Software Design: Architecture, Component-Level and Interface

Software Engineering I

AY 2024/25

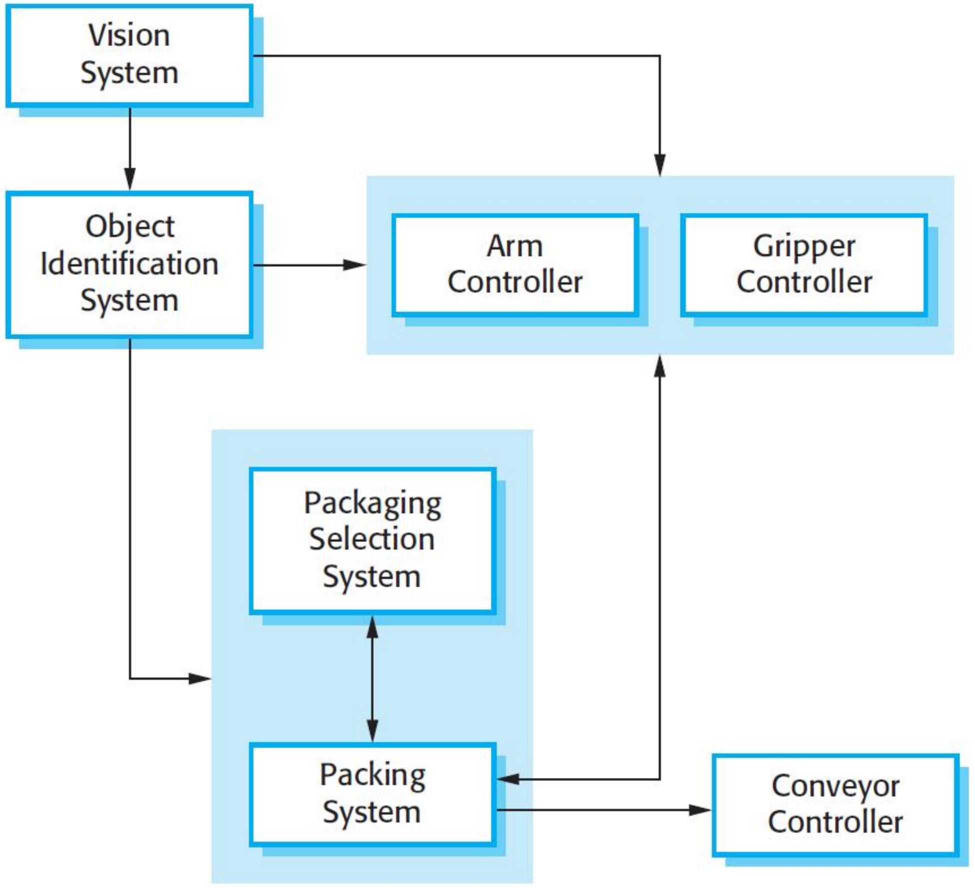
Week 9

OUTLINE

- 1. Architecture Design
- 2. Component-level Design
- 3. User Interface Design

1.1a Architecture Design - The Basics

- Architectural design is concerned with understanding how a system should be organized and designing the overall structure of that system.
- It is the first stage in the software design process, and the critical link between design and requirements engineering.
- It identifies the main structural components in a system and the relationships between them.
- The output of this design process is a description of the software architecture.



1.1b Architecture Design - The Basics

- Design software architectures at **two levels of abstraction:**

- **Architecture in the small** is concerned with the architecture of individual programs. At this level, we are concerned with the way that an individual program is decomposed into components. This chapter is mostly concerned with program architectures.

- **Architecture in the large** is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components. These enterprise systems are distributed over different computers, which may be owned and managed by different companies.

1.2a Architecture Design - Why it Matters

- Software architecture affects performance, robustness, distributability, and maintainability
- Individual component – implement functional requirements
- System architecture – fulfill non-functional requirement

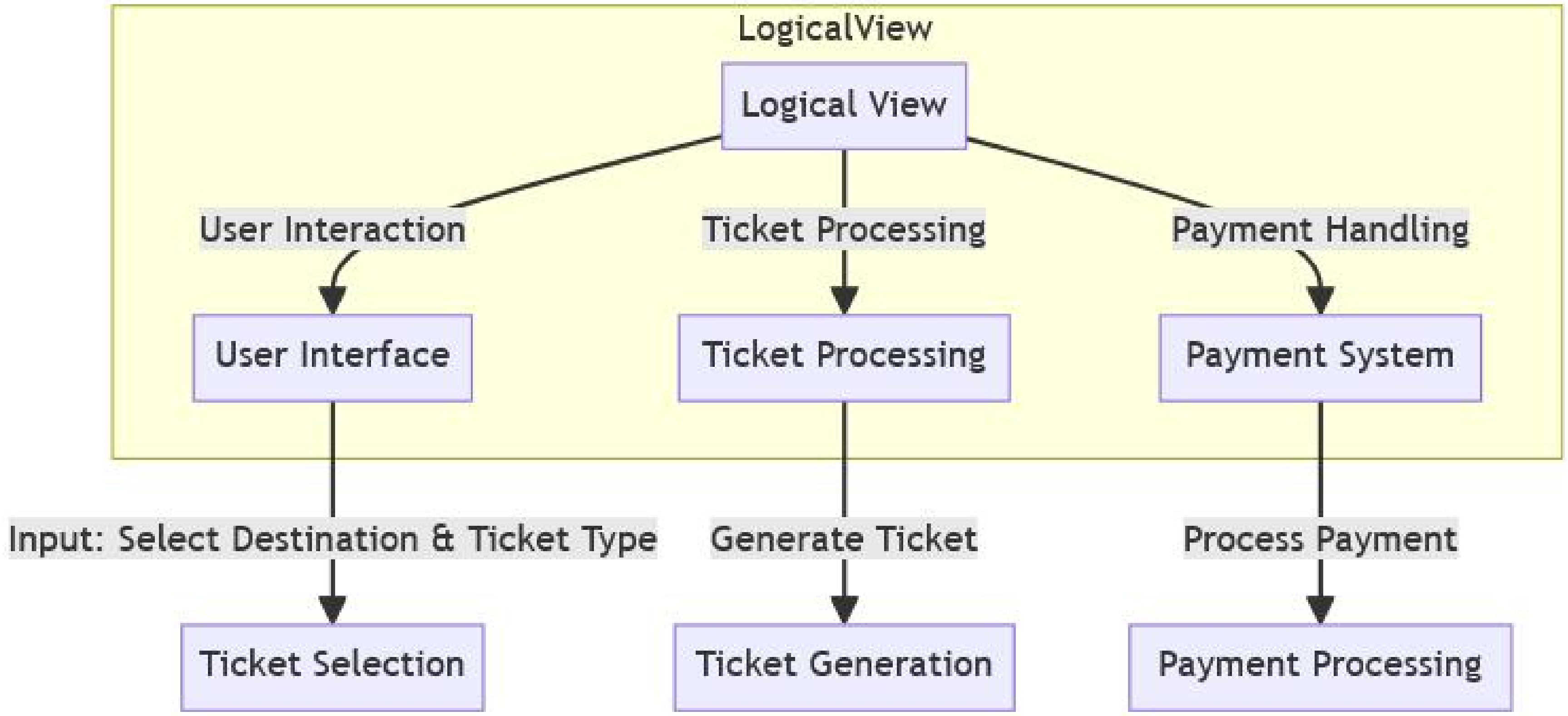
1.2b Architecture Design - Why it Matters

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse
  - The architecture may be reusable across a range of systems
  - Product-line architectures may be developed.

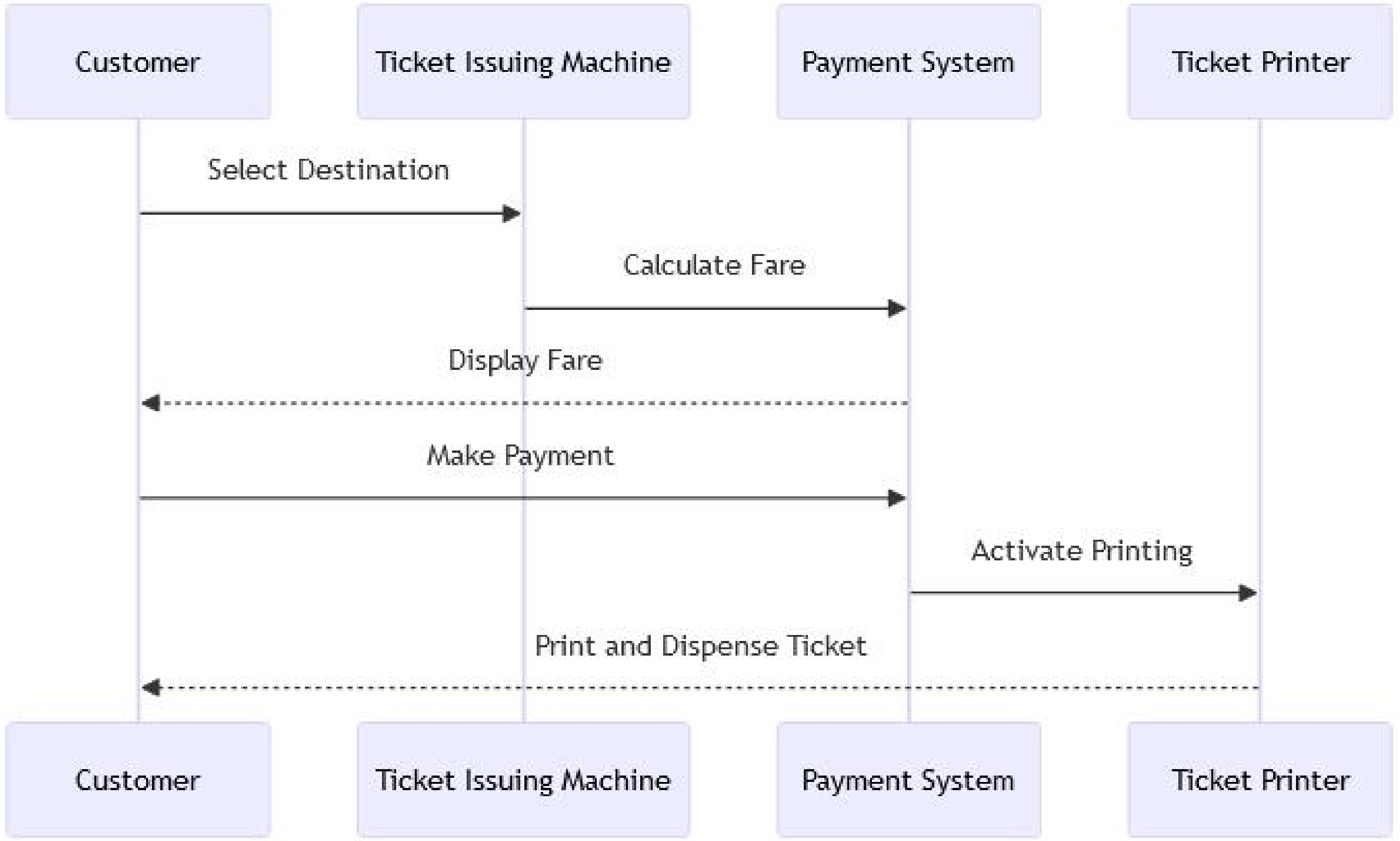
1.3 Architecture Design - Different Views

- A **logical view**, which shows the key abstractions in the system as objects or object classes.
- A **process view,** which shows how, at run-time, the system is composed of interacting processes.
- A **development view**, which shows how the software is decomposed for development.
- A **physical view**, which shows the system hardware and how software components are distributed across the processors in the system.

