

# 01 소프트웨어 공학 개념

## Section 1. 소프트웨어 공학

### 1. 소프트웨어 공학(Software Engineering)

#### (1) 소프트웨어 공학의 정의

- 소프트웨어 위기를 극복하고 효율적으로 품질 높은 소프트웨어를 개발하기 위한 학문
- 소프트웨어를 개발하는데 있어서, 어떻게 개발할지, 무엇을 개발할지와 같은 방법 도구, 이론을 모두 포함한 포괄적인 개념

#### (2) 소프트웨어의 위기의 원인

- 소프트웨어 특성에 대한 이해 부족
- 소프트웨어 관리 방법론 부재
- 올바른 설계 없이 프로그래밍에만 치중
- 소프트웨어 개발에 대한 전문적 교육 부족
- 작업일정과 비용의 추정치가 부정확

#### (3) 소프트웨어의 위기의 결과

- 개발 인력의 부족과 인건비 상승
- 소프트웨어 성능 및 신뢰성 부족
- 개발 기간 및 비용의 증가
- 소프트웨어 품질저하 및 유지보수 비용 증가
- 소프트웨어의 생산성 저하

### 2. 소프트웨어 공학의 3R

#### (1) 소프트웨어 공학의 3R의 정의

- 완성된 소프트웨어를 기반으로 역공학, 재공학, 재사용을 통해 소프트웨어의 생산성을 극대화 하는 기법

#### (2) 소프트웨어 3R의 필요성

- 소프트웨어 유지보수 효율성 향상 및 비용 절감
- 소프트웨어 개발 생산성 향상
- 소프트웨어 이해, 변경, 테스트 용이
- 소프트웨어 변경 요구사항에 대한 신속한 대응
- 소프트웨어 위기 극복

#### (3) 역공학(Rreverse Engineering)

- 기존 개발된 시스템을 CASE도구를 이용하여 사양서, 설계서 등의 문서로 추출하는 작업
- 개발 단계를 역으로 올라가 기존 개발된 시스템의 코드나 데이터로부터 설계 명세서나 요구 분석서 등을 도출하는 작업

- 역공학의 특징
  - 상용화되거나 기 개발된 소프트웨어의 분석을 도와줌
  - 기존 시스템의 자료와 정보를 설계 수준에서 분석 가능해 유지보수성 향상
  - 기존 시스템 정보를 Repository에 보관하여 CASE의 사용을 용이하게 함

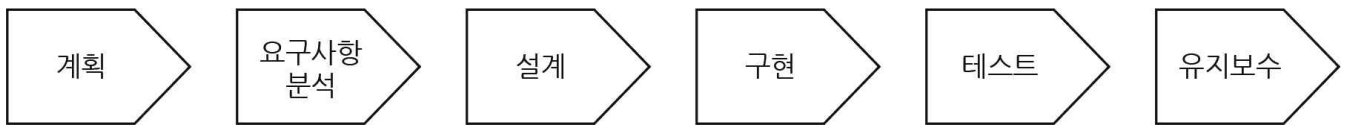
#### (4) 재공학(Re-engineering)

- 기존 시스템을 널리 사용되는 프로그래밍 표준에 맞추거나 고수준의 언어로 재구성하고, 이기종에서 사용할 수 있도록 변환하는 작업
- 재공학의 특징
  - 현 시스템의 유지보수성 향상
  - 시스템의 이해와 변형을 용이하게 하며, 유지보수 비용 및 시간 절감
  - 표준 준수 및 CASE의 사용 용이

#### (5) 재사용(Reuse)

- 이미 개발되어 그 기능, 성능 및 품질을 인정받았던 소프트웨어의 전체 또는 일부분을 다시 사용
- 재사용의 특징
  - 소프트웨어 생산의 TCO (Total Cost Overhead) 절감
  - 높은 품질의 소프트웨어 생산을 위한 공유 및 활용 효과
- 재사용의 범위
  - 함수와 객체 재사용 : 클래스나 함수 단위로 구현한 소스코드를 재사용
  - 컴포넌트 재사용
  - 애플리케이션 재사용
- 재사용 방법
  - 합성 중심(Composition Based) : 전자 칩과 같은 소프트웨어 부품, 즉 블록(모듈)을 만들어서 끼워 맞추어 소프트웨어를 완성시키는 방법
  - 생성 중심(Generation Based) : 추상화 형태로 쓰여진 명세를 구체화하여 프로그램을 만드는 방법

### 3. 소프트웨어 개발 단계



#### (1) 계획

- 무엇을 개발할 것인지 명확하게 정의
- 개발 범위를 결정
- 시스템의 성격을 파악하여 비용과 기간을 예측

#### (2) 요구사항 분석(Requirements Analysis)

- 개발할 소프트웨어의 기능과 제약조건, 목표 등을 고객과 함께 정의
- 요구사항의 정확한 이해 및 요구사항 유도
- 과다하거나 불필요한 요구사항에 대한 협상 및 조율
- 요구사항의 적합성 검토 및 향후 예측
- 현재 실행 환경에 대한 분석

#### (3) 소프트웨어 설계(Design)

- 시스템이 어떻게 동작하는지를 정의
- 요구사항 분석 단계에서 산출된 요구사항을 기준으로, 입력자료, 처리내용, 출력자료 등을 정의
- 설계 구분
  - 시스템 구조 설계 : 모듈간의 관계와 구조 설계
  - 프로그램 설계 : 각 모듈의 처리 절차나 알고리즘 설계
  - 사용자 인터페이스 설계 : 사용자가 시스템을 사용하기 위해 보여지는 부분을 설계

#### (4) 구현(Development)

- 프로그래밍 언어를 이용하여 실제로 프로그램을 작성
- 코딩과 디버깅이 이루어지며, 단위(모듈) 테스트를 진행

#### (5) 테스트(Testing)

- 구현된 소프트웨어가 요구사항을 만족하는지 검사
- 실행 결과가 예상 결과와 맞는지 검사하고 평가
- 테스트 계획, 통합 테스트 결과서 등을 작성

#### (6) 유지보수(Maintenance)

- 소프트웨어를 사용하며 문제점을 수정하고, 새로운 기능을 추가
- 소프트웨어를 좀 더 발전시키는 단계

## Section 2. 소프트웨어 개발 방법론

### 1. 소프트웨어 개발 방법론 종류

#### (1) 구조적 방법론

- 절차지향 소프트웨어 개발 방법론
- 제한된 구조에서 코드 생성 및 순차적 실행
- 구조적 방법론 기본 개발 과정

과정	설명
요구사항 분석	- 고객의 요구사항을 끌어내어 명세화 하는 과정
구조적 분석	- 고객이 원하는 기능/환경/데이터를 종합하여 데이터 흐름도 작성
구조적 설계	- 모듈 중심 설계 과정
구조적 프로그래밍	- 순차, 선택, 반복의 논리 구조 구성으로 프로그램 작성

- 구조적 방법론 구성요소 : 데이터 흐름도(DFD), 자료사전(DD), 상태전이도(STD), 소단위 명세서(Minispec)

#### (2) 정보공학 방법론

- 기업의 주요 부분을 계획, 분석, 설계, 구축에 정형화된 기법들을 상호 연관성 있게 통합, 적용하는 데이터 중심 방법론
- 빠른 결과물 확인이 가능하며 단순 S/W 개발이 아닌 기업의 경영전략에 초점을 둔다.
- 정보공학 방법론 기본 개발 과정

과정	설명
정보전략계획 수립단계	- 현행 업무프로세스와 시스템을 분석하고 미래 아키텍처와 전략계획을 수립
업무영역 분석단계	- 기업의 업무 현황을 분석해서 개념 수준의 데이터와 프로세스를 설계하는 업무분석 단계 - 데이터 모델링 : ERD - 프로세스 모델링 : 프로세스 계층도(PHD), 프로세스 의존도(PDD), 자료흐름도(DFD)
시스템 설계단계	- 실질적으로 시스템을 설계하는 단계 - 논리적 ER 다이어그램으로 데이터를 설계하고 분할 다이어그램, 액션 다이어그램, 의존 다이어그램을 사용해 프로세스를 설계
시스템 구축단계	- 설계명세서를 토대로 데이터베이스를 생성하고, 소스코드를 구현한다.

#### (3) 객체지향 개발 방법론

- 현실세계의 개체(Entity)를 속성(Attribute)과 메서드(Method)형태로 표현
- 객체, 클래스 간의 관계를 식별하여 설계 모델로 변환하는 방법론
- 분석과 설계, 구현의 전 과정을 객체 중심으로 개발
- 전체 프로세스 방향성 유지와 상속에 의한 재사용성 향상
- 특징 : 캡슐화, 정보은닉, 상속, 다형성, 추상화

#### (4) CBD( Component Based Development ) 분석 방법론

- 재사용 가능한 컴포넌트의 개발 또는 상용 컴포넌트를 조합해 어플리케이션 개발
- 새로운 기능 추가가 쉬운 확장성
- 생산성 및 품질이 향상
- 시스템 유지보수 비용 최소화

#### (5) 애자일 방법론

- 기존 방법론들이 절차를 중시한 나머지 변화에 빠른 대응을 할 수 없는 단점 개선을 위해 등장
- 애자일 방법론 종류 : XP( eXtreme Programming ), SCRUM, FDD, Crystal 방법론 등
- 애자일 선언문

공정과 도구보다 개인과 상호작용을  
포괄적인 문서보다 작동하는 소프트웨어를  
계약 협상보다 고객과의 협력을  
계획을 따르기보다 변화에 대응하기를

왼쪽에 있는 것들도 가치가 있지만,  
우리는 오른쪽에 있는 것들에 더 높은 가치를 둔다.

#### (6) 개발 방법론 선택 기준

- 프로젝트 특성 및 규모
- 프로젝트 참여자의 수준
- 가용 자원의 정도(인력, 장비, 시간, 비용)
- 요구사항의 명확도
- 위험도

## 2. 소프트웨어 개발 모델

### (1) 폭포수 모델(Waterfall Model)

- 계획, 분석, 설계, 구현, 테스트, 운영 등 전 과정을 순차적으로 접근하는 개발모델
- 각 단계의 검증 후에 다음 단계를 진행한다.
- 각 단계가 순차적으로 진행되며, 병행되거나 거슬러 반복 진행되지 않는다.
- 가장 오래된 모형으로 적용 경험과 성공사례가 많다.
- 요구사항의 변경이 어렵다.
- 단계별 정의가 분명하고, 단계별 산출물이 명확하다.

### (2) 프로토타이핑 모델(Prototyping Model)

- 고객이 요구한 주요 기능을 프로토타입으로 구현하여 완성해가는 모델
- 개발자가 구축할 소프트웨어의 모델을 사전에 만들어 요구사항을 효과적으로 유도하고 수집한다.
- 프로토타이핑에 의해 만들어진 프로토타입은 폐기될 수 있고, 재사용될 수도 있다.
- 순서
  - 계획수립 → 프로토타입 개발 → 사용자 평가 → 구현 → 인수
- 장점
  - 사용자의 요구사항을 충실히 반영할 수 있다.
  - 비교적 빠른 기간 안에 사용자가 평가할 수 있는 결과물이 만들어진다.
  - 오류를 초기에 발견할 수 있다.
  - 변경이 용이하다.
- 단점
  - 최종적으로 시간과 비용이 훨씬 많이 들 수 있다.
  - 사용자가 실제 제품과 혼동할 수 있다.
  - 문서작성이 소홀해질 수 있다.
  - 프로토타입 폐기에 따른 비용이 든다.

### (3) 나선형 모델(Spiral Model)

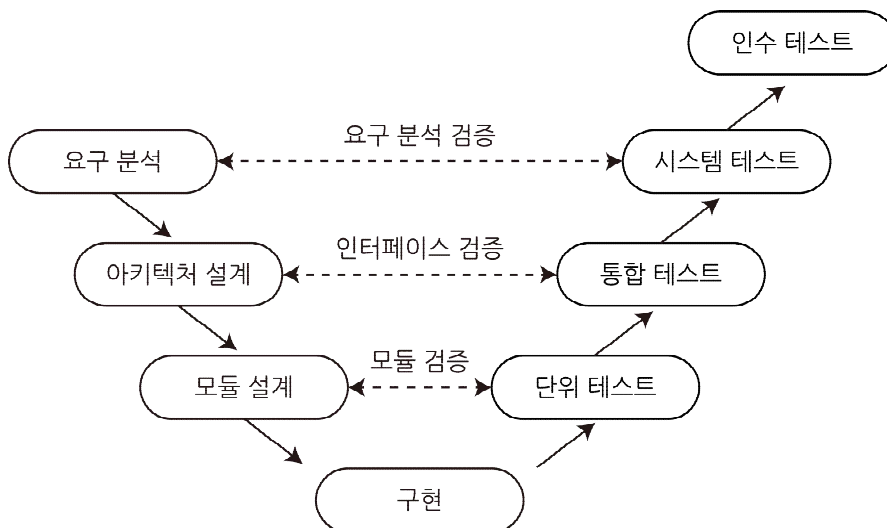
- 폭포수 모델과 프로토타이핑 모델의 장점을 수용하고, 위험 분석을 추가한 점증적 개발 모델
- 프로젝트 수행 시 발생하는 위험을 관리하고 최소화하려는 것이 목적
- 대규모 프로젝트 및 위험 부담이 큰 시스템 개발에 적합
- 순서
  - 계획수립(planning) → 위험분석(Risk Analysis) → 공학적 개발(Development) → 평가(Evaluation)
- 장점
  - 위험분석 과정으로 위험성이 큰 프로젝트를 수행할 수 있다.
  - 고객의 요구사항을 보다 더 상세히 적용할 수 있다.
- 단점
  - 시간과 비용이 많이 들 수 있다.
  - 반복 단계가 길어질수록 프로젝트 관리가 어렵다.

**(4) RAD(Rapid Application Development) 모델**

- 매우 짧은 개발 주기를 강조하는 점진적 소프트웨어 개발 방식
- 강력한 소프트웨어 개발 도구를 이용하여 매우 짧은 주기로 개발을 진행하는 순차적 소프트웨어 개발 프로세스
- CASE(Computer Aided Software Engineering) 도구를 이용해 시스템을 개발
- 개발 기간이 60일~90일 정도로 짧다.
- 기술적으로 위험이 적고 빠른 개발이 요구될 때 사용이 적합

**(5) V 모형**

- 폭포수 모델에 시스템 검증과 테스트 작업을 강조
- 높은 신뢰성이 요구되는 분야에 적합함

**(6) 4세대 기법(4th Generation Techniques)**

- CASE등의 자동화도구를 이용하여 요구사항 명세로부터 원시코드를 자동으로 생성

**3. 애자일(Agile) 방법론****(1) 애자일 방법론의 개념**

- 애자일 방법론은 소프트웨어 개발 방법에 있어서 아무런 계획이 없는 개발 방법과, 계획이 지나치게 많은 개발 방법들 사이에서 타협점을 찾고자 하는 방법론
- 애자일 개발 방법론이란 어느 특정 개발 방법론을 가리키는 말은 아니고 애자일 개발을 가능하게 해주는 다양한 방법론 전체를 일컫는 말
- 경량(Lightweight) 프로세스라고도 함
- 프로젝트를 시작한 후 끊임없이 개선 노력

**(2) 애자일 프로세스의 등장배경**

- 기존 소프트웨어 개발 방법론의 문제점을 개선하기 위해서 등장
- 기존 소프트웨어 개발 방법론의 주요 문제점
  - 계약과 계획준수를 중요시하는 갑과 을의 문화가 지배적
  - 문서를 중시함
  - 프로세스나 톨 적용을 중시함
  - 성과가 나쁠 때 계획 또는 통제의 실패로 인식함

**(3) 애자일 선언문**

- 공정과 도구보다 개인과 상호작용을
- 포괄적인 문서보다 작동하는 소프트웨어를
- 계약 협상보다 고객과의 협력을
- 계획을 따르기보다 변화에 대응하기를
- 우리는 왼쪽 항목의 가치를 인정하면서도 오른쪽 항목을 더 중요하게 여긴다.

**(4) 애자일 특징**

- 고객과 개발자의 지속적인 소통을 통하여 변화하는 요구사항을 신속하게 수용
- 개발자 개인의 가치보다는 팀의 목적을 우선시 하며 고객의 의견을 가장 우선
- 팀원들과의 주기적인 회의와 제품을 시연함으로써 소프트웨어를 점검
- 작업 계획을 짧게 세우고, 반복적으로 수행함으로 변화에 유연하게 대처

**(5) 애자일 방법론 종류****① XP(eXtream Programming)**

- 특징
  - 문서보다는 코드를 중시하고, 5가지 핵심가치와 12개 실천 항목이 존재
  - 개발을 세분화 하여 1~3주의 반복으로 개발을 진행
- XP 5가지 핵심가치
  - 용기 : 고객의 요구사항 변화에 능동적인 대처
  - 존중 : 개발자의 역량을 존중하고 충분한 권한과 권리를 부여
  - 의사소통 : 개발자, 관리자, 고객 간의 원활한 의사소통
  - 피드백 : 의사소통에 따른 즉각적인 피드백
  - 단순성 : 부가적 기능, 사용되지 않는 구조와 알고리즘 배제



- 12가지 실천사항

실천사항	설명
짝 프로그래밍 (Pair Programming)	하나의 작업을 2명의 프로그래머가 코딩·리뷰 공동 수행
계획 세우기 (Planning Game)	게임처럼 선수와 규칙, 목표를 두고 기획 수행
테스트 기반 개발 (Test Driven Development)	선 단위 테스트 후 실제 코드 작성
고객 상주 (Whole Team)	개발 효율을 위해 고객을 프로젝트 팀원으로 상주
지속적인 통합 (Continuous Integration)	상시 빌드 및 배포가 가능한 상태로 유지
코드 개선 (Design Improvement)	코드 개선 작업 수행(가시성, 성능 등), 불필요한 기능 제거 및 리팩토링
작은 릴리즈 (Small Releases)	짧고 잦은 릴리즈로 고객이 변경사항을 볼 수 있게 함 (작은 피드백)
코딩 표준 (Coding Standards)	표준화된 관례에 따라 코드 작성
공동 코드 소유 (Collective Code Ownership)	시스템에 있는 소스코드는 팀의 모든 프로그래머가 언제라도 수정 가능
간단한 디자인 (Simple Design)	가능한 가장 간결한 디자인 상태 유지
시스템 메타포어 (System Metaphor)	최종적으로 개발 되어야 할 시스템의 구조를 조망
작업시간 준수 (Sustainable Pace)	일주일에 40 시간 이상 작업 금지, 2주 연속 오버타임 금지

## ② 스크럼 (SCRUM)

- 소프트웨어에 포함될 기능·개선점에 대한 우선 순위를 부여
- 개발 주기는 30일 정도(스프린트)로 조절하고 개발 주기마다 실제 동작할 수 있는 결과를 제공
- 개발 주기마다 적용할 기능이나 개선에 대한 목록을 작성
- 날마다 15분 정도의 회의
- 항상 팀 단위로 생각한다.

## ③ 그 외 애자일 방법론

- 크리스탈 패밀리(Crystal)
  - 프로젝트의 규모와 영향의 크기에 따라서 여러 종류의 방법론을 제공
- Feature-Driven Development (FDD)
  - feature마다 2주 정도의 반복 개발을 실시
  - 기능 주도 개발
- Adaptive Software Development, ASD
  - 합동 애플리케이션 개발을 사용하는 방법론

## 02 프로젝트 계획 및 분석

### Section 1. 프로젝트 계획

#### 1. 프로젝트 관리

##### (1) 프로젝트 관리의 개념

- 특정한 목적을 가진 프로젝트를 한정된 기간, 예산, 자원 내에서 사용자가 만족할 만한 제품을 개발하도록 행하는 기술적, 관리적 활동

##### (2) 프로젝트 관리의 목적

- 납기준수, 예산준수, 품질 준수를 통한 고객 만족 달성
- 고품질의 제품 개발 및 개발 절차 준수

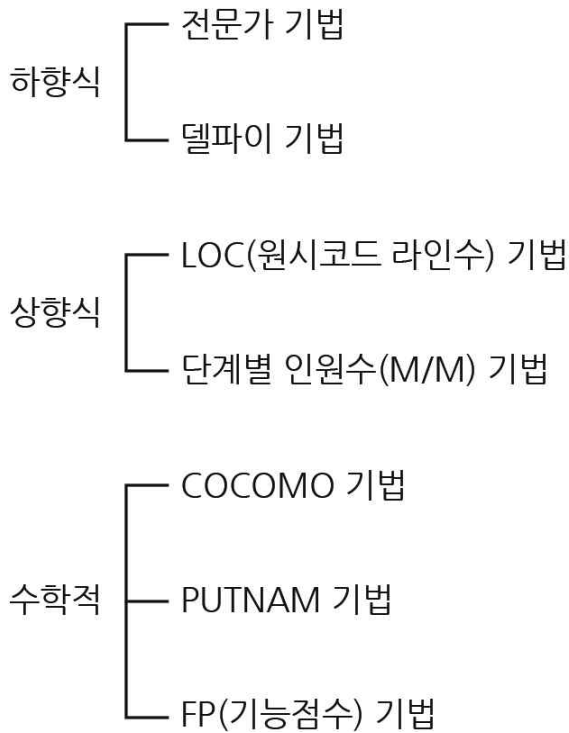
##### (3) 프로젝트 핵심 관리대상(3P)

- 사람(People)
  - 프로젝트 관리에서 가장 기본이 되는 요소
- 문제(Problem)
  - 처리해야 할 내용을 분석하고 설계한다.
- 프로세스(Process)
  - 소프트웨어 개발에 필요한 골격을 제공한다.

##### (4) PMBOK(Project Management Body of Knowledge)

- PMI(Project Management Institute)에서 제작한 프로젝트 관리 프로세스 및 지식 체계
- PMBOK 5단계 프로세스 그룹
  - ㉠ 1단계 : 프로젝트 착수
    - 광범위한 프로젝트의 범위를 정하는 단계
  - ㉡ 2단계 : 프로젝트 계획
    - 프로젝트의 세부 범위를 정의하고, 프로젝트 관리 계획을 만드는 단계
    - 비용, 품질, 기간, 사용 가능한 자원이 포함됨
  - ㉢ 3단계 : 프로젝트 실행
    - 프로젝트가 개발되고 완료되는 단계
  - ㉣ 4단계 : 프로젝트 통제
    - 계획 대비 목표의 진척상황을 모니터링하고 성과를 측정하는 단계
  - ㉤ 5단계 : 프로젝트 종료
    - 프로젝트가 요구사항을 만족하는지 검증하고, 고객으로부터 확인 받는 단계

## 2. 개발 비용 산정



### (1) 하향식 산정 기법(Top-Down)

#### ① 전문가 기법

- 조직 내 경험이 있는 전문가에게 비용 산정을 의뢰하여 산정하는 기법

#### ② 델파이 기법

- 여러 전문가의 의견을 종합하여 판단하는 기법
- 특정 전문가의 주관적인 편견을 보완하기 위해 여러 명의 전문가로 구성된다.

### (2) 상향식 산정 기법(Bottom-Up)

#### ① LOC(원시코드 라인수) 기법

- 각 기능의 원시 코드 라인 수의 비관치(가장 많은 라인 수), 낙관치(가장 적은 라인 수), 중간치(기대치, 평균 라인 수)를 측정 후 예측치를 구하고, 이를 이용해 비용을 산정하는 기법
- 추정 LOC
  - $(\text{낙관치} + (4 * \text{중간치}) + \text{비관치}) / 6$

#### ② 단계별 인원수(M/M) 기법

- 소프트웨어 개발 생명주기 각 단계별로 적용시켜 모든 단계의 비용을 산정하는 기법
- LOC 보다 정확성을 기하기 위한 기법

### (3) 수학적 산정 기법

#### ① COCOMO 기법

- 개발할 S/W의 규모를 예측한 후 S/W 종류에 따라 각 비용 산정 공식에 대입하여 비용을 산정하는 기법
- LOC 기법을 개발유형에 따라 다르게 적용한 기법
- 개발 유형

개발유형	설명
조직형 (Organic Mode)	- 5만 라인 이하의 프로젝트 - 일반 업무용 소프트웨어
반분리형 (Semidetached Mode)	- 30만 라인 이하의 프로젝트 - 운영체제, DBMS 등
내장형 (Embedded Mode)	- 30만 라인 이상의 프로젝트 - 미사일 유도 시스템, 신호기 제어 시스템 등

#### ② Putnam 기법

- Putnam이 제안한 생명 주기 예측 모형
- 소프트웨어 생명 주기의 전 과정 동안의 노력의 분포를 가정해주는 모형
- 시간에 따른 함수로 표현되는 Rayleigh-Norden 곡선의 노력 분포도를 기초로 한다.
- 대형 프로젝트에서 이용되는 기법이다.
- 개발 기간이 늘어날수록 프로젝트 적용 인원의 노력이 감소한다.
- SLIM : Rayleigh-Norden 곡선과 Putnam 예측 모형을 기초로 개발한 자동화 추정 도구

#### ③ 기능 점수 기법(FP, Function Point)

- 개요
  - 소프트웨어가 가지는 기능의 개수를 기준으로, 소프트웨어의 규모를 측정하는 기법
  - 1979년 IBM사의 A.J.Albrecht 가 고안한 방식이다.
  - 객관적이고 정량적인 소프트웨어의 규모를 산출 할 수 있게 되었다.
  - ESTIMACS : FP모형을 기초로 개발된 자동화 추정 도구
- 비용산정에 이용되는 요소
  - 자료 입력(입력 양식)
  - 정보 출력(출력 보고서)
  - 명령어(사용자 질의수)
  - 데이터 파일
  - 필요한 외부 루틴과의 인터페이스

### 3. 개발 일정 산정

#### (1) 소프트웨어 개발 일정 계획

- 소프트웨어를 개발하기 위해 어떤 작업이 필요한지 정의하고, 작업들의 우선순위를 정하여, 프로젝트 일정에 대한 계획을 세우는 것
- 작업 순서
  - 작업분해(Work Breakdown Structure)
  - CPM 네트워크 작성
  - 최소 소요 기간을 구함
  - 소요 M/M, 기간 산정하여 CPM 수정
  - 간트 차트로 표현

#### (2) WBS(Work Breakdown Structure)

- 프로젝트 목표를 달성하기 위해 필요한 활동과 업무를 세분화 하는 작업
- WBS 작성방법
  - 전체를 큰 단위로 분할
  - 각각의 부분에 대해 좀 더 작은 단위로 분해하여 계층적으로 표현
  - 담당인원을 배치해 구성도 작성
- WBS 역할
  - 프로젝트에서 수행할 업무를 식별
  - 일정계획과 산정
  - 전체일정 진행상황 파악
  - 이해당사자들 간의 의사소통

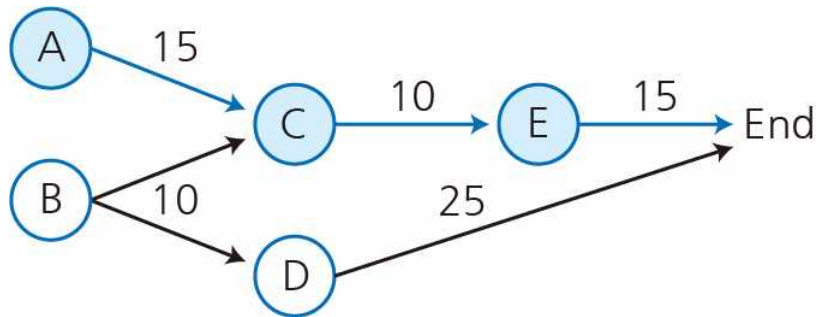
#### (3) Network Chart(PERT/CPM)

- PERT
  - 미 해군의 Polaris 미사일 개발 프로젝트의 일정계획 및 진행과정을 효율적으로 관리하기 위해 개발됨
  - 전체 프로젝트의 시간단축을 목표로 함
  - 개발기간을 낙관치, 기대치, 비관치로 나누어 예측치를 구한다.
  - $\text{예측치} = (\text{낙관치} + (4 * \text{기대치}) + \text{비관치}) / 6$
- CPM
  - 미국의 듀폰(Du pont)사와 레밍톤(Remington)사의 화학공장 유지 및 관리를 위해 개발됨
  - 최소의 비용 추가 투입을 고려하여 전체 프로젝트의 시간단축을 목표로 함
- PERT/CPM
  - 작업의 선/후행 관계를 고려하여 전체작업의 완료시간을 결정하고(PERT), 추가비용 투입을 고려하여 전체작업 완료시간을 단축하는(CPM) 네트워크 분석 기법
  - 복잡한 대형 프로젝트를 효율적으로 계획 및 통제하기 위해 개발된 기법
  - 임계경로(Critical Path) : 프로젝트를 끝내기 위해 필요한 최소 소요기간

• CPM 소작업 리스트

작업	선행작업	소요기간(일)
A	-	15
B	-	10
C	A, B	10
D	B	25
E	C	15

• CPM 네트워크 작성



• 소요기간 산정

- 임계경로(Critical Path) : 40일
- D의 가장 빠른 착수일 : 10일
- D의 가장 늦은 착수일 : 15일
- D의 여유기간 : 5일

(4) 간트 차트(Gantt chart)

- 일정 계획의 최종 산출물
- 프로젝트 일정관리를 위한 바(bar)형태의 도구
- 각 업무별로 일정의 시작과 끝을 그래픽으로 표시하여 전체 일정을 한눈에 볼 수 있다.
- 각 업무(activities) 사이의 관계를 보여준다.

Work Break-down Structure						9월																		
Procedures	Steps	Tasks	담당자	schedule		산출물/비고	1주		2주					3주										
				시작일	종료일		금	토	일	월	화	수	목	금	토	일	월	화	수	목	금	토	일	월
							1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1.0.0.	착수 및 프로젝트 관리																							
	1.1.0.	프로젝트 관리																						
		1.1.1. 착수보고		9월 14일	9월 20일	착수문서 일제																		
		1.1.2. 착수 OT(범위, 일정, 요구사항 등)		9월 14일	9월 20일	착수문서 일제																		
		1.1.3. 계약		9월 14일	9월 14일	계약서																		
	1.2.0.	프로젝트 관리 실무																						
		1.2.1. 주간회의		매주 목	매주 목	주간 보고서																		
		1.2.2. 완료보고		2월 9일	2월 9일	완료보고																		
2.0.0.	분석/설계																							
	2.1.0.	분석_조사																						
		2.1.1. 조사양식 제작		9월 7일	9월 7일	조사양식, 리스트																		
		2.1.2. 조사양식 컨펌		9월 11일	9월 11일	리스트변경																		
3.0.0.	디자인																							
	3.1.0.	메인 디자인																						
		3.1.1. 메인디자인 시안 컨펌		9월 13일	9월 13일	메인 시안																		
		3.1.2. 메인디자인 시안 수정		9월 14일	9월 29일																			
4.0.0.	프론트 개발																							
	4.1.0.	관리자 페이지 프론트 개발_일반관리																						
		4.1.1. 로그인 페이지																						
		4.1.2. 관리자 신청 페이지																						

## Section 2. 요구사항 분석

### 1. 현행 시스템 분석

#### (1) 현행 시스템 파악

##### ① 현행 시스템 파악의 정의

- 현행 시스템이 어떤 하위 시스템으로 구성되어 있는지
- 제공하는 기능이 무엇인지
- 다른 시스템들과 어떤 정보를 주고받는지
- 어떤 기술요소를 사용하고 있는지
- 사용하고 있는 소프트웨어 및 하드웨어는 무엇인지
- 네트워크는 어떻게 구성되어 있는지

##### ② 현행 시스템 파악의 목적

- 향후 개발하고자 하는 시스템의 개발범위 및 이행 방향성 설정에 도움을 주는 것이 목적

##### ③ 현행 시스템 파악 절차

- 1단계 - 현행 시스템의 구성, 기능, 인터페이스 현황을 파악하는 단계
- 2단계 - 현행 시스템의 아키텍처 및 소프트웨어 구성 현황을 파악하는 단계
- 3단계 - 현행 시스템의 하드웨어 및 네트워크 구성 현황을 파악하는 단계

#### (2) 플랫폼 기능 분석

##### ① 플랫폼 정의

- 어플리케이션을 구동시키는데 필요한 하드웨어와 소프트웨어의 결합
- 공급자와 수요자 등 복수 그룹이 참여하여 각 그룹이 얻고자 하는 가치를 공정한 거래를 통해 교환할 수 있도록 구축된 환경

##### ② 플랫폼 기능

- 소프트웨어 개발과 운영비용이 감소하고 생산성이 향상
- 플랫폼 내의 커뮤니티를 형성하고 네트워크 효과를 유발

##### ③ 플랫폼의 유형

- 싱글 사이드 플랫폼(single-side platform)
  - 제휴 관계를 통해 소비자와 공급자를 연결하는 형태
- 투 사이드 플랫폼(two-side platform)
  - 두 그룹을 중개하고 모두에게 개방하는 형태
- 멀티 사이드 플랫폼(multi-side platform)
  - 다양한 이해관계 그룹을 연결하여 중개하는 형태

### (3) 운영체제 분석

#### ① 운영체제 개념

- 컴퓨터 시스템 자원을 효율적으로 관리하여 사용자가 컴퓨터를 편리하게 사용할 수 있도록 환경을 제공하는 시스템 소프트웨어
- 컴퓨터 시스템이 제공하는 모든 하드웨어, 소프트웨어를 사용할 수 있도록 해줌
- 사용자와 하드웨어간의 인터페이스를 담당

#### ② 운영체제 종류

- 유닉스(UNIX)
- 리눅스(Linux)
- 윈도우즈(Windows)
- 맥 OS(Mac OS)
- iOS
- Android
- 윈도우폰 OS

### (4) 네트워크 분석

#### ① 네트워크 개념

- 노드(컴퓨터)들이 자원을 공유할 수 있게 하는 디지털 전기 통신망
- 노드 간 연결을 통해 서로에게 데이터를 교환

#### ② 프로토콜

- 데이터를 교환하기 위해 사용하는 통신 규칙
- 프로토콜의 3요소
  - 구문(Syntax) : 데이터의 형식이나 부호화 및 신호 레벨을 규정
  - 의미(Semantic) : 전송의 조작이나 오류 제어를 위한 제어 정보에 대한 규정
  - 타이밍(Timing) : 접속되어 있는 개체 간의 통신 속도의 조정이나 메시지의 순서 제어 규정

### (5) DBMS 분석

#### ① DBMS(Database Management System) 개념

- 사용자, 어플리케이션등의 상호 작용을 위해 데이터를 저장하고 분석하는 소프트웨어
- 데이터베이스 생성, 조회, 변경 등의 관리

#### ② 현행 시스템 데이터베이스 분석

- DBMS의 종류, 버전, 구성방식, 스토리지 크기, 백업 주기 분석
- 테이블 수량, 데이터 증가 추이, 백업 방식 등을 분석



## (6) 미들웨어(Middleware) 분석

### ① 미들웨어 개념

- 양 쪽을 연결하여 데이터를 주고 받을 수 있도록 중간에서 매개 역할을 하는 소프트웨어

### ② 미들웨어 종류

- RPC(Remote Procedure Call)
  - 클라이언트가 원격에서 동작하는 프로시저를 호출하는 시스템
- MOM(Message Oriented Middleware)
  - 응용 소프트웨어 간의 데이터 통신을 위한 소프트웨어
  - 메시지 기반의 비동기형 메시지를 전달하는 방식의 미들웨어
- ORB (Object Request Broker)
  - 객체지향 시스템에서 객체 및 서비스를 요청하고 전송할 수 있도록 지원하는 미들웨어
- DB 접속 미들웨어
  - 애플리케이션과 데이터베이스 서버를 연결해주는 미들웨어
- TP 모니터
  - 온라인 트랜잭션을 처리 및 감시하는 미들웨어
- WAS(Web Application Server)
  - 동적인 콘텐츠를 처리하기 위한 미들웨어
- ESB(Enterprise Service Bus)
  - 메시지 기반으로 느슨한 결합형태의 표준 인터페이스 통신을 지원하는 미들웨어
  - 기업 안팎에 있는 모든 시스템 환경을 연동하는 미들웨어

## 2. 요구 공학

### (1) 요구공학 개념

- 고객 요구를 체계적으로 수집, 분석, 명세화, 검증하고 추적, 변경되는 요구사항을 도출하고 관리하는 기법

### (2) 요구공학의 필요성

- 분석의 어려움
  - 이해부족, 의사소통, 잦은 요구사항의 변경
- 요구사항 변화
  - 요구사항은 개발초기에 불완전하고, 개발 동안 지속적으로 변화
- 관점별 차이 발생
  - 묵시적 요구사항, 변경과 추적에 대한 문제, 해당 업무에 대한 지식

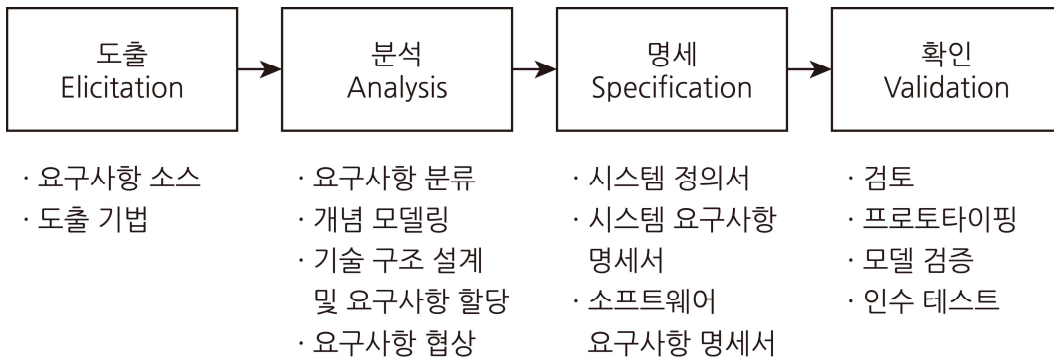
### (3) 요구사항의 분류

#### ① 참여자 관점

- 사용자 요구사항
  - 사용자의 관점에서 소프트웨어에 대해 원하는 사항들
- 시스템 요구사항
  - 설계자의 관점에서 하드웨어와 소프트웨어가 갖춰야 하는 것들
- 소프트웨어 요구 사항
  - 개발자의 관점에서 소프트웨어가 갖춰야 하는 사항들

#### ② 요구사항 내용의 종류

- 기능적 요구사항
  - 소프트웨어를 구성하는 기능들이 무엇인지를 정의한 것
- 비기능적 요구사항
  - 소프트웨어의 기능들에 대한 조건과 제약 사항들이 무엇인지 정의한 것
  - 보안, 성능, 품질, 안정성 등

**(4) 요구사항 개발 프로세스****① 도출**

- 소프트웨어가 해결해야 할 문제를 이해하고 요구사항이 어디에 있고, 어떻게 수집할 것인가를 확인
- 요구사항 도출 기법
  - 이해 관계자 분석
  - 브레인 스토밍
  - 인터뷰
  - 문서 분석 및 검토
  - 포커스 그룹
  - 인터페이스 분석
  - 관찰
  - 프로토타이핑
  - 요구사항 워크숍
  - 설문 조사

**② 분석**

- 요구사항들 간에 상충 되는 것을 해결
- 소프트웨어의 범위를 파악
- 업무환경과의 상호작용 파악(도메인 분석)
- 구조적 분석 도구
  - DFD(Data Flow Diagram) : 자료 흐름도
  - Data Dictionary : 자료 사전
  - Mini-Spec : 소단위 명세서
  - ERD(Entity Relationship Diagram) : 개체 관계도
  - STD(State Transition Diagram) : 상태 전이도
- 객체지향 분석 도구
  - UML(Unified Modeling Language)
  - 모델링

## ③ 명세

- 체계적으로 검토, 평가, 승인될 수 있는 문서를 작성
- 시스템정의, 시스템 요구사항, 소프트웨어 요구사항을 작성
- 요구사항 명세 기법

구분	정형 명세 기법	비정형 명세 기법
기반	- 수학, 논리학	- 자연어, 그림 중심
작성기법	- 수학적 기호, 정형화된 표기법	- 일반 명사, 동사 등의 자연어를 기반으로 서술하거나 다이어그램으로 작성
장점	- 명세 오류 및 모호성 쉽게 파악	- 사용자/개발자간 의사전달 용이
단점	- 작성이 어렵고, 시간이 많이 소모됨	- 내용이 모호하고, 완전한 검증이 곤란
언어 종류	- VDM, Z, Petri-net, CSP	- FSM, Decision Table, ER 모델링, State Chart(SADT) 등

- 산출물
  - 시스템 정의서
  - 시스템 요구사항 명세서
  - 소프트웨어 요구사항 명세서

## ④ 확인

- 분석가가 요구사항을 이해했는지 확인(Validation)
- 요구사항 문서가 일관성 있고 완전한지 검증(Verification)
- 이해 관계자들이 문서를 검토하고, 형상관리를 수행

## (5) 요구사항 분석 도구

## ① 요구사항 분석 CASE(Computer Aided Software Engineering) 도구

- 요구사항을 자동으로 분석하고, 요구사항 분석 명세서를 기술하는 도구
- 소프트웨어 개발 전반에 걸쳐 적용된다.
- CASE 도구의 분류

분류	설명
상위 CASE	- 생명주기 전반부에 사용되며, 소프트웨어의 계획과 요구분석, 설계 단계를 지원한다. - 모순검사, 오류검사, 자료흐름도 작성 등의 기능을 수행한다.
하위 CASE	- 생명 주기 후반부에 사용되며, 코드의 작성과 테스트, 문서화 하는 과정을 지원한다. - 구문 편집기, 코드 생성기 등의 기능을 수행한다.
통합 CASE	- 소프트웨어 생명주기에 포함되는 전체 과정을 지원한다.

