

흥달샘과 함께하는

—

정보처리기사 실기 최종정리 특강

[1과목 - 소프트웨어 구축]

1억뷰 N잡

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.

YouTube 흥달샘 (<https://bit.ly/3KtwdLG>)

E-Mail hungjik@naver.com

네이버 카페 흥달샘의 IT 이야기 (<https://cafe.naver.com/sosozl/>)

01 소프트웨어 공학 개념

Section 1. 소프트웨어 공학

1. 소프트웨어 공학(Software Engineering)

- 소프트웨어 위기를 극복하고 효율적으로 품질 높은 소프트웨어를 개발하기 위한 학문

2. 소프트웨어 공학의 3R

(1) 역공학(Reverse Engineering)

- 기존 개발된 시스템을 CASE 도구를 이용하여 요구 분석서, 설계서 등의 문서로 추출하는 작업

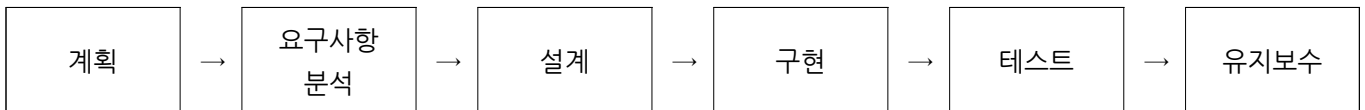
(2) 재공학(Re-Engineering)

- 기존 소프트웨어의 기능을 개선시키는 방법(예방 유지보수 측면)
- 예방 유지보수(Preventive Maintenance) 측면에서 소프트웨어 위기 해결
- 재공학 과정
 - 분석(Analysis) → 재구성(Restructuring) → 역공학(Reverse Engineering) → 이관(Migration)

(3) 재사용(Reuse)

- 재사용 방법
 - ① 합성 중심(Composition Based, 블록 구성) : 모듈을 끼워 맞춰 완성
 - ② 생성 중심(Generation Based, 패턴 구성) : 추상화 형태

3. 소프트웨어 개발 단계



Section 2. 소프트웨어 개발 방법론

1. 소프트웨어 개발 방법론 종류

(1) 구조적 방법론

- 절차지향 소프트웨어 개발 방법론

(2) 정보공학 방법론

- 기업의 주요 부분을 통합, 적용하는 데이터 중심 방법론

(3) 객체지향 개발 방법론

- 현실세계의 개체(Entity)를 속성(Attribute)과 메서드(Method)형태로 표현

(4) CBD(Component Based Development) 분석 방법론

- 컴포넌트를 조합해 애플리케이션 개발

(5) 애자일 방법론

- 변화에 빠른 대응을 하기 위해 등장한 방법론

2. 소프트웨어 개발 모델

- (1) 폭포수 모델(Waterfall Model)
 - 계획, 분석, 설계, 구현, 테스트, 운영 등 전 과정을 순차적으로 접근하는 개발모델
- (2) 프로토타이핑 모델(Prototyping Model)
 - 주요 기능을 시제품 형태로 구현하여 완성해가는 모델
- (3) 나선형 모델(Spiral Model)
 - 위험 분석을 추가한 점증적 개발 모델
 - 계획 → 위험분석 → 개발 → 평가
- (4) RAD(Rapid Application Development) 모델
 - CASE 도구를 이용해 빠른 시스템을 개발
- (5) V 모형
 - 단위 테스트 → 통합 테스트 → 시스템 테스트 → 인수 테스트
- (6) 4세대 기법(4th Generation Techniques)
 - 요구사항 명세서로부터 원시코드를 자동으로 생성할 수 있게 해주는 모델

3. 애자일(Agile) 방법론

- (1) 애자일 방법론의 개념
 - 신속한 반복 작업을 통해 실제 작동 가능한 소프트웨어를 개발하는 방법론
- (2) 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.
- (3) 애자일 방법론 종류
 - 1) XP(eXtream Programming)
 - ① XP 5가지 핵심가치
 - 용기, 존중, 의사소통, 피드백, 단순성
 - ② 12가지 실천사항

