

Software Architecture & Design

CS350 Introduction to Software Engineering

Shin Yoo

Architecture

 180 languages ▾

Article [Talk](#)

[Read](#) [View source](#) [View history](#)

From Wikipedia, the free encyclopedia




For the profession, see [Architect](#). For other uses, see [Architecture \(disambiguation\)](#).

Architecture is the art and technique of designing and building, as distinguished from the skills associated with construction.^[3] It is both the process and the product of sketching, conceiving,^[4] [planning](#), [designing](#), and [constructing buildings](#) or other [structures](#).^[5] The term comes from [Latin](#) *architectura*; from [Ancient Greek](#) ἀρχιτέκτων (*arkhitéktōn*) 'architect'; from ἀρχι- (*arkhi-*) 'chief', and τέκτων (*téktōn*) 'creator'. Architectural works, in the material form of buildings, are often perceived as cultural symbols and as [works of art](#). Historical civilizations are often identified with their surviving architectural achievements.^[6]

The practice, which began in the [prehistoric era](#), has been used as a way of expressing [culture](#) for civilizations on all seven [continents](#).^[7] For this reason, architecture is considered to be a form of [art](#). Texts on architecture have been written since ancient times. The earliest surviving text on [architectural theories](#) is the 1st century AD treatise *De architectura* by the Roman architect [Vitruvius](#), according to whom a good building embodies *firmitas*, *utilitas*, and *venustas* (durability, utility, and beauty). Centuries later, [Leon Battista Alberti](#) developed his ideas further, seeing beauty as an objective quality of buildings to be found in their proportions. [Giorgio Vasari](#) wrote *Lives of the Most Excellent Painters, Sculptors, and Architects* and put forward the idea of style in the Western arts in the 16th century. In the 19th century, [Louis Sullivan](#) declared that "[form follows function](#)". "Function" began to replace the classical "utility" and was understood to include not only practical but also aesthetic, psychological and cultural dimensions. The idea of [sustainable architecture](#) was introduced in the late 20th century.



In adding the dome to the [Florence Cathedral \(Italy\)](#) in  the early 15th century, the architect [Filippo Brunelleschi](#) not only transformed the building and the city, but also the role and status of the [architect](#).^{[1][2]}

Software Architecture

- How are we going to organize the parts of the software system?
 - What are the parts?
 - What is the structure that puts the parts together?
- These are the questions that sit between requirements (higher level, abstract goals) and implementation (low level, concrete artifacts).
- Individual components implement functional requirements; overall architecture has much greater impact on non-functional requirements such as performance, robustness, maintainability, etc.

Can this be taught/learnt?

- Going from a set of requirements to an architectural design involves specific domain, context, and other circumstances.
 - Where does the general principle ends and the specific domain starts?
 - “Computer science education cannot make anyone an expert programmer any more than studying brushes and pigments can make somebody an expert painter.” - Eric Raymond
- We will do our best to cover the generic now, and continue to do our best to learn from our experiences :

Questions about Architectural Design

(taken from Sommerville, 9th ed.)

- Is there a generic application architecture that can be the template?
- What will be the fundamental approach used to structure the system?
- How will the structural components in the system be decomposed into sub-components?
- What architectural organization is best for delivering the non-functional requirements of the system?
- How should the architecture be documented?
- How will the system be distributed across hardware cores and processors?

Important Viewpoints

(taken from Sommerville, 9th ed.)

- Logical View: reveals key abstractions in the system as objects or classes - are system requirements related to these objects?
- Process View: looks at how the system is composed of interacting processes - will we have sufficient performance?
- Development View: looks at the way the system can be decomposed into smaller components that can be implemented - are they feasible, and can they be handled by the development team?
- Physical View: identifies system hardware, and looks at the way software components are distributed across processors - can they be deployed and maintained?

Architecture vs. (Interior) Design

- Architecture: which house do I build?
 - number of floors, number and type of rooms, number of restrooms, location of kitchen...
- Design: how do I build each section?
 - materials for floor, color of wall, lighting fixture, furniture...

