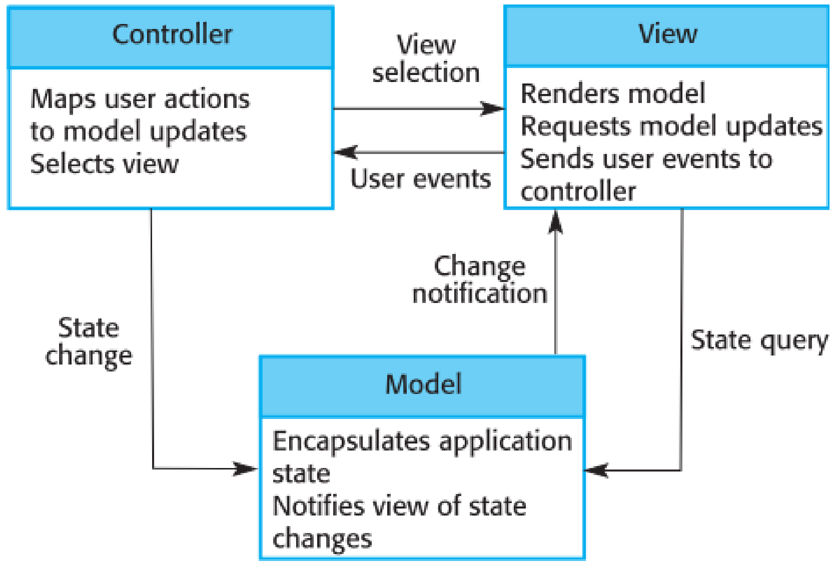
Universidade de Aveiro
Departamento de Electrónica,
Telecomunicações e Informática

S/W Patterns for Interactive Systems



Human-Computer Interaction

Beatriz Sousa Santos, 2020

- A complete application consists of:
  - User interface (UI)
  - Core functions of the application

- How can we effectively develop interactive applications with these two parts?

- Follow an established development methodology suited for interactive systems:

Architectural patterns

S/W Patterns – historical perspective

- 1987

  First small pattern language for **designing user interfaces**
  by K. Beck and W. Cunningham
  (inspired by building and urban architect C. Alexander)

- 1993

  Pattern Languages of Programming (PLoP) conference series
  by The Hillside Group
  https://www.hillside.net/conferences

- 1994

  "Design Patterns: Elements of Reusable Object-Oriented Software"
  by Erich Gamma et al. (aka the Gang of Four (GoF))
  https://hillside.net/patterns/about-patterns

S/W Patterns

- Proven **solution to a problem in a context** (Gamma et al., 1994)

- Each pattern documents a reusable solution, encapsulates knowledge about successful practices, and provides information about its usefulness and tradeoffs

- Many companies have written pattern collections:

  Amazon,

  Google,

  IBM,

  Microsoft,

  Oracle,

  Siemens, etc.

  https://hillside.net/patterns/about-patterns

Content of a Pattern

- Different formats are used for describing patterns, generally including:
  - **Name**
  - **Problem**
  - **Context**
  - **Solution**
  - Forces
  - Resulting Context
  - Examples
  - Rationale
  - Related Patterns
  - Known Uses
- May include an Abstract providing an overview of the pattern and indicating the types of problems it addresses and the target audience

https://pubs.opengroup.org/architecture/togaf9-doc/arch/chap22.html

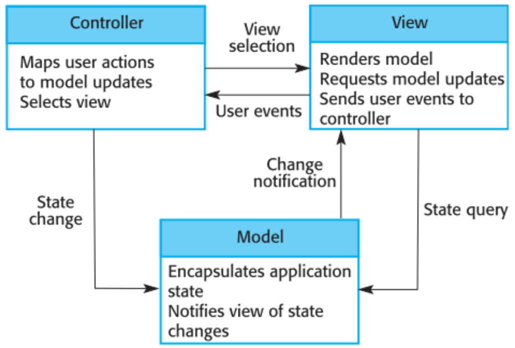
Forces of a Pattern

- Security, robustness, reliability, fault-tolerance
- Efficiency, performance, throughput, bandwidth requirements, space utilization
- Ease-of-use
- Ease-of-construction
- Completeness and correctness
- Scalability (incremental growth on-demand)
- Extensibility, evolvability, maintainability
- Modularity, independence, re-usability, openness, composability (plug-and-play), portability
- etc.

