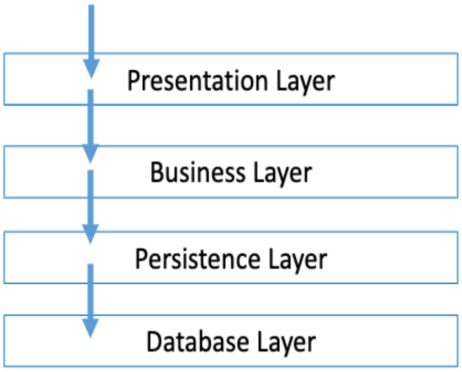
< 아키텍처 스타일 패턴 >

1. 레이어 패턴 (Layered pattern)

가장 일반적으로 사용하는 패턴으로, **시스템을 계층화하여 구성 (n-tier 패턴)**
설명 : 하위 레이어가 제공하는 기능을 상위 레이어가 이용함으로써 각 레이어를 단순화/ 계층간의 관계는 사용가능의 관계로 표현됨
핵심 : 각 레이어는 해당 레이어가 의존하는 직접적인 하위 레이어만 알면 된다.
장점 : 모듈의 재사용성 높여 유지보수성 & (정보은닉의 원칙 적용) -> 이식성에 좋은 패턴으로, 각 레이어가 책임을 다하도록 설계만 하면 시스템 전체도 문제 없이 작동 가능하다.
단점: 추가적인 실행 시 오버헤드 (너무 많은 계층으로 성능 감소 발생)
활용: 일반적인 데스크톱 애플리케이션, E-Commerce 웹 애플리케이션



일반적인 정보 시스템에서 공통적으로 볼 수 있는 계층 4가지는 다음과 같다.

- **프레젠테이션 계층** (Presentation layer) - **UI 계층** (UI layer) 이라고도 함
- **애플리케이션 계층** (Application layer) - **서비스 계층** (Service layer) 이라고도 함
- **비즈니스 논리 계층** (Business logic layer) - **도메인 계층** (Domain layer) 이라고도 함
- **데이터 접근 계층** (Data access layer) - **영속 계층** (Persistence layer) 이라고도 함

1-1) 웹 애플리케이션에서의 레이어 분리

일반적인 웹 애플리케이션에 적용한 레이어 패턴: 프리젠테이션, 비즈니스, 데이터 액세스 레이어

- 프리젠테이션 레이어: 시스템 사용자와의 인터페이스를 담당하는 레이어로, 웹 브라우저를 통해 사용자의 입력을 받아 하위 레이어인 비즈니스 로직 레이어에 전달하고, 처리 결과를 다시 웹 브라우저에 표시하거나 화면 이동을 제어하는 일을 담당

- 비지니스 레이어: 애플리케이션에서 구현해야 할 고유의 작업을 처리하기 위한 레이어 업무 로직으로 번역

- 데이터 액세스 레이어: 비즈니스 로직 레이어와 데이터베이스를 중개하기 위한 레이어로, 데이터베이스에 연결하려면 긴 코드를 작성해야 하는데, 이와 같은 귀찮은 처리를 비즈니스 로직에서 분리해 데이터베이스 연결 절차를 일일이 의식하지 않아도 이용할 수 있게 하는 것이 데이터 액세스 레이어의 역할을 한다. (통합 레이어라고도 칭함)