<ul style="list-style-type: none"> • 짝 프로그래밍(Pair Programming) • 계획 세우기(Planning Game) • 테스트 기반 개발(Test Driven Development) • 고객 상주(Whole Team) • 지속적인 통합(Continuous Integration) • 코드 개선(Design Improvement) 	<ul style="list-style-type: none"> • 작은 릴리즈(Small Releases) • 코딩 표준(Coding Standards) • 공동 코드 소유(Collective Code Ownership) • 간단한 디자인(Simple Design) • 시스템 메타포어(System Metaphor) • 작업시간 준수(Sustainable Pace)
---	--
 - 2) 스크럼(Scrum)
 - 개발 주기는 30일 정도로 조절하고 개발 주기마다 실제 동작할 수 있는 결과를 제공
 - 스프린트 계획 회의를 통해 스프린트 백로그를 작성한다.

3) 그 외 애자일 방법론

① 크리스털(Crystal)

- 프로젝트의 규모와 영향의 크기에 따라서 여러 종류의 방법론을 제공

② FDD(Feature-Driven Development)

- 신규 기능 단위로 하는 개발 방법론

③ ASD(Adaptive Software Development)

- 합동 애플리케이션 개발을 사용하는 방법론

④ 린(Lean)

- 도요타 린 시스템 품질기법을 소프트웨어 개발 프로세스에 적용해서 낭비 요소를 제거하여 품질을 향상시킨 방법론

4. IT 서비스 관리

(1) SLM(Service Level Management)

- 서비스의 품질을 높이는 일련의 관리 및 활동

(2) SLA(Service Level Agreement)

- 서비스 수준을 명시적으로 정의한 문서

(3) ITSM(Information Technology Service Management)

- IT 서비스를 구현, 전달 및 관리하기 위한 일련의 정책과 관행

(4) ITIL(IT Infrastructure Library)

- IT 서비스를 쉽게 제공하고 관리할 수 있는 가이드 혹은 프레임워크
- ITSM을 실현하는 도구 또는 방법

02 프로젝트 계획 및 분석

Section 1. 프로젝트 계획

1. 프로젝트 관리

- (1) 프로젝트 핵심 관리대상(3P)
 - 1) 사람(People)
 - 2) 문제(Problem)
 - 3) 프로세스(Process)
- (2) PMBOK(Project Management Body of Knowledge)
 - PMI(Project Management Institute)에서 제작한 프로젝트 관리 프로세스 및 지식 체계
 - 착수 → 계획 → 실행 → 통제 → 종료

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
- (2) 상향식 산정 기법(Bottom-Up)
 - 1) LOC(원시코드 라인 수) 기법
 - 비관치, 낙관치, 중간치를 측정 후 예측치를 구하고, 이를 이용해 비용을 산정하는 기법
 - 2) 단계별 인원수(M/M) 기법
 - 소프트웨어 개발 생명주기 각 단계별로 적용시켜 모든 단계의 비용을 산정하는 기법
- (3) 수학적 산정 기법
 - 1) COCOMO 기법
 - 조직형(Organic Mode) : 5만 라인 이하
 - 반분리형(Semidetached Mode) : 30만 라인 이하
 - 내장형(Embedded Mode) : 30만 라인 이상
 - 2) Putnam 기법
 - 시간에 따른 함수로 표현되는 Rayleigh-Norden 곡선의 노력 분포도를 기초로 한다.
 - SLIM : Rayleigh-Norden 곡선과 Putnam 예측 모형을 기초로 개발한 자동화 추정도구
 - 3) 기능 점수 기법(FP, Function Point)
 - 소프트웨어가 가지는 기능의 개수를 기준으로, 소프트웨어의 규모를 측정하는 기법
 - ESTIMACS : FP모형을 기초로 개발된 자동화 추정 도구
 - 기능 분류 : 내부 논리 파일(ILF), 외부 연계파일(EIF), 외부입력(EI), 외부출력(EO), 외부 조회(EQ)

3. 개발 일정 산정

(1) 작업 순서

- 1) 작업분해(Work Breakdown Structure)
- 2) CPM 네트워크 작성
- 3) 최소 소요 기간을 구함
- 4) 소요 M/M, 기간을 산정하여 CPM 수정
- 5) 간트 차트로 표현

(2) WBS(Work Breakdown Structure)

