

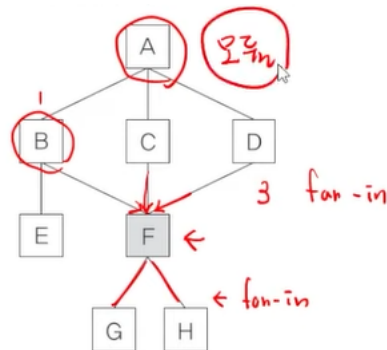
# 소프트웨어 설계 - 2

## 11.소프트웨어 설계 모델링

- 소프트웨어 설계 분류
  - 상위설계 (뼈대 만들기): 아키텍처 설계 - 데이터 설계 - 인터페이스 정의 - 사용자 인터페이스 설계
  - 하위 설계(구체화): 모듈 설계 - 자료 구조 설계 - 알고리즘 설계
- 설계 대상
  - 구조 모델링: 모듈, 컴포넌트, 구조를 모델링하는것
  - 행위 모델링: 기능, 동작 등을 모델링
- 소프트웨어 구조도

Fan-in	주어진 한 모듈을 제어하는 상위 모듈 수
Fan-out	주어진 한 모듈이 제어하는 하위 모듈 수
Depth	최상위 모듈에서 주어진 모듈까지의 깊이
Width	같은 등급(level)의 모듈 수
Super ordinate	다른 모듈을 제어하는 모듈
Subordinate	어떤 모듈에 의해 제어되는 모듈

- 예



- Fan-in/Fan-out을 분석하면 시스템 복잡도 파악이 가능하다.

Fan-in이 높은 경우	<ul style="list-style-type: none"> <li>• 재사용 측면에서 잘된 설계로 볼 수 있다.</li> <li>• 시스템 구성 요소 중 일부가 동작하지 않으면 시스템이 중단되는 단일 장애 발생 가능성이 있다.</li> <li>• 단일 장애 발생을 방지하기 위해 중점 관리가 필요하다.</li> </ul>
Fan-out이 높은 경우	<ul style="list-style-type: none"> <li>• 불필요한 타 모듈의 호출 여부를 확인한다.</li> <li>• Fan-out을 단순하게 설계할 수 있는지 검토한다.</li> </ul>

- 복잡도 최적화를 위한 조건: Fan-in은 높이고 Fan-out은 낮추도록 설계한다.

- 코드 설계(자료를 분류하기 위한 기준)

목적	특성
✓고유성	코드는 그 뜻이 1:1로 확실히 대응할 수 있어야 한다.
✓분류 편리성	목적에 적합한 분류가 가능해야 한다.
배열의 효율성	바람직한 배열을 얻을 수 있어야 한다.
간결성	짧고 간결 명료해야 한다.
유지보수 편리성	유지 관리가 쉬워야 한다.
코드의 독립성	다른 코드 체계와 중복되지 않아야 한다.
코드의 편의성	이해가 쉽고, 사용하는데 편리해야 한다.
추가 · 삭제 편리성	추가와 삭제가 편리해야 한다.

#### ◆ 순차 코드(Sequence Code) 20.6

- 코드화 대상 항목을 어떤 일정한 배열로 일련번호를 배당하는 코드로 확장성이 좋으며, 단순해서 이해하기 쉽고, 기억하기 쉽다.
- 항목 수가 적고, 변경이 적은 자료에 적합하며, 일정 순서대로 코드를 할당하므로 기억 공간 낭비가 적다.
- 누락된 번호를 삽입하기 어렵고 명확한 분류 기준이 없어 코드에 따라 분류가 어려워 융통성이 낮다.