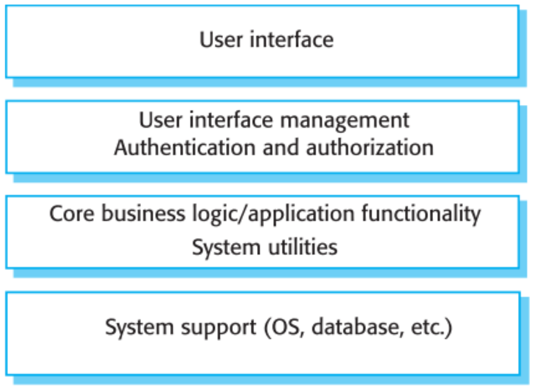
https://pubs.opengroup.org/architecture/togaf9-doc/arch/chap22.html

Architectural patterns often used in interactive s/w

– Model View Control (MVC)



– Layered architecture


User interface
User interface management
Authentication and authorization
Core business logic/application functionality
System utilities
System support (OS, database, etc.)

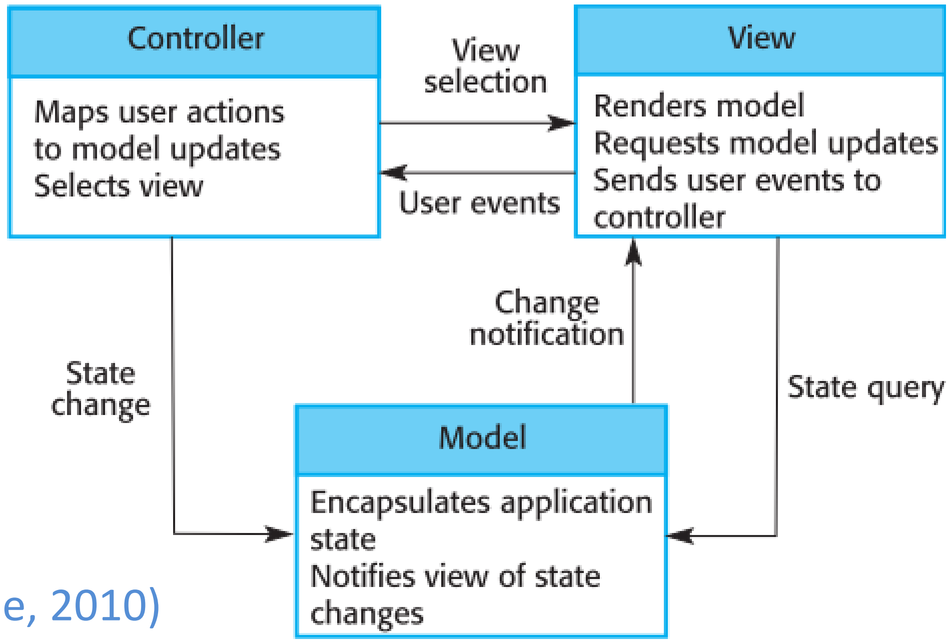
(Sommerville, 2010)

Model–view–controller (MVC)

- MVC is an architectural pattern/paradigm commonly used for interactive systems
- Was proposed in the 70’s as a computational architecture for interactive programs by the designers of SmallTalk (one of the first object-oriented and modular languages)
- It expresses the "core of the solution" to a problem while allowing it to be adapted for each system

Model–view–controller (MVC)

- The application is divided in three parts:
  - Model
  - View
  - Controller



(Sommerville, 2010)

- Separating internal representations of information from the ways it is presented to and accepted from the user
- It decouples these major components allowing for efficient code reuse and parallel development.

