# 01장. 요구사항 확인

## 001. 소프트웨어 생명 주기

### 소프트웨어 생명 주기(Software Life Cycle)

* 소프트웨어를 개발하기 위한 설계, 운용, 유지보수 등의 과정을 각 단계별로 나눈 것

### 폭포수 모형(Waterfall Model)

* 이전 단계로 돌아갈 수 없음
* 각 단계를 확실히 매듭짓고 그 결과를 철저하게 검토하여 승인 과정을 거친 후에 다음 단계를 진행함
* 가장 오래됨, 가장 폭넓게 사용됨, 고전적 생명 주기 모형이라고도 함

### 프로토타입 모형(Prototype Model, 원형 모형)

* 실제 개발될 소프트웨어에 대한 견본품(Prototype)을 만들어 최종 결과물을 예측하는 모형

### 나선형 모형(Spiral Model, 점진적 모형)

* 여러 번의 소프트웨어 개발 과정을 거쳐 점진적으로 개발하는 모형
* 보헴이 제안함, 폭포수 + 프로토타입 모형에다가 위험 분석 기능 추가



위험 분석 개발 및 검증 고객 평가

계획 수립

고객의 요구가 추가되면 다시 돌아감

### 애자일 모형(Agile Model)

* 고객의 요구사항 변화에 유연하게 대응할 수 있도록 일정한 주기를 반복하면서 개발하는 모형
* 스크럼(Scrum),

: 팀이 중심이 되어 개발의 효율성을 높이는 기법

: 제품 책임자(PO, 백로그 작성), 스크럼 마스터(SM, 가이드 역할), 개발팀(DT)으로 구성

: 계획하여 진행(스프린트)한 후 회의와 검토를 거쳐 회고

* XP(eXtreme Programming)

: 고객의 요구사항에 유연하게 대응하기 위해 고객의 참여와 개발 과정의 반복을 극대화 -> 생산성 향상

: 계획하고 진행(이터레이션)한 후 검사하고 출시(릴리즈)

: 핵심 가치 5가지 - 의사소통, 단순성, 용기, 존중, 피드백

: Pair Programming(짝 프로그래밍), Collective Ownershil(공동 코드 소유), Test-Driven Development(테스트 주도 개발), Whole Team(전체 팀),

Continuous Integration(계속적인 통합), Refactoring(리팩토링), Small Releases(소규모 릴리즈)

* 칸반(Kanban), Lean, 기능 중심 개발(FDD)

### 소프트웨어 공학(SE; Software Engineering)

* 소프트웨어의 위기를 극복하기 위한 방안으로 연구된 학문

## 현행 시스템 파악

### 1) 절차

* 1단계 : 시스템 구성 파악, 시스템 기능 파악, 시스템 인터페이스 파악
* 2단계 : 아키텍처 구성 파악, 소프트웨어 구성 파악
* 3단계 : 하드웨어 구성 파악, 네트워크 구성 파악
* 구기인이 아랐소 라고 하네

## 개발 기술 환경 파악

### 개요

* OS, DBMS, 미들웨어 등을 선정할 때 고려해야 할 사항을 기술하고 오픈 소스를 사용할 때 주의해야 할 내용 제시

### 운영체제(OS, Operating System)

* 컴퓨터 시스템의 자원을 효율적으로 관리하며, 사용자가 컴퓨터를 편리하고 효율적으로 사용할 수 있도록 환경을 제 공하는 소프트웨어
* 운영체제 관련 요구사항 식별 시 고려사항

: 가용성, 성능, 기술 지원, 주변 기기, 구축 비용

### 데이터베이스 관리 시스템(DBMS; DataBase Management System)

* 사용자와 데이터베이스 사이에서 사용자의 요구에 따라 정보를 생성해주고, 데이터베이스를 관리해주는 소프트웨어
* DBMS 관련 요구사항 식별 시 고려사항

: 가용성, 성능, 기술 지원, 상호 호환성, 구축 비용

### 웹 애플리케이션 서버(WAS; Web Application Server)

* 동적인 콘텐츠를 처리하기 위해 사용되는 미들웨어
* WAS 관련 요구사항 식별 시 고려사항

: 가용성, 성능, 기술 지원, 구축 비용

### 오픈 소스(Open Source)

* 누구나 제한 없이 사용할 수 있도록 소스 코드를 공개한 소프트웨어
* 오픈 소스 관련 요구사항 식별 시 고려사항

: 라이선스의 종류, 사용자 수, 기술의 지속 가능성

## 요구사항 정의

### 1) 요구사항

* 소프트웨어가 어떤 문제를 해결하기 위해 제공하는 서비스에 대한 설명과 운영되는 데 필요한 제약조건
* 기능 요구사항(Functional requirements)

: 기능이나 수행과 관련된 요구사항

* 비기능 요구사항(Non-functional requirements)

: 품질이나 제약사항과 관련된 요구사항

* 사용자 요구사항(User requirements)

: 사용자 관점에서 본 시스템이 제공해야 할 요구사항

* 시스템 요구사항(System requirements)

: 개발자 관점에서 본 시스템 전체가 제공해야 할 요구사항

## 요구사항 개발 프로세스

### 요구사항 개발 프로세스

* 요구사항을 체계적으로 도출하고 분석한 후 명세서에 정리한 다음 확인 및 검증하는 활동
* 도분명확
* 도출(Elicitation, 요구사항 수집)

: 시스템 개발에 관련된 사람들이 서로 의견을 교환하여 요구사항을 식별하고 이해하는 과정

: 청취와 인터뷰, 설문, 브레인스토밍, 워크샵, 프로토타이핑, 유스케이스

* 분석(Analysis)

: 요구사항 중 이해되지 않는 부분을 걸러내기 위한 과정

: 타당성 확보, 비용과 일정에 대한 제약 설정

: 구조적 분석 기법

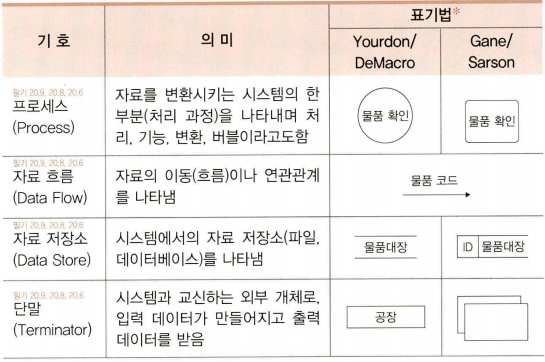
* + 자료의 흐름과 처리를 중심으로 하는 요구사항 분석 방법(조직도 형태로 하향식 세분화)
  + 자료 흐름도(DFD; Data Flow Diagram)

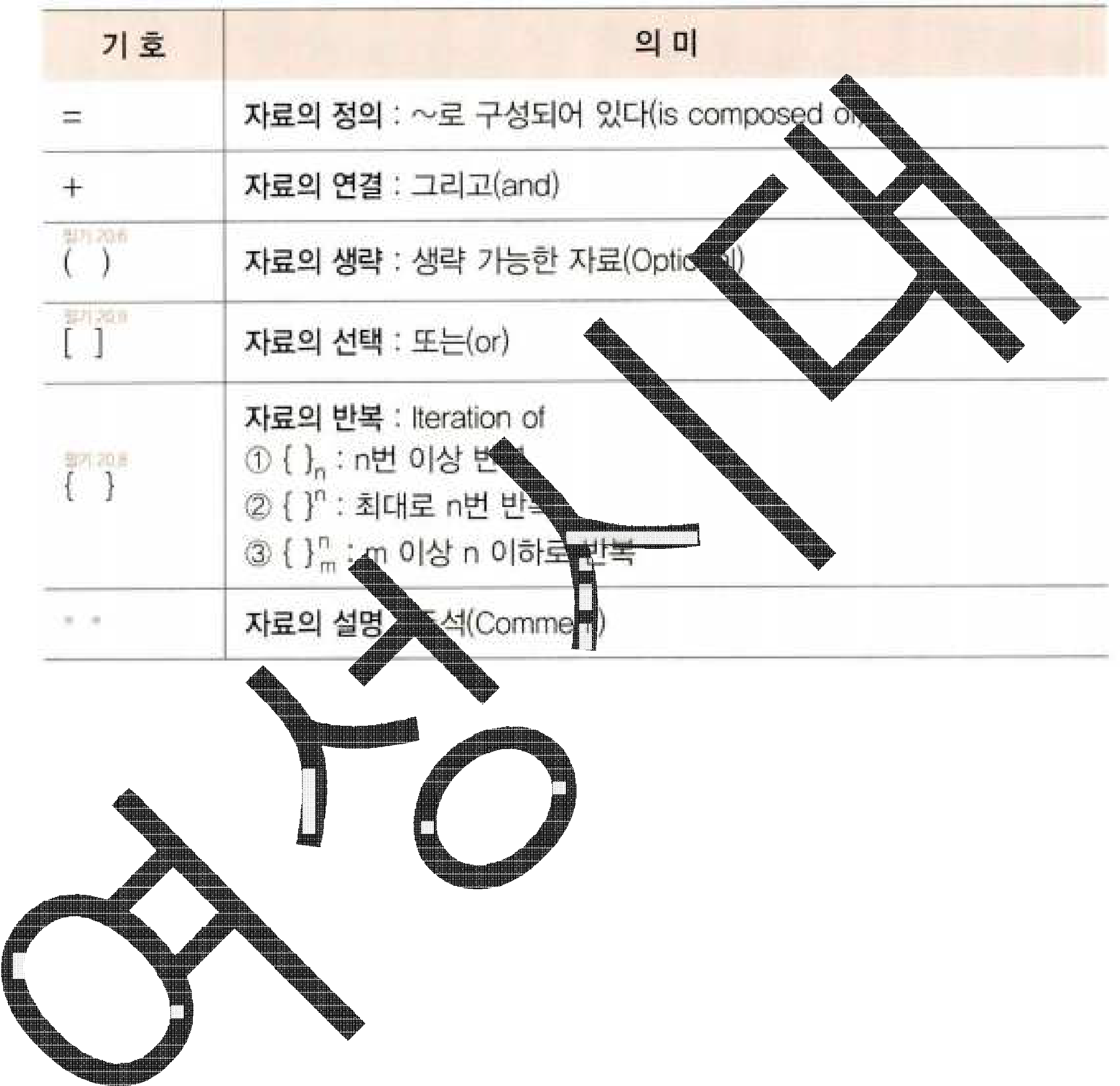
: 자료의 흐름 및 변환 과정과 기능을 도형 중심으로 기술하는 방법

* + 자료 사전(DD; Data Dictionary)

: 자료 흐름도에 있는 자료를 더 자세히 정의하고 기록한 것

: 데이터를 설명하는 데이터(데의터의 데이터 또는 메타 데이터라고도 함)



: 요구사항 분석용 CASE(자동화 도구)

* + 요구사항을 자동으로 분석하고, 요구사항 분석 명세서를 기술하도록 개발된 도구
  + SADT, SREM = RSL/SREVS, PSL/PSA, TAGS

: HIPO(Hierachy Input Process Output)

* + 시스템 실행 과정인 입력, 처리, 출력의 기능을 표현한 것
  + HIPO Chart : 시스템의 기능을 여러 개의 고유 모듈로 분할하여 이들 간의 인터페이스를 계층 구조로 표현한 것
* 명세(Specification)

: 분석된 요구사항을 바탕으로 모델을 작성하고 문서화하는 것

: 정형 명세 기법 - 수학적 원리 기반, 모델 기반

* + 일관성 있음 -> 완전성 검증 가능, 사용자 이해 어려움

: 비정형 명세 기법 - 상태/기능/객체 중심

* + 자연어를 기반으로 서술 또는 다이어그램으로 작성
  + 일관성 떨어짐 -> 해석이 달라질 수 있음, 사용자 이해 쉬움
* 확인(Vaildation, 검증)

: 요구사항 명세서가 정확하고 완전하게 작성되었는지를 검토

### 요구공학(Requirements Engineering)

* 요구사항을 정의하고 분석 및 관리하는 프로세스를 연구하는 학문

## 010. UML(Unified Modeling Language)

### UML(Unified Modeling Language)

* 시스템 개발 과정에서 의사소통이 원활하게 이루어지도록 표준화한 대표적인 객체지향 모델링 언어
* 사물(Things)

: 다이어그램 안에서 관계가 형성될 수 있는 대상들

: 구조 사물, 행동 사물, 그룹 사물, 주해 사물

* 관계(Relationships)

: 사물과 사물 사이의 연관성을 표현하는 것

: 연관 관계 - 2개 이상의 사물이 서로 관련되어 있는 관계(실선, 방향성은 화살표, 양방향은 실선만) (Association) - ex) 사람과 집, 선생님과 학생

: 집합 관계 - 하나의 사물이 다른 사물에 포함되어 있는 관계(부분이 전체에게 속이 빈 마름모로 연결) (Aggregation) - ex) 컴퓨터와 프린터

: 포함 관계 - 포함하는 사물의 변화가 포함되는 사물에게 영향을 미치는 관계

(Composition) - 전체와 부분은 서로 독립될 수 없음(부분이 전체에게 속이 채워진 마름모로 연결)

* + ex) 문과 열쇠

: 일반화 관계 - 하나의 사물이 다른 사물에 비해 더 일반적이거나 구체적인 관계 (Generalization) - 하위(자식)가 상위(부모)에게 속이 빈 화살표로 연결

* + ex) 커피와 아메리카노

: 의존 관계 - 서로에게 영향을 주는 짧은 시간 동안만 연관을 유지하는 관계

(Dependency) - 영향을 주는 사물(이용자)이 영향을 받은 사물(제공자)에게 점선 화살표로 연결

* + ex) 등급과 할인율

: 실체화 관계 - 할 수 있거나 해야 하는 기능으로, 서로를 그룹화할 수 있는 관계 (Realization) - 사물에서 기능에게 속이 빈 점선 화살표로 연결

* + ex) 새와 날 수 있다
* 다이어그램(Diagram)

: 사물과 관계를 도형으로 표현한 것

: 정적 모델링 - 사용자가 요구한 기능을 구현하는 데 필요한 자료들의 논리적인 구조를 표현(개발자 관점)

* + 주로 구조적 다이어그램 사용

: 구조적 다이어그램

|  |  |
| --- | --- |
| 클래스(Class) | * 클래스와 클래스가 가지는 속성 클래스 사이의 관계를 표현 * 구성요소 : 클래스, 제약조건, 관계 * 연관 클래스 : 연관 관계에 있는 두 클래스에 추가적으로 표현해야 할 속성이   나 오퍼레이션이 있는 경우 생성하는 클래스 |
| 객체(Object) | * 클래스에 속한 사물(객체)들, 즉 인스턴트를 특정 시점의 객체와 객체 사이의   관계로 표현함   * 럼바우 객체지향 분석 기법에서 객체 모델링에 활용됨 |
| 컴포넌트(Component) | 실제 구현 모듈인 컴포넌트 간의 관계나 컴포넌트 간의 인터페이스를 포현  구현 단계에서 사용됨 |
| 배치(Deployment) | 결과물, 프로세스, 컴포넌트 등 물리적 요소들의 위치를 표현  구현 단계에서 사용됨 |
| 복합체 구조  (Composite Structure) | 클래스나 컴포넌트가 복합 구조를 갖는 경우 그 내부 구조를 표현함 |
| 패키지(Package) | * 모델 요소들을 그룹화한 패키지간의 의존 관계를 표현 * 구성요소 : 패키지, 객체, 의존 관계 |

: 동적 모델링 - 시스템의 내부 구성 요소들의 상태 변화 과정과 과정에서 발생하는 상호작용을 표현

* + 주로 행위 다이어그램 사용

: 기능 모델링 - 개발될 시스템이 갖춰야 할 기능을 사용자와 공유하기 위해 그림으로 표현하는 것

: 행위 다이어그램

|  |  |  |
| --- | --- | --- |
| 기 능 | 유스케이스 (Use case) | * 개발될 시스템을 이용해 수행할 수 있는 기능을 사용자의 관점에서 표현한   것   * 구성요소 : 시스템/시스템 범위, 액터, 유스케이스, 관계 |
| 활동(Activity) | - 시스템이 사용자의 관점에서 시스템이 수행하는 기능을 처리 흐름에 따라 순 |

|  |  |  |
| --- | --- | --- |
|  |  | 서대로 표현한 것(순서도 형태)  - 구성요소 : 액션/액티비티, 시작 노드, 종료 노드, 조건(판단) 노드, 병합 노  드, 포크 노드, 조인 노드, 스윔레인 |
| 동 적 | 시퀀스(Sequence) | * 상호 작용하는 시스템이나 객체들이 주고받는 메시지를 그림으로 표현함 * 구성요소 : 액터, 객체, 생명선, 실행상자, 메시지, 객체 소멸, 프레임 |
| 커뮤니케이션 (Communication) | * 동작에 참여하는 객체들이 주고받는 메시지와 객체들간의 연관 관계를 표현   함(협업 다이어그램이라고도 불렸음)   * 구성요소 : 액터, 객체, 링크, 메시지 |
| 상태 (State) | * 객체들 사이에서 발생하는 이벤트에 의한 객체들의 상태 변화를 그림으로 표   현한 것   * 럼바우 객체지향 분석 기법에서 동적 모델링에 활용됨 * 구성요소 : 상태, 시작 상태, 종료 상태, 상태 전환, 이벤트, 프레임 |
| ? | 상호작용 개요  (Interaction Overview) | 상호작용 다이어그램 간의 제어 흐름을 표현함 |
| 타이밍(Timing) | 객체 상태 변화와 시간 제약을 명시적으로 표현함 |

### 스테레오 타입(Stereotype)

* UML에서 표현하는 기본 기능 외에 추가적인 기능을 표현하는 것

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ≪include≫ | 포함 관계 | ≪interface≫ | 인터페이스 정의 | ≪constructor≫ | 생성자 역할 수행 |
| ≪extend≫ | 확장 관계 | ≪exception≫ | 예외 정의 |  |  |

## 소프트웨어 개발 방법론

### 소프트웨어 개발 방법론

* 소프트웨어 개발, 유지보수 등에 필요한 수행 방법과 각종 기법 및 도구를 체계적으로 정리하여 표준화한 것
* 밑에 방법론 + 애자일 방법론이 있음

### 구조적 방법론

* 사용자 요구사항을 파악하여 문서화하는 처리(Process) 중심의 방법론
* 분할과 정복(Divide and Conqure) 원리 적용
* 타당성 검토 -> 계획 -> 요구사항 -> 설계 -> 구현 -> 시험 -> 운용 및 유지보수

### 정보공학 방법론

* 계획, 분석, 설계, 구축에 정형화된 기법들을 통합 및 적용하는 자료(Data) 중심의 방법론
* 정보전략 계획 수립 -> 업무 영역 분석 -> 업무 시스템 설계 -> 업무 시스템 구축

### 객체지향 방법론

* 객체들을 조립해서 필요한 소프트웨어를 구현하는 방법론
* 구성요소 : 객체, 클래스, 메시지
* 기본 원칙 : 캡슐화, 정보 은닉, 추상화, 상속성, 다형성
* 요구 분석 -> 설계 -> 구현 -> 테스트 및 검증 -> 인도

### 컴포넌트 기반 방법론

* 컴포넌트를 조합하여 새로운 애플리케이션을 만드는 방법론
* 개발 준비 -> 분석 -> 설계 -> 구현 -> 테스트 -> 전개 -> 인도

### 제품 계열 방법론

* 제품에 적용하고 싶은 공통된 기능을 정의하여 개발하는 방법론
* 임베디드 소프트웨어를 만드는 데 적합함

## S/W 공학의 발전적 추세

### 소프트웨어 재사용(Software Reuse)

* 이미 개발되어 인정받은 소프트웨어를 다른 소프트웨어 개발이나 유지에 사용하는 것
* 합성 중심(Composition-Based)

: 블록을 만들어서 끼워 맞춰 소프트웨어를 완성시키는 방법으로, 블록 구성 방법이라고도 함

* 생성 중심(Generation-Based)

: 추상화 형태로 써진 명세를 구체화하여 프로그램을 만드는 방법으로, 패턴 구성 방법이라고도 함

### 소프트웨어 재공학(Software Reenegineering)

* 기존 시스템을 이용하여, 보다 나은 시스템을 구축하고, 새로운 기능을 추가하여 소프트웨어 성능을 향상시키는 것

### CASE(Computer Aided Software Engineering)

* 소프트웨어 개발 과정에서 사용되는 과정 전체 또는 일부를 컴퓨터와 전용 소프트웨어 도구를 사용하여 자동화하는 것

## 비용 산정 기법

### 소프트웨어 비용 결정 요소

* 프로젝트 요소 : 제품 복잡도, 시스템 크기, 요구되는 신뢰도
* 자원 요소 : 인적 자원, 하드웨어 자원, 소프트웨어 자원
* 생산성 요소 : 개발자 능력, 개발 기간

### 하향식 비용 산정 기법

* 과거의 유사한 경험을 바탕으로 전문 지식이 많은 개발자들이 참여한 회의를 통해 비용을 산정하는 비과학적 방법
* 처음 개발하면 얼마가 들지 모르니까 전문가가 보고 대충 얼마쯤 들겠다고 말해주는 것
* 전문가 감정 기법 : 경험이 많은 두 명 이상의 전문가에게 비용 산정을 의뢰하는 기법
* 델파이 기법 : 전문가 감정 기법의 주관적인 편견을 보완하기 위해 많은 전문가의 의견을 종합하여 산정

### 상향식 비용 산정 기법

* 세부적인 작업 단위별로 비용을 산정한 후 집계하여 전체 비용을 산정하는 방법
* LOC(원시 코드 라인 수, source Line Of Code) 기법

: 각 기능의 원시 코드 라인 수의 비관치(최댓값), 낙관치(최솟값), 기대치(평균)를 측정하여 예측치를 구함

: 예측치 = (낙관치 + (4 \* 기대치) + 비관치) / 6

* 개발 단계별 인월수(Effor Per Task)

: LOC 기법을 보완하기 위해 기능을 구현시키는 데 필요한 노력을 생명 주기의 각 단계별로 산정

* 수학적 산정 기법

: 개발 비용 산정의 자동화를 목표로 함

: COCOMO 모형

* + 보헴이 제안한 LOC에 의한 비용 산정 기법
  + 조직형(Organic Mode)

: 기관 내부에서 개발된 중, 소 규모의 소프트웨어(5만(50KDSI)라인 이하)

* + 반분리형(Semi-Detached Mode)

: 조직형과 내장형의 중간형 소프트웨어(30만(300KDSI)라인 이하)

* + 내장형(Embedded Mode)

; 초대형 규모의 소프트웨어(30만(300KDSI)라인 이상)

* + 모형의 종류 : 기본형(Basic), 중간형(intermediate), 발전형(Detailed)

: Putnam 모형

* + 소프트웨어 생명 주기의 전 과정 동안에 사용될 노력의 분포를 예상하는 모형
  + Rayleigh-Norden 곡선의 노력 분포도를 기초로 함(개발기간↑ → 노력↓)

: 기능 점수(FP; Function Point) 모형

* + 소프트웨어의 기능을 증대시키는 요인별로 기능 점수(FP)를 구한 후 비용을 산정

: 비용 산정 자동화 추정 도구 - SLIM, ESTIMACS

## 프로젝트 일정 계획

### PERT(Program Evaluation and Review Technique, 프로그램 평가 및 검토 기술)

* 전체 작업의 상호 관계를 표시하는 네트워크
* 개발 경험이 없어 소요 기간 예측이 어려운 프로젝트 일정 계획에 사용함
* 노드(작업), 간선(낙관치, 기대치, 비관치)으로 구성됨

### CPM(Critical Path Method, 임계 경로 기법)

* 작업을 나열하고 작업에 필요한 소요 기간을 예측하는 데 사용하는 기법
* 가장 소요 기간이 오래 걸리는 경로를 찾으면 됨
* 노드(작업), 간선(작업 사이의 전후 의존 관계)으로 구성됨

### 간트 차트

* 프로젝트의 작업 일정을 막대 도표를 이용하여 표시하는 프로젝트 일정표
* 이정표, 작업 일정, 작업 기간, 산출물로 구성됨

## 소프트웨어 개발 방법론 결정

* 프로젝트 관리와 재사용 현황을 소프트웨어 개발 방법론에 반영하고, 확정된 소프트웨어 생명 주기와 개발 방법론에 맞춰 소프트웨어 개발 단계, 활동, 작업, 절차 등을 정의하는 것
* 프로젝트 관리 : 최소의 비용으로 시스템을 개발하기 위한 전반적인 활동

## 소프트웨어 개발 표준

* 소프트웨어 개발 단계에서 수행하는 품질 관리에 사용되는 국제 표준

### ISO/IEC 12207

* ISO에서 만든 표준 소프트웨어 생명 주기 프로세스
* 기본 생명 주기 프로세스, 지원 생명 주기 프로세스, 조직 생명 주기 프로세스로 구분됨

### CMMI(Capability Maturity Model Integration)

* 소프트웨어 개발 조직의 업무 능력 및 조직의 성숙도를 평가하는 모델

- 초기(Initial) - 관리(Managed) - 정의(Defined) - 정량적 관리(Quantitatively Managed) - 최적화(Optimizing)

### SPICE(Software Process Improvement and Capability dEtermination)

* 소프트웨어 품질 및 생산성 향상을 위해 소프트웨어 프로세스를 평가 및 개선하는 국제 표준
* 공식 명칭은 ISO/IEC 15504
* 구성 : 고객-공급자, 공학 프로세스, 지원 프로세스, 관리 프로세스, 조직 프로세스

- 불완전(Incomplete) - 수행(Performed) - 관리(Managed) - 확립(Estabilished)

* + 예측(Predicable) - 최적화(Optimizing)

## 소프트웨어 개발 방법론 테일러링

* 소프트웨어 개발 방법론의 절차, 사용 기법 등을 수정 및 보완하는 작업
* 테일러링 : 기성복이 아니라 맞춤 정장 같은 느낌
* 내부적 기준 : 목표 환경, 요구사항, 프로젝트 규모, 보유 기술
* 외부적 기준 : 법적 제약사항, 표준 품질 기준

## 소프트웨어 개발 프레임워크

* 소프트웨어 개발에 공통적으로 사용되는 구성 요소와 아키텍처를 일반화하여 제공해주는 반제품 형태의 소프트웨어 시스템
* 스프링 프레임워크(Spring Framework)

: 자바 플랫폼을 위한 오픈 소스 경량형 애플리케이션 프레임워크

* 전자정부 프레임 워크

: 대한민국의 공공부문 정보화 사업 시 정보 시스템의 구축을 위해 기능 및 아키텍처를 제공

* 닷넷 프레임워크(.NET Framework)