*** 프레젠테이션/ 데이터 각 계층은 반드시 미들웨어인 비지니스 계층을 통해서만 통신이 가능하다**

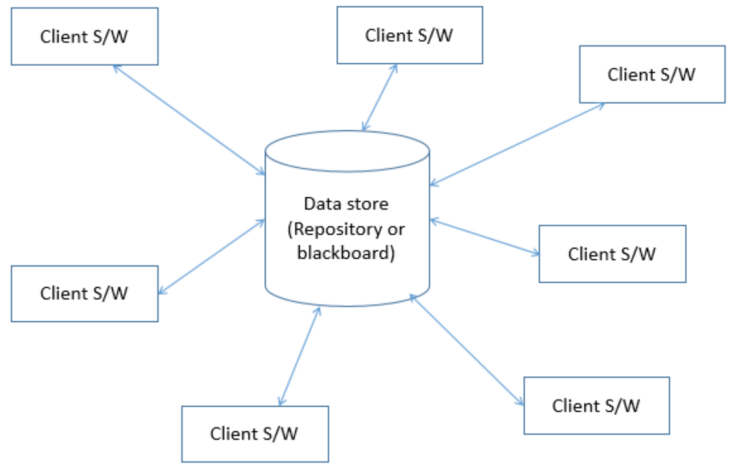
1-2) 웹 애플리케이션의 레이어와 MVC 패턴이 상반되는 것이 X

컨트롤러와 뷰는 그대로 프리젠테이션, 모델은 비즈니스 로직 레이어 & 데이터 액세스 레이어로 나뉨

웹 애플리케이션에서 빈번하게 발생하는 데이터베이스 접속 처리를 데이터 액세스 레이어로 분리함으로써 비즈니스 로직 데이터와 데이터 액세스 레이어라는 각 계층의 개발을 좀 더 간단히 할 수 있다.

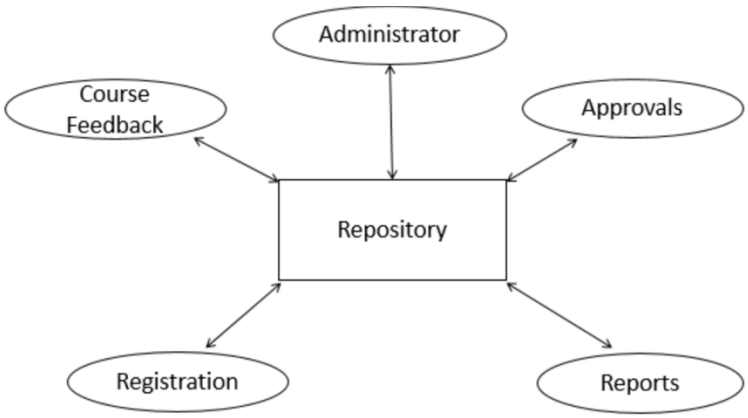
2. Data Centered Architecture

목표 : 데이터의 통합성을 달성하는 것
설명: 데이터를 수정하는 다른 구성요소에 의해 데이터가 중앙 집중화되고 액세스 되는 아키텍처 스타일.
핵심: 데이터 접근자 간의 상호 작용 또는 통신은 데이터 저장소를 통해서만 이루어진다. 데이터는 클라이언트 간의 유일한 통신 수단
구성 요소:
- Data Store(Repository) - 영구 데이터 저장소를 제공, 현재 상태를 나타냄
- Data Accessor (데이터 접근자) - 중앙 데이터 저장소에서 작동하고 계산을 수행하며 결과를 되돌릴 수 있음



* 제어 흐름은 아키텍처를 두가지 범주로 구분

a.Repository 아키텍처 스타일



: Repository 에 저장되는 데이터를 중심으로 Client 들이 데이터를 교환하는 구조

데이터 저장소는 수동적이며 데이터 저장소의 클라이언트는 능동적으로 논리 흐름을 제어하고, 참여하는 구성 요소는 데이터 저장소에서 변경 사항을 확인한다.

DBMS, 라이브러리 정보 시스템, CORBA 의 인터페이스 저장소, 컴파일러 및 CASE (컴퓨터 지원 소프트웨어 엔지니어링) 환경에서 널리 사용

장점: 대량 데이터 저장 / 데이터 무결성 , 백업 및 복원 기능 제공 / 소프트웨어 구성요소 간 일시적 오버헤드 줄임
단점: 데이터 구조의 변경이 클라이언트에게 큰 영향을 끼침/ 데이터 분산의 어려움과 비싼 비용

b.Blackboard 아키텍처 스타일

데이터 저장소는 능동적(활성화)이며 데이터 저장소의 클라이언트는 수동적이다. 따라서, 논리적 흐름은 데이터 저장소의 현재 데이터 상태에 의해 결정된다.

구성요소:

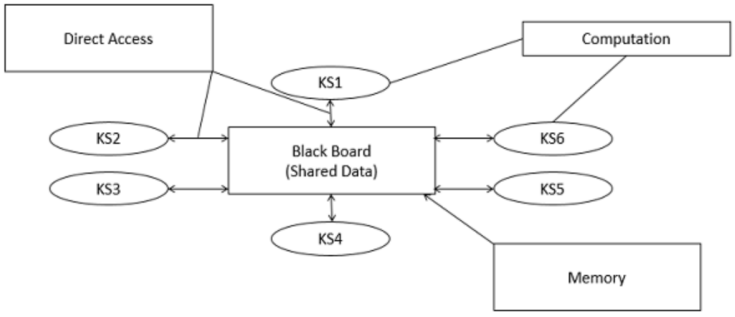
블랙보드 (blackboard) — 솔루션의 객체를 포함하는 구조화된 전역 메모리

지식 소스 (knowledge source) — 자체 표현을 가진 특수 모듈

제어 컴포넌트 (control component) — 모듈 선택, 설정 및 실행을 담당한다.

