

핵심 요약 노트

정보 처리 기사 필기

# 목 차

1. 소프트웨어 설계 .....	1
2. 소프트웨어 개발 .....	4
3. 데이터베이스 구축 .....	7
4. 프로그래밍 언어 활용 .....	9
5. 정보시스템 구축 관리 .....	12



## ■ 소프트웨어 생명 주기 (Software Development Life Cycle, SDLC)

폭포수	선형 순차적 개발 / 고전적, 전통적 개발 모형 / Step-by-Step
HIPO (Hierarchy Input Process Output)	하향식 설계 방식 / 가시적, 총체적, 세부적 다이어그램으로 구성 기능과 자료의 의존 관계 동시 표현 / 이해 쉽고 유지보수 간단
프로토타입	고객의 need 파악 위해 <b>견본/시제품</b> 을 통해 최종 결과 예측 인터페이스 중심 / <b>요구사항 변경 용이</b>
나선형 (Spiral)	폭포수 + 프로토타입 + <b>위험 분석</b> 기능 추가 (위험 관리/최소화) <b>점진적 개발 과정 반복</b> / 정밀하며 유지보수 과정 필요 X <b>★ 계획 수립 → 위험 분석 → 개발 및 검증 → 고객 평가</b>
애자일 (Agile)	일정한 짧은 주기(Sprint 또는 Iteration) 반복하며 개발 진행 → 고객 요구사항에 유연한 대응 (고객 소통/상호작용 중시) Ex. XP(eXtreme Programming), Scrum, FDD (기능중심), 린 (LEAN), DSDM (Dynamic System Development Method)

**하향식 설계 (Top-down):** **절차 지향** (순차적) / 최상위 컴포넌트 설계 후 하위 기능 부여

→ 테스트 초기부터 사용자에게 시스템 구조 제시 가능

**상향식 설계 (Bottom-up):** **객체 지향** / 최하위 모듈 먼저 설계 후 이들을 결합하고 검사

→ 인터페이스 구조 구조 변경 시 상위 모듈도 같이 변경 필요하여 **기능 추가 어려움**

**\* Component:** 명백한 역할을 가지며 재사용되는 모든 단위 / 인터페이스 통해 접근 가능

## ■ 스크럼 (Scrum) 기법

제품 책임자 (Product Owner / PO)	. 요구사항이 담긴 백로그(Back log)를 작성 및 우선순위 지정 . 요구사항을 책임지고 의사결정
스크럼 마스터 (Scrum Master / SM)	. 원활한 스크럼 위해 객관적 시각으로 조언 및 가이드 제시 . 진행사항 점검 및 장애요소 논의 후 해결
개발팀 (Development Team / DT)	. PO와 SM을 제외한 모든 팀원 . 백로그에 스토리 추가 가능하나 우선순위는 지정 불가

## ※ 스크럼 개발 프로세스

스프린트 계획 회의 → **스프린트** → 일일 스크럼 회의 → 스프린트 검토 회의 → 스프린트 회고  
**개발 기간 2~4주 단기간 목표**

## ■ 익스트림 프로그래밍 (eXtreme Programming, XP)

- 고객의 요구사항을 유연하게 대응하기 위해 고객 참여와 신속한 개발 과정을 반복
- 핵심 가치 : 용기 / 단순성 / 의사소통 / 피드백 / 존중

**※ 피드백:** 시스템의 상태와 사용자의 지시에 대한 효과를 보여 주어 사용자가 명령에 대한 진행 상황과 표시된 내용을 해석할 수 있게 도와줌

- 기본 원리 : 전체 팀 / 소규모 릴리즈 / 테스트 주도 개발 / 계속적인 통합 /  
공동 소유권 (Collective ownership) / 짝(pair) 프로그래밍 /  
디자인 개선 (리팩토링) / 애자일(Agile) 방법론 활용  
상식적 원리 및 경험 추구, **개발 문서보단 소스코드에 중점 (문서화 X)**

## ■ 요구사항 : 어떠한 문제를 해결하기 위해 필요한 조건 및 제약사항을 요구

소프트웨어 개발/유지 보수 과정에 필요한 기준과 근거 제공

## ▶ 요구사항의 유형

기능적 요구	실제 시스템 수행에 필요한 요구사항 <i>ex. 금융 시스템은 조회/인출/입금/충금 기능이 있어야 한다.</i>
비기능적 요구	성능, 보안, 품질, 안정성 등 실제 수행에 보조적인 요구사항 <i>Ex. 모든 화면이 3초 이내에 사용자에게 보여야 한다.</i>

## ▶ 요구사항 개발 프로세스 ★순서 중요

① 도출/추출	이해관계자들이 모여 요구사항 정의 (식별하고 이해하는 과정) Ex. 인터뷰, 설문, 브레인스토밍, 청취, 프로토타이핑, 유스케이스
② 분석	사용자 요구사항에 타당성 조사 / 비용 및 일정에 대한 제약 설정 Ex. 관찰, 개념 모델링, 정형 분석, 요구사항 정의 문서화
③ 명세	요구사항 체계적 분석 후 승인가능하도록 문서화
④ 확인/검증	요구사항 명세서가 정확하고 완전하게 작성되었는지 검토

## ▶ 요구사항 분석 기법 : 분류 / 개념 모델링 / 할당 / 협상 / 정형 분석

## ▶ 요구사항 확인 기법

요구사항 검토	문서화된 요구사항을 확인 (일반적 방법)
프로토타이핑	요구사항이 반영된 프로토타입 지속 제작 (피드백 후 반복 제작)
모델 검증	요구사항 분석 단계에서 개발된 모델이 충족되는지 검증
인수 테스트	사용자 측면에서 실제 사용 환경 내 요구사항이 충족되는지 검증

구조 / 행동 / 그룹 / 주해



## ■ UML (Unified Modeling Language) 구성 요소 : **사물** / **관계** / **다이어그램**

→ 고객/개발자 간 원활한 의사소통을 위해 **표준화한 대표적 객체지향 모델링 언어**

연관 관계 (Association)	2개 이상의 사물이 서로 관련
집합 관계	하나의 사물이 다른 사물에 포함
포함 관계	집합 관계의 특수 형태
일반화 관계 (Generalization)	한 사물이 다른 사물에 비해 일반/구체적인지 표현
의존 관계 (Dependency)	사물 간 서로에게 영향을 주는 관계
실체화 관계 (Realization)	한 객체가 다른 객체에게 오퍼레이션을 수행하도록 지정 / 서로를 그룹화할 수 있는 관계

구조, 정적 다이어그램 (클래스배백패)	클래스 (Class) / 객체 (Object) / 컴포넌트 (Component) / 배치 (Deployment) / 복합체 (Composite) / 패키지 (Package)
행위, 동적 다이어그램 (유시커상활타상)	. 유스케이스 (Use case) : 사용자의 요구를 분석 . 순차 (Sequence) : 시스템/객체들이 주고받는 메시지 표현 → 구성항목: 액터* / 객체 / 생명선 / 메시지 / 제어 삼각형 . 커뮤니케이션 (Communication) : 객체들 간의 연관까지 표현 . 상태 (State) : 상태가 어떻게 변화하는지 표현 . 활동 (Activity) : 처리의 흐름을 순서에 따라 표현 . 타이밍 : 객체 상태 변화와 시간 제약을 명시적으로 표현 . 상호작용 개요 : 상호작용 다이어그램 간 제어 흐름 표현

**※ 액터:** 시스템과 상호작용하는 사람이나 다른 시스템에 의한 역할

- 사용자 액터 : 기능을 요구하는 대상이나 시스템의 수행결과를 통보받는 사용자
- 시스템 액터 : 본 시스템과 데이터를 주고 받는 연동 시스템

## ■ User Interface (UI) : **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 <b>동적인 형태 모형</b> <b>*인터랙션:</b> UI를 통해 시스템을 사용하는 일련의 상호작용 (동적효과)
Mockup	실제 화면과 유사한 <b>정적인 형태 모형</b>
Use case	사용자 측면 요구사항 및 목표를 다이어그램으로 표현

## - UI 프로토타입의 장/단점

장점	. 사용자를 설득하고 이해시키기 쉬움 / 사전에 오류 예방 가능 . 개발 시간 단축 (요구사항 및 기능의 불일치 방지)
단점	. 반복적 개선/보완 작업으로 작업 시간 증가, 자원 소모 . 부분적 프로토타이핑 시 중요 작업 누락 및 생략 가능



## ■ 소프트웨어 아키텍처

- 1. 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 ▲

- 2. 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

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

## 4. 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

## ※ 소프트웨어 아키텍처 설계 과정

- ① 설계 목표 설정 → ② 시스템 타입 결정 → ③ 아키텍처 패턴 적용  
 → ④ 서브시스템 구체화 → ⑤ 검토

## ※ 소프트웨어 아키텍처의 품질 속성

- 성능 / 보안 / 가용성 / 기능성 / 변경 용이성 / 확장성 / 배치성 / 안정성

## ■ 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 셸 - 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 → <b>하나의 모델 대상 다수 뷰 생성</b> ▶ 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

**EAI (Enterprise Application Integration):** 기업 응용 프로그램을 하나로 통합

**FEP (Front-End Processor):** 입력 데이터를 프로세스가 처리하기 전에 미리 처리하여  
프로세스 처리 시간을 줄여주는 프로그램

## ■ 객체 지향 (Object-oriented)

- 객체와 속성, 클래스와 멤버, 전체와 부분 등으로 나누어 분석

객체 (Object)	<b>고유 식별자</b> / 하나의 독립된 존재 / 일정한 기억장소 보유 상태(state) = 객체가 가질 수 있는 조건, 속성 값에 의해 정의 행위(연산, Method) = 객체가 반응할 수 있는 메시지 집합
클래스 (Class)	공동 속성과 연산(행위)을 갖는 <b>객체들의 집합</b> / 데이터 추상화 단위 ※ <b>인스턴스 (Instance)</b> : 클래스에 속한 각각의 객체 ※ <b>Operation</b> : 클래스의 동작 / 객체에 대해 적용될 메서드 정의
캡슐화 (Encapsulation)	데이터와 데이터 처리 함수를 <b>하나로 묶음</b> 세부 내용 은폐(정보 은닉) → 외부 접근 제한 결합도 낮음 / 재사용 용이 / 인터페이스 단순 / 오류 파급효과 낮음
상속 (Inheritance)	상위 클래스의 속성과 연산을 하위 클래스가 물려받는 것 ※ <b>다중 상속</b> : 단일 클래스가 두 개 이상의 상위 클래스로부터 상속
다형성 (Polymorphism)	하나의 메시지에 각 객체 별 고유 특성에 따라 여러 형태의 응답

**오버라이딩 (Overriding):** 상위클래스로부터 상속받은 메서드를 하위클래스에서 재정의  
→ 단, 메서드 이름 / 매개변수 / 반환 타입은 동일해야 함

**오버로딩 (Overloading):** 메서드 이름은 동일하나 매개변수 개수 또는 타입을 다르게 지정

## ※ 분석 방법론

- . 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)**

객체 모델링 (Object) → 객체 다이어그램 / 객체들간의 관계 규정

동적 모델링 (Dynamic) → 상태 다이어그램 / 시스템 행위 기술

기능 모델링 (Function) → 자료 흐름도 / 다수의 프로세스들 간의 처리 과정 표현

## ※ 데이터/자료 흐름도 (DFD, Data flow diagram) 구성요소:

프로세스 (Process) / 자료 흐름 (Flow) / 자료 저장소 (Data store) / 단말 (Terminator)  
 원 화살표 평행선 사각형

→ 구조적 분석 기법에 이용 / 시간 흐름 명확한 표현 불가 / 버블(bubble) 차트

## ※ 5대 설계 원칙 (SOLID)

- . **단일 책임 원칙 (SRP, Single Responsibility Principle)**  
 → 모든 클래스/객체는 하나의 책임만 / 완전한 캡슐화  
 . **개방 폐쇄의 원칙 (OCP, Open Closed Principle)**  
 → 확장에는 Open하고, 수정에는 Close되어야 한다.  
 . **리스코프 교체 원칙 (LSP, Liskov Substitution Principle)**  
 → 상위 클래스의 행동 규약을 하위 클래스가 위반하면 안된다.  
 . **인터페이스 분리 원칙 (ISP, Interface Segregation Principle)**  
 → 클라이언트가 비사용 메서드에 의존하지 않아야 한다.  
 . **의존성 역전 원칙 (DIP, Dependency Inversion Principle)**  
 → 의존 관계 수립 시 변화하기 어려운 것에 의존해야 한다.