: Windows 프로그램의 개발 및 실행 환경을 제공하는 프레임워크

* 특성 : 모듈화, 재사용성, 확장성, 제어의 역흐름



# 02장. 데이터 입 · 출력 구현

## 데이터 전환

* 기존 정보 시스템에 축적되어 있는 데이터를 추출(Extraction)하여 변환(Transformation)한 후, 적재(Loading)하는 일련의 과정(ETL)
* 데이터 전환 계획서 : 데이터 전환 작업에 필요한 모든 계획을 기록하는 문서

## 데이터 검증

* 원천 시스템의 데이터를 목적 시스템의 데이터로 전환하는 과정이 정상적으로 수행되었는지 확인하는 과정
* 방법 : 로그 검증, 기본 항목 검증, 응용 프로그램 검증, 응용 데이터 검증, 값 검증
* 단계 : 추출 – 전환 – DB 적재 – DB 적재 후 – 전환 완료 후

## 오류 데이터 측정 및 정제

* 고품질의 데이터를 운영 및 관리하기 위해 수행
* 데이터 품질 분석 -> 오류 데이터 측정 -> 오류 데이터 정제
* 오류 상태 : Open, Assigned, Fixed, Closed, Deferred, Classified
* 데이터 정제 요청서 : 데이터 정제와 관련된 전반적인 내용을 문서로 작성한 것
* 데이터 정제 보고서 : 정제된 원천 데이터가 정상적으로 정제되었는지 확인한 결과를 문서로 작성한 것

## 데이터베이스 개요

### 데이터저장소

* 데이터들을 논리적인 구조로 조직화하거나 물리적인 공간에 구축한 것을 의미함

### 데이터베이스(Database)

* 공동으로 사용될 데이터를 중복을 배제하여 통합하고 저장장치에 저장하여 항상 사용할 수 있도록 운영하는 운영 데 이터

### DBMS(DataBase Management System; 데이터베이스 관리 시스템)

* 사용자의 요구에 따라 정보를 생성해주고 데이터베이스를 관리해주는 소프트웨어
* 데이터의 종속성과 중복성의 문제를 해결하기 위해 제안된 시스템
* 필수 기능 : 정의(Definition), 조작(Manipulation), 제어(Control)

### 데이터의 독립성

* 논리적 독립성 : 응용 프로그램과 데이터베이스를 독립시킴
* 물리적 독립성 : 응용 프로그램과 보조기억장치 같은 물리적 장치를 독립시킴

### 스키마(Schema)

* 데이터베이스의 구조와 제약조건에 관한 전반적인 명세를 기술한 것
* 외부 스키마 : 사용자나 응용 프로그래머가 각 개인의 입장에서 필요로 하는 데이터베이스의 논리적 구조를 정의
* 개념 스키마 : 데이터베이스의 전체적인 논리적 구조
* 내부 스키마 : 물리적 저장장치의 입장에서 본 데이터베이스 구조

## 데이터베이스 설계

### 고려사항

* 무결성 : 정해진 제약조건을 항상 만족해야 함
* 일관성 : 처음부터 끝까지 변함없이 일정해야 함
* 회복 : 시스템 장애가 발생했을 때 장애 발생 직전의 상태로 복구할 수 있어야 함
* 보안 : 불법적인 데이터의 노출 또는 변경이나 손실로부터 보호할 수 있어야 함
* 효율성 : 응답시간의 단축, 시스템의 생산성, 저장 공간의 최적화 등이 가능해야 함
* 데이터베이스 확장 : 지속적으로 데이터를 추가할 수 있어야 함

### 설계 순서

① 요구 조건 분석

* + 데이터베이스를 사용할 사람들로부터 필요한 용도를 파악하는 것

② 개념적 설계(정보 모델링, 개념화)

* + 현실 세계에 대한 인식을 추상적 개념으로 표현하는 과정

③ 논리적 설계(데이터 모델링)

* + 현실 세계에서 발생하는 자료를 특정 DBMS가 지원하는 논리적 자료 구조로 변환(mapping)시키는 과정

④ 물리적 설계(데이터 구조화)

* + 논리적 구조로 표현된 데이터를 물리적 구조의 데이터로 변환하는 과정

⑤ 구현

* + 논리적 설계와 물리적 설계에서 도출된 데이터베이스 스키마를 파일로 생성하는 과정

\* 요괴(개)놈(논)이 물구 갔다!

## 데이터 모델의 개념

* 현실 세계의 정보들을 단순화, 추상화하여 체계적으로 표현한 개념적 모형

### 구성요소

* 개체(Entity) : 데이터베이스에 표현하려는 것으로, 개념이나 정보 단위 같은 현실 세계의 대상체
* 속성(Attribute)

: 데이터베이스를 구성하는 가장 작은 논리적 단위

: 속성의 특성에 따른 분류

|  |  |
| --- | --- |
| 기본 속성 | 업무 분석을 통해 정의한 속성 |
| 설계 속성 | 원래 업무상 존재하지 않고 설계 과정에서 도출해내는 속성 |
| 파생 속성 | 다른 속성으로부터 계산이나 변형 등의 영향을 받아 발생하는 속성 |

: 속성의 개체 구성 방식에 따른 분류

|  |  |
| --- | --- |
| 기본키 속성(Primary Key Attribute) | 개체를 유일하게 식별할 수 있는 속성 |
| 외래키 속성(Foreign Key Attribute) | 다른 개체와의 관계에서 포함된 속성 |
| 일반 속성 | 개체에 포함되지만 기본키, 외래키는 포함되지 않음 |

: 디그리(Degree) - 속성의 수

* 관계(Relationship) : 개체와 개체 사이의 논리적인 연결

: 관계의 형태 - 1:1, 1:N, N:M

: 관계의 종류 - 종속, 중복, 재귀, 배타

### 표시할 요소

* 구조(Structure), 연산(Operation), 제약조건(Constraint)

### 데이터 모델 종류

* 개념적 데이터 모델 : 현실 세계에 대한 인간의 이해를 돕기 위해 현실 세계에 대한 인식을 추상적 개념으로 표 현하는 과정
* 논리적 데이터 모델 : 개념적 구조를 컴퓨터 세계의 환경에 맞도록 변환하는 과정
* 물리적 데이터 모델 : 실제 컴퓨터에 데이터가 저장되는 방법을 정의하는 물리 데이터베이스 설계 과정

## 식별자(Identifier)

* 하나의 개체 내에서 인스턴스를 유일하게 구분할 수 있는 구분자
* 식별자의 분류 : 대표성 여부 - 주 식별자(특징: 유일성, 최소성, 불변성, 존재성), 보조 식별자

: 스스로 생성 여부 - 내부 식별자, 외부 식별자

: 단일 속성 여부 - 단일 식별자, 복합 식별자

: 대체 여부 - 원조 식별자, 대리 식별자

* 후보 식별자 : 개체에서 각 인스턴스를 유일하게 식별할 수 있는 속성 또는 속성 집합

## E-R(개체-관계) 모델

* 개체와 개체 간의 관계를 기본 요소로 이용하여 현실 세계의 무질서한 데이터를 개념적인 논리 데이터로 표현하기 위한 방법
* 데이터를 개체, 관계, 속성으로 묘사함

## 관계형 데이터베이스의 구조 / 관계형 데이터 모델

### 관계형 데이터 베이스

* 2차원적인 표를 이용해서 데이터 상호 관계를 정의하는 데이터베이스

### 관계형 데이터베이스의 릴레이션 구조

1. **튜플(Tuple)**

* 릴레이션을 구성하는 각각의 행
* 속성의 모임으로 구성됨, 파일 구조에서 레코드와 같은 의미
* 카디널리티(Cardinality) : 튜플의 수, 기수, 대응수라고도 함

### 속성(Attribute)

* 데이터베이스를 구성하는 가장 작은 논리적 단위
* 디그리(Degree) : 속성의 수, 차수라고도 함

### 도메인(Domain)

* 하나의 애트리뷰트가 취할 수 있는 같은 타입의 원자값들의 집합
* ex) 성별 속성의 도메인은 ‘여’와 ‘남’으로 그 외의 값은 입력될 수 없음

### 관계형 데이터 모델(Relational Data Model)

* 2차원적인 표를 이용해서 데이터 상호 관계를 정의하는 DB 구조

## 관계형 데이터베이스의 제약조건 – 키(Key)

### 키(Key)

* 조건에 만족하는 튜플을 찾거나 순서대로 정렬할 때 기준이 되는 속성

### 후보키(Candidate Key)

* 속성들 중에서 튜플을 유일하게 식별하기 위해 사용되는 속성들의 부분집합(대통령 선거 후보들 전부)
* 유일성(Unique), 최소성(Minimality)를 모두 만족시켜야 함

### 기본키(Primary Key)

* 후보키 중에서 특별히 선정된 주키(당선된 대통령)

### 대체키(Alternate Key)

* 후보키가 둘 이상일 때 기본키를 제외한 나머지 후보키(당선되지 못한 나머지 후보들)

### 슈퍼키(Super Key)

* 속성들의 집합으로 구성된 키
* 유일성은 만족하지만 최소성은 만족하지 못함(다른 필드와 묶어서 만들기 때문)

### 외래키(Foreign Key)

* 다른 릴레이션의 기본키를 참조하는 속성 또는 속성들의 집합

## 관계형 데이터베이스의 제약조건 – 무결성(Integrity)

* 데이터베이스에 저장된 데이터 값과 현실 세계의 실제값이 일치하는 정확성

### 무결성의 종류

|  |  |
| --- | --- |
| 개체 무결성 | 기본 테이블의 기본키를 구성하는 어떤 속성도 Null 값이나 중복값을 가질 수 없다는 규정  (공백은 Null이 아님) |
| 참조 무결성 | 외래키 값은 Null이거나 참조 릴레이션의 기본키 값과 동일해야 함 |
| 도메인 무결성 | 주어진 속성값이 정의된 도메인에 속한 값이어야 한다는 규정 |
| 사용자 정의 무결성 | 속성값들이 사용자가 정의한 제약조건에 만족되어야 한다는 규정 |
| NULL 무결성 | 릴레이션의 특정 속성 값이 Null이 될 수 없도록 하는 규정 |
| 고유 무결성 | 릴레이션의 특정 속성에 대해 각 튜플이 갖는 속성값들이 서로 달라야 한다는 규정 |
| 키 무결성 | 하나의 릴레이션에는 적어도 하나의 키가 존재해야 한다는 규정 |
| 관계 무결성 | 릴레이션에 어느 한 튜플의 삽입 가능 여부 또느 한 릴레이션과 다른 릴레이션의 튜플들 사  이의 관계에 대한 적절성 여부를 지정한 규정 |

1. **데이터 무결성 강화** : 애플리케이션, 데이터베이스 트리거, 제약조건

## 관계대수 및 관계해석

### 관계대수

* 원하는 정보와 그 정보를 검색하기 위해서 어떻게 유도하는가를 기술하는 절차적인 언어

### 순수 관계 연산자

|  |  |  |
| --- | --- | --- |
| Select | 선택 조건을 만족하는 튜플의 부분집합을 구하여 새로운 릴레이션을 만드는 연산 | σ  (시그마) |
| Project | 속성 리스트에 제시된 속성값만을 추출하여 새로운 릴레이션을 만드는 연산 | π(파이) |
| Join | 공통 속성을 중심으로 두 개의 릴레이션을 하나로 합쳐서 새로운 릴레이션을 만듦 | ⋈ |
| Division | X⊃Y인 두 개의 릴레이션 R(X)과 S(Y)가 있을 때, R의 속성이 S의 속성값을 모두 가  진 튜플에서 S가 가진 속성을 제외한 속성만을 구하는 연산 | ÷ |

1. **일반 집합 연산자**

* 수학적 집합 이론에서 사용하는 연산자

- 합집합(∪), 교집합(∩), 차집합(−), 교차곱(×)

### 관계해석(Relational Calculus)

* 관계 데이터의 연산을 표현하는 방법
* 원하는 정보가 무엇이라는 것만 정의하는 비절차적 특성을 지님

## 이상/함수적 종속

### 이상(Anomaly)

* 테이블에서 데이터의 중복이 발생하고, 이 중복(Redundancy)으로 인해 문제가 발생하는 현상
* 삽입 이상(Insertion Anomaly)

: 테이블에 데이터를 삽입할 때 의도와는 상관없이 원하지 않은 값들로 인해 삽입할 수 없게 되는 현상

* 삭제 이상(Deletion Anomaly)

: 테이블에서 한 튜플을 삭제할 때 의도와는 상관없는 값들도 함께 삭제되는 현상‘

* 갱신 이상(Update Anomaly)

: 테이블에서 튜플에 있는 속성값을 갱신할 때 일부 튜플의 정보만 갱신되어 불일치성이 생기는 현상

### 함수적 종속(Functional Dependency)

* 학번, 이름, 학년, 학과라는 속성을 가지는 테이블에서 학번으로는 이름을 찾을 수 있지만 이름으로는 학번을 찾을 수 없음(동명이인이 있을 수 있기 때문) : 학번(결정자) -> 이름(종속자)
* 완전 함수적 종속 : X와 Y가 모두 결정자이고 Z가 종속자인 경우에 X, Y -> Z
* 부분 함수적 종속 : X와 Y가 모두 결정자이지만 X만으로도 Z를 결정할 수 있을 때 X -> Z

## 정규화(Normalization)

### 정규화(Normalization)

* 테이블의 속성들이 상호 종속적인 관계(함수적 종속)를 갖는 특성을 이용하여 테이블을 무손실 분해하는 과정
* 가능한 한 중복을 제거하여 삽입, 삭제, 갱신 이상의 발생 가능성을 줄이는 것이 목적

<주문목록>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **제품번호** | **제품명** | **재고수량** | **주문번호** | **고객번호** | **주소** | **주문수량** |
| 1001 | 모니터 | 2000 | A345 | 100 | 서울 | 150 |
| D347 | 200 | 부산 | 300 |
|  |  |  | A210 | 300 | 광주 | 600 |
| 1007 | 마우스 | 9000 | A345 | 100 | 서울 | 400 |
|  |  |  | B230 | 200 | 부산 | 700 |
| 1201 | 키보드 | 2100 | D347 | 200 | 부산 | 300 |

### 제 1정규형

* 모든 속성의 도메인이 원자값만으로 되어 있는 정규형

주문번호, 제품번호

-> 고객번호, 주소, 주문수량 주문번호 -> 고객번호, 주소 고객번호 -> 주소

### 제 2정규형

* 제 1정규형을 만족하면서 기본키가 아닌 모든 속성이 기본키에 대해 완전 함수적 종속을 만족하는 정규형
* 주문번호 -> 고객번호, 주소만 완전 함수적 종속이 아니므로 따로 테이블은 만들어줌

주문번호, 제품번호 -> 주문수량 주문번호 -> 고객번호, 주소

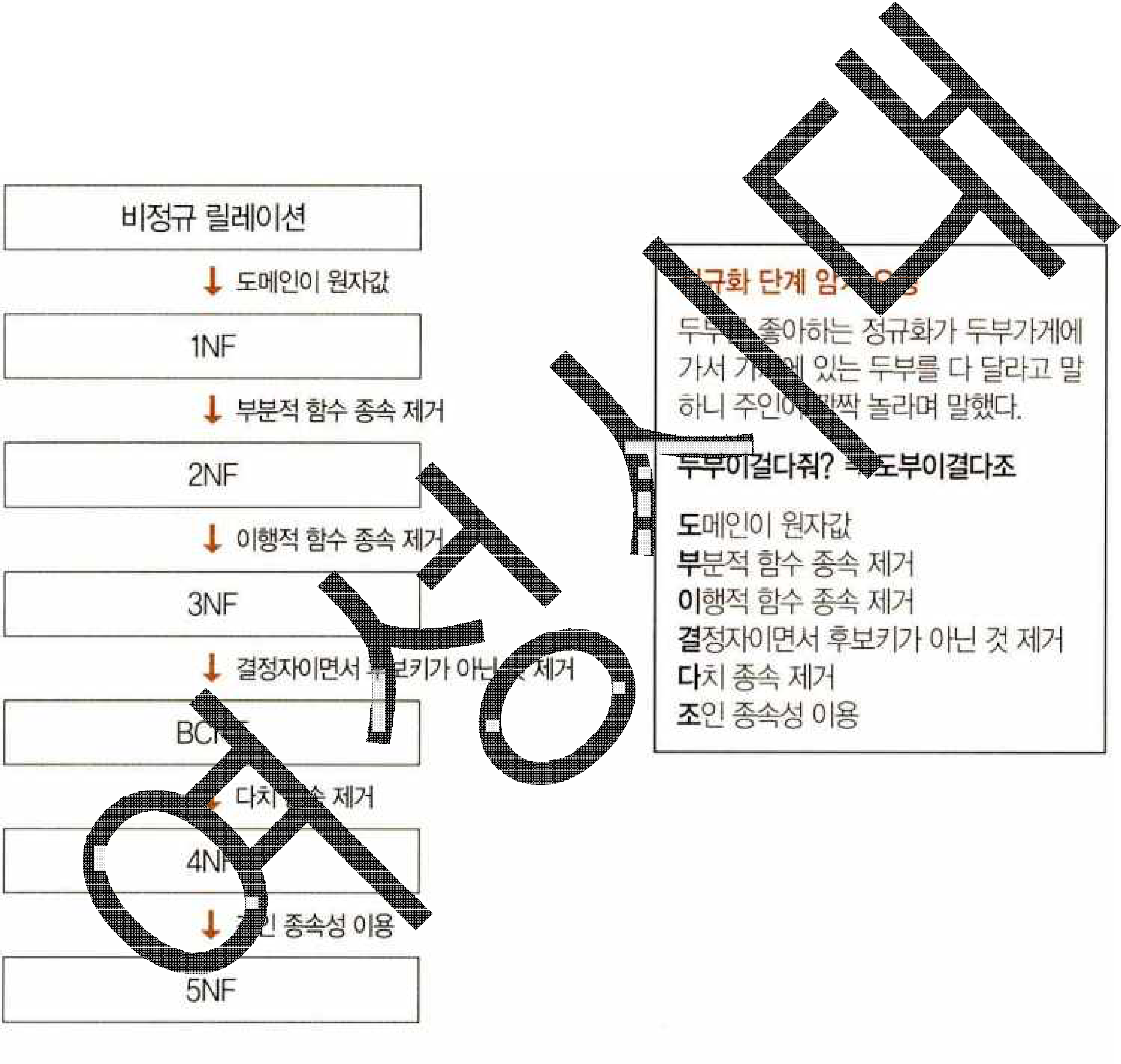
### 제 3정규형

* 제 2정규형을 만족하면서 기본키가 아닌 모든 속성이 기본키에 대해 이행적 함수적 종속을 만족하지 않는 정규형
* 이행적 함수 종속 : A->B이고 B->C일 때 A->C를 만족하는 관계
* 주문번호 -> 고객번호이면서 고객번호 -> 주소이기 때문에 테이블을 따로 분리해줌

### BCNF(3과 4 사이라서 제 3.5정규형이라고도 함)

* 모든 결정자가 후보키인 정규형

### 제 4정규형

* 다중 값 종속 A->->B가 존재할 경우 해당 테이블의 모든 속성이 A에 함수적 종속 관계를 만족하는 정규형
* 다중 값 종속 : A, B, C 3개의 속성을 가진 테이블에서 어떤 복합 속성(A, C)에 대응하는 B값의 집합이 A값에만 종속되고 C값에는 무관한 경우

### 제 5정규형

* 테이블의 모든 조인 종속이 해당 테이블의 후보키를 통해서만 성립되는 정규형

## 반정규화(Denormalization)

### 반정규화

* 정규화된 데이터 모델을 의도적으로 통합, 중복, 분리하여 정규화 원칙을 위배하는 행위

### 테이블 통합

* 두 개의 테이블이 조인(Join)되어 사용되는 경우가 많을 경우 성능 향상을 위해 아예 하나의 테이블로 만들어 사용

### 테이블 분할

* 테이블을 수직 또는 수평으로 분할하는 것
* 수평 분할 : 레코드(Record)를 기준으로 테이블을 분할
* 수직 분할 : 하나의 테이블에 속성이 너무 많을 경우 속성을 기준으로 테이블을 분할

### 중복 테이블 추가

* 작업의 효율성을 향상시키기 위해 테이블을 추가하는 것
* 집계 테이블의 추가 : 집계데이터를 위한 테이블을 생성하고, 각 원본 테이블에 트리거를 설정하여 사용
* 진행 테이블의 추가 : 이력 관리 등의 목적으로 추가하는 테이블
* 특정 부분만을 포함하는 테이블의 추가 : 데이터가 많은 테이블의 특정 부분만을 사용하는 경우 해당 부분만으로

새로운 테이블을 생성

### 중복 속성 추가

* 조인해서 데이터를 처리할 때 데이터를 조회하는 경로를 단축하기 위해 자주 사용하는 속성을 하나 더 추가하는 것

## 시스템 카탈로그

### 시스템 카탈로그(System Catalog)

* 다양한 객체에 관한 정보를 포함하는 시스템 데이터베이스
* 카탈로그들이 생성되면 데이터 사전에 저장되기 때문에 좁은 의미로는 데이터 사전이라고도 함

### 메타 데이터(Meta-Data)

* 시스템 카탈로그에 저장된 정보

### 데이터 디렉터리(Data Directory)

* 데이터 사전에 수록된 데이터에 접근하는 데 필요한 정보를 관리 유지하는 시스템

## 데이터베이스 저장 공간 설계

### 데이터베이스 저장 공간 설계

* 테이블 : 데이터베이스의 가장 기본적인 객체
* 컬럼 : 테이블의 열을 구성하는 요소
* 테이블스페이스 : 테이블이 저장되는 논리적인 영역

### 테이블 종류

* 일반 테이블, 클러스터드 인덱스 테이블, 파티셔닝 테이블, 외부 테이블, 임시 테이블

## 트랜잭션 분석 / CRUD 분석

### 트랜잭션(Transaction)

* 논리적 기능을 수행하기 위한 작업의 단위 또는 한꺼번에 모두 수행되어야 할 일련의 연산들
* 특징 : Atomicity(원자성)
  + 트랜잭션의 연산은 데이터에 모두 반영되도록 완료(Commit)되든지 아니면 전혀 반영되지 않도록 복구(Rollback)되어야 함

: Consistency(일관성)

* + 트랜잭션이 그 실행을 성공적으로 완료하면 언제나 일관성 있는 데이터베이스 상태로 변환함

: Isolation(독립성, 격리성, 순차성)

* + 둘 이상의 트랜잭션이 동시에 병행 실행되는 경우 어느 하나의 트랜잭션 실행 중에 다른 트랜잭션 의 연산이 끼어들 수 없음

: Durability(영속성, 지속성)

* + 성공적으로 완료된 트랜잭션의 결과는 시스템이 고장나더라도 영구적으로 반영되어야 함

### CRUD 분석

* 프로세스와 테이블 간에 CRUD 매트릭스를 만들어서 트랜잭션을 분석하는 것
* CRUD 매트릭스 : 2차원 형태의 표로서, 행(Row)에는 프로세스를, 열(Column)에는 테이블을, 행과 열이 만 나는 위치에는 프로세스가 테이블에 발생시키는 변화를 표시하여 프로세스와 데이터 간의 관계를 분석하는 분석표

### 트랜잭션 분석

* CRUD 매트릭스 기반으로 테이블에 발생하는 트랜잭션 양을 분석하여 테이블에 저장하는 데이터의 양을 유추하고 이를 근거로 DB의 용량 산정 및 구조의 최적화를 목적으로 함
* 트랜잭션 분석서 : 단위 프로세스와 CURD 매트릭스를 이용하여 작성함

## 인덱스

### 인덱스(Index)

* 데이터 레코드를 빠르게 접근하기 위해 <키 값, 포인트> 쌍으로 구성되는 데이터 구조
* 종류 : 트리 기반 인덱스
  + 인덱스를 저장하는 블록들이 트리 구조를 이루고 있는 것

: 비트맵 인덱스

* + 인덱스 컬럼의 데이터를 Bit 값인 0 또는 1로 변환하여 인덱스 키로 사용하는 방법

: 함수 기반 인덱스

* + 컬럼의 값 대신 컬럼에 특정 함수나 수식을 적용하여 산출된 값을 사용하는 것

: 비트맵 조인 인덱스

* + 다수의 조인된 객체로 구성된 인덱스

: 도메인 인덱스

* + 개발자가 필요한 인덱스를 직접 만들어서 사용하는 것

### 클러스터드 인덱스/넌클러스더드 인덱스

* 실제 데이터가 정렬이 되었냐 안 되었냐의 차이
* 클러스터드 인덱스 : 인덱스 키의 순서에 따라 데이터가 정렬되어 저장되는 방식
* 넌클러스터드 인덱스 : 인덱스의 키 값만 정렬되어 있고 실제 데이터는 정렬되지 않는 방식

## 뷰 / 클러스터

### 뷰(View)

* 하나 이상의 기본 테이블로부터 유도된 가상 테이블
* 논리적 데이터 독립성을 제공하지만 독립적인 인덱스를 가질 수는 없음

### 클러스터(Cluster)

* 데이터 액세스 효율을 향상시키기 위해 동일한 성격의 데이터를 동일한 데이터 블록에 저장하는 물리적 저장 방법
* 단일 테이블 클러스터링 : 처리 범위가 넓을 경우
* 다중 테이블 클러스터링 : 조인이 많이 발생하는 경우

## 파티션

### 1) 파티션(Partition)

* 대용량의 테이블이나 인덱스를 작은 논리적 단위인 파티션으로 나누는 것
* 범위 분할(Range Partitioning)

: 지정한 열의 값을 기준으로 분할함

* 해시 분할(Hash Partitioning)

: 해시 함수를 적용한 결과값에 따라 데이터를 분할함

* 조합 분할(Composite Partitioning)

: 범위 분할로 분할한 다음 해시 함수를 적용하여 다시 분할하는 방식

## 분산 데이터베이스 설계

### 데이터베이스 용량 설계

* 데이터가 저장될 공간을 정의하는 것

### 분산 데이터베이스 설계

* 논리적으로는 하나의 시스템에 속하지만 물리적으로는 네트워크를 통해 연결된 여러 개의 사이트에 분산된 데이터베 이스
* 목표 : 위치 투명성(Location Transparnecy)
  + 엑세스하려는 데이터베이스의 실제 위치를 알 필요 없이 단지 데이터베이스의 논리적인 명칭만으 로 엑세스할 수 있음

: 중복 투명성(Replication Transparency)

* + 동일 데이터가 여러 곳에 중복되어 있더라도 사용자는 마치 하나의 데이터가 존재하는 것처럼 사 용하고, 시스템은 자동으로 여러 자료에 대한 작업을 수행함