1.3.1 Logical View - An automated ticket-issuing system used by passengers at a railway station

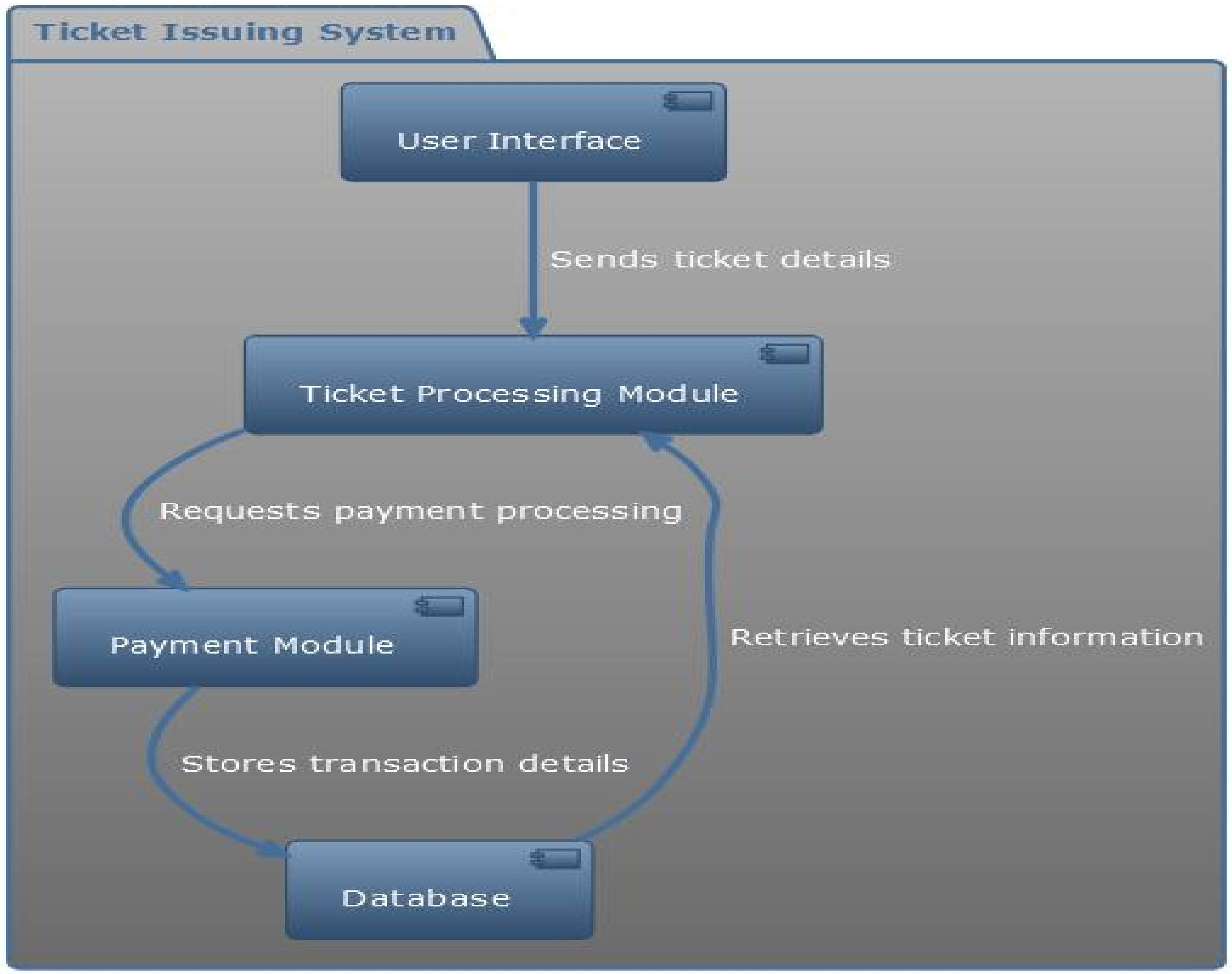


1.3.2 Process View - An automated ticket-issuing system used by passengers at a railway station



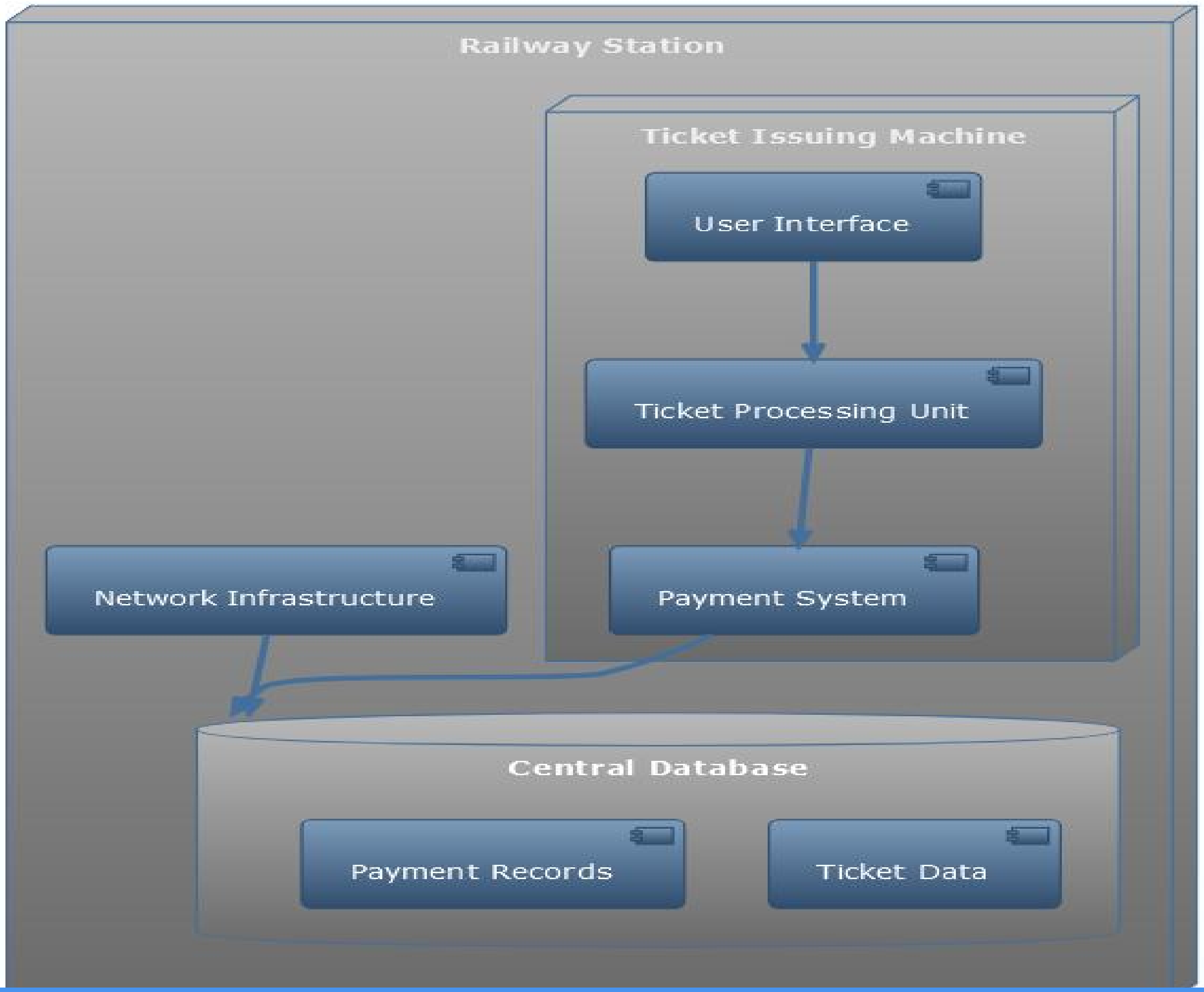
代码(类相关)

1.3.3 Development View - An automated ticket-issuing system used by passengers at a railway station



物理机器

1.3.4 Physical View - An automated ticket-issuing system used by passengers at a railway station



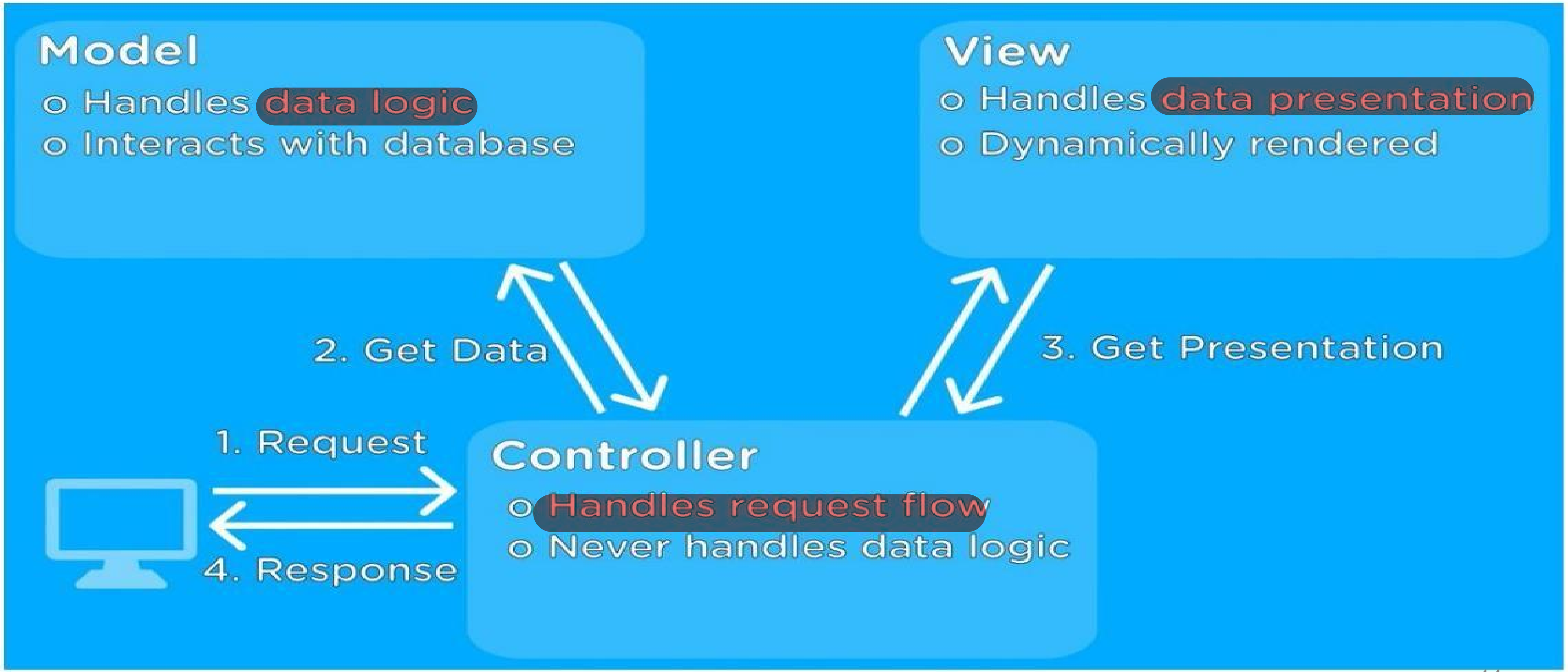
1.4 Architecture Design - Achitecture Patterns 架构模式

- Patterns are means of representing, sharing and reusing knowledge.
- An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments.
- Patterns should include information about when they are and when the are not useful.
- Patterns may be represented using tabular and graphical descriptions.

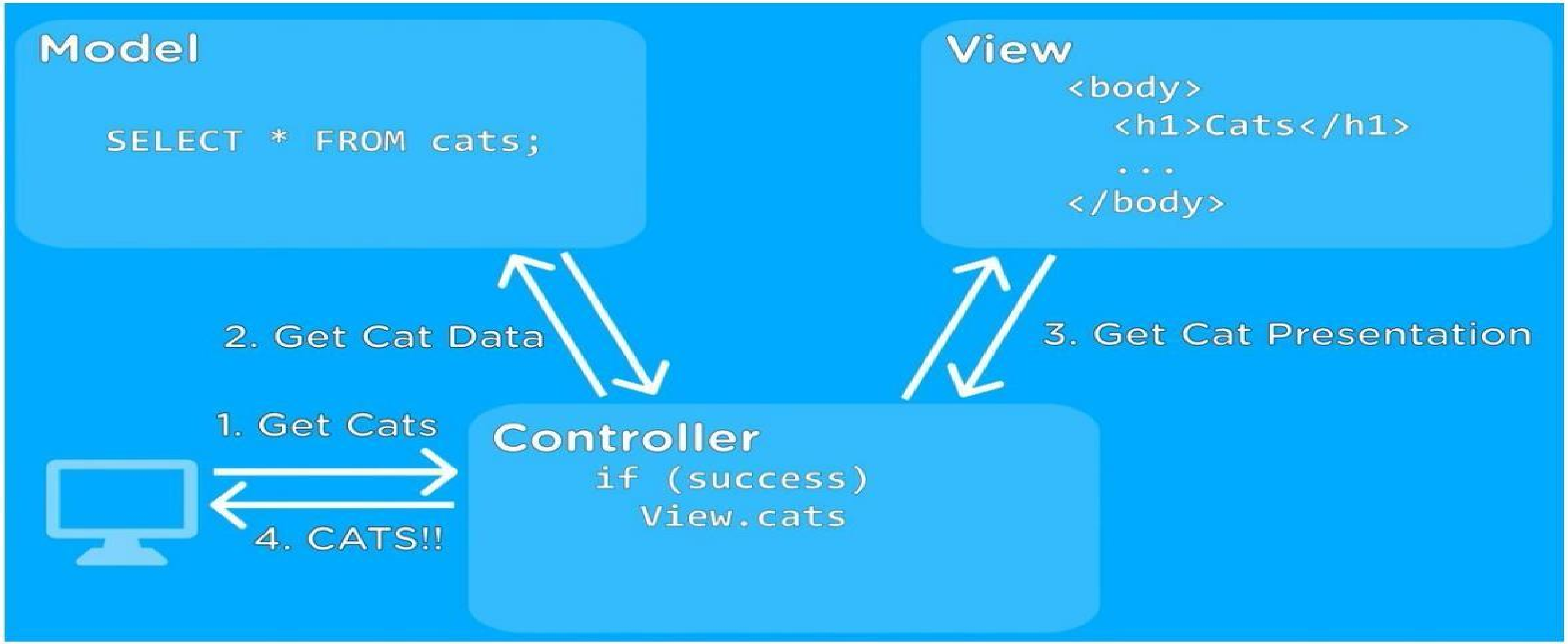
1.4.1a The MVC Pattern

- Model-View-Controller (MVC pattern)
- This pattern is the basis of interaction management in many web-based systems.
- It includes three major interconnected components:
  - **Model:** central component of the pattern that directly manages the data, logic and rules of the application
  - **View:** can be any output representation of information, such as a chart or a diagram.
  - **Controller:** accepts input and converts it to commands for the model or view, enables the interconnection between the views and the model

1.4.1b The MVC Pattern



1.4.1c The MVC Pattern



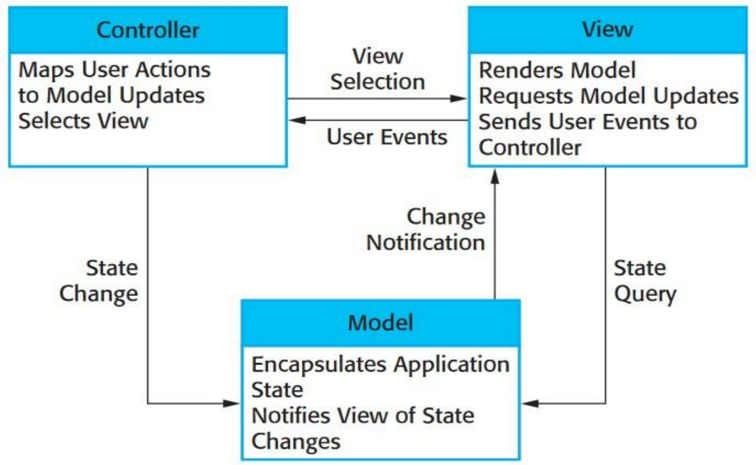
```
SELECT * FROM cats;
```

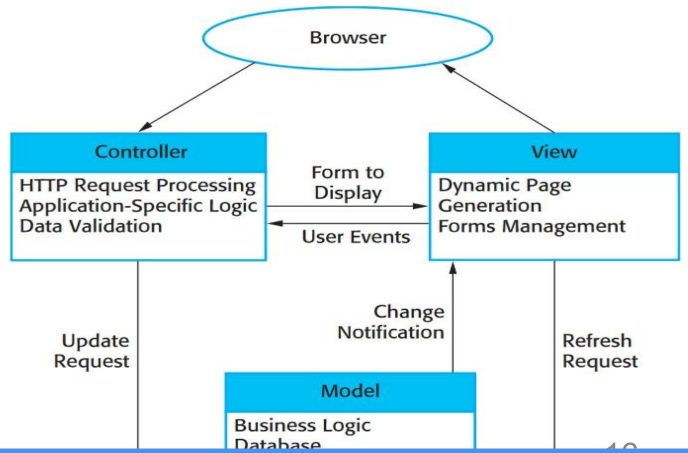
```
<body>
  <h1>Cats</h1>
  ...
</body>
```

```
if (success)
  View.cats
```

1.4.1d The MVC Pattern

Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model. See Figure 6.3.
Example	Figure 6.4 shows the architecture of a web-based application system organized using the MVC pattern.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.