## ■ CASE (Computer-Aided Software Engineering)

- 소프트웨어 개발 시 요구분석/설계/구현/검사 과정을 **자동화(표준화)**해주는 작업  
 - 1980년대 소개된 이후 90년대부터 자주 사용  
 - 장점: 비용 절감 / 모듈 재사용성 향상 / SW 품질 및 생산성 향상 / 유지보수 간편  
 - 지원 기능: SW 생명 주기 전체 단계를 연결 / 자동화 통합 도구 제공 / 그래픽 기능 제공 (문서화, 명세화 도움) / 자료 흐름도 작성  
 모델의 오류 검증 / 모델 간 모순검사 / 원시코드 생성

## ■ 데이터베이스 관리 시스템 (DBMS)

- 사용자 요구에 따라 정보 생성하고, 데이터베이스를 관리해주는 소프트웨어  
 - 분석 시 고려사항: 가용성 (무결성) / 성능 (효율성) / 상호 호환성 (일관성) / 기술지원 / 회복 / 보안 / 데이터베이스 확장 / 구축 비용



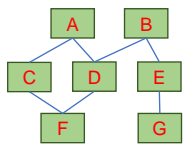
## ■ 요구사항 검증 (Requirements Validation)

- 실제로 고객이 원하는 시스템을 제대로 정의했는지 점검하는 과정
- 시스템 개발 완료 후 문제 발생 시 막대한 재작업 비용 발생하기에 검증 중요  
(실제 요구사항 반영 여부 / 문서 상 요구사항이 서로 상충되지 않는지 점검)
- ① 동료 검토 (Peer review) : 작성자가 내용 설명 후 동료들이 결함 검토
- ② 워크 스루 (Walk through) : 요구사항 명세서 미리 배포 후 짧은 검토 회의 진행
- ③ 인스펙션 (Inspection) : 작성자 제외한 다른 전문가들이 결함 검토

※ 검증 체크리스트 : 기능성 (Functionality) / 완전성 (Completeness) / 일관성 (Consistency) / 명확성 (Unambiguity) / 검증 가능성 (Verifiability) / 추적 가능성 (Traceability) / 변경 용이성 (Easily Changeable)

▶ 팬인 (Fan-in) : 자신을 사용하는 모듈의 수 → 팬인이 높으면 재사용 측면 설계 우수  
but, 단일 장애점 발생 가능성 높아 집중적인 관리/테스트 필요

▶ 팬아웃 (Fan-out) : 자신이 사용하는 모듈의 수 → 불필요한 호출 가능성 (단순화 필요)



Q. D의 팬인의 개수는? 2개 (A, B)

Q. D의 팬아웃의 개수는? 1개 (F)

## ■ 코드 (Code) 부여 체계

순차/순서 코드 (Sequence)	최초의 자료부터 차례로 일련번호 부여 Ex) 1, 2, 3, 4, 5 ...
블록 코드 (Block)	공통적인 블록으로 먼저 구분 후 각 블록 내 일련번호 부여 Ex) 101 ~ 150 : 중학생 / 201 ~ 250 : 고등학생
10진 코드 (Decimal)	0~9까지 10진 분할 후, 다시 추가 10진 분할 → 필요만큼 반복 1000: 음악 / 1100 : 국악 / 1110 : 거문고
그룹 분류 코드 (Group Classification)	대/중/소 분류 후 각 그룹 안에 일련번호 부여 Ex) 1-01-001 : 서울-초등-1학년
연상 코드 (mnemonic)	관계있는 숫자/문자/기호를 사용하여 코드 부여 Ex) S-03-13 : S 핸드폰 3월 13일 제조
표의 숫자 코드 (Significant digit)	물리적 수치를 그대로 코드에 적용 (길이, 넓이, 부피 등) Ex) 150-80-120 : 150x80x120 길이/폭/높이 서랍장
합성 코드 (Combined)	2개 이상의 코드를 조합 Ex) KE243: 대한항공 243기

## ■ 자료 사전 (Data Dictionary) 기호

=	정의	[ ]	택일 / 선택
+	구성	( )	생략
**	설명 / 주석	{ }	반복

## ■ 미들웨어 (Middleware)

- 운영체제와 애플리케이션 간에 중간 매개 역할 (사용자가 내부 동작 확인 필요 X)
- 클라이언트/서버 간 통신 담당, 시스템 간 표준화된 연결 지향 (데이터 교환 일관성)

DB (Database)	원격 데이터베이스와 연결 (2-Tier 아키텍처) → OCBC (MS) / IDAPI (볼랜드) / Glue (오라클)
RPC (Remote Procedure Call)	원격 프로시저를 로컬 프로시저처럼 호출 → Entera (이규브시스템스), ONC/RPC (OSF)
MOM (메시지 지향)	메시지 기반 비동기형 메시지 전달 (이기종 분산 데이터 시스템) 느린 대신 안정적 응답 필요 시 / 송수신속 간 메시지 큐 활용 → MQ (IBM) / Message Q (오라클) / JMS (JCP)
TP-Monitor (Transaction Processing)	온라인 업무 처리 (항공기, 철도 예약) / 빠른 응답 속도 필요 시 트랜잭션 처리를 감시/제어 → Tuxedo (오라클) / Tmax
Legacyware	기존 앱에 새로운 업데이트 기능 부여
ORB (Object Request Broker)	객체 지향 / 코바(CORBA) 표준 스펙 구현 / 네트워크 호출 → Orbix (Micro Focus), CORBA (OMG)
WAS (Web Application Server)	동적 콘텐츠 처리 / 웹 환경 구현 적합 (클라이언트/서버 X) HTTP 세션 처리 기능 / 미션-크리티컬 기업 업무 포괄 JAVA, EJB 컴포넌트 기반 구현 가능 → Web Logic (오라클), WebSphere (IBM), JEU, Tomcat

## ■ 디자인 패턴 : GoF (Gang of Four) 처음 제안하여 구체화



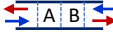
- 서브시스템에 속하는 컴포넌트들과 그 관계를 설계하기 위한 참조 모델  
cf) 아키텍처 패턴 : 전체 시스템의 구조를 설계
- 객체 지향 프로그래밍 설계 시 자주 발생하는 문제에 대한 반복적 해결 방법

생성 패턴 (5개)	Abstract Factory : 연관 객체들을 그룹 생성 후 추상적 표현 Builder : 건축하듯 조립, 객체 생성 → 동일 객체 생성에도 다른 결과 Factory Method : 객체 생성을 서브클래스에서 결정 Prototype : 원본 객체를 복제 Singleton : 여러 프로세스가 하나의 객체를 동시에 참조 불가
구조 패턴 (7개)	Adaptor : 비호환 인터페이스에 호환성 부여하도록 변환 Bridge : 구현부에서 추상층 분리 후 독립적으로 확장하도록 구성 Composite : 복합, 단일 객체를 구분없이 사용 Decorator : 상속 사용 없이 객체 기능을 동적으로 확장 Façade : 상위 인터페이스 구성 후 서브클래스 기능을 간편하게 사용 Flyweight : 인스턴스 공유하여 메모리 절약 Proxy : 접근이 힘든 객체를 연결하는 인터페이스 역할
행위 패턴 (11개)	Chain of Responsibility : 한 객체 내 처리 불가 시 다음 객체로 이관 Command : 요청 명령어들을 추상/구체 클래스로 분리 후 단순화 Interpreter : 언어에 문법 표현 정의 Iterator : 동일 인터페이스 사용 Mediator : 객체들간 복잡한 상호작용을 캡슐화하여 객체로 정의 Memento : 객체를 특정 시점의 상태로 돌릴 수 있는 기능 제공 Observer : 한 객체 상태 변화 시 상속되어 있는 객체들에 변화 전달 State : 객체의 상태에 따라 동일 동작 다르게 처리 Strategy : 동일 계열 알고리즘을 개별적으로 캡슐화하여 상호 교환 Template Method : 서브클래스 내 공통 내용을 상위 클래스에 정의 Visitor : 처리 기능을 분리하여 별도의 클래스로 구성 분리된 처리 기능은 각 클래스를 방문하여 수행





## ■ 자료의 구조

선형 구조	배열 (Array)	<ul style="list-style-type: none"> <li>정적 구조 / 기억장소 추가 어려움 / 첨자 사용</li> <li>반복적 데이터 처리 작업에 적합</li> <li>데이터마다 동일 이름 변수 사용해 처리 간편</li> </ul>
	스택 (Stack)	<ul style="list-style-type: none"> <li>리스트 한쪽 끝으로만 자료의 삽입/삭제</li> <li>후입선출 (Last-in-First-out, LIFO)</li> <li>인터럽트 처리 / 서버루틴 호출 작업에 활용</li> <li>※ Underflow : 삭제할 데이터가 없을 때</li> <li>※ Overflow : 삽입할 공간이 없을 때</li> </ul> 
	큐 (Queue)	<ul style="list-style-type: none"> <li>리스트 한쪽은 노드 삽입 / 반대쪽은 노드 삭제</li> <li>선입선출 (First-in-First-out, FIFO)</li> </ul> 
	데크 (Deque)	<ul style="list-style-type: none"> <li>리스트 양쪽 끝에서 삽입/삭제 가능</li> </ul> 
	선형 리스트 (Linear List)	<ul style="list-style-type: none"> <li>연속 리스트 : 자료구조가 연속되는 기억장소에 저장 데이터 중간 삽입 시 연속된 빈 공간 필요 삽입/삭제 시 자료의 이동 필요</li> <li>연결 리스트 : 노드의 포인터 부분을 이용해 서로 연결 노드의 삽입/삭제 작업 용이 포인터가 필요해 기억 공간 효율 낮음 포인터 찾는 시간 필요 → 접근 속도 느림</li> </ul>
비선형 구조	트리 (Tree)	<ul style="list-style-type: none"> <li>노드(Node)와 가지(Branch)로 구성된 그래프 (사이클 X)</li> <li>→ 차수 (Degree) : 각 노드에서 뻗어나온 가지 수</li> <li>※ 트리의 차수 : 각 노드의 차수 중 최대 차수</li> <li>→ 단말 (Terminal) : 자식/Degree가 없는 노드</li> </ul>
	그래프 (Graph)	<ul style="list-style-type: none"> <li>방향 그래프 : 정점 연결 선 방향 존재 / <math>n(n-1)</math></li> <li>무방향 그래프 : 방향 X / 최대 간선 수 = <math>n(n-1)/2</math></li> </ul>

■ 빌드 자동화 도구 : 소스 코드를 소프트웨어로 변환하는 과정에 필요한 전처리, 컴파일 등의 작업을 수행하는 소프트웨어

Ant (Another Neat Tool)	<ul style="list-style-type: none"> <li>자바 프로젝트의 공식적 빌드 자동화 도구 (XML 기반)</li> <li>표준 없어 개발자가 모든 것 정의 / 스크립트 재사용 어려움</li> </ul>
Maven	<ul style="list-style-type: none"> <li>Ant 대안 / 표준이 있어 예외 사항만 기록</li> <li>컴파일과 빌드 동시 수행 가능 / 의존성 설정 후 Library 관리</li> </ul>
Gradle	<ul style="list-style-type: none"> <li>Ant 및 Maven 보완 / <b>안드로이드</b> 공식 개발 도구 (<b>Groovy 기반</b>)</li> <li>의존성 활용 / 그루비 기반 / 빌드 캐시 기능 지원 → 속도 향상</li> <li>실행할 명령을 모아 태스크(Task)로 만든 후 태스크 단위 실행</li> </ul>
Jenkins	<ul style="list-style-type: none"> <li>JAVA 기반 오픈 소스 형태 / 서버리스 컨테이너에서 실행</li> <li>형상 도구 관리와 연동 가능 (SVN, Git)</li> </ul>

## ■ 디지털 저작권 관리 (DRM, Digital Rights Management)

- 콘텐츠 제공자/분배자/소비자, 패키저 (배포가능한 형태로 암호화하는 프로그램)
- 클리어링 하우스 (Clearing House) : 사용 권한, 라이선스 발급, 결제 관리
- DRM 컨트롤러 : 배포된 콘텐츠의 이용 권한을 통제하는 프로그램
- 보안 컨테이너 : 콘텐츠 원본을 안전하게 유통하기 위한 전자적 보안 장치
- 기술 요소 : 암호화 / 키 관리 / 식별 기술 / 저작권 표현 / 암호화 파일 생성 정책 관리 / 크랙 방지 / 인증

## ■ 소프트웨어 형상 관리 (SCM: Software Configuration Management)

- 개발 과정에서 **SW 변경사항을 관리**하기 위한 일련의 활동 / 개발 전체 단계 적용
- 중요성: 변경사항 추적/통제, 무절제한 변경 방지, 개발 진행 이력 확인
- 형상 관리 역할: 배포본 관리 용이 / 불필요한 소스 수정 제한 / 여러 개발자 동시 개발

형상 식별	관리 대상에 이름/번호 부여 후 계층 구조로 구분 → 수정/추적 용이
형상 통제	식별된 형상 항목에 대한 변경 요구 검토 (기준선 반영될 수 있게)
형상 감사	기준선(Base line)의 무결성 평가 위해 확인/검증/검열 과정 진행
형상 기록	형상 식별/통제/감사 작업 결과를 기록/관리하고 보고서 작성

※ 형상 관리 도구 : CVS, SVN, Git

※ 제품 SW의 형상 관리 역할 : 배포본 관리 유용 / 소스 수정 제한

동일 프로젝트에 대해 여러 개발자 참여 후 동시 개발 가능

## ■ 소프트웨어 버전 등록 용어

저장소 (Repository)	최신 버전 및 변경 내역 관련 정보 저장소
가져오기 (Import)	버전 관리 미진행 상태의 초기 저장소에 처음으로 파일을 복사
체크아웃 (Check-out)	프로그램 수정 위해 저장소에서 파일을 받아옴
체크인 (Check-in)	체크아웃한 파일 수정 후 저장소에 새로운 버전으로 파일을 갱신
커밋 (Commit)	체크인 시 이전 갱신된 내용이 있는 경우 '충돌(Conflict)' 알린 후 Diff 도구 이용해 수정 후 갱신 완료
동기화 (Update)	저장소에 있는 최신 버전 상태로 자신의 작업 공간을 동기화

※ 버전 등록 과정 : 가져오기 → 인출 → 커밋 → 동기화 → 차이

## ■ 소프트웨어 버전 관리 도구

공유폴더	<ul style="list-style-type: none"> <li>버전 관리 자료가 로컬 컴퓨터 내 공유 폴더에 저장/관리</li> <li>개발자는 개발이 완료된 파일을 약속된 공유 폴더에 매일 복사</li> <li>담당자는 해당 공유 파일을 본인 PC에 복사해 이상 유무 확인</li> <li>Ex) SCCS, RCS(Revision Control System), PVCS, QVCS</li> </ul>
클라이언트/서버	<ul style="list-style-type: none"> <li>버전 관리 자료가 중앙 시스템(서버)에 저장/관리</li> <li>자신의 PC(클라이언트)로 복사/작업 후 중앙 서버에 업로드</li> <li>하나의 파일을 여러 개발자가 작업할 경우 경고 메시지 출력</li> <li>서버 문제 발생 시 개발자간 협업 및 버전 관리 작업 중단</li> <li>Ex) CVS, SVN(Subversion)</li> </ul>
분산 저장소	<ul style="list-style-type: none"> <li>원격 저장소와 개발자 PC의 로컬 저장소에 함께 저장/관리</li> <li>원격 저장소에 문제 발생해도 로컬 저장소 이용해 작업 가능</li> <li>로컬 저장소에 우선 반영(commit) 후 원격저장소에 반영 (Push)</li> <li>로컬 저장소에서 작업하여 처리속도 빠름</li> <li>Ex) Git, Bitkeeper</li> </ul>
SVN (Subversion)	<ul style="list-style-type: none"> <li>CVS 개선 버전 / 아파치 재단 2000년 발표</li> <li>커밋(Commit)할 때 마다 리비전(Revision) 1씩 증가</li> <li>서버는 주로 유닉스 (UNIX) / 오픈소스로 무료 사용 가능</li> <li>CVS의 한계인 파일이나 디렉터리의 이름 변경, 이동 가능</li> </ul>
Git (깃)	<ul style="list-style-type: none"> <li>원격 저장소: 여러 사람들이 협업을 위해 버전을 공동 관리 장소</li> <li>로컬 저장소: 개발자들이 본인의 실제 개발을 진행하는 장소</li> <li>브랜치(Branch) 이용 시 기존 버전 관리 틀에 영향없이 다양한 기능 테스트 가능 / 파일의 변화를 스냅샷(Snapshot)으로 저장</li> <li>버전의 흐름 파악 가능</li> </ul>

## ■ 소스 코드 품질분석 도구

- 정적 분석 도구 : pmd (코드 결함), cppcheck (C++), checkstyle (Java), SonarQube, Find Bug, cobertura, ccm
- 동적 분석 도구 : Valance, Valgrind, Avalanche