: 병행 투명성(Concurrency Transparency)

* + 분산 데이터베이스와 관련된 다수의 트랜잭션들이 동시에 실현되더라도 그 트랜잭션의 결과는 영 향을 받지 않음(일관성 유지)

: 장애 투명성(Failure Transparency)

* + 트랜잭션, DBMS, 네트워크, 컴퓨터 장애에도 불구하고 트랜잭션을 정확하게 처리(무결성 보장)
* 분산 설계 방법 : 테이블 위치 분산

: 분할 - 규칙: 완전성, 재구성, 상호 중첩 배제

* + 방법: 수평 분할, 수직 분할

: 할당

## 데이터베이스 이중화/서버 클러스터링

### 데이터베이스 이중화(Database Replication)

* 동일한 데이터베이스를 복제하여 관리하는 것
* 변경 내용의 전달 방식에 따라 Eager 기법(즉시!)과 Lazy 기법으로 나뉨
* 구성 방법 : 활동-대기(Active-Stanby) 방법
  + 한 DB가 활성 상태로 서비스하고 있으면 다른 DB는 대기하다가 장애 발생하면 대신 수행

: 활동-활동(Active-Active) 방법

* + 두 개의 DB가 서로 다른 서비스를 제공하다가 둘 중 하나가 문제면 나머지가 대신 수행

### 클러스터링(Clustering)

* 두 대 이상의 서버를 하나의 서버처럼 운영하는 기술
* 고가용성 클러스터링 : 하나의 서버에 장애가 발생하면 다른 노드(서버)가 받아 처리하여 서비스 중단을 방지
* 병렬 처리 클러스터링 : 하나의 작업을 여러 개의 서버에 분산하여 처리하는 방식

### RTO/PRO

* RTO(Recovery Time Objective, 목표 복구 시간)

: 비상사태 또는 업무 중단 시점으로부터 복구되어 가동될 때까지의 소요 시간

* RPO(Recovery Point Objective, 목표 복구 시점)

: 비상사태 또는 업무 중단 시점으로부터 데이터를 복구할 수 있는 기준점

## 데이터베이스 보안

### 데이터베이스 보안

* 권한이 없는 사용자가 액세스하는 것을 금지하기 위해 사용되는 기술

### 암호화(Encryption)

* 평문을 암호문으로 변환하는 것
* 암호화(Encryption) 과정, 복호화(Decryption) 과정
* 암호화 기법 : 개인키 암호 방식(Private Key Encryption), 공개키 암호 방식(Public Key Encryption)

### 접근통제

* 데이터가 저장된 객체와 이를 사용하려는 주체 사이의 정보 흐름을 제한하는 것
* 접근통제 3요소 : 접근통제 정책, 접근통제 메커니즘, 접근통제 보안모델
* 접근통제 기술

: 임의 접근통제(DAC, Discretionary Acess Control)

* + 데이터에 접근하는 사용자의 신원에 따라 접근 권한을 부여하는 방식

: 강제 접근통제(MAC, Mandatory Access Control)

* + 주체와 객체의 등급을 비교하여 접근 권한을 부여하는 방식

: 역할기반 접근통제(RBAC, Role Based Access Control)

* + 사용자의 역할에 따라 접근 권한을 부여하는 방식

### 접근통제 정책

* 어떤 주체가 언제, 어디서, 어떤 객체에게, 어떤 행위에 대한 허용 여부를 정의하는 것

### 접근통제 종류

* 신분 기반 정책

: 주체나 그룹의 신분에 근거하여 객체의 접근을 제한하는 방법

: IBP(Individual-Based Policy) - 최소 권한 정책으로, 단일 주체에게 하나의 객체에 대한 허가를 부여함

: GBP(Group-Based Policy) - 복수 주체에 하나의 객체에 대한 허가를 부여함

* 규칙 기반 정책

: 주체가 갖는 권한에 근거하여 객체의 접근을 제한하는 방법

: MLP(Multi-Level Policy) - 사용자나 객체별로 지정된 기밀 분류에 따른 정책

: CBP(Compartment-Based Policy) - 집단별로 지정된 기밀 허가에 따른 정책

* 역할 기반 정책

: GBP의 변형된 정책으로, 주체의 신분이 아니라 주체가 맡은 역할에 근거하여 객체의 접근을 제한

### 접근통제 매커니즘

* 정의된 접근통제 정책을 구현하는 기술적인 방법

### 접근통제 보안 모델

* 보안 정첵을 구현하기 위한 정형화된 모델
* 기밀성 모델 : 군사적인 목적으로 개발된 최초의 수학적 모델
* 무결성 모델 : 기밀성 모델에서 발생하는 불법적인 정보 번경을 방지하기 위해 무결성을 기반으로 개발된 모델
* 접근통제 모델 : 접근통제 매커니즘을 보안 모델로 발전시킨 것

### 접근통제 조건

* 접근통제 매커니즘의 취약점을 보완하기 위해 접근통제 정책에 부가하여 적용할 수 있는 조건
* 값 종속 통제(Value-Dependent Control) : 객체에 저장된 값에 따라 다르게
* 다중 사용자 통제(Multi-User Control) : 다수의 사용자가 동시에 접근을 요구하는 경우
* 컨텍스트 기반 통제(Context-Based Control)

### 감사 추적

* 사용자나 애플리케이션이 데이터베이스에 접근하여 수행한 모든 활동을 기록하는 기능

## 데이터베이스 백업

### 데이터베이스 백업

* 데이터베이스에 저장된 데이터를 보호하고 복구하기 위한 작업

### 로그 파일

* 데이터베이스의 상태 변화를 시간의 흐름에 따라 모두 기록한 파일

### 데이터베이스 복구 알고리즘

* NO-UNDO/REDO : 데이터베이스 버퍼의 내용을 비동기적으로 갱신한 경우의 복구 알고리즘(Commit했을 때)
* UNDO/NO-REDO : 데이터베이스 버퍼의 내용을 동기적으로 갱신한 경우의 복구 알고리즘(중간중간 저장)
* UNDO/REDO : 데이터베이스 버퍼의 내용을 동기/비동기적으로 갱신한 경우의 복구 알고리즘
* NO-UNDO/NO-REDO : 데이터베이스 버퍼의 내용을 동기적으로 저장 매체에 기록하지만 데이터베이스와는 다른 영역에 기록한 경우의 복구 알고리즘

### 백업 종류

* 물리 백업 : 데이터베이스 파일을 백업
* 논리 백업 : DB내의 논리적 객체들을 백업하는 방법

## 스토리지

### 1) 스토리지(Storage)

* 대용량의 데이터를 저장하기 위해 서버와 저장장치를 연결하는 기술
* 종류 : DAS(Direct Attached Storage)
  + 서버와 저장장치를 전용 케이블로 직접 연결하는 방식(외장하드)

: NAS(Network Attached Storage)

* + 서버와 저장장치를 네트워크를 통해 연결하는 방식

: SAN(Storage Area Network)

* + 서버와 저장장치를 연결하는 전용 네트워크를 별도로 구성하는 방식

## 논리 데이터 모델의 변환

* 엔티티(Entity)를 테이블로 변환
* 슈퍼타입/서브타입을 테이블로 변환
* 슈퍼타입 기준 테이블 변환
* 서브타입 기준 테이블 변환
* 개별타입 기준 테이블 변환
* 속성을 컬럼으로 변환
* 관계를 외래키로 변환

## 물리 데이터 모델 품질 검토

* 기준 : 정확성, 완전성, 준거성, 최신성, 일관성, 활용성

## 자료 구조

* 자료를 기억장치의 공간 내에 저장하는 방법과 자료 간의 관계, 처리 방법 등을 연구 분석하는 것

### 선형 구조(Linear Structure)

* + 배열(Array)

: 크기와 형(Type)이 동일한 자료들이 순서대로 나열된 자료의 집합

* + 연속 리스트(Contiguous List)

: 연속되는 기억장소에 저장되는 자료 구조(데이터 삽입을 위해서는 연속된 빈 공간이 있어야 함)

* + 연결 리스트(Linked List)

: 자료들을 임의의 기억공간에 기억시키되, 노드의 포인터 부분을 이용하여 서로를 연결시킨 구조

* + 스택(Stack)

: 리스트의 한쪽 끝으로만 자료의 삽입, 삭제 작업이 이루어지는 자료 구조(LIFO)

* + 큐(Queue)

: 리스트의 한쪽에서는 삽입 작업이 이루어지고 다른 한쪽에서는 삭제 작업이 이루어지는 구조(FIFO)

* + 데크(Deque)

: 양쪽에서 삽입, 삭제 가능

### 비선형 구조(Non-Linear Structure)

* + 그래프(Graph)

: 정점(Vertex)와 간선(Edge)의 두 집합으로 이루어지는 자료 구조

: 사이클이 없는 그래프를 트리(Tree)라고 함

: 방향 그래프의 최대 간선 수 = n(n-1)

: 무방향 그래프의 최대 간선 수 = n(n-1)/2

* + 트리(Tree)

: 정점(Node)과 선분(Branch)을 이용하여 사이클을 이루지 않도록 구성한 그래프의 특수한 형태

: 노드(Node)

* + - 트리의 기본 요소로서 자료 항목과 다른 항목에 대한 가지를 합친 것

: 근 노드(Root Node)

* + - 트리의 맨 위에 있는 노드

: 디그리(Degree, 차수)

* + - 각 노드에서 뻗어나온 가지의 수

: 단말 노드(Terminal Node) = 잎 노드(Leaf Node)

* + - 자식이 하나도 없는 노드, 즉 Degree가 0인 노드

: 비단말 노드(Non-Terminal Node)

* + - 자식이 하나라도 있는 노드

: 조상 노드(Ancestors Node)

* + - 임의의 노드에서 근 노드에 이르는 경로상에 있는 노드들

: 자식 노드(Son Node)

* + - 어떤 노드에 연결된 다음 레벨의 노드들

: 부모 노드(Parent Node)

* + - 어떤 노드에 연결된 이전 레벨의 노드들

: 형제 노드(Brother Node, Sibling)

* + - 동일한 부모를 갖는 노드들

: Level

: 깊이(Depth, Height)

* + - Tree에서 노드가 가질 수 있는 최대의 레벨

: 숲(Forest)

* + - 여러 개의 트리가 모여 있는 것

: 트리의 디그리

* + - 노드들의 디그리 중에서 가장 많은 수

: 이진 트리

* + - 차수(Degree)가 2 이하의 노드들로 구성된 트리
    - 트리의 운행법

: Preorder 운행 - Root -> Left -> Right

: Inorder 운행 - Left -> Root -> Right

: Postorder 운행 - Left -> Right -> Root

* + - 수식의 표기법

: 전위 표기법(PreFix) - 연산자 -> Left -> Right, +AB

: 중위 표기법(InFix) - Left -> 연산자 -> Right, A+B

: 후위 표기법(PostFix) - Left -> Right -> 연산자, AB+

## 정렬(Sort)

### 삽입 정렬(Insertion Sort)

* 이미 순서화된 파일에 새로운 하나의 레코드를 순서에 맞게 삽입시켜 정렬하는 방식

### 선택 정렬(Selection Sort)

* 최솟값을 찾아 첫 번째 레코드 위치에 놓고, 다시 최솟값을 찾아 두 번째 레코드 위치에 놓는 방식을 반복

### 버블 정렬(Burbble Sort)

* 인접한 두 개의 레코드 키값을 비교하여 그 크기에 따라 레코드 위치를 서로 교환하는 정렬 방식

### 쉘 정렬(Shell Sort)

* 매개변수의 값으로 서브 파일을 구성하고, 각 서브 파일을 Insertion 정렬 방식으로 순서 배열

### 퀵 정렬(Quick Sort)

* 키를 기준으로 작은 값은 왼쪽, 큰 값은 오른쪽 서브 파일에 분해시키는 정렬 방식

### 힙 정렬(Heap Sort)

* 전이진 트리를 이용한 정렬 방식

### 2-Way 합병 정렬(Merge Sort)

* 이미 정렬되어 있는 두 개의 파일을 한 개의 파일로 합병하는 정렬 방식

### 기수 정렬(Radix Sort) = Bucket Sort

* Queue를 이용하여 자릿수(Dight)별로 정렬하는 방식



# 03장. 통합 구현

## 통합 구현

### 통합 구현

* 송, 수신 모듈과 중계 모듈 간의 연계를 구현하는 것

### 통합 구현의 구성요소

* 송신 시스템과 송신 모듈
* 수신 시스템과 수신 모듈
* 중계 시스템
* 연계 데이터
* 네트워크

### 연계 요구사항 분석

* 연계 데이터를 식별 및 표준화하여 연계 데이터를 정의하는 것

### 연계 데이터 식별 및 표준화의 절차

① 연계범위 및 항목 정의

② 연계 코드 변환 및 매핑

③ 연계 데이터 식별자와 변경 구분 추가

④ 연계 데이터 표현 방법 정의

⑤ 연계 정의서 및 명세서 작성

## XML(eXtensible Markup Language)

### XML(eXtensible Markup Language)

* 특수한 목적을 갖는 마크업 언어를 만드는 데 사용되는 다목적 마크업 언어

### SOAP(Simple Object Access Protocol)

* 네트워크 상에서 HTTP/HTTPS, SMTP 등을 이용하여 XML을 교환하기 위한 통신 규약

### WSDL(Web Services Description Language)

* 웹 서비스와 관련된 서식이나 프로토콜 등을 표준적인 방법으로 기술하고 게시하기 위한 언어
* XML로 작성되며 UDDI의 기초가 됨

## 연계 테스트

* 연계 시스템과 연계 시스템의 구성요소가 정상적으로 동작하는지 확인하는 활동
* 케이스 작성 -> 환경 구축 -> 수행 -> 수행 결과 검증

# 04장. 서버 프로그램 구현

## 개발 환경 구축

### 개발 환경 구축

* 개발 프로젝트를 이해하고 소프트웨어 및 하드웨어 장비를 구축하는 것

### 하드웨어 환경

* 사용자와의 인터페이스 역할을 하는 클라이언트(Client), 클라이언트와 통신하여 서비스를 제공하는 서버(Server)로 구성됨
* 서버의 종류 : 웹 서버, 웹 애플리케이션 서버(WAS), DB 서버, 파일 서버

: WAS는 웹 서버와 DB 서버 또는 웹 서버와 파일 서버 사이에서 인터페이스 역할을 수행함

### 소프트웨어 환경

* 클라이언트와 서버 운영을 위한 시스템 소프트웨어와 개발에 사용되는 개발 소프트웨어로 구성됨
* 시스템 소프트웨어의 종류 : OS, 웹 서버 및 WAS 운용을 위한 서버 프로그램, DBMS 등
* 개발 소프트웨어의 종류 : 요구사항 관리 도구, 설계/모델링 도구, 구현 도구, 빌드 도구, 테스트 도구,

형상 관리 도구

### 웹 서버의 기능

* HTTP/HTTPS 지원, 통신 기록, 정적 파일 관리, 대역폭 제한, 가상 호스팅, 인증

### 개발 언어 선정 기준

* 적정성, 효율성, 이식성, 친밀성, 범용성

## 소프트웨어 아키텍처

### 소프트웨어 아키텍처

* 소프트웨어를 구성하는 요소들 간의 관계를 표현하는 시스템의 구조 또는 구조체

### 모듈화(Modularity)

* 시스템의 기능들을 모듈(특정 기능을 처리할 수 있는 소스 코드) 단위로 나누는 것

### 추상화(Abstraction)

* 전체적이고 포괄적인 개념을 설계한 후 구체화 시켜 나가는 것
* 과정 추상화(전반적인 흐름 파악), 데이터 추상화(데이터 구조를 대표), 제어 추상화(이벤트 발생 대표)

### 단계적 분해(Stepwise Refinement)

* 상위의 중요 개념으로부터 하위의 개념으로 구체화 시키는 분할 기법

### 정보 은닉(Information Hiding)

* 모듈 내부에 포함된 절차와 자료들의 정보가 감추어져 다른 모듈이 접근하거나 변경하지 못하도록 하는 기법

### 상위 설계와 하위 설계

|  |  |  |
| --- | --- | --- |
|  | 상위 설계 | 하위 설계 |
| 별칭 | 아키텍처 설계, 예비 설계 | 모듈 설계, 상세 설계 |
| 설계 대상 | 시스템의 전체적인 구조 | 시스템의 내부 구조 및 행위 |
| 세부 목록 | 구조, DB, 인터페이스 | 컴포넌트, 자료 구조, 알고리즘 |

1. **소프트웨어 아키텍처의 품질 속성**

* 품질 평가 요소들을 구체화 시켜 놓은 것
* 품질 평가 요소들의 종류 : 시스템 측면, 비즈니스 측면, 아키텍처 측면

### 소프트웨어 아키텍처 설계 과정

① 설계 목표 설정

② 시스템 타입 설정

③ 아키텍처 패턴 적용

④ 서브 시스템 구체화

⑤ 검토

### 협약(Contract)에 의한 설계

* 컴포넌트를 설계할 때 클래스에 대한 여러 가정을 공유할 수 있도록 명세한 것
* 명세에 포함될 조건 : 선행 조건 - 호출되기 전에 참

: 결과 조건 - 수행된 후 만족

: 불변 조건 - 실행되는 동안 항상 만족

## 아키텍처 패턴

### 아키텍처 패턴

* 아키텍처를 설계할 때 참조할 수 있는 전형적인 해결 방식 또는 예제

### 레이어 패턴(Layers Pattern)

* 시스템을 계층으로 구분하여 구성하는 고전적인 방법의 패턴

### 클라이언트-서버 패턴(Client-Server Pattern)

* 하나의 서버 컴포넌트와 다수의 클라이언트 컴포넌트로 구성되는 패턴

### 파이프-필터 패턴(Pipe-Filter Pattern)

* 데이터 스트림 절차의 각 단계를 필터로 캡슐화하여 파이프를 통해 전송하는 패턴

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

* 서브시스템을 모델, 뷰, 컨트롤러로 구조화하는 패턴

1. **기타 패턴**

|  |  |
| --- | --- |
| Master-Slave Pattern | 슬레이브 컴포넌트에서 처리된 결과물을 다시 돌려받는 방식으로 작업을 수행하는  패턴 |
| Broker-Pattern | 사용자가 원하는 서비스와 특성을 브로커 컴포넌트에 요청하면 브로커 컴포넌트가  요청에 맞는 컴포넌트와 사용자를 연결해주는 패턴 |
| Peer-To-Peer Pattern(P2P) | 피어(Peer)라 불리는 하나의 컴포넌트가 클라이언트가 될 수도, 서버가 될 수도 있  는 패턴 |
| Event-Bus Pattern | 소스가 특정 채널에 이벤트 메시지를 발행하면 해당 채널을 구독한 리스너들이 메시  지를 받아 이벤트를 처리하는 패턴 |
| Blackboard Pattern | 모든 컴포넌트들이 고유 데이터 저장소와 블랙보드 컴포넌트에 접근이 가능한 패턴 |
| Interpreter Pattern | 프로그램 코드의 각 라인을 수행하는 방법을 지정하고, 기호마다 클래스를 갖도록  구성된 패턴 |

## 객체지향(Object-Oriented)

### 객체지향

* 각 요소들을 객체(Object)로 만든 후, 객체들을 조립해서 소프트웨어를 개발하는 기법

### 객체지향의 구성요소

* 객체(Object) : 데이터와 이를 처리하기 위한 함수를 묶어놓은 소프트웨어 모듈
* 클래스(Class) : 공통된 속성과 연산을 갖는 객체의 집합
* 메시지(Message) : 객체들 간의 상호작용에 사용되는 수단으로, 객체의 동작이나 연산을 일으키는 외부의 요 구사항

### 객체지향의 특징

* 캡슐화(Encapsulation) : 외부에서의 접근을 제한하기 위해 인터페이스를 제외한 세부 내용을 은닉하는 것
* 상속(Inheritance) : 상위 클래스의 모든 속성과 연산을 하위 클래스가 물려받는 것
* 다형성(Polymorphism) : 하나의 메시지에 대해 각각의 객체가 가지고 있는 고유한 방법으로 응답할 수 있는 능력
* 연관성(Relationship) : 두 개 이상의 객체들이 상호 참조하는 관계

: is member of - 연관화

: is instance of - 분류화

: is part of - 집단화

: is a - 일반화, 특수화/상세화

## 객체지향 분석 및 설계

### 객체지향 분석(OOA; Object Oriented Analysis)

* 사용자의 요구사항과 관련된 객체, 속성, 연산, 관계 등을 정의하여 모델링하는 작업

### 객체지향 분석의 방법론

* Rumbaugh 방법 : 분석 활동을 객체 모델, 동적 모델, 기능 모델로 나누어 수행함
* Booch 방법 : 클래스와 객체들을 분석 및 식별하고 클래스의 속성과 연산을 정의함
* Jacobson 방법 : 유스케이스를 강조하여 사용함
* Coad와 Yourdon 방법 : E-R 다이어그램을 사용하여 객체의 행위를 모델링함
* Wirfs-Brock 방법 : 분석과 설계 간의 구분이 없고, 고객 명세서를 평가해서 설계 작업까지 연속적으로 수행함

### 럼바우(Rumbaugh) 분석 기법

* 모든 소프트웨어 구성 요소를 그래픽 표기법으로 이용하여 모델링하는 기법
* 분석 활동 순서 : 객체 모델링 -> 동적 모델링 -> 기능 모델링

### 객체지향 설계 원칙

* 변경이나 확장에 유연한 시스템을 설계하기 위해 지켜져야 할 원칙
* 단일 책임 원칙(SRP) : 객체는 단 하나의 책임만 가져야 함
* 개방-폐쇄 원칙(OCP) : 기존의 코드를 변경하지 않고 기능을 추가할 수 있도록 설계해야 함
* 리스코프 치환 원칙(LSP) : 자식 클래스는 최소한 부모 클래스의 기능은 수행할 수 있어야 함
* 인터페이스 분리 원칙(ISP) : 자신이 사용하지 않는 인터페이스와 의존 관계를 맺거나 영향을 받지 않아야 함
* 의존 역전 원칙(DIP) : 의존 관계 성립 시 추상성이 높은 클래스와 의존 관계를 맺어야 함

## 모듈

### 모듈(Module)

* 모듈화를 통해 분리된 시스템의 각 기능
* 모듈의 독립성은 결합도와 응집도에 의해 측정됨

### 결합도(Coupling)

* 모듈 간에 상호 의존하는 정도

|  |  |
| --- | --- |
| 내용 결합도  (Content Coupling) | 한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 직접 참조하거나 수정할 때 |
| 공통(공유) 결합도  (Common Coupling) | 공유되는 공통 데이터 영역을 여러 모듈이 사용할 때 |
| 외부 결합도  (External Coupling) | 어떤 모듈에서 선언한 데이터(변수)를 외부의 다른 모듈에서 참조할 때 |
| 제어 결합도  (Control Coupling) | 어떤 모듈이 다른 모듈 내부의 논리적인 흐름을 제어하기 위해 제어 신호나 제어 요소를  전달하는 결합도 |
| 스탬프 결합도  (Stamp Coupling) | 모듈 간의 인터페이스로 배열이나 레코드 등의 자료 구조가 전달될 때의 결합도 |
| 자료 결합도  (Data Coupling) | 모듈 간의 인터페이스가 자료 요소로만 구성될 때의 결합도 |

### 응집도(Cohesion)

* 모듈의 내부 요소들이 서로 관련되어있는 정도

|  |  |
| --- | --- |
| 기능적 응집도  (Functional Cohesion) | 모듈 내부의 모든 기능 요소들이 단일 문제와 연관되어 수행될 경우 |
| 순차적 응집도  (Sequential Cohesion) | 모듈 내 하나의 활동으로부터 나온 출력 데이터를 그 다음 활동의 입력 데이터로  사용할 경우 |
| 교환(통신)적 응집도  (Communication Cohesion) | 동일한 입력과 출력을 사용하여 서로 다른 기능을 수행하는 구성요소들이 모였을  경우 |

|  |  |
| --- | --- |
| 절차적 응집도  (Procedural Cohesion) | 모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성 요소들이 그 기능을 순차적으  로 수행할 경우 |
| 시간적 응집도  (Temporal Cohesion) | 특정 시간에 처리되는 몇 개의 기능을 모아 하나의 모듈로 작성할 경우 |
| 논리적 응집도  (Logical Cohesion) | 유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들로 하나의 모듈이 형성되  는 경우 |
| 우연적 응집도  (Coincidental Cohesion) | 모듈 내부의 각 구성 요소들이 서로 관련 없는 요소로만 구성된 경우 |

### 팬인(Fan-In)/팬아웃(Fan-Out)

* 팬인 : 어떤 모듈을 제어하는 모듈의 수
* 팬아웃: 어떤 모듈에 의해 제어되는 모듈의 수

### N-S 차트(Nassi-Schneiderman Chart)

* 논리의 기술에 중점을 두고 도형을 이용해 표현하는 방법

## 단위 모듈

### 단위 모듈(Unit Module)

* 한 가지 동작을 수행하는 기능을 모듈로 구현한 것
* 단위 모듈 구현 과정 : 단위 기능 명세서 작성 -> 입, 출력 기능 구현 -> 알고리즘 구현

### IPC(Inter-Process Communication)

* 모듈 간 통신 방식을 구현하기 위해 사용되는 대표적인 프로그래밍 인터페이스 집합
* IPC의 대표 메소드 5가지

: Shared Memory, Socket, Semaphores, Pipes&named Pipes, Message Queueing

### 단위 모듈 테스트(단위 테스트)

* 모듈이 정해진 기능을 정확히 수행하는지 검증하는 것

### 테스트 케이스(Test Case)

* 소프트웨어가 사용자의 요구사항을 정확하게 준수했는지를 확인하기 위한 테스트 항목에 대한 명세서

## 공통 모듈

### 공통 모듈

* 여러 프로그램에서 공통으로 사용할 수 있는 모듈

### 공통 모듈 명세 기법의 종류

* 정확성(Correctness) : 시스템 구현 시 해당 기능이 필요하다는 것을 알 수 있도록 정확히 작성함
* 명확성(Clarity) : 해당 기능을 이해할 때 중의적으로 해석되지 않도록 명확하게 작성함
* 완전성(Completeness) : 시스템 구현을 위해 필요한 모든 것을 기술함
* 일관성(Consistency) : 공통 기능들 간 상호 충돌이 발생하지 않도록 작성함
* 추적성(Traceability) : 기능에 대한 요구사항의 출처, 관련 시스템 등의 관계를 파악할 수 있도록 작성함