1  
2  
3  
4  
A  
B  
C

#### • 구조적 개발 방법론

- 구조적 분석 : 자료의 흐름(자료흐름도, 자료 사전, 소단위 명세서 사용)
- 자료흐름도(Data Flow Diagram) == blue chart

#### ◆ 데이터(자료) 흐름도

구성 요소	의미	표기법
프로세스 (Process)	자료를 변환시키는 시스템의 한 부분을 나타낸다.	프로세스 이름
자료 흐름 (Data Flow)	자료의 이동(흐름)을 나타낸다.	자료 이름 →
자료 저장소 (Data Store)	시스템에서의 자료 저장소(파일, 데이터베이스)를 나타낸다.	자료 저장소 이름
단말 (Terminator)	<ul style="list-style-type: none"> <li>• 자료의 발생지와 종착지를 나타낸다.</li> <li>• 시스템의 외부에 존재하는 사람이나 조직체이다.</li> </ul>	단말 이름

#### • 표기법

기호	의미	설명
$=$	자료의 정의	~로 구성되어 있다(is compose of).
$A + B$	자료의 연결	그리고(and, along with)
$(N)$	자료의 생략	생략 가능한 자료(optional)
$A \text{ or } B$	자료의 선택	다중 택일(selection), 또는(or)
$\{ \}$	자료의 반복 (iteration of)	$\{ \}_n$ : 최소 n번 이상 반복 $\{ \}^n$ : 최대 n번 이하 반복 $\{ \}_m^n$ : m번 이상 n번 이하 반복
$**$	자료의 설명	주석(comment)

## 12. 모듈

- 재사용 가능
- 모듈의 독립성은 결합도와 응집도로 측정됨
- 서브루틴 = 서브 시스템 = 작업단위
- 상속 가능
- 결합도(Coupling)
  - 결합도는 낮을수록 독립성이 향상되어 유지보수에 이점이 있음

결합도 약함 ↑	자료 결합도 (Data Coupling)	<ul style="list-style-type: none"> <li>• 모듈 간의 인터페이스가 자료 요소로만 구성된 경우로 다른 모듈에 영향을 주지 않는 가장 바람직한 결합도이다.</li> <li>• 모듈 간의 내용을 전혀 알 필요가 없다.</li> </ul>
	스탬프 결합도 Stamp Coupling)	<ul style="list-style-type: none"> <li>• 두 모듈이 같은 자료 구조를 조회하는 경우의 결합도이며, 자료 구조의 어떠한 변화 즉 포맷이나 구조의 변화는 그것을 조회하는 모든 모듈 및 변화되는 필드를 실제로 조회하지 않는 모듈까지도 영향을 미치게 된다.</li> <li>• 배열, 레코드, 구조 등이 모듈 간 인터페이스로 전달되는 경우와 관계된다.</li> </ul>

	제어 결합도 (Control Coupling)	어떤 모듈이 다른 모듈의 내부 논리 조작을 제어하기 위한 목적으로 제어 신호를 이용하여 통신하는 경우이며, 하위 모듈에서 상위 모듈로 제어 신호가 이동하여 상위 모듈에 처리 명령을 부여하는 권리 전도 현상이 발생하게 된다.
	외부 결합도 (External Coupling)	어떤 모듈에서 외부로 선언한 변수(데이터)를 다른 모듈에서 참조할 경우와 관계된다.
	공통 결합도 (Common Coupling)	여러 모듈이 공통 자료 영역을 사용하는 경우로 공통 데이터 영역 내용을 수정하면 이 데이터를 사용하는 모든 모듈에 영향을 준다.

결합도 강함 ↓	내용 결합도 (Content Coupling)	<ul style="list-style-type: none"> <li>• 가장 강한 결합도를 가지고 있으며, 한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 조회하도록 설계되었을 경우와 관계된다.</li> <li>• 한 모듈에서 다른 모듈의 내부로 제어 또는 이동된다.</li> <li>• 한 모듈이 다른 모듈 내부 자료의 조회 또는 변경이 가능하다.</li> <li>• 두 모듈이 같은 문자(Literals)의 공유가 가능하다.</li> </ul>
		모듈 관계 ↑

- 응집도(Cohension)
  - 요소간의 관련성이 적음 = 응집도가 낮음

<div>↑</div> <div>응집도 강함</div> <div>↑</div>	기능적 응집도 (Functional Cohesion)	모듈 내부의 모든 <b>기능</b> 요소들이 한 문제와 연관되어 수행되는 경우와 관계된다.
	순차적 응집도 (Sequential Cohesion)	한 모듈 내부의 한 기능 요소에 의한 출력 자료가 다음 기능 요소의 입력 자료로 제공되는 경우와 관계된다.
	교환적 응집도 (Communicational Cohesion)	같은 입력과 출력을 사용하는 소 작업이 모인 경우와 관계된다.
	절차적 응집도 (Procedural Cohesion)	모듈이 다수의 관련 기능을 가질 때 모듈 내부의 기능 요소들이 그 기능을 순차적으로 수행할 경우와 관계된다.