## ■ EAI (Enterprise Application Integration)

- 기업 내 각종 앱/플랫폼 간 정보 전달/연계/통합 등 상호작용 연동하는 모듈 연계 방식

Point to Point	데이터 간 포인트-포인트 개별 연결 / 변경 및 재사용 어려움
Hub & Spoke	중앙(Hub) 시스템을 통한 데이터 전송 → 중앙 집중형 방식 확장/유지보수 유리하나, 중앙 허브 장애 발생 시 전체 시스템 영향
Message Bus	앱 사이에 미들웨어를 두고 처리 / 확장성 및 대용량 처리 가능
Hybrid	Hub & Spoke와 Message bus의 혼합형 → 데이터 병목 현상 최소화



## ■ 애플리케이션 테스트 기본 원리

- 테스트는 기본적으로 결함이 존재함을 밝히는 것 (무결함을 증명할 수는 없음)  
→ 완벽한 테스트는 근원적으로 불가능 (무한 경로, 무한 입력 불가)
- 결함 집중 : **파레토(Pareto) 법칙** - 20%의 모듈에서 전체 결함 80% 발생
- **살충제 패러독스** : 동일한 테스트 케이스에 의한 반복 테스트는 새로운 버그 발견 X
- 오류-부재의 궤변 : 결함이 없다 해도 요구사항 미충족 시 품질 저하
- Brooks의 법칙 : 지연되는 프로젝트에 인력 추가 투입 시 더 지연

## ■ 애플리케이션 테스트 분류

### ① 프로그램 실행 여부

정적 테스트	프로그램 실행 X / 명세서, 소스 코드만 분석 Ex. 워크 시트, 인스펙션, 코드 검사
동적 테스트	프로그램 실행 후 오류 검사 ex. 화이트/블랙박스 테스트

### ② 테스트 기반

명세 기반	사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 구현하는지 확인 → ex. 동등 분할 / 경계값 분석 (블랙박스)
구조 기반	SW 내부 논리 흐름에 따라 테스트 케이스 작성/확인 ex. 구문 기반 / 결정 기반 / 조건 기반 (화이트박스)
경험 기반	테스터의 경험을 기반으로 수행 ex. 에러 추정, 체크리스트, 탐색적 테스트

### ③ 시각(관점) 기반

검증 (Verification)	<b>개발자</b> 의 시각에서 제품의 생산 과정 테스트 Ex. 단위/통합/시스템 테스트
확인 (Validation)	<b>사용자</b> 의 시각에서 생산된 제품의 결과 테스트 Ex. 인수 테스트 (알파 / 베타)

### ④ 목적 기반

회복 (Recovery)	시스템에 인위적 결함 부여 후 정상으로 회복되는 과정 확인
안전 (Security)	외부 불법 침입으로부터 시스템을 보호할 수 있는지 확인
강도 (Stress)	과부하 시 SW 정상 구동 여부 확인
성능 (Performance)	실시간 성능 및 전체적인 효율성 진단 (응답 시간, 업무 처리량)
구조 (Structure)	SW 내부 논리적 경로 및 소스 코드 복잡도 평가
회귀 (Regression)	SW 내 변경 또는 수정된 코드에 새로운 결함이 없음을 확인
병행 (Parallel)	변경 및 기존 SW에 동일한 데이터 입력 후 결과 비교

## ■ 정렬 알고리즘

<b>삽입</b>	이미 정렬된 파일에 새로운 레코드를 순서에 맞게 삽입하여 정렬
<b>선택</b>	N개의 레코드 중 최소값을 찾아 첫번째에 배치, 나머지 N-1개 레코드 중 최소값 찾아 두번째 배치, 이를 반복하여 정렬
<b>버블</b>	인접한 두 개의 레코드 키 값 비교 후 크기에 따라 레코드 위치 교환
퀵	자료 이동 최소화 후 하나의 파일을 부분적으로 나눠가면서 정렬
2-way 합병	이미 정렬된 2개의 파일을 한 개의 파일로 합병하여 정렬
셸	입력파일을 매개변수 값으로 서브파일 구성 후 각 서브파일을 삽입 정렬 방식으로 순서 배열하는 과정을 반복
힙	완전 이진 트리를 이용한 정렬

## ■ 화이트박스 테스트 (White Box Test)

- 모듈 안의 내용(작동) 직접 볼 수 있으며, 내부의 **논리적인 모든 경로를 테스트**
- 소스 코드의 모든 문장을 한 번 이상 수행 / **논리적 경로 점검** (선택, 반복 수행)
- 테스트 데이터 선택하기 위해 **검증 기준 (Coverage)** 정함  
→ **기초 경로 검사** : 대표적 화이트박스 테스트 기법 (동적 테스트) / 사이클 최대 한 번 (Base Path Test) 측정 결과는 실행 경로의 기초를 정의하는 지침으로 사용  
→ 제어 구조 검사 : **조건 검사 / 루프 검사 / 자료 흐름 검사** (Data flow test)

## ■ 블랙박스 테스트 (Black Box Test)

- 모듈 내부의 내용 알 수 없음 / 소프트웨어 인터페이스에서 실시되는 테스트
- SW 각 기능이 완전히 작동되는 것을 입증하는 테스트로 '**기능 테스트**' 라고 함  
→ **동치 분할 검사 / 경계값 분석 / 원인-효과 그래프 검사 / 비교 검사 / 오류 예측 검사**

## ■ 개발단계에 따른 애플리케이션 테스트

→ 소프트웨어를 이루는 기본 단위 (독립적 기능)

### ① 단위 테스트 (Unit test) : 최소 단위(모듈/컴포넌트) 기반 테스트

주로 구조 기반 테스트 진행 / 기능성 테스트 최우선

### ② 통합 테스트 (Integration test)

- 단위 테스트 후 모듈을 통합하는 과정에서 발생하는 오류 및 결함을 찾는 테스트 기법

<b>하향식</b> (Top-down)	. 상위 모듈에서 하위 모듈 방향으로 통합 . 깊이 우선 통합법, 넓이 우선 통합법 . 초기부터 사용자에게 시스템 구조 보여줌 . <b>스텝(Stub)</b> : 모듈의 기능을 단순히 수행하는 도구 (시험용 모듈)
<b>상향식</b> (Bottom-up)	. 하위 모듈에서 상위 모듈 방향으로 통합 . 하나의 주요 제어 모듈과 관련된 종속 모듈의 그룹인 <b>클러스터 (Cluster)</b> 와 <b>드라이버(Driver)</b> 사용 / <b>스텝(Stub) 미사용</b>

### ③ 시스템 테스트 (System test) : 개발된 SW의 컴퓨터 시스템 내 작용여부 점검

실제 사용 환경과 유사한 테스트 환경 ▶ **기능적 및 비기능적** 테스트 구분  
블랙박스 화이트박스

### ④ 인수 테스트 (Acceptance test) : 사용자의 요구사항 충족 여부 확인

- **알파 테스트** : 통제된 환경에서 사용자가 개발자와 함께 확인
- **베타 테스트** : 통제되지 않은 환경에서 **개발자 없이** 여러 명의 사용자가 검증

## ■ 테스트 케이스 (Test case)

- 개발된 SW가 사용자의 요구사항을 잘 반영했는지 확인 위해 설계된 **입력값(데이터), 실행 조건, 기대 결과** 등으로 구성된 테스트 항목에 대한 명세서
- 테스트의 목표/방향 결정 후 테스트 케이스를 작성함

## ■ 테스트 시나리오 : 테스트 케이스를 적용하는 구체적인 절차를 명세한 문서

※ 유의사항 : 시스템/모듈/항목별 여러 개의 시나리오로 분리해 작성  
사용자의 요구사항과 설계 문서 등을 토대로 작성

## ■ 테스트 오라클 : 테스트 결과가 올바른지 판단하기 위한 기준

- 특성 : 제한된 검증 / 수학적 기법 / 자동화 기능
- 종류 : 참 (모든 오류 검출) / 샘플링 / 휴리스틱(추정) / 일관성 (수행 전/후 비교)

## ■ 테스트 하네스 (Test Harness)

테스트 드라이버 (Test Driver)	시험 대상의 하위 모듈 호출 / 모듈 테스트 수행 후의 결과 도출 → 상향식 통합 테스트에서 사용
테스트 스텝 (Test Stub)	제어 모듈이 호출하는 하위 모듈의 역할 단순 수행 → 하향식 테스트에 사용
테스트 스위트 (Test Suites)	시스템에 사용되는 테스트 케이스의 집합 (컴포넌트 / 모듈)
테스트 케이스	사용자의 요구사항 준수 여부 확인 목적 테스트 항목 명세서 (입력값, 실행 조건, 기대 결과)
테스트 스크립트	자동화된 테스트 실행 절차에 대한 명세서
목 오브젝트 (Mock Object)	사용자의 행위 조건부 입력 시 계획된 행위를 수행하는 객체



## ■ 데이터 통신을 이용한 인터페이스 구현

- 애플리케이션 영역에서 데이터 포맷을 인터페이스 대상으로 전송하고, 이를 수신 측에서 파싱(Parsing) 후 해석하는 방식

. **JSON** (JavaScript Object Notation) : 속성-값 쌍 (Attribute-Value Pairs) 으로 이뤄진 데이터 오브젝트를 전달하기 위해 사용하는 개방형 표준 포맷  
→ **AJAX**에서 많이 사용 / XML을 대체하는 주요 데이터 포맷  
**JavaScript 사용한 비동기 통신기술**

. **XML** (eXtensible Markup Language) : HTML 문법의 비호환성과 SGML의 복잡성 해결

## ■ 인터페이스 구현 검증 도구

xUnit	Java, C++, .Net(닷넷) 등 다양한 언어 지원 단위 테스트 프레임워크
STAF	서비스 호출, 컴포넌트 재사용 등 다양한 환경을 지원
FitNesse	웹 기반 테스트케이스 설계/실행/결과 확인 지원
NTAF	STAF의 재사용 및 확장성 + FitNesse의 협업 기능 통합 (현재는 폐기됨)
Selenium	다양한 브라우저 및 개발 언어를 지원하는 웹 어플리케이션 테스트
watir	Ruby 언어 기반 어플리케이션 테스트 프레임워크

## ■ 알고리즘

- 선형 검색 : 하나씩 순서대로 비교하며 원하는 결과값을 찾아내는 검색  
- 이진 검색 : 검색 수행 전에 데이터 집합이 정렬되어야 함

## ■ 알고리즘 시간 복잡도

복잡도	알고리즘	설명
$O(1)$	해시 함수	자료 크기 무관 일정한 속도
$O(\log_2 N)$	이진 탐색	로그형 복잡도
$O(n)$	순차 탐색	선형 복잡도 : 입력 자료를 하나씩 처리 (정비례)
$O(N \log_2 N)$	힙 / 합병(병합) / 퀵(평균) 정렬	선형 로그형 복잡도
$O(N^2)$	선택 / 버블 / 삽입 *퀵(최악) 정렬	대부분의 경우 힙/합병 보다 복잡도가 큼 $N^2 > N \log_2 N$ ( $N > 2$ )

\* 퀵 정렬의 경우 평균 복잡도  $O(N \log_2 N)$ 를 따르지만, 최악의 경우는  $O(N^2)$ 를 따름

## ※ 해시함수 종류

중첩법 (폴딩법)	레코드 키를 여러 부분으로 나누고, 나눈 부분의 각 숫자를 더하거나 XOR 한 값을 홈 주소로 사용하는 방식
제산법	레코드키로 해시표의 크기보다 큰 수 중에서 가장 작은소수로 나눈 나머지를 홈 주소로 삼는 방식
기수변환법	키 숫자의 진수를 다른 진수로 변환시켜 주소 크기를 초과한 높은 자릿수는 절단하고, 이를 다시 주소 범위에 맞게 조정하는 방식
숫자분석법	키 값을 이루는 숫자의 분포를 분석하여 비교적 고른 자리를 필요한 만큼 택해서 홈 주소로 삼는 방식

기타 : 제곱법, 무작위 법

## ■ 알고리즘 설계 기법

Divide and Conquer	문제를 최소 단위로 나누어 풀고, 각각을 다시 병합
Dynamic Programming	더 작은 문제의 연장선으로 여기고 과거에 구한 해를 활용
Greedy	현 시점에서 가장 최적의 방법을 선택
Backtracking	모든 조합을 시도하여 문제의 답을 선택 → 문제의 부모 노드로 되돌아간 후 다른 자손 노드 검색

## ■ 통합 개발 환경 (IDE, Integrated Development Environment)

- 개발에 필요한 다양한 툴을 하나의 인터페이스로 통합/제공

Compile : 고급언어(자연어)를 저급언어(기계어)로 변환

Debugging : 프로그램에서 발견되는 버그를 찾아 수정

Deployment : 소프트웨어를 최종 사용자에게 전달

- 이클립스 (IBM) / 비주얼 스튜디오 (MS) / 엑스 코드 (Apple) / 안드로이드 (Google)

## ■ 인스펙션 (Inspection)

- 계획 → 사전교육 → 준비 → 인스펙션 회의 → 수정 → 후속조치  
↑

■ **리팩토링 (Refactoring)** : 겉으로 보이는 동작 변화 없이 내부 구조를 변경 (SW 쉽게 이해하고, 적은 비용으로 수정 가능할 수 있게)

## ■ SW 구현 단계

- 코딩 작업 계획 → 코딩 진행 → 컴파일 (컴퓨터 언어로 변환) → 코드 테스트

## ■ 제품 SW 패키징 시 고려사항

- 보안 고려 / 사용자의 편의성 / 암호화 알고리즘 적용 / 다양한 기기종 연동 고려

## ■ 인터페이스 보안을 위한 네트워크 영역에 적용될 수 있는 Solution

- IPSec / SSL / S-HTTP

## ■ 트리 순회 방법

<b>전위 순회</b> (Pre-order) 	<p>A - B - D - E - H - I - C - F - G - J</p>
<b>중위 순회</b> (In-order) 	<p>D - B - H - E - I - A - F - C - G - J</p>
<b>후위 순회</b> (Post-order) 	<p>D - H - I - E - B - F - J - G - C - A</p>

## ■ 정렬 방법

### ① 선택 정렬 (Selection sort)

- 최소값을 찾아 첫번째 숫자와 위치 교환, 이후 정렬된 값을 제외한 최소 숫자를 정렬되지 않은 숫자 중 첫번째 숫자와 다시 위치 교환, 이를 반복

시작: 