### 재사용(Reuse)

* 이미 개발된 기능들을 새로운 시스템이나 기능 개발에 사용하기 적합하도록 최적화하는 작업
* 재사용 규모에 따른 분류 : 함수와 객체, 컴포넌트, 애플리케이션

### 효과적인 모듈 설계 방안

* 결합도↓ 응집도↑ -> 모듈의 독립성↑ 재사용성↑
* 복잡도↓ 중복성↓ 일관성 유지

## 코드

### 코드(Code)

* 자료의 분류, 조합, 집계, 추출을 용이하게 하기 위해 사용하는 기호
* 주요 기능 : 식별(성격 구분), 분류(데이터 그룹화), 배열, 표준화, 간소화

1. **코드의 종류**

|  |  |
| --- | --- |
| 순차 코드  (Sequence Code) | 일정 기준에 따라서 최초의 자료부터 차례로 일련번호를 부여 |
| 블록 코드  (Block Code) | 공통성이 있는 것끼리 블록으로 구분하고, 각 블록 내에서 일련번호를 부여 |
| 10진 코드  (Decimal Code) | 코드화 대상 항목을 0~9까지 10진 분할하고, 다시 그 각각에 대해 10진 분할을 필  요한 만큼 반복하는 방법, 도서 분류식 코드라고도 함 |
| 그룹 분류 코드  (Group Classification Code) | 일정 기준에 따라 대분류, 중분류, 소분류 등으로 구분하고 각 그룹 안에서 일련번  호를 부여 |
| 연상 코드  (Mnemonic Code) | 코드화 대상 항목의 명칭이나 약호와 관계있는 숫자나 문자, 기호를 이용하여 코드  부여 |
| 표의 숫자 코드  (Significant Digit Code) | 코드화 대상 항목의 성질을 물리적 수치 그대로 코드에 적용시키는 방법으로 유효  숫자 코드라고도 함 |
| 합성 코드  (Combined Code) | 필요한 기능을 하나의 코드로 수행하기 어려운 경우 2개 이상의 코드를 조합하여  만드는 방법 |

## 디자인 패턴

### 디자인 패턴(Design Pattern)

* 모듈 간의 관계 및 인터페이스를 설계할 때 참조할 수 있는 전형적인 해결 방식 또는 예제
* 아키텍처 패턴은 건물의 윤곽을 잡는 가이드라인, 디자인 패턴은 그보다 더 세세한 인테리어

### 생성 패턴(Creational Pattern)

* 클래스나 객체의 생성과 참조 과정을 정의하는 패턴

|  |  |
| --- | --- |
| 추상 팩토리  (Abstract Factory) | 구체적인 클래스에 의존하지 않고 인터페이스를 통해 서로 연관, 의존하는 객체들의 그룹으  로 생성하여 추상적으로 표현함 |
| 빌더(Builder) | 작게 분리된 인스턴스를 건축하듯이 조합하여 객체를 생성함 |
| 팩토리 메소드  (Factory Method) | 객체 생성을 서브 클래스에서 처리하도록 분리하여 캡슐화한 패턴 |
| 프로토타입  (Prototype) | 원본 객체를 복제하는 방법으로 객체를 생성하는 패턴 |
| 싱글톤(Singleton) | 하나의 객체를 생성하면 생성된 객체를 어디서든 참조할 수 있지만, 여러 프로세스가 동시에  참조할 수는 없음 |

### 구조 패턴(Structural Pattern)

* 클래스나 객체들을 조합하여 더 큰 구조로 만드는 패턴

|  |  |
| --- | --- |
| 어댑터(Adapter) | 호환성이 없는 클래스들의 인터페이스를 다른 클래스가 이용할 수 있도록 변환해줌 |
| 브리지(Bridge) | 구현부에서 추상층을 분리하여, 서로가 독립적으로 확장할 수 있도록 구성한 패턴 |
| 컴포지트(Composite) | 여러 객체를 가진 복합 객체와 단일 객체를 구분 없이 다루고자 할 때 사용하는 패턴 |
| 데코레이터(Decorator) | 객체 간의 결합을 통해 능동적으로 기능들을 확장할 수 있는 패턴 |
| 퍼싸드(Facade) | 복잡한 서브 클래스들을 피해 더 상위에 인터페이스를 구성함으로써 서브 클래스들의 기  능을 간편하게 사용할 수 있도록 하는 패턴 |
| 플라이웨이트  (Flyweight) | 인스턴스가 필요할 때마다 매번 생성하는 것이 아니고 가능한 공유해서 사용함으로써 메  모리를 절약하는 패턴 |
| 프록시(Proxy) | 접근이 어려운 객체와 여기에 연결하려는 객체 사이에서 인터페이스 역할을 수행함 |

### 행위 패턴(Behavioral Pattern)

* 클래스나 객체들이 서로 상호작용하는 방법이나 책임 분배 방법을 정의하는 패턴

|  |  |
| --- | --- |
| 책임 연쇄  (Chain of Responsibility) | 요청을 처리할 수 있는 객체가 둘 이상 존재하여 한 객체가 처리하지 못하면 다음 객  체로 넘어가는 형태의 패턴 |
| 커맨드(Command) | 요청을 객체의 형태로 캡슐화하여 재이용하거나 취소할 수 있도록 요청에 필요한 정  보를 저장하거나 로그에 남기는 패턴 |
| 인터프리터(Interpreter) | 언어에 문법 표현을 정의하는 패턴 |
| 반복자(Iterator) | 자료 구조와 같이 접근이 잦은 객체에 대해 동일한 인터페이스를 사용하도록 함 |
| 중재자(Mediator) | 수많은 객체들 간의 복잡한 상호작용을 캡슐화하여 객체로 정의하는 패턴 |
| 메멘토(Memento) | 특정 시점에서의 객체 내부 상태를 객체화함으로써 이후 요청에 따라 객체를 해당 시  점의 상태로 돌릴 수 있는 기능을 제공하는 패턴 |
| 옵서버(Observer) | 한 객체의 상태가 변화하면 객체에 상속되어 있는 다른 객체들에게 변화된 상태를 전  달하는 패턴 |
| 상태(State) | 객체의 상태에 따라 동일한 동작을 다르게 처리해야 할 때 사용하는 패턴 |
| 전략(Strategy) | 동일한 계열의 알고리즘들을 개별적으로 캡슐화하여 상호 교환할 수 있게 정의함 |
| 템플릿 메소드  (Template Method) | 상위 클래스에서 골격을 정의하고, 하위 클래스에서 세부 처리를 구체화하는 구조 |
| 방문자(Visitor) | 각 클래스들의 데이터 구조에서 처리 기능을 분리하여 별도의 클래스로 구성 |

## 개발 지원 도구

### 통합 개발 환경(IDE; Integrated Development Environment)

* 개발에 필요한 다양한 툴을 하나의 인터페이스로 통합하여 제공하는 환경
* 종류 : Eclipse, Visual Studio, Xcode, Android Studio, IDEA

### 빌드 도구

* 빌드 : 소스 코드 파일들을 컴퓨터에서 실행할 수 있는 제품 소프트웨어로 변환하는 과정 또는 결과물
* Ant(Another Neat Tool) : 아파치에서 개발한 자바 프로젝트의 공식적인 빌드 도구
* Maven : 아파치에서 Ant의 대안으로 개발, 의존성을 설정해 라이브러리 관리
* Gradle : 한스 도커가 Ant와 Maven을 보완하여 개발, 안드로이드 스튜디오의 공식 빌드 도구

: 그루비 기반의 빌드 스크립트를 사용

### 협업 도구

* 개발에 참여하는 사람들이 서로 다른 작업 환경에서 원활히 프로젝트를 수행할 수 있도록 도와주는 도구

## 서버 개발

### 서버 개발

* 웹 애플리케이션의 로직을 구현할 서버 프로그램을 제작하여 WAS에 탑재하는 것

### 서버 개발 프레임워크

* 다양한 네트워크 설정, 요청 및 응답 처리, 아키텍처 모델 구현 등을 손쉽게 처리할 수 있도록 클래스나 인터페이스 를 제공하는 소프트웨어
* Spring : JAVA를 기반으로 만든 프레임워크, 전자정부 표준 프레임워크의 기반 기술
* Node.js : JavaScript를 기반으로 만든 프레임워크
* Django : Python을 기반으로 만든 프레임워크
* Codeigniter : PHP를 기반으로 만든 프레임워크
* Ruby on Rails: Ruby를 기반으로 만든 프레임워크

### 서버 개발 과정

* DTO/VO, SQL, DAO, Service, COntroller를 각각 구현하는 과정
* DTO/VO 구현 : 데이터 교환을 위해 사용할 객체를 만듦
* SQL 구현 : 데이터의 삽입, 변경, 삭제 등의 작업을 수행할 SQL문을 생성
* DAO 구현 : 데이터베이스에 접근하고, SQL을 활용하여 데이터를 실제로 조작하는 코드를 구현
* Service 구현 : 사용자의 요청에 응답하기 위한 로직을 구현
* Controller 구현 : 사용자의 요청에 적절한 서비스를 호출하여 그 결과를 사용자에게 반환하는 코드를 구현

## 보안 및 API

### 소프트웨어 개발 보안

* 보안 취약점을 최소화하여 보안 위협으로부터 안전한 소프트웨어를 개발하기 위한 일련의 보안 활동
* 소프트웨어 개발 보안 점검 항목

: 세션 통제, 입력 데이터 검증 및 표현, 보안 기능, 시간 및 상태, 에러처리, 코드 오류, 캡슐화, API 오용

### API(Application Programming Interface)

* 라이브러리를 이용할 수 있도록 규칙 등을 정의해놓은 인터페이스

## 배치 프로그램

### 배치 프로그램(Batch Program)

* 여러 작업들을 미리 정해진 일련의 순서에 따라 일괄적으로 처리하도록 만든 프로그램
* 필수 요소 : 대용량 데이터 - 대량의 데이터를 처리할 수 있어야 함

: 자동화 - 사용자의 개입 없이 수행되어야 함

: 견고성 - 중단되는 일이 없어야 함

: 안전성/신뢰성 - 오류가 발생하면 오류의 발생 위치, 시간 등을 추적할 수 있어야 함

: 성능 - 다른 응용 프로그램의 수행을 방해하지 않아야 함

* + 지정된 시간 내에 처리가 완료되어야 함

### 배치 스케줄러(Batch Scheduler)

* 일괄 처리(Batch Processing) 작업이 설정된 주기에 맞춰 자동으로 수행되도록 지원해주는 도구
* 종류 : 스프링 배치, Quartz, Cron

## 패키지 소프트웨어

### 1) 패키지 소프트웨어(Package Software)

* 기업에서 일반적으로 사용하는 여러 기능을 통합하여 제공하는 소프트웨어(일반 시중에서 판매)
* 반대 개념은 전용 개발 소프트웨어라고 함

# 05장. 인터페이스 구현

## 시스템 인터페이스 요구사항 분석

### 시스템 인터페이스 요구사항

* 개발할 시스템과 외부 시스템을 연동하는 데 필요한 시스템 인터페이스에 대한 요구사항을 기술한 것

### 시스템 인터페이스 요구사항 분석

* 요구사항을 분류하고 구체적으로 명세한 후 이를 이해관계자에게 전달하는 일련의 과정

### 시스템 인터페이스 요구사항 분석 절차

* 요구사항 목록 작성 -> 자료 준비 -> 기능적, 비기능적으로 분류 -> 분석 후 내용 추가나 수정 -> 전달

## 인터페이스 요구사항 검증

### 인터페이스 요구사항 검증

* 수행 순서 : 요구사항 검토 계획 수립 -> 검토 및 오류 수정 -> 베이스라인 설정

### 요구사항 검증 방법

* 요구사항 검토(Requirements Review)

: 동료검토(Peer Review)

* + 요구사항 명세서 작성자가 직접 설명하고 동료들이 이를 들으면서 결함을 발견

: 워크스루(Walk Throuh)

* + 미리 요구사항 명세서를 사전 검토한 후에 짧은 검토 회의를 통해 결함 발견

: 인스펙션(Inspection)

* + 요구사항 명세서 작성자를 제외한 다른 전문가들이 명세서를 확인하며 결함 발견
* 프로토타이핑(Prototyping)

: 견본품(Prototype)을 만들어 최종 결과물을 예측

* 테스트 설계

: 테스트 케이스를 생성하여 요구사항이 현실적으로 테스트 가능한지를 검토

* CASE 도구 활용

: 일관성 분석을 통해 요구사항 변경사항의 추적, 분석, 관리, 표준 준수 여부를 확인

### 인터페이스 요구사항 검증의 주요 항목

* 완전성(Competeness) : 누락되지 않고 완전하게
* 일관성(Consistency) : 모순되거나 충돌되는 점 없이
* 명확성(Unambiguity) : 명확히 이해할 수 있는지
* 기능성(Functionality) : 어떻게 보다 무엇을
* 검증 가능성(Varifiability) : 요구사항과 일치하는지 검증 가능한지
* 추적 가능성(Traceability) : 요구사항 명세서와 설계서를 추적할 수 있는지
* 변경 용이성(Easily Changeable): 요구사항 명세서의 변경이 쉬운지

## 인터페이스 시스템 식별

* 인터페이스별로 인터페이스에 참여하는 시스템들을 송신 시스템과 수신 시스템으로 구분하여 작성하는 것

## 송 · 수신 시스템 식별

### 식별 대상 데이터

* 송 · 수신 시스템 사이에서 교환되는 데이터로, 규격화된 표준 형식에 따라 전송됨

### 인터페이스 표준 항목

* 송 · 수신 시스템을 연계하는 데 표준적으로 필요한 데이터
* 시스템 공통부와 거래 공통부로 나뉨

### 송 · 수신 데이터 항목

* 송 · 수신 시스템이 업무를 수행하는 데 사용하는 데이터

### 공통 코드

* 시스템들에서 공통으로 사용하는 코드

### 정보 흐름 식별

* 개발할 시스템과 내, 외부 시스템 사이에서 전송되는 정보들의 방향성 식별

### 송 · 수신 데이터 식별

* 개발할 시스템과 연계할 내, 외부 시스템 사이의 정보 흐름과 데이터베이스 산출물을 기반으로 식별

## 인터페이스 방법 명세화

### 인터페이스 방법 명세화

* 내, 외부 시스템이 연계하여 작동할 때 인터페이스별 송, 수신 방법, 송, 수신 데이터, 오류 식별 및 처리 방안에 대 한 내용을 문서로 정리하는 것

### 시스템 연계 기술

* DB LInk, API/Open API, 연계 솔루션, Socket, Web Service

### 인터페이스 통신 유형

* 단방향, 동기, 비동기

### 인터페이스 처리 유형

* 실시간 방식, 지연 처리 방식, 배치 방식

## 미들웨어 솔루션

### 미들웨어(Middleware)

* 운영체제와 응용 프로그램 사이에서 다양한 서비스를 제공하는 소프트웨어

### DB

* 클라이언트에서 원격의 데이터베이스와 연결하는 미들웨어

### RPC(원격 프로시저 호출)

* 원격 프로시저를 마치 로컬 프로시저(내 컴퓨터에 있는 것처럼)처럼 호출하는 미들웨어

### MOM(메시지 지향 미들웨어)

* 비동기형 메시지를 전달하는 미들웨어

### TP-Monitor(트랜젝션 처리 모니터)

* 트랜잭션을 처리 및 감시하는 미들웨어

### ORB(객체 요청 브로커)

* 코바(CORBA) 표준 스펙을 구현한 객체지향 미들웨어

### WAS(웹 애플리케이션 서버)

* 동적인 콘텐츠를 처리하기 위한 미들웨어

## 모듈 간 공통 기능 및 데이터 인터페이스 확인

### 모듈 간 공통 기능 및 데이터 인터페이스

* 모듈 간 공통 기능 : 모듈의 기능 중에서 공통적으로 제공되는 기능
* 데이터 인터페이스 : 모듈 간 교환되는 데이터가 저장될 파라미터

### 인터페이스 설계서

* 교환 데이터 및 관련 업무, 송, 수신 시스템 등에 대한 내용을 정의한 문서
* 종류 : 일반적인 인터페이스 설계서, 정적 · 동적 모형을 통한 인터페이스 설계서

## 모듈 연계를 위한 인터페이스 기능 식별

### 모듈 연계

* 내부 모듈과 외부 모듈 또는 내부 모듈 간 데이터의 교환을 위해 관계를 설정하는 것

### EAI(Enterprise Application Integration)

* 기업 내 각종 애플리케이션 및 플랫폼 간의 상호 연동이 가능하게 해주는 솔루션
* Point-to-Point

; 가장 기본적인 애플리케이션 통합 방식, 1:1 연결

* Hub & Spoke

: 단일 접점인 허브 시스템을 통해 데이터를 전송하는 중앙 집중형

* Message Bus(ESB 방식)

: 애플리케이션 사이에 미들웨어를 두어 처리하는 방식

* Hybrid

: Hub & Spoke와 Message Bus의 혼합 방식

: 그룹 내에서는 Hub & Spoke 방식을, 그룹 간에는 Message Bus 방식을 사용

### ESB(Enterprise Service Bus)

* 애플리케이션 간 표준 기반의 인터페이스를 제공하는 솔루션

### 웹 서비스(Web Service)

* 네트워크의 정보를 표준화된 서비스 형태로 만들어 공유하는 기술
* SOAP : HTTP, HTTPS, SMTP 등을 활용하여 XML 기반의 메시지를 네트워크 상에서 교환하는 프로토콜
* UDDI : WSDL을 등록하여 서비스와 서비스 제공자를 검색하고 접근하는 데 사용됨
* WSDL : 웹 서비스명, 서비스 제공 위치, 프로토콜 등 웹 서비스에 대한 상세 정보를 XML 형식으로 구현함

### 모듈 간 연계 기능 식별, 6) 모듈 간 인터페이스 기능 식별

- 책 참고(1-419P)

## 모듈 간 인터페이스 데이터 표준 확인

* 모듈 간 인터페이스에 사용되는 데이터 형식을 표준화하는 것
* 데이터 인터페이스 확인 -> 인터페이스 기능 확인 -> 인터페이스 데이터 표준 확인

## 인터페이스 기능 구현 정의

* 인터페이스를 실제로 구현하기 위해 인터페이스 기능에 대한 구현 방법을 기능별로 기술하는 것
* 컴포넌트 명세서 확인 -> 인터페이스 명세서 확인

-> 일관된 인터페이스 기능 구현 정의 -> 정의된 인터페이스 기능 구현 정형화

* 모듈 세부 설계서 : 모듈의 구성요소와 세부적인 동작 등을 정의한 설계서

: 컴포넌트 명세서, 인터페이스 명세서

## 인터페이스 구현

### 인터페이스 구현

* 송, 수신 시스템 간의 데이터 교환 및 처리를 실현해주는 작업

### 데이터 통신을 이용한 인터페이스 구현

* 애플리케이션 영역에서 데이터 포맷을 인터페이스 대상으로 전송하면 이를 수신 측에서 Parsing하여 해석하는 방식
* 주로 JSON이나 XML 형식의 데이터 포맷 사용

### 인터페이스 엔티티(개체)를 이용한 인터페이스 구현

* 인터페이스가 필요한 시스템 사이에 별도의 인터페이스 엔티티를 두어 상호 연계하는 것

### JSON(JavaScript Object Notation)

* 데이터 객체를 속성 · 값의 쌍(Attribute-Value Pairs) 형태로 표현하는 개방형 표준 포맷

### AJAX(Asynchronous JavaScript and XML)

* 클라이언트와 서버 간에 XML 데이터를 주고 받는 비동기 통신 기술

## 인터페이스 예외 처리

* 구현된 인터페이스가 동작하는 과정에서 기능상 예외 상황이 발생했을 때 이를 처리하는 절차
* 데이터 통신을 이용한 방법, 인터페이스 엔티티를 이용한 방법이 있음

## 인터페이스 보안

* 인터페이스의 보안 취약점을 분석한 후 적절한 보안 기능을 적용하는 것

### 인터페이스 보안 기능 적용

* 네트워크 영역 : 인터페이스 송, 수신 간 스니핑 등을 이용한 데이터 탈취 및 변조 위협을 방지하기 위해 네트워크 트래픽에 대한 암호화를 설정함
* 애플리케이션 영역 : 소프트웨어 개발 보안 가이드를 참조하여 애플리케이션 코드 상의 보안 취약점을 보완하는 방향으로 애플리케이션 보안 기능을 적용함
* 데이터베이스 영역 : 데이터베이스, 스키마, 엔티티의 접근 권한과 프로시저, 트리거 등 데이터베이스 동작 객체 의 보안 취약점에 보안 기능을 적용함

### 데이터 무결성 검사 도구

* 인터페이스 보안 취약점을 분석하는 데 사용되는 도구

## 인터페이스 구현 검증

* 인터페이스가 정상적으로 문제없이 작동하는지 확인하는 것

### 인터페이스 구현 검증 도구

|  |  |
| --- | --- |
| xUnit | Java(Junit), C++(Cppunit), Net(Nunit) 등 다양한 언어를 지원하는 단위 테스트 프레임워크 |
| STAF | 서비스 호출 및 컴포넌트 재사용 등 다양한 환경을 지원함 |
| FitNess | 웹 기반 테스트 케이스 설계, 실행, 결과 확인 등을 지원함 |
| NTAF | FitNess의 장점인 협업 기능과 STAF의 장점인 재사용 및 확장성을 통합한 NHN(Naver)의 테스트 자  동화 프레임워크 |
| Selenium | 다양한 브라우저 및 개발 언어를 지원하는 웹 애플리케이션 테스트 프레임워크 |
| watir | Ruby를 사용하는 애플리케이션 테스트 프레임워크 |

1. **인터페이스 구현 감시 도구**

* 인터페이스 동작 상태는 APM을 사용하여 감시할 수 있음

### APM(Application Performance Management/Monitoring, 애플리케이션 성능 관리)

* 애플리케이션의 성능 관리를 위해 다양한 모니터링 기능을 제공하는 도구
* 대표적인 APM : Scouter, Jennifer
* APM의 유형 : 리소스 방식, 엔드투엔드 방식

# 06장. 화면 설계

## 사용자 인터페이스

### 사용자 인터페이스(UI, User Interface)

* 사용자와 시스템 간의 상호작용이 원활하게 이루어지도록 도와주는 장치나 소프트웨어

### 사용자 인터페이스의 구분

|  |  |
| --- | --- |
| CLI(Command Line Interface) | 텍스트 형태(도스) |
| GUI(Graphic User Interface) | 아이콘이나 메뉴를 마우스로 선택하여 작업 수행 |
| NUI(Natural User Interface) | 사용자의 말이나 행동으로 기기를 조작 |

1. **사용자 인터페이스의 기본 원칙**

* 직관성 : 누구나 쉽게 이해하고 사용할 수 있어야 함
* 유효성 : 사용자의 목적을 정확하고 완벽하게 달성해야 함
* 학습성 : 누구나 쉽게 배우고 익힐 수 있어야 함
* 유연성 : 사용자의 요구사항을 최대한 수용하고 실수를 최소화해야 함

## UI 표준 지침

* UI 표준 : 전체 시스템에 포함된 모든 UI에 공통적으로 적용될 내용
* UI 지침 : UI 개발 과정에서 꼭 지켜야 할 공통의 조건
* UI 스타일 가이드 : 개발자나 디자이너들이 UI를 작성할 때 기준이 되는 규칙들

: 구동 환경 정의 -> 레이아웃 정의 -> 네비게이션 정의 -> 기능 정의 -> 구성 요소 정의

## UI 설계 도구

### 와이어프레임(Wireframe)

* 페이지에 대한 개략적인 레이아웃이나 뼈대를 설계하는 도구

### 목업(Mockup)

* 실제 화면과 유사하게 만든 정적인 형태의 모형

### 스토리보드(Story Board)

* 와이어프레임에 콘텐츠에 대한 설명, 페이지 간 이동 흐름 등을 추가한 문서

### 프로토타입(Prototype)

* 실제 구현된 것처럼 테스트가 가능한 동적인 형태의 모형

### 유스케이스(Use Case)

* 사용자의 요구사항을 기능 단위로 표현하는 것

## UI 요구사항 확인

* 새로 개발할 시스템에 적용할 UI 관련 요구사항을 조사해서 작성하는 단계
* UI 요구사항 확인 순서 : 목표 정의 -> 활동 사항 정의 – UI 요구사항 작성
* UI 요구사항 작성 순서 : 요구사항 요소 확인 -> 정황 시나리오 작성 -> 요구사항 작성
* 요구사항 요소 : 데이터 요구, 기능 요구, 제품/서비스의 품질, 제약사항

## 품질 요구사항

### 품질 요구사항

* 소프트웨어 품질은 소프트웨어에 대한 요구사항이 사용자의 입장에서 얼마나 충족하는가를 나타냄

소프트웨어의 품질 특성과 평가를 위한 국제 표준

ISO/IEC 9126

|  |  |
| --- | --- |
| ISO/IEC 25010 | ISO/IEC 9126에 호환성과 보안성을 강화하여 개정 |
| ISO/IEC 12119 | 패키지 소프트웨어의 일반적인 제품 품질 요구사항 및 테스트를 위한 국제 표준 |
| ISO/IEC 14598 | 소프트웨어 품질의 측정과 평가에 필요 절차를 규정한 표준 |

### ISO/IEC 9126 소프트웨어 품질 특성

* 기능성(Functionality)

: 소프트웨어가 사용자의 요구사항을 정확하게 만족하는 기능을 제공하는지 여부

* 신뢰성(Reliability)

: 주어진 시간 동안 주어진 기능을 오류 없이 수행할 수 있는 정도

* 사용성(Usability)

: 사용자가 정확하게 이해하고 사용하며, 향후 다시 사용하고 싶은 정도

* 효율성(Efficiency)

: 사용자가 요구하는 기능을 얼마나 빠르게 처리할 수 있는지 정도

* 유지 보수성(Maintainability)

: 환경의 변화 또는 새로운 요구사항이 발생했을 때 소프트웨어를 개선하거나 확장할 수 있는 정도

* 이식성(Portability)

: 소프트웨어가 다른 환경에서도 얼마나 쉽게 적용할 수 있는지 정도

## UI 설계

### UI 설계서

* 사용자의 요구사항을 바탕으로 UI 설계를 구체화하여 작성하는 문서
* UI 설계서 표지 작성 -> UI 설계서 개정 이력 작성 -> UI 요구사항 정의서 작성