<div>↓</div> <div>응집도 약함</div> <div>↓</div>	시간적 응집도 (Temporal Cohesion)	특정 시간에 처리되는 여러 기능을 모아 한 개의 모듈로 작성할 경우와 관계된다.
	논리적 응집도 (Logical Cohesion)	유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들로 하나의 모듈이 형성되는 경우와 관계된다.
	우연적 응집도 (Coincidental Cohesion)	모듈 내부의 각 기능 요소들이 서로 관련이 <b>없는</b> 요소로만 구성된 경우와 관계된다.

- 응집도는 강하게, 결합도는 약하게 하는게 좋음
- 복잡도와 중복은 줄이기
- 상식적인 선에서 생각하면 됨
- 모듈 vs 컴포넌트
  - 컴포넌트는 인터페이스로 연결됨
  - 모듈은 단위마다 실행가능하지만 특정 기능(결제 등)을 완벽히 수행하지 못함

#### ◆ 재사용 규모에 따른 구분 20.9

- 함수와 객체 : 클래스, 메소드 단위로 소스 코드를 재사용한다.
- 애플리케이션 : 공통 업무를 처리할 수 있도록 구현된 애플리케이션을 공유하여 재사용한다.
- 컴포넌트 : 컴포넌트 자체 수정 없이 인터페이스를 통하여 컴포넌트 단위로 재사용한다.

## 13 소프트웨어 아키텍처

- 뼈대 만들기
- 시스템 품질 속성 7가지 : 성능, 사용 운용성, 보안성, 시험 용이성, 가용성, 변경 용이성, 사용성
- 소프트웨어 아키텍처 특징 : 간략성, 추상화, 가시성, 복잡도 관리 종류(과정, 데이터, 제어 추상화가 있음)
- 아키텍처 프레임워크 요소
  - 프레임워크 기본 구조를 제공하는 거

요소	설명
Architecture Description(AD)	<ul style="list-style-type: none"> <li>• 아키텍처를 기록하기 위한 산출물이다.</li> <li>• 하나의 AD는 System의 하나 이상의 View로 구성한다.</li> </ul>
이해관계자 (Stakeholder)	소프트웨어 시스템 개발에 관련된 모든 사람과 조직을 의미하며, 고객, 개발자, 프로젝트 관리자 등을 포함한다.
관심사 (Concerns)	<p>같은 시스템에 대해 서로 다른 이해관계자의 의견이다.</p> <p>☞ 사용자 입장 : 기본 기능 + 신뢰성/보안성 요구</p>
관점 (Viewpoint)	서로 다른 역할이나 책임으로 시스템이나 산출물에 대한 서로 다른 관점이다.
관점 View	View: 이해관계자들과 이들이 가지는 생각이나 견해로부터 전체 시스템을 표현(4 + 1 View Model)

- 소프트웨어 아키텍처 4+1 View Model
  - Logical View(분석 및 설계)
  - Implementation View(프로그래머)
  - Process View(시스템 통합자)
  - Deployment View(시스템 엔지니어)
  - Use case View(사용자)
- 소프트웨어 아키텍처 설계 원리
  - 단순성
  - 효율성
  - 분할, 계층화
  - 추상화 \*\*
  - 모듈화 \*\*
- 소프트웨어 평가 방법론
  - SAAM
  - ATAM
  - CBAM
  - ARID
  - 뭐 있는지 정도만 보고가기

## 소프트웨어 아키텍처 패턴

- 아키텍처 패턴

종류	Layered, Client-Server, Master-Slave, Pipe-Filter, Broker, Peer to Peer, Event-Bus, MVC(Model View Controller), Blackboard, Interpreter
장점	<p>개발 시간 단축, 고품질 소프트웨어, 안정적 개발 가능, 개발 단계 관계자 간 의사소통이 간편함, 시스템 구조 이해도가 높아 유지보수에 유리하다.</p> <p>아키텍처 패턴 = 아키텍처 스타일 = 표준 아키텍처</p>

