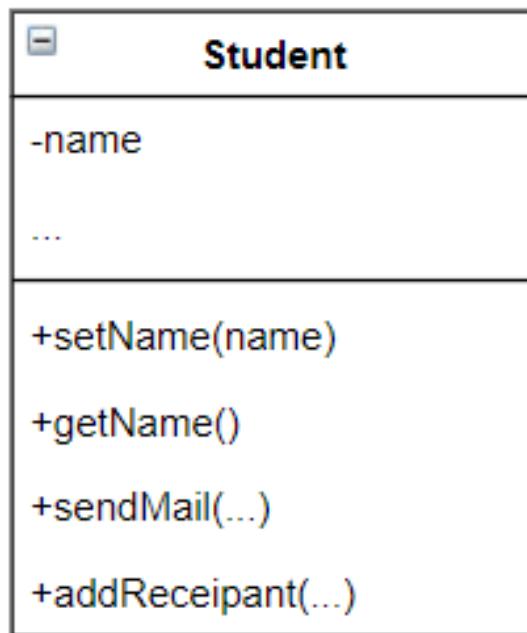


# Coupling

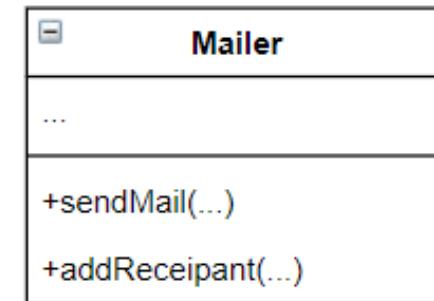
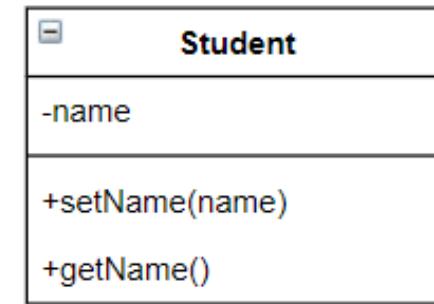


# Cohesion

**Low cohesion (bad)**



**High cohesion (Good)**



# Architectural design

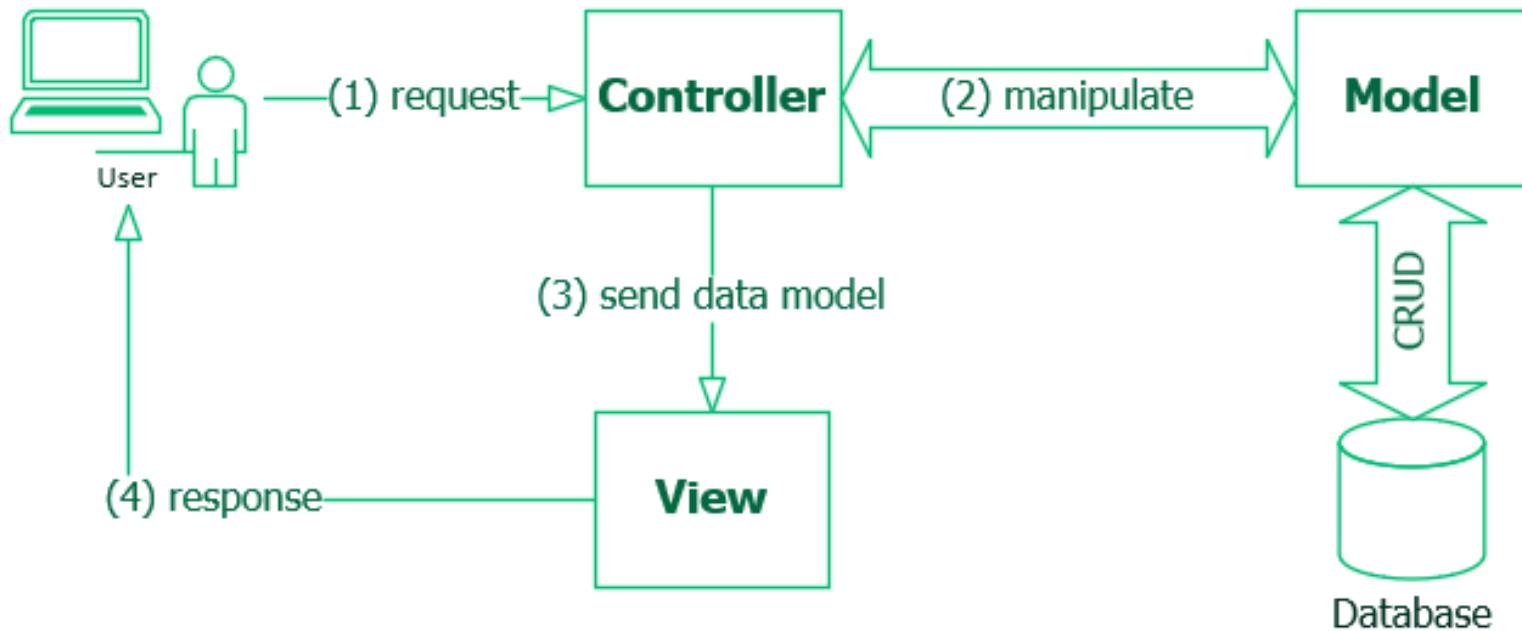
# Overview

- Architectural design involves understanding how a software system is organized and designing the overall structure of that system.
- The output of the architectural design is an architectural model consisting of a set of components that communicate and interact with each other.
- Architectural design is the earliest phase in the design process. Refactoring the system architecture is often expensive because it affects a lot of components in the system

# Architectural pattern (model)

- Patterns are a means of representing, sharing, and reusing knowledge.
- An architectural pattern is a pattern of good design practice, tried and tested in different environments.
- Templates often include information on pros/cons and when to use them.
- Patterns can be represented by tabular and graphical descriptions.

# MVC model



Separate display and interaction of system data. The system is structured into three logical components that interact with each other.

The **Model** component manages system data and the operations associated with that data.

The **View** component defines and manages how data is presented (displayed) to the user.

The **Controller** component manages user interactions and passes these interactions to the **View** and **Model** respectively.

# MVC model - Example

2. Controller requests for customer  
with id 3 from model



3. Model searches database for  
customer with id 3

4. View formats customer data into  
human readable format, sends to  
browser

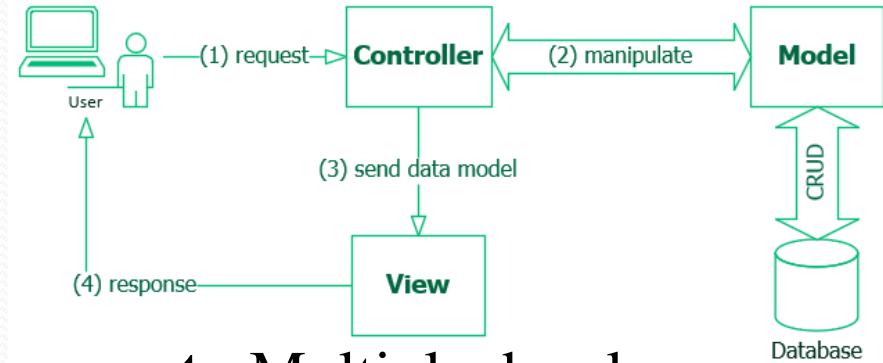
# MVC model

## Pros:

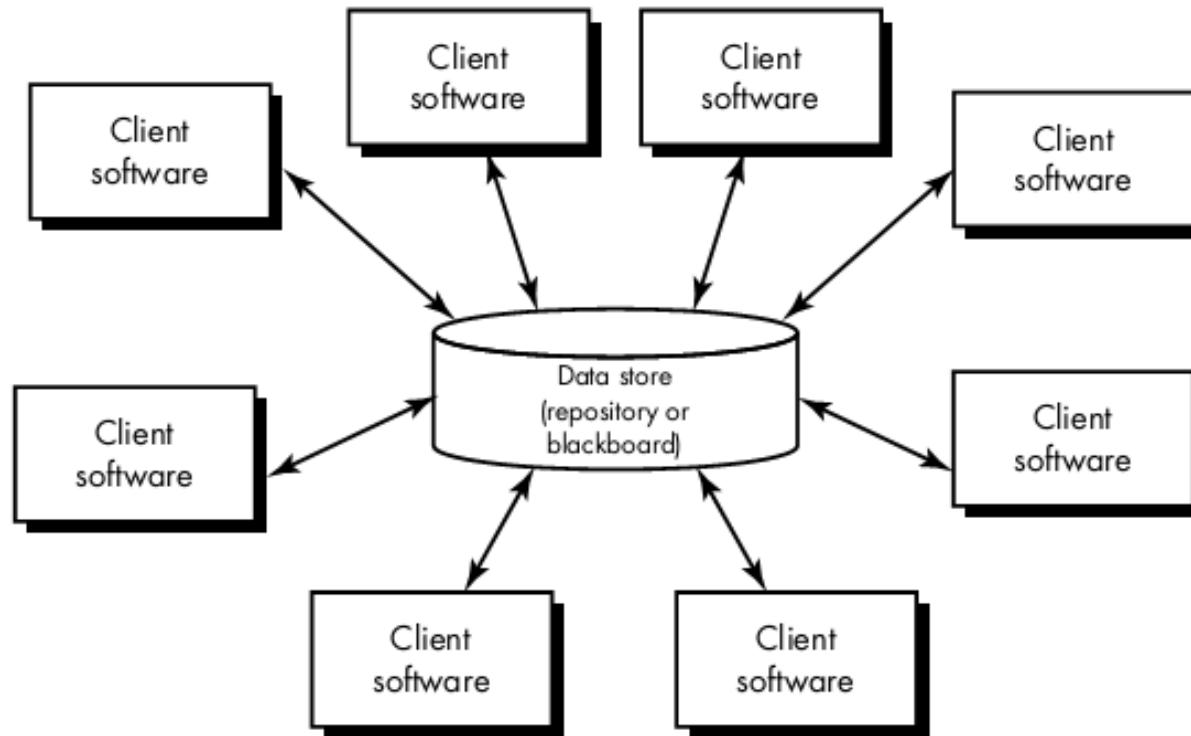
- **Concurrent (parallel) development** : Multiple developers (devs) can work on Model, View and Controller simultaneously.
- **High cohesion** : MVC allows logical grouping of related actions in a Controller, Data manipulation of a table (CRUD) in a Model.
- **Loose Coupling**: The essence of the MVC framework is that there is a low coupling between Model, View and Controller.
- **Easy to maintain**: Easy to change and add features later.

