※ 객체지향 분석 모델

. Booch (부치) : 미시적, 거시적 개발 프로세스를 모두 사용 (클래스/객체 분석 및 식별)
. Jacobson (제이콥슨) : Use case를 사용 (사용자, 외부 시스템이 시스템과 상호작용)
. Coad-Yourdon : E-R 다이어그램 사용 / 객체의 행위 모델링
. Wirfs-Brock : 분석과 설계 구분 없으며 고객 명세서 평가 후 설계 작업까지 연속 수행
. **Rumbaugh (럼바우) :** 가장 일반적으로 사용, 객체/동적/기능 모델로 구분
객체 모델링 (Object) → **객**체 다이어그램 / 객체들 간의 관계 규정/정의
동적 모델링 (Dynamic) → **상**태 다이어그램 / 시스템 동적인 행위 기술
기능 모델링 (Function) → **자**료 흐름도(DFD) / 다수의 프로세스들 간의 처리 과정 표현

▶ 요구사항 명세

정형 명세	수학적 원리 / 정확하고 간결한 요구사항 표현 가능 어려운 표기법으로 사용자 이해 어려움 (VDM, Z, Petri-net, CSP)
비정형 명세	자연어, 그림 중심 / 쉬운 자연어 사용으로 의사소통 용이하나 작성자에 따라 모호한 내용으로 일관성 떨어짐 (FSM, Decision Table, E-R 모델, State Chart)

③ 소프트웨어 설계

■ 소프트웨어 설계 원리

1. 분할과 정복 : 여러 개의 작은 서브시스템으로 나눠서 각각을 완성

2. 모듈화 (Modularity) : 시스템 기능을 모듈 단위로 분류하여 성능/재사용성 향상
- 모듈 크기 ▲ → 모듈 개수 ▼ → 모듈간 통합비용 ▼ (but, 모듈 당 개발 비용 ▲)
- 모듈 크기 ▼ → 모듈 개수 ▲ → 모듈간 통합비용 ▲

3. 추상화 (Abstraction) : 불필요한 부분은 생략하고 필요한 부분만 강조해 모델화
→ 문제의 포괄적인 개념을 설계 후 차례로 세분화하여 구체화 진행
① 과정 추상화 : 자세한 수행 과정 정의 X, 전반적인 흐름만 파악가능하게 설계
② 데이터(자료) 추상화 : 데이터의 세부적 속성/용도 정의 X, 데이터 구조를 표현
③ 제어 추상화 : 이벤트 발생의 정확한 절차/방법 정의 X, 대표 가능한 표현으로 대체

4. 단계적 분해 (Stepwise refinement) : 하향식 설계 전략 (by Niklaus Wirth)
- 추상화의 반복에 의한 세분화 / 세부 내역은 가능한 뒤로 미루어 진행

5. 정보 은닉 (Information Hiding)
- 한 모듈 내부에 포함된 절차/자료 관련 정보가 숨겨져 다른 모듈의 접근/변경 불가
- 모듈을 독립적으로 수행할 수 있어 요구사항에 따라 수정/시험/유지보수가 용이함

■ 아키텍처 패턴

Layer	시스템을 계층으로 구분/구성하는 고전적 방식 (OSI 참조 모델)
Client-server	하나의 서버 컴포넌트와 다수의 클라이언트 컴포넌트로 구성 클라이언트와 서버는 요청/응답 제외 시 서로 독립적 * 컴포넌트(Component) : 독립적 업무/기능 수행 위한 실행코드 기반 모듈
Pipe-Filter	데이터 스트림 절차의 각 단계를 필터 컴포넌트로 캡슐화 후 데이터 전송 / 재사용 및 확장 용이 / 필터 컴포넌트 재배치 가능 단방향으로 흐르며, 필터 이동 시 오버헤드 발생 변환, 버퍼링, 동기화 적용 (ex. UNIX 쉘 – Shell)
Model-view Controller	모델 (Model) : 서브시스템의 핵심 기능 및 데이터 보관 뷰 (View) : 사용자에게 정보 표시 컨트롤러 (Controller) : 사용자로부터 받은 입력 처리 → 각 부분은 개별 컴포넌트로 분리되어 서로 영향 X → 하나의 모델 대상 다수 뷰 생성 ▶ 대화형 애플리케이션에 적합
Master-slave	마스터에서 슬레이브 컴포넌트로 작업 분할/분리/배포 후 슬레이브에서 처리된 결과물을 다시 돌려 받음 (병렬 컴퓨팅)
Broker	컴포넌트와 사용자를 연결 (분산 환경 시스템)
Peer-to-peer	피어를 한 컴포넌트로 산정 후 각 피어는 클라이언트가 될 수도, 서버가 될 수도 있음 (멀티스레딩 방식)
Event-bus	소스가 특정 채널에 이벤트 메시지를 발행 시 해당 채널을 구독한 리스너들이 메시지를 받아 이벤트를 처리함
Blackboard	컴포넌트들이 검색을 통해 블랙보드에서 원하는 데이터 찾음
Interpreter	특정 언어로 작성된 프로그램 코드를 해석하는 컴포넌트 설계