9	6	7	3	5
---	---	---	---	---

 → 

3	6	7	9	5
---	---	---	---	---

 → 

3	5	7	9	6
---	---	---	---	---

  
→ 

3	5	6	9	7
---	---	---	---	---

 → 

3	5	6	7	9
---	---	---	---	---

### ② 삽입 정렬 (Insert sort)

- 두번째 숫자와 첫번째 숫자 비교하여 크기 순으로 정렬, 이후 3번째 숫자를 앞서 정렬된 숫자들 사이에 크기 순에 맞게 재정렬, 이를 반복

시작: 

9	6	7	3	5
---	---	---	---	---

 → 

6	9	7	3	5
---	---	---	---	---

 → 

6	7	9	3	5
---	---	---	---	---

  
→ 

3	6	7	9	5
---	---	---	---	---

 → 

3	5	6	7	9
---	---	---	---	---

### ③ 버블 정렬 (Bubble sort)

- 왼쪽부터 인접한 두 숫자 간 크기 비교하여 위치 교환,  
→ 1 Pass 마다 정렬되는 숫자들 중 가장 큰 숫자를 오른쪽으로 밀어서 배치

시작: 

9	6	7	3	5
---	---	---	---	---

 → 

6	7	3	5	9
---	---	---	---	---

 → 

6	3	5	7	9
---	---	---	---	---

  
→ 

3	5	6	7	9
---	---	---	---	---

 → 

3	5	6	7	9
---	---	---	---	---





## ■ 데이터베이스

- 특정 조직 내 필요한 데이터들의 모임, 공용으로 소유/유지/이용하는 공용 데이터

<b>개념적 설계</b>	독립적인 개념 스키마 모델링 / <b>트랜잭션 모델링</b> / E-R 다이어그램
<b>논리적 설계</b>	종속적인 논리 스키마 설계 / <b>트랜잭션 인터페이스 설계</b> / 정규화 논리적 데이터 베이스 구조로 Mapping / 스키마의 평가 및 정제
<b>물리적 설계</b>	목표 DBMS 종속적인 물리적 구조 데이터로 변환 / 반정규화 오브젝트, 접근방법, 인덱스, 뷰 용량 설계 / <b>트랜잭션 작성</b> → <b>저장 레코드의 형식/순서/접근 경로 설계</b>

## ■ 데이터 모델 구성 요소

- 구조 (Structure): 논리적 개체 간 관계, 데이터 구조, 정적 성질 표현
- 연산 (Operation): 실제 데이터를 처리하는 작업에 대한 명세
- 제약 조건 (Constraint): DB에 저장될 수 있는 실제 데이터의 논리적 제약 조건

## ■ 스키마 (Schema)

<b>개념 스키마</b>	. 사용자와 DB 관리자 관점의 스키마 / 데이터베이스 전체를 정의 → 데이터 개체/관계/제약조건/접근권한/무결성 규칙 명세
<b>내부 스키마</b>	. DB 설계자/개발자 관점의 스키마 . 개념 스키마를 물리적 저장장치에 구현하는 방법을 정의 → 물리적 구조 / 내부 레코드의 물리적 순서
<b>외부 스키마</b>	. 사용자 관점의 스키마 . 사용자, 프로그램마다 다양한 형태의 논리적 구조로 존재

## ■ E-R 다이어그램 표기법

	개체		속성		다중값 속성
	관계		기본키		개체-속성 연결

## ■ 관계형 데이터베이스

- 릴레이션(Relation)이라는 표(Table)로 표현
- 속성(Attribute) = 열 = 필드 / 튜플(Tuple) = 행 = 레코드
- 차수(Degree): 속성의 개수 / 기수(Cardinality): 튜플의 개수
- 도메인(Domain): 하나의 속성에서 취할 수 있는 원자값들의 집합 (ex. 성별: 남/여)

속성 (열)

학번	이름	과목	점수

Atomic value

## ※ 릴레이션 특징

- 튜플과 속성은 유일하며 **순서와 무관**
- 튜플은 삽입/삭제 등에 의해 계속 변함 / 튜플은 서로 상이한 값을 가짐
- 속성의 값은 분해 불가 / 동일할 수 있음
- 튜플을 식별하기 위해 속성(필드)의 일부를 Key로 설정
- 속성은 Null 값을 가질 수 있으나, 기본키에 해당되는 속성은 Null 값을 가질 수 없음

## ■ 키 (Key)

<b>후보키</b> (Candidate Key)	. 기본키로 사용 가능한 속성 / 모든 릴레이션에는 후보키 존재 . 모든 튜플에 대해 유일성/최소성 만족시켜야 함 ※ 유일성: 하나의 키 값으로 하나의 튜플 유일하게 식별 ※ 최소성: 모든 레코드 식별하는데 최소한의 속성으로만 구성
<b>기본키</b> (Primary Key)	. 후보키 중에서 선택되어, 중복된 값과 NULL 값 가질 수 없음 . 유일성/최소성 만족하며 튜플 식별하기 위해 반드시 필요한 키
<b>대체키</b> (Alternate Key)	. 후보키가 둘 이상일 때 기본키를 제외한 나머지 후보키
<b>슈퍼키</b> (Super Key)	. 한 릴레이션 내 속성들의 집합으로 구성된 키 . 모든 튜플에 대해 유일성은 만족하지만, 최소성은 만족 불가
<b>외래키</b> (Foreign Key)	. 다른 릴레이션의 기본키를 참조하는 속성 or 속성들의 집합 . 참조되는 릴레이션 기본키와 대응되어 릴레이션 간 참조 관계

## ■ 무결성 (Integrity)

<b>개체 무결성</b>	. 기본키를 구성하는 어떤 속성도 NULL/중복 값 가질 수 없음 . 기본키의 속성 값이 NULL값이 아닌 원자 값을 갖는 성질
<b>도메인/속성 무결성</b>	. 릴레이션 내 튜플들이 각 속성의 도메인에 지정된 값만 가짐
<b>참조 무결성</b>	. 외래키는 NULL 또는 참조 릴레이션의 기본키 값과 동일 . 릴레이션은 참조할 수 없는 외래키 값을 가질 수 없음
<b>사용자 정의 무결성</b>	. 속성 값들의 사용자가 정의한 제약 조건에 만족
<b>데이터 무결성 강화</b>	. 데이터 특성에 맞는 적절한 무결성을 정의하고 강화 . 강화 방법: 제약조건, 어플리케이션, 데이터베이스 트리거

## ■ 관계 대수 - 원하는 정보의 검색 과정을 정의하는 **절차적 언어**

### ① 순수 관계 연산자

$\sigma$	Select (선택)	조건을 만족하는 튜플들의 부분 집합 (수평 연산)
$\pi$	Project (추출)	속성들의 부분 집합, 중복 제거 (수직 연산)
$\bowtie$	Join (조인)	두 개의 릴레이션을 하나로 합쳐 새로운 릴레이션 형성
$\div$	Division (나누기)	A의 속성이 B의 속성 값을 모두 가진 튜플에서 (A $\supset$ B) B가 가진 속성을 제외한 나머지 속성들만 추출

- . **세타 조인**: 두 릴레이션 속성 값을 비교 후 조건을 만족하는 튜플만 반환
- . **동등 조인**: 조건이 정확하게 '=' 등호로 일치하는 결과를 반환
- . **자연 조인**: 동등 조인의 결과에서 중복된 속성을 제거한 결과를 반환

### ② 일반 집합 연산자

$\cup$	Union (합집합)	두 릴레이션의 합 추출 / 중복은 제거
$\cap$	Intersection (교집합)	두 릴레이션의 중복되는 값만 추출
$-$	Difference (차집합)	A 릴레이션에서 B 릴레이션 간 중복되지 않는 값을 추출
$\times$	Cartesian Product (교차곱)	두 릴레이션의 가능한 모든 튜플의 집합 → 속성/컬럼끼리 더하기 / 튜플끼리는 곱하기

## ■ 관계 해석 - 원하는 정보 자체를 정의하는 **비절차적 언어**

<b>논리 연산자</b>	$\vee$	OR	원자식 간 "또는" 관계로 연결
	$\wedge$	AND	원자식 간 "그리고" 관계로 연결
	$\neg$	NOT	원자식에 대한 부정
<b>정량자</b>	$\forall$	전칭 정량자	모든 가능한 튜플 "For All"
	$\exists$	존재 정량자	어떤 튜플 하나라도 존재 "There Exists"

## ■ 정규화

- 이상(Anomaly) 현상이 발생하지 않도록 **중복성/종속성을 최소화**하기 위한 작업
- **논리적 설계 단계**에서 수행하며, 속성 수가 적은 테이블로 분할되어 관리가 용이해짐
- 데이터 구조 안정성 최대화 / 데이터 삽입 시 릴레이션 재구성 필요 최소화

※ 이상현상 종류 - 삽입 이상: 불필요한 데이터가 함께 삽입  
삭제 이상: 필요한 데이터가 삭제  
갱신 이상: 일부만 수정되어 데이터 불일치 → 정보 모순 발생

## ■ 정규화 과정

<b>제 1 정규형</b>	<b>모든 도메인(Domain)이 원자 값만</b> 으로 되어 있음
<b>제 2 정규형</b>	기본키가 아닌 속성이 기본키에 대한 완전 함수적 종속 만족 <b>부분적 함수 종속을 제거</b> 한 정규형
<b>제 3 정규형</b>	기본키가 아닌 모든 속성이 기본키에 대해 <b>이행적 함수 종속 관계</b> 를 만족하지 않음 ( $X \rightarrow Y, Y \rightarrow Z$ 이면 $X \rightarrow Z$ )
<b>BCNF (보이스/코드)</b>	릴레이션 R에서 <b>모든 결정자가 후보키</b> 인 정규형
<b>제 4 정규형</b>	릴레이션 R에 다치 종속이 성립하는 경우 R의 모든 속성이 A에 함수적 종속 관계를 만족하는 정규형
<b>제 5 정규형</b>	릴레이션 R의 모든 조인 종속이 R의 후보키를 통해서만 성립



## ■ 데이터 사전 (시스템 카탈로그)

- 모든 데이터 객체에 대한 정의나 명세에 관한 정보를 유지/관리하는 시스템 테이블
- 데이터베이스에 대한 데이터인 메타데이터(Metadata)를 저장
- SQL문을 이용하여 내용 검색 가능
- 시스템이 자동으로 생성/유지하며, **일반 사용자가 직접 내용 갱신 불가**
- ※ **데이터 디렉토리 (Data Directory)** : 데이터 사전 접근에 필요한 위치 정보 포함

## ■ 인덱스 (INDEX)

- 데이터 레코드의 빠른 접근/조회 위해 별도로 구성된 순서 데이터
- 데이터베이스에서 테이블 삭제 시 인덱스도 같이 삭제 (**사용자 직접 변경 가능!**)
- 인덱스 생성 (CREATE) / 인덱스 삭제 (DROP)
- 종류 : 클러스터 / 논클러스터 / 트리 및 함수 기반 / 비트맵 / 비트맵 조인 / 도메인

## ■ 트랜잭션

- 데이터베이스 상태를 변환시키는 **하나의 논리적 기능을 수행하기 위한 작업 단위**
- **COMMIT** : 트랜잭션 정상 종료 후 변경된 내용을 DB에 반영하는 명령어
- **ROLLBACK** : 트랜잭션 **비정상 종료** 후 모든 변경 작업을 취소하고 이전 상태로 원복

<b>원자성</b> (Atomicity)	트랜잭션 연산이 정상적으로 수행되거나 (Commit), 아니면 어떠한 연산도 수행되지 않아야 함 (Rollback)
<b>일관성</b> (Consistency)	시스템의 고정 요소는 트랜잭션 수행 전/후로 동일해야 함
<b>독립성</b> (Isolation)	개별 트랜잭션은 다른 트랜잭션의 간섭 및 영향 받지 않아야 함
<b>영속성</b> (Durability)	완료된 트랜잭션 결과는 영구적으로 기록되어야 함

## ■ 뷰 (View)

- 기본 테이블에 기반을 둔 이름을 가지는 **가상 테이블** (기본 테이블과 동일 형태)
- 저장장치 내 가상으로, **논리적으로 존재** (실존하는 **물리적 존재 X**)
- 기본 테이블 및 뷰 삭제 시 해당 테이블/뷰를 기초로 정의된 다른 뷰도 자동 삭제
- ▷ 장점 : **논리적 데이터 독립성 제공** / 데이터 자동 보안 제공 / 데이터 관리 용이
- ▷ 단점 : **독립적 인덱스 보유 불가** / ALTER 변경 불가 / 삽입, 삭제, 갱신, 연산 제약

## ■ 분산 데이터베이스

- 논리적으로는 하나의 시스템에 존재하나, 물리적으로는 연결된 다수의 컴퓨터에 분산되어 있는 데이터 베이스
- 구성요소 : 분산 처리기 (분산된 지역 컴퓨터) / 분산 데이터 베이스 / 통신 네트워크
- 목표 : 위치 투명성 (Location) / 중복 투명성 (Replication) / 병행 투명성 (Concurrency) 분할 투명성 (Division) / 장애 투명성 (Failure)
- **사용자 입장에서 데이터베이스의 위치/중복/병행/분할/장애 여부 인식할 필요 X**

## ■ 로킹 단위 (Locking) : 로킹의 대상이 되는 객체의 크기

- 로킹 : 데이터베이스 병행 제어 위해 트랜잭션(transaction)이 접근하고자 하는 데이터를 잠가(lock) 다른 트랜잭션이 접근하지 못하도록 하는 **병행 제어 기법**
- 대상 : 데이터베이스 / 파일 / 레코드
- . 로킹 단위 ▼ → 로크의 수 ▲ → 로킹 오버헤드 ▲ → 병행성(공유도) ▲
- . 로킹 단위 ▲ → 로크의 수 ▼ → 로킹 오버헤드 ▼ → 병행성(공유도) ▼

## ※ 병행제어 기법 종류

- . **로킹 기법** : 일관성과 무결성을 유지하기 위한 트랜잭션의 순차적 진행을 보장
- . **낙관적 검증** : 일단 트랜잭션을 수행하고, 트랜잭션 종료 시 검증을 수행
- . **타임 스탬프 기법** : 타임 스탬프를 부여해 **보여진 시간에 따라 트랜잭션 수행**
- . **다중버전 동시성 제어(MVCC)** : 타임스탬프를 비교해 직렬가능성이 보장되는 적절한 버전을 선택해 접근하도록 함

## ■ SQL (Structured Query Language) 분류

### ① 데이터 정의어 (DDL : Data Definition Language) → 논리/물리적 데이터 구조 정의

<b>CREATE</b> (생성)	CREATE <b>DOMAIN / SCHEMA / TABLE / VIEW / INDEX</b> → <b>생성</b>
<b>ALTER</b> (변경)	TABLE 이름 변경 → ALTER TABLE / <b>컬럼 추가</b>
<b>DROP</b> (삭제)	DROP <b>DOMAIN / SCHEMA / TABLE / VIEW / INDEX</b> → <b>삭제</b> ※ <b>CASCADE</b> : 참조하는 모든 개체 함께 제거 ※ <b>RESTRICTED</b> : 제거할 요소를 다른 개체가 참조 시 제거 취소