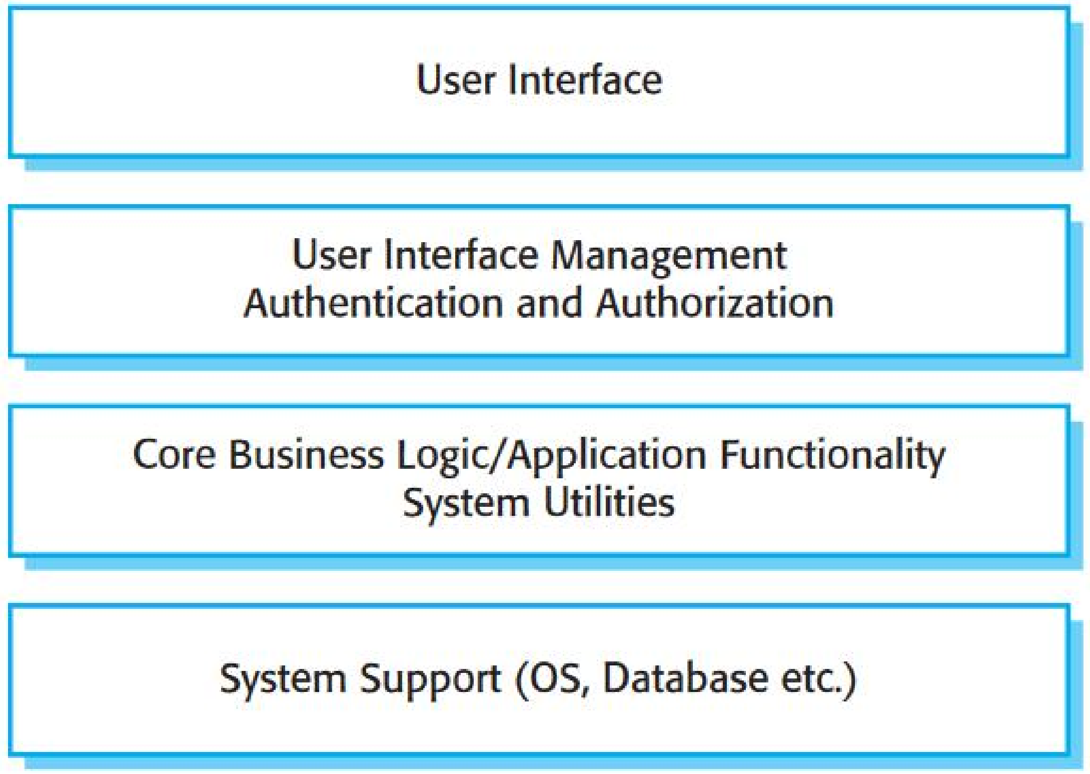
1.4.2a The Layered Pattern 分层模式

- Layered Pattern: the system functionality is organized into separate layers, and each layer only relies on the facilities and services offered by the layer immediately beneath it.

- This layered approach supports the incremental development of systems. As a layer is developed, some of the services provided by that layer may be made available to users.

- Performs poorly in the high-performance applications, because it is not efficient to go through multiple layers to fulfil a business request. It is a good choice for situations with a very tight budget and time constraints.

1.4.2b The Layered Pattern


User Interface
User Interface Management
Authentication and Authorization
Core Business Logic/Application Functionality
System Utilities
System Support (OS, Database etc.)