- 계층(layered) 패턴

- MVC(Model View Controller) 패턴
  - Model : 핵심기능 + 데이터
  - View : 사용자에게 정보 표시
  - Controller : 입력처리
- Client - Server 패턴 : 내가 아는 그거
- 파이프 필터(Pipe Filters) 패턴 : 파이프는 데이터를 보내는 역할, 필터는 데이터를 필터링 (변경)하는 역할
  - 장점 : 필터 교환과 재조합으로 높은 유연성
  - 단점 : 상태정보 공유를 위해 비용이 소모, 데이터 변환에 과부하 가능성
  - 활용 : Compiler, Lexical Anl, Context Anl, Semantic Anl, Code gen
- Peer to Peer : 분산 컴퓨팅에 사용
- 브로커(Broker) : 컴포넌트가 컴퓨터와 사용자를 연결하는 역할
- BlackBoard : 음성인식, 신호해석 등 구체적인 해결책이 없을때
- 이벤트 버스(Event - Bus) : Event Listener, 채널 이벤트 버스, 이벤트 소스 등 4가지 주요 컴포넌트를 가짐
- 인터프리터 : SQL에서 그거

## 15. 객체지향 설계


- 구조적 프로그래밍
  - 구조적이니까 이해가 쉽고, 디버깅이 쉬움
  - GOTO(분기)문은 사용 안함
- 절차적 프로그래밍
  - 순서대로
  - 함수기반 프로그래밍
  - C, Basic 등
- 객체지향의 구성요소(내가 아는 그거)
  - Class
  - Object
  - Message : 클래스끼리 주고받는 통신
- 객체지향의 5가지 특징
  - 캡슐화 : 정보은닉의 효과, 프로그래머가 내부를 이해할 필요가 없음
  - 정보은닉 : 속성과 메서드를 숨김(단순화하기 위해), Side effect를 줄이기 위해
  - 추상화 : 필요한것만 추출해서(종류 : 기능, 제어, 자료 추상화)
  - 상속성 : 컨트롤하기가 쉬워짐
  - 다형성
- 객체지향 기법에서의 관계성
  - is member of : 연관성, 참조 및 이용관계
  - is part of : 집단화, 객체 간의 구조적 집약 관계
  - is a(상속과 유사) : 일반화, 특수화, 클래스 간의 개념적인 포함관계

- Overloading(내가 아는 그거) : 같은 클래스 내에서
- Overriding(내가 아는 그거) : 상속관계에서
- 객체지향 설계 원칙(SOLID)
  - 단일 책임의 원칙(SRP : Single Responsibility Principle)
  - 개방 - 폐쇄의 원칙(OCF : Open Closed Prin) : 확장은 개방, 수정은 폐쇄
  - 리스코프치환 원칙(LSP : Liskov Substitution Prin) : 부모 클래스의 위치에 자식클래스가 들어가도 정상작동해야함(상속관계에서 지켜야함)
  - 인터페이스 분리 원칙(ISP : Interface Segregation Prin) : 사용하지 않는 메서드랑 의존 x , 영향 x
  - 의존 역전 원칙(DIP : Dependecny Inversion Prin) : 의존관계를 맺으면 바꾸기 어렵고 잘 안바뀌는거랑 의존하기

- 객체지향 방법론

Booch	<ul style="list-style-type: none"> <li>• OOD(Object Oriented Design)</li> <li>• 설계 부분만 존재하며 문서를 강조하여 다이어그램 기반으로 개발되었다.</li> </ul>	<ul style="list-style-type: none"> <li>• 분석과 설계가 분리되지 않는다.</li> <li>• 정적 모델과 동적 모델로 표현된다.</li> </ul>
OOSE (Jacobson)	<ul style="list-style-type: none"> <li>• Object Oriented SW Engineering</li> <li>• Use Case의 한 접근 방법이다.</li> <li>• Use Case를 모든 모델의 근간으로 활용된다.</li> </ul>	<ul style="list-style-type: none"> <li>• 분석, 설계 및 구현으로 구성된다.</li> <li>• 기능적 요구사항 중심이다.</li> <li>• 시스템 변화에 유연하다.</li> </ul>