- 종류

종류	설명
SADT	<ul style="list-style-type: none"> <li>- Structured Analysis and Design Technique</li> <li>- SoftTech 사에서 개발</li> <li>- 시스템 정의, 요구사항 분석, 시스템/소프트웨어 설계를 이용되는 구조적 분석 및 설계도구</li> </ul>
SREM	<ul style="list-style-type: none"> <li>- Software Requirements Engineering Methodology</li> <li>- 실시간 처리 소프트웨어 시스템에서 요구사항을 명확히 기술하도록 할 목적으로 개발</li> </ul>
PSL/PSA	<ul style="list-style-type: none"> <li>- 미시간 대학에서 개발한 것으로 PSL과 PSA를 사용하는 자동화 도구</li> <li>- PSL(Problem Statement Language) : 문제 기술언어</li> <li>- PSA(Problem Statement Analyzer) : PSL로 기술한 요구사항을 자동으로 분석하여 다양한 보고서를 출력하는 문제 분석기</li> </ul>
TAGS	<ul style="list-style-type: none"> <li>- Technology for Automated Generation of Systems</li> <li>- 시스템 공학 방법 응용에 대한 자동 접근 방법</li> <li>- 개발 주기의 전 과정에 이용할 수 있는 통합 자동화 도구</li> </ul>

## ② HIPO(Hierarchy Input Process Output)

- HIPO 의 개념
  - 하향식 소프트웨어 개발을 위한 문서화 도구
  - 시스템의 기능을 여러 개의 고유 모듈들로 분할하여 이들 간의 계층구조를 표현한 도표
- HIPO 의 기능
  - 분석 및 설계 도구로 사용된다.
  - 하향식 개발에 적합하다.
  - 체계적인 문서관리에 효율적이다.
  - 기능과 자료의 의존관계를 명시할 수 있다.
- HIPO Chart 종류

종류	설명
가시적 도표 (Visual Table of Content)	<ul style="list-style-type: none"> <li>- 시스템의 전체 기능과 흐름을 보여주는 Tree(계층) 구조</li> <li>- 가시적 도표에는 입력, 처리, 출력 없음</li> </ul>
총체적 도표 (Overview Diagram)	<ul style="list-style-type: none"> <li>- 프로그램을 구성하는 기능을 기술한 것</li> <li>- 입력, 처리, 출력에 대한 전반적인 정보 제공</li> </ul>
세부적 도표 (Detail Diagram)	<ul style="list-style-type: none"> <li>- 총체적 도표에 표시된 기능을 구성하는 기본 요소들을 상세히 기술하는 도표</li> <li>- 총체적 도표와 같은 모양이지만 내용만 좀 더 복잡하게 들어간 형태</li> </ul>

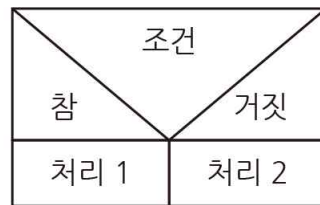
## ③ NS chart(Nassi-Shneiderman / 나시 슈나이더만 차트)

- NS chart 의 개념
  - 논리의 기술에 중점을 두고 도형을 이용한 표현 방법이다
  - 프로그램의 처리 흐름을 상자 형태의 그림으로 나타낸 그림
  - 연속, 선택, 반복 등의 제어 논리 구조를 표현한다

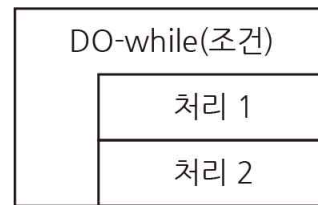
- 표현방법



순차처리 구조



선택 구조



반복 구조

- 특징

- 논리의 기술에 중점을 둔 도형을 이용한 표현 방법이다
- 그리기가 어렵다.(전문성이 있어야 한다)
- 순차, 선택, 반복으로 표현한다
- 임의의 제어 이동이 어렵다(GOTO 구조 불가)
- 이해하기 쉽고 코드 변환이 용이하다

### 3. 요구사항 분석 모델링

#### (1) 모델링의 개념

- 복잡한 시스템을 쉽게 이해하기 위해 불필요한 부분을 생략하고 추상화하여 간단한 모델로 표현하는 것을 의미
- 소프트웨어를 구성하는 모듈들을 식별하고, 이것들의 연결을 그림으로 표현
- 요구분석 과정에 의해 수집된 개념과 정보를 분석하여 UML과 같은 방법을 이용하여 모델로 비주얼화

#### (2) 모델링이 주는 도움

- 소프트웨어를 이해하는 데 도움
- 이해관계자들 사이에서 문제를 해결 할 수 있도록 해준다.
- 파악한 개념을 사용자와 고객에게 전달할 때 도움을 준다.
- 설계, 구현, 테스트, 유지보수에 개념적인 기준을 제공한다.

#### (3) 모델링 구분

- 기능적 모델링
- 정적 모델링
- 동적 모델링

#### (4) 분석 모델의 종류

- 구조적 분석 모델
- 객체 지향 분석 모델
- 정보공학 분석 모델
- 정형화 분석 모델

#### (5) 구조적 분석 모델

##### ① 구조적 분석 방법론

- 도형화된 도구를 이용해 정형화된 분석 절차에 따라 사용자 요구사항을 파악하고 문서화하는 분석 기법
- 하향식 기능 분해 기법 등을 사용하는 특성

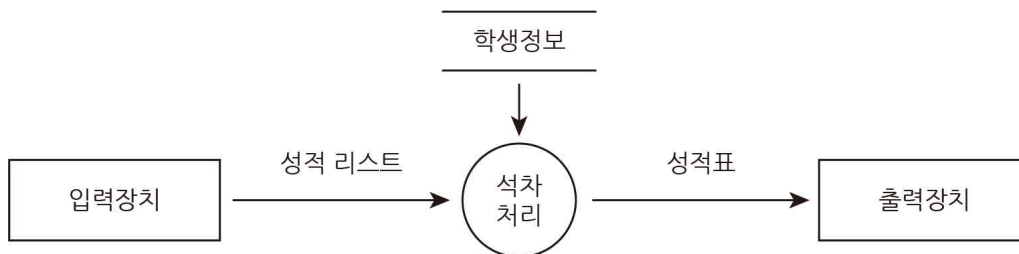
## ② 구조적 분석 도구

### ㉠ 자료 흐름도(DFD, Data Flow Diagram)

- 가장 보편적으로 사용되는 시스템 모델링 도구로서, 기능 중심의 시스템을 모델링 하는데 적합
- 자료의 흐름과 처리 과정을 도형 중심으로 기술
- 자료 흐름 그래프 또는 버블 차트라고도 함
- 자료 흐름도 구성요소

구성요소	설명	기호
처리 과정(Process)	자료를 변환시키는 처리 과정을 나타낸다.	○
자료 흐름(Data Flow)	자료의 이동을 나타낸다.	→
자료 저장소(Data Store)	파일, 데이터베이스 등 자료가 저장되는 곳을 나타낸다.	▬ ▬ ▬
단말(Terminator)	데이터의 입출력 주체(사용자)를 나타낸다.	□

- 자료 흐름도 사례



### ㉡ 자료사전(Data Dictionary, DD)

- 자료흐름도에 기술된 모든 자료들에 대한 사항을 자세히 정의
- 자료사전 사용 기호

기호	의미	설명
=	자료의 정의	~로 구성되어 있다
+	자료의 연결	그리고, 순차(and)
( )	자료의 생략	생략 가능한 자료
[   ]	자료의 선택	여러 대안 중 하나 선택
{ }	자료의 반복	자료의 반복
**	자료의 설명	주석

- 자료사전 예시



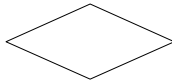
자료 흐름	쇼핑몰 회원정보는 회원번호, 회원성명, 전화번호, 휴대폰번호로 구성되어 있고, 전화번호와 휴대폰번호는 둘 중 하나만 선택이 가능하다.
표기형식	회원정보 = 회원번호 + 회원성명 + [전화번호   휴대폰번호]

## ㉔ 소단위 명세서(Mini-Specification)

- 자료 흐름도에서 어떤 일이 수행되는지를 정의하기 위해 각 처리들이 수행하는 업무를 상세하게 작성
- 프로세스 명세서라고도 한다.
- 소단위 명세서는 구조적 언어이고, 선후 조건문, 의사결정표 등이 사용

## ㉕ 개체 관계도(Entity Relationship Diagram, ERD)

- 시스템에서 처리되는 개체(자료)와 개체의 구성과 속성, 개체 간의 관계를 표현하여 자료를 모델화 하는데 사용
- 개체 관계도 구성

속성	설명	ERD 기호
개체(Entity)	업무의 중심이 되는 실체	
속성(Attribute)	업무에 속하는 구체적인 항목	
관계(Relationship)	업무와 업무의 연관관계	

## ㉖ 상태 전이도(State Transition Diagram, STD)

- 시스템에 어떤 일이 발생할 경우 시스템의 상태와 상태 간의 전이를 모델화한 것으로, 상태 전이도를 통해 개발자는 시스템의 행위를 정의

## (6) 객체 지향 분석 모델

## ① 객체 지향 분석

- 사용자의 요구사항을 분석하여 요구된 문제와 관련된 모든 클래스, 이와 연관된 속성과 연산, 그들 간의 관계 등을 정의하여 모델링하는 작업

## ② 객체지향 분석 방법론

## ㉑ Rumbaugh(럼바우) 방법

- 가장 일반적으로 사용되는 방법, 분석 활동을 객체 모델, 동적 모델, 기능 모델로 나누어 수행
- 분석 절차

종류	설명
객체 모델링 (Object Modeling)	<ul style="list-style-type: none"> <li>- 시스템에서 요구되는 객체를 찾아내어 속성과 연산 식별 및 객체들 간의 관계를 규정해 객체 다이어그램으로 표현</li> <li>- 세 가지 모델 중 가장 선행되어야 함</li> </ul>
동적 모델링 (Dynamic Modeling)	<ul style="list-style-type: none"> <li>- 상태 다이어그램을 이용하여 시간의 흐름에 따라 제어 흐름, 동작 순서 등 동적인 행위 표현</li> <li>- 객체나 클래스의 상태, 사건을 중심으로 표현</li> </ul>
기능 모델링 (Functional Modeling)	<ul style="list-style-type: none"> <li>- 자료 흐름도(DFD)를 이용해 다수의 프로세스들 간의 자료 흐름을 중심으로 처리 과정 표현</li> <li>- 어떤 데이터를 입력하여 어떤 결과를 구할 것인지를 표현</li> </ul>



- ㉠ Booch(부치) 방법
  - 미시적 개발 프로세스와 거시적 개발 프로세스를 모두 사용하는 분석 방법
- ㉡ Jacobson 방법
  - Use case를 강조하여 사용하는 분석 방법
- ㉢ Coad와 Yourdon 방법
  - E-R 다이어그램을 사용하여 객체의 행위를 모델링, 객체 식별, 구조 식별, 주제 정의 등의 과정으로 구성하는 기법
- ㉣ Wirfs-Brock 방법
  - 분석과 설계간 구분 없음
  - 고객 명세서를 평가해서 설계 작업까지 연속적으로 수행

## 03 소프트웨어 설계

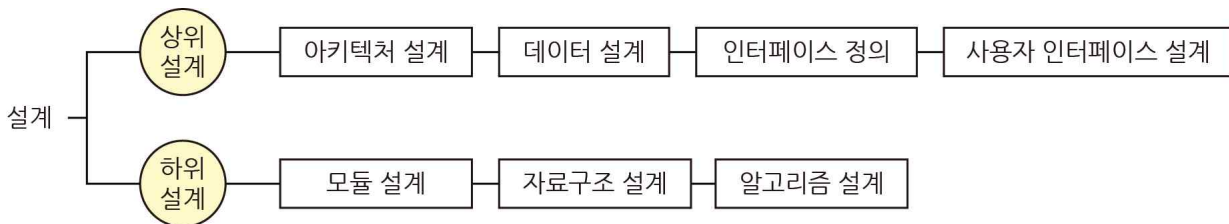
### Section 1. 소프트웨어 설계의 기본 원칙

#### 1. 소프트웨어 설계

##### (1) 소프트웨어 설계의 개념

- 요구사항 명세서를 참조하여 소프트웨어의 구체적인 설계서를 작성하는 단계
- 물리적으로 구현이 가능하도록 시스템을 구체적으로 정의하는 단계

##### (2) 소프트웨어 설계의 종류



##### ① 상위설계

- 아키텍처 설계
  - 시스템의 전체적인 구조 설계
- 데이터 설계
  - 시스템에 필요한 정보를 설계
  - 데이터 베이스 설계
- 인터페이스 정의
  - 시스템의 구조와 서브 시스템들 사이의 인터페이스를 명확히 정의
- 사용자 인터페이스 설계
  - 사용자가 익숙하고 편리하게 사용하도록 인터페이스 설계

##### ② 하위설계

- 모듈 설계
  - 각 모듈의 실제적인 내부를 알고리즘 형태로 표현
- 자료구조 설계
  - 자료구조, 변수 등에 대한 상세한 정보를 설계
- 알고리즘 설계
  - 업무의 처리 절차 등을 설계

### (3) 소프트웨어 설계의 원리

- 분할과 정복(Divide & Conquer)
  - 규모가 큰 소프트웨어를 여러 개의 작은 서브 시스템으로 나누어 하나씩 완성 시킨다.
- 추상화(Abstraction)
  - 특정한 목적과 관련된 필수 정보만 추출하여 강조하고, 관련이 없는 세부 사항을 생략함으로써, 본질적인 문제에 집중할 수 있도록 한다.
  - 자세한 구현 전에, 상위 레벨에서 제품의 구현을 먼저 생각해보는 것
  - 추상화 기법

추상화 기법	설명
과정 추상화	자세한 단계를 고려하지 않고 상위 수준에서 수행 흐름만 먼저 설계
데이터 추상화	데이터 구조를 대표할 수 있는 표현으로 대체하는 것이다.
제어 추상화	여러 명령들을 간단한 표현으로 대체하는 것이다.

- 단계적 분해(Gradual Decomposition)
  - 기능을 점점 작은 단위로 나누어 점차적으로 구체화 하는 방법
- 모듈화(Modulization)
  - 실제로 개발할 수 있는 작은 단위로 나눈다.
- 정보은닉(Information Hiding)
  - 다른 객체에게 자신의 정보를 숨기고, 자신의 연산만을 통해 접근이 가능하도록 한다.
  - 클래스 외부에서 특정 정보에 대한 접근을 막는다는 의미
  - 캡슐화와 밀접한 관계가 있다.

## 2. 설계 모델링

### (1) 설계 모델링 개념

- 소프트웨어를 구성하는 모듈들을 식별하고, 이것들의 연결을 그림으로 표현한 것
- 소프트웨어를 만들기 위한 계획 또는 만들어야 할 물건을 의미있게 표현한 것
- 소프트웨어에 대하여 여러 엔지니어들이 공통된 개념을 공유하는데 도움을 준다.

### (2) 설계 모델링 원칙

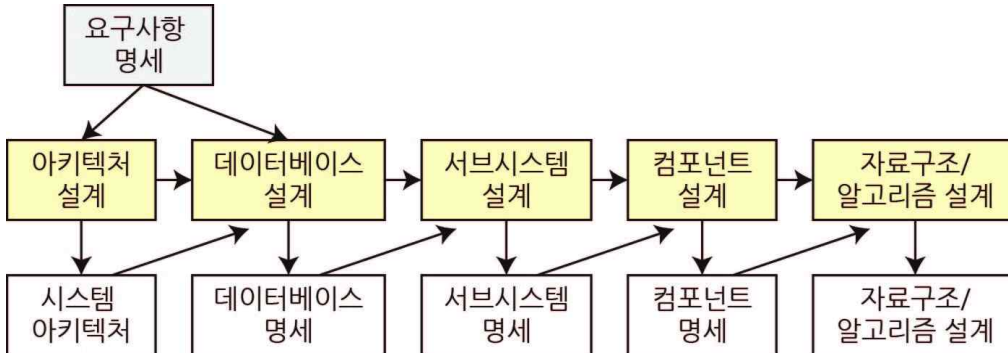
- 소프트웨어 설계는 변경이 용이하도록 구조화되어야 한다.
- 한 함수 안에 특정 기능을 수행하는 데 필요한 자료만을 사용하도록 한다.
- 요구사항 분석에서 얻은 정보를 이용하여 반복적 방법을 통해 이루어져야 한다.
- 독립적이고 기능적인 특성들을 지닌 모듈 단위로 설계되어야 한다.

### (3) 설계 모델링 유형

- 구조 모델링
  - 소프트웨어를 구성하는 컴포넌트들의 유형, 인터페이스, 내부 설계 구조 및 이들의 연결 구조를 모델링
  - 시스템의 구성 요소들과 이들 사이의 구조적인 관계와 특성들의 모델링
  - UML 정적 다이어그램

- 행위 모델링
  - 소프트웨어의 구성요소들의 기능들이 언제, 어떠한 순서로 기능을 수행해야 작용하는지를 모델링
  - 시스템의 구성 요소들이 언제 어떠한 순서로 수행되는가와 같은 동적 특성들의 모델링
  - UML 동적 다이어그램

#### (4) 소프트웨어 설계 절차 및 유형



유형	설명
아키텍처 설계	시스템을 구성하는 서브시스템들과 그들 간의 관계를 파악하고 명세한다.
데이터베이스 설계	시스템 구현에 사용되는 데이터의 구조를 자세하게 설계하고 명세한다.
서브시스템 설계	각 서브시스템이 담당하는 서비스와 제약사항들에 대해 명세한다.
컴포넌트 설계	서브시스템이 수행하는 기능을 여러 컴포넌트에 할당하고, 컴포넌트들의 인터페이스를 설계하고 명세한다.
자료구조와 알고리즘 설계	컴퓨터에 자료를 효율적으로 저장하는 방식과, 자료구조 내에서 기본적인 연산방법을 설계하고 명세
협약에 의한 설계	클래스에 대한 여러 가정을 공유하도록 명세 - 선행 조건 : 컴포넌트 오퍼레이션 사용 전에 참이 되어야할 조건 - 결과 조건 : 사용 후 만족되어야 할 결과조건 - 불변 조건 : 오퍼레이션이 실행되는 동안 항상 만족되어야할 조건

## Section 2. 소프트웨어 아키텍처

### 1. 소프트웨어 아키텍처

#### (1) 소프트웨어 아키텍처(SoftWare Architecture) 개념

- 소프트웨어의 골격이 되는 기본구조
- 시스템의 컴포넌트 사이의 관계를 정의한 구조

#### (2) 소프트웨어 아키텍처의 특징

특징	설명
간략성	이해하고 추론할 수 있을 정도의 간결성 유지
추상화	시스템의 추상적인 표현을 사용
가시성	시스템이 포함해야 하는 것들을 가시화
관점 모형	이해당사자의 관심사에 따른 모형 제시
의사소통수단	이해당사자 간 원활한 의사소통 수단으로 이용

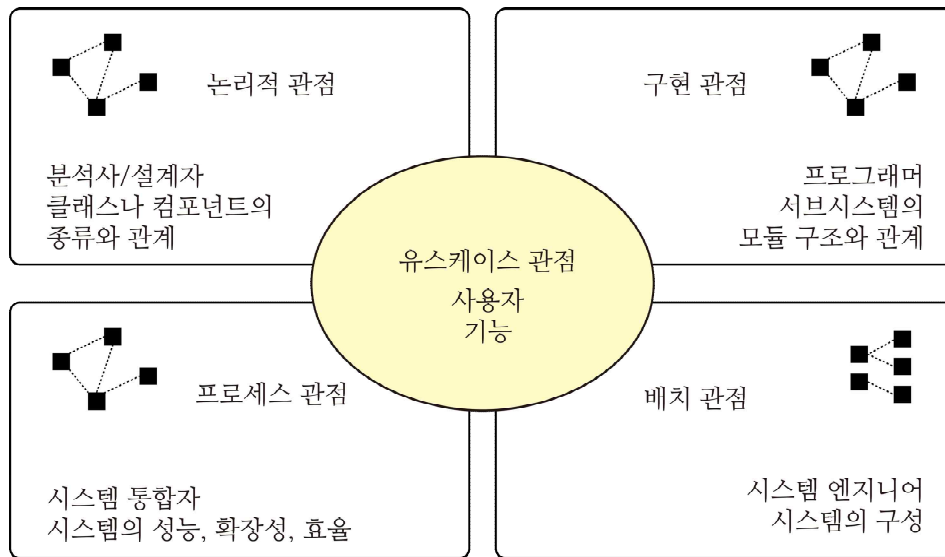
#### (3) 소프트웨어 아키텍처 프레임워크 구성요소

- 아키텍처 명세서 (Architecture Description)
  - 아키텍처를 기록하기 위한 산출물
- 이해관계자 (Stakeholder)
  - 소프트웨어 시스템 개발에 관련된 모든 사람과 조직
  - 고객, 개발자, 프로젝트 관리자 등 포함
- 관심사 (Concerns)
  - 동일한 시스템에 대해 서로 다른 이해관계자 의견
  - 예) 사용자 : 기본적인 기능, 신뢰성, 보안등의 요구
- 관점 (Viewpoint)
  - 서로 다른 역할이나 책임으로 시스템이나 산출물에 대한 서로 다른 관점
- 뷰(View)
  - 이해 관계자들과 이들이 가지는 생각이나 견해로부터 전체 시스템을 표현

#### (4) 소프트웨어 아키텍처 4+1 뷰

- 소프트웨어 아키텍처 4+1 뷰 개념
  - 고객의 요구사항을 정리해 놓은 시나리오를 4개의 관점에서 바라보는 소프트웨어적인 접근 방법
  - 복잡한 소프트웨어 아키텍처를 다양한 이해관계자들이 바라보는 관점
  - View는 시스템의 여러 가지 측면을 고려하기 위한 다양한 관점을 바탕으로 정의

• 4+1 View Model 과 구성요소



구성요소	설명
논리 뷰 (logical view)	<ul style="list-style-type: none"> <li>- 시스템의 기능적인 요구사항</li> <li>- 시스템이 최종 사용자를 위해 해야 하는 것을 나타낸다.</li> </ul>
구현 뷰 (implementation view)	<ul style="list-style-type: none"> <li>- 개발 환경 안에서 정적인 소프트웨어 모듈의 구성</li> <li>- 개발자 관점에서 소프트웨어 구현과 관리적인 측면을 컴포넌트 다이어그램으로 표현</li> </ul>
프로세스 뷰 (process view)	<ul style="list-style-type: none"> <li>- 프로그램 실행시의 시스템 표현</li> <li>- Thread와 Process에 의한 동작을 중점적으로 표현</li> <li>- 동시성, 분산처리, 시스템 통합, 오류 허용 등을 표현</li> </ul>
배치 뷰 (deployment view)	<ul style="list-style-type: none"> <li>- 다양한 실행 파일과 다른 런타임 컴포넌트가 해당 플랫폼 또는 컴퓨팅 노드에 어떻게 매핑 되는가를 표현</li> <li>- 가용성, 신뢰성, 성능, 확장성 등의 시스템의 비기능적인 요구사항 고려</li> <li>- 물리적인 노드의 구성과 상호 연결 관계를 배포 다이어그램으로 표현</li> </ul>
유스케이스 뷰 (use case view)	<ul style="list-style-type: none"> <li>- 아키텍처를 도출하고 설계하는 작업을 주도하는 뷰</li> <li>- 다른 뷰를 검증하는데 사용</li> <li>- Use Case Diagram이 사용</li> <li>- +1 에 해당하며 유스케이스가 나머지 4개 뷰에 모두 참여하면서 영향을 준다.</li> </ul>

**(5) 소프트웨어 아키텍처 품질속성**

- 정확성 (Correctness)
  - 소프트웨어가 사용자의 요구기능을 충족시키는가?
- 신뢰성 (Reliability)
  - 기능이 오차나 오류 없이 동작하는가?
- 효율성 (Efficiency)
  - 기능을 수행하는데 적절한 자원이 소요되는가?
- 무결성 (Integrity)
  - 허용되지 않는 사용이나 자료 변경을 제어하는가?
- 사용 용이성 (Usability)
  - 쉽게 배우고 사용할 수 있는가?
- 유지보수성 (Maintainability)
  - 변경 및 오류 교정 시 쉽게 수정할 수 있는가?
- 시험 용이성 (Testability)
  - 개선, 유지보수 등에 있어서 테스트를 하기 용이하게 되어 있는가?
- 유연성 (Flexibility)
  - 새로운 요구사항에 대해서도 쉽게 개선 및 적용 가능한가?
- 이식성 (Potability)
  - 다양한 플랫폼 및 하드웨어에서 동작 하는가?
- 재사용성 (Reusability)
  - 개발된 기능을 다른 목적으로 사용하기 용이한가?
- 상호 운용성 (Interoperability)
  - 다른 소프트웨어와 상호 교류가 용이한가?

**(6) 소프트웨어 아키텍처 평가**

- 소프트웨어 아키텍처 평가 개념
  - 아키텍처의 접근법이 품질속성(보안, 성능, UI 등)에 미치는 영향을 판단하여 아키텍처 적합성을 판단하고 평가하는 표준 기법
- 소프트웨어 아키텍처 평가기법 유형

관점	유형	내용
가시성	가시적 평가	Inspection, Review, Validation & Verification
	비가시적 평가	SAAM, ATAM, CBAM, ARID, ADR
시점	이른 평가	아키텍처 구축과정 중 어느 때나 평가 가능
	늦은 평가	기존 시스템의 요구사항에 대한 아키텍처의 적합성을 판단할 때 사용

## 2. 소프트웨어 아키텍처 패턴

### (1) 소프트웨어 아키텍처 패턴의 개념

- 주어진 문맥 안에서 소프트웨어 아키텍처의 공통적인 발생 문제에 대한 일반적인, 재사용 가능한 해결책
- 소프트웨어 공학의 다양한 문제를 해결
  - 컴퓨터 하드웨어 성능
  - 비즈니스 위험의 최소화
  - 고가용성
- 일부 아키텍처패턴은 소프트웨어 프레임워크 안에 구현되어 있다.

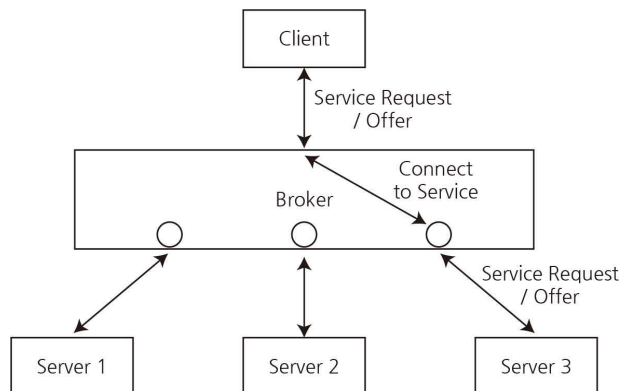
### (2) 소프트웨어 아키텍처 패턴 종류

- 계층화 패턴(Layered pattern)
  - n-티어 아키텍처 패턴으로 부른다.
  - 하위 모듈을 그룹으로 나눌 수 있는 구조화된 프로그램에서 사용
  - 각 서브 시스템이 하나의 계층이 되고 하위층이 제공하는 서비스를 상위층의 서브 시스템이 이용할 수 있는 구조
  - 예) OSI 7계층, TCP/IP 4계층
- 클라이언트-서버 패턴(Client-Server Pattern)
  - 다수의 클라이언트와 하나의 서버로 구성
  - 서버는 클라이언트에게 서비스를 제공하며 데이터를 관리하는 역할
  - 예) 일반적인 웹 프로그램
- 마스터-슬레이브 패턴(Master-Slave Pattern)
  - 마스터 컴포넌트가 동등한 구조의 슬레이브 컴포넌트로 작업을 분산하고, 슬레이브가 결과값을 반환하면 최종 결과값을 계산하는 구조
  - 예) 컴퓨터와 주변장치
- 파이프-필터 패턴(Pipe-Filter Pattern)
  - 데이터 스트림을 생성하고 처리하는 시스템에서 사용 가능한 패턴
  - 필터 컴포넌트에서 각 처리과정을 실행하며, 처리된 데이터는 파이프를 통해 전송
  - 서브시스템이 입력데이터를 받아 처리하고 결과를 다음 서브시스템으로 넘겨주는 과정을 반복
  - 예) Unix 셸처리



- 브로커 패턴(Broker Pattern)

- 분리된 컴포넌트로 구성된 분산 시스템에서 사용되는 패턴
- 각 컴포넌트들은 원격 서비스를 통해 서로 상호작용을 할 수 있으며 브로커 컴포넌트가 컴포넌트 간의 통신을 조절



- 피어 투 피어 패턴(Peer to Peer Pattern)

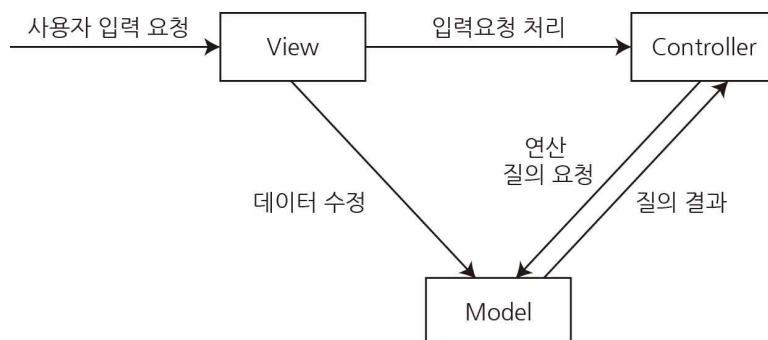
- 피어라 부르는 각 컴포넌트 간에 서비스를 주고 받는 패턴
- 피어 객체 하나가 클라이언트, 서버의 역할을 모두 수행하는 구조
- 예) 파일 공유(P2P)

- 이벤트-버스 패턴(Event-Bus Pattern)

- 이벤트 버스를 통해 특정 채널로 메시지를 발행
- 리스너가 구독한 채널에 소스가 서비스를 제공하면 채널이 리스너에게 서비스를 제공
- 예) 알림 서비스

- 모델-뷰-컨트롤러 패턴(Model-View-Controller Pattern) - MVC Pattern

- 3개의 각 컴포넌트는 각자의 역할을 갖고 사용자에게 서비스를 제공
- 자료의 저장, 제어, 표현 기능을 분리하여 재사용을 증진
- 모델 : 도메인의 기능과 자료를 저장하고 보관
- 뷰 : 사용자에게 결과를 표시
- 컨트롤러 : 사용자로부터 입력을 받아 연산을 처리
- 예) 일반적인 웹 어플리케이션 설계 아키텍처



- 블랙보드 패턴(Blackboard Pattern)

- 명확히 정의된 해결 전략이 알려지지 않은 문제에 대해서 유용한 패턴

- 인터프리터 패턴(Interpreter Pattern)

- 특정 언어로 작성된 프로그램을 해석하는 컴포넌트를 설계할 때 사용되는 패턴

## Section 3. UML

### 1. UML(Unified Modeling Language)

#### (1) UML 개념

- 프로그램 설계를 표현하기 위해 사용하는 표기법
- 시스템 개발 과정에서 이해관계자 사이에 의사소통을 원활하게 이루어지게 하기 위하여 표준화한 모델링 언어
- 소프트웨어 시스템, 업무 모델링, 시스템의 산출물을 규정하고 시각화, 문서화하는 언어
- 프로그램언어가 아닌 기호와 도식을 이용하여 표현하는 방법을 정의한다.

#### (2) UML 특징

- 가시화 언어
  - 소프트웨어의 개념 모델을 시각적인 그래픽 형태로 작성한다.
- 명세화 언어
  - 분석, 설계, 구현 단계의 각 과정에서 필요한 모델을 명세화 할 수 있는 언어
- 구축 언어
  - 명세화된 설계모델은 다양한 언어의 소스코드로 변환하여 구축할 수 있다.
- 문서화 언어
  - 일련의 과정을 문서로 남겨 계속 유지 보수한다.