## Cons:

- Increases programming complexity and requires more programmer skills. Not suitable for applications with simple interaction and data models.



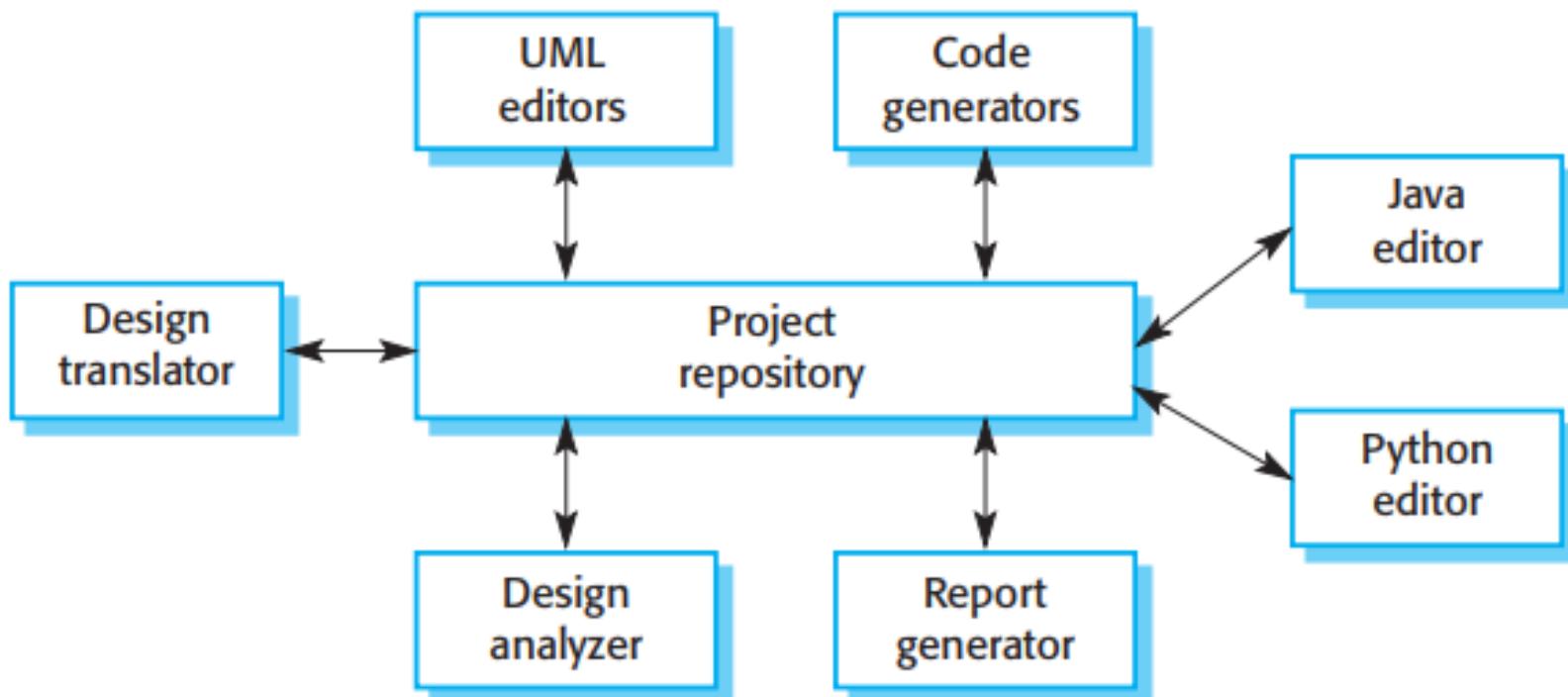
# Repository (Data centric) model



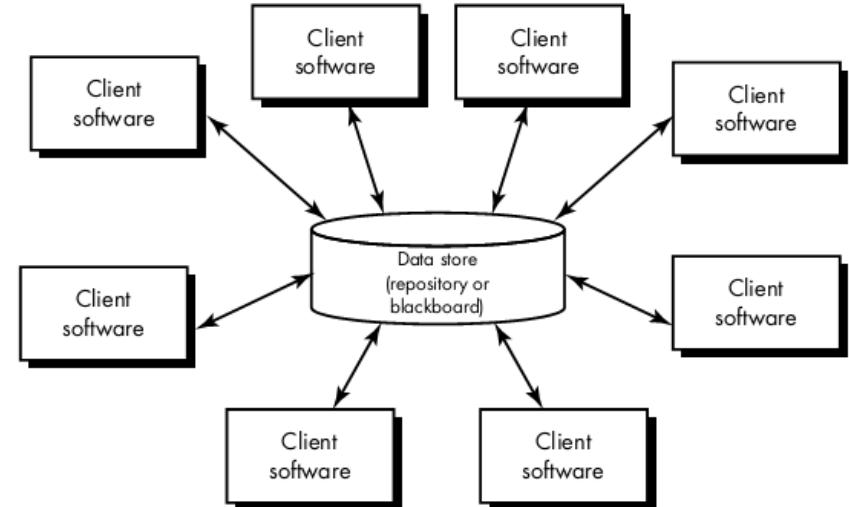
All data in a system is managed in a central repository. All subsystems can access this data store. Subsystems do not interact directly (interact only indirectly through repositories)

# Repository (Data centric) model

- Example: Repository of IDE



# Repository model



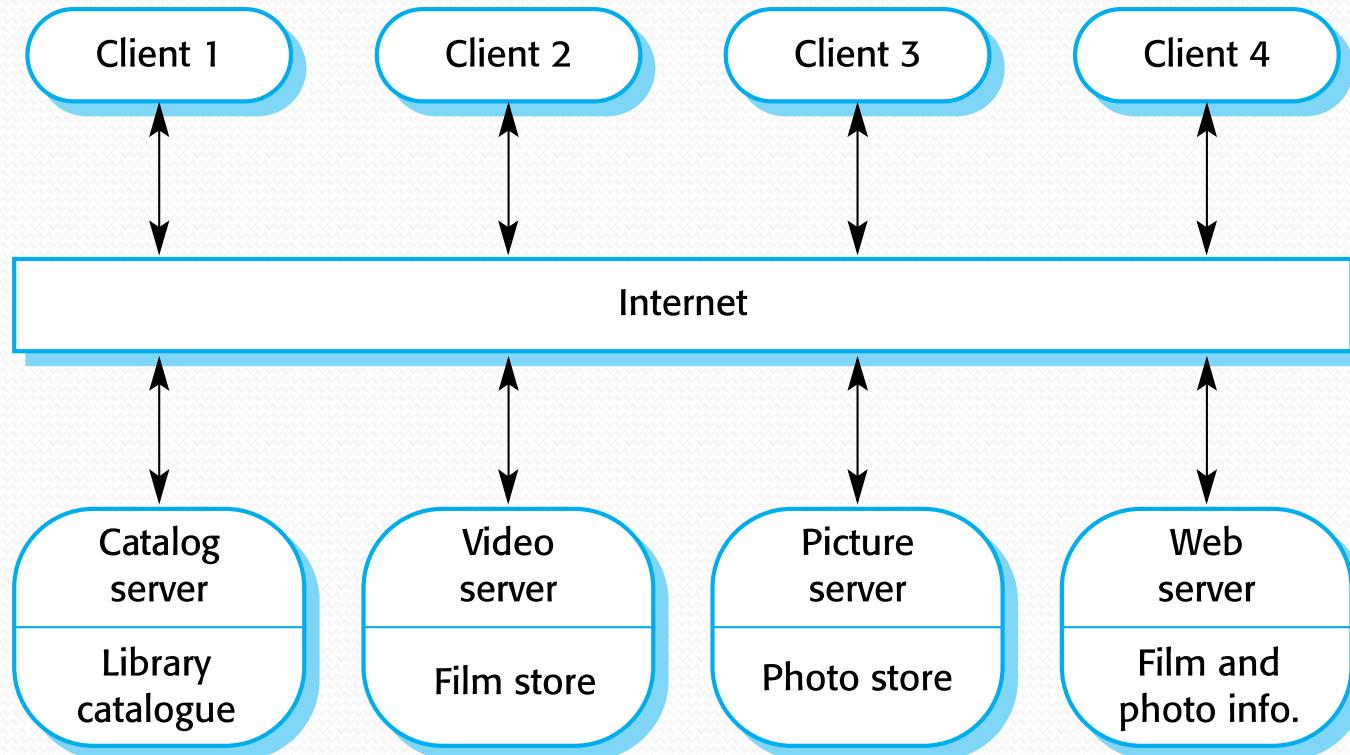
## Pros:

- Components can be independent, regardless of the existence of other components. Changes made by one component can be propagated to all other components.
- All data can be managed consistently and with integrity because it's all in one place.

