System Architecture

Architecture Design

- Understading how a software system should be organized
- Designing the overall structure of the system

Often represented using simple block diagrams

- High-level picture of the system structure
- Each block represents a component in the system
- Arrows represent data or signals passed from one component to another

The advantages: easy to understand. Disadvantages: too informal, doesn't show the type of relationships among components or externally visible properties.

Architectural Decision

- How will the system be distributed across hardware cores and processors?
- What will be the fundamental approach used to structure the system?
- What architectural pattern or syles might be used?
- How should the architecture of the system be documented?

4+1 View Model

- Logical view
  - Key abstractions in the system as objects or object classes
  - Relate system requirements to objects
- Process view
  - How the system is composed of interacting processes
  - Usingful for making judgement about non-functional requirements
- Development view
  - How the software is decomposed for development
  - Useful for software managers and programmers
- Physical view

  - How the system hardware and software components are distributed across the processors in the system
  - Useful for system engineers
- +1 link all views through common use cases or scenarios

Architectural Patterns

A pattern is a means for representing, sharing, and reusing knowledge; and it may be represented using tabular and graphical descriptions.

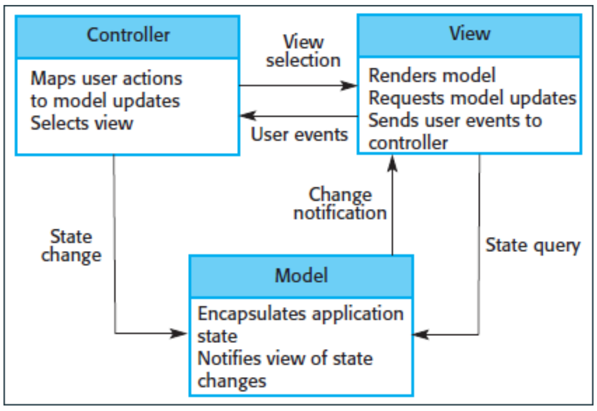
An architectural pattern is a stylized, abstract description of good design practice. It has been tried and tested in diff environments and it should include info about when they are and are not useful and the strengths and weaknesses.

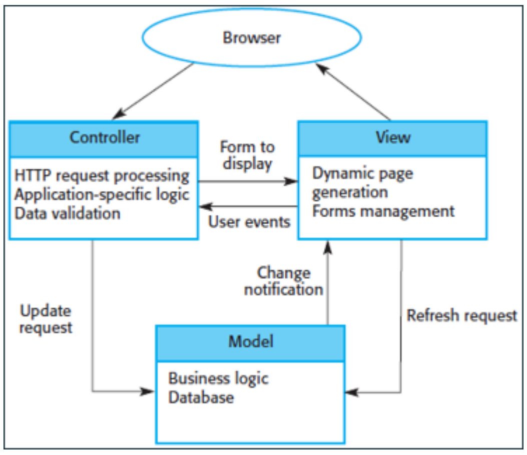
Examples:

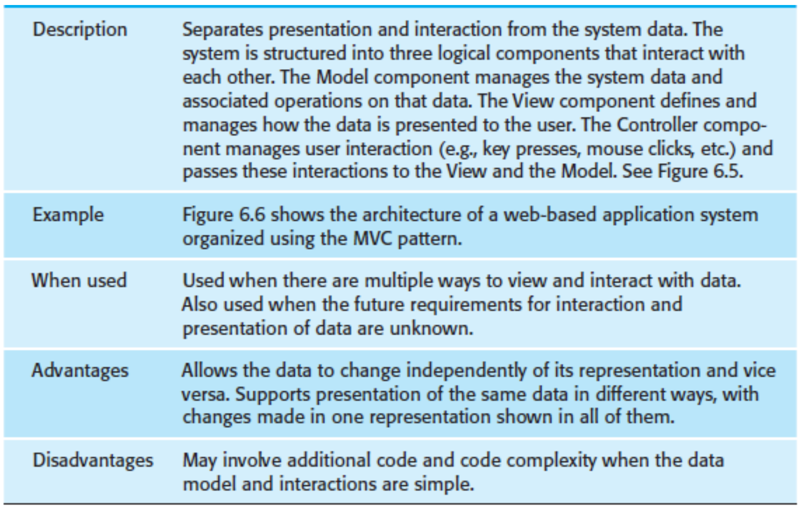
- Model-View-Controller (MVC)
- Layered architecture
- repository architecture
- Client-server architecture
- Pipe and filter architecture

Model-View-Controller

Often used for web applications and it splits the system into three different components: Model - stores and manages data; View - GUI; Controller - converts input from the View into demands to retrieve or update data from the Model and passes info from Model to the View.