SW Architecture vs. SW Design

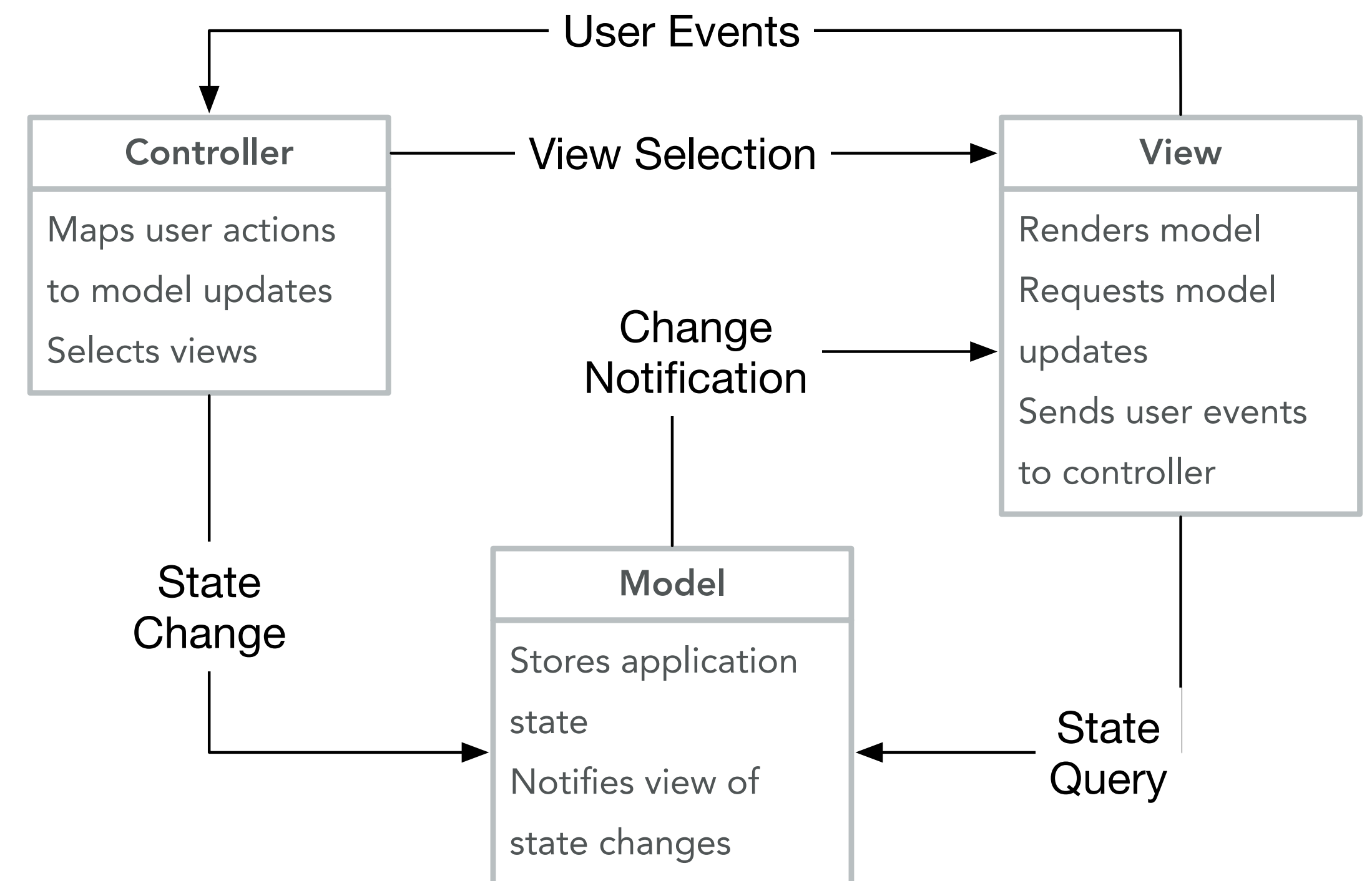
- Architecture: what is being developed?
 - types of components, how they communicate with each other, how storage is provided...
- Design: how are the components developed?
 - which classes, API design, data modeling...

