Objects and Design Lecture Notes

Notes based on in-person CS2340 Lectures in Spring 2024.

Krish Katariya

Last updated: **January 30, 2024**

Contents

1. Requirements Engineering

Definition 1.1

Requirements Engineering, (RE) is the process of establishing the needs of **stakeholders** that are to be solved by software.

> So, what is the importance of RE?

One of the reasons that RE exists today is because of most software failures are actually caused by **poor requirement definitions**, rather than a lack of qualified resources or inadequate risk management.

Software runs on some hardware and is developed for a purpose. Re is about identifying that process. However, identifying that task can often be an extremely hard task.

- Sheer **complexity** of the purpose/requirements
- People often don't even know what they want themselves
- Multiple stakeholders can have conflicting requirements

Definition 1.2

Non-functional requirements: criteria that can be used to judge the operation of a system, rather than specific behaviors.

Basically, they describe how the systems performs aka. its quality, rather than what it does (actions).

Definition 1.3

User Requirements: Requirements written for customers
- Often written in natural language and doesn't have any technical details

System Requirements: Requirements written for developers
- Detailed functional and non-functional requirements
- Clearly and rigorously specified

Note 1.1

Sometimes, you may need to prioritize some resources over others, known as resource prioritization. This often happens if aren't able to satisfy all the requirements so you need need to prioritize them.

2. Software Architecture

Definition 2.1

Software Architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

In other words Software Architecture is:
- The set of design decisions that must be made early
- The decisions that you wish you could get early on
- The decisions that are **hard to change**

Note 2.1

Software architects are often people with great communication skills
- They need to understand, and explain requirements to different stakeholders
- Can translate client's features to technical language

A good software architecture represents a high internal quality of software, one that is not necessarily visible to customers and users but matters in the long term.

Architecture is often about the long term, because it may change a little over time but will rarely change a lot.

2.1. Client Server Architecture

Definition 2.1.1

Client Server Architecture: a system with an organized set of services and associated servers, and clients that access and use the services.

Major Components:
- Set of servers that offer services to other components, such as print servers, email servers, etc
- Set of clients that call on the services offered by servers
- A network that allows the clients to access the servers

Note 2.1.1

The client and server can actually be on the same computer! However, they need to be run on different processors.

Example 2.1.1

An example of client-server architecture is youtube. The client is the user device (integrated UI), and the server is youtube.

Note that the architectural style is monolithic, meaning a single executable performs all of the server-side functions for the application.

Pros

- It is a centralized system that keeps all the data and its controls in one place
  - High level of scalability, organization, and efficiency

Cons

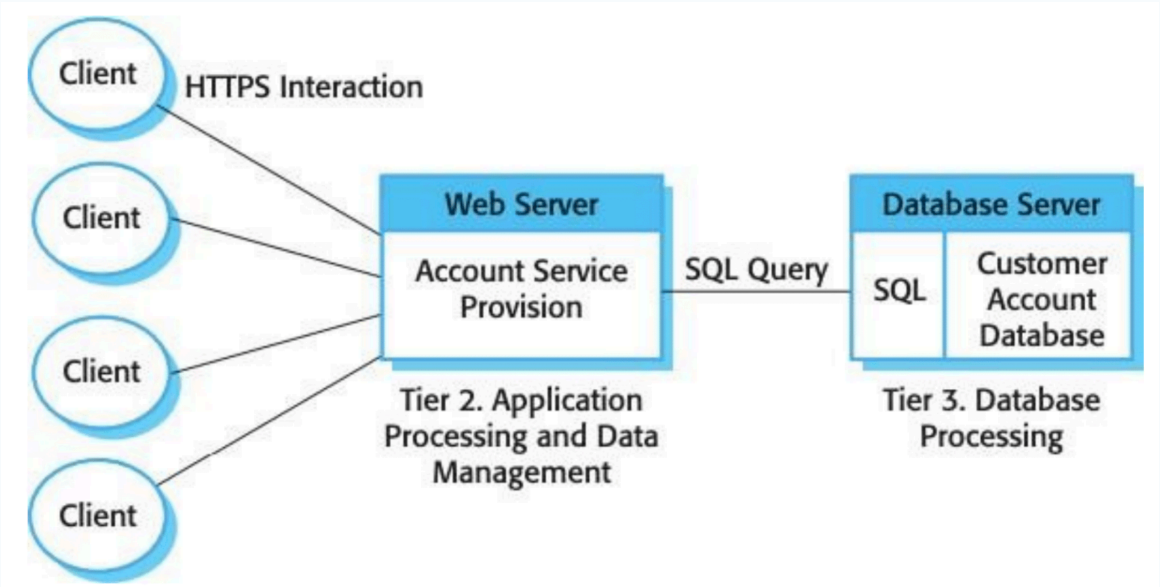
- Each service is a new **point of failure**
- It can be susceptible to DDOS attacks