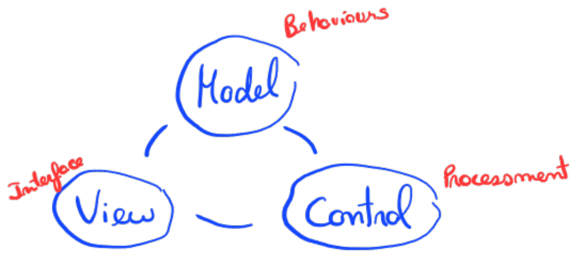
- Originally developed for desktop computing, MVC has been widely adopted as an architecture for Web applications (and others…)

- Frameworks vary in their interpretations in the way that the MVC responsibilities are divided between the client and server

- Some web MVC frameworks take a thin client approach that places almost the entire model, view and controller logic on the server: the model exists entirely on the server

- Other frameworks allow the MVC components to execute partly on the client

MVC Components



- The ***model*** is the central component of the pattern:

It expresses the application's behavior in terms of the problem domain, independent of the UI. It manages the data, logic and rules of the application

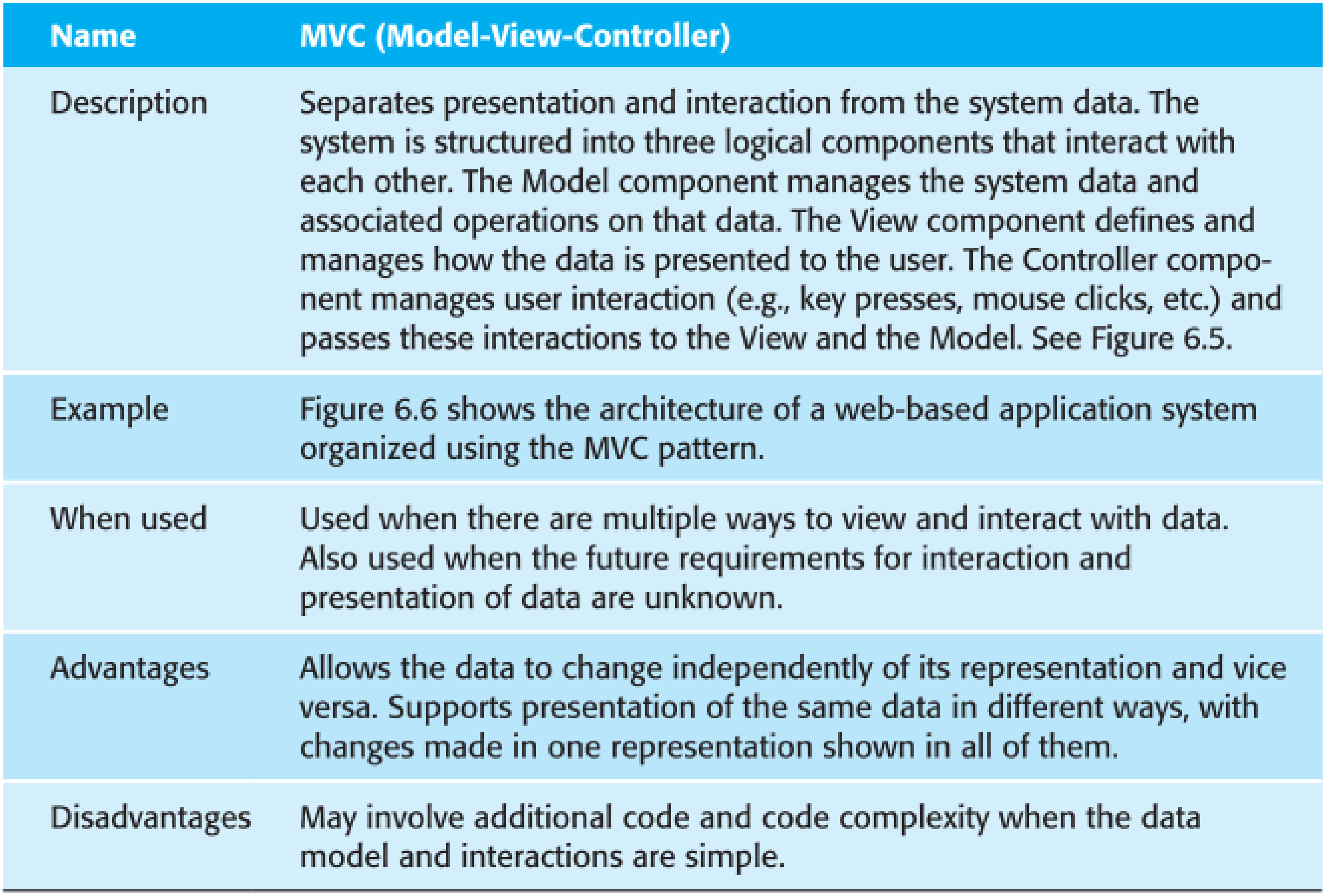
- A ***view*** can be any output representation of information:

Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants

- The ***controller*** accepts input and converts it to commands for the model or view

MVC Interactions
Model View Control

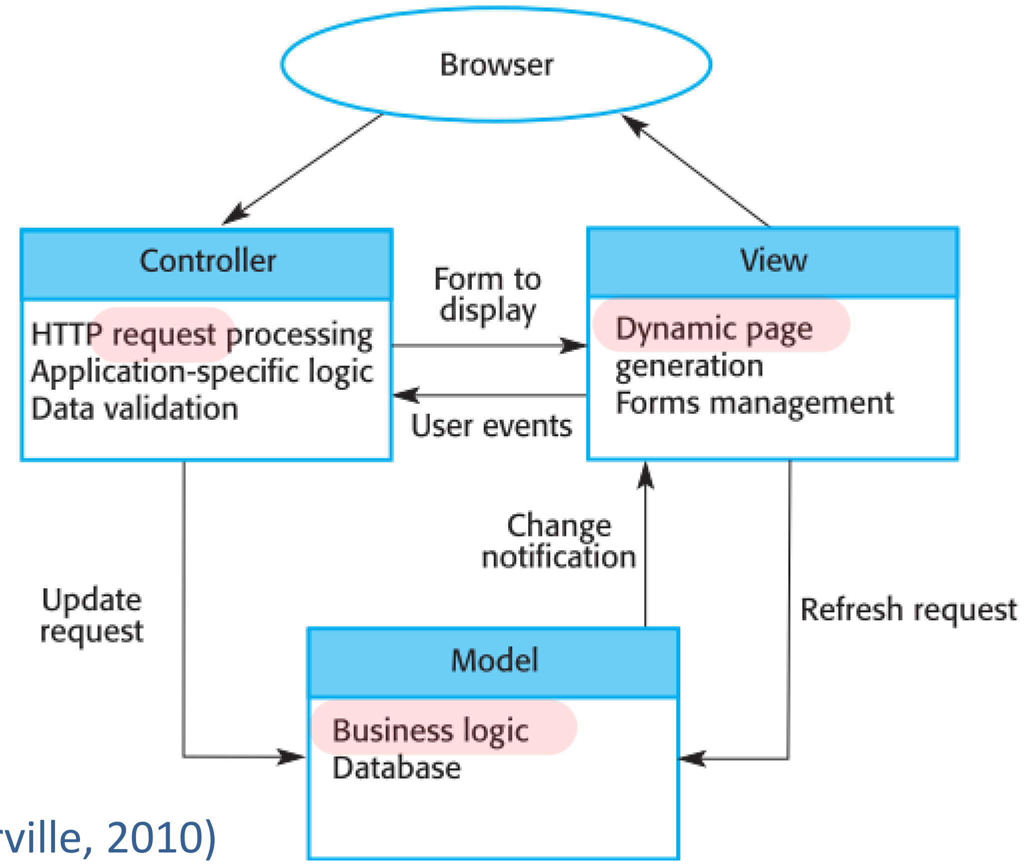
- This design defines the interactions among the three components
- The model is responsible for managing the data of the application. It receives user input from the controller
- The view means presentation of the model in a particular format
- The controller is responsible for responding to the user input and perform interactions on the data model objects. The controller receives the input, optionally validates the input and then passes it to the model



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 6.5.
Example	Figure 6.6 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	May involve additional code and code complexity when the data model and interactions are simple.