### 2. UML 구성요소

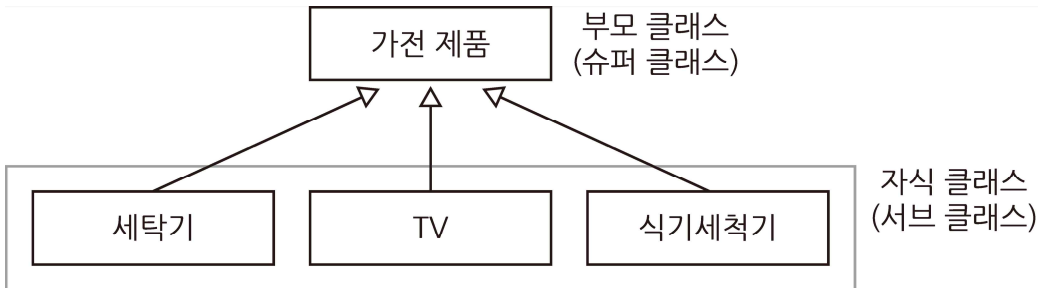
#### (1) 사물(Things)

- 구조사물
  - 시스템의 개념적, 물리적 요소
  - 예) 클래스, 유즈케이스, 컴포넌트 등
- 행동사물
  - 시간과 공간에 따른 요소들의 행위
  - 예) 상호작용, 상태머신
- 그룹사물
  - 요소들을 그룹으로 묶은 것
  - 예) 패키지
- 주해사물
  - 부가적 설명이나 제약조건
  - 예) 주석, 노트

## (2) 관계(Relationships)

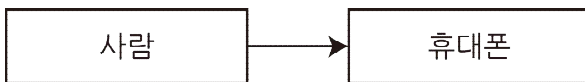
### • 일반화 관계(Generalization)

- 한 클래스가 다른 클래스를 포함하는 상위 개념일 때의 관계
- 객체지향 개념에서는 일반화 관계를 상속관계(Inheritance)라고 함



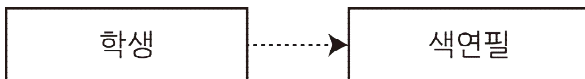
### • 연관관계(Association)

- 2개 이상 사물이 서로 관련된 관계
- 한 클래스가 다른 클래스에서 제공하는 기능을 사용할 때 표시



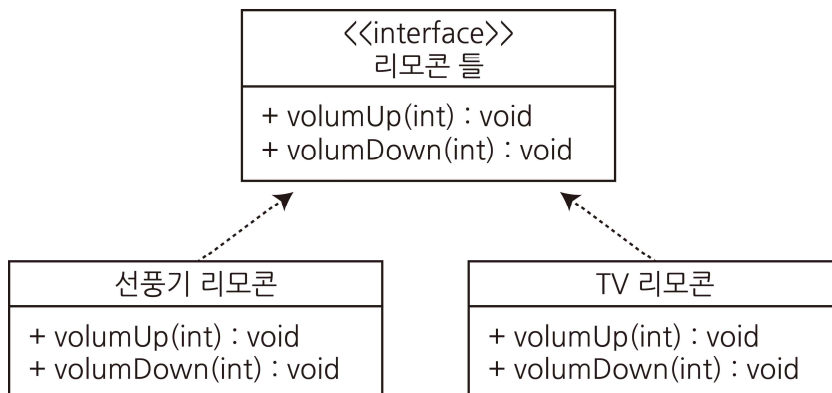
### • 의존관계(Dependency)

- 연관 관계와 같이 한 클래스가 다른 클래스에서 제공하는 기능을 사용할 때 표시
- 연관 관계와 차이점은 두 클래스의 관계가 한 메서드를 실행하는 동안과 같이 매우 짧은 시간만 유지
- 한 클래스의 명세가 바뀌면 다른 클래스에 영향을 줌
- 한 클래스가 다른 클래스를 오퍼레이션의 매개변수로 사용하는 경우

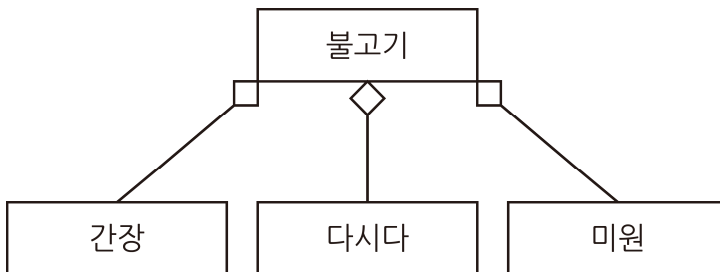


### • 실체화 관계 (Realization)

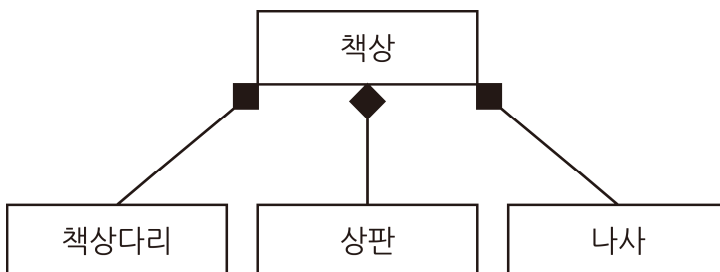
- 인터페이스를 구현받아 추상 메서드를 오버라이딩 하는 것을 의미
- 한 객체가 다른 객체에게 오퍼레이션을 수행하도록 지정



- 집합 관계 - 집약관계 (Aggregation)
  - 한 객체가 다른 객체를 소유하는 'has a' 관계
  - 전체 객체의 라이프타임과 부분 객체의 라이프타임은 독립적
  - 전체 객체가 사라진다 해도 부분 객체는 사라지지 않음



- 집합관계 - 합성관계 (Composition)
  - 부분 객체가 전체 객체에 속하는 관계로 긴밀한 필수적 관계
  - 전체 객체의 라이프타임과 부분 객체의 라이프 타임은 의존적
  - 전체 객체가 없어지면 부분 객체도 없어짐



## (3) 다이어그램(Diagram)

## • 구조 다이어그램

종류	설명
클래스 다이어그램	- 클래스의 속성과 클래스 사이의 관계를 표현 - 시스템 구조 파악 및 구조상 문제점을 도출
객체 다이어그램	- 클래스에 속한 객체(인스턴스)를 특정 시점의 객체와 객체 사이 관계로 표현
컴포넌트 다이어그램	- 컴포넌트 사이 관계나 인터페이스를 표현
배치 다이어그램	- 결과물, 프로세스, 컴포넌트 등 물리적 요소들의 위치를 표현 - 노드와 통신 경로를 표현
복합체 다이어그램	- 클래스나 컴포넌트가 복합구조를 가질 시 그 내부 구조를 표현
패키지 다이어그램	- 유즈케이스나 클래스 등 모델 요소들을 그룹화한 패키지들의 관계 표현

## • 행위 다이어그램

종류	설명
유즈케이스 다이어그램	- 유저의 요구를 분석하며 기능 모델링 작업에 사용 - 시스템의 기능을 나타내기 위해 사용자의 요구를 추출하고 분석하는 데 사용 - 외부에서 보는 시스템의 동작으로, 객체들이 어떻게 상호작용하는지 모델링 - 구성요소 : Actor, Use Case, System
시퀀스 다이어그램	- 특정 행동이 어떠한 순서로 어떤 객체와 상호작용 하는지 표현 - 현재 시스템이 어떠한 시나리오로 움직이고 있는지를 나타냄 - 구성요소 : 활성객체, 메시지, 생명선, 제어사각형 - 메시지유형 : 동기 메시지, 비동기 메시지, 반환 메시지, 자체 메시지
커뮤니케이션 다이어그램	- 동작에 참여한 객체들이 주고받는 메시지와 객체 간 연관까지 표현
상태 다이어그램	- 객체가 자신이 속한 클래스의 상태 변화 및 다른 객체 간 상호작용에 따라 상태 변화 표현
활동 다이어그램	- 시스템이 어떤 기능을 수행하는지에 따라 객체 처리 로직이나 조건에 따른 처리 흐름을 순서에 따라 표현
상호작용 다이어그램	- 상호작용 다이어그램 간 제어 흐름 표현
타이밍 다이어그램	- 객체 상태 변화와 시간 제약 명시적 표현

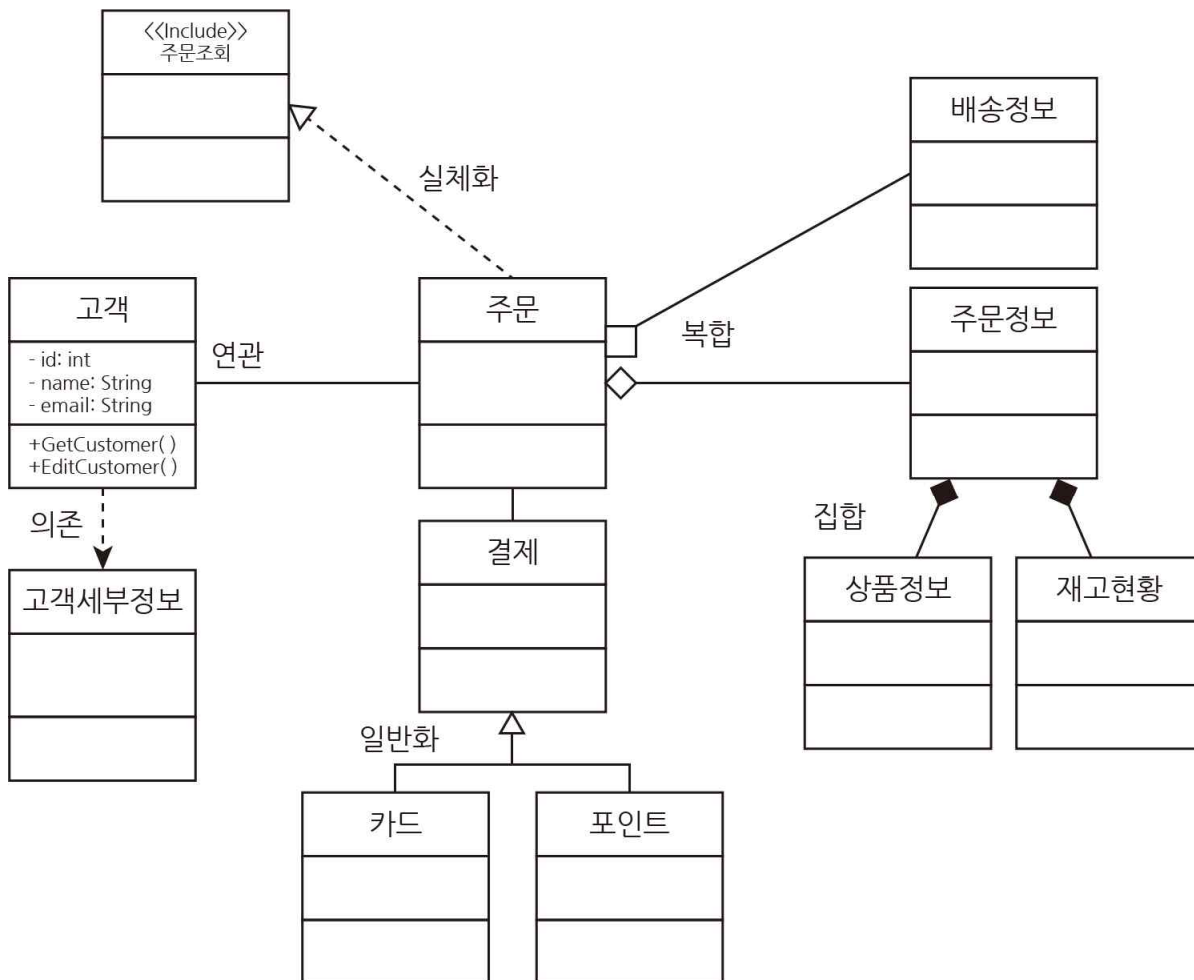
### 3. 주요 다이어그램

#### (1) 클래스 다이어그램

##### ① 클래스 다이어그램 개념

- 자기만의 속성(attribute)과 일정한 행동(behavior)으로 구성
- 여러 개의 클래스들은 서로 연관이나 상속, 의존 관계 등으로 서로 간의 상호작용을 표현

##### ② 클래스 다이어그램 표현



##### ③ 접근 제한자 표기법

표기법	접근 제한자	사용 범위
-	private	해당 클래스 내에서만 접근 가능
#	protected	상속, 동일 패키지 내에서만 접근 가능
+	public	어디서든 접근 가능

## (2) 유스케이스 다이어그램

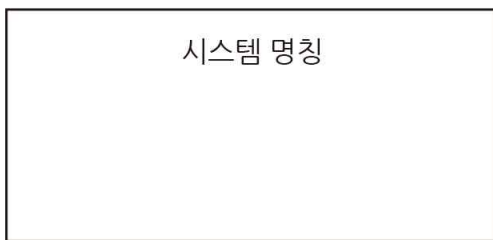
### ① 유스케이스 다이어그램 개념

- 시스템과 사용자의 상호작용을 다이어그램으로 표현
- 사용자의 관점에서 시스템의 서비스 혹은 기능 및 그와 관련한 외부 요소를 보여준다.
- 프로젝트에 대한 요구사항을 정의하고 세부기능을 분석하며 개발 범위를 정할 때 작성한다.

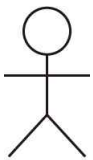
### ② 유스케이스 다이어그램 구성요소

- 시스템(System)
  - 만들고자 하는 프로그램 명칭

〈시스템의 표현방법〉



- 액터(Actor)
  - 시스템의 외부에 있고 시스템과 상호작용을 하는 사람, 시스템을 표현



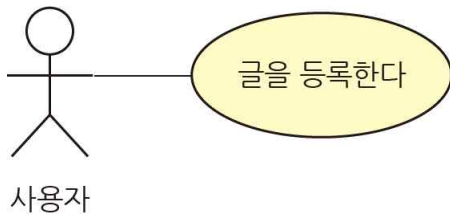
액터명

- 유스케이스(Usecase)
  - 사용자 입장에서 바라본 시스템의 기능

〈유스케이스의 표현방법〉

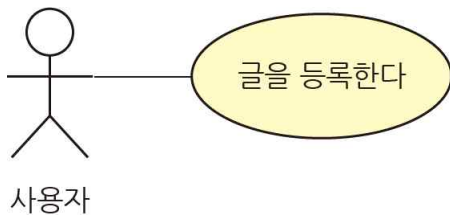


- 관계(Relation)
  - 액터와 유스케이스 사이의 의미있는 관계

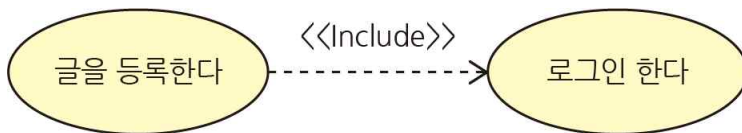


### ③ 유스케이스 다이어그램

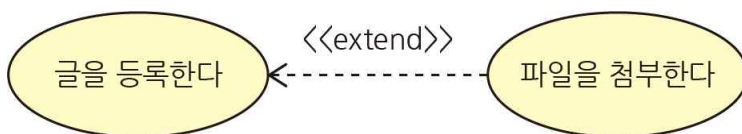
- 연관관계(Association)
  - 유스케이스와 액터간의 상호작용이 있음을 표현
  - 유스케이스와 액터를 실선으로 연결



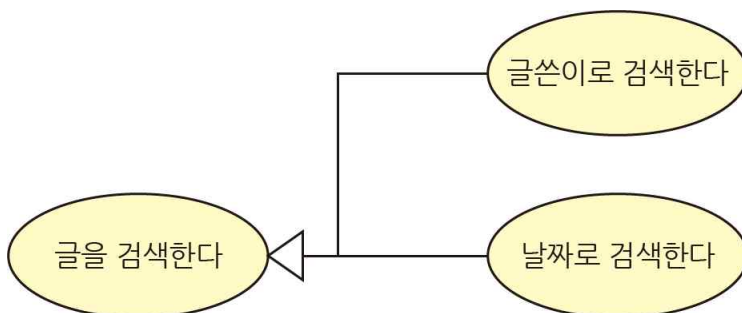
- 포함 관계(Include)
  - 유스케이스를 수행 할 때 반드시 실행되어야 하는 경우



- 확장 관계(Extend)
  - 유스케이스를 수행 할 때 특정 조건에 따라 확장 기능 유스케이스를 수행하는 경우



- 일반화 관계(Generalization)
  - 유사한 유스케이스 또는 액터를 모아 추상화한 유스케이스





### (3) 시퀀스 다이어그램

#### ① 시퀀스 다이어그램 개념

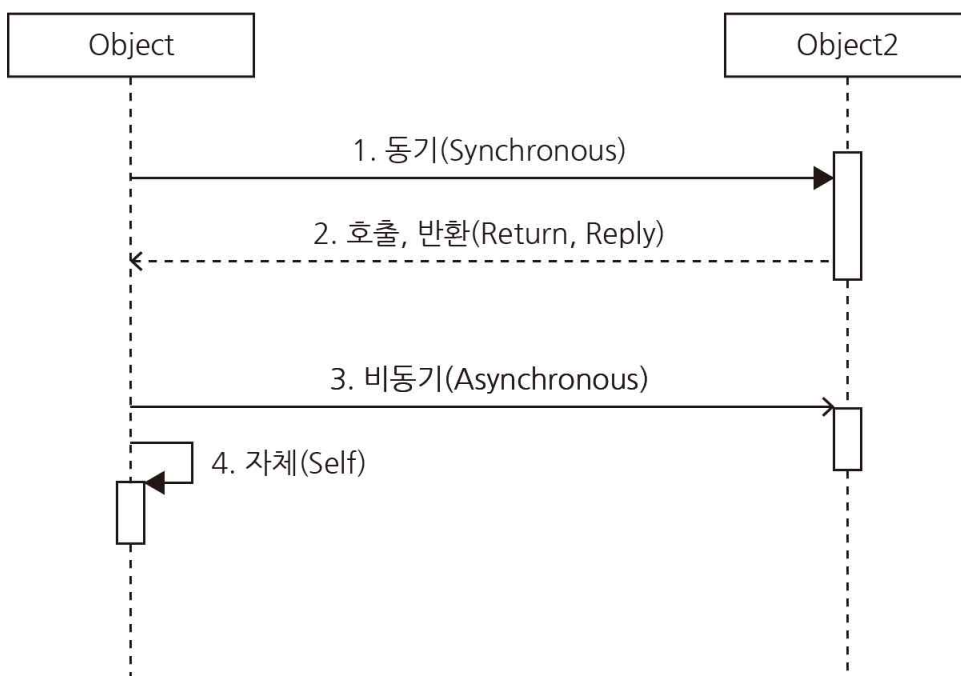
- 객체간의 상호작용 메시지 시퀀스를 시간의 흐름에 따라 나타내는 다이어그램

#### ② 시퀀스 다이어그램 구성요소

- 객체(Object)와 생명선(Lifeline)
  - 객체(활동 주체)는 직사각형으로 표현
  - 라이프라인은 객체에서 이어지는 점선으로 표현
  - 점선은 위에서 아래로 갈수록 시간의 경과를 의미
- 활성화 박스(Activation Box)
  - 생명선상에서 길다란 직사각형으로 표현
  - 현재 객체가 어떤 활동을 하고 있음을 의미
- 메시지(Message)
  - 인스턴스 간 주고 받은 데이터
  - 메시지의 유형

유형	설명
동기 메시지 (Sync message)	요청을 보낸 후에 반환이 올 때까지 대기
비동기 메시지 (Async message)	요청을 보낸 다음 반환을 기다리지 않고 다른 작업을 수행
자체 메시지 (Self message)	자기 자신에게 요청을 보냄
반환 메시지 (Reply/Return message)	요청에 대해 메시지를 반환

- 메시지의 표현

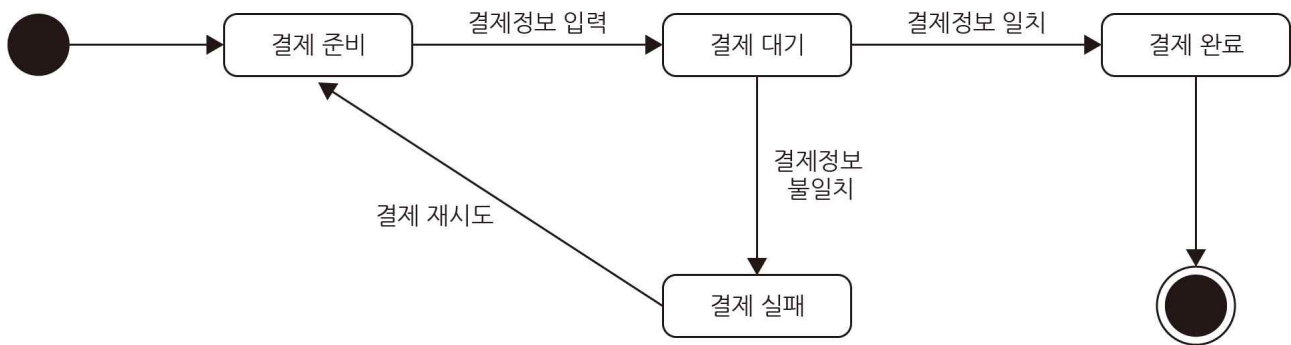


#### (4) 상태 다이어그램

##### ① 상태 다이어그램 개념

- 한 객체의 상태 변화를 나타내는 다이어그램

##### ② 상태 다이어그램 예시



## 04 화면 설계

### Section 1. UI 설계

#### 1. UI (User Interface) 개념

##### (1) UI 개념

- 컴퓨터, 웹 사이트, 시스템 등의 정보기기와 사용자가 서로 상호작용을 할 수 있도록 연결해주는 매개체
- 디스플레이 화면, 아이콘, 검색창, 키보드, 문자, 색상, 폰트 등의 여러 요소들이 포함
- UI의 핵심은 특별한 설명이 없이, 사용자 누구나 알아보기 쉽고, 편리하게 이용할 수 있도록 만든 보편적이고 직관적인 디자인 이어야 함

##### (2) UX (User eXperience) 개념

- 사용자가 컴퓨터, 웹 사이트, 시스템 등 정보기기의 UI를 직/간접적으로 이용하여 경험한 모든 것
- 모바일 화면에 대한 사용자의 경험, 포장박스를 개봉할 때의 느낌, 디바이스 인터페이스를 사용하며 느끼는 만족감이나 불편함 등 사용할 때의 모든 행동, 느낌, 감정 등이 UX에 포함
- 사용자들의 경험을 분석하여 사용자의 불만족을 최소화하고, 보다 편리하게 이용할 수 있도록 디자인
- UI를 기반으로 사용자들의 공감과 만족을 이끌어내는 역할이 UX

##### (3) UI 유형

- CLI (Command Line Interface)
  - 사용자가 컴퓨터 자판을 이용해 명령을 입력하여 컴퓨터를 조작하는 시스템
- GUI (Graphical User Interface)
  - 그래픽과 텍스트로 이루어져 있어, 사용자의 입력이나 출력이 마우스 등을 통해 이루어짐
- AUI (Auditory User Interface)
  - 보다 나은 사용자 경험을 제공하기 위해 만들어진 임베디드 사운드
- NUI (Natural User Interface)
  - 특별한 하드웨어 없이 인간의 자연스러운 움직임을 인식하여 정보를 제공

#### 2. UI 설계

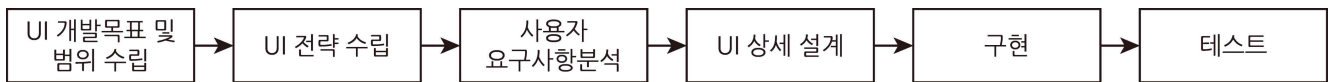
##### (1) UI 요구사항 구분

###### ① 기능적 요구사항

- 시스템이 제공해야 하는 기능에 대한 요구사항
- 입력, 출력, 데이터, 연산에 관한 요구사항

###### ② 비기능적 요구사항

- 사용성, 효율성, 신뢰성, 유지 보수성, 재사용성 등 품질에 관한 요구사항
- 플랫폼, 사용 기술 등 시스템 환경에 관한 요구사항
- 비용, 일정 등 프로젝트 계획에 관한 요구사항

**(2) UI 설계 절차****① UI 개발목표 및 범위 수립**

- 해당 프로젝트의 UI 부분의 목표와 범위에 대한 계획을 수립하는 단계
- 프로젝트 계획 전반에 UI 계획을 반영한다.
- 이해 당사자의 UI 요구 사항을 조사하고, 정의한다.
- UI 트렌드 및 해당 서비스의 사용자를 분석한다.

**② UI 전략 수립**

- 서비스를 사용할 사용자 조사 및 시장 조사를 통해 UI, UX 전략을 수립하는 단계
- 기술적 관점에서도 전략을 수립한다.

**③ 사용자 요구사항 분석**

- 사용자 조사 결과를 기반으로 사용자 요구사항을 추출하고 선별하여 요구사항을 분석하는 단계
- 프로토타입(Prototype)을 제작

**④ UI 상세 설계**

- UI 기능에 대한 구조와 화면 간의 상호 흐름, 예외 처리 등의 UI 전반에 걸친 설계를 하는 단계

**⑤ 구현**

- HTML5, CSS3, JavaScript 등의 언어들로 구현하는 단계

**⑥ 테스트**

- 소프트웨어의 사용성을 검증하는 단계

**(3) UI 설계 원칙**

- 직관성
  - 누구나 쉽게 이해하고 사용할 수 있어야 한다.
- 유효성
  - 사용자의 목적을 정확하게 달성해야 한다.
- 학습성
  - 누구나 쉽게 배우고 익힐 수 있어야 한다.
- 유연성
  - 사용자의 요구사항을 최대한 수용하며, 오류를 최소화해야 한다.

**(4) UI 설계 도구****① 와이어프레임(Wireframe)**

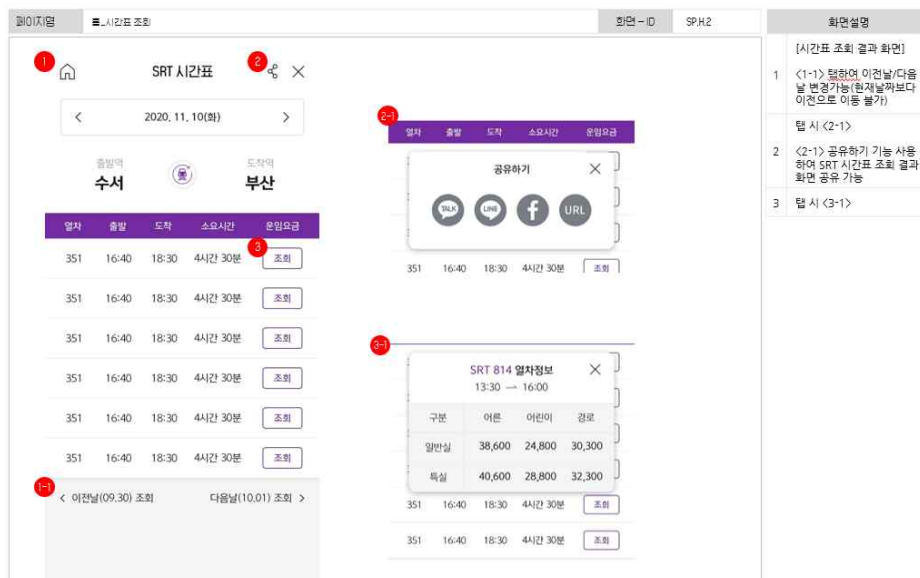
- 와이어프레임의 개념
  - 선(Wire)으로 틀(Frame)을 잡는다는 뜻
  - 화면 단위의 레이아웃을 설계하는 작업
  - 간단한 모양만을 사용하여 인터페이스를 시각적으로 묘사
  - 와이어프레임에는 내용, 구조, 흐름, 기능 등이 포함
  - 제작 방법에 제한은 없지만, 실무에서는 손그림이나 파워포인트로 작성

## ② 스토리보드

- 개발 후 완성된 콘텐츠의 최종 결과를 예상할 수 있는 기초 문서
- 디자이너와 개발자가 최종적으로 참고하는 설계 산출 문서
- 정책, 프로세스 및 콘텐츠 구성, 와이어 프레임, 기능 정의 등 서비스 구축을 위한 대부분의 정보가 수록
- 스토리보드 작성 절차

작성문서	설명
표지	- 프로젝트명, 문서버전, 최종업데이트 일자, 작성자 등을 작성한다.
문서의 버전	- 변경된 문서버전, 변경일, 변경된 내용, 작성자 등이 포함된다.
Index 작성	- 스토리보드를 빠르게 찾아갈 수 있는 차례를 작성한다.
IA(Information Architecture) 작성	- 소프트웨어의 사이트 맵을 작성한다.
공통모듈 작성	- 스토리보드 전반에서 사용될 공통 모듈을 작성한다. - 웹의 경우 기본 구성화면 (Header, Footer, GNB, LNB 등) 등의 내용을 공통 모듈 정의서로 작성한다.
화면설계와 description 작성	- 실제로 구현될 화면을 설계한다. - 해당 화면에서 설명이 필요한 부분에 넘버링을 하고, 설명을 따로 적어둔다.

### • 스토리보드 작성 예시



## ③ 프로토타입

- 프로토타입은 실제 서비스와 흡사한 모형을 만드는 작업
- 와이어프레임이나, 스토리보드에 인터랙션(동적효과)을 적용함으로써 실제 구현된 것처럼 시뮬레이션 할 수 있다.
- 실제 개발하는 것보다 단시간에 구현이 가능하기 때문에 사용자 경험에 대한 테스트를 진행할 수 있다.

## ④ 목업(Mockup)

- 와이어프레임보다 좀 더 실제 화면과 유사하게 만든 정적인 형태의 모형
- 시간적으로만 구성 요소를 배치하는 것으로 실제로 구현되지는 않는다

### ⑤ 유스케이스(Use Case)

- 사용자 측면에서의 요구사항으로, 사용자가 원하는 목표를 달성하기 위해 수행할 내용을 기술한다

## 3. 감성공학

### (1) 감성공학의 개념

- 인간의 심상을 구체적인 물리적 설계 요소로 번역하여 이를 실현하는 기술
- 인간의 감성과 이미지를 물리적인 디자인 요소로 해석하여 구체적 제품으로 만들어내는 공학적 접근 방법
- 감성공학 = 마음속의 이미지 파악(심리학) + 형상화(인간공학) + 구체적인 제품 생산(생산공학)
- 요소화 → 형상화 → 구현 → 생산

### (2) 제품과 관련된 인간의 감성

- 감각적 감성
  - 제품에 관하여 사용자가 느끼는 감성, 제품의 외관, 색상, 디자인에 관련된 것
- 기능적 감성
  - 제품의 성능과 사용시 편리함에 대한 것
- 문화적 감성
  - 개인이 속한 사회나 문화에 관련된 것

### (3) 감성공학의 접근 방법

- 1류 접근 방법
  - 의미 미분법
  - 인간의 감성을 표현하는 어휘를 이용하여 제품에 대한 이미지를 조사하고, 그 분석을 통해 제품 디자인 요소 연계
- 2류 접근 방법
  - 문화적 감성의 일부를 반영한 개념
  - 1류와 기본틀은 같으나, 감성어휘 수집의 전 단계에서 평가자들의 생활 양식을 고려하는 것이 추가
  - 제품에 대한 소비자군의 소속지역, 생활양식, 의식문화가 많은 영향을 미칠 때 사용하는 접근 방법
  - 1류와 함께 감성의 심리적 특성을 강조한 접근 방법
- 3류 접근 방법
  - 특정 시제품을 사용하여 감각 척도를 계측하고, 정량화된 값을 환산
  - 대상 제품의 물리적 특성에 대한 객관적 지표 연관분석을 통해 제품 설계에 응용
  - 인간 감각계측과 이의 활용이 강조된 접근 방법

## 4. UI 설계 지침

### (1) 한국 HCI 연구회 설계 지침

- 가시성의 원칙 (Visibility)
  - 소프트웨어의 기능을 노출 시켜 최대한 조작이 쉽도록 한다.
- 조작 결과 예측의 원칙 (Natural Mapping)
  - 사용자가 소프트웨어를 조작하여 작동시킨 결과를 조작 부위만 보고도 예측 가능하게 설계 한다.
- 일관성의 원칙 (Consistency)
  - 소프트웨어의 조작방식에 일관성을 제공하여 사용자가 쉽게 기억하고 빠르게 적응할 수 있게 한다.
- 단순성의 원칙 (Simplicity)
  - 소프트웨어의 기능구조를 단순화시켜 조작에 요구되는 노력을 최소화한다.
- 지식 배분의 원칙 (Knowledge in World & Head)
  - 학습하기 쉽고 기억하기 쉽게 설계해야 한다.
- 조작오류의 원칙 (Design for Error)
  - 사용간 발생한 오류를 쉽게 발견하고 수정 또한 쉽게 이루어져야 한다.
- 제한사항 선택사용의 원칙 (Constraints)
  - 소프트웨어를 조작할 때 선택의 여지를 줄여 조작 방법이 명확하도록 유도한다.
- 표준화의 원칙 (Standardization)
  - 소프트웨어의 기능 구조와 디자인을 한 번 학습한 이후 보다 효과적으로 서비스를 사용할 수 있어야 한다.
- 행동 유도성의 원칙 (Affordance)
  - 사용자가 디자인을 보고 기능 및 조작법을 유추할 수 있도록 해야 한다.
- 접근성의 원칙 (Accessibility)
  - 사용자의 성별, 연령등 다양한 계층의 사용자가 받아드릴 수 있는 사용자인터페이스를 구축해야 한다.

### (2) 전자정부 웹 사이트 UI·UX 설계 기준

- 사용자에게 필요한 정보와 기능을 제공한다.
- 작업에 소요되는 시간과 단계를 최소화 한다.
- 직관적이고 일관성 있게 만든다.
- 사용자가 원하는 방식으로 이용할 수 있게 만든다.
- 사용자가 실수하지 않게 만든다.
- 모든 유형의 사용자가 이용할 수 있게 만든다.
- 원하는 서비스와 정보를 쉽게 찾을 수 있게 만든다.

## Section 2. UI 구현

### 1. 화면 레이아웃 구성

#### (1) 레이아웃(Layout)의 개념

- 특정 공간에 여러 구성 요소를 보기 좋게 효과적으로 배치하는 작업
- 레이아웃 작성 방법
  - DIV 요소를 이용한 레이아웃
  - SPAN 요소를 이용한 레이아웃
  - TABLE 요소를 이용한 레이아웃
  - 시맨틱(semantic) 태그를 이용한 레이아웃

#### (2) HTML5

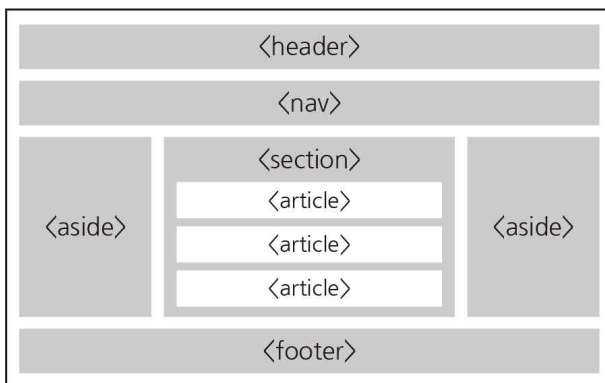
##### ① HTML5 개념

- 월드와이드웹(World Wide Web)을 통해 제공되는 정보를 나타낼 목적으로 사용되는 마크업 언어
- HTML의 5번째 버전을 의미한다.
- 웹 페이지의 기본 구조를 담당

##### ② HTML5 특징

- 플러그인의 설치 없이 동영상이나 음악을 웹 브라우저 상에서 재생
- 다양한 2차원 그래픽 표현 가능(SVG 태그, CANVAS 이용)
- 실시간으로 서버와 양방향 통신 가능
- GPS를 통한 위치 확인 및 장치 접근이 가능
- 오프라인 상태에서도 작업이 가능

##### ③ 시맨틱 요소



TAG	설명
<header>	- 헤더 영역 - 제목, 내비게이션, 검색 등의 내용을 포함
<nav>	- 메인 메뉴나 목차 등을 정의
<section>	- 맥락이 같은 요소들을 주제별로 그룹화
<article>	- 본문의 주요 내용이 들어가는 공간
<aside>	- 본문 외의 콘텐츠 영역 - 사이드 메뉴나 광고 등의 영역
<footer>	- 푸터 영역 - 작성자나 회사 정보 등을 포함



## ④ INPUT 요소

Email

Password

Textarea 

Textarea

Radios ☒ Default radio  
☐ Second default radio  
☐ Disabled radio

Checkbox ☐ Check me out

- 텍스트 입력 (text, textarea)
  - text : 한 줄의 텍스트 입력
  - textarea : 여러 줄의 텍스트 입력
- 비밀번호 입력 (password)
  - 입력받은 문자를 별표나 작은 원으로 표시
- 라디오 버튼 (radio)
  - 여러 개의 라디오 버튼 옵션 중에서 단 하나만의 값을 선택
- 체크 박스 (checkbox)
  - 여러 개의 체크박스 중에서 여러 개의 옵션 값을 선택
- 파일 선택 (file)
  - 사용자 컴퓨터의 파일을 입력
- 선택 입력 (select)
  - 여러 개의 드롭다운 리스트(drop-down-list) 중에서 한 개의 옵션을 선택
- 버튼 (button)
  - 사용자가 클릭했을 때 작업을 수행
- 전송 (submit)
  - 입력 받은 데이터를 서버로 전송
- 필드셋 (fieldset)
  - 관련된 데이터를 하나로 묶어준다.