2.2. N-Tier Architecture

One of the fundamental problems with client-server architecture is that logical layers in the system (presentation, application processing, data management, and database) must be mapped onto two processors: client and server;

In N-tier architecture, the different layers are separate processes that may execute on different processors.

Example 2.2.1



In this example, we have a 3 tier model w/ a web server and database.

2.3. Peer-to-Peer Architecture

Definition 2.3.1

Peer-to-peer systems: decentralized systems in which computations can be carried out by any node in the network. There is no clear distinction between clients and servers.

When to use?

- When the system is **computationally intensive** and you can break up the process into many independent steps

- Where the system involves the exchange of information between individual computers on a network and there is no need for this information to be centrally stored or managed

Pros:
- Highly redundant and very fault-tolerant

Cons:
- Concerns about privacy and security

2.4. Blackboard Architecture

Attention 2.4.1

This section is non-lecture notes because I don't understand the lecture for this. **Blackboard architecture** is an architecture that is used to solve difficult problems where we don't have a pre-defined algorithm to use. It is inspired behind how people work at blackboards adding incremental solutions until they come to a collective solution together.

I think(?) that **knowledge sources** are modules/algorithms that are able to help potentially solve the problem. They usually have a specialized frame of thinking unique to its others.

Flow of Information:
1. Initialization: the blackboard gets setup with the initial problem and any data.
2. Activation: the controller selects and activates one or more knowledge sources based on the current state of the problem and available data
3. Execution: the knowledge sources independently analyze the problem and apply specialized algorithms or techniques and produce partial solutions.
4. Conflict resolution: if multiple knowledge sources conflict, use a conflict resolution mechanism
5. Update: the knowledge sources update the blackboard with their outputs.
6. Repeat.

Key ideas are that problem-solving is now:
- **Incremental** - complete solutions are constructed piece by piece
- **Opportunistic** - system chooses the actions to take next that will allow it to make the best progress.

Main Components
- Blackboard - a structured global memory containing objects from the solution space
- Knowledge sources - specialized modules w/ their own representation for reading/writing on blackboard
- Control component - selects, configures, and executes modules

2.5. Pipe and Filter Architecture

The **pipe and filter architecture** style is data-centric and structured around how data flows through the application.

- Firstly, the application takes in data as input
- Next, a series of transformations are sequentially applied
- Finally, the application returns the processed data as output

Note 2.5.1

The name for pipe and filter comes from Unix where you can link processes using pipes

Example 2.5.1

One example is a company that sends invoices to people. First you read invoices, then identify payments on the invoices, then either ask to pay or send reminders.

Pros:

- Easy to understand and supports transformation reuse
- Workflow style matches the structure of many business processes
- Evolution by adding transformations is Easy
- Can be implemented as a sequential or concurrent system

Cons:

- 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 can increase overall system overhead