-> 시스템 구조 작성 -> 사이트 맵 작성 -> 프로세스 정의서 작성 -> 화면 설계

### UI 흐름 설계

* 흐름을 파악하여 화면과 폼을 설계하는 단계
* 기능 작성 -> 입력 요소 확인 -> 유스케이스 설계 -> 기능 및 양식 확인

### UI 상세 설계

* 실제 설계 및 구현을 위해 모든 화면에 대해 자세하게 설계를 진행하는 단계
* 요구사항 확인 -> UI 설계서 표지 및 개정 이력 작성 -> UI 구조 설계 -> 메뉴 구조 설계 -> 화면 설계

### UI 시나리오 문서

* 요건 : 완전성, 일관성, 이해성, 가독성, 수정 용이성, 추적 용이성

## HCI/UX/감성공학

### HCI(Human Computer Interaction or Interface)

* 사람이 시스템을 보다 편리하고 안전하게 사용할 수 있도록 연구하고 개발하는 학문

### UX(User Experience)

* 사용자가 시스템이나 서비스를 이용하면서 느끼고 생각하게 되는 총체적인 경험
* 특징 : 주관성, 정황성, 총체성

### 감성공학

* 제품이나 작업 환경을 사용자의 감성에 알맞도록 설계 및 제작하는 기술

# 07장. 애플리케이션 테스트 관리

## 애플리케이션 테스트

### 애플리케이션 테스트

* 애플리케이션에 잠재되어 있는 결함을 찾아내는 일련의 행위 또는 절차

### 애플리케이션 테스트의 기본 원리

* 완벽한 테스트 불가능

: 소프트웨어에 아예 결함이 없다고 증명할 수는 없음

* 파레토 법칙(Pareto Principle)

: 애플리케이션의 20%에 해당하는 코드에서 전체 결함의 80%가 발견된다는 법칙

* 살충제 패러독스(Pesticide Paradox)

: 동일한 테스트 케이스로 동일한 테스트를 반복하면 더이상 결함이 발견되지 않는 현상

* 테스팅은 정황(Context) 의존

: 다른 정황, 다른 테스트

* 오류-부재의 궤변(Absence of Errors Fallacy)

: 소프트웨어의 결함을 모두 제거해도 사용자의 요구사항을 만족시키지 못하면 안 됨

* 테스트와 위험은 반비례
* 테스트의 점진적 확대
* 테스트의 별도 팀 수행

## 애플리케이션 테스트의 분류

### 프로그램 실행 여부에 따른 테스트

* 정적 테스트

: 프로그램을 실행하지 않고 명세서나 소스 코드를 대상으로 분석하는 테스트

: 워크스루, 인스펙션, 코드 검사 등

* 동적 테스트

: 프로그램을 실행하여 오류를 찾는 테스트

: 블랙박스 테스트(기능 테스트)

* + 각 기능이 완전히 작동되는 것을 입증하는 테스트
  + 동치 분할 검사(Equivalence Partitioning Testing, 동치 클래스 분해)

: 타당한 입력 자료와 타당하지 않은 입력 자료를 균등하게 입력 후 맞는 결과가 출력되는지

* + 경계값 분석(Boundary Value Analysis)

: 입력 조건의 경계값을 테스트 케이스로 선정하여 검사하는 기법

* + 원인-효과 그래프 검사(Cause-Effect Graphing Testing)

: 입력 데이터 간의 관계와 출력에 영향 미치는 상황을 분석해 효용성이 높은 케이스를 선정

* + 오류 예측 검사(Error Guessing)

: 과거의 경험이나 확인자의 감각으로 테스트하는 기법

* + 비교 검사(Comparison Testing)

: 여러 버전의 프로그램에 동일한 테스트 자료를 제공하여 동일한 결과가 나오는지 테스트

: 화이트박스 테스트

* + 원시 코드의 논리적인 모든 경로를 테스트하여 테스트 케이스를 설계하는 방법
  + 기초 경로 검사(Base Path Testing)

: 테스트 케이스 설계자가 절차적 설계의 논리적 복잡성을 측정할 수 있게 해주는 기법

* + 제어 구조 검사(Control Structure Testing)

: 조건 검사(조건), 루프 검사(반복), 데이터 흐름 검사(변수의 정의와 변수 사용의 위치)

* + 검증 기준

: 문장 검증 기준(Statement Coverage)

* + - 모든 구문이 한 번 이상 수행되도록

: 분기 검증 기준(Branch Coverage)

* + - 모든 조건문이 한 번 이상 수행되도록

: 조건 검증 기준(Condition Coverage)

* + - 모든 조건문의 조건이 True인 경우와 False인 경우가 한 번 이상 수행되도록

: 분기/조건 기준(Branch/Condition Coverage)

* + - 모든 조건문과 각 조건문에 포함된 개별 조건식의 결과가 True인 경우와 False 인 경우가 한 번 이상 수행되도록

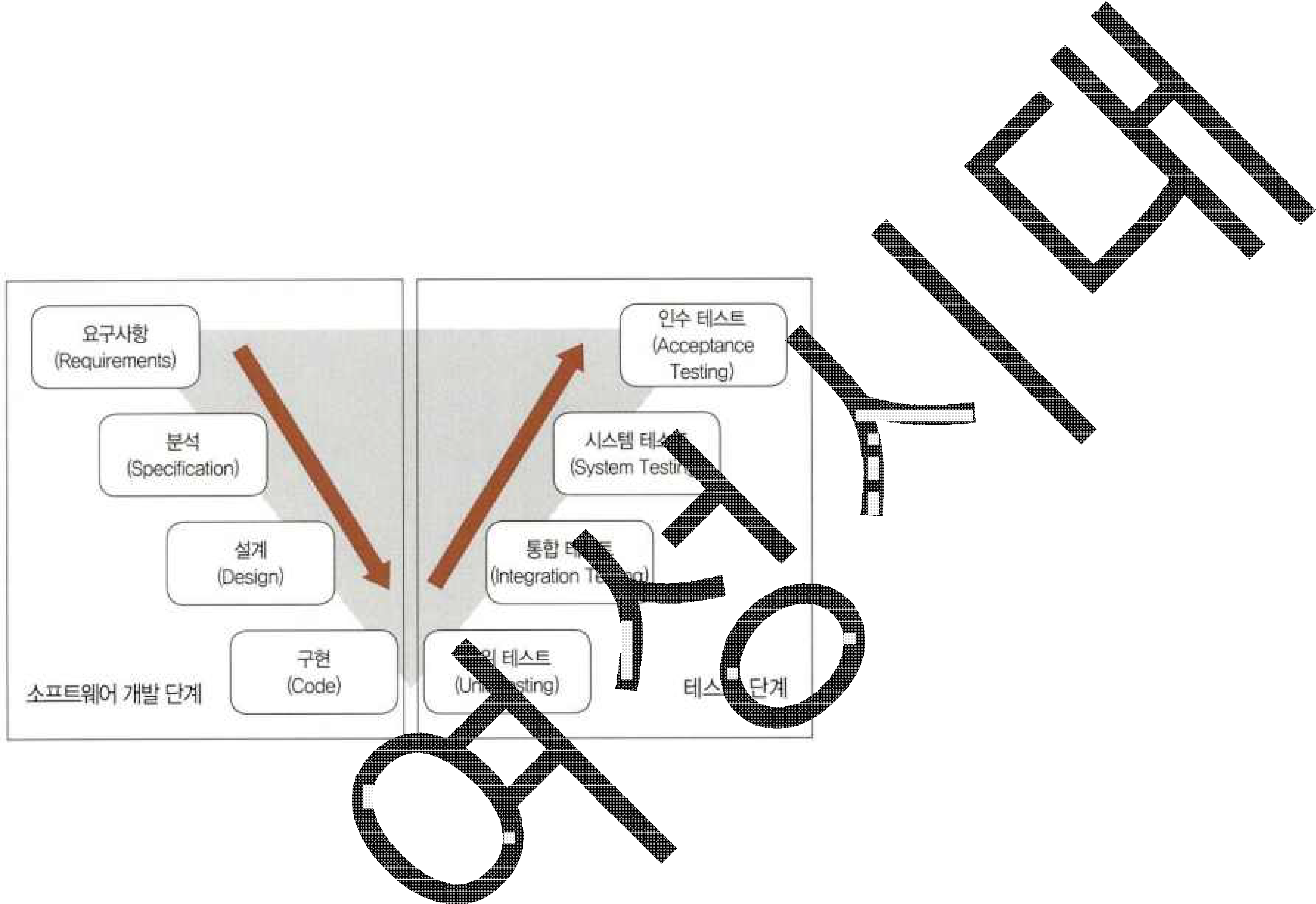
### 테스트 기반(Test Bases)에 따른 테스트

* 명세 기반 테스트 : 사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 만들어 구현하는지
* 구조 기반 테스트 : 소프트웨어 내부의 논리 흐름에 따라 테스트 케이스를 작성하고 확인
* 경험 기반 테스트 : 유사 소프트웨어나 기술 등에 대한 테스터의 경험을 기반으로 수행하는 테스트

### 시각에 따른 테스트

* 검증(Verification) 테스트 : 개발자의 시각에서 제품의 생산 과정을 테스트하는 것
* 확인(Validation) 테스트 : 사용자의 시각에서 생산된 제품의 결과를 테스트하는 것

### 통합 테스트(Integration Test)



**4) 목적에 따른 테스트**

- 회복, 안전, 강도, 성능, 구조, 회귀, 병행 테스트

**109. 개발 단계에 따른 애플리케이션 테스트**

- 소프트웨어 생명 주기의 V-모델

**1) 단위 테스트(Unit Test)**

* 모듈이나 컴포넌트에 초첨을 맞춰 테스트
* 구조 기반 테스트(화이트박스 테스트)와 명세 기반 테스트(블랙박스 테스트)로 나뉨
* 단위 테스트가 완료된 모듈들을 결합하여 하나의 시스템으로 완성시키는 과정에서의 테스트
* 비점진적 통합 방식

: 단계적으로 통합하는 절차 없이 모든 모듈이 미리 결헙되어 있는 프로그램 전체를 테스트

: 빅뱅 통합 테스트 방식

* 점진적 통합 방식

: 모듈 단위로 단계적으로 통합하면서 테스트하는 방법

: 하향식 통합 테스트(Top Down Integration Test)

* + 상위 모듈에서 하위 모듈 방향으로 통합하면서 테스트하는 기법

① 주요 제어 모듈은 작성된 프로그램을 사용하고, 종속 모듈들은 스텁(Stub)으로 대체

② 통합 방식에 따라 하위 모듈인 스텁들이 한 번에 하나씩 실제 모듈로 교체됨

③ 모듈이 통합될 때마다 테스트 실시

④ 새로운 오류가 발생하지 않음을 보증하기 위해 회귀 테스트 실시

\* 회귀 테스트 – 통합 테스트로 인해 변경된 모듈이나 컴포넌트에 새로운 오류가 있는지 확인하는 테스트

: 상향식 통합 테스트(Bottom Up Integration Test)

* + 하위 모듈에서 상위 모듈 방향으로 통합하면서 테스트하는 기법

① 하위 모듈들을 클러스터(Cluster)로 결합함

② 상위 모듈에서 더미 모듈인 드라이버(Driver)를 작성

③ 통합된 클러스터 단위로 테스트

④ 테스트 완료 후 클러스터는 프로그램 구조의 상위로 이동하여 결합하고 드라이버는 실제 모듈로 대체됨

: 혼합식 통합 테스트

* + 하위 수준에서는 상향식 통합, 상위 수준에서는 하향식 통합을 사용하여 최적의 테스트 지원

### 시스템 테스트(System Test)

* 개발된 소프트웨어가 완벽하게 수행되는가를 점검하는 테스트

### 인수 테스트(Acceptance Test)

* 사용자의 요구사항을 충족하는지에 중점을 두고 테스트
* 사용자 인수(사용자가), 운영상의 인수(시스템 관리자가), 계약 인수, 규정 인수, 알파 테스트(사용자가 개발자 앞에 서), 베타 테스트(사용자가 여러 명의 사용자 앞에서)

## 애플리케이션 테스트 프로세스

### 애플리케이션 테스트 프로세스

* 개발된 소프트웨어 사용자의 요구대로 만들어졌는지, 결함이 없는지 등을 테스트하는 절차
* 테스트 계획 -> 테스트 분석 및 디자인 -> 테스트 케이스 및 시나리오 작성

-> 테스트 수행 -> 테스트 결과 평가 및 리포팅 -> 결함 추적 및 관리

### 결함 관리 프로세스

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

## 테스트 케이스/테스트 시나리오/테스트 오라클

### 테스트 케이스(Test Cae)

* 사용자의 요구사항을 정확하게 준수했는지를 확인하기 위해 설계된 테스트 항목에 대한 명세서

### 테스트 시나리오(Test Scenario)

* 테스트 케이스를 적용하는 순서에 따라 여러 개의 테스트 케이스를 묶은 집합

### 테스트 오라클(Test Oracle)

* 테스트 결과가 올바른지 판단하기 위해 사전에 정의된 참값을 대입하여 비교하는 기법
* 특징 : 제한된 검증, 수학적 기법, 자동화 기능
* 종류 : 참(True) 오라클, 샘플링(Sampling) 오라클, 추정(Heuristic) 오라클, 일관성 검사(Consistent) 오라클

## 테스트 자동화 도구

### 테스트 자동화

* 사람이 반복적으로 수행하던 테스트 절차를 스크립트 형태로 구현하는 자동화 도구를 적용함

### 정적 분석 도구(Static Analysis Tools)

* 프로그램을 실행하지 않고 분석하는 도구

### 테스트 실행 도구(Test Execution Tools)

* 스크립트 언어를 사용하여 테스트를 실행하는 도구
* 데이터 주도 접근 방식, 키워드 주도 접근 방식

### 성능 테스트 도구(Performance Test Tools)

* 가상의 사용자를 만들어 테스트를 수행함으로써 성능의 목표 달성 여부를 확인하는 도구

### 테스트 통제 도구(Test Control Tools)

* 테스트 계획 및 관리, 테스트 수행, 결함 관리 등을 수행하는 도구

### 테스트 하네스 도구(Test Harness Tools)

* 테스트가 실행될 환경을 시뮬레이션 하여 컴포넌트 및 모듈이 정상적으로 테스트 되도록 하는 도구
* 구성요소 : 테스트 드라이버, 테스트 스텁, 테스트 슈트, 테스트 케이스, 테스트 스크립트, 목 오브젝트

### 테스트 수행 단계별 테스트 자동화 도구

- 책 참고(1-492P)

## 결함 관리

### 결함(Fault)

* 소프트웨어가 개발자가 설계한 것과 다르게 동작하거나 다른 결과가 발생되는 것
* 결함 관리 계획 -> 결함 기록 -> 결함 검토 -> 결함 수정 -> 결함 재확인

-> 결함 상태 추적 및 모니터링 활동 -> 최종 결함 분석 및 보고서 작성

### 결함 상태 추적

* 결함 관리 측정 지표 : 결함 분포, 결함 추세, 결함 에이징

### 결함 추적 순서

* 결함 등록 -> 결함 검토 -> 결함 할당 -> 결함 수정 -> 결함 조치 보류 -> 결함 종료 -> 결함 해제

### 결함 분류

* 시스템 결함, 기능 결함, GUI 결함, 문서 결함

### 결함 심각도

* 애플리케이션에 발생한 결함이 전체 시스템에 미치는 치명도를 나타내는 척도

### 결함 우선 순위

* 발견된 결함 처리에 신속성을 나타내는 척도

### 결함 관리 도구

* Mantis, Trac, Redmine, Bugzilla

## 애플리케이션 성능 분석

### 애플리케이션 성능

* 최소한의 자원을 사용하여 최대한 많은 기능을 신속하게 처리하는 정도
* 애플리케이션 성능 측정 지표

: 처리량(Throughput)

* + 일정 시간 내에 애플리케이션이 처리하는 일의 양

: 응답 시간(Response Time)

* + 애플리케이션에 요청을 전달한 시간부터 응답이 도착할 때까지 걸린 시간

: 경과 시간(Time Around TIme)

* + 애플리케이션에 작업을 의뢰한 시간부터 처리가 완료될 때까지 걸린 시간

: 자원 사용률(Resource Usage)

* + 애플리케이션이 의뢰한 작업을 처리하는 동안의 자원 사용률

### 성능 테스트 도구

* 부하나 스트레스를 가하면서 애플리케이션의 성능 측정 지표를 점검하는 도구
* JMeter, LoadUI, OpenSTA

### 시스템 모니터링 도구

* 애플리케이션이 실행되었을 때 시스템 자원의 사용량을 확인하고 분석하는 도구
* Scouter, Zabbix

## 복잡도

### 1) 시간 복잡도(Complexity)

* 알고리즘을 수행하기 위해 프로세스가 수행하는 연산 횟수를 수치화한 것
* 빅오 표기법(Big-O Notation) : 알고리즘의 실행시간이 최악일 때를 표기하는 방법
* 세타 표기법(Big-θ Notation) : 알고리즘의 실행시간이 평균일 때를 표기하는 방법
* 오메가 표기법(Big-Ω Notation) : 알고리즘의 실행시간이 최상일 때를 표기하는 방법

### 빅오 표기법으로 표현한 최악의 알고리즘 시간 복잡도

- 책 참고(1-501P)

### 순환 복잡도(Cyclomatic Complexity)

* 논리적인 복잡도를 측정하기 위한 소프트웨어의 척도

## 애플리케이션 성능 개선

### 소스 코드 최적화

* 나쁜 코드(Bad Code)를 배제하고, 클린 코드(Clean Code)로 작성하는 것
* 나쁜 코드 : 스파게티 코드, 외계인 코드

### 클린 코드 작성 원칙

* 가독성 : 누구든지 코드를 쉽게 읽을 수 있도록 작성함
* 단순성 : 코드를 간단하게 작성함
* 의존성 배제 : 코드가 다른 모듈에 미치는 영향을 최소화함
* 중복성 최소화 : 코드의 중복을 최소화함
* 추상화 : 상위에서는 간략하게 애플리케이션의 특성을 나타내고 상세 내용은 하위에서 구현함

### 소스 코드 최적화 유형

* 클래스 분할 배치 : 하나의 클래스는 하나의 역할만 수행하도록 응집도를 높이고 크기를 작게 작성
* 느슨한 결합 : 추상화된 자료 구조와 메소드를 구현함으로써 클래스 간의 의존성을 최소화

### 소스 코드 품질 분석 도구

* 정적 분석 도구 : 작성한 소스 코드를 실행하지 않고 분석
* 동적 분석 도구 : 작성한 소스 코드를 실행하여 분석

### 소스 코드 품질 분석 도구의 종류

* 정적 분석 도구 : pmd, cppcheck, SonarQube, checkStyle, ccm, cobertura
* 동적 분석 도구 : Avalanche, Valgrind

# 09장. 소프트웨어 개발 보안 구축

## Secure SDLC

### Secure SDLC(소프트웨어 개발 생명 주기)

* SDLC에 보안 강화를 위한 프로세스를 포함한 것
* CLASP : SDLC의 초기 단계에서 보안을 강화하기 위해 개발된 방법론
* SDL : MS사에서 기존의 SDLC를 개선한 방법론
* Seven Touchpoints : 소프트웨어 보안의 모범사례를 SDLC에 통합한 방법론

### SDLC 단계별 보안 활동

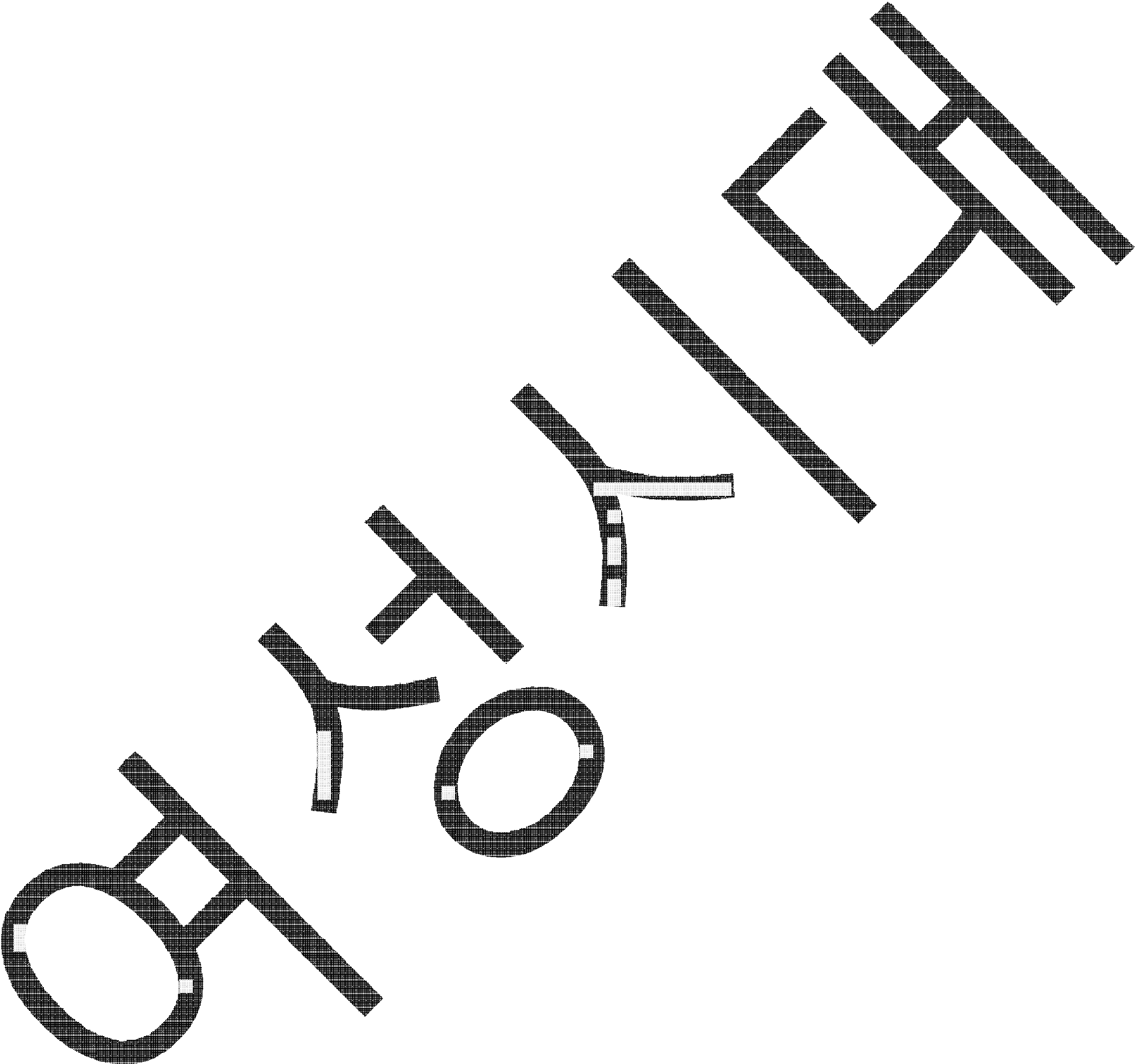
* 요구사항 분석 -> 설계 -> 구현 -> 테스트 -> 유지보수

### 소프트웨어 개발 보안 요소

* 기밀성(Confidentiality)

: 시스템 내의 정보와 자원은 인가된 사용자에게만 접근이 허용됨

* 무결성(Integrity)

: 시스템 내의 정보는 오직 인가된 사용자만 수정할 수 있음

* 가용성(Availability)

: 인가받은 사용자는 시스템 내의 정보와 자원을 언제라도 사용할 수 있음

* 인증(Authentication)

: 시스템 내의 정보와 자원을 사용하려는 사용자가 합법적인 사용자인지를 확인하는 모든 행위

* 부인 방지(NonRepudiation)

: 데이터를 송, 수신한 자가 송, 수신 사실을 부인할 수 없도록 송, 수신 증거를 제공함

### 시큐어 코딩(Secure Coding)

* 보안 요소들을 고려하며 코딩하는 것

## 세션 통제

### 세선 통제

* 세션의 연결과 연결로 인해 발생하는 정보를 관리하는 것
* 보안 약점 : 불충분한 세션 관리, 잘못된 세션에 의한 정보 노출

### 세션 설계 시 고려사항

- 책 참고(2-137P)

### 세션ID 의 관리 방법

* 최소 128비트 길이로, 안전한 난수 알고리즘을 적용해서, URL Rewrite 기능을 사용하지 않는 방향으로
* 나머지는 책 참고(2-137P)

## 입력 데이터 검증 및 표현

* 입력 데이터 검증 및 표현의 보안 약점

|  |  |
| --- | --- |
| SQL 삽입(Injection) | 웹 응용 프로그램에 SQL을 삽입하여 내부 DB 서버의 데이터를 유출 및 변조하고  관리자 인증을 우회하는 보안 약점 |
| 경로 조작 및 자원 삽입 | 데이터 입출력 경로를 조작하여 서버 자원을 수정, 삭제할 수 있는 보안 약점 |
| 크로스사이트 스크립팅(XSS) | 웹페이지에 악의적인 스크립트를 삽입하여 방문자들의 정보를 탈취하거나 비정상적  인 기능 수행을 유발하는 보안 약점 |
| 운영체제 명령어 삽입 | 외부 입력값을 통해 시스템 명령어의 실행을 유도함으로써 권한을 탈취하거나 시스  템 장애를 유발 |
| 위험한 형식 파일 업로드 | 악의적인 명령어가 포함된 스크립트 파일을 업로드함으로써 시스템에 손상을 주거나  시스템을 제어할 수 있는 보안 약점 |
| 신뢰되지 않는 URL 주소로 | 입력값으로 사이트 주소를 받는 경우 이를 조작하여 방문자를 피싱 사이트로 유도하 |

|  |  |
| --- | --- |
| 자동접속 연결 | 는 보안 약점 |
| 메모리 버퍼 오버플로 | 연속된 메모리 공간을 사용하는 프로그램에서 할당된 메모리의 범위를 넘어선 위치  에서 자료를 읽거나 쓰려고 할 때 발생하는 보안 약점 |

## 보안 기능

* 보안 기능의 보안 약점

|  |  |
| --- | --- |
| 적절한 인증 없이 중요기능 허용 | 보안검사를 우회하여 인증과정 없이 중요한 정보 또는 기능에 접근 및 변  경이 가능함 |
| 부적절한 인가 | 접근제어 기능이 없는 실행경로를 통해 정보 또는 권한을 탈취할 수 있음 |
| 중요한 자원에 대한 잘못된 권한 설정 | 권한 설정이 잘못된 자원에 접근하여 해당 자원을 임의로 사용할 수 있음 |
| 취약한 암호화 알고리즘 사용 | 암호화된 환경설정 파일을 해독하여 비밀번호 등의 중요 정보를 탈취할 수  있음 |
| 중요정보 평문 저장 및 전송 | 암호화되지 않은 평문 데이터를 탈취하여 중요한 정보를 획득할 수 있음 |
| 하드코드 된 암호화 키 | 암호화된 키도 하드코드된 경우 유출 시 역계산 또는 무차별 대입 공격에  의해 탈취될 수 있음 |