### (3) CSS(Cascading Style Sheet)

#### ① CSS 개념

- HTML과 함께 웹을 구성하는 기본 프로그래밍 요소
- 색상이나 크기, 이미지 크기나 위치, 배치 방법 등 웹 문서의 디자인 요소를 담당

#### ② CSS 특징

- HTML로 부터 디자인적인 요소를 분리해 정의할 수 있다.
- 잘 정의된 css 는 서로 다른 여러 웹페이지에 적용할 수 있다.
- 자바스크립트와 연계해 동적인 콘텐츠 표현이나 디자인 적용 가능
- 다양한 기기(PC, 스마트폰, 태블릿 등)에 맞게 탄력적으로 바뀌는 콘텐츠(반응형 웹) 적용 가능

#### ③ CSS3

- 이전 버전 CSS와 완전히 호환되는 CSS의 최신 표준 권고안
- 차세대 웹 개발을 위한 새로운 표준

### (4) JavaScript

#### ① JavaScript 개념

- 모질라 재단의 프로토타입 기반의 프로그래밍 언어로, 스크립트 언어에 해당된다.
- HTML, CSS와 함께 웹을 구성하는 요소 중 하나이다.
- 클라이언트 단에서 웹 페이지가 동작하는 것을 담당

#### ② JavaScript 특징

- 웹 문서에 삽입해서 사용하는 스크립트 언어
- 이벤트 중심(event-driven)의 프로그래밍 언어
- 프로토타입 기반 객체 지향 언어
- 값에 따라 변수의 형변환이 자동으로 이루어지는 동적 형변환
- 주로 웹 브라우저에서 사용되나, Node.js 와 같은 프레임워크를 사용하면 서버측 프로그래밍도 가능

#### ③ JavaScript 프레임워크

- React
  - 유저 인터페이스를 만드는 데 사용되는 오픈 소스 자바스크립트 라이브러리
  - 페이스북에서 개발
  - 싱글 페이지 애플리케이션(SPA)이나 모바일 애플리케이션 개발에 사용될 수 있다.
- Vue.js
  - 자바스크립트로 개발된 컴포넌트 구조 기반 프론트엔드 프레임워크
  - 고성능의 싱글 페이지 애플리케이션(SPA)을 구축하는데 이용가능하다.
  - Evan You에 의해 개발
- AngularJS
  - 자바스크립트 기반의 오픈 소스 프론트엔드 웹 애플리케이션 프레임워크
  - 구글에서 개발
- Ajax(Asynchronous JavaScript and XML)
  - 비동기적인 웹 애플리케이션의 제작을 위한 웹 개발 기법

## 2. UI 관련용어

용어	설명
웹 표준	월드 와이드 웹의 측면을 서술하고 정의하는 공식 표준이나 다른 기술 규격을 말한다.
웹 호환성	이용자의 단말기 (PC, 모바일 기기 등)의 하드웨어 및 소프트웨어 환경이 다른 경우에도 동등한 서비스를 제공할 수 있는 것을 말한다.
웹 접근성	장애인과 비장애인 모두가 동등하게 웹 사이트에 접근하여 이용할 수 있도록 보장하는 방식을 말한다.
반응형 웹	PC, Mobile 등 다양한 디바이스에서 화면크기에 맞춰 하나의 사이트를 보여준다.
인포그래픽 (Infographic)	정보(Information)와 그래픽(Graphic)의 합성어로, 복잡한 정보를 쉽고 빠르게 전달하기 위해 정보를 분석, 정리하여 차트, 그래프, 아이콘, 그래픽스, 이미지 등을 활용하여 시각화한 것을 말한다.
브랜드 아이덴티티(BI)	사용자에게 전달하고자 하는 특정 브랜드의 가치와 의미를 반영한 심적 표상을 말한다.
네비게이션 (Navigation)	하이퍼링크를 따라 웹 공간의 정보를 요청하고 받아오는 웹 브라우징을 의미하며, 웹 사이트를 탐색하기 위한 도구를 뜻하기도 한다.
아코디언(Accordion)	사용자가 원하는 정보만 선택적으로 볼 수 있게 접을 수 있는 내용 패널을 말한다.
플레이스 홀더 (Placeholder)	사용자가 값을 입력하는 데 참고할 수 있도록, 입력 필드에 제공되는 간략한 텍스트 도움말을 말한다.
필터링(Filtering)	원하지 않는 데이터를 차단하거나, 원하는 데이터만 볼 수 있도록 해주는 기능을 말한다.
입력 폼(Input Form)	다양한 입력 필드로 구성되어, 사용자가 웹 서버로 전송할 정보를 입력 할 수 있는 웹 문서의 일부를 말한다.
입력 필드(Input Field)	사용자가 정보를 입력하거나 선택하는데 이용되는 사용자 인터페이스 요소를 말한다.
썸네일(Thumbnail)	커다란 이미지를 축소하여 제공한 이미지를 말한다.
레이블(Label)	입력폼을 구성하는 다양한 입력 필드를 식별하기 위해 사용하는 명칭을 말한다.
대체 텍스트 (Alternative Text)	콘텐츠를 대신하기 위해 제공되는 텍스트를 의미한다.
초점(Focus)	웹 페이지에서 사용자가 선택한 해당 요소에 있을 때, 해당 요소에 Focus가 있다고 한다.

## 05 서버 프로그램 구현

### Section 1. 개발 환경 구축

#### 1. 서버 환경 구축

##### (1) 웹 서버 (WEB)

- 클라이언트에게 정적 파일(HTML, CSS, JS, 이미지)을 제공하는 웹서버 어플리케이션이 설치된 하드웨어
- 이미지, CSS, JS, HTML 문서를 클라이언트에게 전달
- Apache Web Server, IIS, nginx, GWS 등

##### (2) 웹 어플리케이션 서버 (WAS)

- 동적인 웹 서비스를 제공하기 위한 미들웨어가 설치된 하드웨어
- 클라이언트 요청에 맞는 동적인 콘텐츠를 생성한다.
- DB조회나 다양한 로직을 처리한다.
- Web Logic, Web Spere, Jeus, Tomcat 등

##### (3) 데이터베이스 서버 (DBMS)

- 데이터의 저장과 관리를 위한 데이터베이스 소프트웨어가 설치된 하드웨어
- Oracle, MySQL, MS-SQL 등

##### (4) 파일서버

- 사용자의 파일을 저장하고, 파일을 공유할 목적으로 구성된 하드웨어

##### (5) Load Balancer

- 여러 대의 서버가 존재할 경우 요청을 적절히 분배해주는 역할
- 분배 방식

종류	설명
Random	- 요청을 랜덤으로 분배한다.
Least loaded	- 가장 적은 양의 작업을 처리하고 있는 서버에게 요청을 할당한다.
Round Robin	- 순서를 정하여 돌아가며 작업 분배한다.

##### (6) CDN(Content Delivery Network)

- 용량이 큰 콘텐츠 데이터(이미지, 비디오 등)를 빠른 속도로 제공하기 위해 사용자와 가까운 곳에 분산되어 있는 데이터 저장 서버
- 클라이언트는 용량이 큰 콘텐츠 데이터를 가까운 CDN에 요청해 멀리 있는 웹서버에서 직접 받는 것보다 빠르게 받을 수 있다.

## (7) 시스템 아키텍처 고려사항

- 확장성 (Scalability)
  - 클라이언트 수가 늘어났을 때 무리 없이 요청을 처리할 수 있는 확장성을 고려
- 성능 (Performance)
  - 요청한 내용을 정확하고 빠르게 돌려주어야 한다.
- 응답 시간 (Latency)
  - 모든 요청은 클라이언트가 불편해하지 않을 정도로 빠른 시간 안에 돌려주어야 한다.
- 처리량 (Throughput)
  - 같은 시간 안에 더욱 많은 요청을 처리
- 접근성 (Availability)
  - 사용자가 언제든지 시스템에 요청을 보내서 응답을 받을 수 있어야 한다.
- 일관성 (Consistency)
  - 사용자가 서버에 보낸 요청이 올바르게 반영되어야 하고, 일정한 결과를 돌려주어야 한다.

## 2. 개발 소프트웨어 환경

### (1) 시스템 소프트웨어

#### ① 운영체제(OS, Operation System)

- 하드웨어 운영을 위한 운영체제
- Windows, Linux, UNIX 등의 환경으로 구성됨

#### ② JVM(Java Virtual Machine)

- JAVA 관련 프로그램을 기동하기 위한 환경
- 모든 개발자가 동일한 버전을 적용하는 것이 좋다.

#### ③ Web Server

- 정적 웹 서비스를 수행하는 미들웨어
- 웹 브라우저 화면에서 요청하는 정적 파일을 제공한다. (Image, CSS, Javascript, HTML 등)
- Apache, Nginx, IIS(Internet Information Server), GWS(Google Webserver) 등

#### ④ WAS(Web Application Server)

- 동적 웹 서비스를 수행하는 미들웨어
- Tomcat, Undertow, JEUS, Weblogic, Websphere 등

#### ⑤ DBMS(Database Management System)

- 데이터 저장과 관리를 위한 데이터베이스 소프트웨어
- Oracle, DB2, Sybase, SQL Server, MySQL 등

## (2) 개발 소프트웨어

### ① 요구사항 관리 도구

- 고객의 요구사항을 수집, 분석, 추적을 쉽게 할 수 있도록 지원한다.
- JFeature, JRequisite, OSRMT, Trello 등

### ② 설계/모델링 도구

- 기능을 논리적으로 표현할 수 있는 통합 모델링 언어(UML) 지원
- Database 설계를 지원하는 도구
- ArgoUML, StarUML, DB Designer 등

### ③ 구현도구

- 소프트웨어 언어를 통해 구현 및 개발을 지원하는 도구
- Eclipse, IntelliJ, Visual Studio 등

### ④ 테스트 도구

- 개발된 모듈들에 대하여 요구 사항에 적합하게 구현되어 있는지 테스트를 지원하는 도구
- 개발된 모듈들에 대한 오류가 없는지 테스트를 지원하는 도구
- 개발된 모듈들에 대하여 성능을 테스트 하는 도구
- JUnit, CppUnit, JMeter, SpringTest 등

### ⑤ 형상관리 도구

- 산출물 및 소스코드의 변경 사항을 버전별로 관리하여, 목표 시스템의 품질 향상을 지원하는 도구
- Git, CVS, SVN 등

### 3. IDE(Integrated Development Environment) 도구

#### (1) IDE 도구의 개념

- 소프트웨어 개발에 필요한 많은 도구의 기능을 하나로 묶어 활용하는 소프트웨어
- 코딩, 디버그, 컴파일, 배포 등 프로그램 개발에 관련된 모든 작업을 하나의 프로그램 안에서 처리하는 환경을 제공하는 소프트웨어
- 기존의 소프트웨어 개발에서는 컴파일러, 텍스트 편집기, 디버거 등을 따로 사용했으나, 편리한 개발을 위해 하나로 묶은 대화형 인터페이스를 제공

#### (2) IDE 도구의 기능

- 소스코드를 작성하기 위한 텍스트 에디터
- 작성한 코드를 기계어로 변환해주는 컴파일
- 작성한 코드에 문제가 없는지 체크해주는 디버거
- 완성된 프로그램을 서버에 업로드 하는 배포
- 추가적인 기능을 제공하는 플러그인

#### (3) IDE 도구의 종류

- 이클립스 (Eclipse)
- 비주얼 스튜디오 (Visual Studio)
- 엑스코드 (X Code)
- IntelliJ IDEA

#### (4) IDE 도구 선정시 고려 사항

기준	설명
적정성	- 대상 업무에 적절한 도구 선정
효율성	- 프로그래밍의 효율성 고려
이식성	- 여러 OS에 개발환경 설치 가능
친밀성	- 프로그래머가 익숙한 언어 및 도구
범용성	- 다양한 개발 사례가 존재

### 4. 협업 도구

#### (1) 협업 도구의 개념

- 여러 사용자가 각기 별개의 작업 환경에서 통합된 하나의 프로젝트를 동시에 수행할 수 있도록 도와주는 소프트웨어
- 소프트웨어 개발을 진행하는 데는 개발자 뿐만 아니라, 디자이너, 기획자, 현업 관리자 등 많은 사람들이 진행을 하게 되고, 공통의 주제인 소프트웨어 개발을 공유해야 한다.
- 구성원 간 일어나는 모든 커뮤니케이션을 하나의 채널에서 가능하게끔 만들어 준다.
- 협업툴의 주요 기능과 형태는 서비스마다 다르지만 대부분 소프트웨어형 서비스(SaaS) 클라우드 기반으로 한다.

#### (2) 협업 도구의 기능

- 전사관리 : 전자결재, 조직도 등
- 프로젝트 관리 : 캘린더, 타임라인, 간트차트, 대시보드 등
- 자체 드라이브 공간
- 문서 공유 지원
- 커뮤니케이션
- 다국어지원
- 타 협업툴간 연동 지원

#### (3) 협업 도구의 분류

- SNS 형
  - 슬랙, 야머, 아지트, 잔디, 워크플레이스 등
- 프로젝트 관리형
  - 트렐로, 구글 스프레드시트, 노션, 아사나 등
- 통합형
  - 콜라비, 플로우, 쿵, 드롭박스 비즈니스 등



#### (4) 협업툴 도입 이유

- 기존 사내 메신저가 불편하다.
- 사용하는 툴이 너무 많다.
- 프로젝트의 일정이 제대로 공유되지 못하여 스케줄에 이상이 생긴다.
- 인수인계가 원활하지 못하다.
- 자료를 각자 가지고 있어 다른 팀원에게 요청해야 한다.
- 서로의 업무를 모른다.

#### (5) 협업 도구 도입 프로세스

- 문제정의
- 문제에 대한 솔루션, 기대효과 정의
- 협업 도구 분석
- 협업 도구 최종 선정

### 5. 형상 관리 도구

#### (1) 형상 관리 도구의 개념

- 소프트웨어 생명주기 동안 발생하는 변경사항을 통제하기 위한 관리 방법
- 소프트웨어의 변경사항을 체계적으로 관리하는 것

#### (2) 형상 관리의 필요성

- 개발 도중 소스코드를 이전 상태로 되돌릴 필요가 있을 경우
- 각 변경점에 대한 이력 확인
- 여러 개발자의 동시 개발에 따른 충돌 해결
- 버그 및 문제점 발생시 추적이 용이
- 기타 산출물의 이력관리도 용이

#### (3) 변경 관리/버전 관리/형상 관리

##### ① 변경 관리

- 소스의 변경 상황을 관리
- 문서의 변경 이력과 복원 등의 기능이 제공

##### ② 버전 관리

- 변경을 관리하기 위한 효과적인 방법
- 체크인, 체크아웃, 릴리즈, 퍼블리싱의 과정을 버전으로 관리할 수 있다.

##### ③ 형상 관리

- 변경 관리와 버전 관리가 포함되고, 프로젝트 진행상황, 빌드와 릴리즈까지 모두 관리할 수 있는 통합 시스템

**(4) 형상 관리 대상**

- 프로젝트 수행 계획서, 요구사항 관리대장, SW 기능 구조도
- 엔티티 정의서, 데이터 흐름도, 용어집
- 인터페이스, ERD, UI 정의서
- 소스 코드, 단위 테스트 관리 대장
- 테스트 계획서/시나리오
- 사용자/운영자 매뉴얼, 최종 산출물

**(5) 형상 관리 절차**

- 형상 식별
  - 형상 관리의 시작으로 시스템을 구성하는 요소들 중 형상 관리의 대상들을 구분하고 관리 목록의 번호를 정의하여 부여한다.
  - 형상 항목은 단순히 소스파일 뿐만 아니라 산출물, 개발이력, 개발과정에서 작성되는 문서까지 모두 포함
- 형상 통제
  - 소프트웨어 형상 변경 요청을 검토하고 승인하여 현재의 베이스라인에 반영될 수 있도록 통제
  - 형상통제가 이루어지기 위해서는 형상 통제 위원회(Configuration Control Board, CCB)의 승인을 통한 변경 통제가 이루어져야 한다.
- 형상 감사
  - 형상 항목의 변경이 계획에 따라 제대로 이뤄졌는지를 검토하고 승인
- 형상 기록/보고
  - 프로젝트 팀, 회사, 클라이언트 등에게 소프트웨어 개발 상태에 대한 보고서를 제공
  - 베이스라인 산출물에 대한 변경과 처리 과정에서의 변경을 모두 기록

**6. 버전 관리 도구****(1) 소프트웨어 버전 관리 도구 개념**

- 동일한 소스 코드에 대한 여러 버전을 관리하는 것
- 개발 중인 소스 코드나, 설계 문서 등의 디지털 문서를 관리하는데 사용
- 문서의 변경 사항들에 숫자나 문자로 이뤄진 버전을 부여해서 구분
- 버전을 통해서 변경된 시간과 변경된 내용, 작업자를 추적할 수 있다.

**(2) 소프트웨어 버전 관리 도구 유형****① 공유 폴더 방식 (RCS, SCCS)**

- 매일 개발 완료 파일은 약속된 위치의 공유 폴더에 복사
- 담당자 한 명이 매일 공유 폴더의 파일을 자기 PC로 복사하고 컴파일하여 에러 확인과 정상 동작 여부 확인
- 정상 동작일 경우 다음날 각 개발자들이 동작 여부 확인

## ② 클라이언트/서버 방식 (CVS, SVN)

- 중앙에 버전 관리 시스템이 항상 동작
- 개발자들의 현재 작업 내용과 이전 작업내용 추적 용이
- 서로 다른 개발자가 같은 파일을 작업했을 때 경고 출력
- Trac이나 CVS view와 같은 GUI 툴을 이용 모니터링 가능

## ③ 분산 저장소 방식 (Git, Betkeeper)

- 로컬 저장소와 원격저장소 구조
- 중앙의 저장소에서 로컬에 복사(clone)한 순간 개발자 자신만의 로컬저장소에 생성
- 개발 완료한 파일 수정 이후 로컬 저장소에 커밋한 이후 다시 원격 저장소에 반영(Push)하는 방식

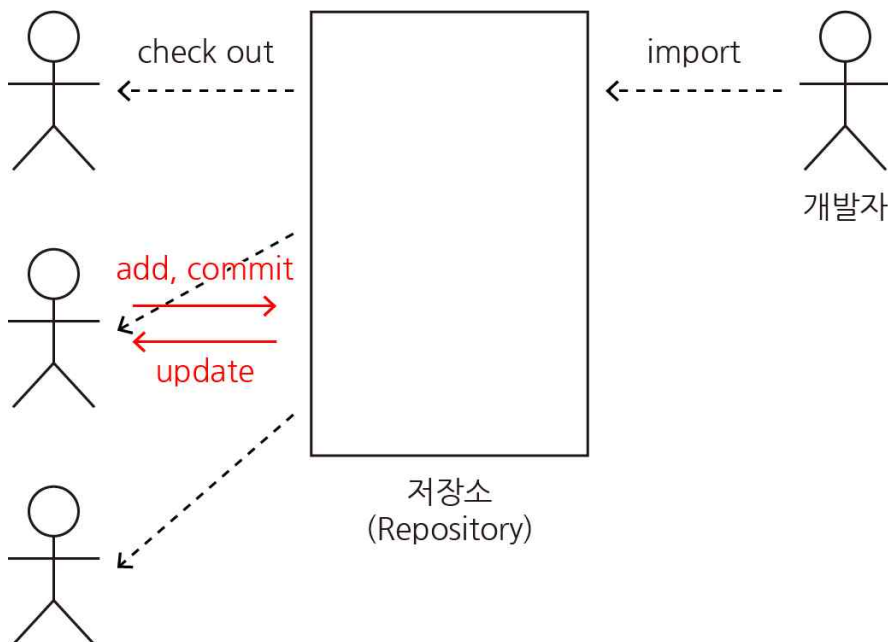
## (3) 버전 관리 도구별 특징

- CVS
  - 오랜 기간 사용된 형상 관리 도구로, 다양한 운영체제를 지원
  - 중앙에 위치한 Repository에 파일을 저장하고, 인가된 모든 사용자가 파일에 접근할 수 있다.
  - 파일의 히스토리를 보존하기 때문에 과거 이력을 확인할 수 있다.
  - Commit 중 오류가 발생하면 롤백되지 않는다.
  - 다른 개발자가 작업 중인 파일에 덮어쓰기가 방지된다.
  - 상대적으로 속도가 느리다.
  - 등록된 파일이나 디렉토리의 변동이 불편하다.
- SVN
  - CVS의 단점을 보완하기 위해 만들어졌다.
  - 최초 1회에 한해 파일 원본을 저장하고, 그 이후에는 실제 파일이 아닌 원본과 차이점을 저장하는 방식
  - 언제든지 원하는 시점으로 복구가 가능
  - Commit 실패시 Rollback 이 가능
  - Trunk, Branches, Tags 의 폴더로 구성하여 형상 관리를 한다.
- Git
  - 리눅스 토발즈가 리눅스 커널의 개발을 위해 만들었다.
  - 원격 서버 Git Repository에 push 하지 않은 채 여러 branch 생성이 가능하다.
  - 로컬 우선 작업을 통해 성능이 SVN, CVS보다 우수하다.
  - 팀 개발을 위한 분산 환경 코딩에 최적화되어 있다.
  - 원격 Repository에 장애가 있어도 버전 관리가 가능하다.
- Clear Case
  - IBM에서 개발된 유료 버전의 형상 관리 툴
  - 서버가 부족할 때 서버를 하나씩 늘려 확장할 수 있다.
- BitKeeper
  - SVN과 비슷한 중앙 통제 방식으로 대규모 프로젝트에서 빠른 속도를 내도록 개발된 버전관리 도구
- RCS(Revision Control System)
  - 소스 파일의 수정을 한 사람만으로 제한하여 다수의 사람이 파일의 수정을 동시에 할 수 없도록 파일을 잠금하는 방식으로 버전 컨트롤을 수행

## (4) 버전 관리 주요 용어

용어	설명
Repository	저장소
Checkout	Repository에서 로컬로 프로젝트를 복사
Commit	로컬의 변경된 내용을 Repository에 저장
Update	Repository에 있는 내용을 로컬에 반영
Add	로컬에서 새로운 파일이 추가되었을 때 Repository에 등록
Trunk	Root 프로젝트
Branch	Root 프로젝트에서 파생된 프로젝트
Merge	Branch에서 진행하던 작업을 Root 프로젝트와 합침
Diff	파일의 비교

## (5) 버전 관리 소프트웨어 사용 방식



- 프로젝트 시작시 프로젝트에 사용될 프레임워크, 기본 문서들을 최초로 import 한다.
- 프로젝트 참여자들은 각자의 계정을 생성하고, 모든 파일들을 인출(Checkout) 한다.
- 참여자는 새로운 파일 생성시 해당 파일을 버전 관리 시스템에 추가(add) 한다.
- 참여자는 기존 파일 수정시 수정된 내용을 저장소에 저장(Commit) 한다.
- 참여자는 로컬에 있는 파일과 다른 버전의 파일이나 신규 파일들을 동기화(Update) 한다.
- 동기화시 두 파일의 내용을 비교(Diff) 할 수 있다.

## (6) 버전 관리 도구 사용 유의점

- 형상 관리 지침에 의거 버전에 대한 정보를 언제든지 접근할 수 있어야 한다.
- 제품 소프트웨어 개발자, 배포자 이외에 불필요한 사용자가 소스를 수정할 수 없도록 해야 한다.
- 동일한 프로젝트에 대해서 여러 개발자가 동시에 개발할 수 있어야 한다.
- 에러 발생 시 최대한 빠른 시간 내에 복구한다.

## 7. 빌드 도구

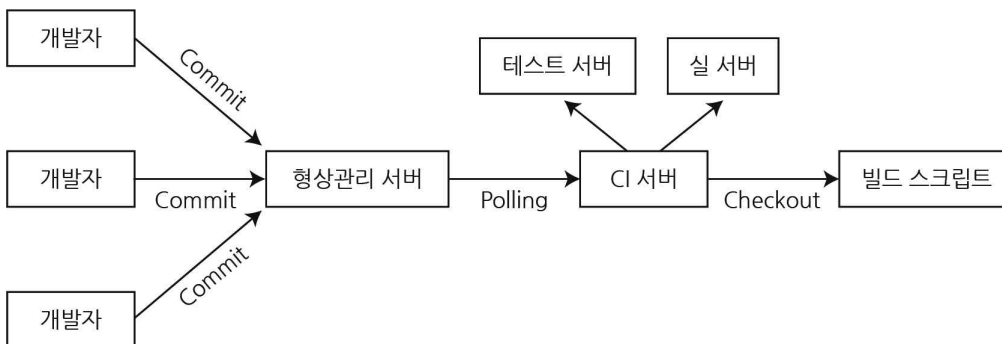
### (1) 빌드의 개념

- 소스코드 파일들을 컴퓨터에서 실행할 수 있는 소프트웨어로 변환하는 일련의 과정
- 빌드의 단계 중 컴파일이 포함이 되어 있는데 컴파일은 빌드의 부분집합
- 빌드 과정을 도와주는 도구를 빌드 툴 이라고 한다.
- 빌드 자동화 도구는 지속적인 통합(Continuous Integration)을 수행할 수 있도록 도와준다.

### (2) 빌드 자동화 도구 특징

- 빌드, 테스트, 배포를 자동으로 수행하는 도구
- 소스 코드를 컴파일, 테스트, 정적분석 등을 실시하여 실행 가능한 애플리케이션으로 자동으로 생성
- 계속해서 늘어나는 라이브러리 자동 추가 및 관리
- 프로젝트를 진행하며 시간이 지남에 따라 라이브러리의 버전을 자동으로 동기화 한다.

### (3) 빌드 자동화 프로세스



- 빌드
  - 개발자가 저장장소에 코드를 커밋한다.
  - 코드 변경 사항은 운영 환경과 일치하는 환경에 통합한다.
- 테스트
  - JenKins나 Ansible과 같은 배포 자동화 툴에서 새로운 코드를 인식하고 일련의 테스트를 수행한다.
  - 테스트를 통과한 빌드는 운영 환경으로 릴리즈 할 수 있다.
- 배포
  - 소프트웨어를 운영 환경에 배포하여 사용자에게 제공한다.

#### (4) 빌드 자동화 도구 종류

- Make
  - 유닉스 계열 운영체제에서 주로 사용되는 프로그램 빌드 도구이다.
  - 파일 간 종속관계를 파악하여 Makefile(기술파일)에 적힌 내용을 컴파일러가 순차적으로 실행하게 한다.
- Ant
  - Java 기반의 빌드 도구로 다른 빌드 도구 보다 역사가 오래 되었다.
  - 개발자가 원하는 형태로 개발을 할 수 있다는 유연성에 장점이 있다.
  - XML 기반의 빌드 스크립트로 개발한다.
  - 스크립트의 재사용이 어렵다.
  - Remote Repository를 가져올 수 없다.
- Maven
  - 프로젝트에 필요한 모든 Dependency를 리스트 형태로 Maven에게 알려 관리할 수 있도록 돕는 방식이다.
  - 필요한 라이브러리를 특정 파일(pom.xml)에 정의해 놓으면 해당 라이브러리와 관련된 라이브러리까지 네트워크를 통해 자동으로 다운받는다.
  - 정해진 라이프사이클에 의하여 작업 수행하며, 전반적인 프로젝트 관리 기능까지 포함한다.
- Jenkins
  - Java 기반의 오픈소스로, 소프트웨어 개발 시 지속적 통합(continuous integration) 서비스를 제공하는 툴
  - 서블릿 컨테이너에서 실행되는 서버 기반 도구
  - SVN, Git 등 대부분의 형상 관리 도구와 연동이 가능
  - 여러 대의 컴퓨터를 이용한 분산 빌드나 테스트가 가능
- Gradle
  - Groovy를 기반으로 한 오픈 소스 형태의 자동화 도구로 안드로이드 앱 개발 환경에서 사용
  - 안드로이드 뿐만 아니라 플러그인을 설정하면 Java, C/C++, Python 등의 언어도 빌드가 가능
  - Groovy를 사용해서 만든 DSL(Domain Specific Language)을 스크립트 언어로 사용
  - Gradle은 실행할 처리 명령들을 모아 태스크로 만든 후 태스크 단위 실행

## Section 2. 개발 프레임워크

### 1. 프레임워크의 개념

- 소프트웨어 개발에 공통적으로 사용되는 구성 요소와 아키텍처를 일반화하여 손쉽게 구현할 수 있도록 여러 가지 기능들을 제공해주는 반제품 형태의 소프트웨어
- 소프트웨어 개발에 바탕이 되는 템플릿과 같은 역할을 하는 클래스들과 인터페이스의 집합
- 소프트웨어 개발시 공통적인 부분을 제공

### 2. 프레임워크의 특징

- 모듈화 (modularity)
  - 프레임워크는 구현을 인터페이스 뒤에 감추는 캡슐화를 통해서 모듈화를 강화
  - 설계와 구현의 변경에 따르는 영향을 최소화시킴으로써 쉽게 소프트웨어의 품질을 향상시킴
- 재사용성 (reusability)
  - 프레임워크가 제공하는 인터페이스는 여러 어플리케이션에서 반복적으로 사용할 수 있는 일반적인 컴포넌트를 정의할 수 있게 함으로써 재사용성을 높여준다.
  - 소프트웨어의 품질, 성능, 신뢰성, 상호 운용성을 향상시키고, 프로그래머의 생산성을 높여준다.
- 확장성(extensibility)
  - 다형성(polymorphism)을 통해 애플리케이션의 프레임워크의 인터페이스를 확장할 수 있게 한다.
- 제어의 역흐름(inversion of control)
  - 프레임워크가 외부의 이벤트에 대해 애플리케이션이 어떠한 메소드들을 수행해야 하는지 결정

### 3. 프레임워크의 구분

- Java 프레임워크
  - 전자정부 표준 프레임워크
  - 스트럿츠
  - 스프링
- ORM 프레임워크
  - 아이바티스(iBatis)
  - 마이바티스(myBatis)
  - 하이버네이트(Hibernate)
- 자바스크립트 프레임워크
  - 앵귤러제이에스(AngularJS)
  - ReactJS
  - ExtJS
- 프론트엔드 프레임워크
  - Bootstrap
  - Foundation
  - MDL

#### 4. 라이브러리(Library)

- 컴퓨터 프로그램에서 빈번하게 사용되는 사전 컴파일된 루틴 또는 리소스(클래스, 템플릿, 설정 데이터 등)를 모아둔 것
- 재사용이 필요한 기능으로 반복적인 코드 작성을 없애기 위해 언제든지 필요한 곳에서 호출하여 사용할 수 있도록 Class나 Function
- 라이브러리는 어플리케이션의 특정 기능, 프레임워크는 어플리케이션의 구조

#### 5. API(Application Programming Interface)

- 일종의 소프트웨어 인터페이스이며 다른 종류의 소프트웨어에 서비스를 제공한다.
- API는 응용 프로그램에서 사용할 수 있도록, 운영 체제나 프로그래밍 언어가 제공하는 기능을 제어할 수 있게 만든 인터페이스
- API 특징
  - 개발 비용 감축
  - 반복 작업 줄이기
  - 쉬운 유지 관리
  - 새로운 수익 채널의 확대
  - 비즈니스 파이의 확장



## Section 3. 모듈 구현

### 1. 단위 모듈 구현

#### (1) 단위 모듈 구현의 개념

- 소프트웨어를 기능 단위로 분해하여 구현하는 기법
- 서브시스템, 서브루틴, 작업 단위 등으로 나누어 각 모듈이 독립적으로 활용될 수 있게 구현
- 모듈의 크기는 작고, 하나의 일만을 수행
- 모듈의 크기가 작으면 읽기 쉽고, 구현하기 쉬우며, 테스트 부담이 적어진다.

#### (2) 단위 모듈 구현 시 장점

- 프로그램의 효율적인 관리 및 성능이 향상
- 전체적인 소프트웨어 복잡성 감소 및 이해성 증대
- 테스트, 모듈 통합, 변경 용이성 쉬움
- 기능의 분리가 가능하고 인터페이스가 단순해짐
- 오류의 파급효과 최소화
- 모듈의 재사용으로 개발과 유지보수가 용이

#### (3) 효과적인 모듈화

- 결합도를 줄이고 응집도를 높여 모듈의 독립성을 높임
- FAN-OUT 최소화, FAN-IN 증가
- 모듈 인터페이스를 평가하여 복잡성과 중복성을 줄이고 일관성을 높인다.
- 기능 예측이 가능한 모듈을 정의
- 하나의 입력과 하나의 출력을 유지

#### (4) 단위 모듈 설계의 원리

- 단계적 분해
  - 처음엔 간단히 작성하고, 점점 세밀하게 작성
- 추상화
  - 복잡한 문제를 일반화하여, 쉽게 이해할 수 있도록 한다.
- 독립성
  - 모듈은 응집도는 높이고 결합도는 낮춰 독립성을 가져야 한다.
- 정보은닉
  - 모듈 내부의 데이터를 외부에 은폐
- 분할과 정복
  - 큰 문제를 작게 나누어 하나씩 해결

### (5) 단위 모듈 작성 원칙

- 정확성 (Correctness)
  - 해당 기능이 실제 시스템 구현 시 필요한지 여부를 알 수 있도록 정확하게 작성
- 명확성 (Clarity)
  - 해당 기능에 대해 일관되게 이해되고 한 가지로 해석될 수 있도록 작성
- 완전성 (Completeness)
  - 시스템이 구현될 때 필요하고 요구되는 모든 것을 기술
- 일관성 (Consistency)
  - 공통 기능들 간에 상호 충돌이 없도록 작성
- 추적성 (Traceability)
  - 공통 기능에 대한 요구사항 출처와 관련 시스템 등의 유기적 관계에 대한 식별이 가능하도록 작성

## 2. 결합도

### (1) 결합도(Coupling)의 개념

- 어떤 모듈이 다른 모듈에 의존하는 정도
- 두 모듈 사이의 연관 관계
- 결합도가 낮을수록 잘 설계된 모듈이다.