Presentation Layer

$\downarrow$

Application Layer

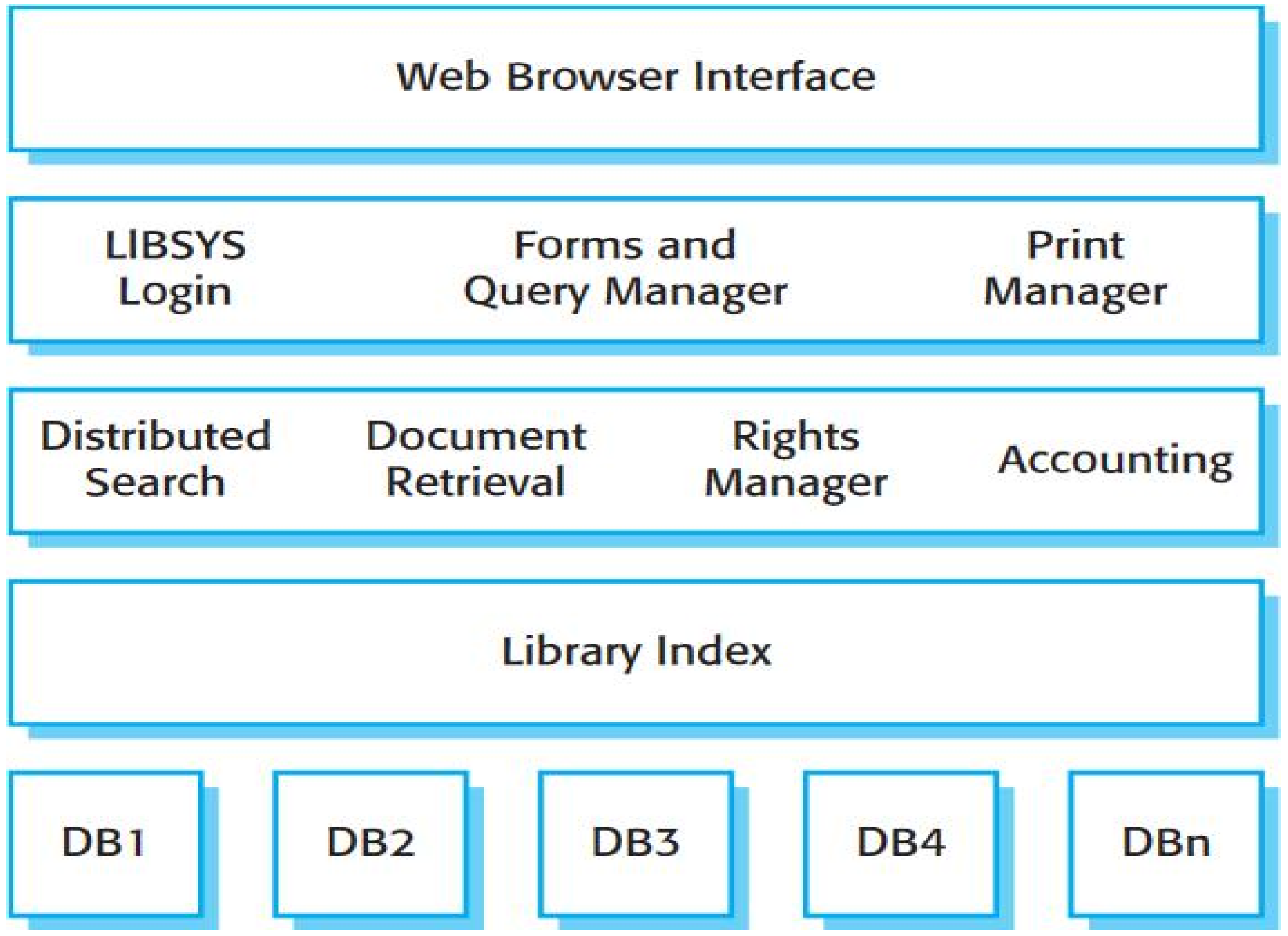
$\downarrow$

Business Logic Layer

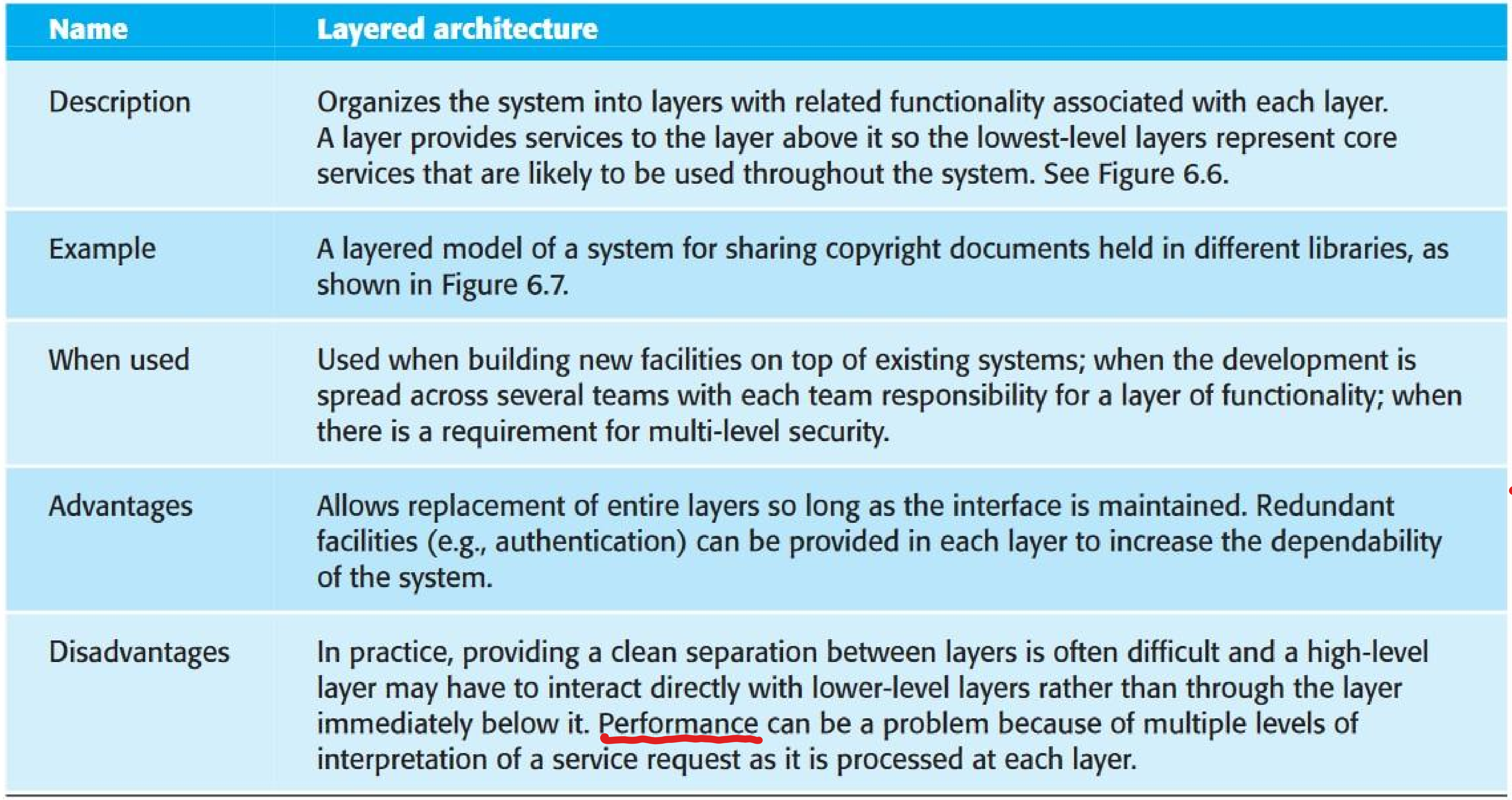
$\downarrow$

Data Access Layer

1.4.2b The Layered Pattern



1.4.2c The Layered Pattern

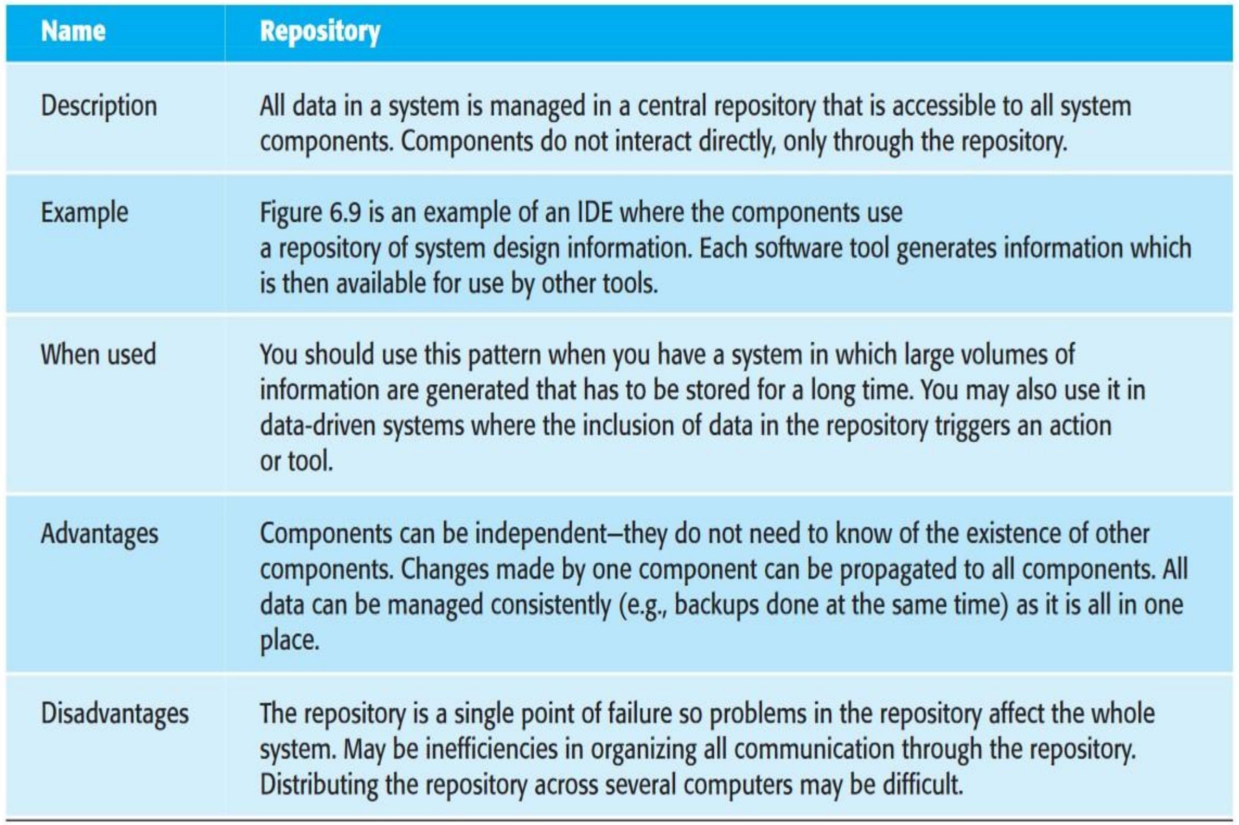


Name	Layered architecture
Description	Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. See Figure 6.6.
Example	A layered model of a system for sharing copyright documents held in different libraries, as shown in Figure 6.7.
When used	Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security.
Advantages	Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

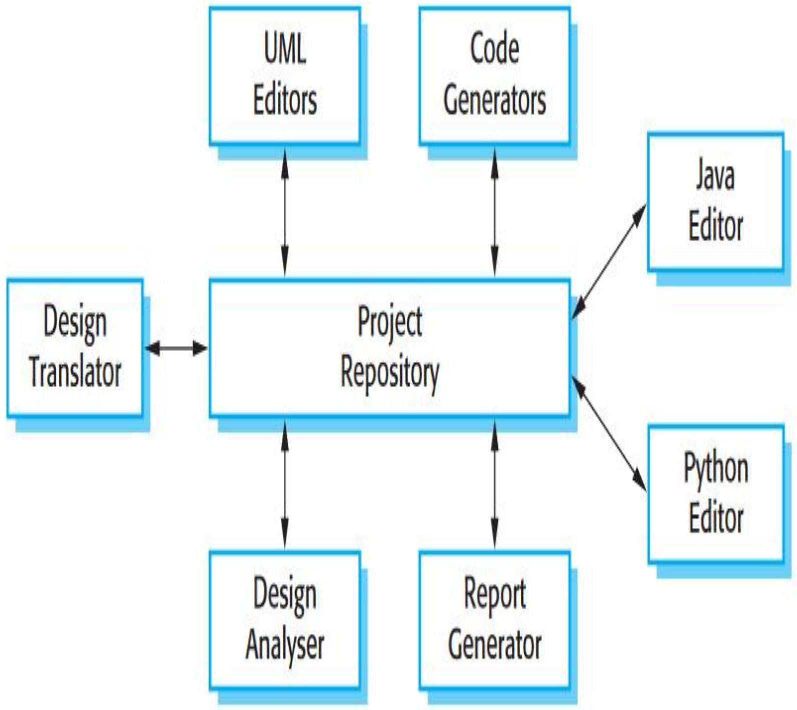
只要接口维持
整层可替代的
扩展性,可靠性

层级多 → 请求
多 → 低性能

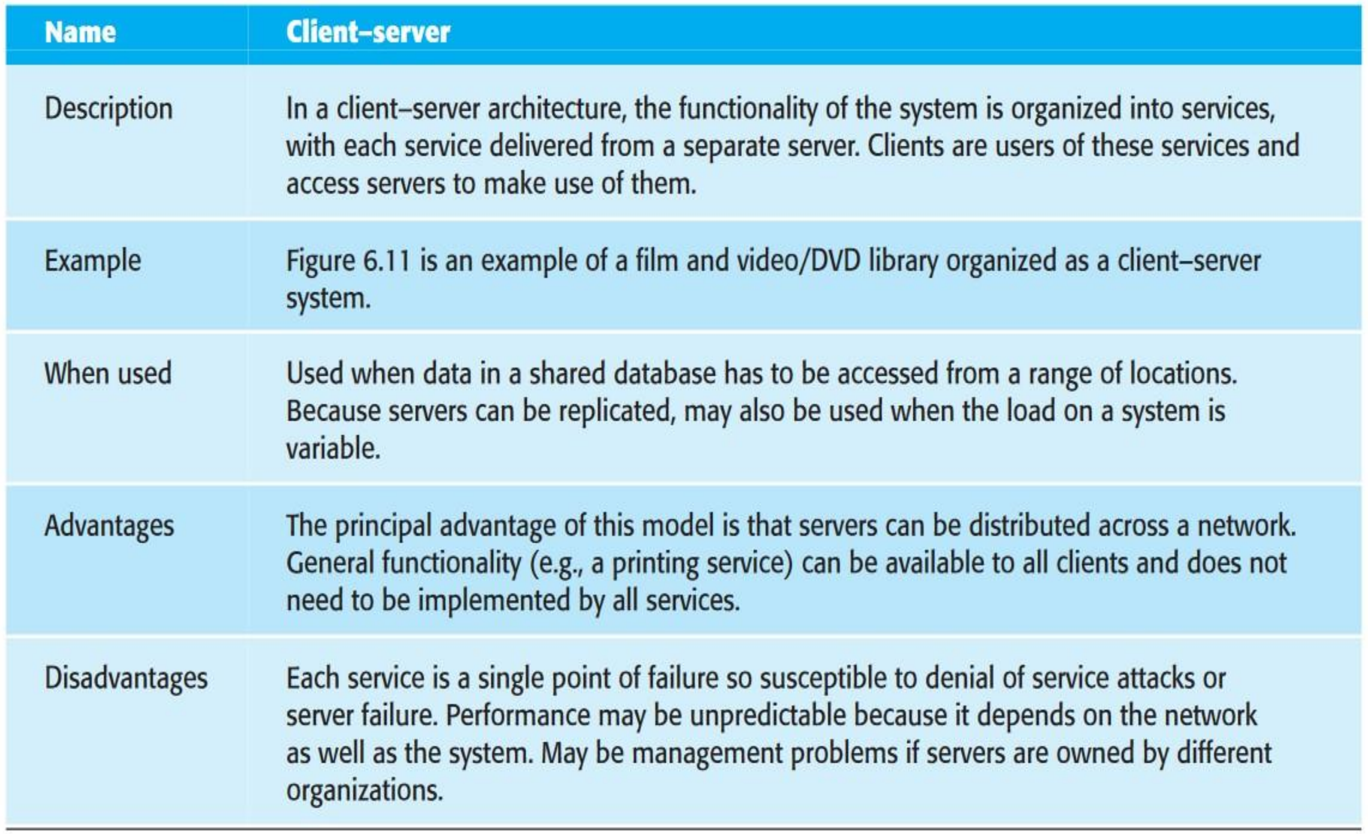
1.4.3 The Repository Pattern 仓库模式



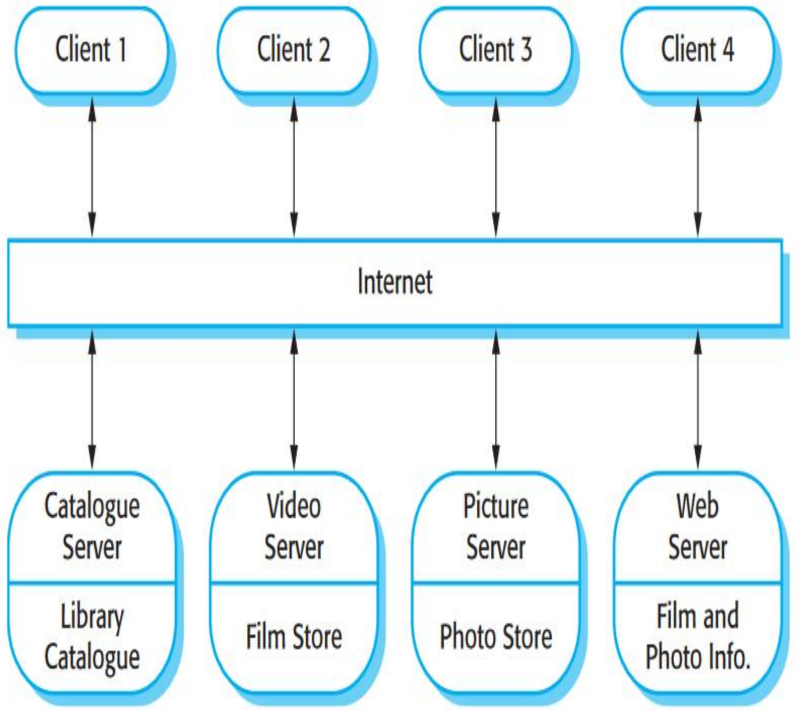
Name	Repository
Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
Example	Figure 6.9 is an example of an IDE where the components use a repository of system design information. Each software tool generates information which is then available for use by other tools.
When used	You should use this pattern when you have a system in which large volumes of information are generated that has to be stored for a long time. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	Components can be independent—they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.
Disadvantages	The repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult.