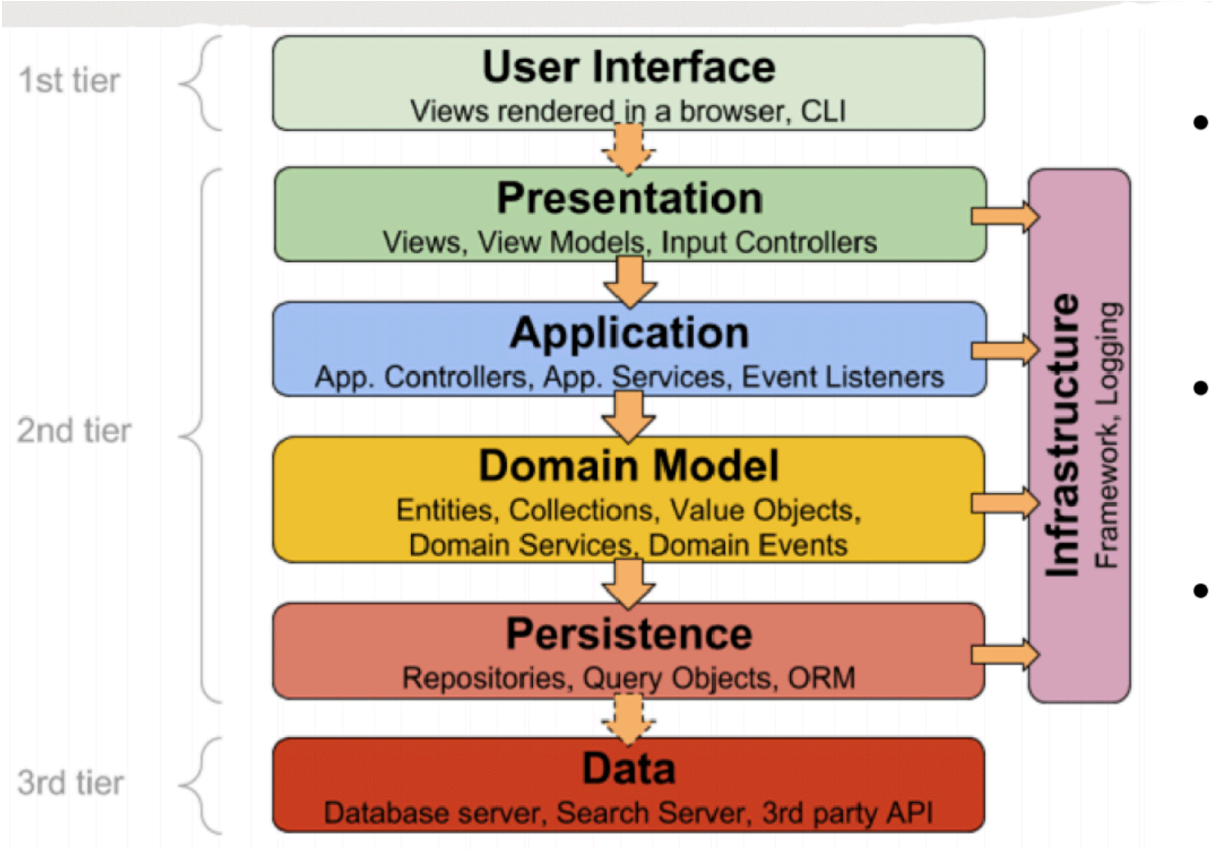
2.6. Layered Architecture

Definition 2.6.1

Layered Architecture: organizes the systems into layers with related functionality/logic associated with each layer.

In each **layered system**, each layer

- Depends on the layer beneath it
- Is independent of the layers on top of it



Pros:

- Changes in one layer usually doesn't impact other layers
- Modularity: promotes separation of Concerns

Cons:
- Potential redundant functionality
- Potential rigidity
- Interaction complexity

2.7. Model-View-Controller (MVC) Pattern

This architecture follows the layered approach. Here, we separate presentation and interaction from the system data

Instead, the system is structured into three logical components:
- **Model Component:** manages the system data and associated operations on the data
- **View Component:** defines and manages how the data is presented to the user
- **Controller Component:** manages user interaction (clicks, etc)

Model and View are connected, as the data from model updates view over time.

2.8. Model-View-ViewModel

This is a derivative of the popular MVC architectural pattern. However, for MVVM, **model and view are not connected**. Instead, the **ViewModel** acts as an intermediary, managing differences between model and view.

Note 2.8.1

MVVM is one of the recommended architectures for developing android applications.

2.9. Event Driven Architecture

Definition 2.9.1

Event Driven Architecture: consists of events producers that create a stream of events and event consumers that listen for those events.

Using EDA, an object can broadcast (look out?) for one or more events, and other objects register an interest in that event. When the event is invoked, the system runs all the related/important procedures.

This is useful when multiple subsystems must process the same events and **real-time processing** is needed with low lag.