## Cons:

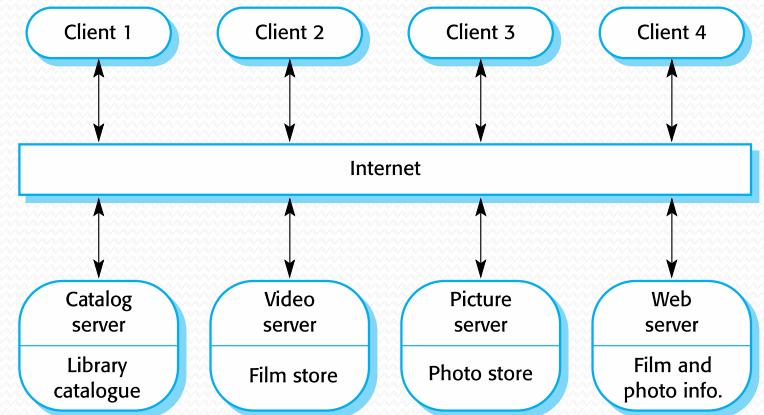
- When the Repository has problems (security, errors) it will affect all subsystems.

# Client – Server model



System functions are organized into services, with each service delivered from a separate Server. Client is the person who uses these services by accessing the servers.

# Client – Server model



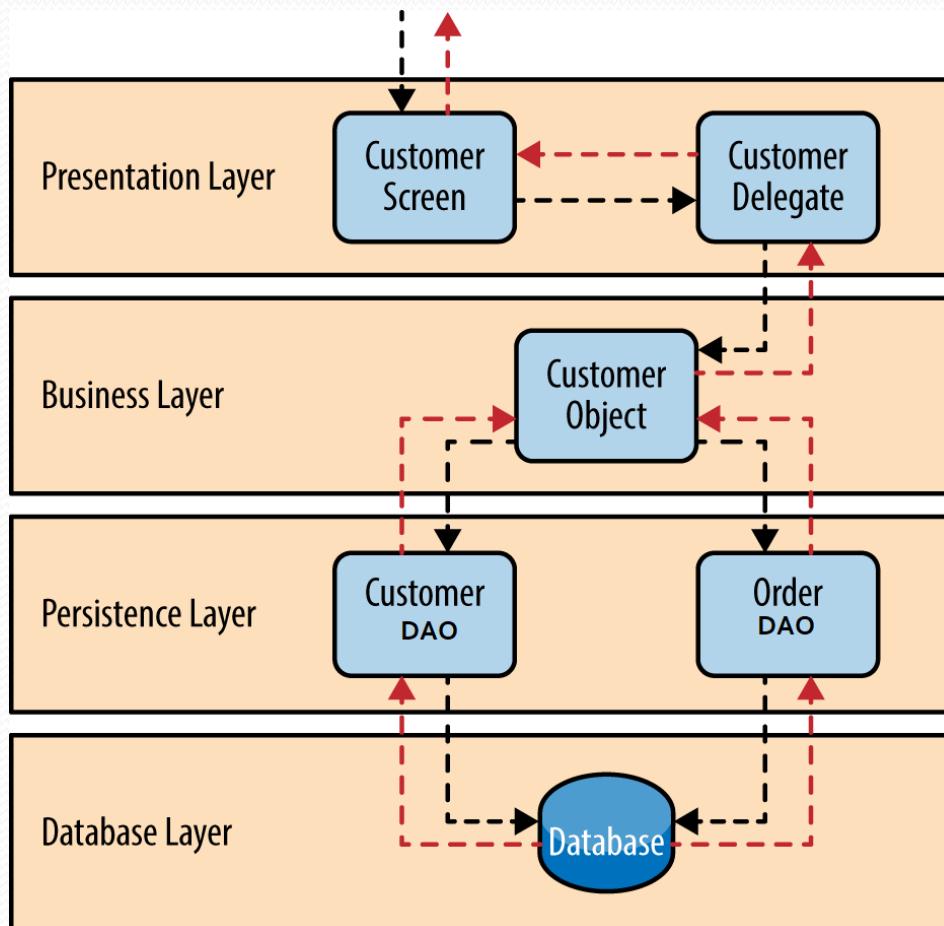
## Pros:

- The main advantage of this model is that the service servers can be distributed on the same network.
- Easily add or upgrade services without affecting other services.

## Cons:

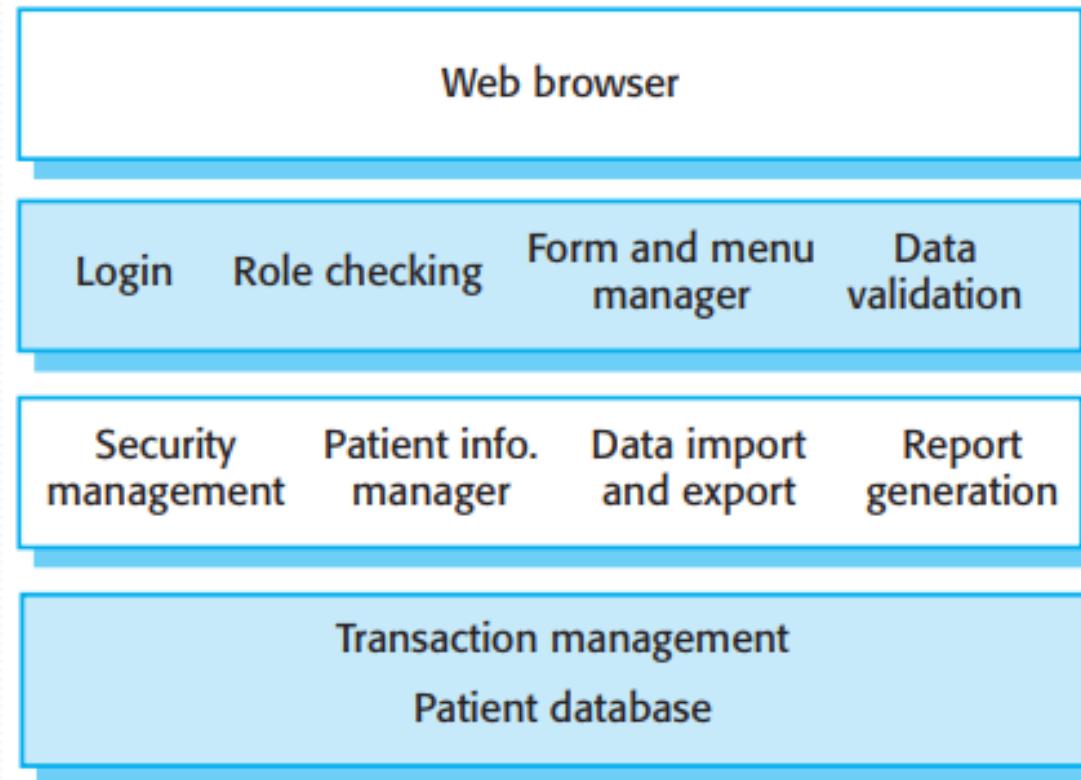
- Vulnerable to Denial of Service (DOS, DDOS) attacks.
- Data may be duplicated, redundant

# Layered model

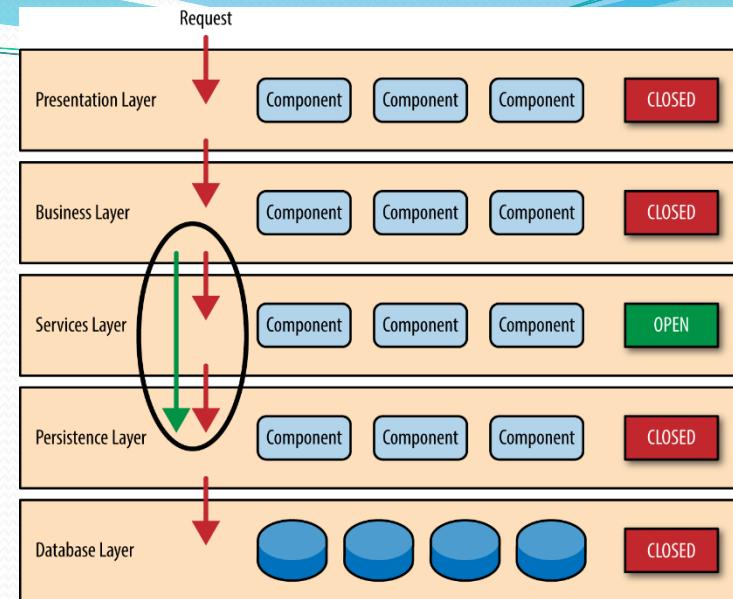


- Organizing the system into layers
- Each layer is a collection of related functions/services.
- A layer provides services to the layer above it.
- The lowest level layers represent core services that are likely to be used system-wide

# Layered model



# Layered model



## Pros:

- Allows replacing the entire layer as long as communication is maintained.
- Redundancy schemes (e.g., authentication) can be provided in each layer to increase system reliability.

## Cons:

- In practice, it is often difficult to clearly separate classes, and a high-level layer may have to interact "beyond the level".