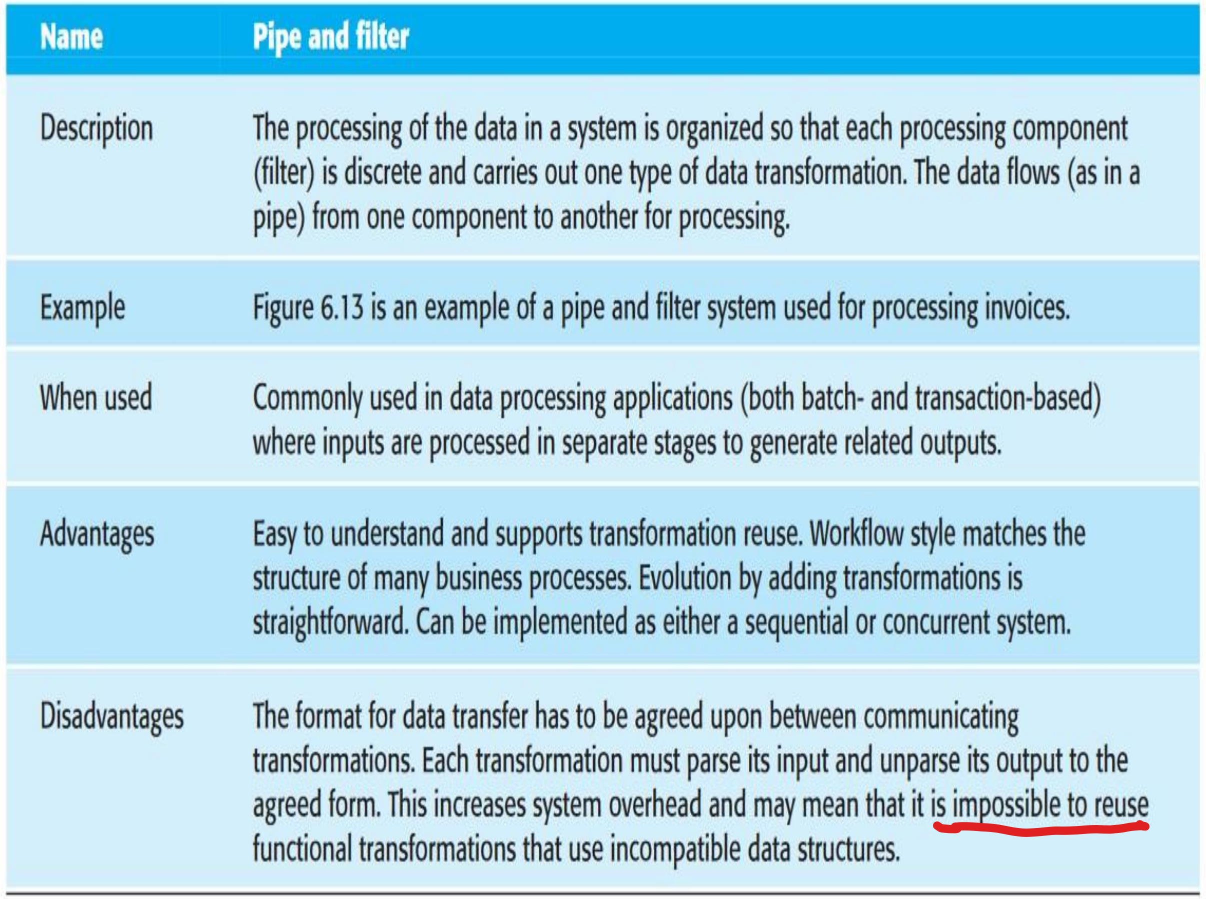
1.4.4 The Client-server Pattern



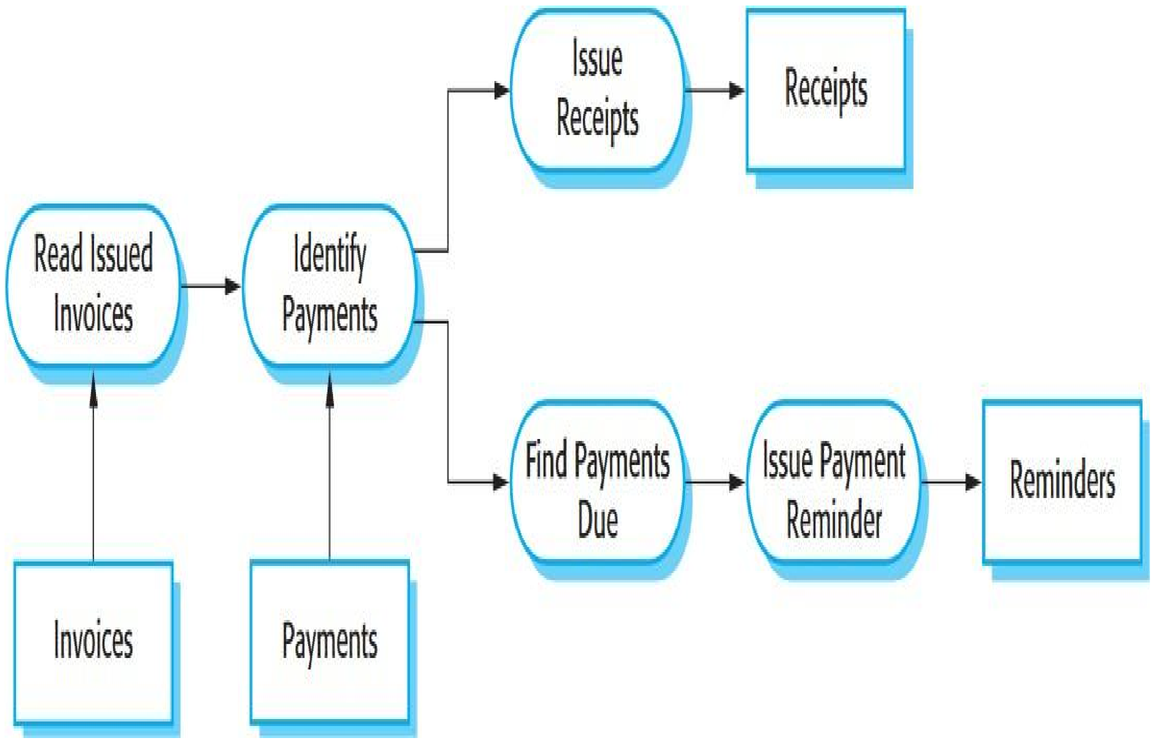
Name	Client–server
Description	In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.
Example	Figure 6.11 is an example of a film and video/DVD library organized as a client–server system.
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations.



1.4.5 The Pipe and Filter Pattern



Name	Pipe and filter
Description	The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
Example	Figure 6.13 is an example of a pipe and filter system used for processing invoices.
When used	Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.
Advantages	Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Can be implemented as either a sequential or concurrent system.
Disadvantages	The format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures.



2.1a Component Level Design - The Basics

- **Component-level design** occurs after the first iteration of architectural design has been completed.
- A complete set of software components is defined during architectural design.
- Component-level design defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each software component.
- A component-level design can be represented using some intermediate representations (e.g. graphical, tabular, or text-based) that can be translated into source code.

2.1b Component Level Design - The Basics

- **A software component** is a modular building block for computer software.
- It can be used to review for correctness and consistency with other components.
- It can be used to access whether data structure, interfaces and algorithms will work
- It should provide sufficient information to guide implementation.

2.2 Component Level Design - Different View

- ➢ Three different view of a component
  - ➢ An object-oriented view
  - ➢ A conventional view
  - ➢ A process view

2.2.1 Object-Oriented View

- **Focus:** Centers on objects as the fundamental building blocks of software. An object encapsulates both data and behavior, representing entities or concepts.
- **Key Concepts:** Includes encapsulation, inheritance, polymorphism, and abstraction. Classes define the structure and behavior of objects.
- **Advantages:** Promotes reusability, modularity, and maintainability. It's particularly effective for complex systems where real-world modeling is beneficial.
- **Usage:** Widely adopted in modern software development, especially for applications requiring extensive data manipulation and complex interactions.

X

Example: Consider you're working on a component named **"Customer Profile Management"**

Object-Oriented View - Classes & Objects:

- <u>CustomerProfile:</u> Represents a customer's profile.
  - Attributes include customerID, name, contactDetails, purchaseHistory.
  - Methods might include updateContactDetails(), addPurchaseRecord().
- <u>ProfileManager:</u> Handles operations related to customer profiles.
  - Attributes include ManagerID, name, contactDetails, Position.
  - Methods include createProfile(CustomerProfile), updateProfile(customerID, CustomerProfile), getProfile(customerID)

2.2.2 Traditional View

- Traditional View (Structured or Procedural View):
  - **Focus:** Emphasizes a top-down approach to software design, focusing on functions or procedures and the flow of data between them.
  - **Key Concepts:** Software is structured into functions or procedures that perform specific tasks. It relies on sequence, selection, and iteration in programming constructs.
  - **Advantages:** Simplicity and straightforwardness, particularly effective for linear and less complex applications. It's easier to follow for small-scale projects.
  - **Usage:** More prevalent in earlier stages of software development history, suitable for applications with a clear sequence of operations like batch processing.

Example: Consider you're working on a component named **"Customer Profile Management"**

Traditional View - Procedures/Functions:

createCustomerProfile(customerID, name, contactDetails): Creates a new customer profile.
updateCustomerProfile(customerID, updatedDetails): Updates an existing customer profile.
retrieveCustomerProfile(customerID): Retrieves the details of a specific customer profile.

2.2.3 Process-related View

- **Focus:** Concerned with the runtime behavior of software components. It looks at how components operate during execution, particularly in terms of processes and threads.
- **Key Concepts:** Includes process management, inter-process communication, concurrency, synchronization, and resource management.
- **Advantages:** Essential for understanding system performance, scalability, and reliability. Critical for systems where real-time processing, concurrency, and resource management are key concerns.
- **Usage:** Relevant for complex, distributed, or real-time systems where understanding the dynamic behavior is crucial for system performance and reliability.

Example: Consider you're working on a component named **"Customer Profile Management"**

Runtime Processes:

- Initialization Process: Loads the product catalog and user data when the system starts.
- Order Processing Workflow: Involves validating the shopping cart, calculating the total cost, and initiating the payment process.
- Payment Processing: Handles authorization, validation of payment details, and confirmation of payment. Must manage concurrency for simultaneous transactions.
- Notification and Logging: Sending order confirmation to the customer and updating the system logs for each transaction.

2.2.4 Summary of the three views

- **The object-oriented view** focuses on modeling the system as a set of interacting objects, each encapsulating data and behavior relevant to online shopping.
- **The traditional view** structures the system as a series of procedures or functions that perform specific tasks like adding products, placing orders, and processing payments.
- **The process view** looks at the system in terms of its runtime behavior, particularly how it handles the flow of orders and payments, and manages concurrency and system resources.

3.1 Interface Design - The Basics 界面设计

- User interface design creates an effective communication medium between a human and a computer.
- Why is it important?
  - A poorly designed user interface will force user to commit mistakes.
  - Users can get easily frustrated 沮丧 using a poorly designed interface regardless of computational power or content.

3.2 Interface Design - The Golden Rules

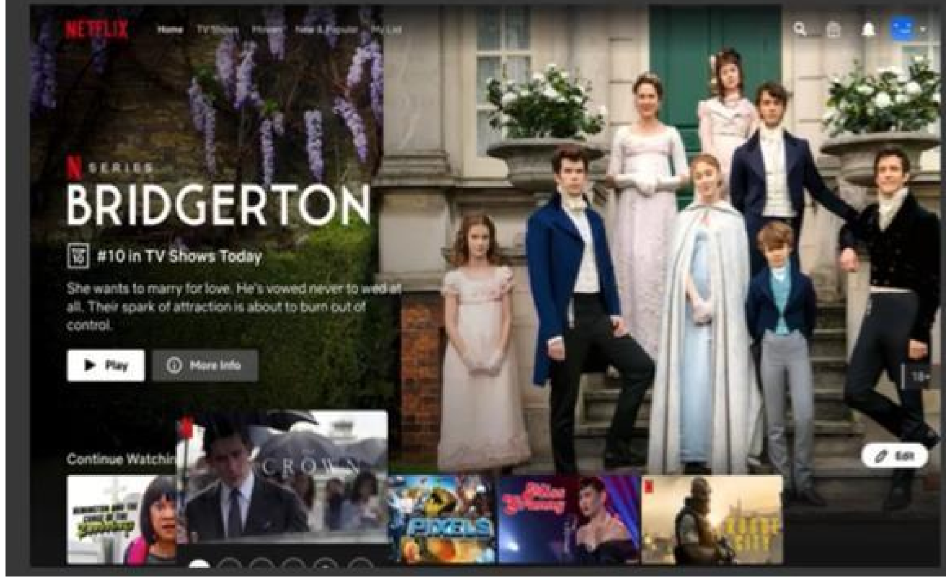
- The golden rules
  - **Place the user in control**
  - **Reduce the user's memory**
  - **Make the interface consistent**

flexible interaction:
be interruptible/undoable
show system status

3.2.1a Place the User in Control

➢ Define interaction modes in a way that does not force a user into unnecessary or undesired actions

Netflix hover auto-play



She wants to marry for love. He's vowed never to wed at all. Their spark of attraction is about to burn out of control.

Auto-anything means the designers have made huge assumptions about the users' desires.

Users are unnecessarily forced to watch a preview of a movie after a split-second hover over a movie or tv show thumbnail when all they want to do is read the details.

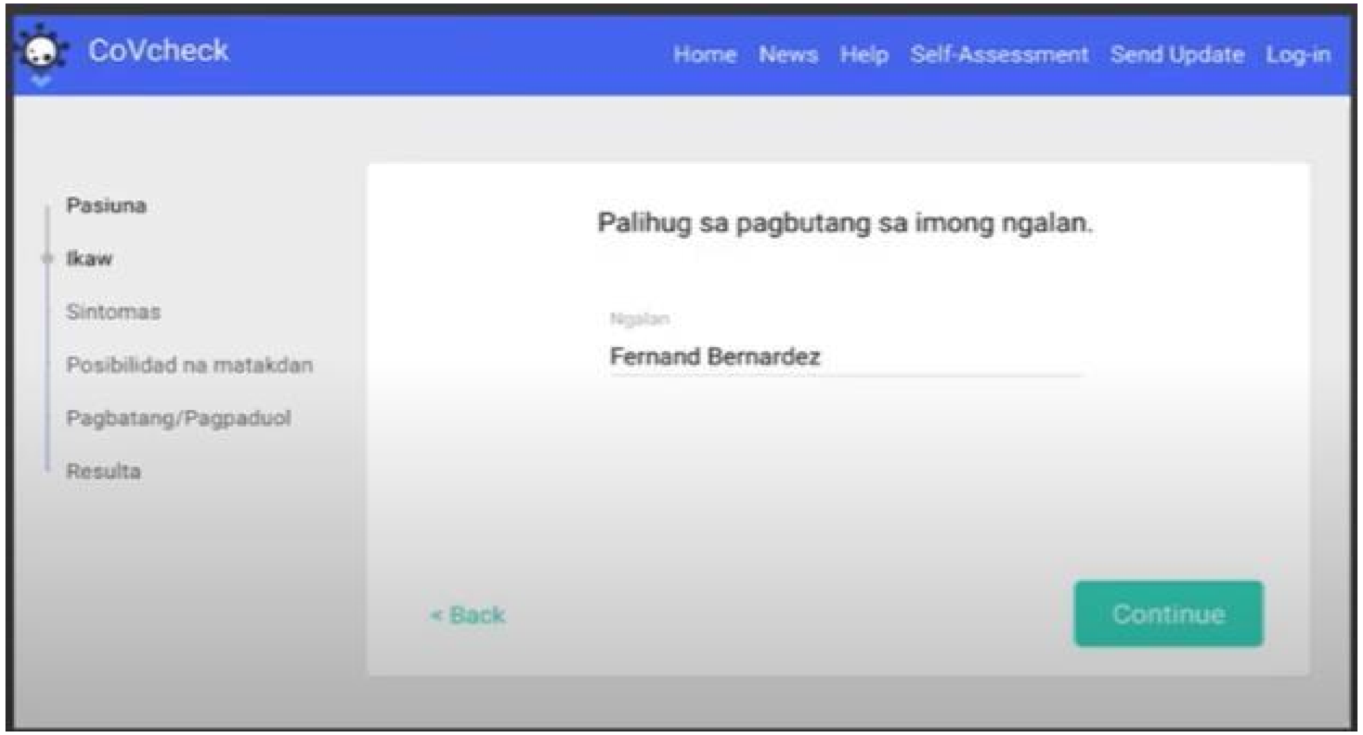
3.2.1b Place the User in Control

- Provide for flexible interaction. Because different users have different interaction preferences, choices should be provided.