### (2) 결합도 유형

구분	설명
자료 결합도 (Data Coupling)	모듈 간의 인터페이스로 값이 전달되는 경우
스탬프 결합도 (Stamp Coupling)	모듈 간의 인터페이스로 배열이나 오브젝트, 스트럭처 등이 전달되는 경우
제어 결합도 (Control Coupling)	단순 처리할 대상인 값만 전달되는 게 아니라 어떻게 처리를 해야 한다는 제어 요소가 전달되는 경우
외부 결합도 (External Coupling)	어떤 모듈에서 선언한 데이터(변수)를 외부의 다른 모듈에서 참조하는 경우
공통 결합도 (Common Coupling)	파라미터가 아닌 모듈 밖에 선언되어 있는 전역 변수를 참조하고 전역 변수를 갱신하는 식으로 상호 작용하는 경우
내용 결합도 (Content Coupling)	다른 모듈 내부에 있는 변수나 기능을 다른 모듈에서 사용하는 경우

### 3. 응집도

#### (1) 응집도(Cohesion)의 개념

- 모듈의 독립성을 나타내는 개념으로, 모듈 내부 구성요소 간 연관 정도
- 정보 은닉 개념의 확장개념으로, 하나의 모듈은 하나의 기능을 수행하는 것을 의미
- 응집도는 높을수록 좋고 결합도는 낮을수록 이상적

#### (2) 응집도 유형

구분	설명
기능적 응집도 (Functional Cohesion)	모듈 내부의 모든 기능이 단일한 목적을 위해 수행되는 경우
순차적 응집도 (Sequential Cohesion)	모듈 내에서 한 활동으로부터 나온 출력값을 다른 활동이 사용할 경우
통신적 응집도 (Communication Cohesion)	동일한 입력과 출력을 사용하여 다른 기능을 수행하는 활동들이 모여 있을 경우
절차적 응집도 (Procedural Cohesion)	모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성 요소들이 그 기능을 순차적으로 수행할 경우
시간적 응집도 (Temporal Cohesion)	연관된 기능이라기보다는 특정 시간에 처리되어야 하는 활동들을 한 모듈에서 처리할 경우
논리적 응집도 (Logical Cohesion)	유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들이 한 모듈에서 처리되는 경우
우연적 응집도 (Coincidental Cohesion)	모듈 내부의 각 구성 요소들이 연관이 없을 경우

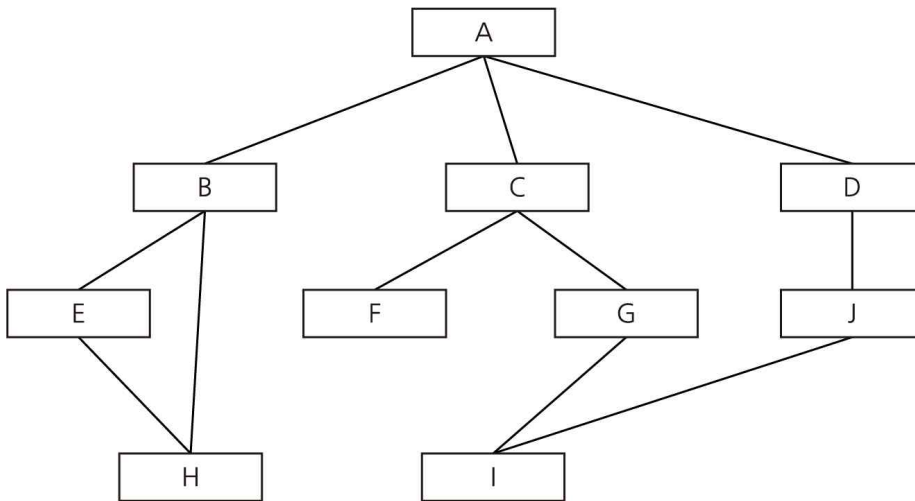
### 4. 팬인(Fan-in), 팬아웃(Fan-out)

#### (1) 팬인(Fan-in), 팬아웃(Fan-out)의 개념

- 소프트웨어의 구성요소인 모듈을 계층적으로 분석하기 위해 활용
- 팬인과 팬아웃 분석을 통해 시스템의 복잡도를 측정
- 시스템 복잡도를 최적화하기 위해서는 팬인은 높게, 팬 아웃은 낮게 설계

구분	설명
팬인 (Fan-in)	- 얼마나 많은 모듈들이 현재 모듈을 호출하는지를 나타낸다. - 해당 모듈로 들어오는 상위 모듈 수
팬아웃 (Fan-out)	- 해당 모듈에서 호출하는 하위 모듈 수

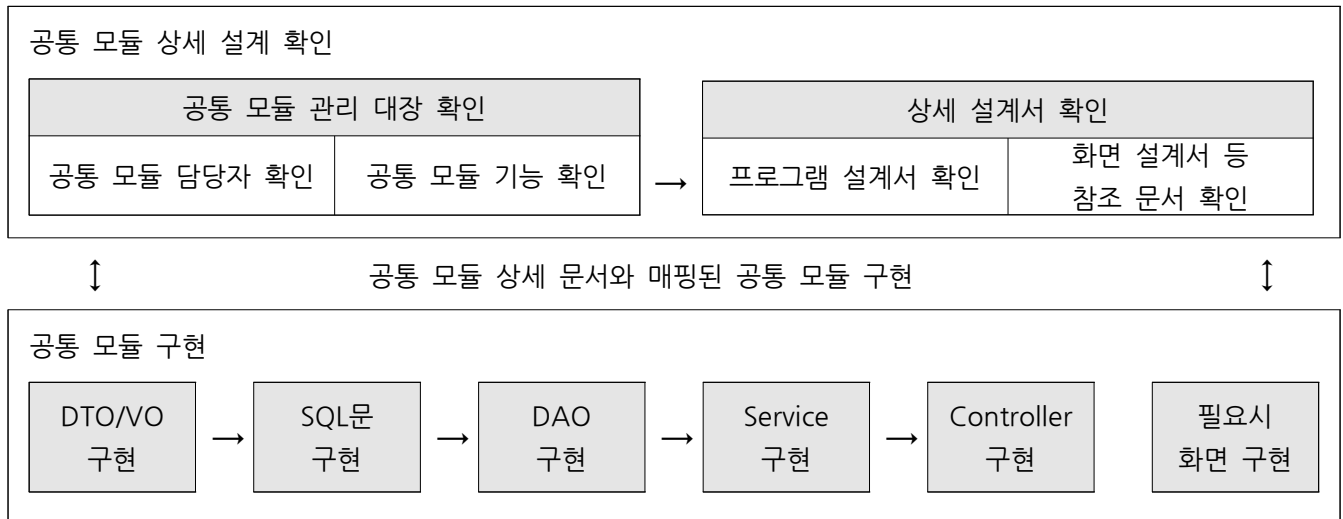
## (2) 팬인/팬아웃 계산법



모듈	팬인(Fan-in)	팬아웃(Fan-out)
A	0	3
B	1	2
C	1	2
D	1	1
E	1	1
F	1	0
G	1	1
H	2	0
I	2	0
J	1	1

## 5. 공통 모듈 구현

### (1) 공통 모듈 구현 순서



### (2) 공통 모듈 구현요소

구현 요소	설명
DTO (Data Transfer Object)	<ul style="list-style-type: none"> <li>- 프로세스 사이에서 데이터를 전송하는 객체</li> <li>- Getter, Setter 메서드만 포함한다.</li> </ul>
VO (Value Object)	<ul style="list-style-type: none"> <li>- 도메인에서 속성들을 묶어서 특정 값을 나타내는 객체</li> <li>- DTO와 동일한 개념이나 차이점은 Read-Only 속성 객체이다</li> </ul>
DAO (Data Access Object)	<ul style="list-style-type: none"> <li>- 실질적으로 DB에 접근하는 객체</li> <li>- DataBase에 접근하기 위한 로직 &amp; 비즈니스 로직을 분리하기 위해 사용</li> </ul>
Service	<ul style="list-style-type: none"> <li>- DAO 클래스를 호출하는 객체</li> </ul>
Controller	<ul style="list-style-type: none"> <li>- 비즈니스 로직을 수행하는 객체</li> </ul>

### (3) 공통 모듈 구현

#### ① DTO/VO 구현

```
package com.njob.vo;

public class CodeVo{
    private String code;
    private String name;
    public String getCode(){
        return this.code;
    }
    public void setCode(String code){
        this.code = code;
    }
    public String getName(){
        return this.name;
    }
    public void setName(String name){
        this.name = name;
    }
}
```

#### ② SQL 문 구현

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
"http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.njob.sql">
    <select id="getCodeList" resultType="com.njob.vo.CodeVo">
        SELECT
            code, name
        FROM tb_code
    </select>

    <update id="updateCodeInfo" parameterType="com.njob.vo.CodeVo">
        UPDATE tb_code
        SET
            name = #{name, jdbcType=CHAR}
        WHERE code = #{code, jdbcType=INTEGER}
    </update>
</mapper>
```

### ③ DAO 구현

```
package com.njob.dao;
import com.njob.vo.CodeVO;

@Repository("codeDao")
public class CodeDao{
    @Autowired
    private SqlSession sqlSession;

    public List<CodeVO> getCodeList() throws Exception {
        return sqlSession.selectList("com.njob.sql.getCodeList");
    }

    public int updateCodeIsUse(CodeVo code) {
        return sqlSession.update("com.njob.sql.updateCodeInfo", code);
    }
}
```

### ④ Service 구현

```
package com.njob.service;
import com.njob.vo.CodeVO;
import com.njob.dao.CodeDao;

@Service("codeService")
public class CodeService{
    @Resource(name="codeDao")
    private CodeDao codeDao;

    public List<CodeVO> getCodeList() throws Exception {
        return codeDao.getCodeList();
    }

    public int updateCodeIsUse(CodeDao codeDao) throws Exception {
        return codeDao.updateCodeIsUse(codeDao);
    }
}
```

## ⑤ Controller 구현

```
package com.njob.controller;
import com.njob.vo.CodeVO;
import com.njob.dao.CodeDao;
import com.njob.service.CodeService;

@Controller
public class CodeController{
    @Autowired
    private CodeService codeService

    @RequestMapping(value = "/code/list")
    public ResponseEntity<List<CodeVO>> list(){
        try{
            List<CodeVO> list = codeService.getCodeList();

            if( list != null ){
                return new ResponseEntity<List<CodeVO>>(list, HttpStatus.OK);
            }
            else{
                return new ResponseEntity<List<CodeVO>>(list, HttpStatus.NO_CONTENT);
            }
        }
        catch(Exception e){
            logger.error(e.getMessage());
        }
    }
}
```



#### (4) Annotation

##### ① Annotation 개념

- 사전적으로는 "주석"이라는 의미를 가지고 있다.
- 자바코드에 주석처럼 달아 특수한 의미를 부여한다.
- 컴파일 또는 런타임에 해석된다.

##### ② Annotation 종류

종류	설명
@Controller	- 스프링 MVC의 컨트롤러 객체임을 명시
@RequestMapping	- 특정 URI 에 매칭되는 클래스나 메소드임을 명시
@RequestParam	- 요청(request)에서 특정한 파라미터 값을 찾아낼 때 사용하는 어노테이션
@RequestHeader	- 요청(request)에서 특정 HTTP 헤더 정보를 추출할 때 사용
@PathVariable	- 현재 URI에서 원하는 정보를 추출할 때 사용
@CookieValue	- 현재 사용자의 쿠키값을 추출할 때 사용
@ModelAttribute	- 자동으로 해당 객체를 뷰까지 전달하도록 한다.
@ResponseBody	- 리턴 타입이 HTTP의 응답 메시지로 전송
@RequestBody	- 요청 문자열이 그대로 파라미터로 전달
@Repository	- DAO 객체
@Service	- 서비스 객체
@Scheduled	- 스프링에서 지원하는 배치 어노테이션

## Section 4. 서버 프로그램 구현

### 1. 서버 프로그램 구현

#### (1) 업무 프로세스 확인

##### ① 업무 프로세스의 개념

- 개인이나 조직이 한 개 이상의 자원 입력을 통해 가치 있는 산출물을 제공하는 활동

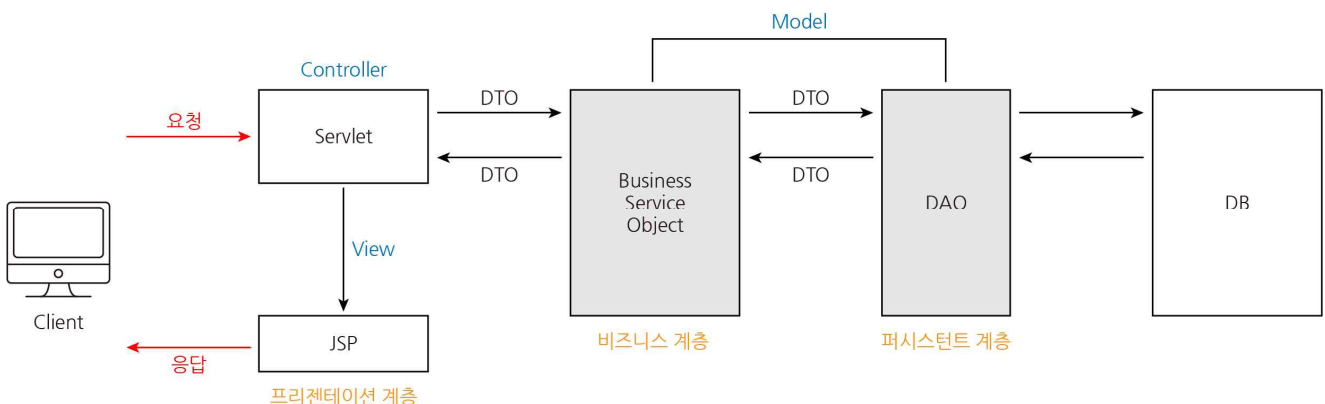


##### ② 업무 프로세스 구성 요소

구성 요소	설명
프로세스 책임자	<ul style="list-style-type: none"> <li>프로세스의 성과와 운영을 책임지는 구성원</li> <li>프로세스를 설계하고 지속적으로 유지한다.</li> </ul>
프로세스 맵	<ul style="list-style-type: none"> <li>상위 프로세스와 하위 프로세스의 체계를 도식화하여 업무의 청사진 표현</li> <li>구조적 분석 기법 : 자료 흐름도</li> <li>객체지향 분석 기법 : ERD</li> </ul>
프로세스 Task 정의서	<ul style="list-style-type: none"> <li>결과물을 산출하기 위해 Task 들의 운영방법을 문서화한다.</li> </ul>
프로세스 성과 지표	<ul style="list-style-type: none"> <li>프로세스의 과정과 결과를 고객 입장에서 정량적으로 표현한 성과 지표</li> </ul>
프로세스 조직	<ul style="list-style-type: none"> <li>프로세스를 성공적으로 수행하기 위해 개인들의 업무를 유기적으로 수행하는 구성원</li> </ul>
경영자의 리더십	<ul style="list-style-type: none"> <li>경영자는 프로세스의 중요성을 인식한다.</li> <li>기업의 경영 방침을 확고하게 한다.</li> </ul>

#### (2) 서버 프로그램 구현

- 업무 프로세스(BusinessLogic)를 기반으로 개발언어, 도구를 이용하여 서버에서 서비스를 제공하는데 필요한 기능을 구현하는 활동
- 서버 프로그램 구현 절차



- 구현 요소

구현 요소	설명
DTO (Data Transfer Object)	- 프로세스 사이에서 데이터를 전송하는 객체 - Getter, Setter 메서드만 포함한다.
VO (Value Object)	- 도메인에서 속성들을 묶어서 특정 값을 나타내는 객체 - DTO와 동일한 개념이나 차이점은 Read-Only 속성 객체이다.
DAO (Data Access Object)	- 실질적으로 DB에 접근하는 객체 - DataBase에 접근하기 위한 로직 & 비즈니스 로직을 분리하기 위해 사용
Service	- DAO 클래스를 호출하는 객체
Controller	- 비즈니스 로직을 수행하는 객체

### (3) MVC 모델의 계층

#### ① 프리젠테이션 계층(Presentation Layer)

- 사용자 인터페이스
- 사용자가 선택할 수 있는 기능 및 부가정보를 전달할 양식을 표현한다.
- JSP 와 JSTL 태그 라이브러리를 결합하는 방식(JAVA의 경우)

#### ② 제어 계층(Control Layer)

- 프리젠테이션 계층과 비즈니스 로직 계층을 분리하기 위한 컨트롤러를 제공
- 어떤 요청이 들어왔을 때 어떤 로직이 처리해야 하는지를 결정한다.
- 사용자 요청을 검증하고 로직에 요청을 전달하는 일과 로직에서 전달된 응답을 적절한 뷰에 연결짓는 역할

#### ③ 비즈니스 로직 계층(Business Logic Layer)

- 핵심 업무를 어떻게 처리하는지에 대한 방법을 구현 하는 계층
- 핵심 업무 로직의 구현과 그에 관련된 데이터의 적합성 검증, 트랜잭션 처리 등을 담당한다.

#### ④ 퍼시스턴스 계층(Persistence Layer)

- 데이터 처리를 담당하는 계층
- 데이터의 생성/수정/삭제/선택(검색)과 같은 CRUD 연산을 수행한다.

#### ⑤ 도메인 모델 계층(Domain Model Layer)

- 각 계층 사이에 전달되는 실질적인 비즈니스 객체
- 데이터 전송 객체(DTO) 형태로 개발자가 직접 제작해서 데이터를 넘기게 된다.

## 2. DBMS 접속기술

### (1) DBMS 접속기술의 개념

- 프로그램에서 DB에 접근하여 DML을 사용 가능하게 하는 기술
- 프로그램이 DB를 사용할 수 있도록 연결해주는 인터페이스

### (2) DBMS 접속기술 종류

#### ① 소켓통신

- 응용프로그램과 DBMS가 주고 받는 통신
- DBMS 사에서 프로토콜을 공개하지 않기 때문에 개발은 어렵다.

#### ② Vender API

- DBMS 사에서 공개한 API를 이용해 DBMS와 통신
- DBMS 사마다 API 사용법이 상이하다.

#### ③ JDBC(Java DataBase Connectivity)

- Java 에서 DB에 접속하고 SQL문을 수행할 때 사용되는 표준 API
- JDBC는 Java SE(Standard Edition)에 포함되어 있다.
- java.sql 패키지와 javax.sql 패키지에 포함되어 있다.
- 접속하려는 DBMS에 맞는 드라이버가 필요함

#### ④ ODBC(Open DataBase Connectivity)

- 데이터베이스에 접근하기 위한 표준 규격
- 개발언어에 관계없이 사용할 수 있다.
- 모든 DBMS에 접근하는 방법을 통일시킴
- 1990년대 초 MS 사에서 개발됨

## 3. ORM(Object-Relational Mapping) 프레임워크

### (1) ORM 프레임워크의 개념

- 객체와 관계형 데이터베이스의 데이터를 자동으로 매핑(연결)해주는 것
- 객체지향 프로그램에서 클래스를 생성하고, 관계형 데이터베이스의 테이블의 내용을 매핑
- 객체지향 프로그램을 통해 데이터베이스의 데이터를 다룬다.

### (2) ORM 장/단점

- 장점
  - 비즈니스 로직에 더 집중할 수 있다.
  - 재사용 및 유지보수의 편리성이 증가한다.
  - DBMS에 대한 종속성이 줄어든다.
- 단점
  - 완벽한 ORM 으로만 서비스를 구현하기 어렵다.
  - 프로시저가 많은 시스템에선 ORM의 객체지향 장점을 활용하기 어렵다.

### (3) 매핑 기술 비교

- SQL Mapper
  - SQL을 명시하여 단순히 필드를 매핑 시키는 것이 목적
  - SQL 문장으로 직접 데이터베이스 데이터를 다룬다.
  - SQL 의존적인 방법
  - 종류 : iBatis, Mybatis, jdbc Templates 등
- OR Mapping (=ORM)
  - 객체를 통해 간접적으로 데이터베이스를 다룬다.
  - 객체와 관계형 데이터베이스의 데이터를 자동으로 매핑
  - ORM을 이용하면 SQL Query 가 아닌 직관적인 코드로 데이터를 조작할 수 있다.
  - 종류 : JPA(Java Persistent API),Hibernate

## 4. 시큐어 코딩 (Secure Coding)

### (1) OWASP(The Open Web Application Security Project)

- 오픈소스 웹 애플리케이션 보안 프로젝트
- 주로 웹에 관한 정보 노출, 악성 파일 및 스크립트 보안 취약점 등을 연구하며 10대 취약점을 발표했다.
- OWASP Top 10
  - 웹 애플리케이션 취약점 중 빈도가 많이 발생하고, 보안상 영향을 줄 수 있는 10가지를 선정하여 발표

### (2) 시큐어 코딩 가이드

#### ① 시큐어 코딩 가이드의 개념

- 해킹 등 사이버 공격의 원인인 보안취약점을 제거해 안전한 소프트웨어를 개발하는 SW 개발 기법
- 개발자의 실수나 논리적 오류로 인해 발생할 수 있는 문제점을 사전에 차단하여 대응하고자 하는 것

#### ② 시큐어 코딩 가이드 항목

- 입력 데이터 검증 및 표현
  - 프로그램 입력값에 대한 검증 누락 또는 부적절한 검증, 데이터형식을 잘못 지정하여 발생하는 보안 약점
  - 보안 약점 종류

종류	설명
SQL Injection	- SQL문을 삽입하여 DB로부터 정보를 열람 및 조작할 수 있는 공격
XSS (크로스 사이트 스크립트)	- 악의적인 스크립트를 포함해 사용자 측에서 실행되게 유도하는 공격
자원 삽입	- 외부 입력값이 시스템 자원 접근 경로 또는 자원 제어에 사용되는 공격
위험한 형식 파일 업로드	- 서버측에서 실행될 수 있는 스크립트파일을 업로드 하여 공격
명령 삽입	- 운영체제 명령어 삽입 - XPath 삽입 - XQuery 삽입 - LDAP 삽입
메모리 버퍼 오버프로	- 입력받는 값이 버퍼를 가득 채우다 못해 넘쳐흘러 버퍼 이후의 공간을 침범하는 공격

- 보안 기능
  - 보안 기능을 부적절하게 구현하는 경우 발생할 수 있는 보안 약점
  - 보안 약점 종류

종류	설명
적절한 인증 없이 중요기능 허용	- 적절한 인증과정이 없이 중요정보(계좌, 개인정보 등)를 열람 할 때 발생하는 보안 약점
부적절한 인가	- 적절한 접근 제어 없이 외부 입력값을 포함한 문자열로 중요 자원에 접근할 수 있는 보안약점
취약한 암호화 알고리즘 사용	- DES, MD5 등 안전하지 않은 알고리즘 사용
하드코딩된 패스워드	- 소스 코드 내에 비밀번호가 하드코딩되어 있어 소스코드 유출시 노출되는 보안 약점
패스워드 평문 저장	- 계정 정보 탈취 시 패스워드 노출
취약한 패스워드 허용	- 비밀번호 조합규칙(영문, 숫자, 특수문자 등)이 미흡하거나 길이가 충분하지 않아 노출될 수 있는 보안약점

- 시간 및 상태
  - 동시 수행을 지원하는 병렬 시스템이나 하나 이상의 프로세스가 동작하는 환경에서 시간 및 상태를 부적절하게 관리하여 발생할 수 있는 보안 약점
  - 보안 약점 종류

종류	설명
경쟁 조건	- 동일 자원에 대한 검사 시점과 사용 시점이 상이하여 동기화 오류, 교착 상태를 유발
종료되지 않는 반복문 또는 재귀 함수	- 종료 조건이 없어 무한 루프에 빠져 자원 고갈을 유발

- 에러 처리
  - 에러를 처리하지 않거나 불충분하게 처리하여 에러 정보에 중요 정보가 포함될 때 발생할 수 있는 보안 약점
  - 보안 약점 종류

종류	설명
오류 메시지 정보 노출	- 응용 프로그램의 민감 정보가 오류 메시지를 통해 노출됨 - 오류 메시지는 사용자가 필요한 최소한의 정보만을 노출해야 한다.
오류 상황 대응 부재	- 예외처리 미구현
부적절한 예외 처리	- 프로그램 수행 중 발생한 예외 조건을 적절히 검사하지 않음

- 코드 오류
  - 개발자가 범할 수 있는 코딩 오류로 인해 유발되는 보안 약점
  - 보안 약점 종류

종류	설명
널 포인터 역참조	- 널 값을 고려하지 않은 코드에서 발생
부적절한 자원 해제	- 자원을 할당받아 사용한 뒤 반환하지 않는 코드에서 발생
해제된 자원 사용	- 해제한 메모리를 참조하는 코드에서 발생
초기화되지 않은 변수 사용	- 지역변수를 초기화하지 않고 사용하는 코드에서 발생

- 캡슐화
  - 중요한 데이터 또는 기능을 불충분하게 캡슐화하거나 잘못 사용해 발생하는 보안 약점
  - 보안 약점 종류

종류	설명
잘못된 세션에 의한 정보 노출	- 멀티 스레드 환경에서 서로 다른 세션 간 데이터가 공유될 수 있음
제거되지 않은 디버그 코드	- 배포 단계에 디버그 코드가 남아 있는 경우 민감 정보가 노출될 수 있음
시스템 정보 노출	- 시스템 내부 데이터가 노출될 수 있음
잘못된 접근 지정자	- private, public 잘못된 접근지정자 사용으로 민감 정보 노출될 수 있음

- API 오용
  - 의도된 사용에 반하는 방법으로 API를 사용하거나 보안에 취약한 API를 사용하여 발생할 수 있는 보안 약점
  - 보안 약점 종류

종류	설명
DNS에 의존한 보안 결정	- 공격자가 DNS 정보를 변조하여 보안 결정을 우회 가능함
취약한 API 사용	- 금지되거나 안전하지 않은 함수를 사용함

## Section 5. 배치 프로그램 구현

### 1. 배치 프로그램

#### (1) 배치의 개념

- 데이터를 일괄적으로 모아서 처리하는 대량의 작업을 처리
- 컴퓨터 흐름에 따라 순차적으로 자료를 처리하는 방식
- 배치 프로그램이란, 대량의 데이터를 모아 정기적으로 반복 처리하는 프로그램이다.

#### (2) 배치 프로그램의 필수 요소

- 대용량 데이터
  - 대용량의 데이터를 처리할 수 있어야 한다.
- 자동화
  - 심각한 오류 상황 외에는 사용자의 개입없이 동작해야 한다.
- 견고함
  - 비정상적인 동작 중단이 발생하지 않아야 한다.
- 안정성
  - 어떤 문제가 발생했을 때, 해당 문제를 추적하고 복구할 수 있어야 한다.
- 성능
  - 주어진 시간에 작업을 완료해야 하고, 다른 어플리케이션의 동작을 방해하지 않아야 한다.

#### (3) 스케줄 관리 종류

##### ① 크론탭 (crontab)

- UNIX, LINUX 계열에서 사용
- 크론탭 형식

분	시	일	월	요일	명령어
---	---	---	---	----	-----

- 항목의 범위

필드	의미	범위
첫 번째	분	0 ~ 59
두 번째	시	0 ~ 23
세 번째	일	1 ~ 31
네 번째	월	1 ~ 12
다섯 번째	요일	0 ~ 6 (0:일요일, 1:월요일)
여섯 번째	명령어	실행할 명령



- 허용 특수문자

특수문자	설명
*	모든 값 (매시, 매일, 매주)
?	특정 값이 아닌 어떤 값이든 상관 없음
-	범위를 지정할 때 (12-14 : 12시부터 14시)
,	여러 값을 지정할 때 (12, 14 : 12시, 14시)
/	증분값, 즉 초기값과 증가치 설정 (* / 20 : 매 20분 마다)

- 설정 예

형식	설명
* * * * * 명령	매분 실행
30 4 * * 0 명령	매주 일요일 4시 30분 실행
10-30 4 * * * 명령	매일 오전 4시 10분부터 30분까지 매분 실행
0,10,20 * * * * 명령	매일 매시간 0분, 10분, 20분 실행
*/30 * * * * 명령	매 30분 마다 실행
30 0 1 1,6 * 명령	1월과 6월, 1일, 0시 30분에 실행

## ② Spring Batch

- 백엔드의 배치처리 기능을 구현하는데 사용하는 프레임워크
- 대용량 및 고성능 배치 작업을 가능하게 하는 고급 기술 서비스 및 기능을 제공
- 배치가 실패하여 작업 재시작을 하게 된다면 처음부터가 아닌 실패한 지점부터 실행
- Batch Job을 관리하지만 Job을 구동하거나 실행시키는 기능은 지원하지 않는다.
- Batch Job을 실행시키기 위해서는 Quartz, Scheduler, Jenkins등 전용 Scheduler를 사용
- 구성요소 : Job, Job Launcher, Step, Job Repository

## ③ Quartz Job Scheduler

- 표준 자바 프로그램으로 만들어진 작업을 지정된 일정에 따라 실행시키는데 사용하는 Java 패키지
- 특정한 시간에 특정한 작업을 한다든지 또는 주기적으로 실행해야 할 Java 애플리케이션을 사전에 정해 놓으면 일정에 따라 실행
- 주요 인터페이스 : Scheduler, Job, JobDetail, Trigger, JobBuilder, TriggerBuilder
- 형식

초 분 시 일 월 요일 년(생략가능)

#### (4) 배치 스케줄러 클래스 작성

- ① 배치 설계서 확인
- ② DTO/VO 구현
- ③ SQL 문 구현
- ④ DAO 구현
- ⑤ Schedule 구현 (매일 1시에 배치 실행)

```
@Scheduled(cron="0 0 1 * * ? ")
public void scheduleExecute(){
    String result = "SUCC";
    try{
        /* 수행해야 할 업무를 코딩한다 */
    }
    catch(Exception){
        result = "FAIL";
    }

    logger.info("스케줄 실행 결과 : " + result );
}
```

## 06 인터페이스 구현

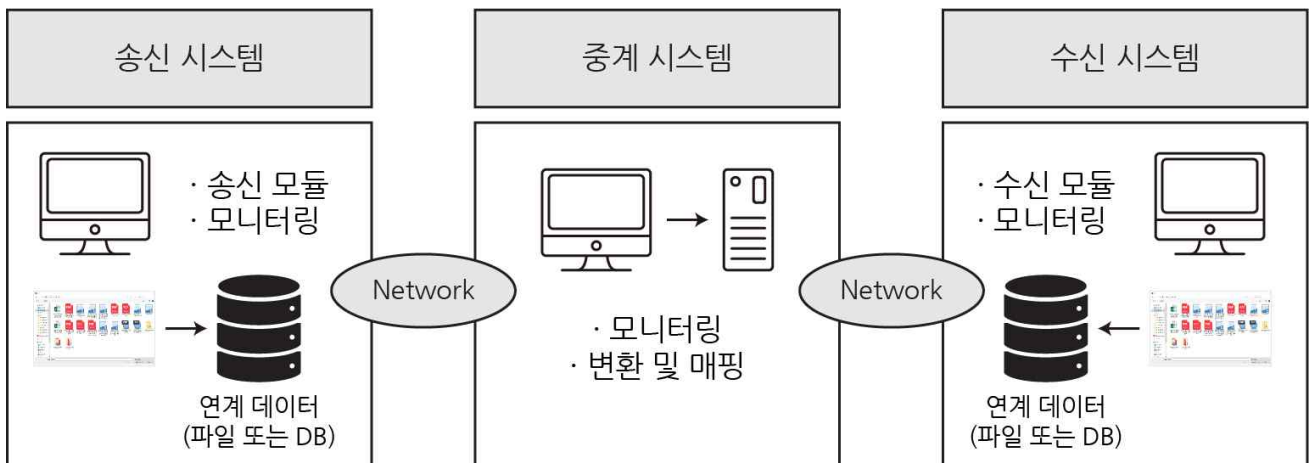
### Section 1. 인터페이스 개요

#### 1. 인터페이스 시스템

##### (1) 인터페이스 시스템의 개념

- 서로 다른 시스템, 장치 사이에서 정보나 신호를 주고 받을 수 있도록 도움을 주는 시스템

##### (2) 인터페이스 시스템 구성



##### ① 송신 시스템

- 연계할 데이터를 생성하여, 연계 테이블 또는 파일 형태로 송신하는 시스템
- 단계별 작업
  - 연계 데이터 생성 및 추출
  - 코드 매핑 및 데이터 변환
  - 인터페이스 테이블 / 파일 생성
  - 로그 기록
  - 데이터 전송

##### ② 수신 시스템

- 수신한 데이터를 데이터 형식에 맞게 저장하고 활용하는 시스템
- 단계별 작업
  - 데이터 수신
  - 코드 매핑 및 데이터 변환
  - 로그 기록
  - 연계 데이터 반영

##### ③ 중계 서버

- 송신 시스템과 수신 시스템 사이에서 데이터를 송수신하고 송수신 현황을 모니터링 하는 시스템
- 연계 데이터의 보안 강화 및 다중 플랫폼 지원 등이 가능

## 2. 연계 시스템 분류와 데이터 식별

### (1) 시스템 분류 체계

- 기업 내부에서 사용하고 있는 시스템 분류 체계를 기반으로 대내외 인터페이스 시스템의 식별자를 정의
- 기업이 수행하는 업무를 대다중으로 파악하여 상위 시스템과 하위 시스템을 구분
- 업무를 기준으로 인터페이스 시스템에 식별 코드를 부여

대분류	ID	중분류	ID
회원	USR	가입	REG
		수정	MOD
		탈퇴	DEL
주문	ORD	주문요청	REQ
		환불요청	REF
		주문취소	DEL

회원 가입 코드 →

USR	REG	001
대분류	중분류	일련번호

### (2) 연계 시스템 식별 정보

구분	설명
대내외 구분 정보	- 기업 내부 시스템인지 외부 기관 시스템인지 구분
기관명	- 대외 기관일 경우 기관명을 기술
시스템 ID	- 시스템 분류 체계에 따라 부여된 식별 번호
한글명	- 시스템 한글명
영문명	- 시스템 영문명(영문 코드)
시스템 설명	- 시스템에 대한 업무, 위치 등에 대한 부가 정보
시스템 위치	- 시스템이 설치된 위치(노드) 정보
네트워크 특성	- 네트워크 속도, 대역폭, 유의 사항 등 네트워크 특성
전용 회선 정보	- 전용 회선을 사용할 경우 전용 회선 연결 방법과 속도 등의 정보
IP/URL	- 시스템 접속에 필요한 IP 또는 URL 정보
Port	- 접속에 필요한 Port 정보
Login 정보	- 시스템 로그인 ID와 암호
DB 정보	- 데이터베이스 연계 시 필요한 DBMS 유형, DBMS 로그인 정보
담당자 정보	- 해당 시스템의 인터페이스 담당자 연락처

**(3) 송수신 데이터 식별**

- 송수신 시스템 사이에서 교환되는 데이터는 규격화된 표준 형식에 따라 전송
- 인터페이스 설계 단계에서는 송수신 시스템 간에 전송되는 표준 항목과 업무 처리용 데이터 및 공통 코드 정보 등을 누락 없이 식별하고 인터페이스 명세서를 작성
- 송수신 전문 구성

구성	설명
전문 공통부	- 인터페이스 표준 항목을 포함 - 인터페이스 ID, 서비스 코드, 접속 IP 등
전문 개별부	- 업무처리에 필요한 데이터를 포함
전문 종료부	- 전송 데이터의 끝을 표시하는 문자 포함

## Section 2. 인터페이스 설계서 확인

### 1. 인터페이스 설계서 구성

#### (1) 인터페이스 목록

- 연계 업무와 연계에 참여하는 송수신 시스템의 정보, 연계 방식과 통신 유형 등에 대한 정보
- 주요 항목

주요 항목	설명
인터페이스 ID	- 인터페이스를 구분하기 위한 식별자, 명명 표준에 맞게 부여
인터페이스명	- 인터페이스의 목적을 나타내는 이름
송신 시스템	- 인터페이스를 통해 데이터를 전송하는 시스템
수신 시스템	- 인터페이스를 통해 전송된 데이터를 이용하는 시스템
대내외 구분	- 인터페이스가 기업 내부 시스템 간 또는 내외부 시스템 간에 발생하는지 여부
연계 방식	- 웹 서비스, FTP, DB Link, Socket 등 아키텍처에서 정의한 인터페이스 방식
통신 유형	- 동기(Request-Reply), 비동기(Send-Receive, Send-Receive-Acknowledge, Publish Subscribe) 등 아키텍처에서 정의한 통신 유형
처리 유형	- 실시간, 배치, 지연 처리 등 인터페이스 처리 유형 - 처리 유형이 실시간인 경우 수시, 그 외 상세 주기를 표시(매일 오전 10시, 매시 10분)
주기	- 인터페이스가 발생하는 주기
데이터 형식	- 고정 길이, XML 등 인터페이스 항목의 데이터 포맷
관련 요구 사항 ID	- 해당 인터페이스와 관련된 요구 사항 식별 정보

#### (2) 인터페이스 정의서

- 데이터 송신 시스템과 수신 시스템 간의 데이터 저장소와 속성 등의 상세 내역을 포함
- 주요 항목

주요 항목	설명
인터페이스 ID	- 인터페이스를 구분하기 위한 식별자, 명명 표준에 맞게 부여
최대 처리 횟수	- 단위 시간당 처리될 수 있는 해당 인터페이스 최대 수행 건수
데이터 크기(평균/최대)	- 해당 인터페이스 1회 처리 시 소요되는 데이터의 평균 크기와 최대 크기
시스템 정보	- 시스템명, 업무, 서비스 명, 연계방식, 담당자/연락처
데이터 정보	- 번호, 필드, 식별자 여부, 데이터 타입, 크기, 설명 등

### (3) 인터페이스 설계서 명세화

- 인터페이스 설계 가이드와 인터페이스 설계서 작성 양식을 준비
- 분석 단계에서 정의된 정보를 인터페이스 목록 양식에 맞춰 작성
- 인터페이스 정의서 작성 양식에 맞춰 송수신 시스템의 정보를 각각 작성
- 인터페이스 식별 당시 작성한 서비스의 프로그램 명세서를 확인하고 보완이 필요한 경우 수정
- 인터페이스 설계서 작성 양식에 맞춰 송수신 데이터 항목 상세를 작성
- 송수신 시스템 간의 코드 변환을 위한 코드 매핑 규칙을 인터페이스 정의서에 작성하고 코드 매핑 시 참고할 수 있도록 코드 매핑 테이블을 별도로 작성
- 인터페이스 설계 내용을 검토하고 보완

시스템명	예약발매시스템	서비스시스템명	API 연계	작성자	이홍직
API ID	API_005	API명	회원 등급 수정	작성일	2016.02.26
기능설명	회원 등급을 수신하고 정보를 UPDATE 한다.				

API 기본정보	응답기관명	응답시스템명	인터페이스 구분	인터페이스 방식	데이터유형	처리유형	요청기관명	요청시스템명	비고
	우리집	예약발매	대외	HTTP	JSON	실시간	주KM		
	protocol		method	domain			prefix	resource	
	https://		POST	api.com			/api	/ebiz/customer/modify.do	

API  
상세정보

Request										
parameter					description	type	length	example	default	비고
wctNo					창구번호	string				공통/필수
cgPsId					담당자 ID	string				공통/필수
uuld					송수신 걸음 값	string				공통/필수
cnntUsripAddr					접속사용자IP주소	string		192.168.0.1		공통/필수
trnOprBzDvCd					사업자구분코드	string		002	002	공통/필수
chgCustClCd					변경고객등급코드	string	1	N:일반, O:우등, S:특별, V:VIP		필수
custHstChgRsnCont					고객이력변경사유내용	string	200	테스트		필수
chgDt					변경일자	string	8	21060226		필수
Response										
parameter					description	type	length	example		비고
wctNo					창구번호	string				공통
cgPsId					담당자ID	string				공통
uuld					송수신걸음값	string				공통
msgCd					응답코드	string	9	IRZ000008		공통
msgTxt					응답메시지	string		정상적으로 처리 되었습니다.		공통
strResult					결과조회성공여부	string		SUCC:성공, FAIL:실패		공통
custNm					고객명	string	50	홍길동		
mbCrdNo					회원카드번호	string	10	1680000014		

## 2. 데이터 표준 확인

### (1) 인터페이스 데이터 표준 개념

- 인터페이스를 위해 인터페이스가 되어야 할 범위의 데이터들의 형식과 표준을 정의하는 것
- 데이터의 공통 영역을 추출하여 정의 하는 경우와, 한쪽의 데이터를 변환하는 경우가 있다

### (2) 인터페이스 데이터 표준 확인

- 상호 인터페이스 해야 할 모듈의 데이터 표준과 함께 인터페이스 데이터 표준을 같이 정의하고 산출물에 표현
- 인터페이스에 필요한 데이터 포맷의 표준을 표현한다.
- 데이터 포맷은 Json, DB, XML 등 다양한 형태로 표현이 가능

### (3) 인터페이스 데이터 표준 확인 절차

- 식별된 데이터 인터페이스를 통해 인터페이스 데이터 표준을 확인
- 식별된 인터페이스 기능을 통해 인터페이스 데이터 표준을 확인
- 데이터 인터페이스 및 식별된 인터페이스 기능을 통해 데이터 표준을 확인



## Section 3. 인터페이스 기능 구현

### 1. 내·외부 모듈 연계 방식

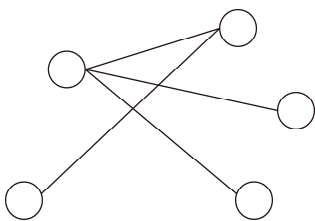
#### (1) EAI(Enterprise Application Integration)

##### ① EAI의 개념

- 기업에서 운영되는 서로 다른 플랫폼 및 애플리케이션들 간의 정보 전달, 연계, 통합을 가능하게 해주는 솔루션
- EAI를 사용함으로써 각 비즈니스 간 통합 및 연계성을 증대시켜 효율성을 높여 줄 수 있으며 각 시스템 간의 확장성을 높여줄 수 있다.

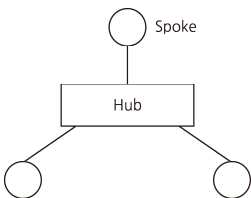
##### ② EAI의 구축유형

- Point-to-Point



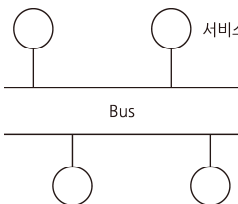
- 중간에 미들웨어를 두지 않고 각 애플리케이션 간 Point to Point 형태로 연결
- 솔루션 구매 없이 통합

- Hub & Spoke



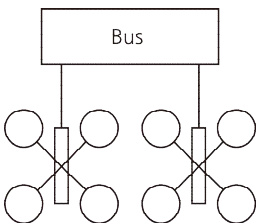
- 단일 접점이 허브 시스템을 통해 데이터를 전송하는 중앙 집중적 방식
- 모든 데이터 전송 보장
- 확장, 유지 보수 용이
- 허브 장애 시 전체 영향

- Message Bus (ESB 방식)



- 애플리케이션 사이 미들웨어(버스)를 두어 처리
- 미들웨어 통한 통합
- 어댑터가 각 시스템과 버스를 두어 연결하므로 뛰어난 확장성, 대용량 처리 가능

- Hybrid



- 그룹 내에는 Hub & Spoke 방식을 그룹 간 메시징 버스 방식을 사용
- 표준 통합 기술, 데이터 병목 현상 최소화

## (2) ESB(Enterprise Service Bus)

- 웹 서비스 중심으로 표준화된 데이터, 버스를 통해 이 기종 애플리케이션을 유연(loosely-coupled)하게 통합하는 핵심 플랫폼(기술)
- ESB는 Bus를 중심으로 각각 프로토콜이 호환이 되게끔 변환이 가능하고 서비스 중심으로 메시지 이동을 라우팅할 수 있다.
- 관리 및 보안이 쉽고 높은 수준의 품질 지원이 가능하지만 표준화가 미비하고 특정 벤더 종속, 성능 문제에서 개선되어야 할 부분이 남아 있다.
- ESB는 애플리케이션 간의 통합 측면에서 EAI와 유사하다고 볼 수 있으나 애플리케이션 보다는 서비스 중심으로 통합을 지향하는 아키텍처 또는 기술이다.

## (3) FEP(Front End Processor)

- 메인프레임에서 통신 과부하를 경감시키기 위해 전처리 작업을 하는 과정을 말한다
- 입력되는 데이터를 컴퓨터의 프로세서가 처리하기 전에 미리 처리하여 프로세서가 처리하는 시간을 줄여주는 프로그램이나 하드웨어
- 전용선 또는 VPN을 이용하 연결된 기관 간, 정해진 인터페이스로 배치파일(SAM)을 전달한다

## 2. 인터페이스 연계 기술

- DB Link
  - 데이터베이스에 제공하는 DB Link 객체를 이용
  - 수신 시스템에서 DB Link를 생성하고 송신 시스템에서 해당 DB Link를 직접 참조하는 방식
- DB Connection
  - 수신 시스템의 WAS에서 송신 시스템 DB로 연결하는 DB Connection Pool을 생성하고 연계 프로그램에서 해당 DB Connection Pool명을 이용
- JDBC
  - 수신 시스템의 프로그램에서 JDBC 드라이버를 이용하여 송신 시스템 DB와 연결
  - DBMS 유형, DBMS 서버 IP와 Port, DB instance 정보가 필요함
- API / OpenAPI
  - 송신 시스템의 애플리케이션 프로그래밍 인터페이스 프로그램
  - API명, 입출력 파라미터 정보가 필요함
- Web Service
  - WSDL(Web Services Description Language), UDDI(Universal Description, Discovery and Integration), SOAP(Simple Object Access Protocol) 프로토콜을 이용하여 연계
- Hyper Link
  - 웹 애플리케이션에서 하이퍼링크(Hyper Link) 이용
- Socket
  - 통신을 위한 소켓(Socket)을 생성하여 포트를 할당하고 클라이언트의 통신 요청 시 클라이언트와 연결하고 통신하는 네트워크 기술

### 3. 인터페이스 전송 데이터

#### (1) JSON (JavaScript Object Notation)

- Javascript 객체 문법으로 구조화된 데이터를 표현하기 위한 문자 기반의 표준 포맷
- Javascript에서 객체를 만들 때 사용하는 표현식
- JSON 표현식은 사람과 기계 모두 이해하기 쉬우며 용량이 작아서, 최근에는 JSON이 XML을 대체해서 데이터 전송 등에 많이 사용한다.
- 프로그래밍 문법이 아닌 단순히 데이터를 표시하는 표현 방법일 뿐이다.
- JSON 데이터는 이름과 값의 쌍으로 이루어진다.
- JSON 문법