설명: 모든 컴포넌트는 블랙보드에 접근한다. 컴포넌트는 블랙보드에 추가되는 새로운 데이터 객체를 생성할 수 있다. 컴포넌트는 블랙보드에서 특정 종류의 데이터를 찾으며, 기존의 지식 소스와의 패턴 매칭으로 데이터를 찾는다.

활용: 음성 인식, 차량 식별 및 추적,단백질 구조 식별 ,수중 음파 탐지기 신호 해석 (이 패턴은 결정 가능한 해결 전략이 알려지지 않은 문제에 유용하다.) -> 알파고와 같은 AI 의 복잡한 문제를 해결할 때 사용



3. Data flow Architecture

입력 데이터는 일련의 계산 또는 조작 컴포넌트에 의해 출력 데이터로 변환

- Pipe-Filters 패턴

설명: 데이터 스트림 처리 시스템을 위한 구조

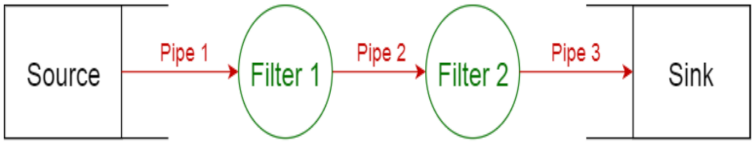
핵심: 컴포넌트- 필터, 커넥터 -파이프

필터는 파이프를 통해 받은 데이터를 변경시켜서 그 결과를 파이프로 전송한다.

장점: 필터 교환과 재조합을 통해 높은 유연성(가변성)을 제공한다. / 간단하고 정의된 인터페이스로 복잡한 통합 문제 줄임

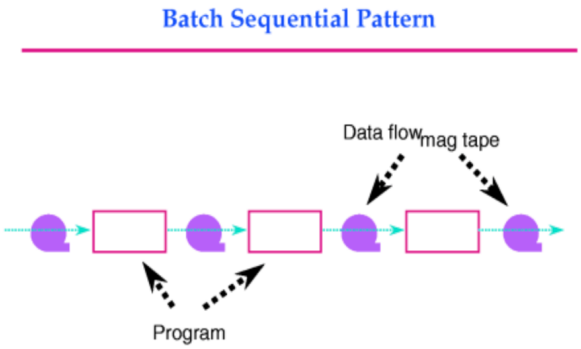
단점: 상태 정보 공유를 위한 비용이 많이 들고 데이터 변환 오버헤드 발생

시스템 상호 작용에 대한 매우 동적인 응답 제공 못함



- **Batch and sequential 패턴**

설명: 여러 개의 서브시스템이 순차적으로 데이터를 처리하는 구조로 서브시스템이 독립적으로 구성된다.
장점: 필터 교환과 재조합을 통해 높은 유연성을 제공한다.
단점: 대기시간이 길고, 처리량이 떨어지며 동시 처리가 어렵다.
활용: 은행이나 Billing System



4. Object-Oriented Architecture

설명: 시스템 컴포넌트는 데이터 및 연산을 캡슐화, 컴포넌트 사이의 의사소통은 메세지 전달을 통해서 한다.
시스템을 협력 객체로 생각한다.

*객체 지향 디자인의 5 가지 컨셉

1. 캡슐화 - 객체에서 클래스의 데이터를 숨기는 것/ 추상화 요소를 바인딩 (외부에 영향 없이 객체 내부 구현 변경 가능)
2. 데이터 보호
3. 상속 - 상위 클래스의 기능을 재사용, 확장하는 방법
4. 인터페이스 - 객체간의 상호작용 요청 목록, "인터페이스를 구현하는 객체는 이런 책임들을 진다"는 목록
5. 다형성 - 여러 형태를 갖는 것 / 한 객체가 여러 타입을 갖는 것 = 한 객체가 여러 타입의 기능을 제공

객체 지향 설계
*** Solid 원칙 : https://gmlwjd9405.github.io/2018/07/05/oop-solid.html 참고**