# Database design

# Database design levels

Concept  
(ERD, Class)

Logic  
(Tables)

Physic  
(DBMS)

# Database design levels

Concept  
(ERD, Class)

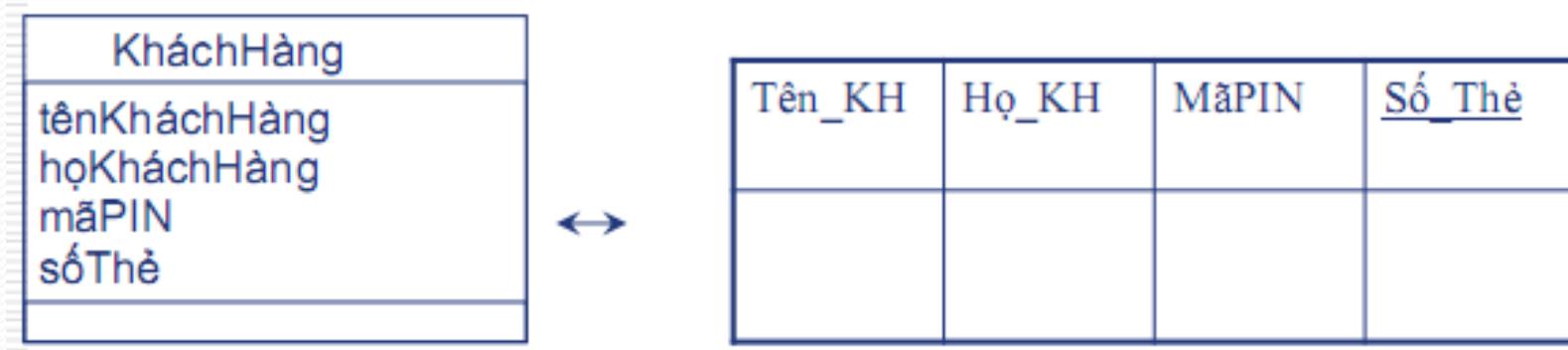
Logic  
(Tables)

Physic  
(DBMS)

# Convert class diagram to relational DB

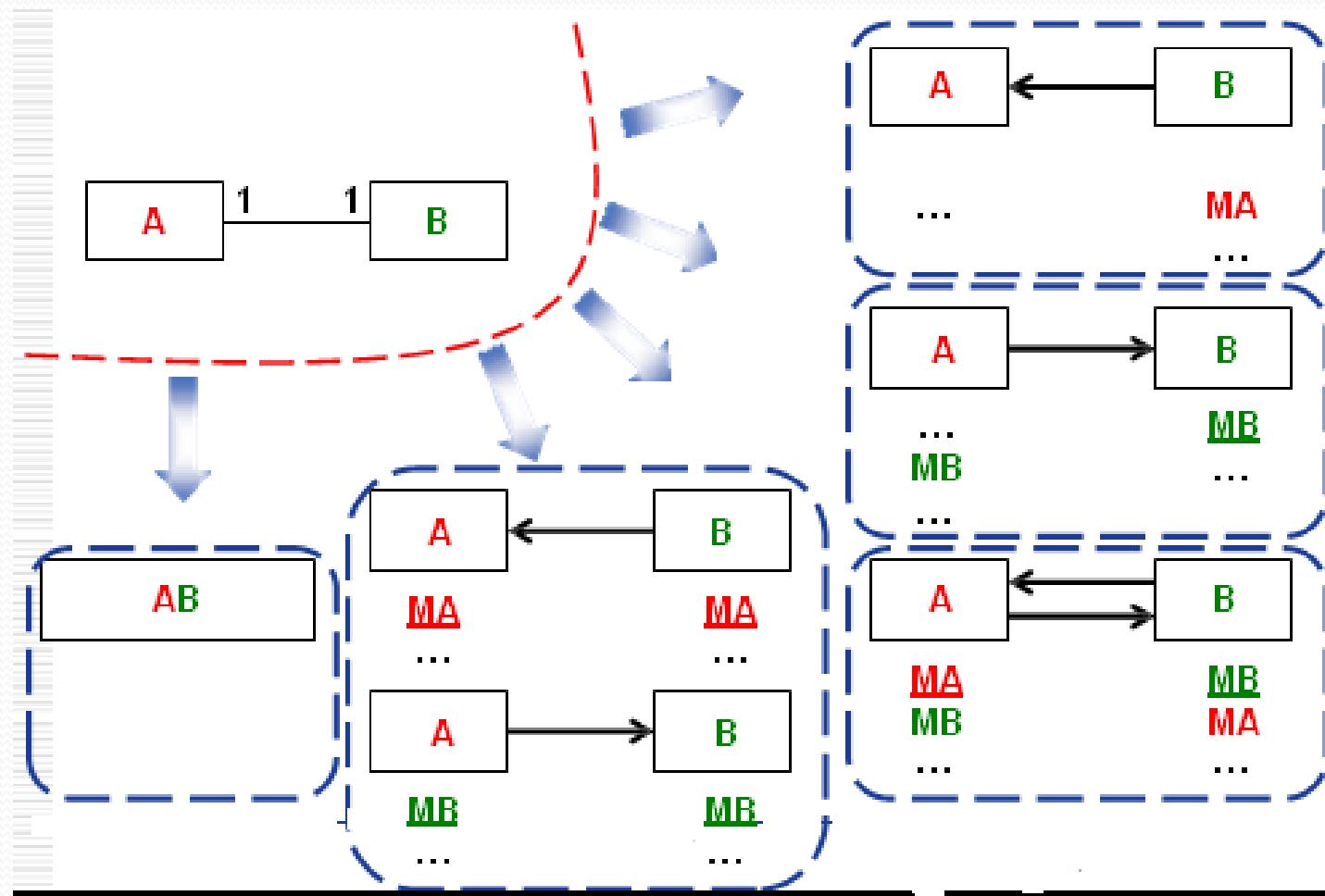
- **Convert class to table:**

- A class → a Table
- A property → a column
- A object of the class → a row (tuple) in the table



# Convert class diagram to relational DB

## 1-1 relationship: Merge or Split



# Convert class diagram to relational DB

■ 1-1

<b>KháchHàng</b>
tênKháchHàng
họKháchHàng
mãPIN
sốThẻ



<b>TàiKhoản</b>
sốTàiKhoản
loạiTàiKhoản
sốDư

Bảng KháchHàng

Tên_KH	Họ_KH	MãPIN	Số_Thẻ

Bảng TàiKhoản

Số_TK	Loại_TK	Số_Dư_TK	Số_Thẻ

(\*): Số\_Thẻ là một khoá của bảng TàiKhoản .

# Convert class diagram to relational DB

■ 1-1

KháchHàng
tênKháchHàng
hoKháchHàng
mãPIN
sốThẻ



TàiKhoản
sốTàiKhoản
loaiTàiKhoản
sốDư

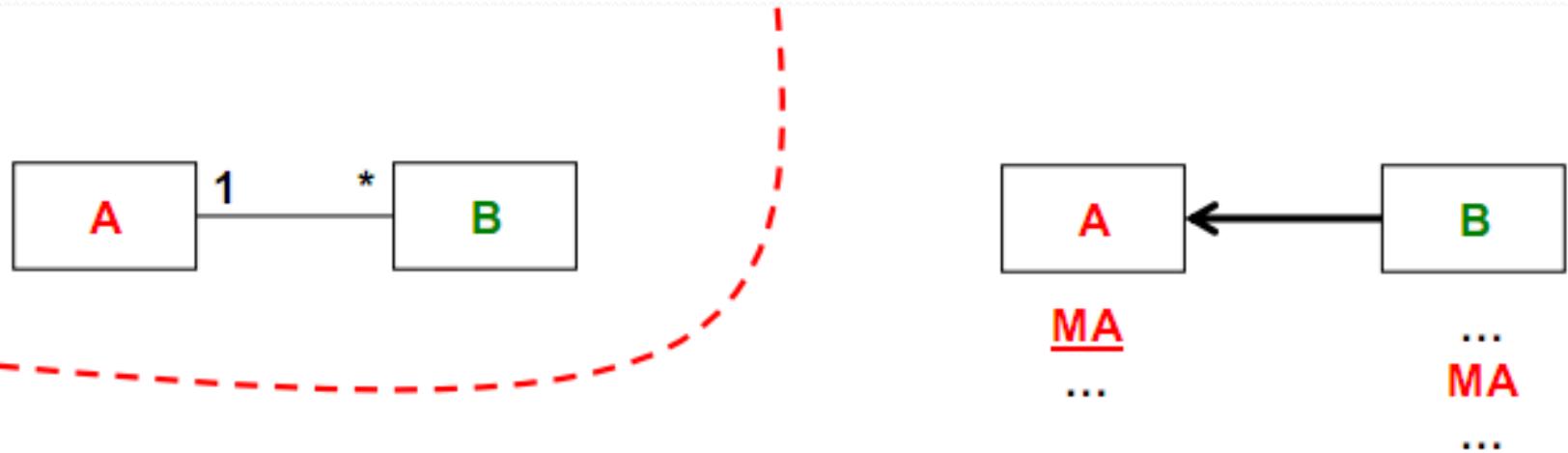
Bảng KháchHàng\_TàiKhoản

Tên_KH	Họ_KH	MãPIN	Số_Thẻ	Số_TK	Loại_TK	Số_Du_TK
			(*)			

(\*): Số\_Thẻ là một khoá của bảng TàiKhoản

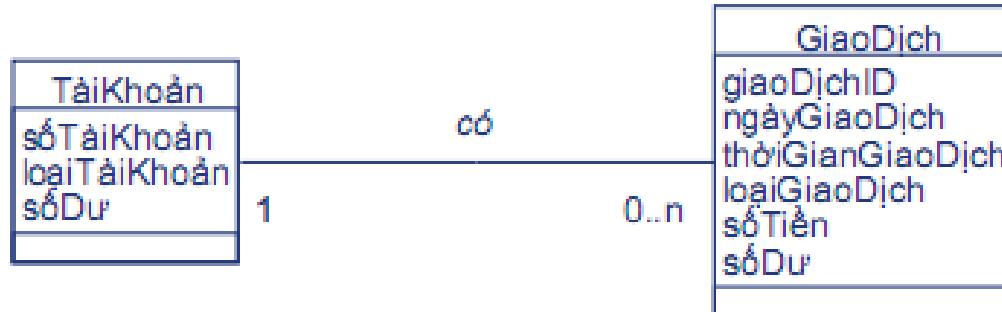
# Convert class diagram to relational DB