```
{
  "firstName": "Kwon",
  "lastName": "YoungJae",
  "email": "kyoje11@gmail.com",
  "hobby": ["puzzles","swimming"]
}
```

#### (2) XML (eXtensible Markup Language)

- 웹에서 구조화한 문서를 표현하고 전송하도록 설계한 마크업 언어
- XML은 문서내용에 대한 구조와 의미를 기술하기 위한 언어
- 특수한 목적을 갖는 마크업 언어를 만드는데 사용하도록 권장하는 다목적 마크업 언어
- 다양한 표현이 가능하고, 확장성이 뛰어나며 간단하다.
- HTML처럼 데이터를 보여주는 목적이 아닌 데이터를 저장하고 전달할 목적으로 만들어졌다.
- HTML처럼 태그가 정해져 있지 않고, 사용자가 직접 정의할 수 있다.
- XML 예제

```
<?xml version="1.0" encoding="UTF-8"?>
<information type="필기">
  <subject>
    <no>1</no>
    <name>소프트웨어설계</name>
    <point>80</point>
  </subject>
  <subject>
    <no>2</no>
    <name>소프트웨어개발</name>
    <point>60</point>
  </subject>
</shop>
```

### (3) CSV (Comma Separated Values)

- 몇 가지 필드를 쉼표(,)로 구분한 텍스트 데이터 및 텍스트 파일
- 표 형태의 데이터를 저장하는 파일 형식
- CSV 예제

```
1,소프트웨어설계,80
2,소프트웨어개발,60
3,데이터베이스구축,85
```

### (4) YAML

- XML, C, 파이썬, 펄, RFC2822에서 정의된 e-mail 양식에서 개념을 얻어 만들어진 '사람이 쉽게 읽을 수 있는' 데이터 직렬화 양식
- 2001년에 클라크 에반스가 고안했고, Ingy dot Net 및 Oren Ben-Kiki와 함께 디자인했다
- 문법은 상대적으로 이해하기 쉽고, 가독성이 좋도록 디자인되었다.
- 들여쓰기 및 XML의 특수기호를 사용하기 때문에, XML과 거의 비슷하다

## 4. 인터페이스 구현

### (1) AJAX(Asynchronous JavaScript and XML)

#### ① AJAX의 개념

- 자바스크립트를 이용해 서버와 브라우저가 비동기 방식으로 데이터를 교환할 수 있는 통신 기능
- 브라우저가 가지고있는 XMLHttpRequest 객체를 이용해서 전체 페이지를 새로 고치지 않고도 페이지의 일부만을 위한 데이터를 로드하는 기법

#### ② 비동기 방식



- 웹페이지를 리로드하지 않고 데이터를 불러오는 방식
- 페이지 리로드시 전체 리소스를 다시 불러와야 하지만, 비동기 방식을 이용할 경우 필요한 부분만을 불러와 사용하므로 불필요한 리소스가 발생하지 않는다.

#### ③ AJAX의 장/단점

- 장점
  - 웹페이지의 속도향상
  - 서버의 처리가 완료될 때까지 기다리지 않고 처리가 가능하다.
  - 서버에서 Data만 전송하면 되므로 전체적인 코딩의 양이 줄어든다.

- 기존 웹에서는 불가능했던 다양한 UI를 가능하게 해준다.
- 단점
  - 히스토리 관리가 되지 않는다.
  - 사용자에게 아무런 진행 정보가 주어지지 않는다.
  - AJAX를 쓸 수 없는 브라우저에 대한 문제 이슈가 있다.
  - 다른 도메인과는 통신이 불가능하다.
  - 페이지 이동없는 통신으로 인한 보안상의 문제가 있다.

#### ④ AJAX 예제

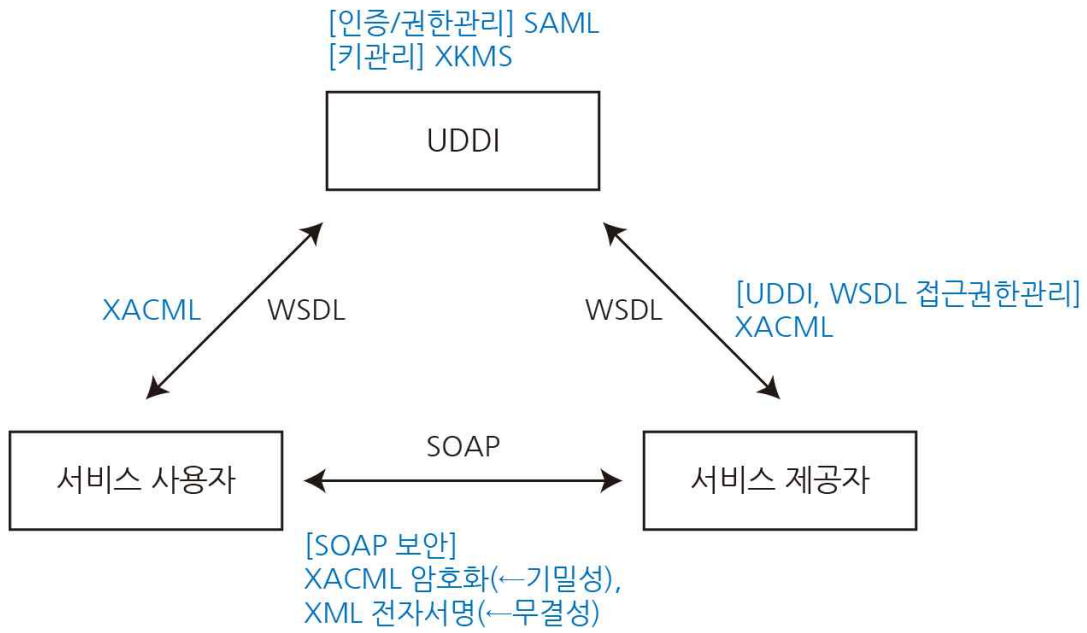
```
$.ajax({  
  url: 'https://www.njobler.net/test',  
  type: 'GET',  
  success: function onData (data) {  
    console.log(data);  
  },  
  error: function onError (error) {  
    console.error(error);  
  }  
});
```

## (2) SOAP(Simple Object Access Protocol)

### ① SOAP의 개념

- HTTP, HTTPS, SMTP 등을 통해 XML 기반의 메시지를 컴퓨터 네트워크 상에서 교환하는 프로토콜
- SOAP은 웹 서비스에서 기본적인 메시지를 전달하는 기반이 된다.
- 보통 RPC(Remote Procedure Call) 패턴을 많이 사용한다.
- 클라이언트에서 서버 쪽으로 메시지를 요청하고, 서버는 그 메시지에 반응한다.
- 레스트풀(RESTful)보다 상대적으로 개발이 어렵다.
- DCOM이나 CORBA의 호환성과 보안 문제로 등장했다.
- SOA 개념을 실현하기 위한 기술이다.

### ② SOAP 구성



- SOAP(Simple Object Access Protocol)
  - HTTP, HTTPS, SMTP등을 사용하여 XML 기반의 메시지를 네트워크 상에서 교환하는 형태의 프로토콜
  - XML을 근간으로 헤더와 바디를 조합하는 디자인 패턴으로 설계되어 있다.
- UDDI(Universal Description, Discovery and Integration)
  - 인터넷에서 전 세계의 비즈니스 업체 목록에 자신의 목록을 등록하기 위한, XML기반의 규격
- WSDL(Web Services Description Language)
  - 웹 서비스 기술언어 또는 기술된 정의 파일의 총칭으로 XML로 기술
  - 서비스 제공 장소, 서비스 메시지 포맷, 프로토콜 등이 기술된다.

### ③ SOAP 보안 프로토콜

- SAML(인증/권한관리)
  - 이기종 시스템 간 권한 확인
  - 인증 및 권한 정보 명세
  - 보안 토큰
- XKMS(키관리)
  - 부인방지
  - 기존 PKI 연동 용이
- XACML(접근제어)
  - 정보 접근을 위한 XML 명세
  - UDDI 및 WSDL 항목 접근 제어

### ④ SOAP 장/단점

- 장점
  - 기본적으로 HTTP 기반 위에서 동작하기 때문에, 프록시와 방화벽에 구애받지 않고 통신이 가능하다.
  - HTTP 이외의 다른 트랜스포트 프로토콜들(SMTP)을 사용할 수 있다.
  - 플랫폼 및 프로그래밍 언어에 독립적이다.
- 단점
  - XML 포맷의 형태로 보내기 때문에 다른 기술과 비교해서 상대적으로 느리다.

### ⑤ SOAP 기본 구조

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Header>
  ...
  </soap:Header>

  <soap:Body>
  ...
  <soap:Fault>
  ...
  </soap:Fault>
  </soap:Body>

</soap:Envelope>
```

### (3) REST

#### ① REST의 개념

- HTTP URI를 통해 자원을 명시하고, HTTP Method(POST, GET, PUT, DELETE)를 통해 해당 자원에 대한 CRUD Operation을 적용하는 것을 의미한다.
- 자원 기반의 구조(ROA, Resource Oriented Architecture)의 개념으로 구현되었다.

#### ② REST 구성요소

- 자원(Resource), URI
  - 서버에 존재하는 데이터의 총칭
  - 모든 자원은 고유의 URI를 가지며 클라이언트는 이 URI를 지정하여 해당 자원에 대해 CRUD 명령을 수행한다.
- 행위(Verb), Method
  - 클라이언트는 URI를 이용해 자원을 지정하고 자원을 조작하기 위해 Method를 사용한다.
  - HTTP 프로토콜에서는 GET , POST , PUT , DELETE 같은 Method를 제공한다.
- 표현(Representation)
  - REST에서 하나의 자원은 JSON , XML , TEXT , RSS 등 여러 형태의 Representation으로 나타낼 수 있다.

#### ③ CRUD Operation, HTTP Method

- Create : POST (자원 생성)
- Read : GET (자원의 정보 조회)
- Update : PUT (자원의 정보 업데이트)
- Delete : DELETE (자원 삭제)



## ④ REST 특징

- 유니폼 인터페이스
  - HTTP 표준만 따르면 어떤 언어나 플랫폼에서 사용해도 사용이 가능한 인터페이스 스타일
  - 특정 언어에 상관없이 사용이 가능하다.
- Stateless(상태 정보 유지 안함)
  - Rest는 상태 정보를 유지하지 않는다.
  - 서버는 각각의 요청을 완전히 다른 것으로 인식하고 처리를 한다.
- Cacheable(캐시가능)
  - HTTP가 가진 캐싱 기능이 적용 가능
- Self-descriptiveness (자체 표현 구조)
  - REST API 메시지만 보고도 쉽게 이해 할 수 있는 자체 표현 구조로 되어있다.

## ⑤ REST 장/단점

- 장점
  - 별도의 인프라 구축이 필요없다.
  - 클라이언트와 서버의 분리
  - 플랫폼에 독립적
  - 쉬운 사용
- 단점
  - 표준이 존재하지 않음
  - HTTP Method의 한계
  - RDBMS와 맞지 않음

## ⑥ REST API 예제

구분	표현
강의 생성	HTTP POST, http://njobler.net/lecture <pre>{   "lec":{     "name":"정보처리실기"     "price":"70,000"   } }</pre>
강의 조회	HTTP GET, http://njobler.net/lecture/정보처리실기
강의 수정	HTTP PUT, http://njobler.net/lecture <pre>{   "lec":{     "name":"정보처리 실기"     "price":"65,000"   } }</pre>
강의 삭제	HTTP DELETE, http://njobler.net/lecture/정보처리실기

## ⑦ RESTful

- REST의 원리를 따르는 시스템

## 5. 인터페이스 보안

### (1) 인터페이스 보안 취약점 분석

- 인터페이스의 보안 취약점을 분석
  - 인터페이스 각 구간의 구현 현황을 분석
  - 인터페이스 각 구간의 보안 취약점을 분석
- 분석된 보안 취약점을 근거로 인터페이스 보안 기능을 적용
  - 네트워크 구간에 보안 기능을 적용
  - 애플리케이션에 보안 기능을 적용
  - 데이터베이스에 보안 기능을 적용

### (2) 인터페이스 보안 기능 적용

#### ① 네트워크 영역

- 인터페이스 송/수신 간 스니핑 등 이용한 데이터 탈취 및 변조 위협 방지 위해 네트워크 트래픽에 대한 암호화 설정
- IPSec, SSL, S-HTTP 등 다양한 방식으로 적용

#### ② 애플리케이션 영역

- 시큐어코딩 가이드를 참조하여 애플리케이션 코드 상 보안 취약점을 보완하는 방향으로 보안 기능 적용

#### ③ DB 영역

- DB, 스키마, 엔티티의 접근 권한과 프로시저, 트리거 등 DB 동작 객체의 보안 취약점에 보안 기능을 적용
- 민감 데이터를 암호화, 익명화 등을 통해 데이터 자체 보안 방안도 고려

## Section 4. 인터페이스 구현 검증

### 1. 인터페이스 검증

#### (1) 인터페이스 구현 검증 도구

- xUnit
  - 다양한 언어를 지원하는 단위 테스트 프레임 워크
- STAF
  - 서비스 호출 및 컴포넌트 재사용 등 다양한 환경을 지원하는 테스트 프레임워크
- FitNesse
  - 웹 기반 테스트케이스 설계, 실행, 결과 확인 등을 지원하는 테스트 프레임워크
- NTAF
  - FitNesse의 장점과 STAF의 장점을 통합한 Naver의 테스트 자동화 프레임워크
- Selenium
  - 다양한 브라우저 및 개발 언어를 지원하는 웹 어플리케이션 테스트 프레임워크
- watir
  - Ruby를 사용하는 애플리케이션 테스트 프레임워크

#### (2) 인터페이스 구현 감시 도구

- APM(Application Performance Management)을 사용하여 동작상태 감시
- 데이터베이스, 웹 애플리케이션의 다양한 정보를 조회하고 분석하여 시각화 함
- 종류 : 스카우터(Scouter), 제니퍼(Jennifer) 등

### 2. 인터페이스 오류 처리

#### (1) 인터페이스 오류 발생 알림

- 화면을 통한 오류 메시지 표시
- 오류 발생시 SMS 발송
- 오류 발생시 이메일 발송

#### (2) 인터페이스 오류 발생 확인

- 인터페이스 오류 로그 확인
- 인터페이스 오류 테이블 확인
- 인터페이스 감시도구로 확인

## 07 객체지향 구현

### Section 1. 객체지향 설계

#### 1. 객체지향 (Object Oriented Programming-OOP)

##### (1) 객체지향 개념

- 현실 세계의 유형, 무형의 모든 대상을 객체(Object)로 나누고, 객체의 행동(method)과 고유한 값(Attribute)을 정의하여 설계하는 방법
- 객체를 만들고 조작하며 객체끼리 관계를 맺음으로써 다수의 객체가 함께 수행될 수 있게 한다.

##### (2) 객체지향 구성요소

- 클래스 (Class)
  - 유사한 종류의 유형/무형의 존재를 속성과 연산을 정의해서 만든 틀
  - 다른 클래스와 독립적으로 디자인 한다
  - 데이터를 추상화 하는 단위
- 객체 (Object)
  - 클래스의 인스턴스
  - 객체는 자신 고유의 속성을 가지며, 클래스에서 정의한 연산을 수행
  - 객체의 연산은 클래스에 정의된 연산에 대한 정의를 공유함으로써 메모리를 경제적으로 사용
- 속성 (Attribute)
  - 객체들이 가지고 있는 고유한 데이터를 단위별로 정의한 것
  - 성질, 분류, 수량, 현재 상태 등에 대해 표현한 값
- 메서드 (Method)
  - 어떤 특정한 작업을 수행하기 위한 명령문의 집합
  - 객체가 가지고 있는 속성들을 변경할 수 있는 하나의 연산
- 메시지 (Message)
  - 객체에게 어떤 행위를 하도록 지시
  - 객체의 메서드를 호출함으로써 객체간의 상호작용을 할 수 있도록 한다.

##### (3) 객체지향언어의 특징

- 캡슐화(Encapsulation)
  - 데이터(Attribute)와 데이터를 처리하는 행동(method)을 하나로 묶은 것
  - 캡슐화된 객체의 세부내용은 외부에 은폐(정보은닉)되어, 오류의 파급 효과가 적다.
  - 캡슐화된 객체들은 재사용이 용이
  - 객체들 간의 메시지를 주고 받을 때, 해당 객체의 세부 내용을 알 필요가 없으므로 인터페이스가 단순해지고, 결합도가 낮아진다.
- 정보은닉(Information Hiding)
  - 캡슐화의 가장 중요한 개념
  - 다른 객체에게 자신의 데이터를 숨기고, 자신이 정의한 행동만을 통하여 접근을 허용

- 상속(Inheritance)
  - 상위클래스(부모클래스)의 모든 데이터와 행동을 하위 클래스가 물려받는 것
  - 상속을 이용하면 하위 클래스는 상위 클래스의 데이터와 행동을 자신의 클래스에 다시 정의하지 않아도 된다.
  - 하위 클래스는 상위 클래스에서 상속받은 요소 외에 새로운 데이터와 행동을 추가하여 사용할 수 있다.
  - 상위클래스의 요소들을 사용할 수 있기 때문에, 소프트웨어 재사용을 증대시키는 중요한 개념
- 다형성(Polymorphism)
  - 하나의 메시지에 대해 각 객체가 가지고 있는 여러 가지 방법으로 응답할 수 있는 개념
  - 객체에서 동일한 메서드명을 인자값의 유형이나 개수만 틀리게 하는 오버로딩이 존재
  - 객체에서 상속받은 메서드를 재정의 하는 오버라이딩이 존재
- 추상화(Abstraction)
  - 어떤 실체로부터 공통적인 부분들만 모아놓은 것
  - 하위클래스들에 존재하는 공통적인 메소드를 상위클래스 혹은 인터페이스로 정의하고, 하위클래스가 해당 메소드를 재정의 하는 것

#### (4) 객체지향 설계원칙(SOLID)

- 단일 책임 원칙(SRP, Single responsibility principle)
  - 한 클래스는 하나의 책임만을 가져야 한다.
- 개방 폐쇄 원칙(OCP, Open-closed principle)
  - 확장에는 열려 있고, 수정에는 닫혀 있어야 한다.
  - 기존의 코드를 변경하지 않으면서(Closed), 기능을 추가할 수 있도록(Open) 설계
- 리스코프 치환 원칙(LSP, Liskov substitution principle)
  - 자식 클래스는 언제나 자신의 부모 클래스를 대체할 수 있어야 한다.
  - 부모 클래스가 들어갈 자리에 자식 클래스를 넣어도 계획대로 작동해야 한다.
- 인터페이스 분리 원칙(ISP, Interface Segregation Principle)
  - 자신이 사용하지 않는 인터페이스는 구현하지 말아야 한다.
  - 자신이 사용하지 않는 인터페이스 때문에 영향을 받아서는 안 된다.
- 의존성 역전 원칙(DIP, Dependency Inversion Principle)
  - 의존 관계를 맺을 때 자주 변화하는 것보다, 변화가 거의 없는 것에 의존해야 한다.
  - 구체적인 클래스보다 인터페이스나 추상 클래스와 의존 관계를 맺어야 한다.

## 2. 디자인패턴

### (1) 디자인 패턴(Design Pattern) 개념

- 객체 지향 프로그래밍 설계를 할 때 자주 발생하는 문제들에 대해 재사용할 수 있도록 만들어놓은 패턴들의 모음
- 이미 만들어져서 잘 되는 것을 활용하여 재사용함으로써 프로그램 최적화에 도움을 준다.
- 효율적인 코드를 만들기 위한 방법론

### (2) 디자인 패턴 구조

- 패턴의 이름과 유형
  - 패턴을 부를 때 사용하는 이름과 패턴의 유형
- 문제 및 배경
  - 패턴이 사용되는 분야 또는 배경, 해결하는 문제
- 솔루션
  - 패턴을 이루는 요소들, 관계, 협동 과정
- 결과
  - 패턴을 사용하면 얻게 되는 이점과 영향
- 사례
  - 간단한 적용사례
- 샘플 코드
  - 패턴이 적용된 원시코드

### (3) GoF 디자인 패턴

- GoF(Gang of Four) 의 디자인 패턴
  - 에리히 감마(Erich Gamma), 리처드 헬름(Richard Helm), 랄프 존슨(Ralph Johnson), 존 블리시디스(John Vissides) 에 의해 개발 영역에서 디자인 패턴을 구체화 하고 체계화 시킴
  - 23가지의 디자인 패턴을 정리
  - 각각의 디자인 패턴을 생성(Creational), 구조(Structural), 행위(Behavioral) 3가지로 분류
- Gof 디자인 패턴 분류

생성 패턴	<ul style="list-style-type: none"> <li>- 객체 생성과 관련한 패턴</li> <li>- 객체 생성에 있어서 프로그램 구조에 영향을 크게 주지 않는 유연성 제공</li> </ul>
구조 패턴	<ul style="list-style-type: none"> <li>- 클래스나 객체를 조합해서 더 큰 구조를 만드는 패턴</li> </ul>
행위 패턴	<ul style="list-style-type: none"> <li>- 객체나 클래스 사이의 알고리즘이나 책임 분배에 관련된 패턴</li> </ul>

생성(Creational) 패턴	구조(Structural) 패턴	행위(Behavioral) 패턴
<ul style="list-style-type: none"> <li>- Abstract Factory</li> <li>- Builder</li> <li>- Factory Method</li> <li>- Prototype</li> <li>- Singleton</li> </ul>	<ul style="list-style-type: none"> <li>- Adapter</li> <li>- Bridge</li> <li>- Composite</li> <li>- Decorator</li> <li>- Facade</li> <li>- Flyweight</li> <li>- Proxy</li> </ul>	<ul style="list-style-type: none"> <li>- Chain of Responsibility</li> <li>- Command</li> <li>- Interpreter</li> <li>- Iterator</li> <li>- Mediator</li> <li>- Memento</li> <li>- Observer</li> <li>- State</li> <li>- Strategy</li> <li>- Template Method</li> <li>- Visitor</li> </ul>

#### (4) 디자인 패턴 종류

##### ① 생성패턴

- 추상 팩토리(Abstract Factory)
  - 구체적인 클래스에 의존하지 않고 서로 연관되거나 의존적인 객체들의 조합을 만드는 인터페이스를 제공하는 패턴
  - 클라이언트 입장에서 실제 구현 클래스를 알 필요 없이 인터페이스만으로 시스템을 조작할 수 있도록 한다.
  - 객체가 생성되거나 구성, 표현되는 방식과 무관하게 시스템을 독립적으로 만들 수 있게 도와준다.
- 빌더(Builder)
  - 복합 객체의 생성과 표현을 분리하여 동일한 생성 절차에서도 다른 표현 결과를 만들어낼 수 있음
  - 객체를 생성하는 과정의 약속과 구체적인 알고리즘 구현을 분리

```
new User.Builder(10)
    .name("이홍직")
    .password("1234")
    .age(43)
    .build();
```

- 팩토리 메소드(Factory Method)
  - 객체 생성 처리를 서브 클래스로 분리해 처리하도록 캡슐화하는 패턴
  - 객체의 생성 코드를 별도의 클래스/메서드로 분리함으로써 객체 생성의 변화에 대비하는 데 유용하다.
  - 상위클래스에서 객체를 생성하는 인터페이스를 정의하고, 하위클래스에서 인스턴스를 생성하도록 하는 방식
  - Virtual-Constructor 패턴이라고도 함
- 프로토타입(Prototype)
  - 원본 객체를 복사함으로써 객체를 생성함
  - prototype을 먼저 생성하고 인스턴스를 복제하여 사용하는 구조
  - java의 clone()을 이용하여 생성하고자 하는 객체에 clone에 대한 Override를 해준다.

- 싱글톤(Singleton)
  - 어떤 클래스의 인스턴스는 하나임을 보장하고 어디서든 참조할 수 있도록 함

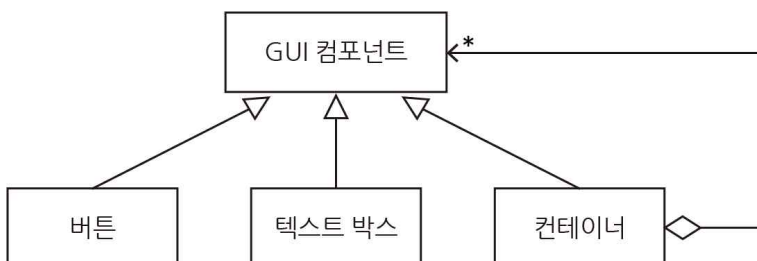
```
public class CarClass{
    private static CarClass car = new CarClass(); // 자신의 객체를 바로 생성
    private CarClass(){} // 생성자가 private 이기 때문에 외부에서 접근 불가
    public static CarClass getInstance(){
        return car;
    }
}
```

## ② 구조 패턴

- 어댑터(Adapter)
  - 클래스의 인터페이스를 다른 인터페이스로 변환하여 다른 클래스가 이용할 수 있도록 함
  - 인터페이스 호환성이 맞지 않아 같이 쓸 수 없는 클래스를 연관 관계로 연결해서 사용할 수 있게 해주는 패턴
  - 클래스의 인터페이스를 사용자가 기대하는 다른 인터페이스로 변환하는 패턴
  - 관련성이 없거나 예측하지 못한 클래스들과 협동하는 재사용 가능한 클래스를 생성할 때 사용한다.
- 브리지(Bridge)
  - 구현부에서 추상층을 분리하여 각자 독립적으로 변형할 수 있게 하는 패턴

```
interface Shape{
    public void draw();
}
class CircleShape implements Shape{
    public void draw(){
        printf("Circle");
    }
}
class SquareShapeimplements Shape{
    public void draw(){
        printf("Square");
    }
}
```

- 컴포지트(Composite)
  - 객체들의 관계를 트리 구조로 구성하여 복합 객체와 단일 객체를 구분없이 다룸





- 데코레이터(Decorator)
  - 주어진 상황 및 용도에 따라 어떤 객체에 다른 객체를 덧붙이는 방식
  - 객체의 결합 을 통해 기능을 동적으로 유연하게 확장 할 수 있게 해주는 패턴
  - 기능 확장이 필요할 때 서브 클래스링(subclassing) 대신 쓸 수 있는 유연한 대안을 제공한다.
  - 상속을 사용하지 않고도 객체의 기능을 동적으로 확장할 수 있도록 해준다.
  - 어떤 객체에 책임(responsibility)을 동적으로 추가 할 수 있도록 한다.
- 퍼사드(Facade)
  - 서브시스템에 있는 인터페이스 집합에 대해 하나의 통합된 인터페이스(Wrapper) 제공
  - 서브시스템 사이의 의사소통 및 종속성을 최소화하기 위하여 단순화된 하나의 인터페이스를 제공하는 패턴
  - 서브시스템의 가장 앞쪽에 위치하면서 서브시스템에 있는 객체들을 사용할 수 있는 역할을 하는 패턴
- 플라이웨이트(Flyweight)
  - 크기가 작은 여러 개의 객체를 매번 생성하지 않고 가능한 한 공유할 수 있도록 하여 메모리를 절약함
  - 많은 수의 객체를 생성해야 할 때 사용하는 패턴
- 프록시(Proxy)
  - 접근이 어려운 객체로의 접근을 제어하기 위해 객체의 Surrogate(대리)나 Placeholder(대체글)를 제공
  - 객체를 직접 참조하지 않고, 객체를 대행하는 객체를 통해 접근 하는 방식

### ③ 행위 패턴

- 역할 사슬 패턴(Chain of Responsibility), 책임 연쇄
  - 요청을 받는 객체를 연쇄적으로 묶어 요청을 처리하는 객체를 만날 때까지 객체 Chain을 따라 요청을 전달함
  - 여러 개의 객체 중에서 어떤 것이 요구를 처리할 수 있는지를 사전에 알 수 없을 때 사용한다.
  - 자신이 처리할 수 없는 경우 다음 객체에게 문제를 넘기게 된다.
- 커맨드(Command)
  - Client가 보낸 요청을 객체로 캡슐화하여 이를 나중에 이용할 수 있도록 필요한 정보를 저장, 로깅, 취소할 수 있게 하는 패턴
  - 요청을 객체로 감싸서 관리하는 패턴
- 인터프리터(Interpreter)
  - 일련의 규칙으로 정의된 문법적 언어를 해석하는 패턴
  - 간단한 언어의 문법을 정의하고 해석하는 패턴
- 반복자(Iterator)
  - 내부를 노출하지 않고 내부에 들어있는 모든 항목에 접근할 수 있게 해 주는 방법을 제공해 주는 패턴
  - 집합 객체의 요소들에 대해 순서대로 접근하는 방법을 제공한다.
  - 배열(Array), 배열리스트(ArrayList), 해시 테이블과 같은 객체를 처리하는 데 사용하는 패턴
  - 서로 다른 집합 객체 구조에 대해 동일한 방법으로 순회할 수 있다.
- 중재자(Mediator)
  - 클래스 간의 복잡한 관계들을 캡슐화하여 하나의 클래스에서 관리하도록 처리하는 패턴
  - 여러 객체들 사이에 중재자를 추가하여 중재자가 모든 객체들의 통신을 담당하도록 하는 패턴
  - 객체간의 통제와 지시의 역할을 하는 중재자를 두어 객체지향의 목표를 달성하게 해준다.

- 메멘토(Memento)
  - 객체의 상태 정보를 저장하고 사용자의 필요에 의하여 원하는 시점의 데이터를 복원 할 수 있는 패턴
  - 캡슐화를 위반하지 않고 객체의 이전 상태를 저장하고 복원할 수 있는 디자인 패턴
- 옵저버(Observer)
  - 객체의 상태 변화를 관찰하는 옵저버들의 목록을 객체에 등록하여 상태 변화시 각 옵저버에게 통지하는 패턴
  - 어떤 객체의 상태가 변할 때 그 객체에 의존성을 가진 다른 객체들이 그 변화를 통지받고 자동으로 갱신될 수 있게 하는 패턴
  - 일대 다의 객체 의존 관계를 정의하며, 한 객체의 상태가 변화되었을 때, 의존 관계에 있는 다른 객체들에게 자동적으로 변화를 통지하고 변경 시킨다.
- 상태(State)
  - 객체의 상태에 따라 동일한 동작을 다르게 처리해야할 때 사용
- 전략(Strategy)
  - 실행 중에 알고리즘을 선택할 수 있게 하는 패턴
  - 특정한 계열의 알고리즘들을 정의하고 각 알고리즘을 캡슐화하며 이 알고리즘들을 해당 계열 안에서 상호 교체가 가능하게 만든다.
  - 특정 컨텍스트에서 알고리즘을 별도로 분리하는 설계 방법
  - 예) GPS 신호를 수신하는 경우와 수신하지 못하는 경우에 따라 차량의 위치를 구하는 다른 알고리즘을 선택하고자 할 때 가장 적합한 설계 패턴
- 템플릿 메소드(Template Method)
  - 특정 작업을 처리하는 일부분을 서브 클래스로 캡슐화하여 전체적인 구조는 바꾸지 않으면서 특정 단계에서 수행하는 내용을 바꾸는 패턴
  - 알고리즘의 구조는 그대로 유지하면서 서브클래스에서 특정 단계를 재정의 할 수 있다.
- 방문자(Visitor)
  - 실제 로직을 가지고 있는 객체(Visitor)가 로직을 적용할 객체를 방문하면서 실행하는 패턴
  - 알고리즘을 객체 구조에서 분리시키는 디자인 패턴
  - 객체지향 원칙(SOLID) 중 하나인 개방-폐쇄 원칙(Open-Closed Principle, OCP)을 적용하는 방법

## 08 애플리케이션 테스트 관리

### Section 1. 애플리케이션 테스트케이스 설계

#### 1. 소프트웨어 테스트

##### (1) 소프트웨어 테스트의 개념

- 구현된 소프트웨어에서, 사용자가 요구하는 기능의 동작과 성능, 사용성, 안정성 등을 만족하기 위하여 소프트웨어의 결함을 찾아내는 활동
- 노출되지 않은 숨어있는 결함(Fault)을 찾기 위해 소프트웨어를 작동시키는 일련의 행위와 절차
- 오류 발견을 목적으로 프로그램을 실행하여 품질을 평가하는 과정
- 개발된 소프트웨어의 결함과 문제를 식별하고 품질을 평가하며 품질을 개선하기 위한 일련의 활동

##### (2) 소프트웨어 테스트의 필요성

- 오류 발견 관점
- 오류 예방 관점
- 품질 향상 관점

##### (3) 소프트웨어 테스트의 기본 원칙

- 테스트는 결함이 존재함을 밝히는 활동이다.
- 완벽한 테스트는 불가능하다.
- 테스트는 개발 초기에 시작해야 한다.
- 결함 집중(Defect Clustering)
  - 애플리케이션 결함의 대부분은 소수의 특정한 모듈에 집중되어 존재한다.
  - 파레토 법칙 : 전체 결과의 80%가 전체 원인의 20%에서 일어나는 현상
- 살충제 패러독스(Pesticide Paradox)
  - 동일한 테스트 케이스로 반복 실행하면 결함을 발견할 수 없으므로 주기적으로 테스트 케이스를 리뷰하고 개선해야 한다.
- 테스트는 정황(Context)에 의존한다.
  - 정황과 비즈니스 도메인에 따라 테스트를 다르게 수행하여야 한다.
- 오류-부재의 귀변(Absence of Errors Fallacy)
  - 사용자의 요구사항을 만족하지 못하는 오류를 발견하고 그 오류를 제거하였다 해도, 해당 애플리케이션의 품질이 높다고 말할 수 없다.