### ② 데이터 조작어 (DML : Data Manipulation Language)

<b>SELECT</b> (검색)	SELECT <b>FROM</b> 테이블명 [WHERE 조건];
<b>INSERT</b> (삽입)	INSERT <b>INTO</b> 테이블명 VALUES 데이터;
<b>DELETE</b> (삭제)	DELETE <b>FROM</b> 테이블명 [WHERE 조건];
<b>UPDATE</b> (변경)	UPDATE 테이블명 <b>SET</b> 속성명 = 데이터 [WHERE 조건];

### ※ SELETE 구문 예시

<b>SELECT</b> [DISTINCT] 필드 이름	SELECT 이름 FROM 학생	이름을 산출 / 학생 테이블에서
<b>FROM</b> 테이블명	WHERE 수학>=80;	수학이 80 이상인 자료만
[ <b>WHERE</b> 조건식]	GROUP BY 학년	학년 필드를 그룹으로 묶어서
[ <b>GROUP BY</b> 필드이름]	HAVING COUNT(*)>=2	레코드 개수가 2이상인 그룹만
[ <b>HAVING</b> 그룹 조건식]	ORDER BY 나이 <b>ASC or</b> <b>생략</b>	나이 오름차순 정렬 (기본)
[ <b>ORDER BY</b> 정렬]	ORDER BY 나이 <b>DESC</b>	나이 내림차순 정렬

### ※ **DISTINCT** : 중복 레코드 제거한 조회 결과 출력

OR	① 부서='경리' <b>OR</b> 부서='영업'		
	② <b>IN</b> ('경리', '영업')		
AND	① 생일>=#2001-1-1# <b>AND</b> 생일<=#2002-12-31#		
	② <b>BETWEEN</b> #2001-1-1# <b>AND</b> #2002-12-31#		
만능문자	LIKE '박%' → 박으로 시작되는 <b>모든 글자</b>		
	LIKE '_은' → 은으로 끝나는 <b>두 글자</b>		
빈 칸	IS NULL	부정문	NOT

### ③ 데이터 제어어 (DCL : Data Control Language)

#### → 데이터 보안/복구/병행수행 제어무결성 유지

<b>COMMIT</b>	<b>수행된 결과를 실제 물리적 디스크로 저장 후 작업의 정상 완료 통보</b>
<b>ROLLBACK</b>	COMMIT 수행되지 않은 작업은 취소하고, 이전 상태로 원복 ※ <b>SAVEPOINT</b> : 트랜잭션 내 ROLLBACK할 저장 위치 지정
<b>GRANT</b>	데이터베이스 사용자에게 사용 권한 부여 → <b>GRANT</b> 권한 리스트 <b>ON</b> 개체 <b>TO</b> 사용자 [ <b>WITH GRANT OPTION</b> ] 부여받은 권한을 다른 사용자에게 재부여
<b>REVOKE</b>	데이터베이스 사용자의 사용 권한 취소 → <b>REVOKE</b> [GRANT OPTION FOR] 권한 리스트 <b>ON</b> 개체 <b>FROM</b> 사용자 다른 사용자에게 권한 부여 가능한 권한을 취소

## ■ 데이터 회복기법

- . **즉시 갱신 기법** : 트랜잭션 실행 상태에서 변경되는 내용을 바로 데이터베이스에 적용  
→ 변경되는 모든 내용을 로그(Log)에 기록하여 장애 시 해당 로그 토대로 복원  
→ Redo / Undo 모두 수행
- . **지연 갱신 기법** : 트랜잭션 수행 후 부분 완료될 때까지는 데이터베이스에 바로 적용하지 않고, 지연시킨 후 부분 완료 시 로그(Log)의 내용을 토대로 저장  
→ 트랜잭션이 실패할 경우 Undo 수행 없이 Redo 만 수행
- . **검사 시점 기법** : 트랜잭션 수행 중간에 검사시점 (Checkpoint) 지정하여  
검사 시점까지 부분 수행 후 완료된 내용을 중간중간 데이터베이스에 저장
- . **그림자 페이징 기법** : 로그 미사용 / 데이터베이스를 동일 크기의 페이지로 분할 후  
각 페이지마다 복사하여 그림자 페이지 보관  
→ 변경 내용은 원본 페이지에만 적용하여 장애 발생 시 해당 페이지 사용/회복



## ■ 프로세스 스케줄링

<b>FIFO</b> (First in First out)	. 먼저 도착한 프로세스가 먼저 처리됨
<b>SJF</b> (Shortest Job First)	. 실행 시간이 짧은 프로세스가 우선 처리 / 가장 낮은 대기 시간
<b>HRN</b> (Highest Response-ratio Next)	. 실행 시간 긴 프로세스에 불리한 SJF 기법 보완 . 실행 시간 짧거나 대기 시간이 긴 프로세스에 우선순위 부여 ※ 우선순위 계산식 = $\frac{\text{대기시간} + \text{서비스시간}}{\text{서비스시간}}$ ( <b>높은 우선순위부터 처리</b> )

## ■ UNIX 운영체제

- 시분할 (Time sharing) 시스템 위한 대화식 운영체제 / 개방형 시스템 (Open system)
- 주로 서버용 컴퓨터에서 사용 / C언어 작성 (이식성, 호환성 높음)
- Multi-user 및 Multi-tasking 모두 지원 (하나 이상의 작업 동시에 수행/처리 가능)
- 트리 구조의 파일 시스템 / 통신망 관리용 운영체제에 적합

## ■ Python 시퀀스 자료

- 리스트(List): 데이터를 연속적으로 저장 가능하며, 필요에 따라 개수 조절 가능
- 튜플(Tuple): 데이터를 연속적으로 저장하지만, 요소의 추가/삭제/변경 불가
- Range: 연속된 숫자를 생성 / 리스트, 반복문에서 자주 사용

## ■ 응집도 (Cohesion)

- 개별 모듈이 독립적인 기능으로 정의되어 있는 정도 / 응집도 ▲ → 품질 ▲

### 높은 응집도

<b>기능적</b> (Function)	모듈 내부의 모든 기능 요소가 단일 문제와 연관되어 수행
<b>순차적</b> (Sequential)	모듈 내 출력 데이터를 다음 활동의 입력 데이터로 사용
<b>통신적</b> (Communication)	동일한 입/출력을 사용하여 서로 다른 기능을 수행
<b>절차적</b> (Procedural)	모듈 내 구성 요소들이 다수 관련 기능을 순차적으로 수행
<b>시간적</b> (Temporal)	특정 시간 내 처리되는 기능을 모아 하나의 모듈로 작성
<b>논리적</b> (Logical)	유사한 성격의 처리 요소들로 하나의 모듈이 형성
<b>우연적</b> (Coincidental)	각 구성 요소들이 서로 관련없는 요소로만 구성

### 낮은 응집도

## ■ 결합도 (Coupling)

- 개별 모듈 간 상호 의존하는 정도 / 결합도 ▼ → 품질 ▲

### 높은 결합도

<b>내용</b> (Content)	서로 다른 모듈 간 자료를 직접 참조/수정
<b>공유</b> (Common)	공유되는 공통 데이터를 여러 모듈이 사용
<b>외부</b> (External)	한 모듈에서 선언한 데이터를 외부의 다른 모듈에서 참조
<b>제어</b> (Control)	제어 신호를 이용한 통신 및 제어 요소를 전달
<b>스탬프</b> (Stamp)	모듈 간 인터페이스로 배열/레코드 등의 자료 구조 전달
<b>자료</b> (Data)	모듈 간의 인터페이스가 자료 요소로만 구성

### 낮은 결합도

## ■ 라이브러리 : 개발 시 자주 사용하는 함수 및 데이터들을 미리 만들어 놓은 집합체

- 표준 라이브러리: 프로그래밍 언어에 기본적 포함, 여러 종류의 모듈/패키지 형태
- 외부 라이브러리: 개발자들이 인터넷에 공유해놓은 기능, 다룬/설치 후 사용 가능

## ■ IEEE 802 LAN 표준안

<b>802.2</b>	논리 링크 제어 (LLC)	<b>802.6</b>	도시형 통신망 (MAN)
<b>802.3</b>	CSMA/CD	<b>802.8</b>	Fiber Optic LANS
<b>802.4</b>	Token Bus	<b>802.9</b>	종합 음성/데이터 네트워크
<b>802.5</b>	Token Ring	<b>802.11</b>	무선 LAN

## ■ OSI 참조 모델

<b>응용 계층</b> (Application)	응용 프로그램이 OSI 환경에 접속 가능한 서비스	Telnet / FTP HTTP / POP SMTP DHCP / SNMP DNS
<b>표현 계층</b> (Presentation)	응용-세션 간 <b>코드/데이터 변환</b> , <b>데이터 암호화/압축</b>	
<b>세션 계층</b> (Session)	컴퓨터 간 세션을 설정/관리/종료하여 적절한 통신 상태 유지	
<b>전송 계층</b> (Transport)	종단(END-to-END) 간 투명한 데이터 전송/전달 주소 설정 / 다중화 / <b>오류 제어</b> / <b>흐름 제어</b> 수행	TCP / UDP
<b>네트워크 계층</b> (Network)	개방 시스템 간 네트워크 연결을 관리 / 데이터 교환 경로 설정(Routing), 트래픽 제어, 패킷 정보 전송	IP / ICMP ARP / RARP
<b>데이터 링크 계층</b> (Data link)	송/수신 간 속도 차이 해결 위한 <b>흐름 제어</b> 기능 프레임 <b>동기화</b> / <b>오류 제어</b> 기능 / 노드 간 프레임 전송	HDLC / PPP LLC
<b>물리 계층</b> (Physical)	전송에 필요한 기계적, 전기적, 기능적 특성을 정의	-

## ■ 응용 계층 프로토콜

<b>Telnet</b>	. 다른 컴퓨터 접속 후 원격 서비스 제공 / 가상의 터미널 기능 수행
<b>FTP</b>	. File Transfer Protocol / 원격 파일 전송 프로토콜 (컴퓨터-인터넷)
<b>HTTP</b>	. Hyper Text Transfer Protocol / WWW 내 HTML 문서 송수신
<b>SMTP</b>	. Simple Mail Transfer Protocol / 전자 우편 교환 서비스
<b>DNS</b>	. Domain Name System / 도메인 네임을 IP 주소로 매핑하는 시스템
<b>SNMP</b>	. Simple Network Management Protocol . TCP/IP 네트워크 관리 프로토콜 (네트워크 기기 정보 전송 규약)

## ■ 전송 계층 프로토콜

<b>TCP</b>	. Transmission Control Protocol / <b>양방향 서비스</b> 제공 . <b>순서 제어/오류 제어/흐름 제어</b> 기능 → 높은 신뢰성 . 프로토콜 헤더는 기본적으로 <b>20~60 Byte</b> . 패킷 단위의 <b>스트림</b> 위주 전달
<b>UDP</b>	. User Datagram Protocol / 비연결형 서비스 제공 . TCP 대비 단순한 헤더 구조로 오버헤드 적고, <b>전송속도 빠름 (제어 X)</b> . 실시간 전송 유리 / 신뢰성보다는 속도가 중요한 네트워크에 활용
<b>RTCP</b>	. Real-time Control Protocol / RTP 패킷의 전송 품질 제어 . 세션에 참여한 각 참여자들에게 주기적으로 제어 정보 전송

## ■ 인터넷 계층 프로토콜

<b>IP</b>	. Internet Protocol / 전송할 데이터에 주소 지정하고 경로를 설정 . 비연결형 데이터그램 방식 / 신뢰성 보장 X / 패킷을 분할 및 병합 . 헤더 체크섬(header checksum) 제공 / 데이터 체크섬은 제공 X
<b>ICMP</b>	. Internet Control Message Protocol / 헤더 8Byte . IP와 함께 <b>통신 간 오류 처리</b> 와 <b>전송 경로 변경</b> 등 제어 메시지 관리
<b>ARP</b>	. Address Resolution Protocol . 호스트 IP주소를 네트워크 접속 장비의 <b>물리적 주소(MAC)</b> 로 바꿈
<b>RARP</b>	. Reverse Address Resolution Protocol . ARP 반대로 물리적 주소를 <b>IP주소로 변환</b> 하는 기능

## ■ 흐름제어 : 원활한 네트워크 흐름을 위해 송수신 간에 패킷의 양/속도를 제어

<b>Stop and Wait</b>	. 수신 측 확인 후 다음 패킷 전송 / 한 번에 하나의 패킷만 전송
<b>Sliding Window</b>	. 수신 측 확인 없이 사전에 정해진 패킷 수만큼 연속적으로 전송 . 한 번에 여러 패킷을 동시에 전송 가능 . 긍정 수신 시 윈도우 크기 증가 / 부정 신호 시 윈도우 크기 감소
<b>Slow start</b>	. 정상 패킷 전송 시 혼잡 윈도우 크기를 패킷마다 1씩 증가시키나 혼잡 현상 발생 시 혼잡 윈도우 크기를 1로 줄임
<b>Congestion Avoidance</b>	. 네트워크 내 패킷의 지연이 너무 높아 트래픽이 붕괴되지 않도록 패킷의 흐름을 제어하는 트래픽 제어



## ■ IPv4 / IPv6 특징

IPv4	IPv6
10.97.135.183 (0~255 = 256개 = 8 Bit)	2544::D6::4B::13F:2C4F 0000~FFFF (각 자리를 <b>콜론</b> 으로 구분)
10진수 (0~9)	16진수 (0~9 + A~F)
8 Bit x 4개 = <b>32 Bit</b>	16 Bit x 8개 = <b>128 Bit</b>
유니, 멀티, <b>브로드캐스트</b>	유니, 멀티, <b>애니캐스트</b>

※ IPv6는 주소의 확장성/용통성/연동성이 높으며, IPv4 대비 전송 속도가 빠름, 인증성/기밀성, 데이터 무결성으로 보안 문제 해결 가능  
패킷 크기 제한 없으며, 등급/서비스별 패킷 구분 가능 → 품질 보장 용이

## ■ 연산자 종류

관계	==	같다	>	크다	<	작다
	!=	같지 않다	>=	크거나 같다	<=	작거나 같다
비트	&	and	모든 비트가 1이면 1			
	^	xor	모든 비트가 같으면 0, 하나라도 다르면 1			
		or	모든 비트 중 하나라도 1이면 1			
	~	not	각 비트의 부정 (0이면 1 산출)			
	<<	-	비트를 왼쪽으로 이동			
	>>	-	비트를 오른쪽으로 이동			
논리	!	not	&&	and		or