3.2.1c Place the User in Control

中断 撤消

- Allow user interaction to be interruptible and undoable. Interrupt sequence of steps without losing work that had been done



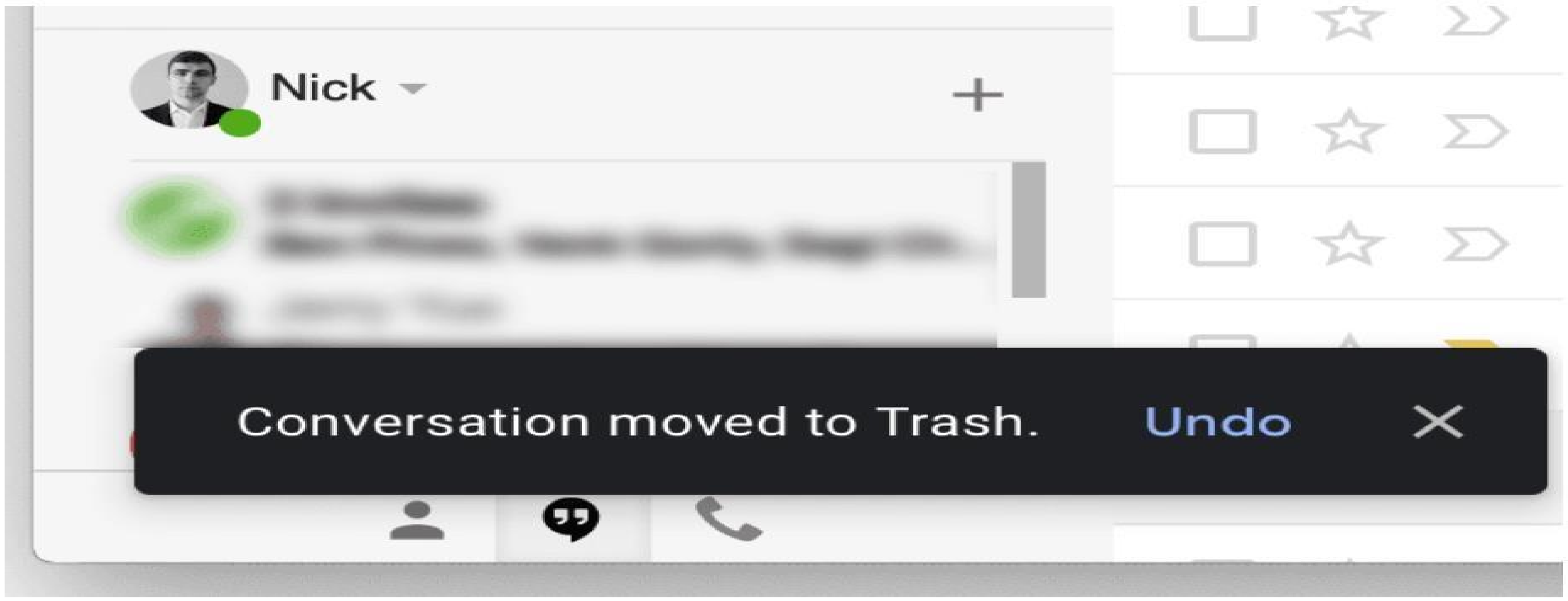
Palihug sa pagbutang sa imong ngalan.

Ngalan
Fernand Bernardez

3.2.1d Place the User in Control

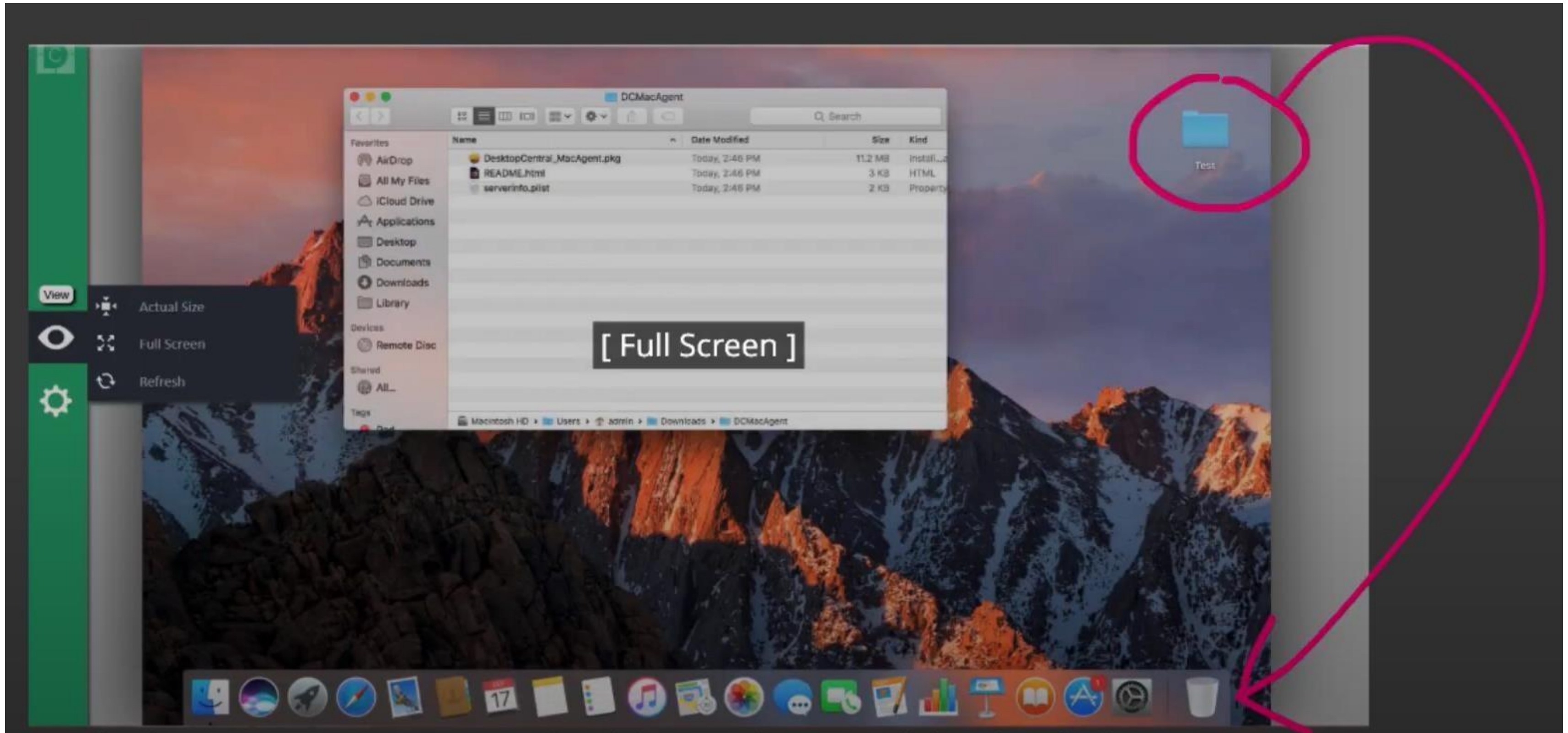
➢ Allow user interaction to be interruptible and undoable.

- Make actions reversible – be forgiving- Undo
- ‘Undo’ can be extremely helpful when users choose system function by mistake. In this case, the undo function serves as an ’emergency exit,’ allowing users to leave the unwanted state.



3.2.1e Place the User in Control

- Design for direct interaction with object that appear on the screen.

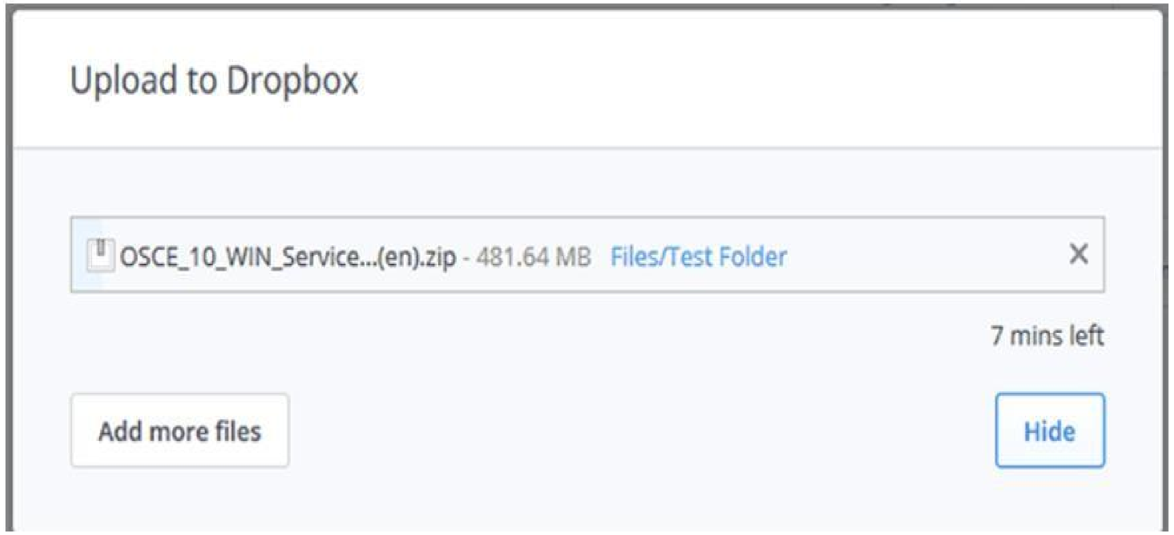


Name	Date Modified	Size	Kind
DesktopCentral_MacAgent.pkg	Today, 2:46 PM	11.2 MB	Install...
README.html	Today, 2:46 PM	3 KB	HTML
serverinfo.plist	Today, 2:46 PM	2 KB	Property

Image from https://www.manageengine.com/eu/products/desktop-central/mac-remote-desktop-sharing.html

3.2.1f Place the User in Control

- Show the visibility of system status.
- Users are much more forgiving when they have information about what is going on and are given periodic feedback about the status of the process.


Upload to Dropbox

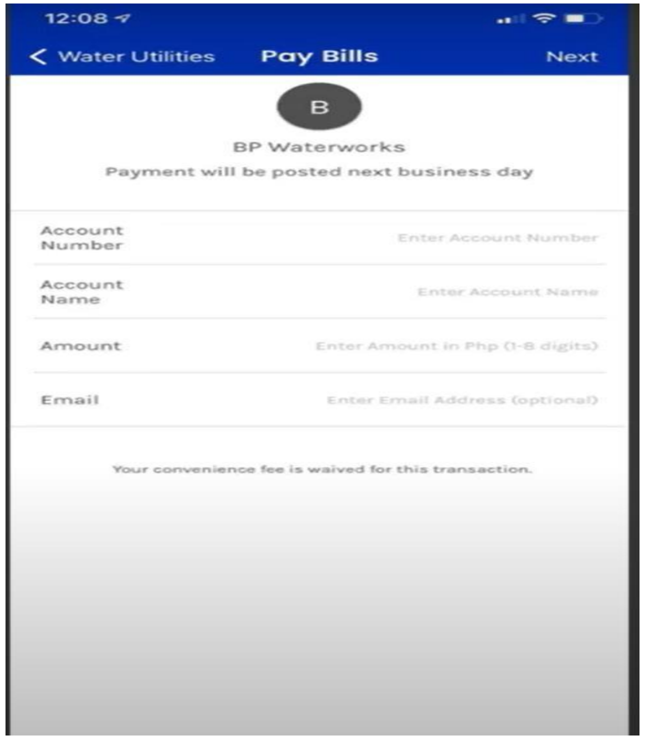
OSCE_10_WIN_Service...(en).zip - 481.64 MB Files/Test Folder

7 mins left

3.2.2a Reduce the User's Memory Load

➢Reduce demand on short-term memory.

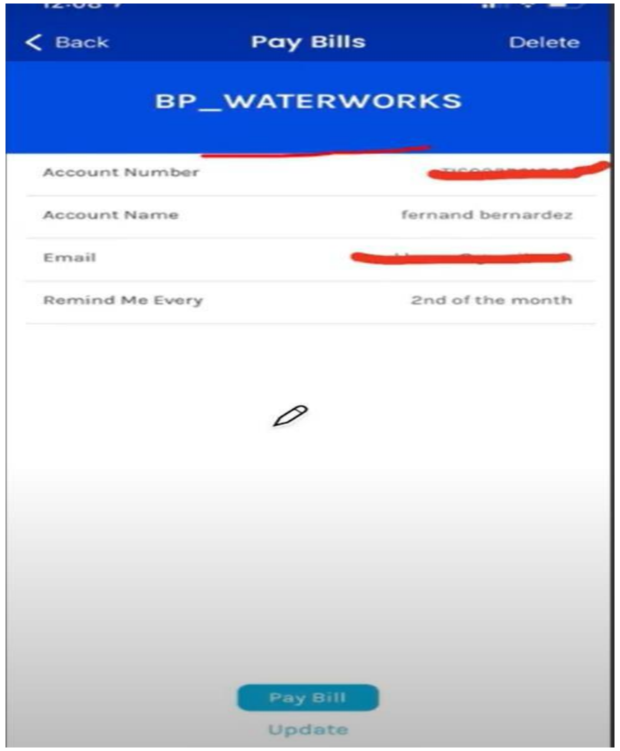
- The interface should be designed to reduce the requirement to remember past actions, input and results.



BP Waterworks
Payment will be posted next business day

Account Number	Enter Account Number
Account Name	Enter Account Name
Amount	Enter Amount in Php (1-8 digits)
Email	Enter Email Address (optional)

Your convenience fee is waived for this transaction.