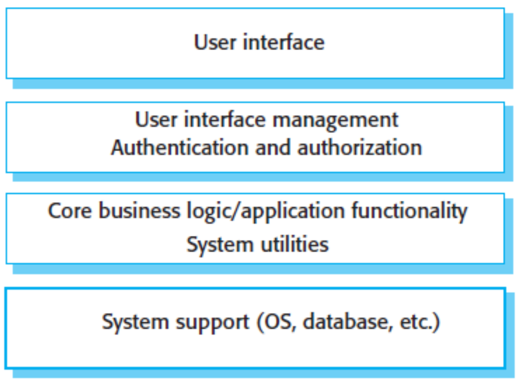


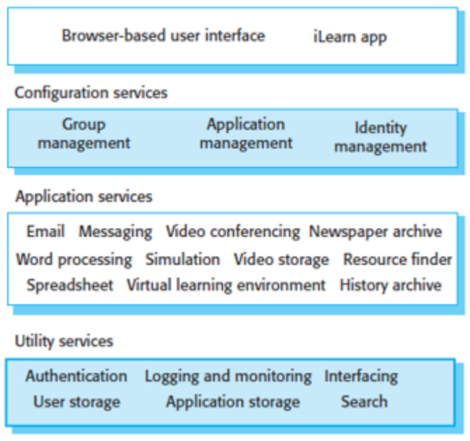


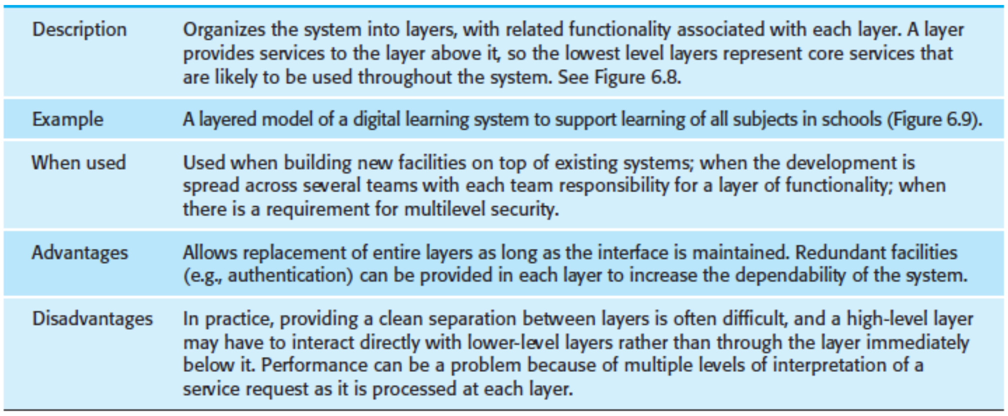
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented 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.

Layered Architecture

Used to model the interfacing of the subsystems and it supports incremental dev of subsystems in diff layers. When a layer interface changes, only the adjacent layers are affected. The number of layers is arbitrary.