##### (4) 테스트 프로세스

- 테스트 계획
- 테스트 분석 및 디자인
- 테스트 케이스 및 시나리오작성
- 테스트 수행
- 테스트 결과 평가 및 리포팅

## (5) 테스트 산출물

- 테스트 계획서
  - 테스트 목적과 범위 정의, 대상 시스템 구조 파악, 테스트 수행 절차, 테스트 일정, 조직의 역할 및 책임 정의, 종료 조건 정의 등 테스트 수행을 계획한 문서
- 테스트 케이스
  - 소프트웨어가 사용자의 요구사항을 준수하는지 확인하기 위해 설계된 입력 값, 테스트 조건, 기대 결과로 구성된 테스트 항목의 명세서
  - 케이스를 작은 단위로 나누어 각 단위의 입력 값, 테스트 조건, 기대 결과를 기술한다.
- 테스트 시나리오
  - 테스트를 위한 절차를 명세한 문서
  - 테스트 수행을 위한 여러 개의 테스트 케이스의 집합으로 테스트 케이스의 동작 순서를 기술한 문서
- 테스트 결과서
  - 테스트 결과를 정리한 문서
  - 테스트 프로세스를 리뷰하고, 테스트 결과를 평가하고 리포팅하는 문서

## 2. 테스트 오라클

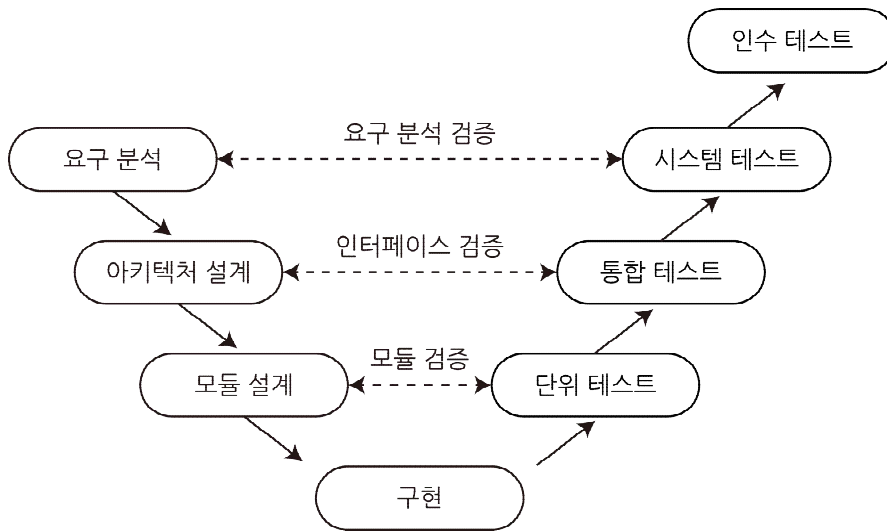
### (1) 테스트 오라클의 개념

- 테스트의 결과가 참인지 거짓인지를 판단하기 위해서 사전에 정의된 참 값을 입력하여 비교하는 기법 및 활동

### (2) 테스트 오라클의 유형

- 참 오라클 (True)
  - 모든 입력 값에 대하여 기대하는 결과를 생성함으로써 발생한 오류를 모두 검출할 수 있는 오라클
  - 항공기, 임베디드, 발전소 소프트웨어 등 크리티컬한 업무
- 샘플링 오라클 (Sampling)
  - 특정한 몇 개의 입력 값에 대해서만 기대하는 결과를 제공해 주는 오라클
  - 일반, 업무용, 게임, 오락등의 일반적인 업무
- 휴리스틱 오라클 (Heuristic)
  - 샘플링 오라클을 개선한 오라클로, 특정 입력 값에 대해 올바른 결과를 제공하고, 나머지 값들에 대해서는 휴리스틱(추정)으로 처리하는 오라클
- 일관성 검사 오라클 (Consistent)
  - 애플리케이션 변경이 있을 때, 수행 전과 후의 결과 값이 동일한지 확인하는 오라클

### 3. 테스트 레벨



#### (1) 단위테스트

- 소프트웨어 개발에서 테스트 프로세스의 첫 단계
- 에러를 줄이기 위한 의도로 작성된 코드에 대한 분석을 진행
- 코드가 효율적으로 작성되었는지, 프로젝트 내에 합의된 코딩 표준을 준수하고 있는지도 검증
- 모듈 설계 단계에서 준비된 테스트 케이스를 이용하며, 코드를 개발한 개발자가 직접 수행

#### (2) 통합 테스트

- 각각의 모듈들을 통합하여, 통합된 컴포넌트 간의 인터페이스와 상호작용 과정의 오류를 발견하는 작업을 수행
- 코드를 직접 확인하는 형태가 아닌 프로그램을 실행하며 테스트를 진행
- 설계 단계에서 준비된 테스트 케이스를 사용하여 테스트가 진행되며, 일반적으로 개발자가 진행

#### (3) 시스템 테스트

- 실제 구현된 시스템과 계획된 사양(specifications)을 서로 비교하는 작업
- 단위, 통합 테스트 후 전체 시스템이 정상적으로 작동하는지 확인
- 시스템 테스트 유형

유형	설명
기능 테스트	<ul style="list-style-type: none"> <li>- 고객의 기능적 요구사항을 중점적으로 테스트한다.</li> <li>- 요구사항에 따른 기능의 구현 여부 및 동작 여부에 대해 테스트를 진행한다.</li> <li>- 테스트 기준은 명세에 따라 확인한다.</li> </ul>
비기능 테스트	<ul style="list-style-type: none"> <li>- 고객의 성능 요구사항을 중점적으로 테스트 한다.</li> <li>- 성능, 신뢰성, 안정성, 유효성, 적합성 등을 확인한다.</li> <li>- 확인하고자 하는 특성에 따라 환경과 도구가 필요하다.</li> </ul>

#### (4) 인수 테스트

- 시스템을 배포하거나 실제 사용할만한 준비가 되었는지에 대해 평가
- 사용자가 요구분석 명세서에 명시된 사항을 모두 충족하는지 판정하고, 시스템이 예상대로 동작하고 있는지를 판정하는 방안을 파악

- 인수 테스트 유형

유형	설명
알파 테스트	<ul style="list-style-type: none"> <li>개발자의 통제 하에 사용자가 개발 환경에서 수행하는 테스트</li> <li>내부에서 진행하는 자체 검사로 실제 사용 환경에서 동작시키며 관련자만 참여한다.</li> </ul>
베타 테스트 (필드 테스트)	<ul style="list-style-type: none"> <li>개발된 소프트웨어를 사용자가 실제 운영환경에서 수행하는 테스트</li> <li>알파 테스트를 거친 후 정식으로 출시하기 전 사용자에게 테스트를 하도록 한다.</li> </ul>
사용자 인수 테스트	<ul style="list-style-type: none"> <li>사용자가 시스템 사용의 적절성을 확인</li> </ul>
운영상 인수 테스트	<ul style="list-style-type: none"> <li>시스템 관리자에 의한 시스템 인수 시 수행</li> <li>백업/복원테스트, 재난복구, 보안 취약성 점검</li> </ul>
계약 인수 테스트	<ul style="list-style-type: none"> <li>계약 상의 인수조건을 준수하는지 확인</li> </ul>
규정 인수 테스트	<ul style="list-style-type: none"> <li>정부의 지침, 법률 또는 안전 규정 등을 준수하는지 확인</li> </ul>

#### 4. 소프트웨어 테스트 기법

##### (1) 프로그램 실행 여부

###### ① 정적 테스트

- 소프트웨어의 실행 없이 소스 코드의 구조를 분석하여 논리적으로 검증하는 테스트
- 경로 분석, 제어 흐름 분석, 데이터 흐름 분석 등을 수행

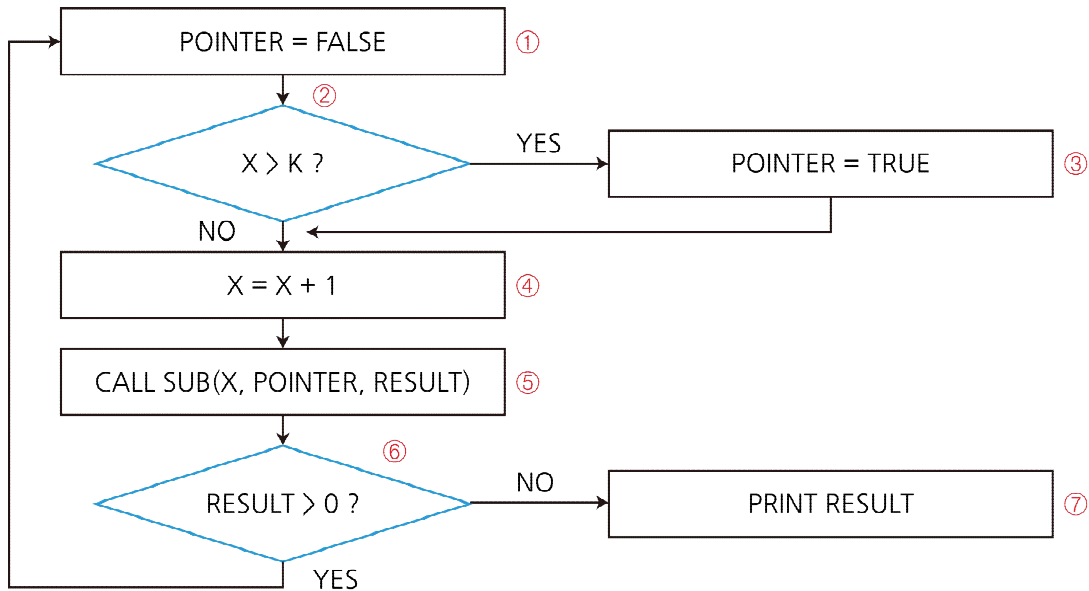
###### ② 동적 테스트

- 소프트웨어를 실행 하여 실제 발생하는 오류를 발견하여 문제를 해결하는 분석 기법
- 다양한 운영 환경에서 소프트웨어를 분석

##### (2) 테스트 기법

###### ① 화이트박스 테스트

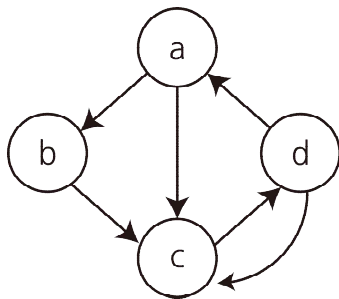
- 소프트웨어의 내부 구조와 동작을 검사하는 테스트 방식
- 소프트웨어 내부 소스코드를 테스트 하는 기법
- 개발자가 내부 소스코드 동작을 추적하여, 동작의 유효성과 코드를 꼼꼼하게 테스트 할 수 있다.
- 개발자 관점의 단위 테스트 방법
- 화이트박스 테스트 기법



기법	설명	검증 방법
문장 검증	프로그램의 모든 문장을 한번 수행하여 검증	1,2,3,4,5,6,7
선택(분기) 검증	선택하는 부분만 검증	1,2,3,4,5,6,7 1,2,4,5,6,1
경로 검증	수행 가능한 모든 경로 검사	1,2,3,4,5,6,7 1,2,3,4,5,6,1 1,2,4,5,6,7 1,2,3,5,6,1
조건 검증	조건이나 반복문 내 조건식을 검사	x>1 or y<10 일 경우 x>1 조건과 y<10 모두 테스트

## ② 기초 경로 검사 (Basic Path Test)

- McCabe가 제안한 것으로 대표적인 화이트박스 테스트 기법
- 계산식 :  $V(G) = E - N + 2$



**③ 블랙박스 테스트**

- 프로그램의 외부 사용자 요구사항 명세를 보면서 테스트, 주로 구현된 기능을 테스트
- 사용자가 소프트웨어에 요구하는 사항과 결과물이 일치하는지 확인
- 사용자 관점의 테스트 방법
- 블랙박스 테스트 기법

기법	설명
동등 분할 기법 (Equivalence Partitioning Testing)	<ul style="list-style-type: none"> <li>- 입력 자료에 초점을 맞춰 테스트 케이스를 만들어 검사하는 방법</li> <li>- 입력 데이터의 영역을 유사한 도메인별로 유효값과 무효값을 그룹핑하여 나누어서 검사</li> </ul>
경계값 분석 (Boundary Value Analysis)	<ul style="list-style-type: none"> <li>- 입력 값의 중간값보다 경계값에서 오류가 발생할 확률이 높다는 점을 이용해 입력 조건의 경계값을 테스트 케이스로 선정한다.</li> </ul>
원인-효과 그래프 검사 (Cause-Effect Graphing Testing)	<ul style="list-style-type: none"> <li>- 입력 데이터 간의 관계와 출력에 영향을 미치는 상황을 체계적으로 분석한 다음 효용성이 높은 테스트 케이스를 선정하여 검사하는 기법</li> </ul>
오류 예측 검사 (Error Guessing)	<ul style="list-style-type: none"> <li>- 과거의 경험이나 테스트의 감각으로 테스트하는 기법</li> </ul>
비교 검사 (Comparison Testing)	<ul style="list-style-type: none"> <li>- 여러 버전의 프로그램에 동일한 테스트 자료를 제공하여 동일한 결과가 출력되는지 테스트하는 기법이다.</li> </ul>

**(3) 테스트에 대한 시각****① 검증(Verification)**

- 소프트웨어의 개발 과정을 테스트
- 올바른 소프트웨어가 만들어지고 있는지 검증

**② 확인(Validation)**

- 완성된 소프트웨어의 결과를 테스트
- 완성된 소프트웨어가 정상적으로 동작하는지 확인
- 완성된 소프트웨어가 사용자의 요구사항을 만족하는지 확인

**(4) 테스트 목적**

- 회복(Recovery) 테스트
  - 시스템에 고의로 실패를 유도하고 시스템이 정상적으로 복구하는지 테스트
- 안전(Security) 테스트
  - 불법적인 소프트웨어가 접근하여 시스템을 파괴하지 못하도록 소스코드 내의 보안적인 결함을 미리 점검하는 테스트
- 강도(Stress) 테스트
  - 시스템에 과다 정보량을 부과하여 과부하 시에도 시스템이 정상적으로 작동되는지를 검증하는 테스트



- 성능(Performance) 테스트
  - 사용자의 이벤트에 시스템이 응답하는 시간, 특정 시간 내에 처리하는 업무량, 사용자 요구에 시스템이 반응하는 속도 등을 테스트
- 구조(Structure) 테스트
  - 시스템의 내부 논리 경로, 소스코드의 복잡도를 평가하는 테스트
- 회귀(Regression) 테스트
  - 변경 또는 수정된 코드에 대하여 새로운 결함 발견 여부를 평가하는 테스트
- 병행(Parallel) 테스트
  - 변경된 시스템과 기존 시스템에 동일한 데이터를 입력 후 결과를 비교하는 테스트

## (5) 테스트 종류

### ① 명세 기반 테스트

- 주어진 명세를 빠짐없이 테스트 케이스로 구현하고 있는지 확인하는 테스트
- 동등 분할, 경계 값 분석, 유한상태 기계기반, 결정 테이블, 정형명세 기반, 유스케이스, 페어와이즈, 직교배열 테스트 등

### ② 구조 기반 테스트

- 소프트웨어 내부 논리 흐름에 따라 테스트 케이스를 작성하고 확인하는 테스트
- 구문 기반, 결정 기반, 조건 기반, 조건결정 기반, 변경조건 결정 기반, 멀티조건 기반 커버리지 테스트 등

### ③ 경험 기반 테스트

- 유사 소프트웨어나 유사 기술 평가에서 테스터의 경험을 토대로 한, 직관과 기술 능력을 기반으로 수행하는 테스트

## 5. 테스트 커버리지

### (1) 테스트 커버리지의 개념

- 주어진 테스트 케이스에 의해 수행되는 소프트웨어의 테스트 범위를 측정하는 테스트 품질 측정 기준
- 테스트의 정확성과 신뢰성을 향상시키는 역할
- 테스트를 얼마나 수행했는지 측정하는 기준

### (2) 테스트 커버리지 유형

#### ① 기능 기반 커버리지

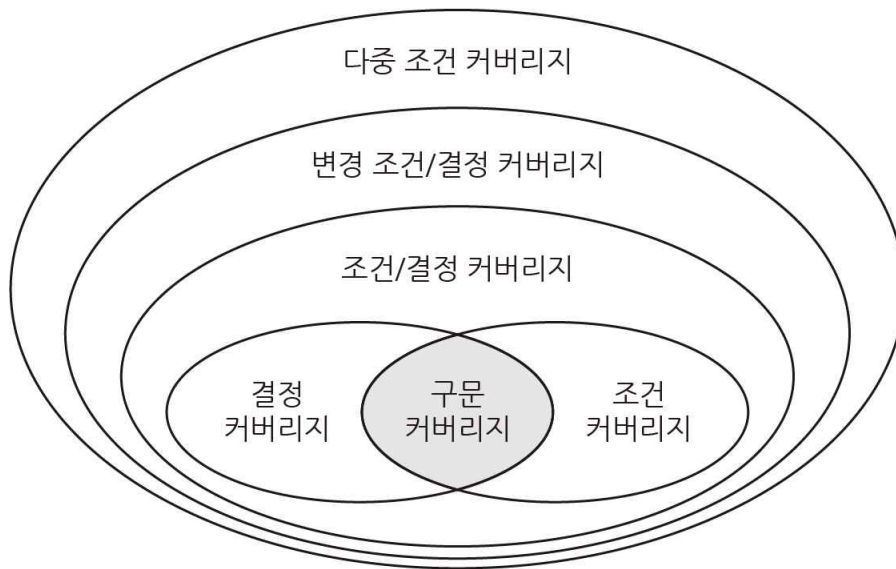
- 테스트 대상 애플리케이션의 전체 기능을 모수로 설정하고, 실제 테스트가 수행된 기능의 수를 측정하는 방법
- 기능 기반 테스트 커버리지는 100% 달성을 목표로 하며, 일반적으로 UI가 많은 시스템의 경우 화면 수를 모수로 사용할 수도 있다.

#### ② 라인 커버리지(Line Coverage)

- 애플리케이션 전체 소스 코드의 Line 수를 모수로 테스트 시나리오가 수행한 소스 코드의 Line 수를 측정하는 방법
- 단위 테스트에서는 이 라인 커버리지를 척도로 삼기도 한다.

### ③ 코드 커버리지(Code Coverage)

- 소프트웨어 테스트 충분성 지표 중 하나로 소스 코드의 구문, 조건, 결정 등의 구조 코드 자체가 얼마나 테스트되었는지를 측정하는 방법



- 구문 커버리지(Statement)
  - 코드 구조 내의 모든 구문에 대해 한 번 이상 수행하는 테스트 커버리지를 말한다.

```
void func(int x) {
    printf("start"); // 1번
    if (x > 0) { // 2번
        printf("process"); // 3번
    }
    printf("end"); // 4번
}
```

// x의 값이 -1일 때, 3번은 수행되지 않는다.  
// 따라서 ( 3 / 4 ) \* 100 = 75%

- 조건 커버리지(Condition)
  - 결정 포인트 내의 모든 개별 조건식에 대해 수행하는 테스트 커버리지를 말한다.

```
void func(int x, int y) {
    if( x > 0 && y < 0 ){
        printf("process");
    }
}
```

x > 0	y < 0	결정포인트
T	F	F
F	T	F

- 개별 조건식이 각각, True와 False 만 만족하면 된다.
- 개별 조건식의 T/F 는 커버되지만 전체 결정포인트의 T/F는 보장 받지 못한다.
- 조건 커버리지를 만족하기 위해서는 ( x=10, y=10 ), ( x=-1, y=-1 )을 넣으면 만족한다.

- 결정 커버리지(Decision)

- 결정 포인트 내의 모든 분기문에 대해 수행하는 테스트 커버리지를 말한다.

<pre>void func(int x, int y) {     if( x &gt; 0 &amp;&amp; y &lt; 0 ){         printf("process");     } }</pre>	x > 0	y < 0	결정포인트
	T	T	T
	T	F	F

- 결정포인트가 각각, True와 False 만 만족하면 된다.
- 개별 조건식에서 오류가 있는 경우 찾지 못할 수 있다.

- 조건/결정 커버리지(Condition/Decision)

- 결정포인트 T/F, 개별조건식 T/F를 가져야 한다.

x > 0	y < 0	결정포인트
T	T	T
F	F	F

- 변경 조건/결정 커버리지(Modified Condition, Decision)

- 모든 결정 포인트 내의 개별 조건식은 적어도 한번 T, F를 가져야 한다.
- 이론적으로 가장 안전한 조합이며 케이스도 줄일 수 있다.

x > 0	y < 0	결정포인트
T	F	F
F	T	F
T	T	T

- 다중 조건 커버리지(Multiple Condition)

- 결정 포인트 내 모든 개별 조건식의 가능한 조합을 100% 보장해야 한다.

x > 0	y < 0	결정포인트
T	T	T
T	F	F
F	T	F
F	F	F

## Section 2. 애플리케이션 통합 테스트

### 1. 결함관리 도구

#### (1) 결함관리 도구의 개념

- 각 단계별 테스트 수행 후 발생한 결함의 재발 방지를 위해, 유사 결함 발견 시 처리 시간 단축을 위해 결함을 추적하고 관리할 수 있게 해주는 도구

#### (2) 결함관리 프로세스

- 에러 발견
- 에러 등록
- 에러 분석
- 결함 확정
- 결함 할당
- 결함 조치
- 결함 조치 검토 및 승인

#### (3) 결함 추이 분석

- 결함 추이 분석
  - 테스트 완료 후 발견된 결함의 결함 관리 측정 지표의 속성 값들을 분석하고, 향후 애플리케이션의 어떤 모듈 또는 컴포넌트에서 결함이 발생할지를 추정하는 작업
- 결함 관리 측정 지표
  - 결함 분포 : 각 애플리케이션 모듈 또는 컴포넌트의 특정 속성에 해당하는 결함의 수를 측정하여 결함의 분포를 분석할 수 있다.
  - 결함 추세 : 테스트 진행 시간의 흐름에 따른 결함의 수를 측정하여 결함 추세를 분석할 수 있다.
  - 결함 에이징 : 등록된 결함에 대해 특정한 결함 상태의 지속 시간을 측정하여 분석할 수 있다.

#### (4) 결함의 식별

- 단계별 결함 유입 분류
  - 기획시 유입되는 결함
  - 설계 시 유입되는 결함
  - 코딩 시 유입되는 결함
  - 테스트 부족으로 유입되는 결함
- 결함 심각도별 분류
  - 치명적 결함(Critical)
  - 주요 결함(Major)
  - 보통 결함(Normal)
  - 낮은 결함(Low, Minor)
- 결함 우선 순위
  - 결정적(Critical) : 24시간 안에 즉시 수정, 전체 기능이 동작하지 않고, 더 이상 테스트도 진행할 수 없다.
  - 높음(High) : 일반적인 결함으로 인해 하나의 기능을 사용할 수 없다.
  - 보통(Medium) : 어떤 기능이 기대치에 못 미칠 경우
  - 낮음(Low) : 종료 기준에 따라 수정하지 않아도 되는 경우

### (5) 결함 관리 항목

- 결함 내용, 결함 ID, 결함 유형, 발견일, 심각도, 우선순위, 시정 조치 예정일, 수정 담당자, 재테스트 결과, 종료일

### (6) 결함 관리 도구 종류

- S/W 테스트 관리
  - TestLink, GanttProject, OpenProj, Redmine
- 결함 추적 관리
  - Mantis, Bugzilla, Trac

## 2. 테스트 자동화 도구

### (1) 테스트 자동화 도구의 개념

- 테스트 도구를 활용하여 반복적인 테스트 작업을 스크립트 형태로 구현함으로써, 테스트 시간 단축과 인력 투입 비용을 최소화하는 한편, 쉽고 효율적인 테스트를 수행할 수 있는 방법

### (2) 테스트 자동화 도구의 장/단점

- 장점
  - 반복되는 테스트 데이터 재입력 작업의 자동화
  - 사용자 요구 기능의 일관성 검증에 유리
  - 테스트 결과 값에 대한 객관적인 평가 기준 제공
  - 테스트 결과의 통계 작업과 그래프 등 다양한 표시 형태 제공
  - UI가 없는 서비스의 경우에도 정밀한 테스트 가능
- 단점
  - 도구 도입 후 도구 사용 방법에 대한 교육 및 학습이 필요
  - 도구를 프로세스 단계별로 적용하기 위한 시간, 비용, 노력이 필요
  - 상용 도구의 경우 고가, 유지 관리 비용이 높아 추가 투자가 필요

### (3) 테스트 자동화 도구 유형

#### ① 정적 분석 도구(Static Analysis Tools)

- 정적 분석 도구는 만들어진 애플리케이션을 실행하지 않고 분석하는 방법
- 대부분의 경우 소스 코드에 대한 코딩 표준, 코딩 스타일, 코드 복잡도 및 남은 결함을 발견하기 위하여 사용
- 테스트를 수행하는 사람이 작성된 소스 코드에 대한 이해를 바탕으로 도구를 이용해서 분석하는 것
- 종류 : pmd, SonarQube, cppcheck, checkstyle 등

#### ② 테스트 실행 도구(Test Execution Tools)

- 테스트를 위해 작성된 스크립트를 실행
- 작성된 스크립트는 각 스크립트마다 특정 데이터와 테스트 수행 방법을 포함
- 테스트 실행 도구 방식 (데이터 주도 접근 방식, 키워드 주도 접근 방식)

### ③ 성능 테스트 도구(Performance Test Tools)

- 애플리케이션의 처리량, 응답 시간, 경과 시간, 자원 사용률에 대해 가상의 사용자를 생성하고 테스트를 수행함으로써 성능 목표를 달성하였는지를 확인하는 도구

### ④ 테스트 통제 도구(Test Control Tools)

- 테스트 관리 도구 : 테스트 계획 및 관리
- 형상 관리 도구 : 테스트 수행에 필요한 데이터와 도구를 관리
- 결함 추적/관리 도구 : 테스트에서 발생한 결함에 대해 관리하거나 협업을 지원

### ⑤ 테스트 장치(Test Harness)

- 테스트 장치의 개념
  - 애플리케이션 컴포넌트 및 모듈을 테스트하는 환경의 일부분
  - 테스트를 지원하기 위한 코드와 데이터
  - 단위 또는 모듈 테스트에 사용하기 위해 코드 개발자가 작성
- 테스트 장치 구성요소

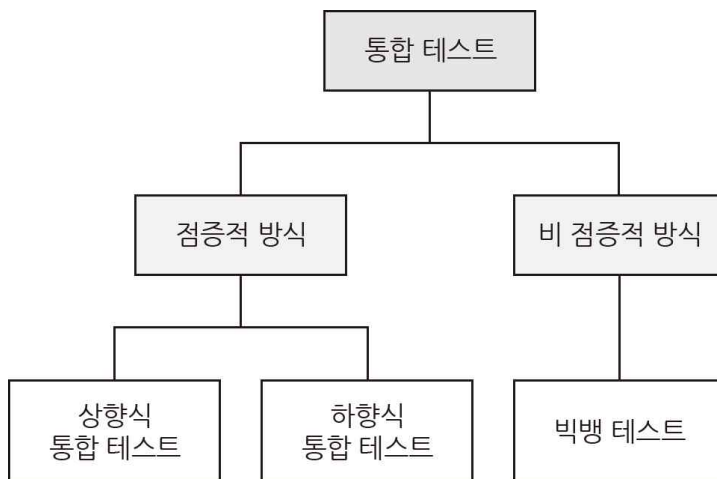
구성요소	설명
테스트 드라이버 (Test Driver)	- 테스트 대상 하위 모듈을 호출하고, 파라미터를 전달하고, 모듈 테스트 수행 후의 결과를 도출하는 등 상향식 테스트에 필요하다.
테스트 스텝 (Test Stub)	- 제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구로 하향식 테스트에 필요하다.
테스트 슈트 (Test Suites)	- 테스트 대상 컴포넌트나 모듈, 시스템에 사용되는 테스트 케이스의 집합을 말한다.
테스트 케이스 (Test Case)	- 입력 값, 실행 조건, 기대 결과 등의 집합을 말한다.
테스트 스크립트 (Test Script)	- 자동화된 테스트 실행 절차에 대한 명세를 말한다.
Mock 오브젝트 (Mock Object)	- 사용자의 행위를 조건부로 사전에 입력해 두면, 그 상황에 예정된 행위를 수행하는 객체를 말한다.

### 3. 통합 테스트

#### (1) 통합 테스트의 개념

- 소프트웨어 각 모듈 간의 인터페이스 관련 오류 및 결함을 찾아내기 위한 체계적인 테스트 기법
- 단위 테스트가 끝난 모듈 또는 컴포넌트 단위의 프로그램이 설계 단계에서 제시한 애플리케이션과 동일한 구조와 기능으로 구현된 것인지를 확인하는 것
- 수행 방법으로는 비점증적 방식(빅뱅), 점증적 방식(상향식, 하향식)이 있다.

#### (2) 통합 테스트 수행 방법의 분류



##### ① 하향식 통합 테스트(Top Down)

- 메인 제어 모듈로부터 아래 방향으로 제어의 경로를 따라 이동하면서 하향식으로 통합하면서 테스트 진행
- '깊이-우선' 또는 '너비-우선' 방식으로 통합
- 아직 개발되지 않은 하위 모듈은 더미 모듈인 스텝(Stub)을 개발하여 테스트 진행
- 장점
  - 장애 위치 파악이 쉬움
  - 중요 모듈은 먼저 테스트 수행
- 단점
  - 많은 스텝이 필요함
  - 하위 모듈의 불충분한 테스트

##### ② 상향식 통합 테스트(Bottom Up)

- 최하위 레벨의 모듈 부터 위쪽 방향으로 제어의 경로를 따라 통합하면서 테스트 진행
- 하위 모듈을 클러스터(Cluster)로 결합하면서 위쪽 방향으로 진행
- 아직 개발되지 않은 하위 모듈은 더미 모듈인 드라이버(Driver)를 개발하여 테스트 진행
- 장점
  - 장애 위치 파악 쉬움
  - 모든 모듈 개발을 위한 시간 낭비가 불필요
- 단점
  - 중요 모듈들이 마지막에 테스트될 가능성이 높음

### ③ 빅뱅 테스트

- 모든 구성 요소들을 한꺼번에 통합하여 테스트
- 소규모 시스템에 편리한 테스트 방식
- 장점
  - 단시간에 테스트를 할 수 있다.
- 단점
  - 장애가 일어난 위치를 파악하기 어렵다.
  - 모든 모듈들이 설계된 후에 시작 할 수 있기 때문에, 테스트 시간이 적어진다.

### ④ 백본 테스트

- 샌드위치 테스트
- 상향식과 하향식의 장점을 이용하는 방식
- 드라이버/스텝을 필요에 따라 만들어 사용
- 대규모 프로젝트에 사용하는 방식
- 비용이 많이 들어간다.

### (3) 통합 테스트 수행 순서

- 통합 테스트 계획서를 검토
- 통합 테스트를 수행할 테스트 환경을 준비
- 통합 테스트 케이스 및 시나리오를 검토
- 통합 모듈 및 인터페이스가 요구사항을 충족하는지 테스트를 수행
- 통합 테스트 결과를 기록



## Section 3. 애플리케이션 성능 개선

### 1. 애플리케이션 성능 저하 원인

#### (1) 데이터베이스 관련 성능 저하

- 데이터베이스 락(DB Lock)
  - 대량의 데이터 조회, 과도한 업데이트시 발생하며, Lock 해제시까지 대기하거나 타임아웃 된다.
- 불필요한 패치(DB Fetch)
  - 결과 세트에서 커서를 옮기는 작업이 빈번할 때 발생한다.
- 연결 누수(Connection Leak)
  - DB Connection 사용 후 반환하지 않을 경우 발생한다.
  - Connection 의 Pool 크기가 너무 작거나 크게 설정된 경우 발생한다.

#### (2) 내부 로직으로 인한 성능 저하 원인

- 파일 관련 오류
  - 대량의 파일을 업로드 하거나 다운로드 할 때 발생한다.
- 코드 오류
  - 코드 오류로 무한 반복등이 수행될 때 발생한다.

#### (3) 외부 호출로 인한 성능 저하

- 외부서버와 인터페이스 시 장시간 수행되거나, 타임아웃이 일어나 성능 저하가 발생한다.

### 2. 애플리케이션 성능 분석

#### (1) 애플리케이션 성능 분석 지표

- 처리량(Throughput)
  - 일정 시간 내에 애플리케이션이 처리하는 일의 양
- 응답 시간(Response Time)
  - 애플리케이션에 요청을 전달한 시간부터 응답이 도착할 때까지 걸린 시간
- 반환/경과 시간(Turn Around Time)
  - 애플리케이션에 요청을 전달한 시간부터 처리가 완료될 때까지 걸린 시간
- 자원 사용률(Resource Usage)
  - 애플리케이션이 작업을 처리하는 동안의 CPU 사용량, 메모리 사용량, 네트워크 사용량 등 자원 사용률

#### (2) 성능 분석 도구

- JMeter
  - HTTP, FTP등 다양한 프로토콜을 지원하는 부하 테스트 도구
- LoadUI
  - 서버 모니터링, Drag&Drop 등 사용자 편의성이 강화된 도구
  - HTTP, JDBC 등 다양한 프로토콜 지원
- OpenSTA
  - HTTP, HPPTS 프로토콜에 대한 부하 테스트 및 생산품 모니터링 도구

### (3) 모니터링 도구

- Scouter
  - 단일 뷰 통합/실시간 모니터링
- NMon
  - 리눅스 서버 자원에 대한 모니터링 도구
- Zabbix
  - 웹기반 서버, 서비스, 애플리케이션 모니터링 도구
- Jeniffer
  - 애플리케이션에서 서버로 유입되는 트랜잭션 수량, 처리시간, 응답시간, 자원 활용율 등을 모니터링

## 3. 정형 기술 검토회의(FTR, Formal Technical Review)

### (1) FTR의 개념

- 소프트웨어 엔지니어가 수행하는 소프트웨어 품질보증 활동
- 개발단계에서 제작되는 문서나 프로그램의 문제점을 찾고, 문제해결을 촉구하는 일반적인 용어
- S/W 개발 산출물을 대상으로 오류를 발견하기 위한 공식적인 활동

### (2) FTR의 목적

- 산출물 요구사항 일치여부
- 소프트웨어가 미리 정한 기준에 따라 표현 되었는가를 확인
- 소프트웨어의 표현에 대한 기능, 논리적 오류
- 프로젝트를 보다 관리하기 쉽게 만든다.

### (3) 검토지침

- 제작자가 아닌 제품의 검토에만 집중한다.
- 문제 영역을 명확히 표현한다.
- 제기된 모든 문제를 바로 해결하고자 하지 말 것
- 검토자들은 사전에 작성한 메모들을 공유한다.
- 논쟁이나 반박을 제한한다.
- 의제를 정하고 그 범위를 유지한다.
- 참가자의 수를 제한하고, 사전 준비를 철저히 하도록 강요한다.
- 자원과 시간 일정을 할당한다.
- 모든 검토자에게 의미있는 교육을 행한다.
- 검토의 과정과 결과를 재검토한다.