## ■ 연산자 우선순위

우선 순위	연산자 분류	연산자	결합 규칙
1	괄호	() [] . → 후위++ 후위--	→
2	단항	++전위 --전위 ~ ! sizeof (type)	←
3	이항	산술 승제	→
4		연산 가감	
5		비트 이동	
6		관계 비교	
7		연산 등가	
8		비트 논리	
9	논리	&&	
10	삼항	조건	←
11	대입	= += -= *= /= %= >= <=	←
		&= ^=  =	
12	순서	,	→

※ 결합규칙: ← 오른쪽에 있는 연산자부터 / → 왼쪽에 있는 연산자부터 순차 계산

■ 예외 처리 (Exception) : 프로그램이 정상적으로 실행되지 않을 때(예외) 이를 대비하여 작성해 놓은 처리 프로세스를 수행

- 원인 : 컴퓨터 하드웨어 문제 / 운영체제 설정 실수 / 라이브러리 손상 / 사용자 입력 실수 / 받아들이지 못하는 연산 / 할당 못하는 기억장치 접근

## ■ 파일 디스크립터 (File Descriptor / File control Block)

- 파일 관리를 위해 시스템이 필요로 하는 정보를 가진 제어 블록 (파일 제어 블록)
- 보통 보조기억장치 내에 저장되어 있다가 해당 파일 Open 시 주기억장치로 이동
- 파일마다 독립적으로 존재 / **사용자가 직접 참조 불가**

## ■ CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance)

- 무선 랜에서 데이터 전송 충돌을 피하기 위해 일정 시간 동안 매체가 비어 있음을 확인 후 데이터 전송

## ■ 기억장치 배치 전략

First Fit	크기 상관없이 빈 영역 중 첫번째 분할 영역에 배치
Best Fit	단편화(남는 저장 공간)를 가장 작게 남기는 분할 영역에 배치
Worst Fit	단편화를 가장 많이 남기는 분할 영역에 배치

## ■ UNIX 구성요소

커널 (Kernel)	. 하드웨어 보호 / <b>프로그램 및 하드웨어 간 인터페이스</b> 역할 . 프로세스 관리, 기억 장치 관리, 파일 관리, 입출력 관리, 데이터 전송 및 변환, 셸 프로그램 실행을 위한 프로세스 및 메모리 관리
셸 (Shell)	. 사용자의 명령 인식/해석 후 커널로 전달, 명령을 수행 (반복적인 명령 프로그램을 만드는 프로그래밍 기능 제공) . 시스템과 사용자 간 인터페이스 역할 . 초기화 파일을 이용해 사용자 환경 설정

※ UNIX Shell 환경 변수 : set, env, printenv, setenv

## ■ Garbage Collector

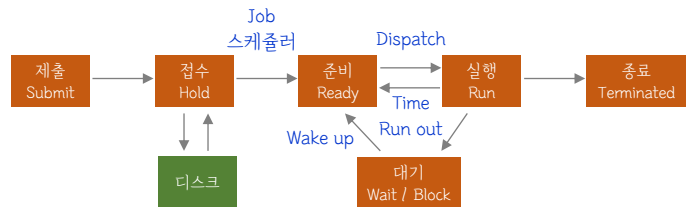
- 변수를 선언만 하고 사용하지 않아 더 이상 사용되지 않은 객체를 제거하는 모듈

## ■ 스크립트 언어

Javascript	. 웹 페이지의 동작 제어 / 클라이언트용 스크립트 언어
Visual Basic	. MS사 개발 언어 / Active X 사용하여 MS사 애플리케이션 제어
ASP	. Active Server Page (MS사 개발 언어) . 서버 측에서 동적으로 수행되는 페이지 만들기 위한 언어
JSP	. Java Server Page (JAVA 기반 서버용 스크립트)
Python	. 귀도 반 로섬(Guido van Rossum) 제작 언어 . 인터프리터 방식 / 객체 지향 언어 / 이식성 높음 / 동적 타이핑
셸 스크립트	. 유닉스/리눅스 기반 셸에서 사용되는 명령어들의 조합 . 컴파일 단계 없어 빠른 실행 속도 / 확장자 : .sh

## ■ 프로세스 (Process)

- 프로세서(CPU)에 의해 처리되는 사용자/시스템 프로그램
- 프로세스 제어 블록 (PCB, Process Control Block)에 프로세스 식별자, 상태 등의 중요 정보를 저장 / 각 프로세스 실행 시 고유 PCB 생성, 작업 완료 후 PCB 제거



※ Dispatch : 준비 상태에서 대기 중인 프로세스가 실행 상태로 전이되는 과정

※ Wake up : 입출력 작업이 완료되어 프로세스가 대기 상태에서 준비 상태로 전이

※ Context Switching : 이전 프로세스의 상태 레지스터 내용 보관하고

(문맥 교환) 다음 프로세스의 레지스터를 적재하는 과정

## ■ 스레드 (Thread) : 프로세스 내 작업 단위 / 프로그램 단위

- 단일: 하나의 프로세스에 하나의 스레드 / 다중: 하나의 프로세스에 복수의 스레드
- 동일 프로세스 환경 내 서로 다른 독립적 다중 스레드 수행 가능

[장점] 한 프로세스 내 복수의 스레드 생성하여 작업 병행성 향상

응답 시간 단축 / 기억 장소 낭비 최소화 / 프로세스 간 통신 향상

운영체제 성능 및 프로그램 처리 효율 향상 / 접근 가능한 기억장치 사용

사용자 수준	. 사용자가 만든 라이브러리를 이용한 스레드 운용 . 속도 빠르나 구현 어려움
커널 수준	. 운영체제의 커널에 의한 스레드 운용 . 구현 쉬우나 속도가 느림





- **교착 상태 (Dead Lock)**: 둘 이상의 프로세스들이 서로 점유하고 있는 자원을 요구하며 무한정 대기하고 있는 상태

#### [교착 상태 필요 충분 조건]

<b>상호 배제</b> (Mutual Exclusion)	한 번에 한 개의 프로세스만이 공유 자원을 사용
<b>점유 및 대기</b> (Hold and Wait)	최소한 하나의 자원을 점유하면서 다른 프로세스에 할당된 자원을 추가로 점유하기 위해 대기하는 프로세스 존재
<b>비선점</b> (Non-preemption)	다른 프로세스에 할당된 자원은 끝날때까지 강제로 선점 불가
<b>환형 대기</b> (Circular Wait)	프로세스들이 원형으로 구성되어 있어 앞/뒤 프로세스 자원을 요구

#### [교착 상태 해결 방법]

<b>예방</b> (Prevention)	. 교착발생 4가지 조건 중 하나 이상을 사전에 제거하여 예방
<b>회피</b> (Avoidance)	. 발생된 교착상태를 적절히 피해가는 방법 ※ <b>은행원 알고리즘 (Banker's Algorithm)</b> : 은행에서 모든 고객의 요구가 충족되도록 현금을 할당하는 데서 유래
<b>발견</b> (Detection)	. 교착상태에 있는 프로세스 및 자원을 발견 / 점검
<b>회복</b> (Recovery)	. 교착상태 유발 프로세스 종료 / 프로세스 내 할당된 자원 선점

#### ■ 페이징 기법 (Paging)

- 가상기억장치에 보관된 프로그램과 주기억장치의 영역을 **동일 크기**로 분할 후 나뉜된 프로그램(페이지)을 동일하게 나뉜된 주기억장치의 영역(페이지 프레임)에 배치하여 실행하는 기법 (내부 단편화 발생 가능성 존재)

- ▶ **작은 페이지 크기**: **페이지 단편화 감소 / 페이지 이동시간 감소 / 워킹 셋 유지**  
**맵 테이블 크기 커지며 매핑 속도 감소 / 입출력 시간 증가**
- ▶ **큰 페이지 크기**: **맵 테이블 크기 작아지며 매핑 속도 증가 / 입출력 시간 감소**  
**페이지 단편화 증가 / 페이지 이동시간 증가 / 불필요 데이터 적재**

#### ■ 페이지 교체 알고리즘

<b>OPT</b>	. 앞으로 가장 오랫동안 사용하지 않을 페이지를 교체 . 페이지 부재 횟수가 가장 적게 발생 / 효율적 교체 알고리즘
<b>FIFO</b>	. First-in First-out . 가장 먼저 들어와 가장 오래 있었던 페이지를 교체
<b>LRU</b>	. Least Recently Used . 최근에 가장 오랫동안 사용하지 않은(오래 전에 사용된) 페이지를 교체
<b>LFU</b>	. Least Frequently Used . 사용 빈도가 가장 적은 페이지를 교체 / 활발한 페이지는 교체 X

#### ■ 스래싱 (Thrashing)

- 프로세스 처리 시간보다 페이지 교체 소요 시간이 더 많아지는 상태  
- 전체 시스템 성능 저하 / 다중 프로그래밍 다발 시 스래싱 발생 → CPU 이용률 저하

[방지 방법] 다중 프로그래밍 적정 수준 유지 / 페이지 부재 빈도 조절 / 워킹 셋 유지  
일부 프로세스 중단 / CPU 성능에 대한 자료의 지속적 관리 및 분석

※ **워킹 셋**: 일정 시간 동안 자주 참조하는 페이지들을 주기억장치에 상주시켜 (Working Set) 페이지 부재 및 교체 현상을 최소화 → 기억장치 사용 안정화

- **Locality**: 프로세스 실행 간 주기억장치 참조 시 일부 페이지만 집중 참조하는 성질
- **시간 구역성**: 한 페이지를 일정 시간 동안 집중적으로 액세스  
→ 반복/순환(Loop), 스택(Stack), 부 프로그램(Sub Routine), 집계 (Totaling)
- **공간 구역성**: 한 페이지 참조 시 인접한 페이지들도 집중적으로 액세스  
→ 배열순회(Array Traversal), 순차적 코드 실행, 같은 영역에 있는 변수 참조

#### ■ Segmentation 기법

- 가상기억장치에 보관된 프로그램을 **다양한 크기**의 논리적 단위로 분할 후 (Segment) 주기억 장치에 배치 및 실행 (외부 단편화 발생 가능성 존재)

※ Paging 기법의 경우, 프로그램을 동일 일정 크기로 분할함

→ 사용자 상호 작용 없이 여러 작업을 미리 정해진 일련의 순서에 따라 일괄적으로 처리

#### ■ 배치 프로그램 필수 요소

<b>대용량 데이터</b>	대량의 데이터 전송/전달/계산 처리 가능
<b>자동화</b>	심각한 오류 상황 제외하고 사용자의 개입 없이 처리 가능
<b>안정성</b>	오류 발생 시 발생 위치/시간 등을 추적 가능
<b>성능</b>	다른 응용 프로그램 수행 방해 없이 지정된 시간 내 처리 가능
<b>견고성</b>	프로그램 처리 시 오류로 인해 중단되는 일 없이 처리 가능

- **재사용**: 이미 개발된 기능을 재구성하여 새로운 시스템/기능 개발에 적용
- 재사용되는 대상은 외부 모듈과의 결합도는 낮고, 응집도를 높여야 함
- 종류: **함수와 객체 / 컴포넌트 / 애플리케이션**

#### ■ 버퍼 오버플로

- 프로그램에서 할당된 메모리의 범위를 넘어선 위치에서 자료를 사용할 때 발생

- 메모리를 다루는 데 오류가 발생하여 잘못된 동작을 하는 프로그램 취약점

#### ■ 변수 작성 규칙

- **영문자, 숫자, \_(underscore)** 사용 가능 / 단, 첫 글자는 숫자 사용 불가 (글자 수 제한 X)
- 대소문자 구분 가능 / 변수 선언 시 문장 끝에 반드시 세미콜론(;) 붙여야 함
- **사용 불가**: 공백 / 특수문자(\* + - / ) / 예약어

#### ■ 접근 제어자

<b>Public</b>	패키지 내/외부 모든 접근 허용
<b>Protected</b>	같은 패키지 내 객체와 상속관계(하위)의 객체만 허용
<b>Default</b>	같은 패키지 내 객체만 허용
<b>Private</b>	현재 객체 내에서만 허용

#### ■ UNIX / LINUX 기본 명령어

cat	파일 내용을 화면에 표시	fsck	파일 시스템 검사/보수
chdir	현재 사용할 디렉터리 위치 변경	getpid	자신의 프로세스 아이디 호출
chmod	파일의 보호 모드 설정	getppid	부모 프로세스 아이디 호출
chown	소유자 변경	ls	디렉터리 내 파일 목록 확인
cp	파일 복사	rm	파일 삭제
exec	새로운 프로세스 수행	wait	상위 프로세스가 하위 프로세스 종료 등의 event 기다림
fork	새로운 프로세스 생성		

#### ■ 언어 별 데이터 타입

C++			JAVA		
정수	short	2 Byte	정수	byte	1 Byte
	int	4 Byte		short	2 Byte
	long	4 Byte		int	4 Byte
	long long	8 Byte		long	8 Byte
실수	float	4 Byte	실수	float	4 Byte
	double	8 Byte		double	8 Byte
	long double	8 Byte	논리	boolean	1 Byte

#### ■ C언어 표준 라이브러리

<b>stdio.h</b>	데이터 입출력 담당	printf, scanf, fprintf, fscanf, fclose
<b>math.h</b>	수학함수	sqrt, pow, abs
<b>string.h</b>	문자열 처리	strlen, strcpy, strcmp
<b>stdlib.h</b>	자료형 변환 / 난수 발생	atoi, atof, srand, rand, malloc, free
<b>time.h</b>	시간 처리	time, clock





## ■ 정보 보안 3대 요소

- 기밀성**: 시스템 내 정보 자원은 인가된 사용자에게만 접근 허용
- 무결성**: 오직 인가된 사용만이 시스템 내 정보 수정 가능
- 가용성**: 인가된 사용자는 권한 범위 내에서 언제든지 자원 접근 가능  
+ 인증 (사용자의 신분 확인) / 부인방지 (송·수신 증거 제공)

## ■ 보안 취약점 점검 분류

- 관리적**: 정보보호 관리체계 보안 통제에 근거하여 취약점 점검
- 기술적**: 서버, 네트워크, PC 보안점검 등을 통한 취약점 점검
- 물리적**: 출입 통제 관리 시스템 및 화재등 관련

## ■ 테일러링(Tailoring) SW 개발 방법론

- 프로젝트 상황/특성에 맞게 정의된 SW 개발 방법론의 절차/사용기법 등을 수정/보완
- 내부적 기준: 목표 환경, 요구사항, 프로젝트 규모(비용/인력), 보유 기술(구성원 능력)