## 1-n relationship: Reference key



# Convert class diagram to relational DB

■ 1-n



Bảng TàiKhoản

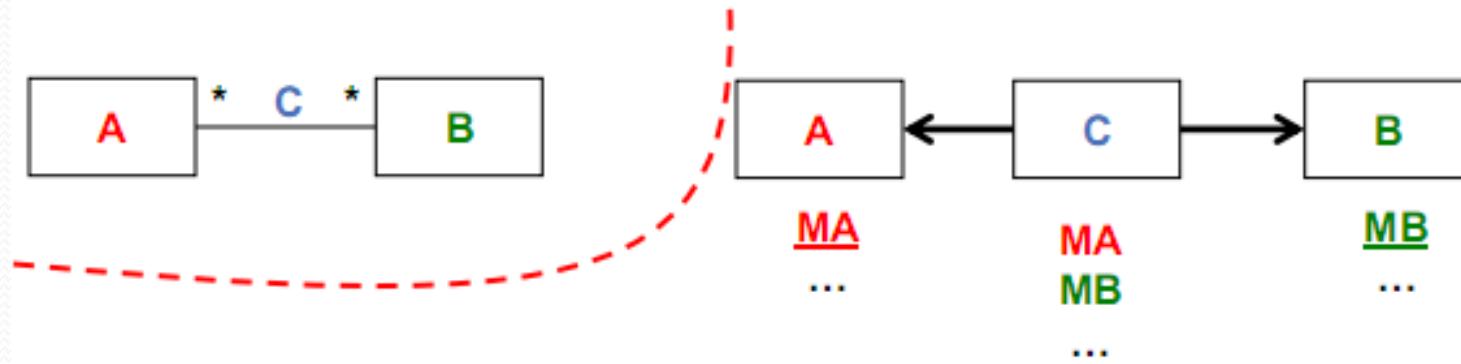
Số TK	Loại_TK	Số_Dư_TK	Số_Thẻ

Bảng GiaoDịch

GD_ID	Ngày_GD	Giờ_GD	Loại_GD	Số_Tiền	Số_Dư	Số_TK

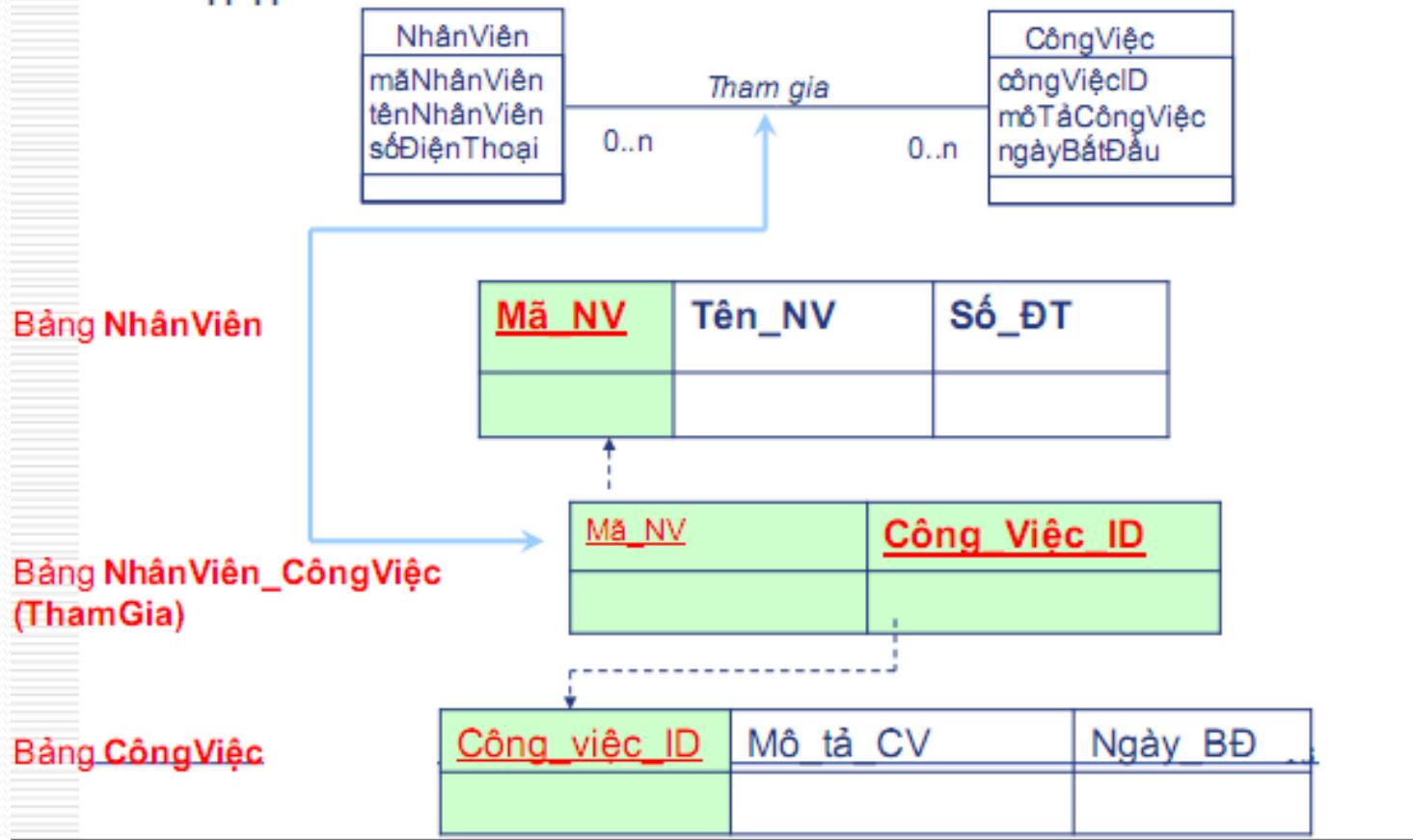
# Convert class diagram to relational DB

n – n relationship: *create an intermediate table*



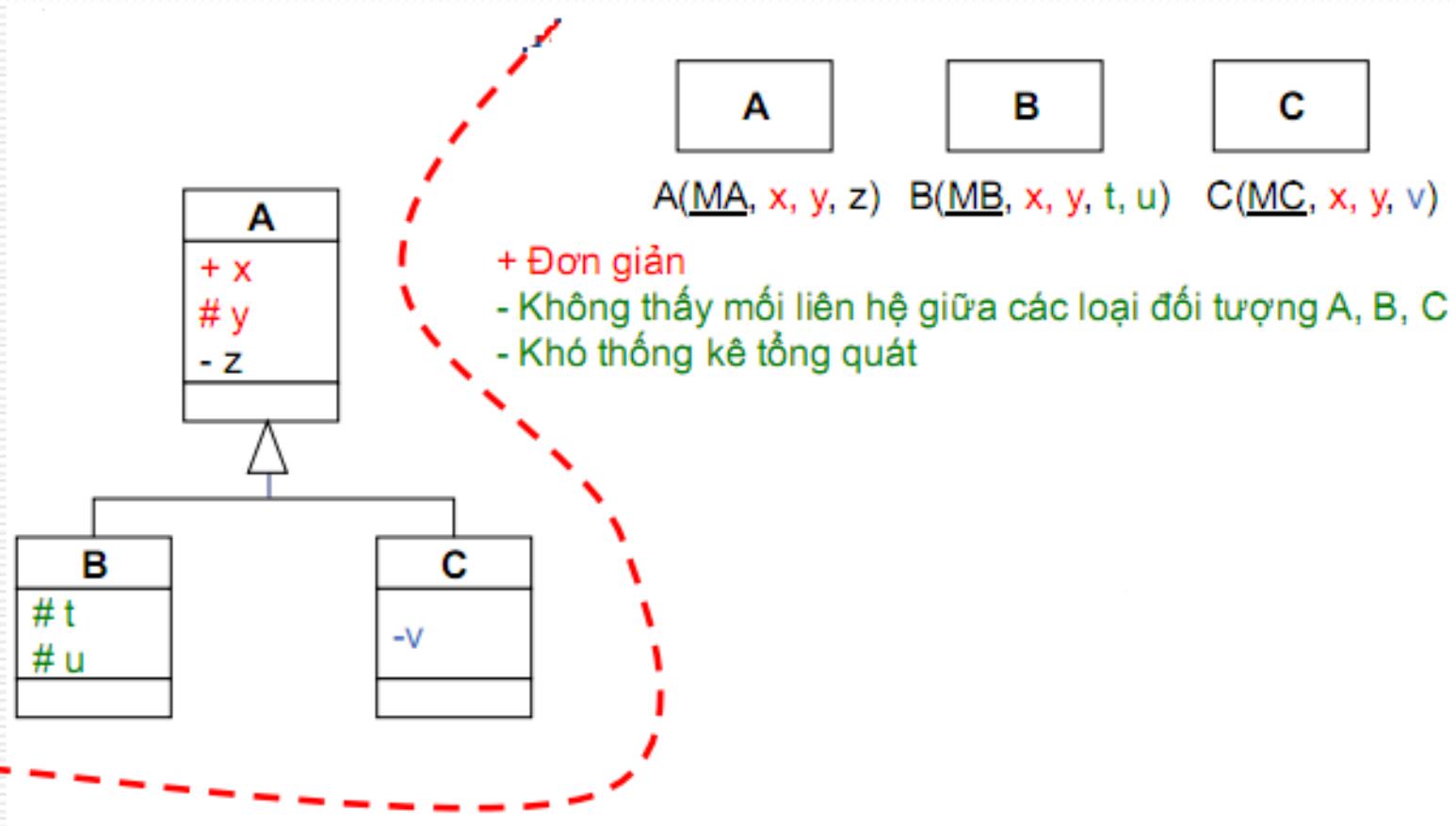
# Convert class diagram to relational DB