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

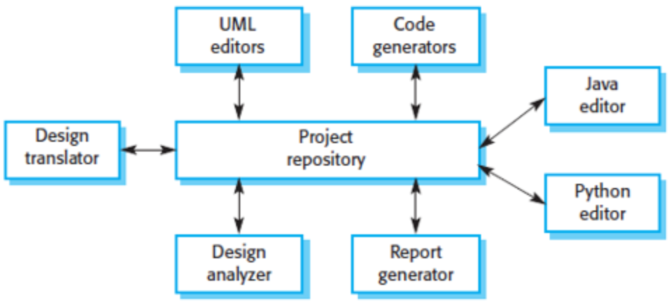


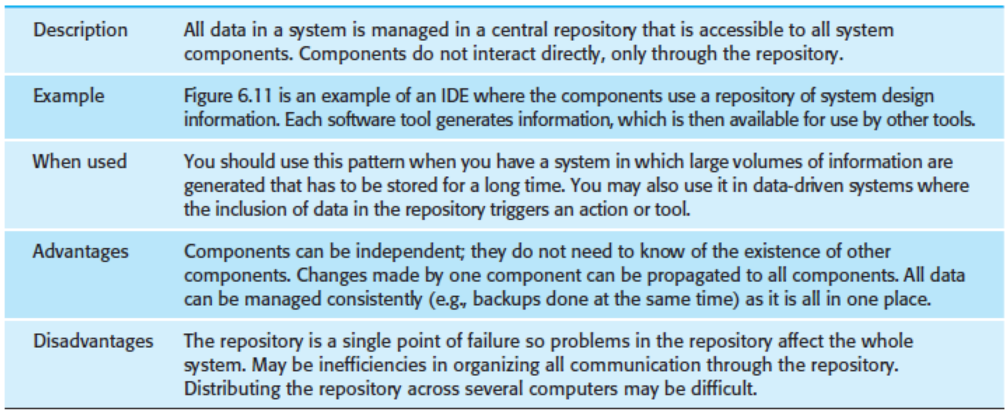


Description	Organizes the system into layers, with related functionality associated with each layer. A layer provides services to the layer above it, so the lowest level layers represent core services that are likely to be used throughout the system. See Figure 6.8.
Example	A layered model of a digital 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.

Repository Architecture

Subsystems must exchange data which can be done by storing datat in central database and which can be accessed by all subsystems or each subsystem maintains its own database and passes data to other subsystems. When large amounts of data need to be shared, the repository model is most commonly used.

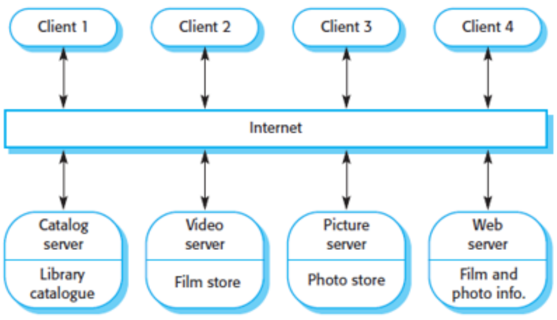


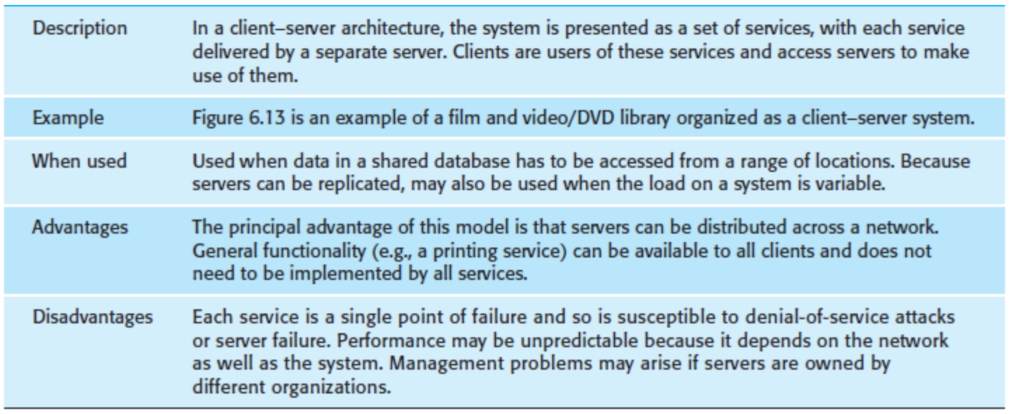


Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
Example	Figure 6.11 is an example of an IDE where the components use a repository of system design information. Each software 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.

Client-Server Architecture

Distributed system model which shows how data and processing is distributed across a range of components. It is a set of stand-alone servers, a set of clients, and a network to connect the two.