## ■ 소프트웨어 개발 프레임워크

- 개발해야 할 애플리케이션 일부분이 이미 내장된 클래스 라이브러리에 구현
- 동일 로직 반복 최소화 / 재사용성 확대 / 생산성 및 유지보수성 향상

<b>스프링 프레임워크</b> (Spring Framework)	. JAVA 플랫폼을 위한 오픈 소스 경량형 프레임워크 . 동적 웹 사이트 개발 / 전자정부 표준 프레임워크
<b>전자정부 프레임워크</b>	. 우리나라 공공부문 정보화 사업 시 효율적인 정보 시스템 구축 지원 위해 필요한 기능/아키텍처를 제공
<b>닷넷 프레임워크</b> (.NET Framework)	. MS에서 개발한 Windows 프로그램 개발 . 공통 언어 런타임(CLR)이라는 가상 머신 상에서 작동

## ■ 양방향 암호화 방식

	대칭키 / 비밀키 / 개인키	비대칭키 / 공개키
<b>특징</b>	동일한 키로 데이터를 암호화/복호화 블록 (Block) 2 bit 이상 연산 스트림 (Stream) 1 bit 씩 연산	암호화 키는 DB 사용자에게 공개 복호화 키는 비밀키로 관리자만
<b>종류</b>	DES, AES, SEED, ARIA	RC4, LFSR RSA, Diffie-Hellman DSA(이산대수), ECC(타원곡선)
<b>키 개수</b>	N(N-1)/2 개	2N 개
<b>장점</b>	알고리즘 단순 암호/복호화 속도 빠름	키 분배 용이 관리해야 할 키의 수 적음
<b>단점</b>	관리해야 할 키의 수 많음	알고리즘 복잡 암호/복호화 속도 느림

- ※ DES: 미국 NBS에서 발표한 개인키 알고리즘 (블록 크기 64비트 / 키 길이 56비트)
- ※ AES: 미국 표준 기술 연구소 (NIST) 개인키 알고리즘 (블록크기 128/192/256비트)
- ※ RSA: MIT 공개키 암호화 알고리즘 / 소인수분해 어려운 큰 소수 숫자 활용

## ■ 단방향 암호화 방식 - 해쉬 (Hash)

- 해쉬 알고리즘(해쉬 함수)로 볼리며, 해쉬 함수로 변환된 값/키를 해쉬값/키라고 함
- 임의의 길이의 입력 데이터를 받아 고정된 길이의 해쉬 값을 변환한다.
- 종류: SHA 시리즈 / HAVAL / MD4 / MD5 / N-NASH / SNEFRU

## ■ 접근 통제 기술 (Access Control)

<b>임의 접근 통제</b> (DAC) Discretionary AC	. <b>사용자의 신원/신분에 따라</b> 접근 권한 부여 . <b>데이터 소유자</b> 가 접근 통제 권한 지정/제어 . 객체 생성자가 모든 권한 갖고, 다른 사용자에게 허가 . SQL 명령어: GRANT / REVOKE
<b>강제 접근 통제</b> (MAC) Mandatory AC	. <b>주체와 객체의 등급을 비교 후 시스템</b> 이 접근 권한 부여 . DB 객체별로 보안 등급 부여 및 사용자별로 인가 등급 부여 . 자신보다 보안 높은 객체에 읽기/수정/등록 불가하나 등급 같은 객체에는 모두 가능, 자신보다 낮은 객체에는 읽기 가능
<b>역할 기반 접근 통제</b> (RBAC) Role-based AC	. <b>사용자의 역할에 따라</b> 접근 권한 부여 ( <b>중요관리자</b> 가 지정) . 다중 프로그래밍에 최적화

## ▶ 하향식 비용 산정 기법

→ 개인적 / 주관적 판단 가능

- 전문가 감정 기법: 조직 내 두 명 이상의 전문가에게 비용 산정을 의뢰하는 기법
- 델파이 기법: 한 명의 조정자와 여러 전문가의 의견을 종합하여 산정  
→ '전문가 감정 기법'의 주관적 편견 보완

## ▶ 상향식 비용 산정 기법

- 프로젝트 세부 작업 단위 별로 비용 정산 후 전체 비용을 산정하는 방법

### [종류]

- . LOC (Source Line of Code): 코드 라인 총 수 / 생산성 / 개발 참여 인원 등으로 계산  
→ **낙관치(a), 비관치(b), 기대치(c)를 측정/예측하여 비용 산정** 
$$= \frac{a + 4c + b}{6}$$

- . 개발 단계별 인월 수 (Effort Per Task): LOC 기법 보완 / 생명 주기 각 단계별로 산정

## ▶ 수학적 비용 산정

- ① COCOMO (Constructive Cost Model): 보헴 (Boehm) 제안 / 원시코드 라인 수 기반  
→ 비용 견적 강도 분석 및 비용 견적의 유연성이 높아 널리 통용됨  
→ 같은 프로젝트라도 성격에 따라 비용이 다르게 산정

유형	조직형 (Organic)	중,소규모 SW용 / 5만 라인(50KDSI) 이하
	반분리형 (Semi-detached)	30만 라인 (300KDSI) 이하의 트랜잭션 처리 시스템
	내장형 (Embedded)	30만 라인 (300KDSI) 이상의 최대형 규모 SW 관리

- ② PUTNAM: SW 생명주기 전 과정에 사용될 **노력의 분포**를 이용한 비용 산정






Rayleigh Norden 곡선의 노력 분포도를 기초로 함

★ SLIM: Rayleigh-Norden 곡선 / Putnam 모형 기초로 개발된 자동화 추정 도구

- ③ Function Point (FP): SW 기능 증대 요인에 **가중치 부여 후** 합산하여 기능점수 산출  
→ SW 기능 증대 요인: 자료 입력 (입력 양식) / 정보 출력 (출력 보고서) / 명령어 (사용자 질의수) / 데이터 파일 / 인터페이스

## ■ 정보 전송 방식

- 단방향: 한쪽 방향으로만 전송 가능 (TV, 라디오)
- 반이중: 한쪽에서 송신하면 다른 쪽에서는 수신만 가능 (무전기)
- 전이중: 동시에 송/수신 가능 (전화기)

<b>스타형/성형</b> 중앙 집중형		중앙노드와 1:1 (P2P) 연결 / 고장 발견, 유지 보수, 확장이 쉬운 중앙노드의 제어장치가 통신망의 처리능력, 신뢰성 결정
<b>버스형</b>		한 개의 회선에 여러 단말 장치 연결, 회선 양 끝 종단장치 필요 단말 장치 하나가 고장나도 전체 영향 없기에 신뢰성 높음 설치/제거 용이, <b>기밀성 낮고 통신 회선의 제한이 있음</b>
<b>링/루프</b> 원/환형		인접한 단말기를 서로 연결 / 양방향 전송 가능 통신망 하나가 고장 시 전체 통신망 마비 <b>단말 장치 추가/제거, 기밀보호 어려움</b>
<b>트리/계층</b> 분산형		나뭇가지 모양으로 계층적 연결 / 분산처리 시스템 구성 방식 <b>확장이 많으면 트래픽 (통신량)이 과중됨</b>
<b>망/매쉬</b> 그물/완전형		모든 지점의 단말기를 서로 연결 / 빠른 응답 시간, 높은 연결성 통신량이 많을 경우 유리함 (공중 데이터 통신망) 회선 장애 발생 시 다른 경로로 전송 / 높은 보안성 및 안정성 <b>단말 장치 추가/제거 어려움</b> (낮은 확장성)

## ■ 침입 탐지 시스템 (IDS: Intrusion Detection System)

- 시스템의 비정상 사용/오용/남용 등을 실시간 탐지
- . **오용 탐지**: 사전에 정립되어 입력해 둔 공격 패턴 감지 시 통보
- . **이상 탐지**: 평균적인 상태 기준에서 비정상적 행위 또는 자원의 사용 감지 시 통보

### [종류]

<b>HIDS</b> (Host-based)	. 시스템 내부 감시/분석, 내부 변화를 실시간으로 감시하여 외부 침입자 작업 기록 및 추적 → OSSEC, md5deep, AIDE, Samhain
<b>NIDS</b> (Network-based)	. 외부 침입 감시/분석, 네트워크 트래픽 감시 → Snort, Zeek

- IDS 설치 가능 위치: 패킷 라우터로 들어오기 전 / 라우터 뒤 / 방화벽 뒤 / **DMZ\***

※ DMZ: 외부 인터넷에 서비스 제공하는 서버가 위치하는 네트워크



## ■ 서비스 공격 유형

<b>서비스 거부 공격</b> (DOS: Denial of Service)	대량의 데이터를 한 곳의 서버에 집중적으로 전송하여 표적이 되는 서버의 정상적인 기능을 방해
<b>분산 서비스 거부 공격</b> (DDOS)	여러 대의 장비에서 한 곳의 서버에 분산 서비스 공격 수행 종류 : Trinoo, Tribe Flood Network, Stacheldraht
<b>Ping of Death</b>	Ping 명령을 전송할 때 <b>허용범위 이상의 ICMP 패킷</b> 을 전송하여 대상 시스템의 네트워크를 마비
<b>Ping Flood</b>	과도한 ICMP 메시지에 의한 응답 과다로 시스템 에러 유발
<b>Smurfing (스머핑)</b>	<b>IP / ICMP 특성</b> 을 악용하여 엄청난 양의 데이터를 한 사이트에 집중적으로 보내 네트워크를 붕괴 상태로 만들
<b>TearDrop</b>	<b>Fragment number 값을 변형</b> 하여 수신측에서 패킷 조립 시 오류로 인한 과부하로 시스템 다운을 유도
<b>LAND Attack</b>	<b>송/수신 IP 주소를 모두 타겟 IP주소로 하여</b> 자기 자신에게 무한히 응답하게 하는 공격
<b>Evil twin attack</b>	실제 존재하는 동일한 이름의 가짜 무선 WIFI 신호를 송출하여 로그온한 사람들의 계정 정보나 신용 정보 등을 빼내는 기법
<b>Switching Jamming</b>	위조된 매체 접근 제어(MAC) 주소를 지속 보내 스위치 MAC 주소를 혼란시켜 더미 허브 (Dummy Hub)처럼 작동

## ■ 정보 보안 침해 공격

해킹	시스템에 침입해 정보를 수정하거나 빼내는 행위
크래킹	시스템에 침입해 정보를 파괴하거나 변경하는 행위
좀비 PC	악성코드에 감염되어 다른 컴퓨터를 조종하는 행위
C&C 서버	해커가 감염된 좀비 PC에 명령을 내리고 악성코드를 제어하기 위한 용도로 사용하는 서버
웜(Worm)	다른 컴퓨터의 취약점을 이용하여 스스로 전파하거나 메일로 전파되며 스스로를 증식
트로이목마	정상적인 프로그램으로 가장하여 숨어 있는 바이러스 (복제 X)
백도어	보안이 제거된 비밀통로로 무단 접근을 위한 통로(뒷문) ★탐지 방법 : 무결성 검사, 로그 분석, SetID 파일, 열린 포트 검사
랜섬웨어	내부 문서 파일 등을 암호화해 사용자가 열지 못하게 하고 이를 인질로 금전을 요구하는 데 사용되는 악성 프로그램
제로 데이 공격	발견된 취약점의 존재를 공표하기 전에 해당 취약점으로 이용한 보안 공격
스니핑	패킷을 엿보면서 계정 정보(ID/PW)를 가로채는 행위
스푸핑	검증된 사람이 네트워크를 통해 데이터를 보낸 것처럼 데이터를 변조하여 접속을 시도하는 일종의 속임수
피싱	악의적 사용자가 지인/유명인 사칭하여 가짜 사이트로 유인한 후 로그인한 사용자의 계정 정보 및 신용 정보 탈취
키로거	키 입력 캐치 프로그램을 이용하여 개인정보를 빼내어 악용하는 행위

## ■ 정보 보안 솔루션

<b>방화벽</b> (Firewall)	내부 → 외부로 나가는 패킷은 그대로 통과 외부 → 내부로 유입되는 패킷은 인증된 패킷만 통과
<b>웹 방화벽</b> (Web Firewall)	일반 방화벽은 탐지 불가능한 SQL 삽입 공격, XSS 등 웹 기반 공격을 방어하는 목적으로 웹 서버에 특화된 방화벽
<b>침입 탐지 시스템</b> (IDS)	시스템의 비정상적인 사용/오용/남용 등을 실시간 탐지
<b>침입 방지 시스템</b> (IPS)	방화벽 + 침입 탐지 시스템 비정상적 트래픽은 능동적으로 차단하고 격리
<b>데이터 유출 방지</b> (DLP)	내부 정보의 외부 유출을 방지 (Data Loss Prevention) 사내 직원이 사용하는 시스템의 모든 정보 탐지/통제
VPN (가상 사설 통신망)	공중 네트워크 암호화 기술을 이용하여 사용자가 마치 자신의 전용 회선을 사용하는 것처럼 하게 함
NAC (Network Access Control)	네트워크에 접속하는 내부 PC의 MAC 주소를 IP 관리 시스템에 등록한 후 일관된 보안 관리 기능을 제공
ESM (Enterprise Security Management)	상기 보안 솔루션에서 발생한 로그 기록을 통합하여 관리 → 종합적인 보안 관리 체계 수립 가능
SDP	‘블랙 클라우드’라 불림 (Software Defined Perimeter) GIG 네트워크 우선권에 따라 DISA에서 수행한 작업에서 발전

## ■ 코드 오류

<b>생략 오류</b> (Omission error)	입력시 한 자리를 빼놓고 기록	ABCD → ABD
<b>필사 오류</b> (Transcription error)	임의의 한 자리를 잘못 기록	ABCD → ABFD
<b>전위 오류</b> (Transposition error)	입력 시 좌우 자리를 바꿔어 기록	ABCD → ACBD
<b>이중 오류</b> (Double transposition)	전위 오류가 두 가지 이상 발생	ABCD → CABD
<b>추가 오류</b> (Addition error)	입력시 한 자리 추가로 기록	ABCD → ABCDE
<b>임의 오류</b> (Random error)	다른 오류가 두 가지 이상 결합	ABCD → ACBDF

## ■ ISO 12207 생명주기

<b>기본 생명주기</b>	획득, 공급, 개발, 운영, 유지보수 프로세스
<b>지원 생명주기</b>	문서화, 형상관리, 품질보증, 검증, 확인, 합동검토, 감사
<b>조직 생명주기</b>	관리, 기반구조, 개선, 교육훈련