(Sommerville, 2010)

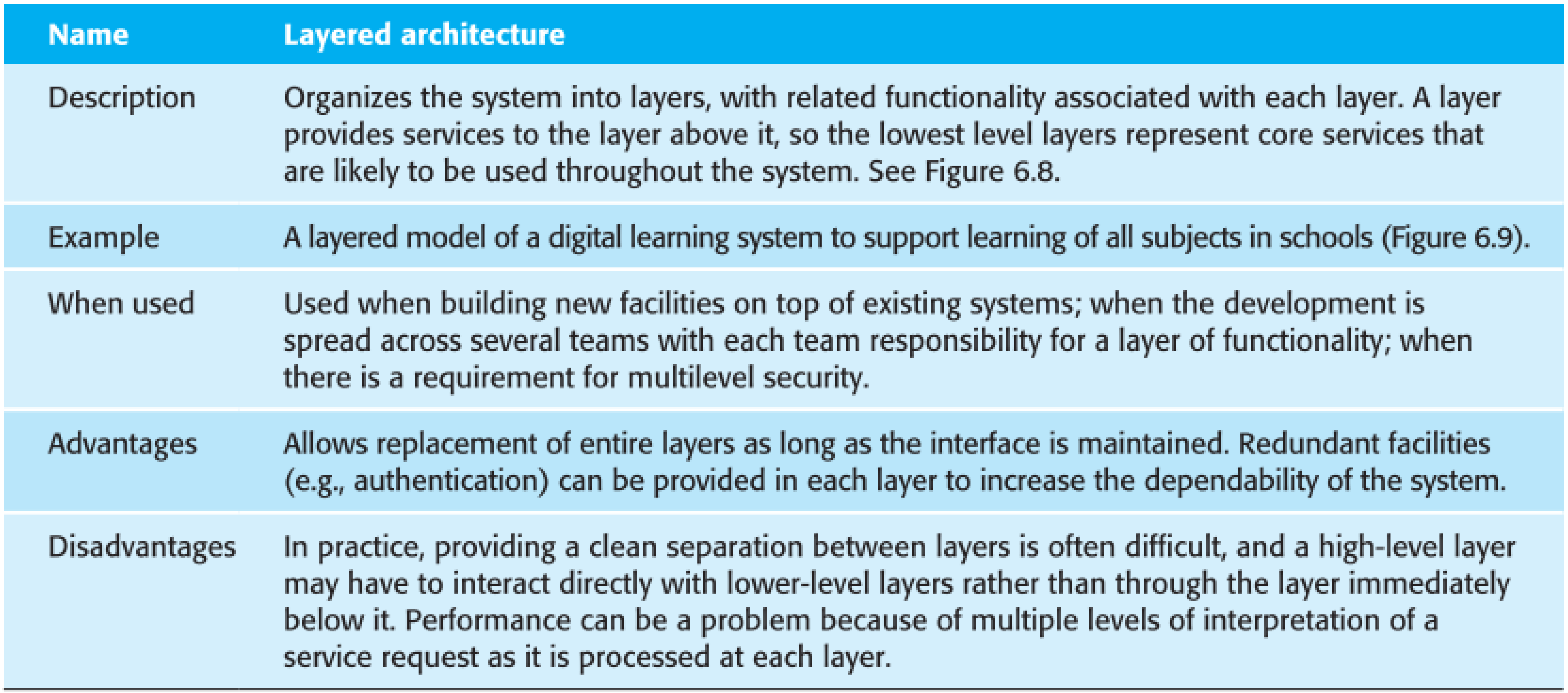
Example:
Web application architecture using MVC



(Sommerville, 2010)