## 시간 및 상태

* 시간과 실행 상태 관리
* 시간 및 상태의 보안 약점

: TOCTOU 경쟁 조건

* + 검사 시점(Time Of Check)와 사용 시점(Time Of use)을 고려하지 않고 코딩하는 경우 발생

: 종료되지 않는 반복문 또는 재귀함수

* + 반복문이나 재귀함수에서 무한 반복되는 경우 발생하는 보안 약점

## 에러처리

* 에러처리의 보안 약점

: 오류 메시지를 통한 정보 노출, 오류 상황 대응 부재, 부적절한 예외처리

## 코드 오류

### 코드 오류의 보안 약점

|  |  |
| --- | --- |
| 널 포인터(Null Pointer) 역참조 | 널 포인터가 가리키는 메모리의 위치에 값을 저장할 때 발생 |
| 부적절한 자원 해제 | 자원을 반환하는 코드를 누락하거나 프로그램 오류로 할당된 자원을 반환하지 못  했을 때 발생 |
| 해제된 자원 사용 | 이미 사용이 종료되어 반환된 메모리를 참조하는 경우 발생 |
| 초기화되지 않은 변수 사용 | 변수 선언 후 값이 부여되지 않은 변수를 사용할 때 발생 |

1. **스택 가드(Stack Guard)**

* 주소가 저장되는 스택에서 발생하는 보안 약점을 막는 기술

## 캡슐화

### 캡슐화(Encapsulation)

* 정보 은닉이 필요한 중요한 데이터와 기능을 불완전하게 캡슐화하거나 잘못 사용함으로써 발생할 수 있는 문제를 예 방하기 위한 보안 점검 항목들

### 캡슐화의 보안 약점

|  |  |
| --- | --- |
| 잘못된 세션에 의한 정보  노출 | 다중 스레드 환경에서 멤버 변수에 정보를 저장할 때 발생 |
| 제거되지 않고 남은 디버그  소스 | 개발 중에 버그 수정이나 결과값 확인을 위해 남겨둔 코드로 인해 발생 |
| 시스템 데이터 정보 노출 | 시스템의 내부 정보를 시스템 메시지 등을 통해 외부로 출력하도록 코딩했을 때 발생 |
| Public 메소드로부터 반환  된 Private 배열 | Private 배열을 Public 메소드에서 반환할 때 발생 |
| Private 배열에 Public 데  이터 할당 | Private 배열에 Public으로 선언된 데이터 또는 메소드의 파라미터를 저장할 때 발생 |

1. **접근 제어자**

* 외부로부터의 접근을 제한하기 위해 사용되는 예약어
* Public > Protected > Default > Private

## API 오용

* API 오용의 보안 약점

: DNS Lookup에 의존한 보안 결정, 취약한 API 사용

## 암호 알고리즘

### 암호 알고리즘

* 중요정보를 보호하기 위해 평문을 암호화된 문장으로 만드는 절차 또는 방법을 의미

### 양방향

* + 개인키 암호화(Private Key Encryption) 기법(대칭, 단일키)

: 동일한 키로 데이터를 암호화하고 복호화하는 기법

: 스트림 암호화 방식

* + - 평문과 동일한 길이의 스트림을 생성하여 비트 단위로 암호화하는 방식
    - 종류 : LFSR, RC4

: 블록 암호화 방식

* + - 한 번에 하나의 데이터 블록을 암호화하는 방식
    - DES(Data Encryption Standard)

: 1975년 미국 NBS에서 발표

* + - SEED : 1999년 KISA에서 개발

: 키 길이에 따라 128, 256으로 분류됨

* + - AES(Advanced Encryption Standard)

: 2001년 NIST에서 DES의 한계를 느끼고 발표

* + - ARIA(Academy, Research Institute, Agency)

: 2004년 국가정보원과 산학연협회가 개발

* + 공개키 암호화(Public Key Encryption) 기법(비대칭)

: 암호화할 때 사용하는 공개키는 공개하고 비밀키는 관리자가 비밀리에 관리

: RSA(Rivest Shamir Adleman)

* + - 1978년에 MIT의 R, S, A가 만듦
    - 큰 숫자를 소인수분해 하기 어렵다는 것에 기반하여 만들어짐

### 단방향

* 해시(Hash)

: 임의의 길이의 입력 데이터나 메시지를 고정된 길이의 값이나 키로 변환하는 것

: SHA 시리즈

* + 1993년 NSA가 설계, NIST에 의해 발표됨

: MD5(Message Digest algorithm 5)

* + 1991년 Rivest가 MD4를ㄹ 대체하기 위해 고안
  + 블록 크기 512비트, 키 길이 128비트

: N-NASH

* + 1989 일본의 NTT에서 발표, 블록과 키 모두 128비트

: SNEFRU

* + 1990년 R.C.Merkle가 발표, 32비트 프로세서에서 구현을 용이하게 할 목적으로 개발됨

## 서비스 공격 유형

1. **서비스 거부(DoS; Denial of Service 공격)**

|  |  |
| --- | --- |
| Ping of Death(죽음의 핑) | 패킷의 크기를 인터넷 프로토콜 허용 범위 이상으로 전송하여 네트워크를  마비시킴 |
| SMURFING(스머핑) | IP나 ICMP의 특성을 악용하여 엄청난 양의 데이터를 한 사이트에 집중적  으로 보냄으로써 네트워크를 불능 상태로 만듦 |
| SYN Flooding | 3-way-handshake 과정을 의도적으로 중단시킴으로써 서버가 대기 상태  에 놓여 정상적인 서비스를 수행하지 못하게 하는 공격 방법 |
| TearDrop | Offset 값을 변경시켜 수신 측에서 과부하를 발견시킴으로써 시스템이 다  운되도록 함 |
| LAND Attack  (Local Area Network Denial Attack) | 패킷을 전송할 때 송신 IP 주소와 수신 IP 주소를 모두 공격 대상의 IP  주소로 하여 자신에 대해 무한히 응답하게 함 |
| DDoS(Distributed Denial of Service) (분산 서비스 거부 공격) | * 여러 곳에 분산된 공격 지점에서 한 곳의 서버에 대해 분산 서비스 공   격을 수행하는 것   * 분산 서비스 공격용 툴(데몬, Demon)   : Thin00, TFN, TFN2K, Stacheldraht |

1. **네트워크 침해 공격 관련 용어**

|  |  |
| --- | --- |
| 스미싱(Smishing) | 문자 메시지를 이용해 사용자의 개인 사용 정보를 빼내는 수법 |
| 스피어 피싱(Spear Phishing) | 일반적인 이메일로 위장한 메일을 지속적으로 발송해 발송 메일의 본문 링  크나 첨부 파일을 클릭하도록 유도해 개인 정보를 탈취 |
| APT(Advanced Persistent Threats) (지능형 지속 위협) | 다양한 IT 기술과 방식들을 이용해 공격 대상의 네트워크에 침투해 활동  거점을 마련한 뒤, 때를 기다리면서 보안을 무력화시키고 정보를 수집해 외  부로 빼돌림 |
| 무작위 대입 공격  (Brute Force Attack) | 암호키를 찾기 위해 적용 가능한 모든 값을 대입하여 공격 |
| 큐싱(Qshing) | QR코드를 통해 악성 앱의 다운로드를 유도하거나 악성 프로그램을 설치하  도록 함 |
| SQL 삽입(Injection) | 전문 스캐너 프로그램 혹은 봇넷 등을 이용해 웹사이트를 무차별적으로 공  격하는 과정에서 취약한 사이트가 발견되면 DB등의 데이터를 조작함 |
| 크로스 사이트 스크립팅  (XSS; Cross Site Scripting) | 웹페이지의 내용을 사용자 브라우저에 표현하기 위해 사용되는 스크립트의  취약점을 악용 |
| 스니핑(Sniffing) | 엿보기, 남의 패킷 정보를 도청하는 해킹 유형 |

1. **정보 보안 침해 공격 관련 용어**

|  |  |
| --- | --- |
| 좀비(Zombie) PC | 악성코드에 감염되어 다른 프로그램이나 컴퓨터를 조종하도록 만들어진 컴퓨터로,  C&C 서버의 제어를 받아 주로 DDoS 공격 등에 이용됨 |
| C&C 서버 | 해커가 원격지에서 감염된 좀비 PC에 명령을 내리고 악성코드를 제어하기 위한 용도 |

|  |  |
| --- | --- |
|  | 로 사용하는 서버 |
| 봇넷(Botnet) | 악성 프로그램에 감염되어 악의적인 의도로 사용될 수 있는 다수의 컴퓨터들이 네트워  크로 연결된 형태 |
| 웜(Worm) | 네트워크를 통해 연속적으로 자신을 복제하여 시스템의 부하를 높여서 시스템을 다운  시킴 |
| 제로 데이 공격  (Zero Day Attack) | 보안 취약점이 발견되었을 때 널리 공표되기도 전에 해당 취약점을 통하여 이루어지는  보안 공격 |
| 키로거 공격  (Key Logger Attack) | 컴퓨터, 사용자의 키보드 움직임을 탐지해 개인 정보를 빼가는 해킹 공격 |
| 랜섬웨어(Ransomware) | 사용자의 컴퓨터에 잠입해 내부 문서나 파일 등을 열지 못하도록 암호화하는 프로그램 |
| 백도어  (Back Door, Trap Door) | 시스템 설계자가 액세스 편의를 위해 시스템 보안을 제거하여 만들어놓은 비밀 통로 |
| 트로이 목마  (Trojan Horse) | 정상적인 기능을 하는 프로그램으로 위장하여 숨어있다가 해당 프로그램이 동작할 때  활성화됨, 자기 복제 능력은 없음 |

## 서버 인증

### 보안 서버

* 개인 정보를 암호화하여 송, 수신할 수 있는 기능을 갖춘 서버

### 인증(Authentication)

* 로그인을 요청한 사용자의 정보를 확인하고 접근 권한을 검증하는 보안 절차
* 지식 기반 인증(Something You Know)

: 사용자가 기억하고 있는 정보를 기반으로 인증을 수행하는 것

: 유형 - 고정된 패스워드, 패스 프레이즈, 아이핀

* 소유 기반 인증(Something You Have)

: 사용자가 소유하고 있는 것을 기반으로 인증을 수행한느 것

: 유형 - 신분증, 메모리 카드(토큰), 스마트카드, OTP

* 생체 기반 인증(Something You Are)

: 사용자의 고유한 생체 정보를 기반으로 인증을 수행하는 것

* 위치 기반 인증(Somewhere You Are)
* 행위 기반 인증(Something You Do)

## 보안 아키텍처/보안 프레임워크

### 보안 아키텍처(Security Archittecture)

* 보안 요소 및 보안 체계를 식별하고 이들 간의 관계를 정의한 구조

### 보안 프레임워크(Security Framework)

* 안전한 정보 시스템 환경을 유지하고 보안 수준을 향상시키기 위한 체계
* ISO 27001은 정보보안 관리를 위한 국제 표준으로, 가장 대표적인 보안 프레임워크

## 로그 분석

* 로그(Log) : 시스템 사용에 대한 모든 내역을 기록해 놓은 것
* 나머지는 책 참고(2-167P)

## 보안 솔루션

### 보안 솔루션

* 외부로부터의 불법적인 침입을 막는 기술 및 시스템

1. **주요 보안 솔루션**

|  |  |
| --- | --- |
| 방화벽(Firewall) | 내부의 네트워크와 인터넷 간에 전송되는 정보를 선별하여 수용, 거부,  수정하는 기능을 가진 침입 차단 시스템 |
| 침입 탐지 시스템  (IDS; Intrusion Detection System) | 컴퓨터 시스템의 비정상적인 사용, 오용, 남용 등을 실시간으로 탐지하  는 시스템   * 오용 탐지 : 미리 입력해 둔 공격 패턴 감지 * 이상 탐지 : 평균적인 시스템의 상태를 기준으로 감지 |
| 침입 방지 시스템  (IPS; Intrusion Prevention System) | 비정상적인 트래픽을 능동적으로 차단하고 격리함 |
| 데이터 유출 방지  (DLP; Data Leakage/Loss Prevention) | 내부 정보의 외부 유출을 방지하는 보안 솔루션 |
| 웹 방화벽(Web Firewall) | 웹 기반 공격을 방어할 목적으로 만들어진 웹 서버에 특화된 방화벽 |
| VPN(Virtual Private Network) (가상 사설 통신망) | 공중 네트워크와 암호화 기술을 이용하여 사용자가 마치 자신의 전용 회선을 사용하는 것처럼 해주는 보안 솔루션 |
| NAC(Network Access Control) | 네트워크에 접속하는 내부 PC의 일관된 보안 관리 기능을 제공 |
| ESM(Enterprise Security Management)  (통합 보안 관계 시스템) | 로그 및 보안 이벤트를 통합하여 관리함 |

## 취약점 분석 · 평가

* 취약점 분석, 평가 계획 수립 -> 취약점 분석, 평가 대상 선별 -> 취약점 분석 수행 -> 취약점 평가 수행

# 10장. 프로그래밍 언어 활용

## 절차적 프로그래밍 언어

### 절차적 프로그래밍 언어

* 일련의 처리 절차를 정해진 문법에 따라 순서대로 기술하는 언어

### 절차적 프로그래밍 언어의 종류

* C : 1972년 미국 벨 연구소의 데니스 리치에 의해 개발됨

: 고급이면서 저급의 특징도 갖추고 이식성도 좋음

* AlGOL : PSCAL과 C언어의 모체가 되는 과학 기술 계산용 언어
* COBOL : 사무 처리용 언어
* FORTRAN : 과학 기술 계산용 언어

## 객체지향 프로그래밍 언어

### 객체지향 프로그래밍 언어

* 현실 세계의 객체를 하나의 객체로 만들어 객체들을 조립해서 프로그램을 작성하는 기법

### 객체지향 프로그래밍 언어의 종류

* JAVA : 멀티스레드 기능을 제공, 운영체제 및 하드웨어에 독립적, 이식성이 강함
* C++ : C언어에 객체지향 개념을 적용한 언어
* Smalltalk : 1세대 객체지향 프로그래밍 언어, 최초로 GUI를 제공함

## 스크립트 언어

### 스크립트 언어

* HTML 문서 안에 직접 프로그래밍 언어를 삽입하여 사용하는 언어
* 서버용, 클리언트용으로 분류됨

### 스크립트 언어의 종류

* JAVA Script : 웹페이지의 동작을 제어하는 데 사용됨
* VB Script : MS사에서 자바 스크립트에 대응하기 위해 제작
* ASP : MS사에서 제작한 서버측에서 동적으로 수행되는 페이지를 만들기 위한 언어
* JSP : JAVA로 만들어진 서버용 스크립트 언어
* PHP : Linux, Unix, Windows OS에서 사용 가능한 서버용 스크립트 언어
* 파이썬 : 객체지향 기능을 지원하는 대화형 인터프리터 언어
* 쉘 스크립트 : 유닉스/리눅스 개열의 Shell에서 사용되는 명령어들의 조합으로 구성된 스크립트 언어
* Basic : 절차지향 기능을 지원하는 대화형 인터프리터 언어

## 선언형 언어

### 선언형 언어

* 프로그램이 수행해야 할 문제를 기술하는 언어
* 함수형 언어와 논리형 언어로 분류됨
* 선언형 언어의 종류

: HTML - 인터넷의 표준 문서인 하이퍼텍스트 문서를 만들기 위해 사용

: LSP - 인공지능 분야에 사용되는 언어

: PROLOG - 논리학을 기초로 한 고급 언어

: XML - 기존 HTML의 단점을 보완하여 웹에서 구조화된 폭넓고 다양한 문서들을 상호 교환할 수 있도록 설계된 언어

: Haskell - 함수형 프로그래밍 언어로 부작용이 없음

### 명령형 언어

* 문제를 해결하기 위한 방법을 기술하는 언어

## 라이브러리

* 자주 사용하는 함수나 데이터들을 미리 만들어 모아놓은 집합체
* 표준 라이브러리와 외부 라이브러리로 분류됨
* 나머지 책 참고(254P)

## 예외 처리(Exception Handling)

* 예외가 발생했을 때 프로그래머가 해당 문제에 대비해 작성해놓은 처리 루틴이 수행되도록 하는 것



# 11장. 응용 SW 기초 기술 활용

## 운영체제의 개념

### 운영체제(OS; Operating System)

* 컴퓨터 시스템의 자원들을 효율적으로 관리하며, 편리하고 효과적으로 사용할 수 있도록 환경을 제공하는 여러 프로 그램의 모임

### 운영체제의 목적

* 처리 능력(Throughput) : 일정 시간 내에 시스템이 처리하는 일의 양
* 반환 시간(Turn Around Time) : 시스템에 작업을 의뢰한 시간부터 처리가 완료될 때까지 걸린 시간
* 사용 가능도(Availability) : 시스템을 사용할 필요가 있을 때 즉시 사용 가능한 정도
* 신뢰도(Reliability) : 시스템이 주어진 문제를 정확하게 해결하는 정도

### 운영체제의 기능

- 책 참고(2-378P)

## 운영체제의 종류

### Windows

* 1990대 Microsoft 사가 개발한 운영체제
* Windows의 주요 특징

|  |  |
| --- | --- |
| 그래픽 사용자 인터페이스  (GUI; Graphic User Interface) | 키보드로 명령어를 직접 입력하지 않고 마우스로 아이콘이나 메뉴를 선택 |
| 선점형 멀티 태스킹 (Preemptive Multi-Tasking) | 동시에 여러 개의 프로그램을 실행하는 멀티태스킹을 하면서 응용 프로그램  실행중 문제가 발생하면 해당 프로그램을 강제 종료하고 모든 시스템 자원을  반환 |
| PnP(Plug and Play)  (자동 감지 기능) | 어떤 하드웨어를 사용하는 데 필요한 시스템 환경을 운영체제가 자동으로 구  성해주는 기능 |
| OLE  (Object Linking and Embedding) | 다른 여러 응용 프로그램에서 작성된 문자나 그림 등의 개체를 현재 작성 중  인 문서에 자유롭게 연결하거나 삽입하여 편집할 수 있게 하는 기능 |
| 255자의 긴 파일명 |  |
| Single-User 시스템 | 컴퓨터 한 대를 한 사람만이 독점해서 사용 |

### UNIX

* AT&T 벨(Bell) 연구소, MIT, General Electric이 공동 개발한 운영체제
* UNIX 시스템의 구성

: 커널(Kernel) - 하드웨어를 보호하고, 프로그램과 하드웨어 간의 인터페이스 역할, 핵심 부분

: 쉘(Shell) - 명령어 해석기, 시스템과 사용자 간의 인터페이스 담당

: 유틸리티 프로그램(Utility Program)

* + 일반 사용자가 작성한 응용 프로그램을 처리하는 데 사용함

### LINUX

* 1991년 리누스 토발즈(Linus Torvalds)가 UNIX를 기반으로 개발한 운영체제

### MacOS

* 1980년대 애플(Apple) 사가 UNIX를 기반으로 개발한 운영체제

### Android

* 구글 사에서 개발한 리눅스 커널 기반의 개방형 모바일 운영체제

### iOS

* 애플 사에서 개발한 유닉스 기반의 모바일 운영체제

## 기억장치 관리

### 반입(Fetch) 전략

* 프로그램이나 데이터를 언제 주기억장치로 적재할 것인지를 결정
* 요구 반입(Demand Fetch) : 실행중인 프로그램이 특정 프로그램이나 데이터 등의 참조를 요구할 때 적재
* 예상 반입(Anticipatory Fetch) : 실행중인 프로그램에 의해 참조될 프로그램이나 데이터를 미리 예상하여 적재

### 배치(Placement) 전략

* 프로그램이나 데이터를 주기억장치의 어디에 위치시킬 것인지를 결정
* 최초 적합(First Fit) : 첫 번째 분할 영역에 배치
* 최적 적합(Best Fit) : 단편화를 가장 적게 남기는 분할 영역에 배치
* 최악 적합(Worst Fit) : 단편화를 가장 많이 남기는 분할 영역에 배치

### 교체(Replacement) 전략

* 이미 사용되고 있는 영역 중에서 어느 영역을 교체하여 사용할 것인지를 결정

## 주기억장치 할당 기법

* 프로그램이나 데이터를 실행시키기 위해 주기억장치에 어떻게 할당할 것인지에 대한 내용

### 1) 주기억장치 할당 기법의 분류

* 연속 할당 기법

: 프로그램을 주기억장치에 연속으로 할당하는 기법

: 단일 분할 할당 기법

* + 한순간에는 오직 한 명의 사용자만이 주기억장치의 사용자 영역을 사용하는 기법
  + 오버레이(Overlay) 기법 : 주기억장치보다 큰 사용자 프로그램을 실행하기 위한 기법
  + 스와핑(Swapping) 기법 : 하나의 프로그램 전체를 주기억장치에 할당하여 사용하다 필요에 따라 다른 프로그램과 교체하는 기법

: 다중 분할 할당 기법

* + 고정 분할 할당 기법 : 운영체제가 주기억장치의 사용자 영역을 여러 개의 고정된 크기로 분할 (= 정적 할당 기법) 하고 준비상태 큐에서 준비 중인 프로그램을 각 영역에 할당하여 수행하는

기법

* + 가변 분할 할당 기법 : 프로그램을 주기억장치에 적재하면서 필요한 만큼의 크기로 영역을 분할 (= 동적 할당 기법) 하는 기법
* 분산 할당 기법

: 프로그램을 특정 단위의 조각으로 나누어 주기억장치 내에 분산하여 할당하는 기법

: 페이징 기법, 세그먼테이션 기법

## 가상기억장치 구현 기법

### 가상기억장치(Virtual Memory)

* 보조기억장치의 일부를 주기억장치처럼 사용하는 것
* 가상기억장치의 일반적인 구현 방법 : 페이징 기법, 세그먼테이션 기법

### 페이징(Paging) 기법

* 프로그램과 주기억장치의 영역을 동일한 크기로 나눈 후 주기억장치의 영역에 적재시켜 실행하는 기법

### 세그먼테이션(Segmentation) 기법

* 프로그램을 다양한 크기의 논리적인 단위로 나눈 후 주기억장치에 적재시켜 실행

## 페이지 교체 알고리즘

### 페이지 교체 알고리즘

* 페이지 부재(Page Fault)가 발생하면 어떤 페이지 프레임을 선택하여 교체할 것인지를 결정

### OPT(OPTimal replacement, 최적 교체)

* 앞으로 가장 오랫동안 사용하지 않을 페이지를 교체

### FIFO(First Int First Out)

* 가장 먼저 들어와서 가장 오래 있었던 페이지를 교체

### LRU(Least Recently Used)

* 최근에 가장 오랫동안 사용하지 않은 페이지를 교체

### LFU(Least Frequently Used)

* 사용 빈도가 가장 적은 페이지를 교체

### NUR(Not Used Recently)

* 최근에 사용하지 않은 페이지를 교체

### SCR(Second Chance Replacement, 2차 기회 교체)

* 가장 오랫동안 주기억장치에 있던 페이지 중 자주 사용되는 페이지의 교체를 방지하기 위한 기법

## 가상기억장치 기타 관리 사항

### 페이지 크기

* 페이지 크기↓ : 페이지 단편화↓, 페이지 이동 시간↓
* 페이지 크기↑ : 페이지 맵 테이블의 크기↓, 매핑 속도↑

### Locality

* 프로세스가 실행되는 동안 주기억장치를 참조할 때 일부 페이지만 집중적으로 참조하는 성질
* 스레싱(Thrashing)을 방지하기 위한 워킹 셋(Working Set) 이론의 기반이 됨

\* 스레싱 : 프로세스의 처리 시간보다 페이지 교체에 소요되는 시간이 더 많아지는 현상

\* 워킹 셋 : 프로세스가 일정 시간 동안 자주 참조하는 페이지들의 집합

* Locality의 종류

: 시간 구역성(Temporal Locality)

* + 프로세스가 실행되면서 하나의 페이지를 일정 시간 동안 집중적으로 액세스하는 현상

: 공간 구역성(Spatial Locality)

* + 프로세스 실행 시 일정 위치의 페이지를 집중적으로 액세스하는 현상

## 프로세스의 개요

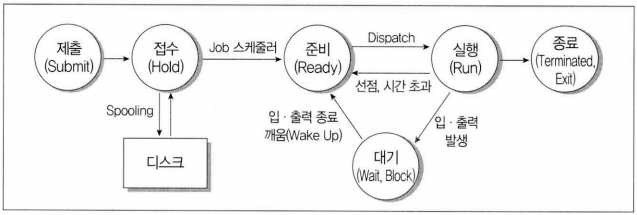
### 프로세스(Process)

* 실행중인 프로그램을 의미

### PCB(Process Control Block, 프로세스 제어 블록)

* 운영체제가 프로세스에 대한 중요한 정보를 저장해놓는 곳

### 프로세스 상태 전이

* 프로세스가 시스템 내에 존재하는 동안 프로세스의 상태가 변하는 것을 의미

|  |  |
| --- | --- |
| 제출(Submit) | 작업을 처리하기 위해 사용자가 작업을 시스템에 제출한 상태 |
| 접수(Hold) | 제출된 작업이 스풀 공간인 디스크의 할당 위치에 저장된 상태 |
| 준비(Ready) | 프로스세가 프로세서를 할당받기 위해 기다리고 있는 상태 |
| 실행(Run) | 준비상태 큐에 있는 프로세스가 프로세서를 할당받아 실행되는 상태 |
| 대기(Wait), 블록(Block) | 프로세스에 입, 출력 처리가 필요하면 현재 실행중인 프로세스가 중단되고, 입, 출력 처 |

|  |  |
| --- | --- |
|  | 리가 완료될 때까지 대기하고 있는 상태 |
| 종료(Terminated, Exit) | 프로세스의 실행이 끝나고 프로세스 할당이 해제된 상태 |

### 프로세스 상태 전이 관련 용어

* Dispatch : 준비상태에서 대기하고 있는 프로세스 중 하나가 실행 상태로 전이되는 과정
* Wake up : 입, 출력 작업이 완료되어 프로세스가 대기 상태에서 준비상태로 전이되는 과정
* Spooling : 입, 출력할 데이터를 직접 입, 출력 장치에 보내지 않고 나중에 한꺼번에 입, 출력하기 위 해 디스크에 저장하는 과정
* Traffic Controller) : 프로세스의 상태에 대한 조사와 통보 담당