SRP (Single Responsibility Principle) 단일 책임 원칙

OCP (Open Closed Principle) 개방 폐쇄 원칙

LSP (Liskov Substitution Principle) 리스코프 치환 원칙

ISP (Interface Segregation Principle) 인터페이스 분리 원칙

DIP (Dependency Inversion Principle) 의존 역전 원칙

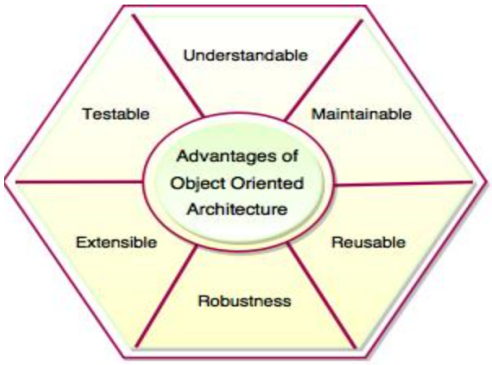


Fig. Advantages of Object Oriented Architecture

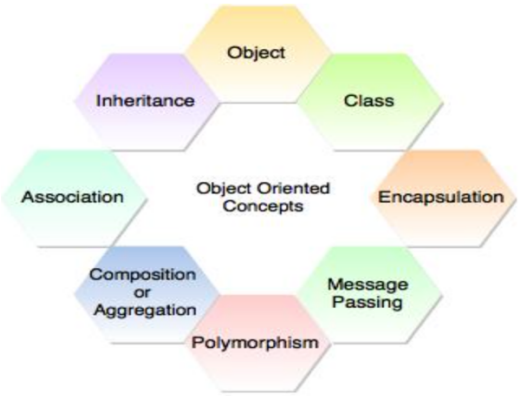


Fig. Object Oriented Concepts

객체 지향 분석

1. **객체 모델링** - 객체와 관련된 시스템의 정적 구조를 개발 / 개체 , 클래스 간의 관계 인식
2. **동적 모델링** - 단일 개체가 이벤트에 응답하는 방식을 설명 (활동은 특정 시점에서 발생)
3. **기능적 모델링** - 객체에서 실행되는 프로세스와 메소드간에 데이터가 이동할 때 데이터가 어떻게 변경되는지를 보여준다.

OO 디자인은 OO 분석 후 다음 단계.
여기서 개념적 모델은 OO 디자인을 사용하여 OO 모델로 더욱 발전

객체 지향 설계

목적 : 시스템의 구조적 아키텍처를 설정하는 것

1) **개념 설계 -** 시스템 구축에 필요한 모든 클래스를 인식, 개별 책임은 각 클래스에 할당 / 고수준 설계에서는 클래스 다이어그램을 사용하여 클래스 간의 관계를 분석하고 상호 작용 다이어그램을 사용하여 이벤트의 흐름을 보여준다.

2) **세부 설계 -**속성과 작업은 상호 작용 다이어그램을 기반으로 각 클래스에 할당 / 디자인의 다음 세부 사항을 설명하기 위해 상태 머신 다이어그램이 개발되어 저수준 디자인이라고 한다.

5. Distributed Architecture

- 클라이언트- 서버 패턴 (Client-Server pattern)

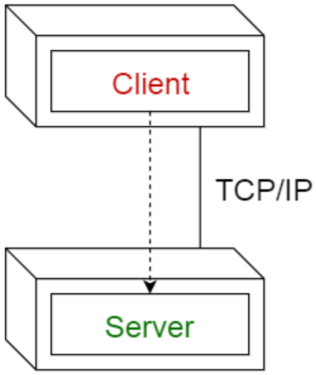
설명 : **하나의 서버 컴포넌트 + 다수의 클라이언트 컴포넌트**로 구성

: 서버 컴포넌트는 다수의 클라이언트 컴포넌트로 서비스를 제공, 클라이언트가 서버에 서비스를 요청하면 서버는 클라이언트에게 적절한 서비스를 제공 (서버는 계속 클라이언트로부터의 요청을 대기

장점 : 직접적으로 데이터 분산, 위치 투명성 제공한다. 클라이언트가 요청할 수 있는 일련의 서비스를 모델링 가능하다.

단점: 요청은 일반적으로 서버에서 별도의 스레드로 처리된다. 프로세스간 통신은 서로 다른 클라이언트가 서로 다르게 표현되므로 오버헤드가 발생한다. 서비스와 서버의 이름을 관리하는 레지스터 부재로 이용 가능한 서비스 탐색이 불편하다.