OMT (Rumbaugh)	<ul style="list-style-type: none"> <li>• Object Modeling Technology</li> <li>• 객체지향 분석, 시스템 설계, Object 설계/구현 4단계로 구성된다.</li> <li>• 객체 모델링 : 객체도를 이용하여 시스템의 정적 구조를 표현한다.</li> <li>• 동적 모델링 : 상태 도를 이용하여 객체의 제어 흐름/상호 반응을 표현한다.</li> <li>• 기능 모델링 : 자료 흐름도를 이용하여 데이터값</li> </ul>	<ul style="list-style-type: none"> <li>• 복잡한 대형 개발 프로젝트에 유용하다.</li> <li>• 기업 업무의 모델링에 있어 편리하고 사용자와 의사소통이 원활하다.</li> <li>• CASE와 연동이 충실하다.</li> </ul>
----------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------

	도를 이용하여 데이터값의 변화 과정을 표현한다.	
 Goad와 Yourdon 방법 21.3, 20.6	객체지향 분석 방법론에서 E-R 다이어그램을 사용하여 객체의 행위를 모델링한다.	객체 식별, 구조 식별, 주체 정의, 속성 및 관계 정의, 서비스 정의 등의 과정으로 구성된다.

- 협약에 의한 설계(이런게 있다 정도만 보고)
  - 선행조건
  - 결과조건
  - 불변조건

## 16. 디자인 패턴

- 아키텍처 패턴은 아파트 설계의 패턴을 정해놓은거
- 인테리어의 패턴을 정해놓은거
- 디자인 패턴의 구성요소
  - 필수요소
    - 패턴의 이름
    - 문제 및 배경
    - 해답
    - 결과
  - 추가요소
    - 알려진 사례
    - 샘플 코드
    - 원리, 정당성, 근거
- GoF(Gangs of Four) 디자인 패턴
  - 생성 패턴
    - 객체를 생성할때 고려하는 패턴

Factory Method	<ul style="list-style-type: none"> <li>• 상위 클래스에서 객체를 생성하는 인터페이스를 정의하고, 하위 클래스에서 인스턴스를 생성하도록 하는 방식이다.</li> <li>• Virtual-Constructor 패턴이라고도 한다.</li> </ul>
Singleton	<ul style="list-style-type: none"> <li>• 전역 변수를 사용하지 않고 객체를 하나만 생성하도록 한다.</li> <li>• 생성된 객체를 어디에서든지 참조할 수 있도록 하는 패턴이다.</li> </ul>
Prototype	<ul style="list-style-type: none"> <li>• prototype을 먼저 생성하고 인스턴스를 복제하여 사용하는 구조이다.</li> <li>• 일반적인 방법으로 객체를 생성한다.</li> <li>• 비용이 많이 소요 되는 경우 주로 사용한다.</li> </ul>
Builder	<ul style="list-style-type: none"> <li>작게 분리된 인스턴스를 조립하듯 조합하여 객체를 생성한다.</li> </ul>

추상팩토리 Abstraction Factory	<ul style="list-style-type: none"> <li>• 구체적인 클래스에 의존하지 않고 서로 연관되거나 의존적인 객체들의 조합을 만드는 인터페이스를 제공하는 패턴이다.</li> <li>• 관련된 서브 클래스를 그룹 지어 한 번에 교체할 수 있다.</li> </ul>
------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------

- 구조 패턴
  - 연결하기 위한 패턴
    - 구성 : Adapter, Bridge, Composite, Decorator, Facade(퍼사드), Flyweight, Proxy

Adapter 21.5	기존에 구현되어 있는 클래스에 기능 발생 시 기존 클래스를 재사용할 수 있도록 중간에서 맞춰주는 역할을 한다.
Bridge	기능 클래스 계층과 구현 클래스 계층을 연결하고, 구현 부에서 추상 계층을 분리하여 각자 독립적으로 변형할 수 있도록 해주는 패턴이다.