- 프로젝트 목표를 달성하기 위해 필요한 활동과 업무를 세분화하는 작업

(3) Network Chart(PERT/CPM)

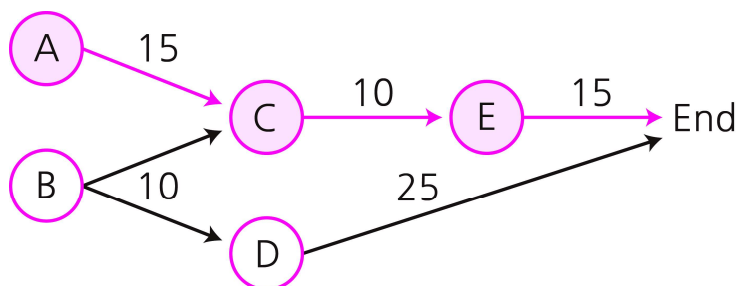
1) PERT/CPM

- 작업의 선/후행 관계를 고려하여 전체작업의 완료시간을 결정하고(PERT), 추가비용 투입을 고려하여 전체작업 완료시간을 단축하는(CPM) 네트워크 분석 기법
- 임계경로(Critical Path) : 프로젝트를 끝내기 위해 필요한 최소 소요기간

2) CPM 소작업 리스트

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

3) CPM 네트워크 작성



4) 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

Section 2. 요구사항 분석

1. 현행 시스템 분석

(1) 플랫폼 기능 분석

1) 플랫폼 기능

- 연결기능, 비용감소, 브랜드 신뢰, 커뮤니티 형성

2) 플랫폼의 유형

- 싱글 사이드, 투 사이드, 멀티 사이드

3) CPND(Contents Platform Network Device)

- 콘텐츠를 플랫폼에 맞게 가공하고 네트워크를 통해 사용자의 단말기로 서비스가 이루어짐을 표현한다.

(2) 미들웨어(Middleware)의 종류

1) 원격 프로시저 호출(RPC, Remote Procedure Call)

- 클라이언트가 원격에서 동작하는 프로시저를 호출하는 시스템

2) 메시지 지향 미들웨어(MOM, Message Oriented Middleware)

- 메시지 기반의 비동기형 메시지를 전달하는 방식의 미들웨어

3) ORB(Object Request Broker)

- 객체지향 시스템에서 객체 및 서비스를 요청하고 전송할 수 있도록 지원하는 미들웨어

4) DB 접속 미들웨어

- 애플리케이션과 데이터베이스 서버를 연결해주는 미들웨어

5) TP 모니터

- 트랜잭션이 올바르게 처리되고 있는지 데이터를 감시하고 제어

6) 웹 애플리케이션 서버(WAS, Web Application Server)

- 동적인 콘텐츠를 처리하기 위한 미들웨어

7) 엔터프라이즈 서비스 버스(ESB, Enterprise Service Bus)

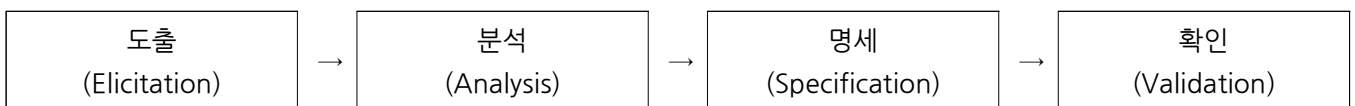
- 기업 안팎에 있는 모든 시스템 환경을 연동하는 미들웨어

2. 요구 공학

(1) 요구공학 개념

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

(2) 요구사항 개발 프로세스



(3) 요구사항 분석 도구

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

- CASE 도구의 분류

분류	설명
상위 CASE	<ul style="list-style-type: none"> - 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, 문서화하는 과정을 지원한다. - 구문 편집기, 코드 생성기 등의 기능을 수행한다.
통합 CASE	- 소프트웨어 생명주기에 포함되는 전체 과정을 지원한다.