Layered Architecture

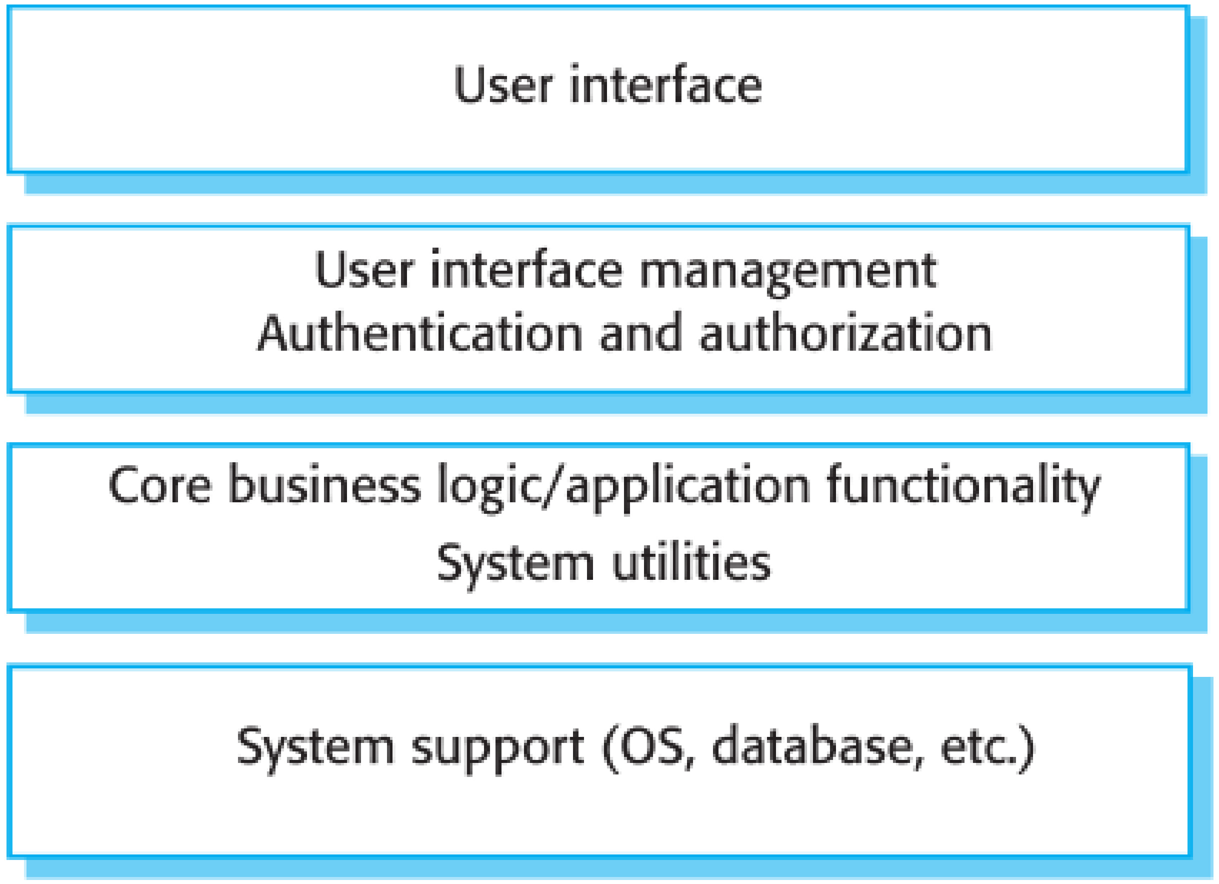
- Separation and independence are fundamental to S/W architectural design because they allow changes to be localized
- The Layered Architecture pattern is another way of achieving separation and independence
- The system functionality is organized into separate layers
- Each layer only relies on the facilities and services offered by the layer immediately beneath it
- It supports the incremental development of systems.
- The architecture is also changeable and portable.



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 6.8.
Example	A layered model of a digital learning system to support learning of all subjects in schools (Figure 6.9).
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 multilevel security.
Advantages	Allows replacement of entire layers as 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.

(Sommerville, 2010)

Example:
A generic layered architecture


User interface
User interface management
Authentication and authorization
Core business logic/application functionality
System utilities
System support (OS, database, etc.)

(Sommerville, 2010)

Main bibliography

- Gamma, R., Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994
- https://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612
- Kim, G., *Human–Computer Interaction- Fundamentals and Practice*. CRC Press, 2015
- Sommerville, I., *Software Engineering*, 10th ed., Pearson Education, 2016

  https://www.amazon.com/Software-Engineering-10th-Ian-Sommerville/dp/0133943038