▪ n-n



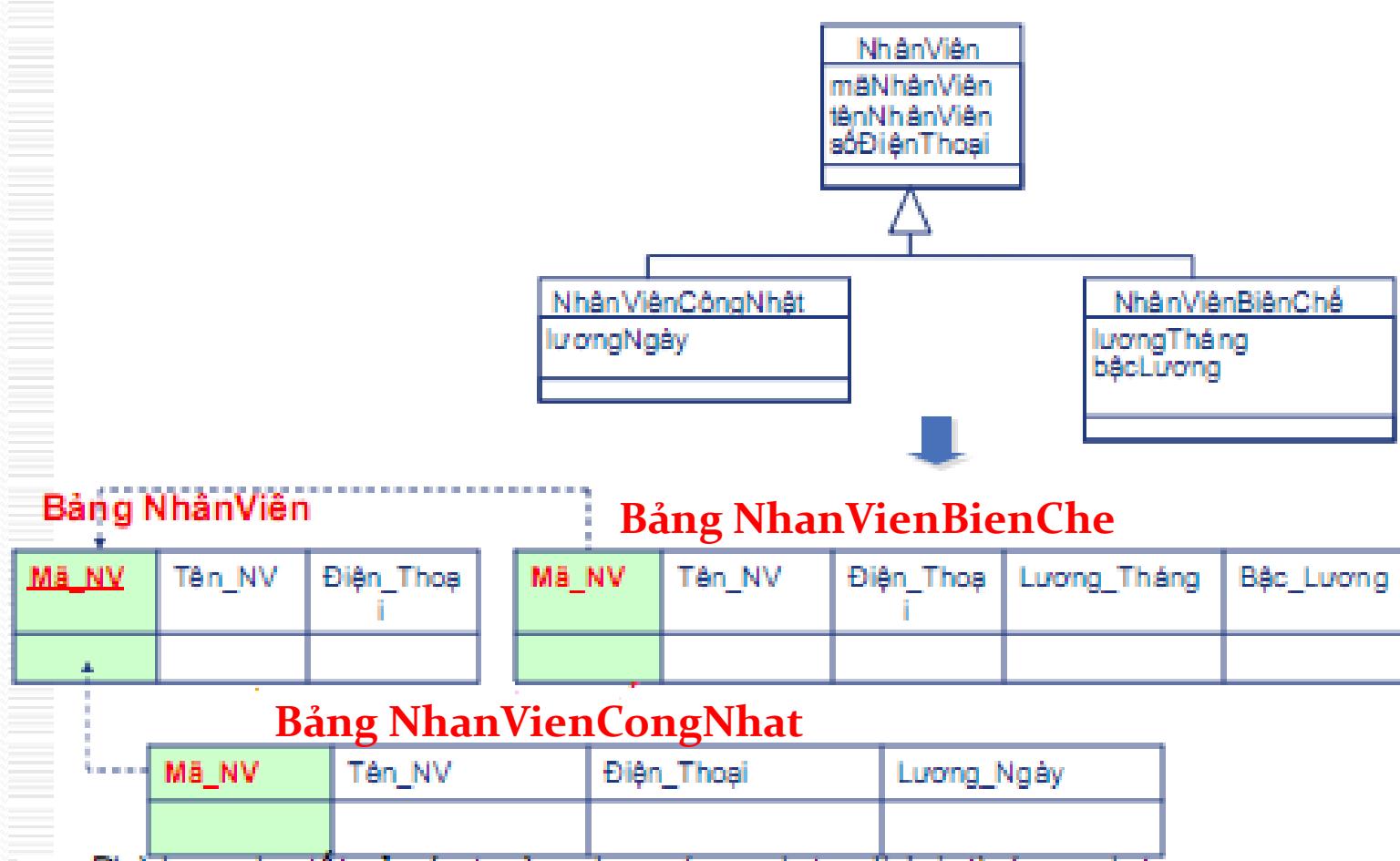
# Convert class diagram to relational DB

## inheritance relationship



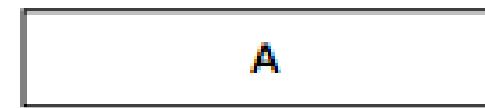
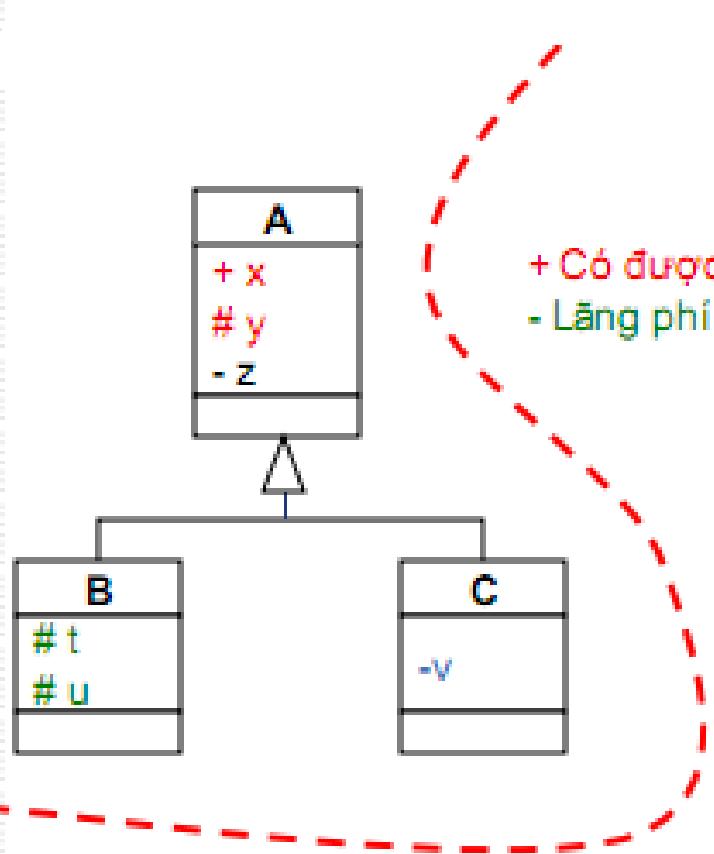
# Convert class diagram to relational DB

## inheritance relation



# Convert class diagram to relational DB

## inheritance relation

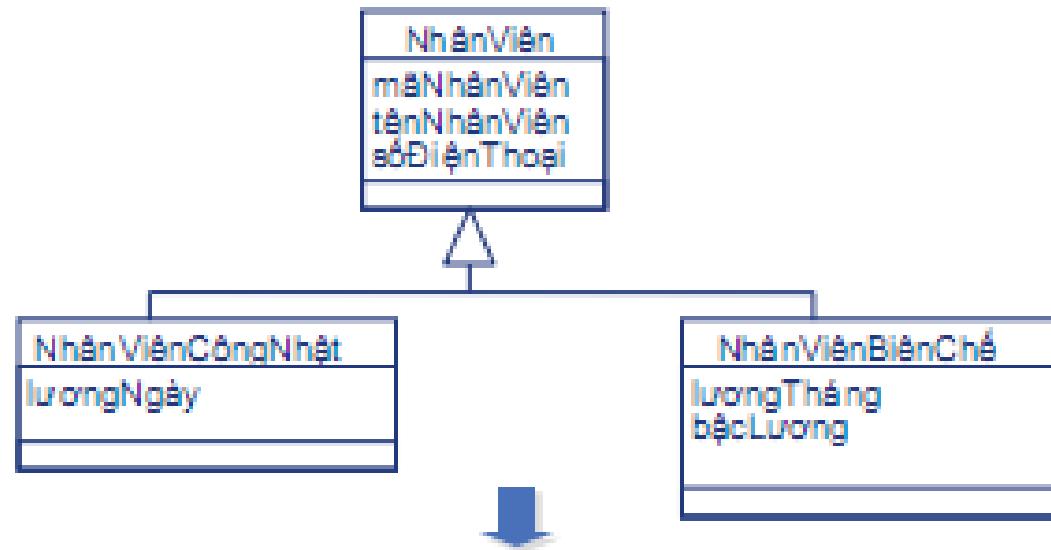


- + Có được cái nhìn tổng quát về các đối tượng
- Lãng phí không gian lưu trữ

Loại	x	y	z	t	u	v
A						
B						
C						

# Convert class diagram to relational DB

## inheritance relation - Example



Mã_NV	Tên_NV	Điện_Thoại	Lương_Ngày	Lương_Tháng	Bậc_Lương	Loại_NV

# Interface Design

# User Interface Design

## User Interface

- The user interface needs to be designed to match the skills, experience and expectations of its users.
- System users often judge a system by its interface rather than its functionality.
- A poor system interface can cause users to make very serious errors..

# User Interface Design

## Human factor in interface design

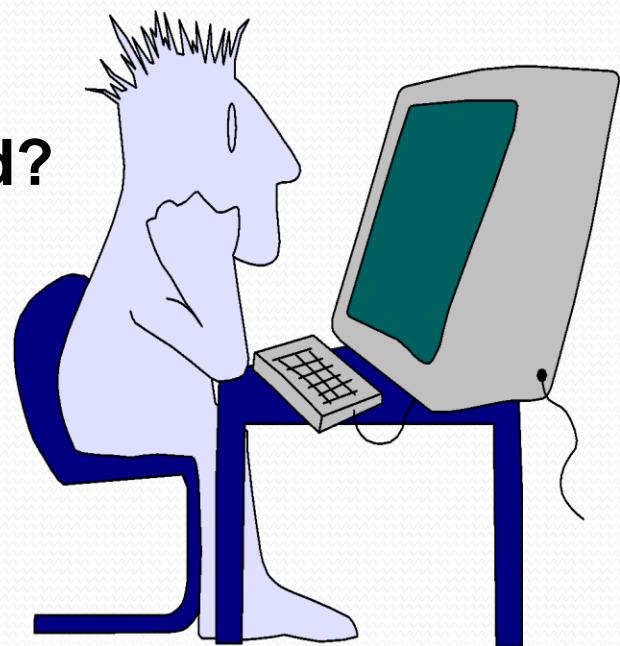
- Human's immediate memory capacity is limited.
- System users' needs
- Experience, Competence
  - Ability to use keyboard, mouse,...
  - Reaction speed, memory ability
- Interests, cultures, ages
  - Colors, languages, symbols
  - Different types of interactions: images, texts, sounds, etc