## 4. 소스코드 품질 분석

### (1) 동료 검토(Peer Review)

- 2~3명이 진행하는 리뷰의 형태
- 작성자가 코드를 설명하고 이해 관계자들이 설명을 들으면서 결함을 발견하는 형태로 진행하는 기법

### (2) 워크스루(Walkthrough)

- 계획된 개발자 검토 회의
- 검토 자료를 회의 전에 배포해서 사전검토 한 후 짧은 시간 동안 회의를 진행하는 형태
- 리뷰를 통해 오류를 검출하고 문서화 하는 기법

### (3) 인스펙션(Inspection)

- 공식적 검사 회의
- 작업자 외 다른 전문가가 검사하는 가장 공식적인 리뷰 기법
- 다른 전문가 또는 팀이 검사하여 오류를 찾아내는 공식적 검토 기법

## 5. 소스코드 품질 분석 도구

### (1) 소스코드 품질 분석 도구의 개념

- 코딩을 하면서 발생하는 문제를 해결하기 위해 사용하는 도구
- 소스코드의 코딩 스타일, 코딩 표준, 코드의 복잡도, 메모리 누수 현상, 스레드 결함과 같은 소스코드로 인해 발생하는 문제를 찾기 위해 사용

### (2) 소스코드 품질 분석 도구 분류

- 정적 분석 도구
  - 프로그램을 실행하지 않고 소프트웨어를 분석하는 방법
  - 소스 코드의 결함 및 취약성을 찾기 위한 도구
  - 소프트웨어 코드내에 잠재된 오류, 버그, 보안취약점 등의 각종 결함들을 소스코드 분석을 통하여 탐지
- 동적 분석 도구
  - 프로그램을 실행하여 코드에 존재하는 메모리 누수나, 스레드의 결함 등을 발견

### (3) 소스코드 품질 분석 도구 종류

구분	도구명	설명
정적 분석 도구	PMD	주로 Java에서 사용하지만, Javascript, PLSQL, XML 등의 언어도 지원
	checkstyle	자바 코드에 대한 소스 코드 표준 준수 검사, 다양한 개발 도구에 통합 가능
	SonarQube	중복코드, 복잡도, 코딩 설계 등을 분석하는 소스 분석 통합 플랫폼
	cppcheck	C, C++코드에 대한 메모리 누수, 오버플로우 등 문제 분석
	ccm	다양한 언어의 코드 복잡도 분석 도구
	cobertura	자바 언어의 소스 코드 복잡도 분석 및 테스트 커버리지 측정
동적 분석 도구	Avalanche	프로그램에 대한 결함 및 취약점 분석
	Valgrind	프로그램 내 존재하는 메모리 및 스레드 결함 등 분석

## 6. 애플리케이션 성능 개선하기

### (1) 코드 최적화의 개념

- 주어진 코드에 대해 같은 작업을 수행 하면서, 실행 시간을 줄이거나 메모리를 줄이는 것
- 기존의 방법보다 계산 횟수를 줄이거나 실행 시간을 더 짧고, 기억용량을 더 적게 코드를 작성
- 알고리즘을 개선하거나, 병목 현상을 제거
- 소스코드를 가지고 협업하거나, 유지운명을 위해 소스코드를 다른 사람들이 읽기 쉽게 작성
- 소스코드 리팩토링을 통해서 코드 스멜(Code Smell)을 제거하고 품질 좋은 코드를 작성

### (2) 코드 스멜(Code Smell)

- 같은 코드가 여러 곳에 존재하는 중복된 코드로 코드 스멜(Code Smell)이 발생
- 클래스, 메소드, 함수, 프로시저의 길이가 매우 길어져서 발생
- 다른 클래스의 메소드들을 너무 많이 사용하는 기능 의존으로 발생
- 다른 클래스의 구현에 세밀하게 의존하는 클래스로서 부적절한 관계를 형성
- 상속을 거부하는 현상으로 부모 클래스의 규약을 지키지 않은 채, 메소드 오버라이드(Method Override)를 하는 경우에 발생

용어	설명
스파게티 코드 (spaghetti code)	<ul style="list-style-type: none"> <li>- 소스 코드가 복잡하게 얽힌 모습을 스파게티의 면발에 비유한 표현이다.</li> <li>- 스파게티 코드는 정상적으로 작동하지만, 사람이 코드를 읽으면서 그 코드의 작동을 파악하기는 어렵다.</li> <li>- GOTO 문을 지나치게 많이 사용하거나, 프로그램을 구조적으로 만들지 않는 경우에 만들어지기 쉽다.</li> </ul>
외계인 코드	<ul style="list-style-type: none"> <li>- 아주 오래되거나 참고 문서 또는 개발자가 없어 유지보수 작업이 어려운 프로그램 코드이다.</li> </ul>

### (3) 리팩토링

- 외부 동작을 바꾸지 않으면서 내부 구조를 개선하는 방법
- 코드가 작성된 후에 디자인을 개선하는 작업
- 모든 것을 미리 생각하기보다는 개발을 하면서 지속적으로 좋은 디자인을 찾는다.
- 이미 작성한 소스코드에서 구현된 일련의 기능을 변경없이, 코드의 가독성과 유지보수성을 높이기 위해 내부 구조를 변경하는 것
- 주요 리팩토링 기법으로는 메서드 정리, 객체 간 기능 이동, 이름 변경, 추측성 일반화가 있다.

### (4) 클린코드

#### ① 클린코드의 개념

- 의존성을 최소로 하고 사람이 이해할 수 있는 가독성, 목적성이 뛰어난 명확한 코드
- 의존성 최소화, 단일 책임의 원칙, 버그유입 최소화, 가독성 향상, 중복코드 최소화, 변경용이, 작은 코드 등의 특징

## ② 클린코드 구현 방법

- 클래스명, 메서드명, 변수명은 명사를 사용하여 의미가 있는 이름을 짓는다.
- 불필요한 주석을 제거하고 의도를 명확히 기술한 주석을 작성
- 리팩토링을 통해 점진적인 개선을 통해 코드의 복잡도를 낮게 한다.
- 코드는 가급적 기능별로 간단하게 작성
- 코드의 중복을 최소화
- 누구든지 코드를 쉽게 읽을 수 있도록 작성
- 다른 모듈에 미치는 영향을 최소화하여 작성

## ③ 클린코드 작성 원칙

작성원칙	설명
가독성	<ul style="list-style-type: none"> <li>- 누구든지 코드를 쉽게 읽을 수 있도록 작성한다.</li> <li>- 코드 작성 시 이해하기 쉬운 용어를 사용하고, 들여쓰기 등을 이용한다.</li> </ul>
단순성	<ul style="list-style-type: none"> <li>- 코드를 간단하게 작성한다.</li> <li>- 한 번에 한 가지를 처리하도록 코드를 작성하고 클래스/메소드/함수 등을 최소 단위로 분리한다.</li> </ul>
의존성 배제	<ul style="list-style-type: none"> <li>- 코드가 다른 모듈에 미치는 영향을 최소화한다.</li> <li>- 코드 변경 시 다른 부분에 영향이 없도록 작성한다.</li> </ul>
중복성 최소화	<ul style="list-style-type: none"> <li>- 코드의 중복을 최소화한다.</li> <li>- 중복된 코드는 삭제하고 공통된 코드를 사용한다.</li> </ul>
추상화	<ul style="list-style-type: none"> <li>- 상위 객체는 간략하게 기능적 특성을 나타내고, 상세 내용은 하위 객체에서 구현한다.</li> </ul>

## 09 소프트웨어 유지보수

### Section 1. 소프트웨어 유지보수

#### 1. 소프트웨어 유지보수

##### (1) 소프트웨어 유지보수의 개념

- 개발 완료 시점부터 폐기될 때까지, 지속적으로 수행하는 작업
- 소프트웨어의 수명을 연장하기 위한 활동
- 소프트웨어 생명주기 동안 가장 많은 비용이 소모되는 단계
- 기능개선 / 하자보수 / 환경적응 / 예방조치가 목적이다.

##### (2) 유지보수의 중요성

- 소프트웨어 예산에서 유지, 보수비용의 비중이 증가하고 있다.
- 신규 프로젝트보다는 기존 소프트웨어 개선에 더 많이 투자할 것이라는 전망
- 소프트웨어 기술발전이 용역 개발보다 패키지 구매쪽으로 변화

##### (3) 유지보수가 어려운 이유

- 업무 프로세스와 구축된 시스템을 이해해야 함
- 유지보수 계약이 개발과 계획과 별개인 경우 시간이 지나면서 소프트웨어 구조와 가독성이 떨어짐

#### 2. 유지보수의 구분

##### (1) 수정 보수(Corrective Maintenance)

- 소프트웨어 구축 시 테스트 단계에 미쳐 발견하지 못한 잠재적인 오류를 찾아 수정한다.
- 수리 보수, 수정 보수, 정정 보수, 하자 보수라고도 한다.

##### (2) 적응 보수(Adaptive Maintenance)

- 운영체제, 하드웨어와 같은 프로그램 환경변화에 맞추기 위해 수행하는 유지보수

##### (3) 향상 보수(Perfective Maintenance)

- 기존 기능과 다른 새로운 기능을 추가하거나, 기존 기능을 개선
- 소프트웨어 확장 및 리모델링
- 유지보수 활동 중 가장 자원이 많이 소모되는 활동

##### (4) 예방 보수(Preventive Maintenance)

- 장래에 유지보수성 또는 신뢰성을 보장하기 위해 선제적으로 하는 유지보수
- 소프트웨어의 잠재적인 오류발생에 대비하여 미리 예방수단을 강구해 두는 유지보수

### 3. 유지보수 비용 예측 방법

- 주먹구구식 방법
- Belady와 Lehman의 방법
- COCOMO 방법
- SMI(Software Maturity Index, 소프트웨어 성숙 색인)

### 4. 유지보수 관련 용어

- 레거시 시스템(legacy system)
  - 낡은 기술이나 방법론, 컴퓨터 시스템, 소프트웨어 등을 말한다.
  - 더이상 쓰이지 않더라도 현대의 기술에 영향을 주는 경우도 포함한다.
- 외계인 코드(Alien Code)
  - 아주 오래되거나 참고 문서 또는 개발자가 없어 유지보수 작업이 어려운 프로그램 코드
  - 프로그램 문서화(Documentation)을 통해 외계인 코드를 방지 가능
- 스파게티 코드(Spaghetti Code)
  - 복잡한 프로그래밍 소스 코드
  - 작동 자체는 제대로 하거나 하는 것처럼 보이지만, 추후 유지보수가 매우 어려워진다.

## 10 제품 소프트웨어 패키징

### Section 1. 국제 표준 제품 품질 특성

#### 1. 제품 품질 국제 표준

##### (1) 제품 품질 국제 표준의 개념

- 소프트웨어 개발 공정 각 단계에서 산출되는 제품이 사용자 요구를 만족하는지 검증하기 위한 국제 표준

##### (2) 소프트웨어 품질 관련 국제 표준

표준	세부 내용	설명
ISO/IEC 9126	<ul style="list-style-type: none"> <li>- 9126-1(품질 모델)</li> <li>- 9126-2(외부 품질)</li> <li>- 9126-3(내부 품질)</li> <li>- 9126-4(사용 품질)</li> </ul>	<ul style="list-style-type: none"> <li>- 품질 특성 및 측정 기준 제시</li> <li>- 기능성, 신뢰성, 사용성, 효율성, 유지보수 용이성, 이식성</li> </ul>
ISO/IEC 14598	<ul style="list-style-type: none"> <li>- 14598-1(개요)</li> <li>- 14598-2(계획과 관리)</li> <li>- 14598-3(개발자용 프로세스)</li> <li>- 14598-4(구매자용 프로세스)</li> <li>- 14598-5(평가자용 프로세스)</li> <li>- 14598-6(평가 모듈)</li> </ul>	<ul style="list-style-type: none"> <li>- 소프트웨어 제품평가에 대한 국제적인 표준으로 ISO 9126의 사용을 위한 절차와 기본 상황 및 소프트웨어 평가 프로세스에 대한 표준 규정한 것</li> <li>- 반복성, 공정성, 객관성, 재현성</li> </ul>
ISO/IEC 12119	소프트웨어 패키지 <ul style="list-style-type: none"> <li>- 제품 설명서</li> <li>- 사용자 문서</li> <li>- 프로그램과 데이터</li> </ul>	<ul style="list-style-type: none"> <li>- 패키지 SW 품질 요구사항 및 테스트</li> </ul>
ISO/IEC 25000	<ul style="list-style-type: none"> <li>- 2500n(9126-1)</li> <li>- 2501n(9126-2)</li> <li>- 2502n(9126-3)</li> <li>- 2503n(9126-4)</li> <li>- 2504n(9126-5)</li> </ul>	<ul style="list-style-type: none"> <li>- S/W 품질평가 통합모델</li> <li>- ISO 9126와 소프트웨어 평가절차 모델 ISO 14598을 통합</li> </ul>



## ① ISO/IEC 9126 의 소프트웨어 품질 특성

품질 특성	설명
기능성 (Functionality)	<ul style="list-style-type: none"> <li>- 소프트웨어가 특정 조건에서 사용될 때, 명시된 요구와 내재된 요구를 만족하는 기능에 대한 소프트웨어 제품의 능력</li> <li>- 부특성 : 적합성, 정확성, 상호 운용성, 보안성, 준수성</li> </ul>
신뢰성 (Reliability)	<ul style="list-style-type: none"> <li>- 소프트웨어가 명세된 조건에서 사용될 때, 성능 수준을 유지할 수 있는 소프트웨어 제품의 능력</li> <li>- 부특성 : 성숙성, 결함 허용성, 복구성</li> </ul>
사용성 (Usability)	<ul style="list-style-type: none"> <li>- 사용자에게 의해 이해 이해되고, 학습되고, 사용되고, 선호될 수 있는 소프트웨어 제품의 능력</li> <li>- 부특성 : 이해성, 학습성, 운영성, 선호도, 준수성</li> </ul>
효율성 (Efficiency)	<ul style="list-style-type: none"> <li>- 사용되는 자원의 양에 따라 요구된 성능을 제공하는 소프트웨어 제품의 능력</li> <li>- 부특성 : 시간 반응성, 자원 활용성, 준수성</li> </ul>
유지보수성 (Maintainability)	<ul style="list-style-type: none"> <li>- 소프트웨어 제품이 변경되는 능력</li> <li>- 소프트웨어의 수정, 개선 등이 포함된다.</li> <li>- 부특성 : 분석성, 변경성, 안정성, 시험성, 준수성</li> </ul>
이식성 (Portability)	<ul style="list-style-type: none"> <li>- 현재 환경에서 다른 환경으로 이전될 수 있는 소프트웨어 제품의 능력</li> <li>- 부특성 : 적응성, 설치성, 공존성, 대체성, 준수성</li> </ul>

## ② ISO/IEC 14598 평가 특성

평가 특성	설명
반복성 (Repetability)	- 특정 제품에 대해 동일 평가자가 동일 사양에 대해 평가 했을 때 동일한 결과가 나와야 한다.
재현성 (Reproducibility)	- 특정 제품에 대해 다른 평가자가 동일 사양에 대해 평가 했을 때 동일하다고 여길 수 있는 결과가 나와야 한다.
공정성 (Impartiality)	- 평가가 특정 결과에 편향되지 않아야 한다.
객관성 (Efficiency)	- 평가 결과가 평가자의 감정이나 의견에 의해 영향을 받지 않아야 한다.

## ③ ISO/IEC 12119 구성요소

구성요소	설명
제품설명서	<ul style="list-style-type: none"> <li>- 소프트웨어 패키지의 속성을 설명하는 문서</li> <li>- 제품 구입에 앞서 제품이 필요한지를 평가할 수 있는 정보 제공</li> </ul>
사용자문서	- 인쇄 또는 비 인쇄 형태의 사용 가능한 전체 문서들의 집합
실행프로그램	- 요구사항이 명확하게 정의된 대상

## ④ ISO/IEC 25000

- ISO 9126과 ISO 14598, ISO 12119, ISO 15288 표준을 5개 영역 중심으로 통합한 소프트웨어 평가 모델 국제 표준

## 2. 프로세스 품질 국제 표준

## (1) 프로세스 품질 국제 표준의 개념

- 소프트웨어 개발 프로세스 등 소프트웨어 관련 업체의 프로세스 관리능력을 평가하고 프로세스를 개선하는데 활용할 수 있는 표준
- 미국이 1987년에 S/W프로세스 성숙도(maturity)를 평가하기 위한 기준인 SW-CMM(Software Capability Maturity Model)을 제정하여 보급해 온 이래 국제표준화기구(ISO)도 관련 표준의 제정을 추진해 오고 있다.

## (2) 국제 프로세스 품질 표준

표준	관점
ISO/IEC 9001	- 조직의 품질 경영 및 품질 보증
ISO/IEC 12207	- 소프트웨어 개발 관련 생명주기
ISO/IEC 15504 (SPICE)	- 소프트웨어 개발 관련해 선정된 프로세스 평가 모델
CMM	- 조직의 소프트웨어 개발 관련 전체 프로세스 평가
CMMI	- 다양한 CMM 모델을 통합한 프로세스 개선 프레임워크

## ① ISO/IEC 12207 구성

생명주기 프로세스	세부 프로세스
기본 생명주기 프로세스	- 획득, 공급, 개발, 운영, 유지보수
지원 생명주기 프로세스	- 문서화, 형상관리, 품질보증, 검증, 확인, 합동검토, 감사, 문제해결
조직 생명주기 프로세스	- 관리, 기반구조, 개선, 교육훈련

## ② ISO/IEC 15504(SPICE)

- ISO에서 표준으로 지정된 프로세스 수행능력 평가 표준 프레임워크
- SPICE 프로세스 능력 수준

수준	단계	설명
0	불안정 단계(Incomplete)	미구현 또는 목표 미달성
1	수행 단계(Performed)	프로세스 수행 및 목적 달성
2	관리 단계(Managed)	프로세스 수행 계획 및 관리
3	확립 단계(Established)	표준 프로세스의 사용
4	예측 단계(Predictable)	프로세스의 정량적 이해 및 통제
5	최적화 단계(Optimizing)	프로세스의 지속적인 개선

### ③ CMM(Capability Maturity Model)

- 소프트웨어 개발 업체들의 업무능력평가 기준을 세우기 위한 평가 모형
- 1991년 미국 국방부의 의뢰를 받아 카네기멜론 대학이 만든 평가 모델
- 소프트웨어 개발능력 측정 기준과 소프트웨어 개발 조직의 성숙도 수준을 평가
- CMM 성숙도 5단계

수준	단계	설명
1	초기 단계(Initial)	- 소프트웨어를 개발하고 있으나 관리는 하고 있지 않은 상태 - 프로세스의 성과를 예측할 수 없는 상태
2	반복 단계(Repeatable)	- 이전의 성공적인 프로젝트의 프로세스를 반복하고 있는 상태 - 같은 것을 반복적으로 실행하며 어느 정도의 통계적 관리가 가능한 상태
3	정의 단계(Defined)	- 프로세스 작업이 잘 정의/이해되고, 프로세스 데이터에 의한 프로젝트 관리도 실행하고 있는 상태 - 프로세스의 기초가 정립되어 계속 진보되고 있는 상태
4	관리 단계(Managed)	- 프로세스 성과를 측정/분석하여 개선시키고, 이를 바탕으로 관리하고 있는 상태 - 정량적 프로세스 관리, 소프트웨어 품질 관리
5	최적화 단계(Optimizing)	- 질적, 양적으로 지속적인 개선이 이루어지고 있는 상태

### ④ CMMi(Capability Maturity Model Integration)

- 시스템과 소프트웨어 영역을 하나의 프로세스 개선 톨로 통합시켜 기업의 프로세스 개선 활동에 광범위한 적용성을 제공하는 모델
- 기존 CMM 에 프로젝트 관리(PM), 프로큐어먼트(Procurement), 시스템 엔지니어링(SE) 등의 요소를 추가한다.
- CMMi 성숙도 5단계

수준	단계	설명
1	초기 단계(Initial)	- 구조화된 프로세스를 갖고 있지 않는 조직
2	관리 단계(Managed)	- 기본적인 프로세스를 갖고 있는 조직 - 기본 프로세스에 따라 업무가 수행되고 기본적인 관리 활동들로부터 구체적인 특정 영역으로 프로세스의 체계가 확대 발전하는 조직
3	정의 단계(Defined)	- 조직 차원의 표준 프로세스를 보유하고 있으며 프로젝트를 수행할 경우 프로젝트의 특성에 따라 적절하게 조정하여 사용
4	정량적 관리 단계 (Quantitatively Managed)	- 프로세스들을 통계적이고 정량적으로 관리하는 조직
5	최적화 단계(Optimizing)	- 질적, 양적으로 지속적인 개선이 이루어지고 있는 상태

## Section 2. 제품 소프트웨어 패키징

### 1. 애플리케이션 패키징

#### (1) 애플리케이션 패키징의 개념

- 개발이 완료된 제품 소프트웨어를 고객에게 전달하기 위한 형태로 패키징하고, 설치와 사용에 필요한 제반 절차 및 환경 등 전체 내용을 포함하는 매뉴얼을 작성하는 활동
- 모듈별로 생성한 실행 파일들을 묶어 배포용 설치 파일을 만드는 것

#### (2) 애플리케이션 패키징 특징

- 애플리케이션 패키징은 개발자가 아닌 사용자 중심으로 진행
- 신규 및 변경 개발 소스를 식별하고, 이를 모듈화하여 상용 제품으로 패키징
- 고객의 편의성을 위해, 신규/변경 이력을 확인하고, 이를 버전 관리 및 릴리즈 노트를 통해 지속적으로 관리
- 사용자의 실행 환경을 이해하고, 범용 환경에서 사용이 가능하도록 일반적인 배포 형태로 분류하여 패키징이 진행

#### (3) 사용자 중심의 패키징 작업

- 사용자 실행 환경의 이해
  - 고객 편의성을 위해 사용자 실행 환경을 우선 고려하여 패키징을 진행
  - OS, 실행 환경, 시스템 사양 및 고객의 사용 방법까지 상세 분류하여 실행 환경을 사전 정의
  - 만약 여러 가지 실행 환경이 나오게 된다면 해당 경우에 맞는 배포본을 분류하여 패키징 작업을 여러 번 수행할 수도 있다.
- 사용자 관점에서의 패키징 고려 사항
  - 사용자의 시스템 환경인 OS, CPU, 메모리, 저장공간, 네트워크 사양 등의 수행 최소 환경을 정의
  - 사용자가 직관적으로 확인할 수 있는 UI(User Interface)를 제공하고, 매뉴얼과 일치시켜 패키징 작업
  - 제품 소프트웨어는 하드웨어와 함께 통합 적용될 수 있도록, 패키징은 Managed Service 형태로 제공되는 것이 좋다.
  - 고객 편의성을 위해 안정적 배포가 될 수 있게 한다.
  - 다양한 사용자 요구사항을 반영하기 위해 항상 패키징의 변경 및 개선 관리를 고려하여 패키징 배포

#### (4) 애플리케이션 패키징 수행 순서

- 기능 식별
  - 개발 소스의 목적 및 기능을 식별
  - 입출력 데이터, 전체적인 기능과 데이터 흐름을 식별
- 모듈화
  - 모듈 단위 분류 및 모듈화를 순서에 맞게 진행
  - 수행을 위한 기능 단위 및 서비스 분류, 기능 공유와 재활용 분류, 모듈 간 결합도와 응집도를 식별
- 빌드 진행
  - 개발된 소스의 컴파일을 진행
  - 정상 기능 단위 및 서비스 분류
  - 빌드 도구 확인 및 정상 수행
  - 컴파일 이외 도구의 다양한 기능 확인

- 사용자 환경 분석
  - 최소 사용자 환경 사전 정의
  - 모듈 단위의 여러 가지 기능별 사용자 환경 테스트
- 패키징 적용 시험
  - 사용자 환경에서의 패키징 적용 시험
  - UI 및 시스템 상의 편의성 체크
- 패키징 변경 개선
  - 패키징 적용시 변경점 도출
  - 최소 사용자 환경에서 서비스 가능한 수준의 개선
  - 개선 버전의 재배포

## 2. 릴리즈 노트

### (1) 릴리즈 노트의 개념

- 소프트웨어 제품과 함께 배포되는 문서들을 말한다.
- 고객이 이미 사용 중인 제품의 경우 릴리즈 노트는 업데이트가 출시될 때 고객에게 전달
- 소프트웨어의 서비스 내용과 수정, 변경 또는 개선되는 일련의 작업들이 릴리즈 노트를 통해 제공

### (2) 릴리즈 노트의 역할

- 릴리즈 노트에는 테스트 결과와 정보가 포함
- 사용자에게 보다 더 확실한 정보를 제공
- 기본적으로 전체적인 제품의 수행 기능 및 서비스의 변화를 공유
- 자동화 개념과 함께하여 적용할 수 있다.

### (3) 릴리즈 노트 작성 항목

작성항목	설명
Header	- 문서 이름(릴리즈 노트 이름), 제품 이름, 버전 번호, 릴리즈 날짜, 참고 날짜, 노트 버전 등
개요	- 제품 및 변경에 대한 간략한 전반적 개요
목적	- 릴리즈 버전의 새로운 기능목록과 릴리즈 노트의 목적에 대한 간략한 개요. - 버그 수정 및 새로운 기능 기술.
이슈 요약	- 버그의 간단한 설명 또는 릴리즈 추가 항목 요약
재현 항목	- 버그 발견에 따른 재현 단계 기술
수정/개선 내용	- 수정 / 개선의 간단한 설명 기술
사용자 영향도	- 버전 변경에 따른 최종 사용자 기준의 기능 및 응용 프로그램 상의 영향도 기술
SW 지원 영향도	- 버전 변경에 따른 SW의 지원 프로세스 및 영향도 기술
노트	- SW 및 HW Install 항목, 제품, 문서를 포함한 업그레이드 항목 메모
면책 조항	- 회사 및 표준 제품과 관련된 메시지. 프리웨어, 불법 복제 방지, 중복 등 참조에 대한 고지 사항
연락 정보	- 사용자 지원 및 문의 관련한 연락처 정보

#### (4) 릴리즈 노트 추가 작성 및 개선 사항 발생의 예외 케이스

- 테스트 단계에서의 베타 버전 출시
- 긴급 버그 수정 시
- 자체 기능 향상을 포함한 모든 추가 기능의 향상
- 사용자 요청에 따른 특이한 케이스 발생

### 3. DRM

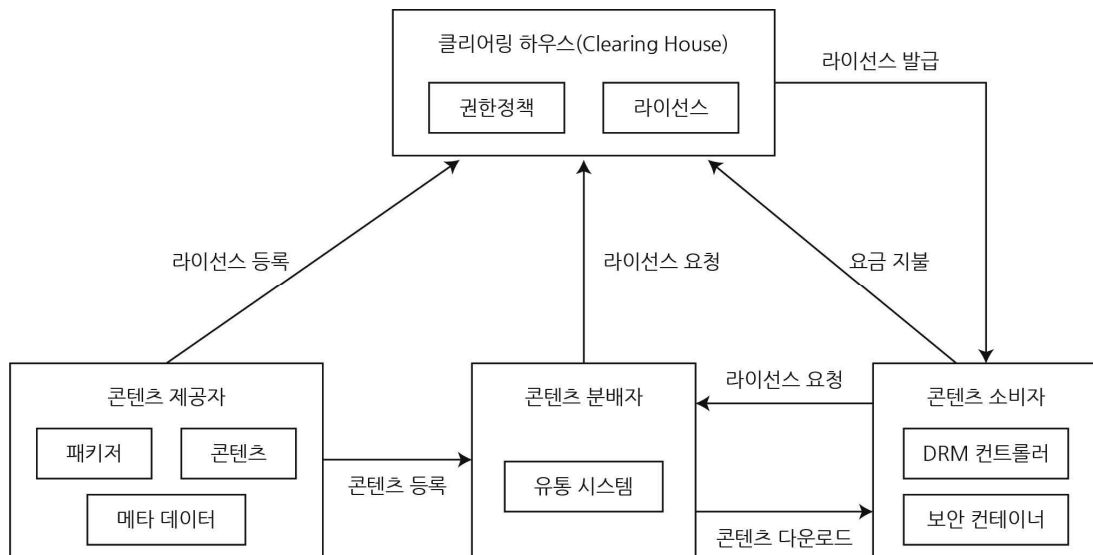
#### (1) DRM(Digital Rights Management)의 개념

- 각종 디지털 콘텐츠의 불법적인 사용을 제한하고, 승인된 사용자의 콘텐츠 사용을 저작권 소유자의 의도에 따라 제어하는 기술
- 콘텐츠의 보호를 위한 암호화 기술과 사용 권한 제어를 위한 라이선스 관리 기술로 구성
- 단순 보안기술 보다는 좀 더 포괄적인 개념으로, 저작권 승인과 집행을 위한 소프트웨어와 보안기술, 지불, 결제 기능 등이 모두 포함

#### (2) DRM의 특징

- 콘텐츠의 공개키로 암호화 하고, 콘텐츠의 비밀키를 판매
- 유료 콘텐츠 사용자에게 서비스 사용료를 부과하고 안전하게 결제한다.
- 저작권의 라이선스에 따른 분배를 투명하게 한다.
- 라이선스를 제공하는 기관과 콘텐츠를 배포하는 기관을 분리하여 투명한 거래구조로 개선한다.
- 콘텐츠를 소비자가 사용하는데 있어 횟수, 날짜, 장비 등의 사용권한을 통제한다.

#### (3) DRM의 구성 및 흐름



- 콘텐츠 제공자(Contents Provider)
  - 콘텐츠를 제공하는 저작권자
- 콘텐츠 분배자 (Contents Distributor)
  - 쇼핑몰 등으로써 암호화된 콘텐츠 제공

- 패키저 (Packager)
  - 콘텐츠를 메타 데이터와 함께 배포 가능한 단위로 묶는 기능
- 보안 컨테이너
  - 원본을 안전하게 유통 하기위한 전자적 보안 장치
- DRM 컨트롤러
  - 배포된 콘텐츠의 이용 권한을 통제
- 클리어링 하우스 (Clearing House)
  - 키 관리 및 라이선스 발급 관리

#### (4) DRM 사용 규칙 제어 기술

##### ① 콘텐츠 식별 체계 (Identification)

- 디지털 콘텐츠에 고유 식별 번호를 부여하여 관리하고 운영
- 대표적으로 DOI(Digital Object Identifier), URI 가 있다.

##### ② 메타데이터 (Meta Data)

- 콘텐츠에 관한 구조화된 데이터
- 콘텐츠의 속성정보

##### ③ 권리표현기술 (Right Expression)

- 콘텐츠에 대한 규칙 설정
- 어느 사용자가 어떠한 권한과 어떠한 조건으로 콘텐츠를 이용할 수 있는지 정의
- 콘텐츠의 사용조건(기간, 횟수) 등에 의해 사용이 제한 될 수 있고, 주로 XML 기반으로 권한 표현 언어가 개발
- XrML(eXtensible rights mark-up language) 기술이 대표적

##### ④ 권리표현 종류

권리표현 종류	설명
Render Permission	사용자에게 콘텐츠가 표현되고 이용되는 권리 형태를 정의
Transport Permission	사용자들 간에 권리의 교환이 이루어지는 권리 형태를 정의
Derivative Permission	콘텐츠의 추출 변형이 가능한 권리 형태를 정의

#### (5) 저작권 보호 기술

##### ① 암호화 기술

- 특정 키를 가진 사용자만이 해당 콘텐츠를 사용할 수 있도록 한다.
- 암호화 키와 복호화 키가 서로 다른 비대칭키 방식과 두 키가 동일한 대칭키 방식 있다.

##### ② 위변조 방지(Tamper-proofing)

- 콘텐츠에 승인되지 않은 조작이 가해졌을 때, 위변조 사항을 감지할 수 있도록 하고, 오류 동작을 일으키게끔 하는 기술
- 부정 조작에 대한 방어 기술

## ③ 워터마킹(Watermarking)

- 콘텐츠에 저작권 정보를 은닉하여, 향후 저작권 분쟁이 일어날 경우, 추적을 통해 저작권자를 확인할 수 있게 해주는 기술
- 워터마킹(Watermarking), 핑거프린팅(Fingerprinting) 으로 구분

관점	워터마킹	핑거프린팅
목적	불법 복제 방지	불법 유통 방지
삽입정보	저작권 정보	저작권 정보 + 구매자 정보
컨텐츠 변화 시점	최초 저작 시점	구매시점 마다
취약점	불법 유통	공모 공격

## (6) DRM 구성요소

구성요소	설명
암호화 (Encryption)	- 콘텐츠 및 라이선스를 암호화하고, 전자 서명을 할 수 있는 기술 - PKI, Symmetric/Asymmetric Encryption, Digital Signature
키 관리 (Key Manangement)	- 콘텐츠를 암호화한 키에 대한 저장 및 배포 기술 (Centralized, Enveloping)
암호화 파일 생성 (Packager)	- 콘텐츠를 암호화된 콘텐츠로 생성하기 위한 기술 - Pre-packaging, On-the-fly Packaging
식별 기술 (Identification)	- 콘텐츠에 대한 식별 체계 표현 기술 - DOI, URI
저작권 표현 (Right Expression)	- 라이선스의 내용 표현 기술 - XrML/MPGE-21 REL, ODRL
정책 관리 (Policy management)	- 라이선스 발급 및 사용에 대한 정책표현 및 관리기술 - XML, Contents Management System
크랙 방지 (Tamper Resistance)	- 크랙에 의한 콘텐츠 사용 방지 기술 - Secure DB, Secure Time Management, Encryption
인증 (Authentication)	- 라이선스 발급 및 사용의 기준이 되는 사용자 인증 기술 - User/Device Authentication, SSO, DiGital Certificate



## Section 3. 제품 소프트웨어 매뉴얼 작성

### 1. 제품 소프트웨어 매뉴얼 작성

#### (1) 제품 소프트웨어 매뉴얼 개념

- 사용자가 제품 구매 후 최초 설치 시 참조하게 되는 매뉴얼
- 제품 소프트웨어 소개, 설치 파일, 설치 절차 등이 포함

#### (2) 제품 소프트웨어 설치 매뉴얼

##### ① 설치 매뉴얼 작성의 기본 사항

- 설치 매뉴얼은 개발자의 기준이 아닌 사용자의 기준으로 작성
- 최초 설치 실행부터 완료까지 순차적으로 진행
- 각 단계별 메시지 및 해당 화면을 순서대로 전부 캡처하여 설명
- 설치 중간에 이상 발생 시 해당 메시지 및 에러에 대한 내용을 분류하여 설명

##### ② 제품 소프트웨어 설치 매뉴얼의 작성 항목

기본 작성 항목	설명
목차 및 개요	<ul style="list-style-type: none"> <li>- 매뉴얼 전체의 내용을 순서대로 요약</li> <li>- 설치 매뉴얼의 주요 특징에 대해 정리</li> <li>- 설치 매뉴얼에서의 구성과 설치 방법, 순서 등에 대해 기술함</li> </ul>
문서 이력 정보	<ul style="list-style-type: none"> <li>- 설치 매뉴얼 변경 이력 정보</li> </ul>
설치 매뉴얼의 주석	<ul style="list-style-type: none"> <li>- 주의 사항: 사용자가 제품 설치 시 반드시 숙지해야 하는 중요한 정보 주석 표시</li> <li>- 참고 사항: 설치 관련하여 영향을 미치는 특별한 사용자 환경 및 상황에 대한 내용 주석 표시</li> </ul>

##### ③ 제품 소프트웨어 설치 환경 체크 항목

확인 항목	체크할 내용
사용자 환경	사용자의 CPU 및 Memory, OS 등의 적합 환경
응용 프로그램	설치 전 다른 응용 프로그램의 종료
업그레이드 버전	업그레이드 이전 버전에 대한 존재 유무 확인
백업 폴더 확인	데이터 저장 폴더를 확인하여 설치 시 폴더 동기화

**④ 제품 소프트웨어 설치 매뉴얼 구성요소**

구성 요소	설명
제품 소프트웨어 개요	- 제품 소프트웨어의 주요 기능 및 UI 설명 - UI 및 화면 상의 버튼, 프레임 등을 도식화하여 설명
설치 관련 파일	- 제품 소프트웨어를 설치하기 위한 관련 파일 설명 - 설치 구동을 위한 exe 실행 - ini나 log 파일 같은 관련 파일
설치 절차	- 소프트웨어 설치 방법을 순서대로 상세히 설명
설치 아이콘	- Windows 구동용 설치 아이콘 설명
프로그램 삭제	- 해당 소프트웨어 삭제 시 원래대로 삭제하는 방법을 설명
설치 환경	- CPU, Memory, OS 등 환경설명
설치 버전 및 작성자	- 소프트웨어 릴리즈 버전 및 작성자 정보
고객 지원 방법 및 FAQ	- 실제 설치 시 자주 발생하는 어려움들을 FAQ로 정리 - 유선 및 E-mail, Website URL

**⑤ 제품 소프트웨어 설치 매뉴얼 작성 순서**

- 기능 식별
- UI 분류
- 설치 파일/백업 파일 확인
- Uninstall 절차 확인
- 이상 Case 확인
- 최종 매뉴얼 적용

**(3) 제품 소프트웨어 사용자 매뉴얼****① 제품 소프트웨어 사용자 매뉴얼의 개요**

- 개발이 완료된 제품 소프트웨어를 고객에게 전달하기 위한 형태로 패키징하고, 설치와 사용에 필요한 제반 절차 및 환경 등 전체 내용을 포함하는 매뉴얼
- 개발된 컴포넌트 사용 시에 알아야 할 내용을 기술하며 패키지의 기능, 패키지의 인터페이스, 포함하고 있는 메서드나 오퍼레이션과 메서드의 파라미터 등의 설명이 포함

**② 사용자 매뉴얼 작성 절차**

- 작성 지침 정의
- 사용자 매뉴얼 구성 요소 정의
- 구성 요소별 내용 작성
- 사용자 매뉴얼 검토

## ③ 사용자 매뉴얼 작성 항목

작성 항목	설명
목차 및 개요	<ul style="list-style-type: none"> <li>- 매뉴얼 전체의 내용을 순서대로 요약</li> <li>- 제품 소프트웨어의 주요 특징에 대해 정리</li> <li>- 사용자 매뉴얼에서의 구성과 실행 방법, 메뉴에 대한 설명을 비롯하여 사용법, 각 항목에 따른 점검 기준, 그리고 설정 방법 등에 대해 기술함</li> </ul>
문서 이력 정보	<ul style="list-style-type: none"> <li>- 사용자 매뉴얼 변경 이력 정보</li> </ul>
사용자 매뉴얼의 주석	<ul style="list-style-type: none"> <li>- 주의 사항: 사용자가 반드시 숙지해야 하는 중요한 정보의 주석 표시</li> <li>- 참고 사항: 특별한 사용자 환경 및 상황에 대한 내용의 주석 표시</li> </ul>
기록 항목	<ul style="list-style-type: none"> <li>- 제품명칭, 모델명, 기록 항목에 대한 문서 번호, 제품 번호, 구입 날짜 등을 기재</li> </ul>
기본 사항	<ul style="list-style-type: none"> <li>- 소프트웨어 개요, 사용방법, 모델/버전별 특징, 기능 및 인터페이스 특징, 구동 환경 등을 기재</li> </ul>
고객 지원 방법 및 FAQ	<ul style="list-style-type: none"> <li>- 소프트웨어 사용시 자주 발생하는 어려움들을 FAQ로 정리</li> <li>- 유선 및 E-mail, Website URL</li> </ul>
준수 정보 & 제한 보증	<ul style="list-style-type: none"> <li>- 시리얼 보존, 불법 등록 사용금지 등의 준수 사항 권고</li> <li>- 저작권 정보 관련 작성</li> </ul>

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 5천만 원 이하의 벌금과 민사상 손해배상을 청구합니다.