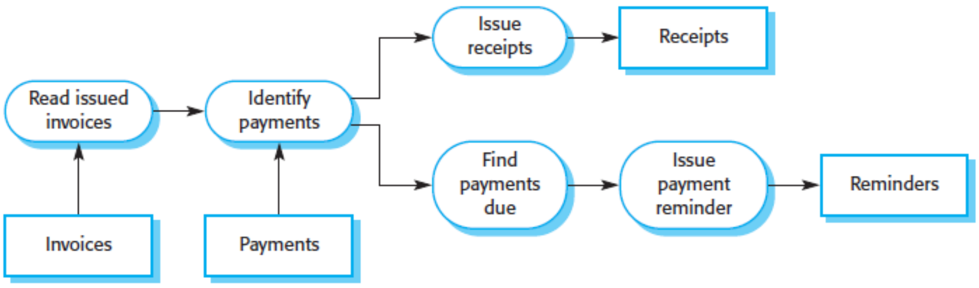


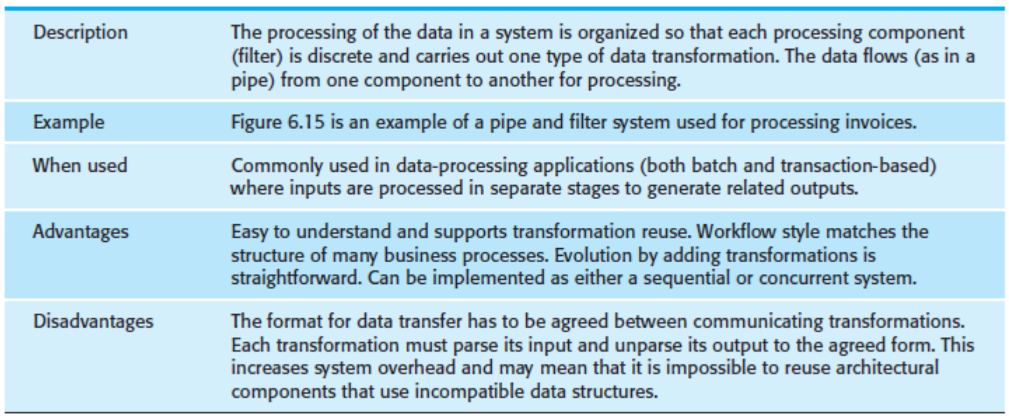


Description	In a client–server architecture, the system is presented as a set of services, with each service delivered by a separate server. Clients are users of these services and access servers to make use of them.
Example	Figure 6.13 is an example of a film and video/DVD library organized as a client–server 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 and so is susceptible to denial-of-service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. Management problems may arise if servers are owned by different organizations.

Pipe and Filter Architecture

Data comes into a process, gets transformed and the resulting output is used as the input for the next process. Not a good choice for interactive systems.





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 6.15 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 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 architectural components that use incompatible data structures.

Design Patterns

Architecture vs Design

Software architecture gives the high-levvel organization of the software and it identifies the main structural components in a system and the relationships between them.

Sofware design gives the code-level design and it identifies what each class doing, their relationships and scope.

Design patterns

A patter is a description of the problem and the essence of its solution. A design pattern is a way of reusing abstract knowledge about a problem and its solution.

It should have High cohesion and low coupling.

Cohesion

Cohesion is the degree of interaction within a module.

There are 7 levels:

- Functional (best): all essential elements for a single task is in one module
- Sequential
- Communicational

- Procedural
- Temporal
- Logical
- Coincidental (worst): Elements have no conceptual relationship other than the location in the source code.

Coupling

Coupling is the degree of interaction between modules.

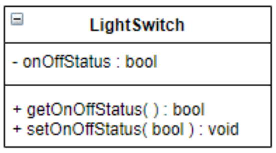
There are 5 levels:

- Data (best): modules are independent from each other and communication by only passing data
- Stamp
- Control
- Common
- Content (worst): a module can modify the data of another module.

Encapsulation

Def: Hides the details from everything; is the process to contain information.

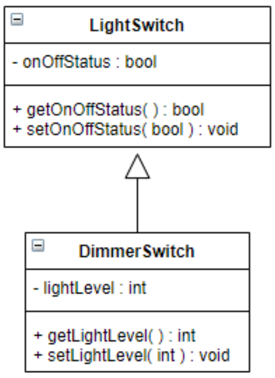
Implementation: variables of a class are private and functions are public so other classes can get and set the data.



Abstraction

Def: Shows only what is needed and hides the unwanted info. It is the process to gain info.

Implementation: Create an abstract class and extend to show more info or add complexity.