Architectural Patterns

- Certain design knowledges about software architecture can be generalized, abstracted, and **reused**.
 - They are reused because they are known to solve some common design issues, such as separation of concerns.
- They are useful templates for you to use when you want to break down the system into smaller components.

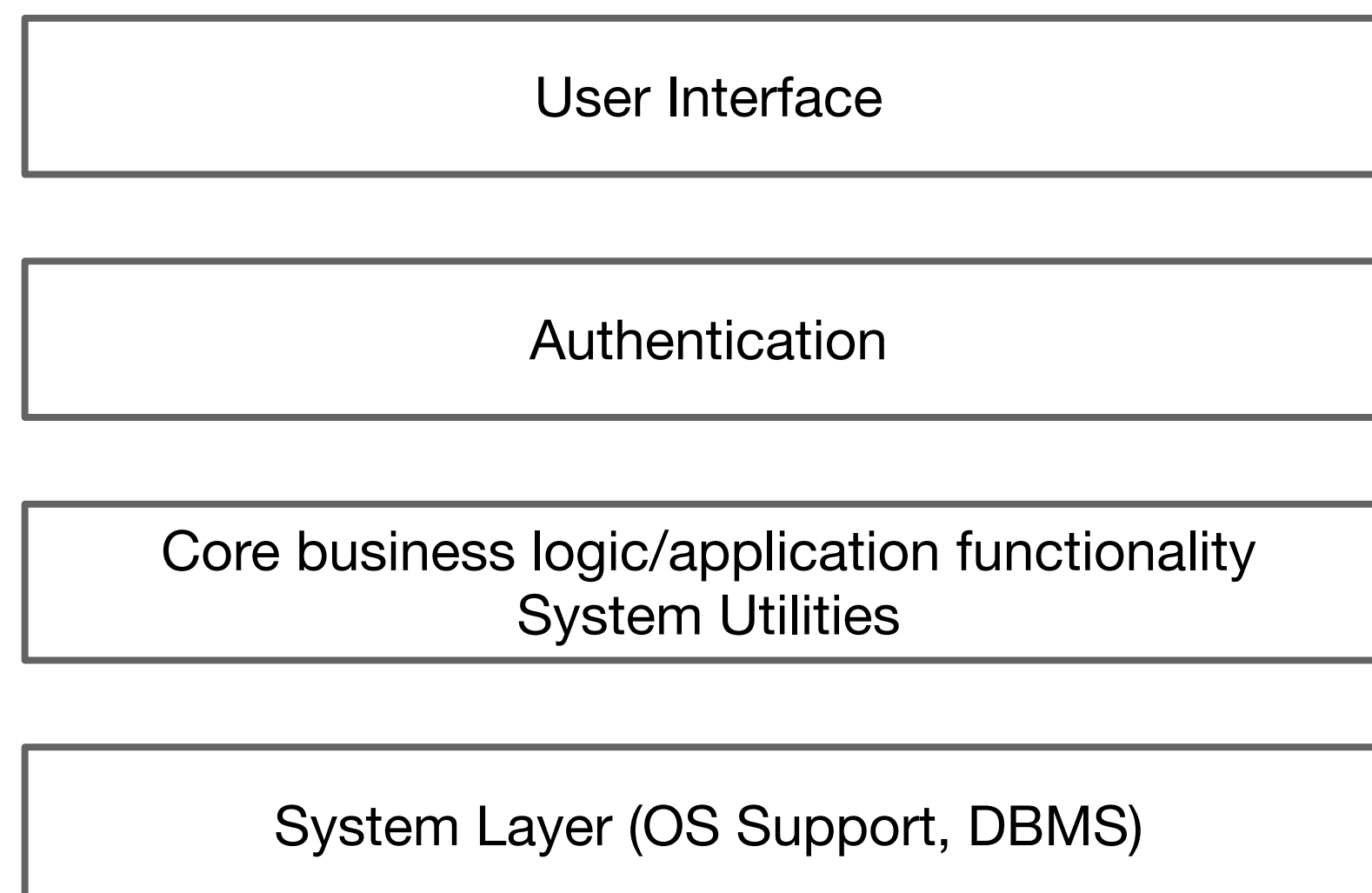
MVC (Model-View-Controller)