2) HIPO(Hierarchy Input Process Output)

① HIPO의 개념

- 하향식 소프트웨어 개발을 위한 문서화 도구
- 시스템의 기능을 여러 개의 고유 모듈들로 분할하여 이들 간의 계층구조를 표현한 도표

② HIPO Chart 종류

- 가시적 도표(Visual Table of Content)
- 총체적 도표(Overview Diagram)
- 세부적 도표(Detail Diagram)

3. 요구사항 분석 모델링

(1) 모델링의 개념

- 소프트웨어를 구성하는 모듈들을 식별하고, 모듈들의 연결을 그림으로 표현





(2) 모델링 구분

- 기능 모델링
- 정적 모델링
- 동적 모델링

(3) 구조적 분석 도구

1) 자료 흐름도(DFD, Data Flow Diagram)

- 기능 중심의 시스템을 모델링하는 데 적합
- 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.	

2) 자료사전(DD, Data Dictionary)

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

기호	의미	설명
=	자료의 정의	~로 구성되어 있다
+	자료의 연결	그리고, 순차(and)



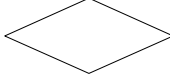
()	자료의 생략	생략 가능한 자료
[]	자료의 선택	여러 대안 중 하나 선택
{ }	자료의 반복	자료의 반복
**	자료의 설명	주석

3) 소단위 명세서(Mini-Specification)

- 자료 흐름도에서 어떤 일이 수행되는지를 정의하기 위해 각 처리들이 수행하는 업무를 상세하게 작성

4) 개체 관계도(ERD, Entity Relationship Diagram)

- 시스템에서 처리되는 구조인 개체와 속성, 개체 간의 관계를 표현하여 모델화하는 데 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	

5) 상태 전이도(STD, State Transition Diagram)

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

(4) 객체지향 분석 방법론

1) Rumbaugh(럼바우) 방법

- 가장 일반적으로 사용되는 방법으로 분석 활동을 객체 모델, 동적 모델, 기능 모델로 나누어 수행
- 분석 절차
 - 객체 모델링(Object Modeling) : 객체 다이어그램
 - 동적 모델링(Dynamic Modeling) : 상태 다이어그램
 - 기능 모델링(Functional Modeling) : 자료흐름도(DFD)

2) Booch(부치) 방법

- 미시적 개발 프로세스와 거시적 개발 프로세스를 모두 사용하는 분석 방법

3) Jacobson 방법

- Use case를 강조하여 사용하는 분석 방법

4) Coad와 Yourdon 방법

- E-R 다이어그램을 사용하여 객체의 행위를 모델링하는 기법

5) Wirfs-Brock 방법

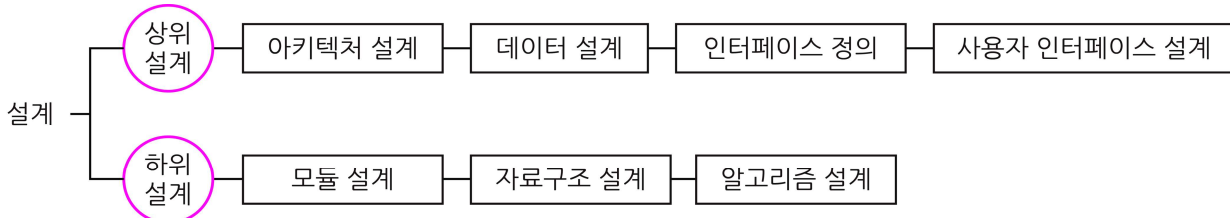
- 분석과 설계 간 구분 없음

03 소프트웨어 설계

Section 1. 소프트웨어 설계의 기본 원칙

1. 소프트웨어 설계

(1) 소프트웨어 설계의 종류



(2) 소프트웨어 설계의 원리

- 분할과 정복(Divide & Conquer)
- 추상화(Abstraction) : 과정, 데이터, 제어
- 단계적 분해(Stepwise Refinement)
- 모듈화(Modulization)
- 정보은닉(Information Hiding)

(3) 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 되어야 할 조건 - 결과 조건 : 사용 후 만족되어야 할 결과조건 - 불변 조건 : 오퍼레이션이 실행되는 동안 항상 만족되어야 할 조건