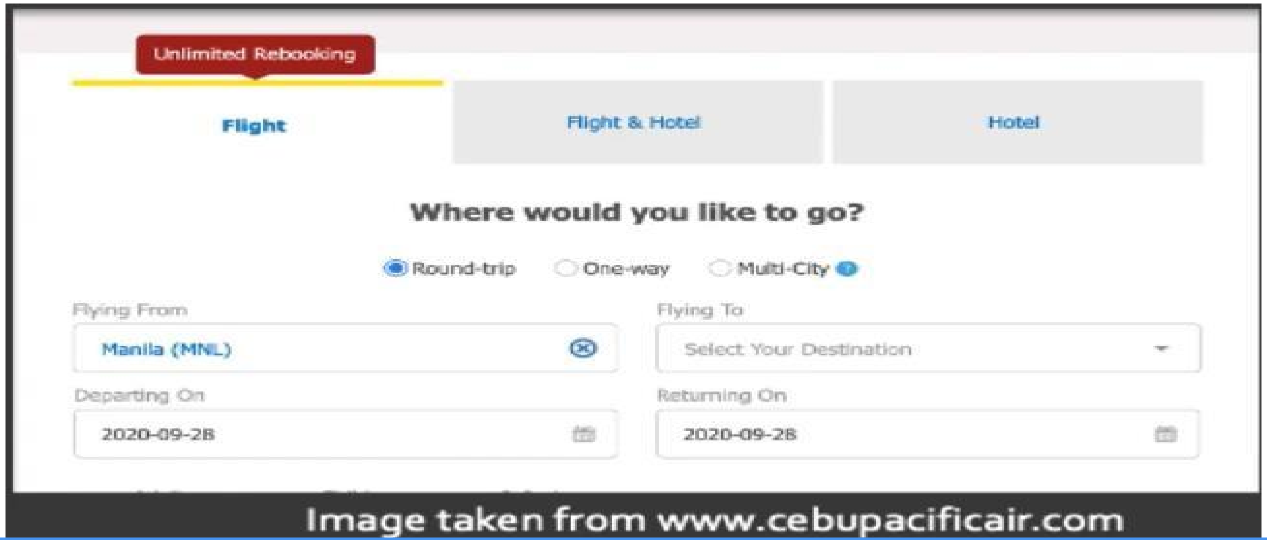
BP_WATERWORKS

Account Number	[redacted]
Account Name	fernand bernardez
Email	[redacted]
Remind Me Every	2nd of the month

3.2.2b Reduce the User's Memory Load

➢ Establish meaningful defaults.

- Default should be what the majority of your users will want.
- Use smart defaults(geolocation, automatic calculation)
- Don't use defaults for input fields that require user attention such as signing up to newsletters or accepting terms of use.



Unlimited Rebooking

Flight | Flight & Hotel | Hotel

Where would you like to go?

[✓ Round-trip] One-way Multi-City

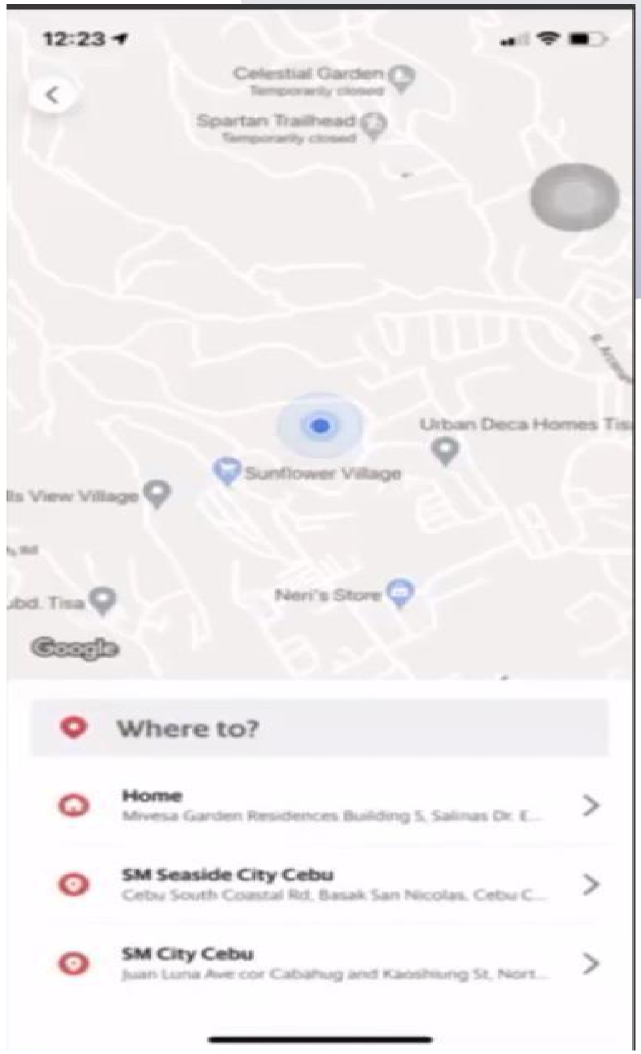
Flying From: Manila (MNL)

Flying To: Select Your Destination

Departing On: 2020-09-28

Returning On: 2020-09-28

Image taken from www.cebupacificair.com



Where to?

Home
Mivesa Garden Residences Building 5, Salinas Dr. E...

SM Seaside City Cebu
Cebu South Coastal Rd, Basak San Nicolas, Cebu C...

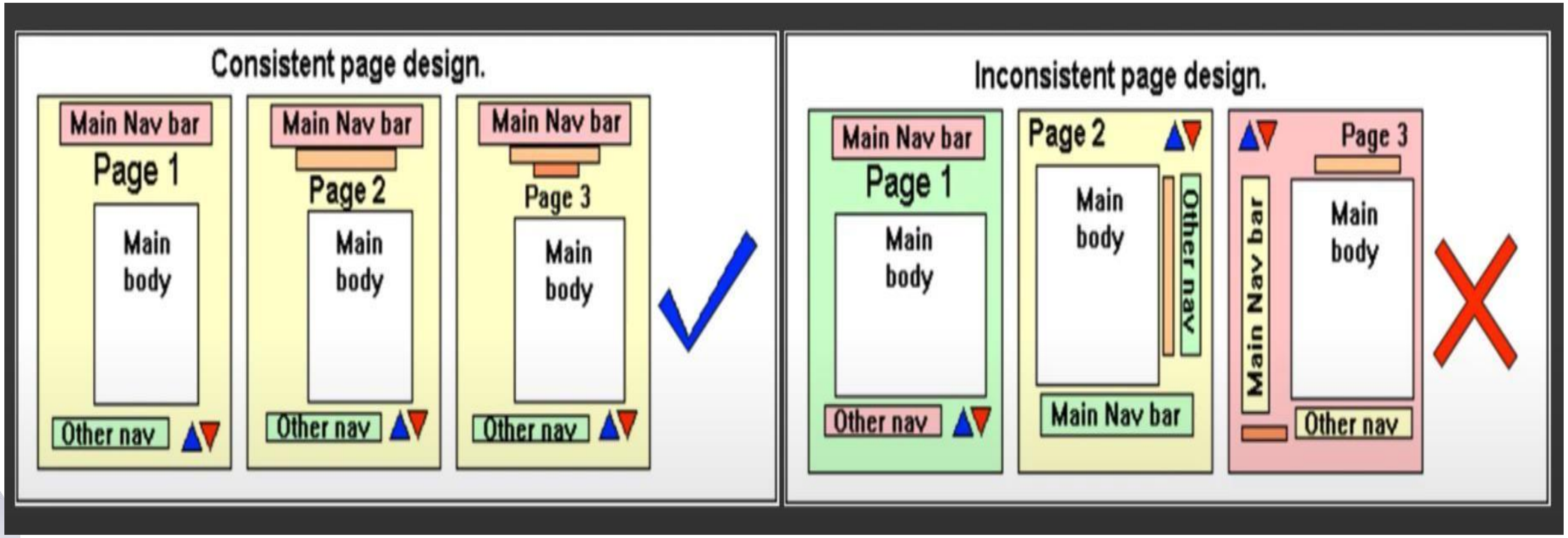
SM City Cebu
Juan Luna Ave cor Cabahug and Kaoshiung St, Nort...

3.2.2c Reduce the User's Memory Load

- The visual layout of the interface should be based on a real-world metaphor. 隐喻
- Using metaphors in UI design allows users to create a connection between the real world and digital experiences.

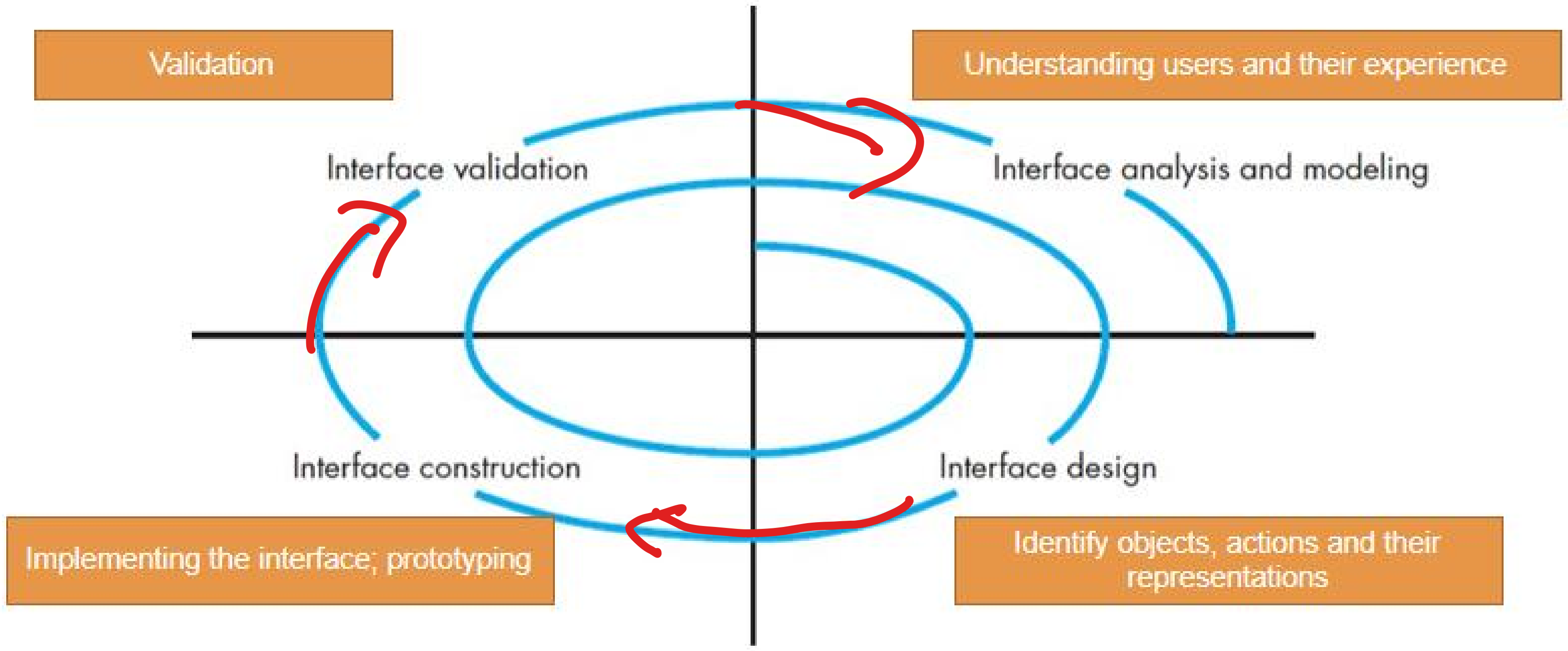
3.2.3 Consistent Interface

- All visual information is organized according to design rules that are maintained throughout all screen displays.



3.3 Interface Design Process

The analysis and design process for user interfaces is iterative and can be represented using a spiral model.


Validation
Understanding users and their experience
Interface validation
Interface analysis and modeling
Interface construction
Interface design
Implementing the interface; prototyping
Identify objects, actions and their representations

3.4 Interface Design Issues

- Response time: System response time has 2 important characteristics: length and variability.

- Help facilities: Help must be available for all system functions. Include help menus, print documents.

- Error handling: describe the problem in a language the user can understand. Never blame the user for the error that occurred.

- Application accessibility: especially for the physically challenged.