활용: 이메일, 문서 공유 및 은행 등의 온라인 애플리케이션



- Multi-tiers

설명: SoC(system on a chip : 단일 칩 시스템)에 따라서 여러개의 Tier 로 구성하는 구조

장점:비즈니스 로직, 인터페이스, 데이터 베이스 등의 분리. 확장, 재사용성이 뛰어나다.

단점:테스트가 어렵고, 네트워크 사용에 따른 비용 발생한다

5. Interaction- Oriented - (MVC)

설명 : 사용자 인터페이스로부터 비즈니스 로직을 분리하여 애플리케이션의 시각적 요소나 그 이면에서 실행되는 비즈니스 로직을 서로 영향 없이 쉽게 고칠 수 있는 애플리케이션을 만들 수 있다.

장점: 동일한 모델, 다양한 뷰 제공 및 뷰 동기화로 실시간 변경 데이터 제공
서로 분리되어 각자의 역할에 집중할 수 있게끔하여 개발을 하고 그렇게 애플리케이션을 만든다면, 유지보수성, 애플리케이션의 확장성, 그리고 유연성이 증가하고, 중복코딩이라는 문제점 또한 사라지게 된다.

단점: 단순 어플리케이션은 복잡성 증가

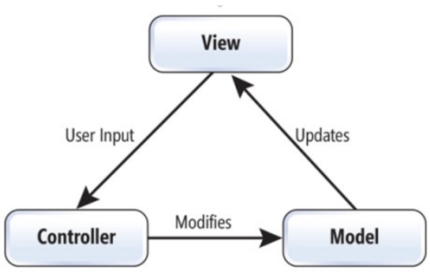
구성 요소 (3 개) :

1)Model

: 모델(model)이란 어떠한 동작을 수행하는 코드를 말한다. 표시 형식에 의존하지 않는 데이터 자체.

2)View
: 사용자에게 실제로 보여지는 부분, 요청에 따른 처리 결과를 받아서 보여주는 영역

3)Controller
: 컨트롤러는 사용자의 액션에 응답하는 컴포넌트, 클라이언트로 부터 받은 요청을 Model 에 전달하고 로직에 따라 실행한 결과를 View 에 전달하는 중재자 역할



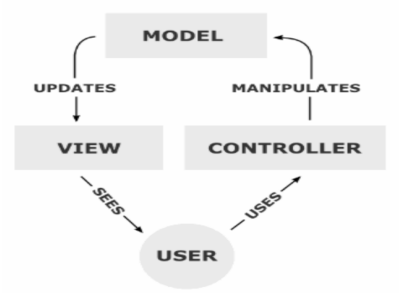
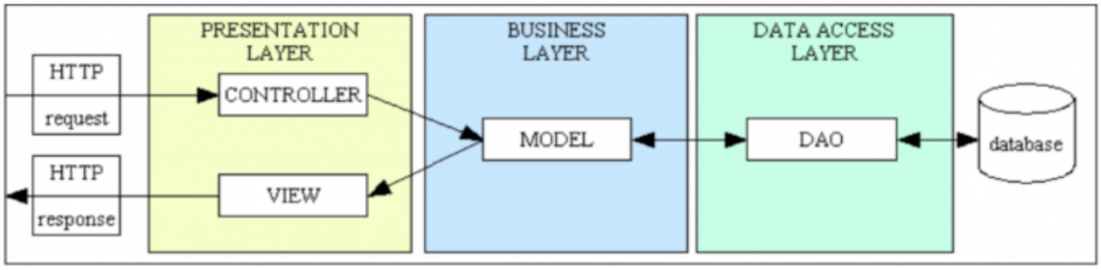


Figure 8a - MVC plus 3 Tier Architecture



+ 부가적으로

객체지향 VS 절차지향

객체지향

- 데이터와 프로시저를 알맞게 묶음
- 설계에 많은 시간이 투자된다.
- 유지보수 및 업그레이드가 쉽다.

절차지향

- 데이터가 여로 프로시저를 이용
- 객체지향에 비해 쉽다.
- 이러한 데이터를 공유하는 방식은 변화가 필요할 때 수정을 복잡하게 만든ㄴ다.

https://ejrtmtm2.files.wordpress.com/2012/09/ed9484eba19ceca09ded8ab8-eab480eba6ac14.pdf

컴포넌트 설계 요약된 pdf