### 스레드(Thread)

* 시스템의 여러 자원을 할당받아 실행하는 프로그램의 단위

## 스케줄링

### 스케줄링(Scheduling)

* 시스템의 여러 자원을 해당 프로세스에게 할당하는 작업
* 장기 스케줄링 : 어떤 프로세스가 시스템의 자원을 차지할 수 있도록 할 것인가를 결정하여 준비상태 큐로 보냄
* 중기 스케줄링 : 어떤 프로세스들이 CPU를 할당받을 것인지를 결정
* 단기 스케줄링 : 프로세스가 실행되기 위해 CPU를 할당받는 시기와 특정 프로세스를 지정하는 작업

### 스케줄링의 목적

* 공정성, 처리율 증가, CPU 이용률 증가, 우선순위 제도, 오버헤드 최소화, 응답시간 최소화, 반환 시간 최소화, 대기 시간 최소화, 균형 있는 자원의 사용, 무한 연기 회피

### 비선점(Non-Preemptive) 스케줄링

* 이미 할당된 CPU를 다른 프로세스가 강제로 빼앗아 사용할 수 없는 스케줄링 기법
* FCFS(First Come First Service, 선입 선출) = FIFO(First In First Out)

: 준비상태 큐에 도착한 순서에 따라 차례로 CPU를 할당

: 대기시간 - 프로세스가 대기한 시간

: 반환시간 - 프로세스의 대기 시간과 실행시간의 합

* SJF(Shortest Job First, 단기 작업 우선)

: 준비상태 큐에서 실행시간이 가장 짧은 프로세스에게 먼저 CPU를 할당

* HRN(Hightest Response-ration Next)

: 대기시간과 서비스(실행) 시간을 이용하는 기법

: 우선순위 계산식 = (대기시간 + 서비스 시간) / 서비스 시간

### 선점(Preemptive) 스케줄링

* 우선순위가 높은 다른 프로세스가 CPU를 강제로 빼앗아 사용할 수 있는 스케줄링 기법

## 환경 변수

### 환경 변수(Environment Variable)

* 시스템 소프트웨어의 동작에 영향을 미치는 동적인 값들의 모임

### Windows의 주요 환경 변수

* 변수명 앞뒤에 %를 입력해야 함

### UNIX/LINUX의 주요 환경 변수

* 변수명 앞에 $를 입력해야 함

## 운영체제 기본 명령어

1. **Windows 기본 명령어**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DIR | COPY | DEL | TYPE | REN | MD | CD | CLS |
| ATTRIB | FIND | CHKDSK | FORMAT | MOVE |  |  |  |

1. **UNIX/LINUX 기본 명령어**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| cat | cd | chmod | chown | cp | rm | find | fsck |
| kill | fork | killall | ls | mkdir | rmdir | mv | ps |
| pwd | top | who |  |  |  |  |  |

## 인터넷

### 인터넷(Internet)

* 전 세계 수많은 컴퓨터와 네트워크들이 연결된 광범위한 컴퓨터 통신망

### IP 주소(Internet Protocol Address)

* 인터넷에 연결된 모든 컴퓨터 자원을 구분하기 위한 고유한 주소
* 8비트씩 4부분, 총 32비트로 구성됨
* A Class : 국가나 대형 통신망
* B Class : 중대형 통신망
* C Class : 소규모 통신망
* D Class : 멀티캐스트 용
* E Class : 실험적 주소이며 공용되지 않음

### 서브네팅(Subnetting)

* 네트워크 주소를 여러 개의 작은 네트워크로 나누어 사용하는 것
* 서브넷 마스크 : 네트워크 주소와 호스트 주소를 구분하기 위한 비트

### IPv6(Internet Protocol version 6)

* 현재 사용하고 있는 IPv4의 주소 부족 문제를 해결하기 위해 개발됨
* 인증성, 기밀성, 데이터 무결성의 지원으로 보안 문제 해결
* 유니캐스트(Unicast) : 1 대 1 통신
* 멀티캐스트(Multicast) : 1 대 다 통신
* 애니캐스트(Anycast) : 단일 송신자와 가장 가까이 있는 단일 수신자 간의 통신(1 대 1에 사용)

### 도메인 네임(Domain Name)

* 숫자로 된 IP 주소를 문자 형태로 표현

## OSI 참조 모델

### 1) OSI(Open System Interconnection) 참조 모델

* ISO(국제표준화기구)에서 제안한 통신 규약(Protocol)

|  |  |  |  |
| --- | --- | --- | --- |
| 계층 | 관련 장비 | 표준 | 설명 |
| 물리 계층 (Physical Layer) | 리피터, 허브 | R S - 2 3 2 C , X.21 | * 전송에 필요한 두 장치 간의 실제 접속과 절단 등   기계적, 전기적, 기능적, 절차적 특성에 대한 규칙 정의   * 물리적 전송 매체와 전송 신호 방식을 정의 |
| 데이터 링크 계층 (Data Link Layer) | 랜카드, 브리지, 스위치 | HDLC, LAPB, LLC, MAC, LAPD, PPP | * 두 개의 인접한 개방 시스템들 간에 신뢰성 있고   효율적인 정보 전송을 할 수 있도록 시스템 간 연 결 설정과 유지 및 종료 담당   * 흐름 제어 기능, 프레임 동기화 기능, 오류 지어   기능, 순서 제어 기능 |
| 네트워크 계층  (Network Layer, 망 계층) | 라우터 | X.25, IP | * 개방 시스템들 간의 네트워크 연결을 관리하는 기 능과 데이터의 교환 및 중계 기능 * 경로 설정(Routing), 데이터 교환 및 중계, 트래 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | 픽 제어, 패킷 정보 전송 |
| 전송 계층 (Transport Layer) | 게이트웨이 | TCP, UDP | * 논리적 안정과 균일한 데이터 전송 서비스를 제공   함으로써 종단(End-to-End) 간에 투명한 데이터 전송을 가능하게 함   * 하위 3계층과 상위 3계층의 인터페이스를 담당 * 주소 설정, 다중화, 오류 제어, 흐름 제어 수행 |
| 세션 계층 (Session Layer) |  |  | * 송, 수신 측 간의 관련성을 유지하고 대화 제어를   담당   * 동기점 : 오류가 있는 데이터의 회복을 위해 사용 |
| 표현 계층 (Presentation Layer) |  |  | - 응용 계층으로부터 받은 데이터를 세션 계층에 보  내기 전에 통신에 적당한 형태로 변환하고, 세션  계층에서 받은 데이터는 응용 계층에 맞게 변환 |
| 응용 계층 (Application Layer) |  |  | * 사용자(응용 프로그램)가 OSI 환경에 접근할 수   있도록 서비스 제공   * 응용 프로세스 간의 정보 교환, 전자 사서함, 파   일 전송, 가상 터미널 등의 서비스 제공 |

## 네트워크 관련 장비

### 네트워크 인터페이스 카드(NIC; Network Interface Card)

* 컴퓨터와 컴퓨터 또는 컴퓨터와 네트워크를 연결하는 장치
* 이더넷 카드(LAN 카드) 혹은 네트워크 어뎁터라고도 함

### 허브(Hub)

* 회선을 통합하여 관리하며 리피터의 역할을 포함함
* 더미 허브(Dummy Hub) : 단순히 연결
* 스위칭 허브(Switching Hub) : 지능형 허브

### 리피터(Repeater)

* 수신한 신호를 재생시키거나 출력 전압을 높여 전송하는 장치

### 브리지(Bridge)

* LAN과 LAN을 연결하거나 LAN 안에서의 컴퓨터 그룹을 연결하는 장치

### 스위치(Switch)

* LAN과 LAN을 연결하여 훨씬 더 큰 LAN을 만드는 장치

### 라우터(Router)

* 브리지와 같이 LAN과 LAN의 연결 기능에 데이터 전송의 최적 경로를 선택하는 기능이 추가된 장치

### 게이트웨이(Gateway)

* OSI 전 계층의 프로토콜 구조가 다른 네트워크를 연결하는 장치(출입구 역할)

## TCP/IP

### 프로토콜(Protocol)

* 데이터 교환을 원활하게 수행할 수 있도록 표준화시켜 놓은 통신 규약
* 기본 요소 : 구문(Syntax), 의미(Semantics), 시간(Timing)

### TCP/IP(Transmission COntrol Protocol/Internet Protocol)

* 서로 다른 기종의 컴퓨터들이 데이터를 주고받을 수 있도록 하는 표준 프로토콜
* TCP : OSI 7계층의 전송 계층에 해당

: 신뢰성 있는 연결형 서비스를 제공

: 패킷의 다중화, 순서 제어, 오류 제어, 흐름 제어 기능을 제공

* IP : OSI 7계층의 네트워크 계층에 해당함

: 데이터그램을 기반으로 하는 비연결형 서비스를 제공

: 패킷의 분해/조립, 주소 지정, 경로 선택 기능을 제공

1. **TCP/IP의 구조**

|  |  |  |
| --- | --- | --- |
| OSI | TCP/IP | 기능 |
| 응용 계층  표현 계층  세션 계층 | 응용 계층 | * 응용 프로그램 간의 데이터 송, 수신 제공 * TELNET, FTP, SMTP, SNMP, DNS, HTTP 등 |
| 전송 계층 | 전송 계층 | * 호스트들 간의 신뢰성 있는 통신 제공 * TCP, UDP, RTCP |
| 네트워크 계층 | 인터넷 계층 | * 데이터 전송을 위한 주소 지정, 경로 설정을 제공 * IP, ICMP, IGMP, ARP, RARP |
| 데이터 링크 계층  물리 계층 | 네트워크 액세스 계층 | * 실제 데이터(프레임)를 송, 수신하는 역할 * Ethrernet, IEE 802, HDLC, X.25, RS-232C, ARQ 등 |

1. **응용 계층의 주요 프로토콜**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FTP  (File Transfer Protocol) | | | 컴퓨터와 컴퓨터 또는 컴퓨터와 인터넷 사이에서 파일을 주고 받을 수 있도  록 하는 원격 파일 전송 프로토콜 | |
| SMTP  (Simple Mail Transfer Protocol) | | | 전자 우편을 교환하는 서비스 | |
| TELNET | | | 멀리 떨어져 있는 컴퓨터에 접속하여 자신의 컴퓨터처럼 사용할 수 있도록  해주는 서비스 | |
| SNMP  (Simple  Protocol) | Network | Management | TCP/IP의 네트워크 관리 프로토콜로, 라우터나 허브 등 네트워크 기기의 네 트워크 정보를 네트워크 관리 시스템에 보내는 데 사용되는 표준 통신 규약 | |
| DNS  (Domain Name System) | | | 도메인 | 네임을 IP 주소로 매핑(Mapping)하는 시스템 |
| HTTP  (Hyper Text Transfer Protocol) | | | World | Wid Web(WWW)에서 HTML 문서를 송수신하기 위한 표준 프로토콜 |

1. **전송 계층의 주요 프로토콜**

|  |  |
| --- | --- |
| TCP  (Transmisson Control Protocol) | * 양방향 연결형 서비스를 제공 * 가상 회선 연결 형태의 서비스를 제공 * 순서 제어, 오류 제어, 흐름 제어 기능을 함 * 스트림 위주, 투명성 보장 |
| UDP  (User Datagram  Protocol) | * 비연결형 서비스 제공 * TCP에 비해 상대적으로 단순한 헤더 구조를 가짐 * 실시간 전송에 유리하며 신뢰성보다는 속도가 중요시되는 네트워크에 사용됨 |
| RTCP  (Real-Time  Control Protocol) | * RTP 패킷의 전송 품질을 제어하기 위한 제어 프로토콜 * 주기적으로 제어 정보 전송 * 최소한의 제어와 인증 기능만 제공 |

1. **인터넷 계층의 주요 프로토콜**

|  |  |
| --- | --- |
| IP(Internet Protocol) | 전송할 데이터에 주소를 지정하고 경로를 설정하는 기능 |
| ICMP(Internet Control  Message Protocol) | IP와 조합하여 통신중에 발생하는 오류의 처리와 전송 경로 변경 등을 위한 제어 메시  지 관리 |
| IGMP(Internet Group  Management Protocol) | 멀티캐스트를 지원하는 호스트나 라우터 사이에서 멀티캐스트 그룹 유지를 위해 사용  됨 |
| ARP(Address Resolution  Protocol) | 호스트의 IP 주소를 호스트와 연결된 네트워크 접속 장치의 물리적 주소(MAC  Address)로 바꿈 |
| RARP(Reverse Address  Resolution Protocol) | ARP와 반대로 물리적 주소를 IP 주소로 변환하는 기능을 함 |

1. **네트워크 액세스 계층의 주요 프로토콜**

|  |  |
| --- | --- |
| Ethernet(IEE 802.3) | CSMA/CD 방식의 LAN |
| IEE 802 | LAN을 위한 표준 프로토콜 |
| HDLC | 비트 위주의 데이터 링크 제어 프로토콜 |
| X.25 | 패킷 교환망을 통한 DTE와 DCE 간의 인터페이스를 제공하는 프로토콜 |
| RS-232C | 공중 전화 교환망(PSTN)을 통한 DTE와 DCE 간의 인트페이스를 제공하는 프로토콜 |

## 네트워크 관련 신기술

|  |  |
| --- | --- |
| IoT  (Internet of Things,  사물 인터넷) | 정보 통신 기술을 기반으로 실세계와 가상 세계의 다양한 사물들을 인터넷으로 서로 연결하여 진보된 서비스를 제공하기 위한 서비스 기반 기술 |
| M2M(Machine to Machine,  사물 통신) | 무선 통신을 이용한 기계와 기계 사이의 통신 |
| 모바일 컴퓨팅  (Mobile Computing) | 휴대형 기기로 이동하면서 자유로이 네트워크에 접속하여 업무를 처리할 수 있  는 환경 |
| 클라우드 컴퓨팅  (Cloud Computing) | 각종 컴퓨팅 자원을 중앙 컴퓨터에 두고 인터넷 기능을 갖는 단말기로 언제 어  디서나 인터넷을 통해 컴퓨터 작업을 수행할 수 있는 가상화된 환경 |
| 그리드 컴퓨팅  (Grid Computing) | 지리적으로 분산되어 있는 컴퓨터를 초고속 인터넷망으로 연결하여 공유함으로  써 하나의 고성능 컴퓨터처럼 활용하는 기술 |
| 모바일 클라우드 컴퓨팅  (MCC; Mobile Cloud  Computing) | 소비자와 소비자의 파트너가 클라우더 서비스를 이용하여 모바일 기기로 클라 우드 컴퓨팅 인프라를 구성하여 여러 가지 정보와 자원을 공유하는 ICT 기술 |
| 인터클라우드 컴퓨팅 (Inter-Cloud Computing) | 각기 다른 클라우드 서비스를 연동하거나 컴퓨팅 자원의 동적 할당이 가능하도  록 여러 클라우드 서비스 제공자들이 제공하는 클라우드 서비스나 자원을 연결  하는 기술 |
| 메시 네트워크  (Mesh Network) | 차세대 이동통신, 홈네트워킹, 공공 안전 등 특수 목적을 위한 새로운 방식의  네트워크 기술 |
| 와이선(Wi-SUN) | 스마트 그리드와 같은 장거리 무선 통신을 필요로 하는 사물 인터넷(IoT) 서비  스를 위한 저전력 장거리 통신 기술 |
| NDN(Named Data Networking) | 콘텐츠 자체의 정보와 라우터 기능만으로 데이터 전송을 수행하는 기술 |
| NGN(Next Generation  Network, 차세대 통신망) | ITU-T에서 개발하고 있는 유선망 기반의 차세대 통신망 |
| SDN(Sofware Defined  Networking, 소프트웨어 정의  네트워킹) | 네트워크를 컴퓨터처럼 모델링하여 여러 사용자가 각각의 소프트웨어로 네트워 킹을 가상화하여 제어하고 관리하는 네트워크 |
| NFC(Near Field  Communication,  근거리 무선 통신) | 고주파(HF)를 이용한 근거리 무선 통신 기술 |
| UWB(Ultra WideBand,  초광대역) | 짧은 거리에서 많은 양의 디지털 데이터를 낮은 전력으로 전송하기 위한 무선  기술로 무선 디지털 펄스라고도 함 |
| 피코넷(PICONET) | 여러 개의 독립된 통신장치가 불루투스 기술이나 UWB 통신 기술을 사용하여  통신망을 형성하는 무선 네트워크 기술 |
| WBAN(Wireless Body  Area Network) | 웨어러블 또는 몸에 심는 형태의 센서나 기기를 무선으로 연결하는 개인 영역  네트워킹 기술 |
| GIS(Geographic Information  System, 지리 정보 시스템) | 지리적인 자료를 수집, 저장, 분석, 출력할 수 있는 컴퓨터 응용 시스템 |
| USN(Ubiquitous Sensor | 각종 센서로 수집한 정보를 무선으로 수집할 수 있도록 구성한 네트워크 |

|  |  |
| --- | --- |
| Network,  유비쿼터스 센서 네트워크) |  |
| SON(Self Organizing Network,  자동 구성 네트워크) | 주변 상황에 맞추어 스스로 망을 구성하는 네트워크 |
| 애드 훅 네트워크  (Ad-hox Network) | 재난 현장과 같이 별도의 고정된 유선망을 구축할 수 없는 장소에서 모바일 호  스트만을 이용하여 구성한 네트워크 |
| 네트워크 슬라이싱 (Network Slicing) | 네트워크에서 하나의 물리적인 코어 네트워크 인프라(Infrastructure)를 독립된  다수의 가상 네트워크로 분리하여 각각의 네트워크를 통해 다양한 고객 맞춤형  서비스를 제공하는 목적으로 하는 네트워크 기술 |
| 저전력 블루투스 기술  (BLE; Bluetooth Low Energy) | 일반 블루투스와 동일한 2.4GHz 주파수 대역을 사용하지만 연결되지 않은 대  기 상태에서는 절전모드를 유지하는 기술 |
| 지능형 초연결망 | 4차 산업혁명 시대를 맞아 새로운 변화에 따라 급격하게 증가하는 데이터 트래  픽을 효과적으로 수용하기 위해 시행되는 과학기술정보통신부 주관 사업 |
| 파장 분할 다중화  (WDM, Wavelength Division  Multiplexing) | 광섬유를 이용한 통신기술의 하나로, 파장이 서로 다른 복수의 신호를 보냄으 로써 여러 대의 단말기가 동시에 통신 회선을 사용할 수 있도록 하는 것 |
| 소프트웨어 정의 데이터 센터  (SDDC, Software Defined  Data Center) | 데이터 센터의 모든 자원을 가상화하여 인력의 개입없이 소프트웨어 조작만으 로 관리 및 제어되는 데이터 센터를 의미 |
| 개방형 링크드 데이터  (LOD, Linked Open Data) | Linked Data와 Open Data의 합성어로, 누구나 사용할 수 있도록 웹상에 공  개된 연계 데이터를 의미 |

1. **네트워크 구축**

### 네트워크(Network)

* 두 대 이상의 컴퓨터를 연결하여 자원을 공유하는 것

### 성형(Star, 중앙 집중형)

* 중앙에 중앙 컴퓨터가 있고, 이를 중심으로 단말장치들이 연결되는 중앙 집중식의 네트워크 구성 형태

### 링형(RIng, 루프형)

* 컴퓨터와 단말장치들을 서로 이웃하는 것끼리 연결시킨 포인트 투 포인트 방식의 구성 형태

### 버스형(Bus)

* 한 개의 통신 회선에 여러 대의 단말장치가 연결되어 있는 형태

### 계층형(Tree, 분산형)

* 중앙 컴퓨터와 일정 지역의 단말장치까지는 하나의 통신 회선으로부터 연결시키고, 이웃하는 단말장치는 일정 지역 내에 설치된 중간 단말장치로부터 다시 연결시키는 형태

### 망형(Mesh)

* 모든 지점의 컴퓨터와 단말장치를 서로 연결하는 형태
* 노드의 수가 n개일 때 n(n-1)/2개의 회선이 필요하고 노드당 n-1개의 포트가 필요

### 네트워크 분류

* 근거리 통신망(LAN; Local Area Network)

: 비교적 가까운 거리에 있는 자원을 연결하여 구성

: 주로 버스형이나 링형 구조 사용

* 광대역 통신망(WAN; Wide Area Network)

: 멀리 떨어진 사이트들을 연결하여 구상

### LAN의 표준안

- 책 참고(2-446P)

### NAT(Network Address Translation, 네트워크 주소 변환)

* 한 개의 정식 IP 주소에 대량의 가상 사설 IP 주소를 할당 및 연결하는 기능

## 스위치

### 스위치(Switch) 분류

* LAN과 LAN을 연결하여 훨씬 더 큰 LAN을 만드는 장치

|  |  |  |
| --- | --- | --- |
| L2 스위치 | OSI 2계층 | MAC 주소를 기반으로 프레임을 전송함 |
| L3 스위치 | OSI 3계층 | L2 스위치에 라우터 기능이 추가된 것으로, IP 주소를 기반으로 패킷을 전송함 |
| L4 스위치 | OSI 4계층 | 로드밸런서가 달린 L3 스위치로, IP 주소 및 TCP/UDP를 기반으로 사용자들의 요  구를 서버의 부하가 적은 곳에 배분하는 로드밸런싱 기능을 제공함 |
| L7 스위치 | OSI 7계층 | IP 주소, TCP/UDP 포트 정보에 패킷 내용까지 참조하여 세밀하게 로드밸런싱함 |

### 스위칭(Switch) 방식

* Store and Forwarding: 데이터를 모두 받은 후 스위칭하는 방식
* Cut-through : 데이터의 목적지 주소만을 확인한 후 바로 스위칭하는 방식
* Fragment Free : Store and Forwarding과 Cut-through 방식의 장점을 결합한 방식

### 백본 스위치(Backbone Switch)

* 네트워크들을 연결할 때 중추적 역할을 하는 네트워크를 백본이라 하고, 백본에서 스위칭 역할을 하는 장비를 백본 스위치라고 함

1. **Hierarchical 3 Layer 모델**

|  |  |
| --- | --- |
| 액세스 계층  (Access Layer) | 사용자가 네트워크에 접속할 때 최초로 연결되는 지점으로, 사용자들로부터 오는 통신을  집약해서 디스트리뷰션 계층으로 전송함, L2 스위치 사용 |
| 디스트리뷰션 계층  (Distribution Layer) | 액세스 계층의 장치들이 연결되는 지점으로, 액세스 계층에서 오는 통신을 집약해서 코어  계층으로 전송함, 라우터, L3 스위치 사용 |
| 코어 계층(Core Layer) | 디스트리뷰션 계층에서 오는 통신을 집약해 인터넷에 연결하는 계층으로 백본 계층이라  고도 함, 백본 스위치 사용 |

## 경로 제어/트래픽 제어

### 경로 제어(Routing)

* 전송 경로 중에서 최적 패킷 교환 경로를 결정하는 기능(네비게이션)

### 경로 제어 프로토콜(Routing Protocol)

|  |  |
| --- | --- |
| IGP  (Interior Gateway Protocol, 내부 게이트웨이 프로토콜) | * 하나의 자율 시스템(AS) 내의 라우팅에 사용되는 프로토콜 * RIP(Routing Information Protocol)   : 현재 가장 널리 사용되는 라우팅 프로토콜로 거리 벡터 라우팅 프로토콜로 도 불리며, 최단 경로 탐색에 Bellman-Ford 알고리즘이 사용됨   * OSPF(Open Shortest Path First Protocol)   : RIP의 단점을 해결하여 새로운 기능을 지원하는 인터넷 프로토콜로, 대규모  네트워크에서 많이 사용됨 |
| EGP  (Exterior Gateway Protocol, 외부 게이트웨이  프로토콜) | 자율 시스템간의 라우팅, 즉 게이트웨이 간의 라우팅에 사용되는 프로토콜 |
| BGP(Border Gateway  Protocol) | 자율 시스템 간의 라우팅 프로토콜로, EGP의 단점을 보완하기 위해 만들어졌음 |

1. **트래픽 제어(Traffic Control)**

* 전송되는 패킷의 흐름 또는 그 양을 조절하는 기능

### 흐름 제어(Flow Control)

* 송, 수신 측 사이에 전송되는 패킷의 양이나 속도를 규제하는 기능
* 정지-대기(Stop-and-Wait)

: 수신 측의 확인 신호(ACK)를 받은 후에 다음 패킷을 전송하는 방식

* 슬라이딩 윈도우(Sliding Window)

: 확인 신호, 즉 수신 통지를 이용하여 송신 데이터의 양을 조절하는 방식

### 폭주 제어(Congestion Control) = 혼합 제어

* 네트워크 내의 패킷 수를 조절하여 네트워크 오버플로를 방지하는 기능을 함
* 느린 시작(Slow Start)

: 윈도우의 크기를 2배씩 지수적으로 증가시켜 초기에는 느리지만 갈수록 빨라짐

* 혼잡 회피(Congestion Avoidance)