国际化

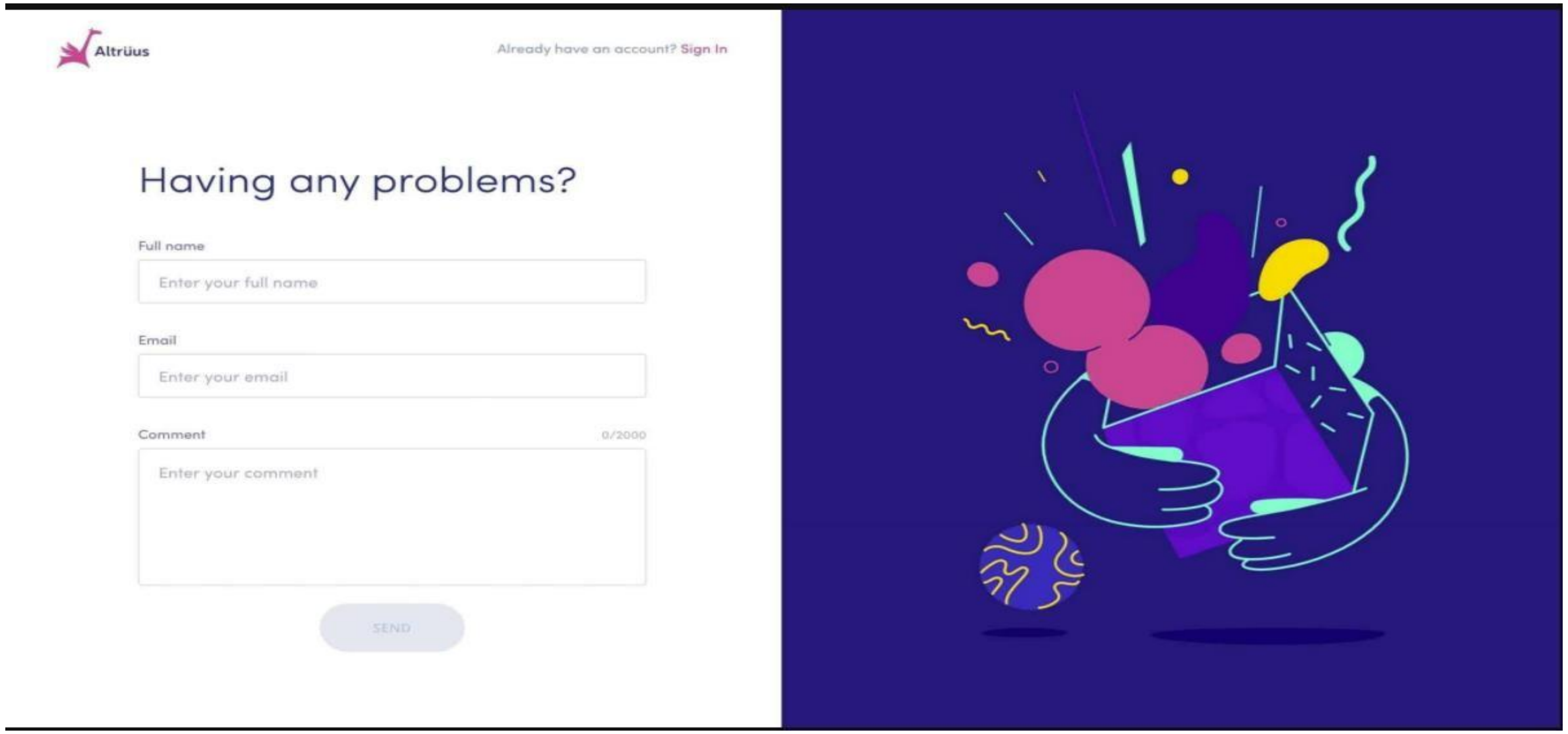
- Internationalization: The Unicode standard has been developed to address the daunting challenge of managing dozens of natural languages with hundred of characters and symbols.

3.4.1 Response Time Issues

- Response time
  - **Length:** It's reported that 47% of consumers expect a web page to load in two seconds or less. Users are easily frustrated with performance and usability issues like load times, small images, and network availability.
  - **Variability:** Variability refers to the deviation from average response time, and in many ways, it is the most important response time characteristic.

3.4.2 Help Facilities Issues

- Help facilities: Help must be available for all system functions. Include help menus, print documents



Having any problems?

Full name
Enter your full name

Email
Enter your email

Comment 0/2000
Enter your comment

3.4.3 Error Handling Issues

- Error handling: every error message or warning produced by an interactive system should have the following characteristics.
    - Describes the problem in jargon that the user can understand.
    - Provides constructive advice for recovering from the error.
    - Indicates any negative consequences of the error (e.g., potentially corrupted data files) so that the user can check to ensure that they have not occurred or correct them if they have

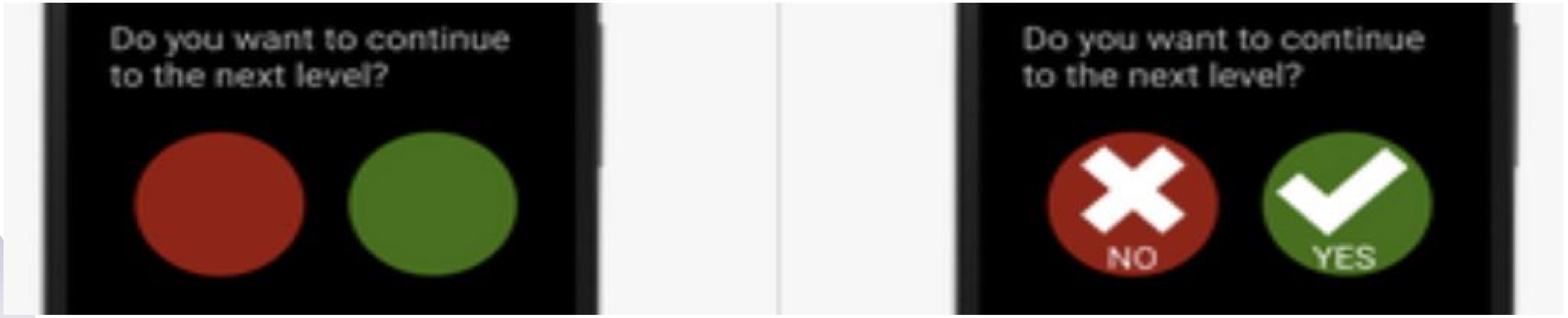
3.4.4 Accessibility Issues

Smartphones have small displays and small virtual keyboards. Some apps rely on speech or other sound signals.

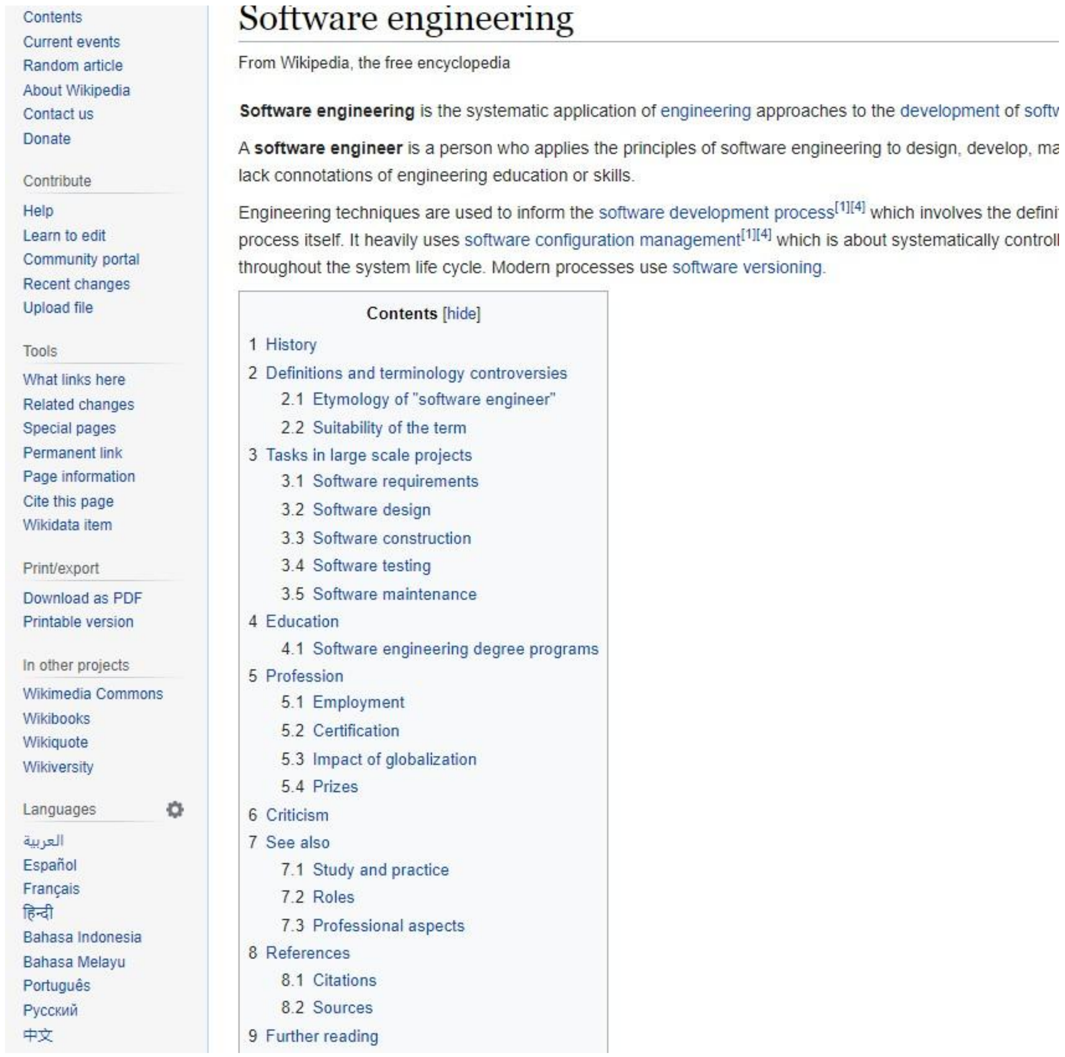
People with poor eyesight, color blindness, hearing loss, or clumsy fingers may have difficulty using your applications.

Android and iOS provide numerous accessibility features and provide online advice about how to build accessible apps.

For your user testing try to find people who do not have perfect eyesight, hearing, etc. Have testers of various ages. Older people are often less able to use touch sensitive screens.


Do you want to continue to the next level?
Do you want to continue to the next level?
NO
YES

3.4.5 Internationalization Issues


Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

In other projects
Wikimedia Commons
Wikibooks
Wikiquote
Wikiversity

Languages
العربية
Español
Français
हिन्दी
Bahasa Indonesia
Bahasa Melayu
Português
Русский
中文

Software engineering

From Wikipedia, the free encyclopedia

Software engineering is the systematic application of engineering approaches to the development of softv

A **software engineer** is a person who applies the principles of software engineering to design, develop, ma
lack connotations of engineering education or skills.

Engineering techniques are used to inform the software development process[1][4] which involves the defini
process itself. It heavily uses software configuration management[1][4] which is about systematically controll
throughout the system life cycle. Modern processes use software versioning.

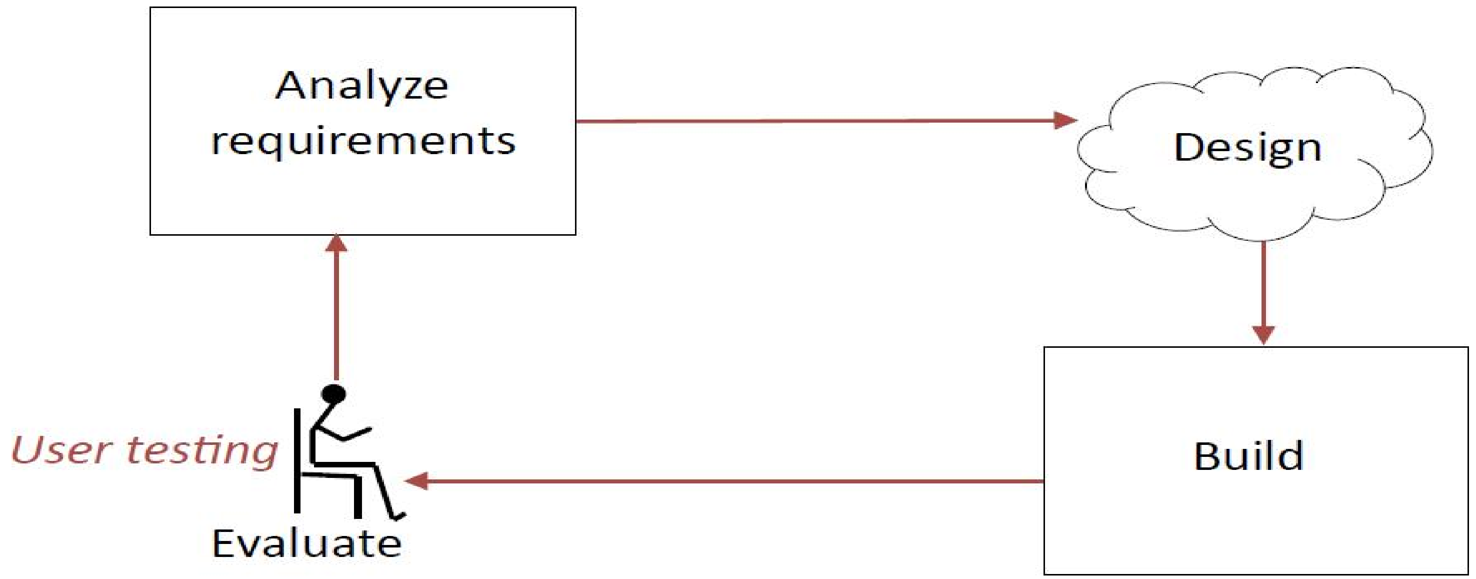
Contents [hide]

1 History
2 Definitions and terminology controversies
2.1 Etymology of "software engineer"
2.2 Suitability of the term
3 Tasks in large scale projects
3.1 Software requirements
3.2 Software design
3.3 Software construction
3.4 Software testing
3.5 Software maintenance
4 Education
4.1 Software engineering degree programs
5 Profession
5.1 Employment
5.2 Certification
5.3 Impact of globalization
5.4 Prizes
6 Criticism
7 See also
7.1 Study and practice
7.2 Roles
7.3 Professional aspects
8 References
8.1 Citations
8.2 Sources
9 Further reading

User interfaces should be designed to accommodate a generic core of functionality that can be delivered to **all who use the software**.

3.5a User Interface Design Evaluation

- Once you create an operational user interface prototype, it must be evaluated to determine whether it meets the needs of the user.



Whenever possible, the design and evaluation should be done by different people.

3.5b User Interface Design Evaluation

How do you measure usability?

Usability comprises the following aspects:

Effectiveness

The accuracy and completeness with which users achieve certain goals

Measures: quality of solution, error rates

Efficiency

The relation between the effectiveness and the resources expended in achieving them

Measures: task completion time, learning time, number of clicks

Satisfaction

The users' comfort with and positive attitudes towards the use of the system

Measures: attitude rating scales

From ISO 9241-11