# UI Design

**Easy to learn?**

**Easy to use?**

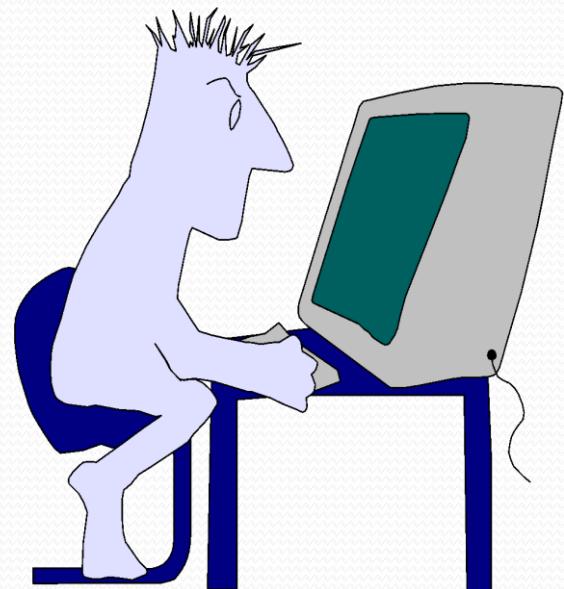
**Easy to understand?**



# UI Design

## Typical Design Errors

- Lack of consistency
- Too much memorization
- No guidance / help
- No context sensitivity
- Poor response
- Arcane/unfriendly



# Golden Rules

- Place the user in control
- Reduce the user's memory load
- Make the interface consistent

# Place the User in Control

- Define interaction modes in a way that does not force a user into unnecessary or undesired actions.
- Provide for flexible interaction.
- Allow user interaction to be interruptible and undoable.
- Streamline interaction as skill levels advance and allow the interaction to be customized.
- Hide technical internals from the casual user.
- Design for direct interaction with objects that appear on the screen.

# Reduce the User's Memory Load

- Reduce demand on short-term memory.
- Establish meaningful defaults.
- Define shortcuts that are intuitive.
- The visual layout of the interface should be based on a real world metaphor.
- Disclose information in a progressive fashion

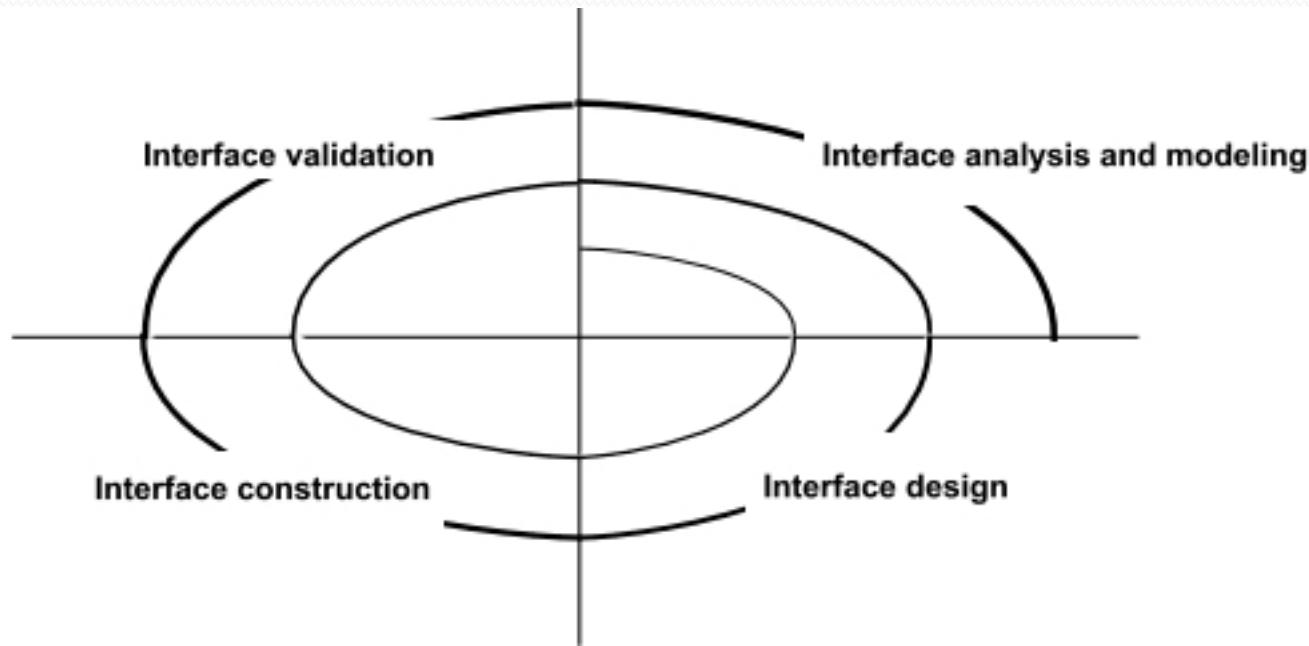
# Make the Interface Consistent

- Allow the user to put the current task into a meaningful context.
- Maintain consistency across a family of applications.
- If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.

# User Interface Design Models

- **User model** — a profile of all end users of the system
- **Design model** — a design realization of the user model
- **Mental model (system perception)** — the user's mental image of what the interface is
- **Implementation model** — the interface “look and feel” coupled with supporting information that describe interface syntax and semantics

# User Interface Design Process



# Interface Analysis

- Interface analysis means understanding
  - (1) the people (end-users) who will interact with the system through the interface;
  - (2) the tasks that end-users must perform to do their work,
  - (3) the content that is presented as part of the interface
  - (4) the environment in which these tasks will be conducted.

# User Analysis

- Are users trained professionals, technician, clerical, or manufacturing workers?
- What level of formal education does the average user have?
- Are the users capable of learning from written materials or have they expressed a desire for classroom training?
- Are users expert typists or keyboard phobic?
- What is the age range of the user community?
- Will the users be represented predominately by one gender?

# User Analysis

- How are users compensated for the work they perform?
- Do users work normal office hours or do they work until the job is done?
- Is the software to be an integral part of the work users do or will it be used only occasionally?
- What is the primary spoken language among users?
- What are the consequences if a user makes a mistake using the system?
- Are users experts in the subject matter that is addressed by the system?
- Do users want to know about the technology the sits behind the interface?

# Task Analysis and Modeling

- Answers the following questions ...
  - What work will the user perform in specific circumstances?
  - What tasks and subtasks will be performed as the user does the work?
  - What specific problem domain objects will the user manipulate as work is performed?
  - What is the sequence of work tasks—the workflow?
  - What is the hierarchy of tasks?

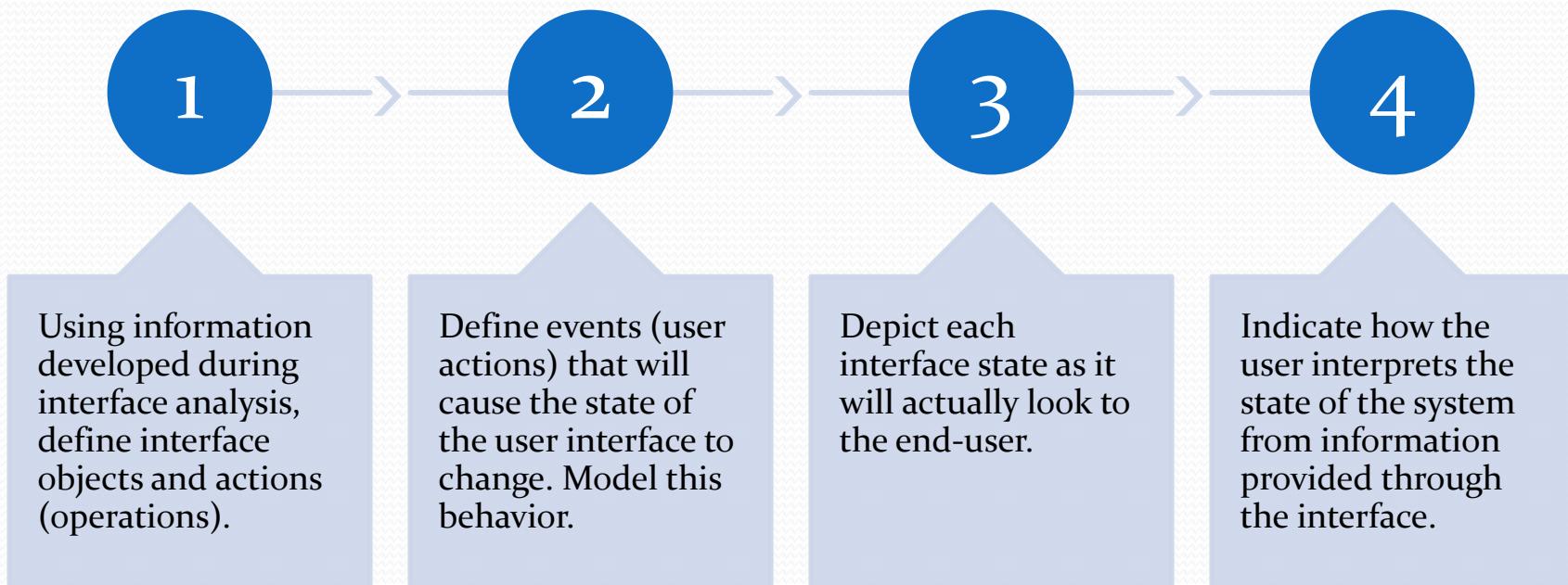
# Task Analysis and Modeling

- Use-cases define basic interaction
- Task elaboration refines interactive tasks
- Object elaboration identifies interface objects (classes)
- Workflow analysis defines how a work process is completed when several people (and roles) are involved

# Analysis of Display Content

- Are different types of data assigned to consistent geographic locations on the screen (e.g., photos always appear in the upper right hand corner)?
- Can the user customize the screen location for content?
- Is proper on-screen identification assigned to all content?
- If a large report is to be presented, how should it be partitioned for ease of understanding?
- Will mechanisms be available for moving directly to summary information for large collections of data.
- Will graphical output be scaled to fit within the bounds of the display device that is used?
- How will color be used to enhance understanding?
- How will error messages and warning be presented to the user?