- 행위패턴
  - 통제 개념을 가지고 있음(구조 패턴과 다른점)



- 구성 : Chain of Responsibility(책임 연쇄), Iterator(반복자), Command(명령), Interpreter(해석자), Memento(기록), Observer(감시자), State(상태), Strategy(전략), Visitor(방문자), Template Method, Mediator(중재자)

Mediator (중재자)

- 객체 간의 통제와 지시의 역할을 하는 중재자를 두어 객체지향의 목표를 달성하게 해준다.
- Virtual-Constructor 패턴이라고도 한다.

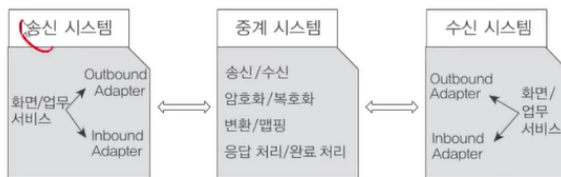
## 17. 인터페이스 요구사항 확인

- 기능적 요구사항
- 비 기능적 요구사항
- CASE(Computer Aid Software Engineering)
- 요구 사항 검토
  - 동료검토
  - 워크스루
  - 인스펙션(감사)

## 18. 인터페이스 대상 식별

- 이렇게 3개가 있다 정도

### ◆ Interface System Process



- 인터페이스 연계 기술

### ◆ 인터페이스 연계 기술

구분	설명
DB Link	<ul style="list-style-type: none"> <li>• DB에서 제공하는 DB Link 객체를 이용하는 것이다.</li> <li>• 수신 시스템의 DB에서 송신 시스템에 접근 가능한 DB Link를 생성한 뒤 송신 시스템에서 DB Link로 직접 참조하여 연계하는 것이다.</li> </ul>
DB Connection	수신 시스템 WAS에서 송신 시스템으로 연결되는 DB Connection Pool을 생성하고 프로그램 소스에서 이를 사용하는 것이다.
API/Open API	송신 시스템의 DB에서 데이터를 읽어 제공하는 Application Programming Interface를 이용하는 것이다.
JDBC	수신 시스템의 프로그램에서 JDBC 드라이버를 이용하여 송신 시스템 DB와 연결하여 사용하는 것이다.

Hyper Link	웹 애플리케이션에서 Hyper Link를 사용하는 방식이다.
Socket 21.3	서버에서 통신을 위한 소켓(Socket)을 생성, 포트를 할당한 뒤 클라이언트의 통신 요청 시 클라이언트와 연결하는 방식이다.
Web Service	웹 서비스에서 WSDL, UDDI, SOAP 프로토콜을 이용하는 방식이다.
연계 솔루션	실제 송·수신 처리와 진행 상황을 모니터링 및 통제하는 EAI 서버, 송·수신 시스템에 설치되는 어댑터(Client)를 이용하는 방식이다.

- 시스템 연계 기술
  - API : 그거
  - WSDL(Web Service Description Language)
  - UDDI : XML 방식 씬
  - SOAP : 통신 프로토콜 중 하나

## 19. 미들웨어 솔루션

- 각 인터페이스의 연결 과정에서 중재하는 역할
- 미들웨어 솔루션의 종류 (gpt에 물어보기)
  - DB
  - TP-Monitor(Tx Processing Monitor)
  - ORB(Object Request Brocker)
  - RPC(Remote Procedure Call)
  - MOM(Message Orientied Middleware)
  - WAS(Web Aplication Server)

### • 분류

#### ◆ 미들웨어 솔루션 분류

DB 미들웨어 (애플리케이션-TO-데이터 방식)	통신 미들웨어 (애플리케이션-TO-애플리케이션 방식)
<ul style="list-style-type: none"> <li>• ODBC(Open Database Application Connectivity)</li> <li>• IDAP(Integrated-Database Application Interface)</li> <li>• DRDA(Distributed Relational Data Access)</li> <li>• OLEDB</li> </ul>	<ul style="list-style-type: none"> <li>• RPC(Remote Procedure Call)</li> <li>• DCE(Distributed Computing Environment)</li> <li>• MOM(Message Oriented Middleware)</li> <li>• ORB(Object Request Broker)</li> <li>• OTM(Object Transaction Monitor)</li> </ul>

▸ Web 서버와 WAS의 구성 형태