■ UML (Unified Modeling Language) → 구성요소 : 사물, 관계, 다이어그램
- 고객/개발자 간 원활한 의사소통을 위해 표준화한 대표적 객체지향 모델링 언어
- Rumbaugh, Booch, Jacobson 등 객체지향 방법론의 장점 통합

※ 인터페이스 : 클래스/컴포넌트가 구현해야하는 오퍼레이션 세트를 정의하는 모델 요소

1) 사물 (Things) : 구조(개념,물리적 요소) / 행동 / 그룹 / 주해(부가적 설명, 제약조건)

2) 관계 (Relationship)

연관 관계 (Association)	2개 이상의 사물이 서로 관련
집합 관계 (Aggregation)	하나의 사물이 다른 사물에 포함 (전체-부분 관계)
포함 관계 (Composition)	집합 관계 내 한 사물의 변화가 다른 사물에게 영향
일반화 관계 (Generalization)	한 사물이 다른 사물에 비해 일반/구체적인지 표현 (한 클래스가 다른 클래스를 포함하는 상위 개념일 때)
의존 관계 (Dependency)	사물 간 서로에게 영향을 주는 관계 (한 클래스가 다른 클래스의 기능을 사용할 때)
실체화 관계 (Realization)	한 객체가 다른 객체에게 오퍼레이션을 수행하도록 지정 / 서로를 그룹화할 수 있는 관계

3) 다이어그램 (Diagram)

구조, 정적 다이어그램 (클래스컴배복패)	**클래스** (Class)	클래스 사이의 관계 및 속성 표현
	객체 (Object)	인스턴스를 객체와 객체 사이의 관계로 표현
	컴포넌트 (Component)	구현 모델인 컴포넌트 간의 관계 표현
	배치 (Deployment)	물리적 요소(HW/SW)의 위치/구조 표현
	복합체 구조 (Composite Structure)	클래스 및 컴포넌트의 복합체 내부 구조 표현
	패키지 (Package)	UML의 다양한 모델요소를 그룹화하여 묶음
행위, 동적 다이어그램 (유시커상활타상)	**유스케이스** (Use case)	사용자의 요구를 분석 (사용자 관점) →사용자(Actor) + 사용 사례 (Use Case)
	시퀀스 (Sequence)	시스템/객체들이 주고받는 메시지 표현 → 구성항목: 액터* / 객체 / 생명선 / 메시지 제어 삼각형
	커뮤니케이션 (Communication)	객체들이 주고받는 메시지와 객체 간의 연관관계까지 표현
	상태 (State)	다른 객체와의 상호작용에 따라 상태가 어떻게 변화하는지 표현
	활동 (Activity)	객체의 처리 로직 및 조건에 따른 처리의 흐름을 순서에 따라 표현
	타이밍 (Timing)	객체 상태 변화와 시간 제약 명시적으로 표현
	상호작용 개요 (Interaction Overview)	상호작용 다이어그램 간 제어 흐름 표현

▶ UI 설계 (User Interface) : 직관성, 유효성(사용자의 목적 달성), **학습성, 유연성**
- 설계 지침 : 사용자 중심 / 일관성 / 단순성 / 결과 예측 / 가시성 / 표준화 / 접근성 / 명확성 / 오류 발생 해결

- **CLI** (Command Line), **GUI** (Graphical), **NUI** (Natural), **VUI** (Voice), **OUI** (Organic)
텍스트 그래픽 말/행동 음성 사물과 사용자 상호작용

- UI 설계 도구

Wireframe	기획 초기 단계에 대략적인 레이아웃을 설계
Story Board	최종적인 산출문서 (와이어프레임-UI, 콘텐츠 구성, 프로세스 등)
Prototype	와이어프레임 / 스토리보드에 인터랙션 적용 실제 구현된 것처럼 테스트가 가능한 동적인 형태 모형 ***인터렉션** : UI를 통해 시스템을 사용하는 일련의 상호작용 (동적효과)
Mockup	실제 화면과 유사한 정적인 형태 모형
Use case	사용자 측면 요구사항 및 목표를 다이어그램으로 표현