# Interface Design Steps



# Design Issues

- Response time
- Help facilities
- Error handling
- Menu and command labeling
- Application accessibility
- Internationalization (I18n)

# Web and Mobile App Interface Design

- *Where am I?* The interface should
  - Provide an indication of the Web or Mobile App that has been accessed
  - Inform the user of her location in the content hierarchy.
- *What can I do now?* The interface should always help the user understand his current options
  - What functions are available?
  - What links are live?
  - What content is relevant?
- *Where have I been, where am I going?* The interface must facilitate navigation.
  - Provide a “map” (implemented in a way that is easy to understand) of where the user has been and what paths may be taken to move elsewhere within the Web or Mobile App.

# Interface Design Principles-I

- ***Anticipation***—A Web or Mobile App should be designed so that it anticipates/predict the user's next move.
- ***Communication***—The interface should communicate the status of any activity initiated by the user
- ***Consistency***—The use of navigation controls, menus, icons, and aesthetics (e.g., color, shape, layout)
- ***Controlled autonomy***—The interface should facilitate user movement throughout the Web or Mobile App, but it should do so in a manner that enforces navigation conventions that have been established for the application.
- ***Efficiency***—The design of the Web or Mobile App and its interface should optimize the user's work efficiency, not the efficiency of the software engineer who designs and builds it or the client-server environment that executes it.

# Interface Design Principles-II

- *Focus*—The Web or Mobile App interface (and the content it presents) should stay focused on the user task(s) at hand.
- *Fitt's Law*—“The time to acquire a target is a function of the distance to and size of the target.”
- *Human interface objects*—A vast library of reusable human interface objects has been developed for Web or Mobile Apps.
- *Latency reduction*—The Web or Mobile App should use multi-tasking in a way that lets the user proceed with work as if the operation has been completed.
- *Learnability*— A Web or Mobile App interface should be designed to minimize learning time, and once learned, to minimize relearning required when the App is revisited.

# Interface Design Principles-III

- ***Maintain work product integrity***—A work product (e.g., a form completed by the user, a user specified list) must be automatically saved so that it will not be lost if an error occurs.
- ***Readability***—All information presented through the interface should be readable by young and old.
- ***Track state***—When appropriate, the state of the user interaction should be tracked and stored so that a user can logoff and return later to pick up where she left off.
- ***Visible navigation***—A well-designed Web or Mobile App interface provides “the illusion that users are in the same place, with the work brought to them.”

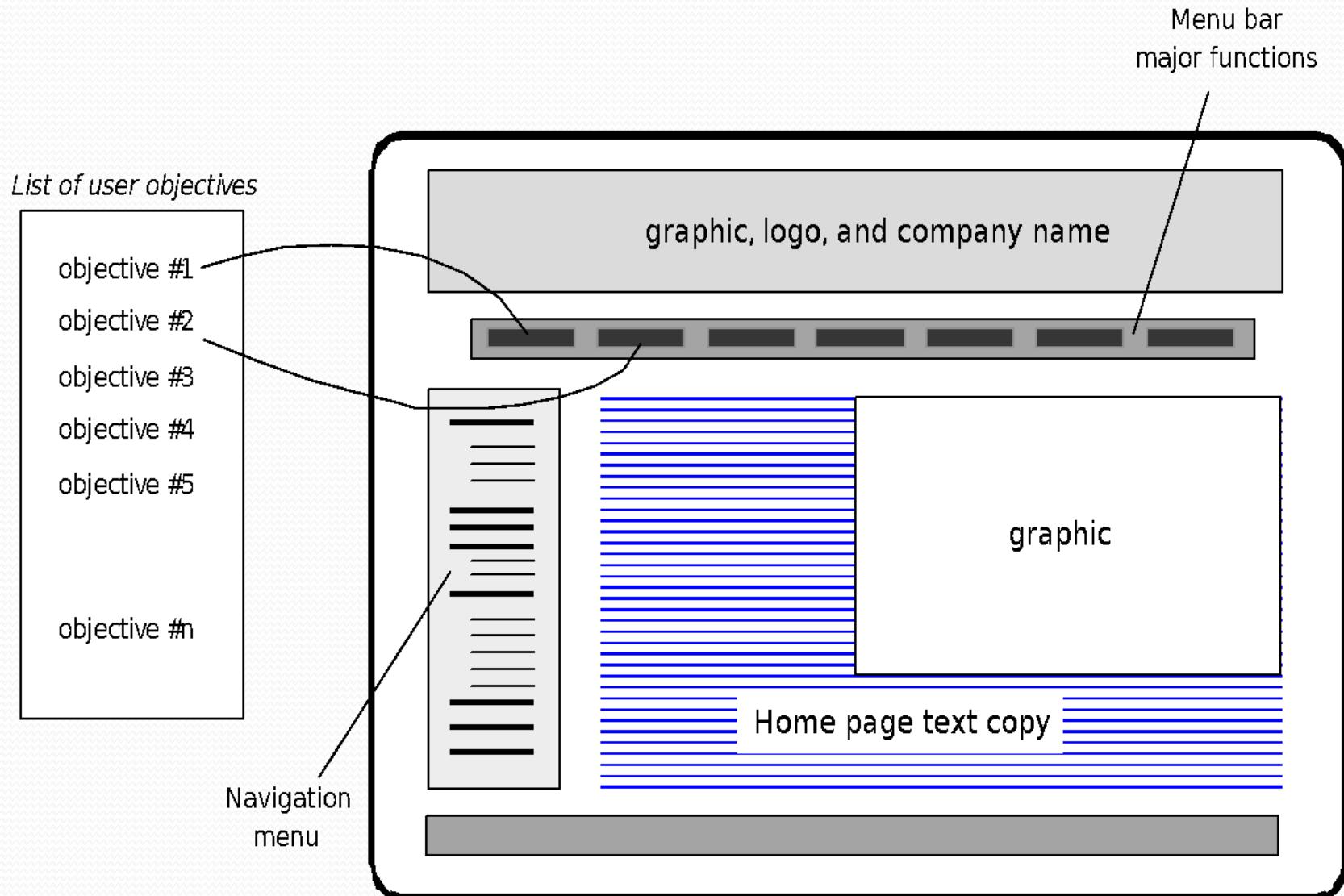
# Interface Design Workflow-I

- Review information contained in the analysis model and refine as required.
- Develop a rough sketch of the Web or Mobile App interface layout.
- Map user objectives into specific interface actions.
- Define a set of user tasks that are associated with each action.
- Storyboard screen images for each interface action.
- Refine interface layout and storyboards using input from aesthetic design.

# Interface Design Workflow-I

- Review information contained in the analysis model and refine as required.
- Develop a rough sketch of the Web or Mobile App interface layout.
- Map user objectives into specific interface actions.
- Define a set of user tasks that are associated with each action.
- Storyboard screen images for each interface action.
- Refine interface layout and storyboards using input from aesthetic design.

# Mapping User Objectives



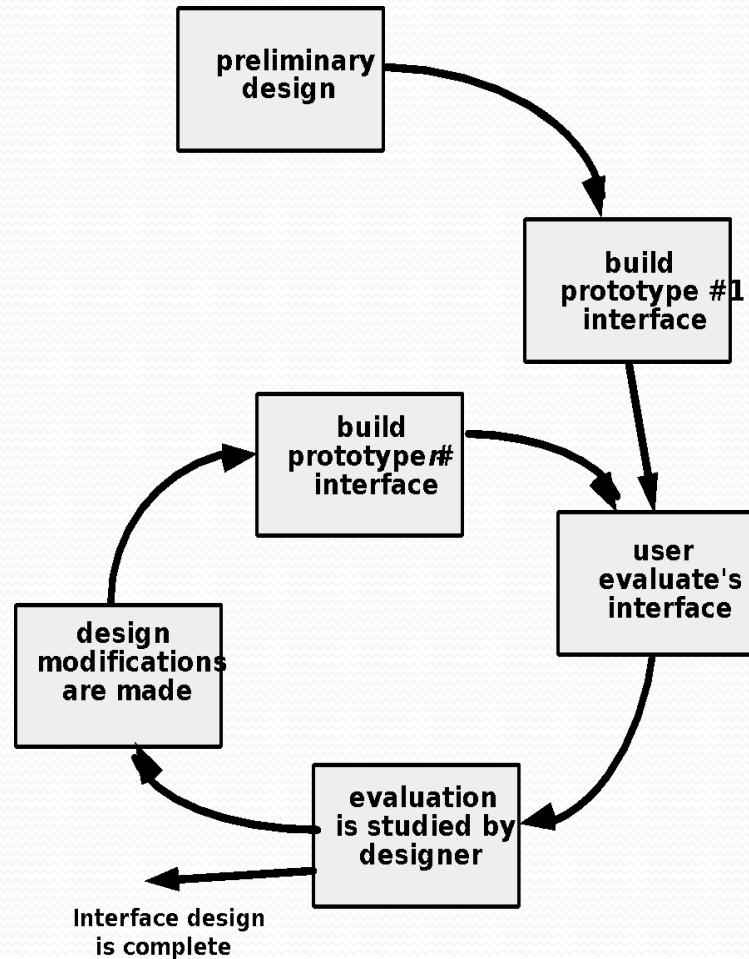
# Interface Design Workflow-II

- Identify user interface objects that are required to implement the interface.
- Develop a procedural representation of the user's interaction with the interface.
- Develop a behavioral representation of the interface.
- Describe the interface layout for each state.
- Refine and review the interface design model.

# Aesthetic Design

- Don't be afraid of white space.
- Emphasize content.
- Organize layout elements from top-left to bottom right.
- Group navigation, content, and function geographically within the page.
- Don't extend your real estate with the scrolling bar.
- Consider resolution and browser window size when designing layout.

# Design Evaluation Cycle



# Play game

<https://cantunsee.space/>