LightSwitch
- onOffStatus : bool
+ getOnOffStatus( ) : bool
+ setOnOffStatus(bool) : void
DimmerSwitch
- lightLevel : int
+ getLightLevel( ) : int
+ setLightLevel(int) : void

Design Problems and Patterns

To use patterns in your design, you need to recognize that any design problem may have an associated pattern that can be applied.

There is three catagory: Creational patterns which is focused on creating objects; Structural patterns which setups the relationship between objects; Behavioral patterns which defins how objects interact with each other.

Examples:

- Iterator
  - Provide a standard way of accessing the elements in a collection, irrespective of how that collection is implemented.
- Facade
  - Tidy up the interfaces to a number of related objects that have often been developed incrementally
- Observer
  - Tell several objects that the state of some other objects has changed
- Decorator
  - Allow for possibility of extending the functionality of an existing class at run-time

Project Management

Focused on making sure the software is delievered on time and is built in accordance with the requirements. Always needed b/c software dev is always subject to budget and software contraints.

Project success criteria

Important goals of project management for most engineering projects:

- Deliver software to the customer at the agreed time
- Keep overall costs within budget
- Deliver software that meets the customer's expectations
- Maintain a coherent and well-functioning development team

Software project differences

The product is intangible and software projects are often "one-off" projects. Software processes are variable and org specific.

Factors influencing project management:

- Company size
- software Customers
- software size
- software type
- org culture
- software dev processes

Project management activities

Project planning

- Project manager are responsible for planning, estimating, and scheduling project dev and assigning tasks to people.
- Ensures the work is build to required standards and monitor progress to check dev is on time and within budget

Risk management

- Project managers assess the risks that may affect the project, monitor the risks, and take action when the problems arise.

People management

- Project managers choose people for their team and establish ways of working that lead to effective team performance.

Reporting

- Project managers report the progress of the project to customers and managers of the company.

Risk management

Risk identification

- Estimation
- Organizational
- People
- Requirements
- Tech
- Tools

Risk analysis

Need to choose the risks that are most significant and monitor them. Assess the probability of a risk occuring and the lv of effect the risk may have on the project.

Risk planning

Need to ask "what-if" questions: What if the performance of open-source software is inadequate? What if an economic downturn leads to budget cuts? etc.

Develop strategies to manage these risks: Avoidance: reduce the chance of them happening; minimization: reduce the impact; contingency: prepare for the worst and have a plan to deal w/ it.

Effort allocation

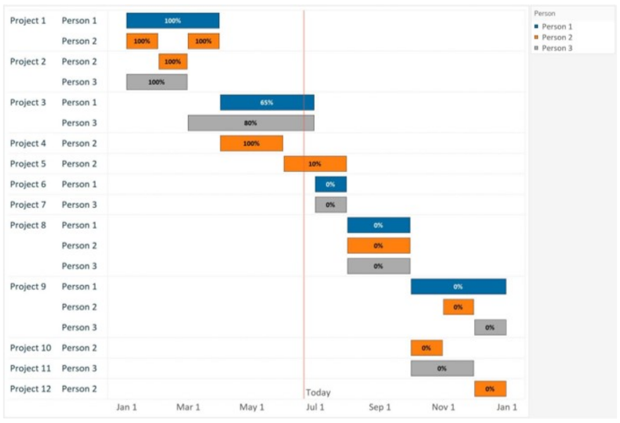
40-20-40% rule: 40-50% for front-end; 15-20% for construction (coding); 30-40% for testing and installation. Some engineering managers believe more than 40% of overall effort should be used for analysis and design. Some that use agile development say less time should be spent on the "front end" and more during the construction phase.

Setting up a schedule

Common items needed:

- Define deliverables and milestones
- Identify tasks that belong to deliverables
- Identify relationships between deliverables and activities
- Determine type and size of resources needed to complete task
- Allocate people to activities
- Create a system to track the progress of each activity

Gantt Chart


Project 1
Person 1
Person 2
Project 2
Person 2
Person 3
Project 3
Person 1
Person 3
Project 4
Person 2
Project 5
Person 2
Project 6
Person 1
Project 7
Person 3
Project 8
Person 1
Person 2
Person 3
Project 9
Person 1
Person 2
Person 3
Project 10
Person 2
Project 11
Person 3
Project 12
Person 2
Today
Jan 1
Mar 1
May 1
Jul 1
Sep 1
Nov 1
Jan 1
Person
Person 1
Person 2
Person 3