## ■ CMMI(Capability Maturity Model Integration) : SW 개발 조직의 업무 능력 평가 모델

<b>1) 초기</b>	프로세스 X	작업자 능력에 따라 성공 여부 결정
<b>2) 관리</b>	규칙화 프로세스	특정한 프로젝트 내의 프로젝트 정의/수행
<b>3) 정의</b>	표준화 프로세스	조직의 표준 프로세스 활용 업무 수행
<b>4) 정량적 관리</b>	예측 가능 프로세스	프로젝트 정량적 관리/통제
<b>5) 최적화</b>	지속 개선 프로세스	프로세스 역량 향상 위한 지속적인 개선

## ■ SPICE (ISO 15504) → 6단계 / 소프트웨어 처리 개선 및 능력 평가 기준

(Software Process Improvement and Capability Determination)

<b>1) 불완전</b>	프로세스가 구현되지 않거나 목적 달성 불가
<b>2) 수행</b>	목적은 달성하지만, 계획/축적 불가
<b>3) 관리</b>	프로세스 수행이 계획되고 관리됨
<b>4) 확립</b>	표준 프로세스를 이용하여 계획/관리
<b>5) 예측 가능</b>	표준 프로세스 능력에 대해 정량적 이해/성능 예측
<b>6) 최적</b>	정의된 프로세스와 표준 프로세스가 지속적으로 개선

## ■ 세션 하이재킹 (Session Hicacking)

- 서버에 접속하고 있는 클라이언트들의 세션 정보를 가로채는 공격 방법
- 정상 연결을 Reset 패킷을 통해 종료시킨 후 재연결 시 침입자에게 연결
- 탐지 방법 : 비동기화 상태 탐지, ACK Storm 탐지, 패킷 유실 탐지, 예상치 못한 접속의 리셋 탐지

## ■ 블루투스 관련 공격

<b>블루버그</b> (Bluebug)	. 블루투스 장비 간 취약한 연결 관리를 이용 (원격조정)
<b>블루재킹</b> (Bluejacking)	. 블루투스를 이용해 스팸 메시지를 익명으로 전파
<b>블루스나프</b> (Bluesnaf)	. 인증없이 정보 교환 가능한 OPP를 이용하여 정보 열람
<b>블루프린팅</b> (Blueprinting)	. 타겟 대상이 될 블루투스 장비를 검색

## ■ 소프트웨어 개발 방법론

<b>구조적 방법론</b>	. 정형화된 분석 절차에 따라 사용자 요구사항을 파악 . 분할과 정복 원리 적용, 쉬운 이해/검증 가능한 코드
<b>정보공학 방법론</b>	. 정형화된 기법들을 상호 통합/적용하는 자료 중심 방법론
<b>컴포넌트 기반 방법론</b> (CBD: Component Based Development)	. 기존의 컴포넌트 조합하여 하나의 새로운 시스템 구현 . 컴포넌트 재사용 / 확장성 보장 / 유지보수 비용 최소화



- **Secure OS** : 기존 운영체제(OS)에 보안 기능을 갖춘 커널 이식한 운영체제  
 - 보안 기능 : 식별 및 인증 / 임의적 접근통제, 강제적 접근통제 / 객체 재사용 보호

#### ■ 네트워크 연결 방식

<b>DAS</b> (Direct Attached Storage)	. 서버와 저장장치를 전용 케이블로 <b>직접 연결</b> (외장하드) . 속도 빠름 / 설치, 운영 간편 / 구축 및 유지보수 비용 저렴 . 다른 서버에서 접근 및 파일 공유 불가 (확장성/유연성 ↓)
<b>NAS</b> (Network Attached Storage)	. 서버와 저장장치를 <b>네트워크를 통해</b> 연결 . 다른 서버에서도 접근 및 파일 공유 가능 (장소 무관) → DAS에 비해 확장성/유연성 높음 . 접속 증가 시 성능 저하
<b>SAN</b> (Storage Area Network)	. DAS 빠른 처리 + NAS 파일 공유성 장점 혼합 방식 . <b>광 채널 스위치</b> 를 이용하여 처리 속도 빠름 . 여러 개의 저장장치/백업장치 단일화 가능 → 확장성/유연성 높으나 초기 구축 비용 높음

#### ■ 보안 관련 용어

<b>BaaS</b> (Blockchain as a Service)	. <b>블록체인 앱 개발 환경</b> 을 클라우드 기반으로 제공 . 노드 추가/제거가 용이, 다른 블록체인 기술 공유
<b>Honeypot</b>	. 비정상적 접근 탐지 위해 <b>침입자를 속여</b> 실제 공격을 당하 는 것처럼 보이고, 그 사이 공격 수법/정보를 수집
<b>OWASP</b>	. <b>웹 정보 노출/악성코드</b> 등을 연구하는 <b>비영리 단체</b> . 10대 보안 취약점을 3~4년에 한번씩 발표/공유
<b>TCP Wrapper</b>	. 외부 컴퓨터의 접속 인가 여부를 점검하여 접속 허용/거부 . 사용자 ID 및 로그 불법 조화를 방지하기 위한 방화벽 역할
<b>DPI</b> (Deep Packet Inspection)	. <b>OSI 7 layer 전 계층</b> 의 프로토콜/패킷 내부 콘텐츠 파악하여 침입 시도/해킹 탐지, 트래픽 조정하는 패킷 분석 기술
<b>Rootkit</b>	. 권한이 없는 사용자가 접근할 수 없는 영역에 접근하여 시스템을 제어하도록 설계된 <b>악성 소프트웨어의 모음</b> . 해킹에 사용되는 기능들을 제공하는 프로그램들의 모음

#### ■ DB 관련 기술

<b>Hadoop</b>	. 오픈 소스 기반 분산 컴퓨팅 플랫폼 (가상화된 대형 스토리지) . 자바 소프트웨어 프레임워크 : 구글, 야후 등에 적용
<b>Tajo</b>	. Apache Hadoop 기반 분산 데이터 웨어하우스 프로젝트
<b>MapReduce</b>	. <b>대용량 데이터 분산 처리</b> 위한 프로그래밍 모델 (구글 개발) . 데이터 분류로 묶고(Map) 중복 데이터 제거(Reduce)하는 작업
<b>Data Mining</b>	. 사용자의 요구에 따라 유용한 정보를 발견하기 위한 기법 → <b>데이터 변수 간 상호관계 규명/패턴화</b> 후 데이터 추출 효율화
<b>OLAP</b> (Online Analytical Processing)	. 다차원 데이터에서 통계적 요약정보를 분석 후 의사결정에 활용 . Roll-up, Drill-down/through/across, Pivoting, Dicing

#### ■ 소프트웨어 정의 기술 (SDE: Software-defined Everything)

<b>소프트웨어 정의 네트워킹</b> (SDN: Software defined Networking)	. 네트워크를 제어부/데이터 전달부로 분리 → 보다 효율적으로 네트워크 제어/관리 . 기존 네트워크에 영향 주지 않으면서 특정 서 비스의 전송 경로 수정 통해 인터넷 문제 처리
<b>소프트웨어 정의 데이터 센터</b> (SDDC: Software Defined Data Center)	. 데이터 센터를 가상화하여 인력 개입 없이 SW만으로 관리/제어 가능한 데이터 센터
<b>소프트웨어 정의 스토리지</b> (SDS: Software Defined Storage)	. 데이터 스토리지를 가상화하여 여러 스토리지 를 하나처럼 관리 / 또는 반대로도 가능

#### ■ 데이터 보안 약점

<b>SQL 삽입 (Injection)</b>	사용자 입력값 등 외부 입력 값이 SQL 쿼리에 삽입되어 공격
<b>크로스사이트 스크립트 (XSS)</b>	검증되지 않은 외부 입력 값에 의해 브라우저에서 악의적인 스크립트 실행/유도하여 방문자 정보유출/비정상적 기능 유발
<b>운영체제 명령어 삽입</b>	운영체제 명령어 입력값이 적절한 사전검증 없이 사용되어 공격자가 운영체제 명령어 조작
<b>경로 조작 및 자원 삽입</b>	데이터 입출력 경로 조작하여 시스템이 보호하는 서버 자원에 무단으로 접근하고 수정/삭제
<b>위험한 형식 파일 업로드</b>	악의적 명령어 포함된 스크립트 파일을 업로드하여 시스템에 손상 및 제어
<b>신뢰되지 않는 URL 주소 자동접속</b>	입력값의 사이트 주소를 조작하여 방문자를 피싱 사이트로 유도
<b>메모리 버퍼 오버플로</b>	할당된 메모리 범위를 초과한 상태에서 자료를 읽거나 쓸 때 발생하여 프로그램 오작동 / 악의적 코드 실행시켜 시스템 공격

#### ■ 경로 제어 프로토콜 (Routing Protocol)

<b>RIP</b> (Routing Information Protocol)	. 거리 벡터 라우팅 프로토콜 / 최대 홉수 15 . 최단 경로 탐색으로 Bellman-Ford 알고리즘 사용 . EGP보다는 IGP에 해당 / 소규모 네트워크 환경 적합
<b>OSPF</b> (Open Shortest Path First Protocol)	. RIP 단점 개선 목적 / 대규모 네트워크에 널리 사용 . 최단 경로 라우팅 지원 (실시간 노드 간 거리, 링크 상태 반영) . 다익스트라 (Dijkstra) 알고리즘 사용

- **인증 (Authentication)** : 사용자의 정보를 확인하고 접근 권한을 검증하는 보안 절차  
 - 주요 유형 : 지식 기반 / 소유 기반 / 생체 기반 / 위치 기반

#### ■ Secure SDLC (Software Development Life Cycle)

- 기존 SDLC에 보안 강화 목적 프로세스 추가 → 보안 관련 비용 최소화

<b>CLASP</b>	. SDLC 초기 단계 보안 강화 목적 개발 방법론 (Secure Software) . 활동 중심/역할 기반의 프로세스 구성 (현재 운용 시스템에 적합)
<b>SDL</b>	. 기존 SDLC 개선 (MS사에서 개발) / 나선형 모델 기반
<b>Seven Touchpoints</b>	. SDLC 각 단계에 관련된 7개의 보안 강화 활동 수행 . <b>SW 보안의 모범 사례</b> 를 SDLC에 통합한 방법론

#### ■ 프로젝트 일정 관리

<b>PERT</b> (Program Evaluation and Review Technique)	. 프로젝트 작업 상호관계를 네트워크로 표현 . 원 노드(작업)와 간선(화살표)으로 구성 → 간선에는 각 작업별 낙관치/기대치/비관치를 기재
<b>CPM</b> (Critical Path method)	. 노드(작업) / 간선(작업 전후 의존 관계) / 박스(이정표) 구성 . 간선(화살표)의 흐름에 따라 작업 진행
<b>간트차트</b>	. 각 작업의 시작/종료 일정을 막대 도표를 이용하여 표현 . 시간선(Time-line) 차트 (수평 막대 길이 = 작업기간) . 작업 경로는 표현 불가 / 계획 변화에 대한 적응성이 낮음



## ■ IT 용어

Stack guard	Stack 상 일정 주소 번지에 프로그래머가 유도하는 Canary를 심고, 스택이 붕괴/변조된 경우에 오버플로우 상태로 가정하여 Canary 체크 후 프로그램 실행을 비정상적으로 중단시키는 방법
Docker	컨테이너 기술을 자동화하여 쉽게 사용하는 오픈소스 프로젝트 SW 컨테이너 안에 응용 프로그램 배치 자동화 역할
Cipher Container	JAVA에서 암호화/복호화 기능을 제공하는 컨테이너
Scytale	암호화 기법으로 문자열의 위치를 바꾸는 방법
TensorFlow	2015년 공개된 구글 브레인 팀의 기계 학습 오픈소스 SW Library
One Seg	일본/브라질에 상용 중인 디지털 TV 방송 기술
Foursquare	위치 기반 소셜 네트워크 서비스
PaaS-Ta	국내 IT 서비스 경쟁력 강화 목적 개발된 개방형 클라우드 인프라 제어, 관리/실행/개발/서비스/운영 환경으로 구성
VLAN (Virtual LAN)	물리적 배치와 상관없이 논리적으로 LAN 구성하는 기술 접속된 장비들의 성능향상 / 보안성 증대 효과
SSO (Single Sign On)	한 번의 로그인으로 다른 사이트 로그인도 허용하는 시스템
MQTT	TCP/IP 기반 네트워크에서 발행-구독 기반의 메시징 프로토콜 푸시기술 기반 경량 메시지 전송 프로토콜 (IBM 개발 주도)
Salt	동일한 패스워드들을 다른 암호 값으로 저장되도록 덧붙이는 무작위의 값 → 같은 패스워드임에도 다른 결과 산출
SSH (Secure Shell)	서로 연결된 컴퓨터 간 원격 명령실행 및 셸 서비스 수행 키를 통한 인증은 클라이언트의 공개키를 서버에 등록해야 함 전송 데이터는 암호화되며, 기본적으로 22번 포트 사용
N-screen	N개의 서로 다른 단말기에서 동일콘텐츠를 자유롭게 이용
ASLR	프로그램 실행마다 스택/힙/라이브러리 주소를 랜덤화하여 공격자로 하여금 메모리 상 주소 예측을 어렵게 함
라우터 (Router)	서로 다른 네트워크 대역에 있는 호스트를 상호간에 통신할 수 있도록 해주는 네트워크 장비
nmap	서버에 열린 포트 정보를 스캐닝하여 보안취약점을 찾는 도구
Tripwire	크래커가 침입하여 백도어를 만들어 놓거나, 설정파일을 변경했을 때 분석하는 도구
Smart Grid	정보 기술을 활용하여 전력망 지능화, 고도화함으로써 고품질의 전력서비스를 제공하고 에너지 이용 효율을 극대화
서비스 지향 아키텍처 (SOA)	구성 계층 : 표현 / 업무 프로세스 / 서비스 중간 / 애플리케이션 / 데이터 저장
Digital twin	물리적 사물을 가상화하여(twin) 실제 자산의 특성 정보를 구현 → 자산 최적화 / 돌발사고 최소화 / 생산성 증가
Mashup	웹에서 제공하는 정보/서비스를 이용하여 새로운 SW 제작
Mesh Network	차세대 이동통신, 홈네트워킹 등 특수 목적을 위한 새로운 방식의 네트워크 기술 / 대규모 네트워크 생성에 최적화
PICONET	여러 개의 독립된 통신장치가 블루투스 기술이나 UWB 통신 기술을 사용하여 통신망을 형성하는 무선 네트워크 기술

이 자료는 대한민국 저작권법의 보호를 받습니다. 작성된 모든 내용의 권리는 작성자에게 있으며, 작성자의 동의 없는 사용이 금지됩니다. 본 자료의 일부 혹은 전체 내용을 무단으로 복제/배포하거나 2차적 저작물로 재편집하는 경우, 5년 이하의 징역 또는 5천만원 이하의 벌금과 민사상 손해배상을 청구합니다.