- When there are stored data (model), its representation (view), and user actions (controller), MVC can be used to manage the interactions between them
- Model is often a database or some other persistence layer
- View is often a GUI
- Controller is often the application/business logic



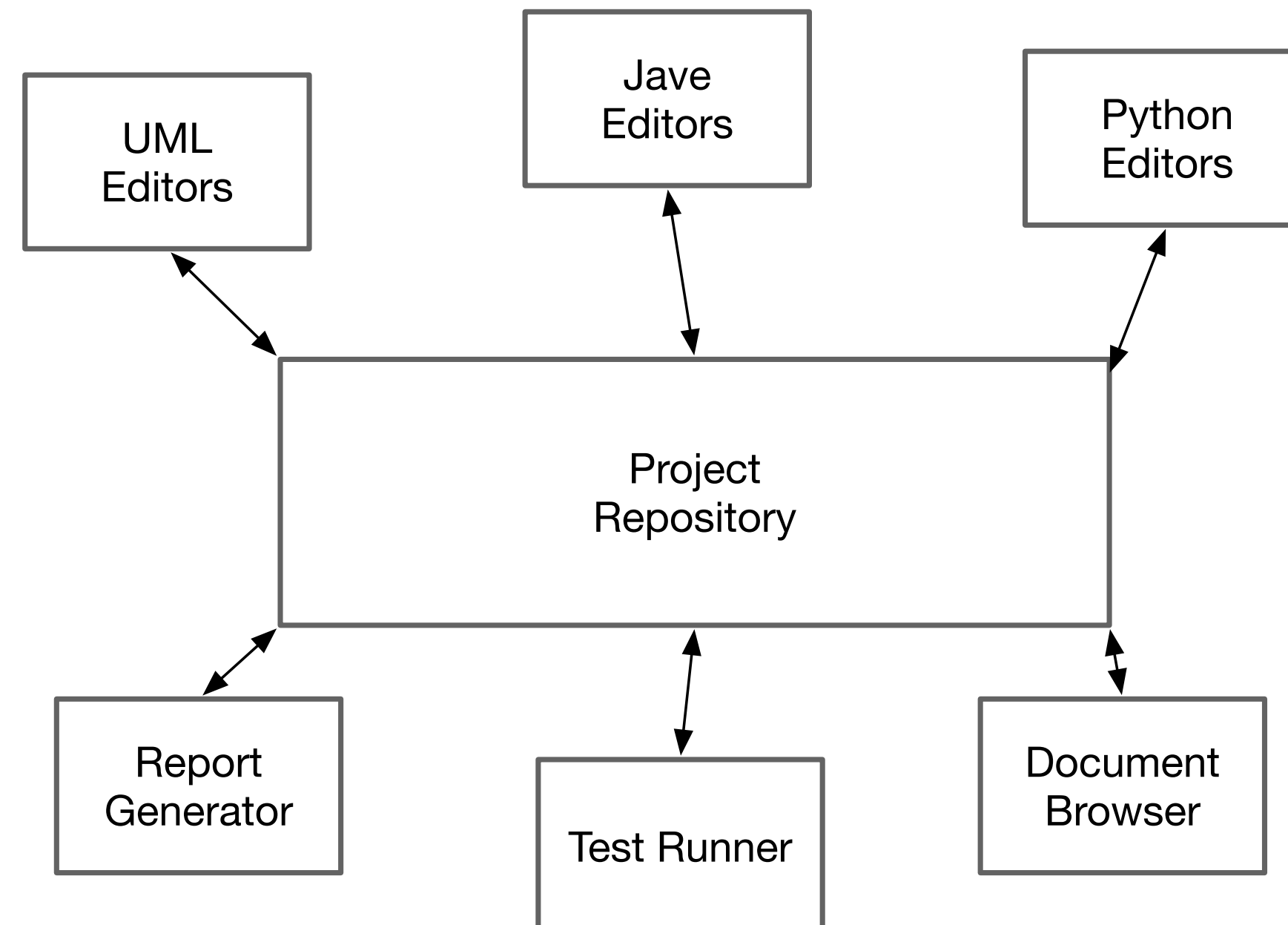
Layered Architecture

- Organise system functionality into layers, each of which only depends on the single underlying layer.
- Separation of concern is achieved with different layers.
- Each layer can be one step in incremental development.