: 느린 시작의 지수적 증가가 임계 값에 도달되면 혼잡으로 간주하고 회피를 피해 윈도우의 크기를 1씩 선형 적으로 증가시켜 혼잡을 예방하는 방식

## SW 관련 신기술

|  |  |
| --- | --- |
| 인공지능(AI; Artificial Intelligence) | 인간의 두뇌와 같이 컴퓨터 스스로 추론, 학습, 판단 등 인간지능적인 작업  을 수행하는 시스템 |
| 뉴럴링크(Neuralink) | 미국의 전기자동차 회사 테슬라(Tesla)의 CEO 일론 머스크(Elon Musk)가  사람의 뇌와 컴퓨터를 결합하는 기술을 개발하기 위해 2017년 3월 설립한  회사 |
| 딥 러닝(Deep Learning) | 인간의 두뇌를 모델로 만들어진 인공 신경망(ANN; Artificial Neural  Network)을 기반으로 하는 기계 학습 기술 |
| 전문가 시스템(Expert System) | 의료 진단 등과 같은 특정 분야의 전문가가 수행하는 고도의 업무를 지원하  기 위한 컴퓨터 응용 프로그램 |
| 증강현실(AR; Augmented Reality) | 실제 촬영한 화면에 가상의 정보를 부가하여 보여주는 기술로, 혼합현실  (MR; Mixed Reality)이라고도 부름 |
| 블록체인(Blockchain) | P2P 네트워크를 이용하여 온라인 금융 거래 정보를 온라인 네트워크 참여  자(Peer)의 디지털 장비에 분산 저장하는 기술 |
| 분산 원장 기술  (DLT; Distributed Ledger  Technology) | 중앙 관리자나 중앙 데이터 저장소가 존재하지 않고 P2P 망내의 참여자들  에게 모든 거래 목록이 분산 저장되어 거래가 발생할 때마다 지속적으로 갱  신되는 디지털 원장을 의미함 |
| 해시(Hash) | 임의의 길이의 입력 데이터나 메시지를 고정된 길이의 값이나 키로 변환하  는 것 |
| 양자 암호키 분배  (QKD; Quantum Key Distribution) | 양자 통신을 위해 비밀키를 분배하여 관리하는 기술 |
| 프라이버시 강화 기술  (PET; Privacy  Enhancing Technology) | - 개인정보 침해 위험을 관리하기 위한 핵심 기술 |
| 공통 평가 기준  (CC; Common Criteria) | * 1999년 6월 8일 ISO 15408 표준으로 채택된 정보 보호 제품 평가 기준 * 정보화 순기능 역할을 보장하기 위해 정보화 제품의 정보 보호 기능과 이   에 대한 사용 환경 등급을 정한 기준 |
| 개인정보 영향평가 제도  (PIA; Privacy Impact Assessment) | 개인 정보를 활용하는 새로운 정보시스템의 도입 및 기존 정보시스템의 중  요한 변경 시 시스템의 구축·운영이 기업의 고객은 물론 국민의 사생활에  미칠 영향에 대해 미리 조사·분석·평가하는 제도 |
| 그레이웨어(Grayware) | 소프트웨어를 제공하는 입장에서는 악의적이지 않은 유용한 소프트웨어라고  주장할 수 있지만 사용자 입장에서는 유용할 수도 있고 악의적일 수도 있는  애드웨어, 트랙웨어, 기타 악성 코드나 악성 공유웨어를 말함 |
| 매시업(Mashup) | 웹에서 제공하는 정보 및 서비스를 이용하여 새로운 소프트웨어나 서비스,  데이터베이스 드을 만드는 기술. 즉 다수의 정보원이 제공하는 콘텐츠를 조  합하여 하나의 서비스로 제공하는 웹 사이트 또는 애플리케이션을 말함 |
| 리치 인터넷 애플리케이션  (RIA; Rich Internet Application) | 플래시 애니메이션 기술과 웹 서버 애플리케이션 기술을 통하여 기존 HTML보다 역동적이도 인터랙티브한 웹 페이지를 제공하는 신개념의 플래 |

|  |  |
| --- | --- |
|  | 시 웹 페이지 제작 기술 |
| 시맨틱 웹(Semantic Web) | 컴퓨터가 사람을 대신하여 정보를 읽고 이해하고 가공하여 새로운 정보를  만들어 낼 수 있도록 이해하기 쉬운 의미를 가진 차세대 지능형 웹 |
| 증발품(Vaporware) | 판매 계획 또는 배포 계획은 발표되었으니 실제로 고객에게 판매되거나 배  포되지 않고 있는 소프트웨어 |
| 오픈 그리드 서비스 아키텍쳐 (OGSA; Open Grid  Service Architecture) | * 기업의 소프트웨어 인프라인 정보시스템을 공유와 재사용이 가능한 서비   스 단위나 컴포넌트 중심으로 구축하는 정보기술 아키텍처   * SOA 기반 애플리케이션 구성 계층   : 표현(Presentation) 계층  : 업무 프로세스(Biz-Process) 계층  : 서비스 중간(Service Intermediary) 계층  : 애플리케이션(Application) 계층  : 데이터 저장(Persistency) 계층 |
| 서비스형 소프트웨어  (SaaS; Software as a Service) | 소프트웨어의 여러 기능 중에서 사용자가 필요로 하는 서비스만 이용할 수  있도록 한 소프트웨어 |
| 소프트웨어 에스크로 (Software Escrow) | 소프트웨어 개발자의 지식재산권을 보호하고 사용자는 저렴한 비요으로 소  프트웨어를 안정적으로 사용 및 유지보수 할 수 있도록 소스 프로그램과 기  술 정보 등을 제3의 기관에 보관하는 것 |
| 복잡 이벤트 처리  (CEP; Complex Event Processing) | 실시간으로 발생하는 많은 사건들 중 의미가 있는 것만을 추출할 수 있도록  사건 발생 조건을 정의하는 데이터 처리 방법 |
| 디지털 트윈(Digital Twin) | - 현실 속의 사물을 소프트웨어로 가상화한 모델 |

1. **소프트웨어 개발 보안**

* 기밀성, 무결성, 가용성을 유지하는 것을 목표로 함
* 나머지 책 참고(2-460P)

## 소프트웨어 개발 보안 활동 관련 법령 및 규정

- 책 참고(2-463P)

## HW 관련 신기술

|  |  |
| --- | --- |
| 고가용성(HA; High Availability) | 긴 시간 동안 안정적인 서비스 운영을 위해 장애 발생 시 즉시 다른 시스  템으로 대체 가능한 환경을 구축하는 메커니즘 |
| 3D Printing  (Three Dimension Printing) | 대상을 평면에 출력하는 것이 아니라 손으로 만질 수 있는 실제 물체로 만  들어내는 것 |
| 4D Printing  (Fourth Dimension Printing) | 특정 시간이나 환경 조건이 갖추어지면 스스로 형태를 변화시키거나 제조  되는 자가 조립(Self-Aseembly) 기술이 적용된 제품을 3D Printing하는  기술 |
| RAID  (Redundant Array of Inexpensive Disk,  Redundant Array of Independent  Disk) | 여러 개의 하드디스크로 디스크 배열을 구성하여 파일을 구성하고 있는 데 이터 블록들을 서로 다른 디스크들에 분산 저장할 경우, 그 블록들을 여러 디스크에서 동시에 읽거나 쓸 수 있으므로 디스크의 속도가 매우 향상되는 데, 이 기술을 RAID라고 함 |
| 4K 해상도 | 차세대 고화질 모니터의 해상도를 지칭하는 용어 |
| 앤 스크린(N-Screen) | N개의 서로 다른 단말기에서 동일한 콘텐츠를 자유롭게 이용할 수 있는 서  비스 |

|  |  |
| --- | --- |
| 컴패니언 스크린  (Companion Screen) | TV 방송 시청 시 방송 내용을 공유하며 추가적인 기능을 수행할 수 있는  스마트폰, 태블릿PC 등을 의미함 |
| 신 클라이언트 PC(Thin Client PC) | 하드디스크나 주변장치 없이 기본적인 메모리만 갖추고 서버와 네트워크로  운용되는 개인용 컴퓨터 |
| 패블릿(Phablet) | 폰(Phone)과 태블릿(Tablet)의 합성어로, 태블릿 기능을 포함한 5인치 이상  의 대화면 스마트폰 |
| C형 유에스비  (Universal Serial Bus Type-C,  USB Type-C, USB-C) | 범용 인터페이스 규격인 USB(Universal Serial Bus)의 표준 중 하나임 |
| 멤스  (MEMS; Micro-Electro Mechanical  Systems) | 초정밀 반도체 제조 기술을 바탕으로 센서, 액추에이터(Actuator) 등 기계  구조를 다양한 기술로 미세 가공하여 전기기계적 동작을 할 수 있도록 한  초미세 장치 |
| 트러스트존 기술 (TrustZone Technology) | 하나의 프로세서(Processor) 내에 일반 애플리케이션을 처리하는 일반 구  역(Normal World)과 보안이 필요한 애플리케이션을 처리하는 보안 구역  (Secure World)으로 분할하여 관리하는 하드웨어 기반의 보안 기술 |
| 엠디스크(M-DISC, Millennial DISC) | 한 번의 기록만으로 자료를 영구 보관할 수 있는 광 저장장치 |
| 멤리스터(Memristor) | 메모리(Memory)와 레지스터(Resister)의 합성어로, 전류의 방향과 양 등  기존의 경험을 모두 기억하는 특별한 소자 |

1. **Secure OS**

### Secure OS

* 보안 기능을 갖춘 커널을 이식하여 외부의 침입으로부터 시스템 자원을 보호하는 운영체제
* 암호적 분리(Cryptographic Separation)

: 내부 정보를 암호화하는 방법

* 논리적 분리(Logical Separation)

: 프로세스의 논리적 구역을 지정하여 구역을 벗어나는 행위를 제한하는 방법

* 시간적 분리(Temploral Separation)

: 동일 시간에 하나의 프로세스만 수행되도록 하여 동시 실행으로 발생하는 보안 취약점을 제거하는 방법

* 물리적 분리(Physical Separation)

: 사용자별로 특정 장비만 사용하도록 제한하는 방법

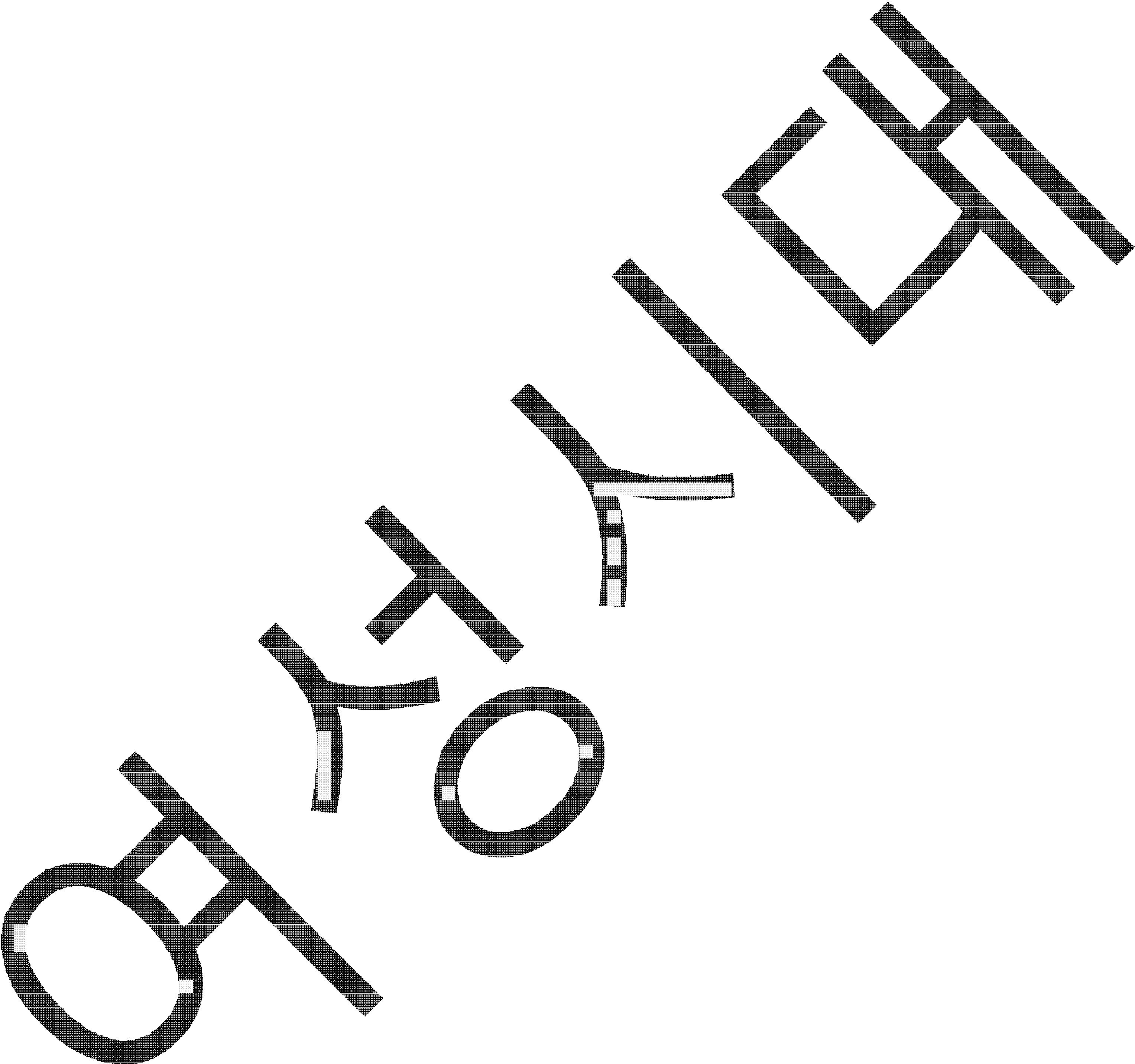
### 참조 모니터(Reference Monitor)

* 보호 대상 객체에 대한 접근 통제를 수행하는 추상머신이며, 이것을 실제로 구현한 것이 보안 커널
* 참조 모니터 보안 커널의 특징 : 격리성, 검증가능성, 완전성

## DB 관련 신기술

|  |  |
| --- | --- |
| 빅데이터(Big Data) | 기존의 관리 방법이나 분석 체계로는 처리하기 어려운 막대한 양의 정형 또  는 비정형 데이터 집합 |
| 브로드 데이터(Broad Data) | 다양한 채널에서 소비자와 상호 작용을 통해 생성된 것으로, 기업 마케팅에  있어 효율적이고 다양한 데이터이며, 이전에 사용하지 않거나 알지 못했던  새로운 데이터나 기존 데이터에 새로운 가치가 더해진 데이터 |
| 메타 데이터(Meta Data) | * 일련의 데이터를 정의하고 설명해주는 데이터 * 컴퓨터에서는 데이터 사전의 내용, 스키마 등을 의미함 * HTML 문서에서는 메타 태그 내의 내용이 메타 데이터임 |
| 디지털 아카이빙(Digital Archiving) | - 디지털 정보 자원을 장기적으로 보존하기 위한 작업 |
| 하둡(Hadoop) | * 오픈소스를 기반으로 한 분산 컴퓨팅 플랫폼 * 일반 PC급 컴퓨터들로 가상화된 대현 스토리지를 형성하고 그 안에 보관 |

|  |  |
| --- | --- |
|  | 된 거대한 데이터 세트를 병렬로 처리할 수 있도록 개발된 자바 소프트웨  어 프레임워크 |
| 맵리듀스(MapReduce) | 대용량 데이터를 분산 처리하기 위한 목적으로 개발된 프로그래밍 모델 |
| 타조(Tajo) | 오픈 소스 기반 분산 컴퓨팅 플랫폼인 아파치 하둡(Apache Hadoop) 기반  의 분산 데이터 웨어하우스 프로젝트 |
| 데이터 다이어트(Data Diet) | 데이터를 삭제하는 것이 아니라 압축하고, 중복된 정보는 중복을 배제하고,  새로운 기준에 따라 나누어 저장하는 작업 |
| 데이터 마이닝(Data Mining) | 대량의 데이터를 분석하여 데이터에 내재된 변수 사이의 상호 관계를 규명  하여 일정한 패턴을 찾아내는 기법 |
| OLAP  (Online Analytical Processing) | * 다차원으로 이루어진 데이터로부터 통계적인 요약 정보를 분석하여 의사   결정에 활용하는 방식   * OLAP 연산 : Roll-up, Drill-down, Drill-through, Drill-across,   Pivoting, Slicing, Dicing |

1. **회복/병행제어**

### 회복(Recovery)

* 데이터베이스가 손상되었을 때 손상되기 이전의 정상 상태로 복구하는 작업
* 회복 기법의 종류

: 연기 갱신 기법(Deferred Upadtae)

* + 트랜잭션이 성공적으로 완료될 때까지 데이터베이스에 대해 실질적인 갱신을 연기하는 방법
  + Redo 작업만 가능

: 즉각 갱신 기법(Immediate Update)

* + 트랜잭션이 데이터를 갱신하면 부분 완료되기 전이라도 즉시 실제 DB에 반영하는 방법
  + Redo와 Undo 모두 사용 가능

: 그림자 페이지 대체 기법(Shadow Paging)

* + 갱신 이전의 DB를 일정 크기의 페이지 단위로 구성하여 각 페이지마다 복사본인 그림자 페이지를 별도 보관해놓고, 실제 페이지를 대성으로 갱신 작업을 수행하다가 장애가 발생하여 트랜잭션 작 업을 Rollback시킬 때는 갱신 이후의 실제 페이지 부분을 그림자 페이지로 대체하여 회복시키는 기법

: 검사점 기법(Check Point)

* + 트랜잭션 실행 중 특정 단계에서 재실행할 수 있도록 갱신 내용이나 시스템에 대한 상황 등에 관 한 정보와 함께 검사점을 로그에 보관해두고, 장애 발생 시 트랜잭션 전체를 철회하지 않고 검사 점부터 회복 작업을 수행하여 회복 시간을 절약하도록 하는 기법

### 병행제어(Concurrency Control)

* 동시에 실행되는 트랜잭션 간의 상호작용을 제어하는 것
* 병행제어 기법의 종류

: 로킹(Locking)

* + 트랜잭션들이 어떤 로킹 단위를 액세스하기 전에 Lock을 요청해서 Lock이 허락되어야만 그 로킹 단위를 액세스할 수 있도록 하는 기법

: 타임 스탬프 순서(Time Stamp Ordering)

* + 트랜잭션과 트랜잭션이 읽거나 갱신한 데이터에 대해 트랜잭션이 실행을 시작하기 전에 시간표 (Time Stamp)를 부여하여 부여된 시간에 따라 트랜잭션 작업을 수행하는 기법

: 최적 병행 수행(검증 기법, 확인 기법, 낙관적 기법)

* + 병행수행하고자 하는 대부분의 트랜잭션이 판독 전용(Read Only) 트랜잭션일 경우, 트랜잭션 간의 충돌률이 매우 낮아서 병행제어 기법을 사용하지 않고 실행되어도 이 중의 많은 트랜잭션은 시스 템 상태를 일관성 있게 유지한다는 점을 이용한 기법

: 다중 버전 기법

* + 타임 스탬프의 개념을 이용하는 기법, 다중 버전 타임 스탬프 기법이라고도 함

### 로킹 단위(Locking Granularity)

* 한꺼번에 로킹할 수 있는 객체의 크기

## 교착상태

### 교착상채(Dead Lock)

* 서로 다른 프로세스가 점유하고 있는 자원을 요구하며 무한정 기다리는 현상

### 교착상태 발생의 필요 충분 조건

* 다음 네 가지 조건이 모두 충족되어야 함
* 상호 배제(Mutual Exclusion)

: 한 번에 한 개의 프로세스만이 공유 자원을 사용할 수 있어야 함

* 점유와 대기(Hold and Wait)

: 최소한 하나의 자원을 점유하고 있으면서 다른 프로세스에 할당되어 사용되고 있는 자원을 추가로 점유하기 위해 대기하는 프로세스가 있어야 함

* 비선점(Non-preemption)

: 다른 프로세스에 할당된 자원은 사용이 끝날 때까지 강제로 빼앗을 수 없어야 함

* 환형 대기(Circular Wait)

: 공유 자원과 공유 자원을 사용하기 위해 대기하는 프로세스들이 원형으로 구성되어 있어 자기에게 할당된 자원을 점유하면서 앞이나 뒤에 있는 프로세스의 자원을 요구해야 함

### 교착상태의 해결 방법

* 예방 기법(Prevention)

: 교착상태가 발생하지 않도록 사전에 시스템을 제어하는 방법(자원 낭비 가장 심함)

* 회피 기법(Avoidance)

: 교착상태가 발생할 가능성을 배제하지 않고 교착상태가 발생하면 적절히 피해나가는 방법

: 주로 은행원 알고리즘(Banker’s Algorithm)이 사용됨

* 발견 기법(Detection)

: 시스템에 교착상태가 발생했는지 점검하여 교착상태에 있는 프로세스와 자원을 발견하는 것

* 회복 기법(Recovery)

: 교착상태를 일으킨 프로세스를 종료하거나 교착상태의 프로세스에 할당된 자원을 선점하여 프로세스나 자원 을 회복하는 것

## 데이터 표준화

### 데이터 표준화

* 시스템을 구성하는 데이터 요소의 명칭, 정의, 형식, 규칙에 대한 원칙을 수립하고 적용하는 것

### 데이터 표준화 구성요소

* 데이터 표준

: 데이터 모델이나 DB에서 정의할 수 있는 모든 오브젝트를 대상으로 데이터 표준화를 수행함

: 표준 단어, 표준 도메인, 표준 코드, 표준 용어

* 데이터 관리 조직

: 데이터 표준 원칙이나 데이터 표준의 준수 여부 등을 관리하는 사람들로, 대표적으로 데이터 관리자가 있음

* 데이터 표준화 절차

: 데이터 표준화 요구사항 수집 -> 데이터 표준 정의 -> 데이터 표준 확장 -> 데이터 표준 관리

### 데이터 표준화의 대상

* 데이터 명칭, 데이터 정의, 데이터 형식, 데이터 규칙

# 12장. 제품 소프트웨어 패키징

1. **소프트웨어 패키징**(빌드랑 헷갈리지 말기!)

* 모듈별로 생성한 실행 파일들을 묶어 배포용 설치 파일을 만드는 것

### 1) 패키징 작업 순서

* 기능 식별 -> 모듈화 -> 빌드 진행 -> 사용자 환경 분석 -> 패키징 및 적용 시험 -> 패키징 변경 개선 -> 배포

## 릴리즈 노트 작성

### 릴리즈 노트(Release Note)

* 소프트웨어 개발 과정에서 정리된 릴리즈(배포) 정보를 최종 사용자인 고객과 공유하기 위한 문서

### 릴리즈 노트 작성 항목

* Header, 개요, 목적, 문제 요약, 재현 항목, 수정/개선 내용도, 사용자 영향도, SW 지원 영향도, 노트, 면책조항, 연 락처

### 릴리즈 노트 작성 순서

* 모듈 식별 -> 릴리즈 정보 확인 -> 릴리즈 노트 개요 작성 -> 영향도 체크

-> 정식 릴리즈 노트작성 -> 추가 개선 항목 식별

## 디지털 저작권 관리(DRM)

### 저작권

* 창작자가 가지는 배타적 독점적 권리로, 타인의 침해를 받지 않을 고유한 권한

### 디지털 저작권 관리(DRM; Digital Right Management)

* 저작권자가 배포한 디지털 콘텐츠가 저작권자의 의도한 용도로만 사용되도록 디지털 콘텐츠 관리 및 보호 기술

### 디지털 저작권 관리의 흐름 및 구성 요소

1. **디지털 저작권 관리의 기술 요소**

* 암호화, 키 관리, 암호화 파일 생성, 식별 기술, 저작권 표현, 정책 관리, 크랙 방지, 인증

## 소프트웨어 설치 매뉴얼 작성

* 개발 초기에서부터 적용된 기준이나 사용자가 소프트웨어를 설치하는 과정에 필요한 내용을 기록한 설명서와 안내서

### 1) 설치 매뉴얼 작성 순서

* 기능 식별 -> UI 분류 -> 설치 파일/백업 파일 확인 -> Uninstall 절차 확인

-> 이상 Case 확인 -> 최종 매뉴얼 적용

## 소프트웨어 사용자 매뉴얼 작성

* 사용자가 소프트웨어를 사용하는 과정에서 필요한 내용을 문서로 기록한 설명서와 안내서

### 1) 사용자 매뉴얼 작성 순서

* 기능 식별 -> 사용자 화면 분류 -> 사용자 환경 파일 확인 -> 초기화 절차 확인

-> 이상 Case 확인 -> 최종 매뉴얼 적용

## 소프트웨어 버전 등록

### 소프트웨어 패키지의 형상 관리

* 개발 과정에서 소프트웨어의 변경 사항을 관리하기 위해 개발된 일련의 활동

### 형상 관리 기능

|  |  |
| --- | --- |
| 형상 식별 | 형상 관리 대상에 이름과 관리 번호를 부여하고 계층 구조로 구분하여 수정 및 추적이 용이하도록 하는  작업 |
| 버전 제어 | 소프트웨어 업그레이드나 유지 보수 과정에서 생성된 다른 버전의 형상 항목을 관리하고, 이를 위해 특  정 절차와 도구를 결합시키는 작업 |
| 형상 통제 | 식별된 형상 항목에 대한 변경 요구를 검토하여 현재의 기준선이 잘 반영될 수 있도록 조정하는 작업 |
| 형상 감사 | 기준선의 무결성을 평가하기 위해 확인, 검증, 검열 과정을 통해 공식적으로 승인하는 작업 |
| 형상 기록 | 형상의 식별, 통제, 감사 작업의 결과를 기록, 관리하고 보고서를 작성하는 작업 |

1. **소프트웨어의 버전 등록 관련 주요 기능**

* 저장소(Repository)

: 최신 버전의 파일들과 변경 내역에 대한 정보들이 저장되어 있는 곳

* 가져오기(Import)

: 버전 관리가 되고 있지 않은 아무것도 없는 저장소에 처음으로 파일을 복사함

* 체크아웃(Check-Out)

: 프로그램을 수정하기 위해 저장소에서 파일을 받아옴

* 체크인(Check-In)

: 체크아웃한 파일의 수정을 완료한 후 저장소의 파일을 새로운 버전으로 갱신함

* 커밋(Commit)

: 체크인을 수행할 때 이전에 갱신된 내용이 있는 경우에는 충돌을 알리고 Diff 도구를 이용해 수정한 후 갱 신을 완료함

* 동기화(Update)

: 저장소에 있는 최신 버전으로 자신의 작업 공간을 동기화함

### 소프트웨어 버전 등록 과정

* Import -> Check-Out -> Commit -> Update -> Diff

## 소프트웨어 버전 관리 도구

### 공유 폴더 방식

* 버전 관리 자료가 지역 컴퓨터의 공유 폴더에 저장되어 관리되는 방식

### 클라이언트/서버 방식

* 버전 관리 자료가 서버에 저장되어 관리되는 방식

### 분산 저장소 방식

* 버전 관리 자료가 하나의 원격 저장소와 분산된 개발자 PC의 지역 저장소에 함께 저장되어 관리되는 방식

### Subversion(서브버전, SVN)

* CVS를 개선한 것으로, 아파치 재단에서 2000년에 발표함
* 클라이언트/서버 구조로, 서버에는 최신 버전의 파일들과 변경 사항이 관리됨
* 주요 명령어

: add, commit, update, checkout, lock/unlock, import, export, info, diff, merge

### Git(깃)

* 리누스 토발즈가 2005년 리눅스 커널 개발에 사용할 관리 도구로 개발한 이후 주니오 하마노에 유지 보수 되고 있 음
* 분산 버전 관리 시스템으로 2개의 저장소, 즉 지역 저장소와 원격 저장소가 존재함
* 주요 명령어

: add, commit, branch, checkout, merge, init, remote add, push, fetch, clone fork

## Git(깃) 명령어 활용

- 책 참고(2-531P)

## 빌드 자동화 도구

* 빌드를 포함하여 테스트 및 배포를 자동화하는 도구

### Jenkins

* JAVA 기반의 오픈소스



형태로, 서블릿 컨테이너에서 실행되는 서버 기반 도구

오픈소스 형태의 자동화 도구

### Gradle

* Groovy를 기반으로 한