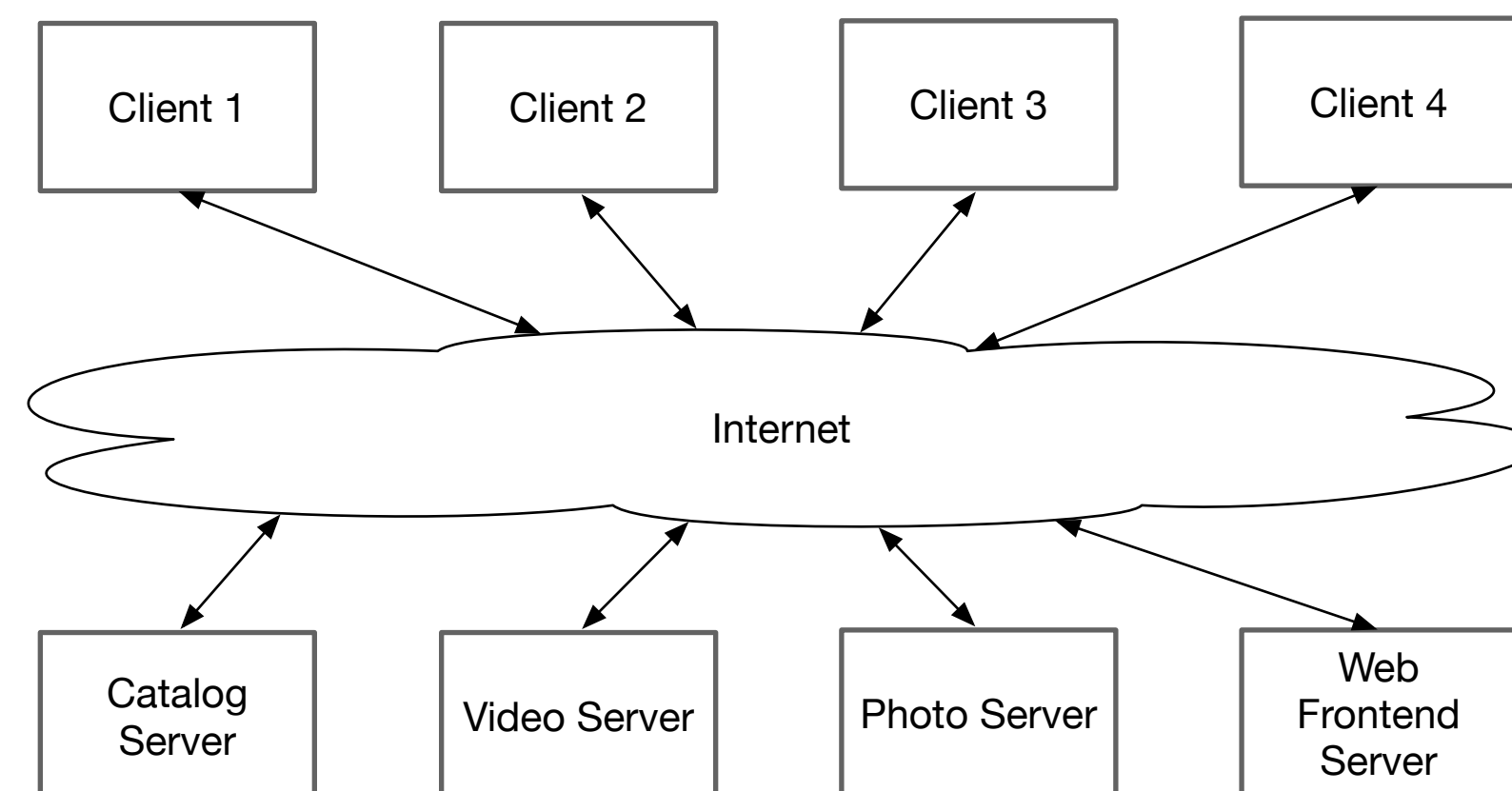
Repository Architecture

- When there is a single repository of data, on which various other components depend
- Components only communicate with each other using the repository (i.e., by changing the state that is recorded in the repository)



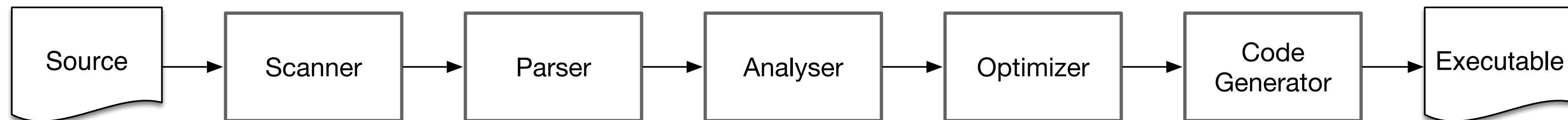
Client/Server Architecture

- A distributed architecture that describes a specific runtime organization
 - A set of servers, providing services to other components;
 - A set of clients, using the services offered by the servers;
 - A network infrastructure, connecting servers and clients



Pipe Architecture

- Entire system can be broken down different transformations being applied to the incoming input (remember Unix pipeline example earlier?).
- Ideal to scenarios with little user interaction.



Software Product Line Architecture

- Software feature set
- A specific
- But it can

The screenshot shows the GitHub pricing page. At the top, there's a navigation bar with the GitHub logo, a search bar, and links for Pull requests, Issues, Codespaces, Marketplace, and Explore. The main heading is "Get the complete developer platform." Below this, there's a question "How often do you want to pay?" with two options: "Monthly" and "Yearly" (which is selected and has a "Get 1 month free" offer). The pricing is shown in three columns: "Free", "Team", and "Enterprise". The "Team" plan is highlighted as "MOST POPULAR".

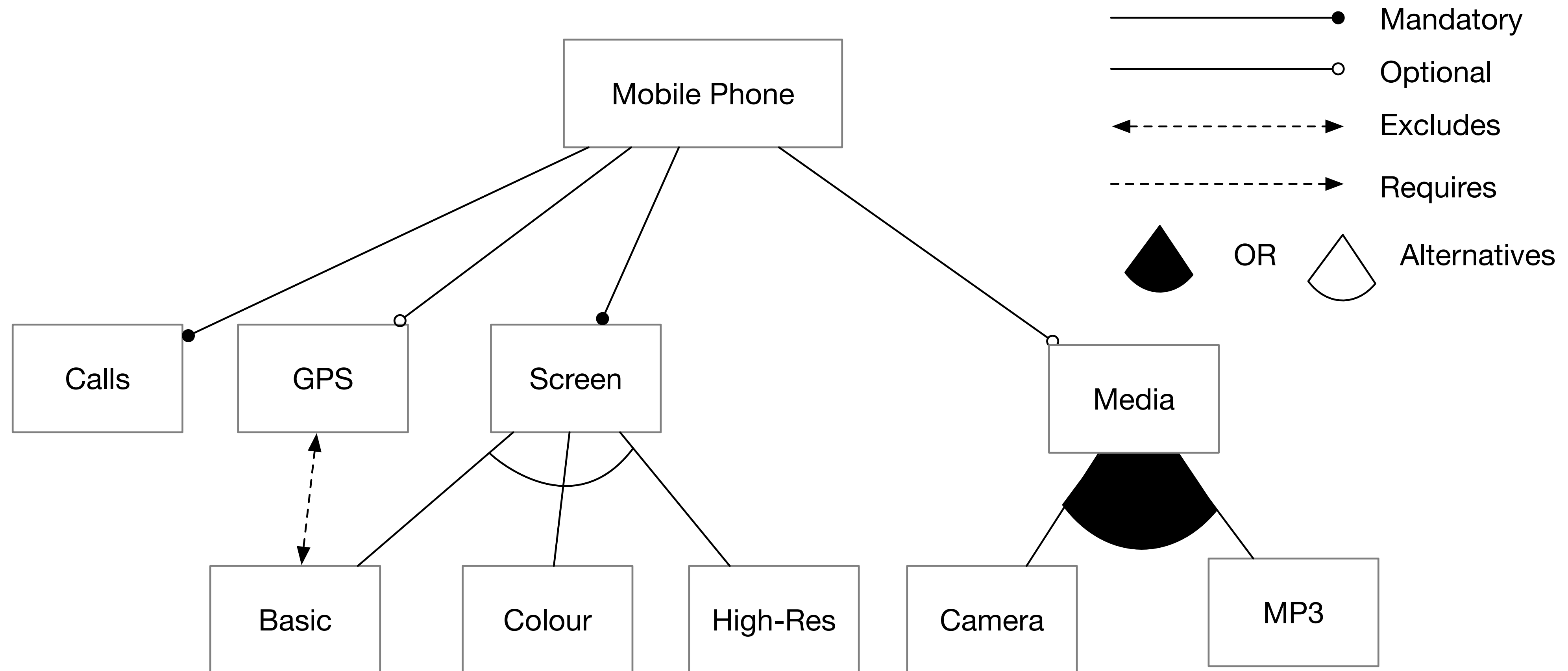
Plan	Price	Key Features
Free	\$0 per month forever	Unlimited public/private repositories, Automatic security and version updates, 2,000 CI/CD minutes/month, 500MB of Packages storage, 120 core-hours Codespaces compute/month, 15GB of Codespaces storage, New issues & projects (in limited beta), Community support.
Team	\$3.67 per user/month for the first 12 months*	Everything included in Free, plus... Access to GitHub Codespaces, Protected branches, Multiple reviewers in pull requests, Draft pull requests, Code owners, Required reviewers, Pages and Wikis, Environment deployment branches and secrets, 3,000 CI/CD minutes/month, 2GB of Packages storage.
Enterprise	\$19.25 per user/month for the first 12 months*	Everything included in Team, plus... Enterprise Managed Users, User provisioning through SCIM, Enterprise Account to centrally manage multiple organizations, Environment protection rules, Audit Log API, SOC1, SOC2, type 2 reports annually, FedRAMP Tailored Authority to Operate (ATO), SAML single sign-on, Advanced auditing, GitHub Connect.

imon

Feature Model

- Feature Models (FMs) are systematic models that capture the commonalities as well as variabilities in SPL Architectures.
 - Features are either mandatory or optional
 - Choices are either alternatives (1 out of N) or ORs (M out of N, $M \leq N$)
 - Additional constraints between features (e.g., camera features in a mobile phone requires high resolution screen)

Feature Model Diagram: An Example



Design vs. Architecture

- Architecture defines the big picture of what you want to build.
- Design concerns how you implement the big picture in more detail.
- Think of building a house (=architecture) vs. interior design (=design)
 - Architectural decisions: how to lay the foundation, which orientation, the overall shape of the house, which room goes where, etc
 - Design decisions: how to make each room functional, in which style, etc
- They do bleed into each other in the middle.

Design Patterns

- Commonly & frequently (re-)used patterns of code: think of architectural patterns, but much more detailed.
- Knowing these patterns may allow you to easily break down a big picture into components you need to implement.
- We will cover some of the popular design patterns in the next lecture.