Section 2. 소프트웨어 아키텍처

1. 소프트웨어 아키텍처

(1) 소프트웨어 아키텍처의 특징

- 간략성, 추상화, 가시성, 관점모형, 의사소통 수단

(2) 소프트웨어 아키텍처 4+1 뷰

1) 소프트웨어 아키텍처 4+1 뷰 개념

- 고객의 요구사항을 정리해 놓은 시나리오를 4개의 관점에서 바라보는 소프트웨어적인 접근 방법
- 복잡한 소프트웨어 아키텍처를 다양한 이해관계자들이 바라보는 관점

2) 4+1 View Model과 구성요소

구성요소	설명
논리 뷰 (Logical View)	- 시스템의 기능적인 요구사항
구현 뷰 (Implementation View)	- 개발자 관점에서 소프트웨어 구현과 관리적인 측면을 컴포넌트 다이어그램으로 표현
프로세스 뷰 (Process View)	- 시스템의 동작을 중점적으로 표현
배치 뷰 (Deployment View)	- 물리적인 노드의 구성과 상호 연결 관계를 배치 다이어그램으로 표현
유스케이스 뷰 (Use Case View)	- 아키텍처를 도출하고 설계하는 작업을 주도하는 뷰 - +1 에 해당하며 유스케이스가 나머지 4개 뷰에 모두 참여하면서 영향을 준다.

2. 소프트웨어 아키텍처 패턴

(1) 계층화 패턴(Layered Pattern)

- 하위 모듈을 그룹으로 나눌 수 있는 구조화된 프로그램에서 사용

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

- 서버는 클라이언트에게 서비스를 제공하며 데이터를 관리하는 역할

(3) 마스터-슬레이브 패턴(Master-Slave Pattern)

- 마스터 컴포넌트가 동등한 구조의 슬레이브 컴포넌트로 작업을 분산하고, 슬레이브가 결과값을 반환하면 최종 결과값을 계산하는 구조

(4) 파이프-필터 패턴(Pipe-Filter Pattern)

- 서브시스템이 입력데이터를 받아 처리하고 결과를 다음 서브시스템으로 넘겨주는 과정을 반복

(5) 브로커 패턴(Broker Pattern)

- 분리된 컴포넌트로 구성된 분산 시스템에서 사용되는 패턴

(6) 피어 투 피어 패턴(Peer to Peer Pattern)

- 피어라 부르는 각 컴포넌트 간에 서비스를 주고받는 패턴

(7) 이벤트-버스 패턴(Event-Bus Pattern)

- 이벤트 버스를 통해 특정 채널로 메시지를 발행

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

- 3개의 각 컴포넌트는 각자의 역할을 갖고 사용자에게 서비스를 제공

(9) 블랙보드 패턴(Blackboard Pattern)

- 명확히 정의된 해결 전략이 알려지지 않은 문제에 대해서 유용한 패턴

(10) 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
- 프로그램 언어가 아닌 기호와 도식을 이용하여 표현하는 방법을 정의한다.

(2) UML 특징

- 가시화 언어, 명세화 언어, 구축 언어, 문서화 언어

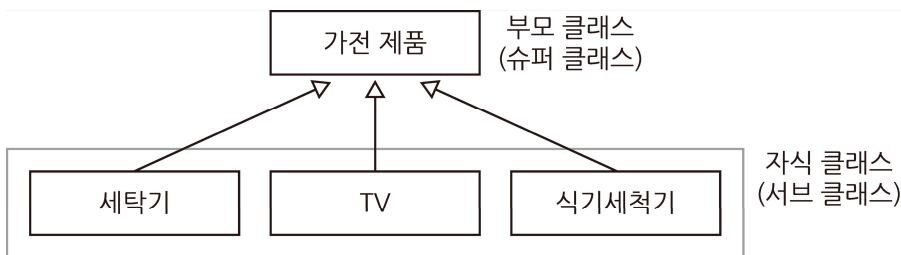
2. UML 구성요소

(1) 사물(Things)

(2) 관계(Relationships)

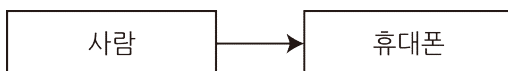
1) 일반화 관계(Generalization)

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



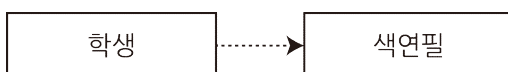
2) 연관관계(Association)

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



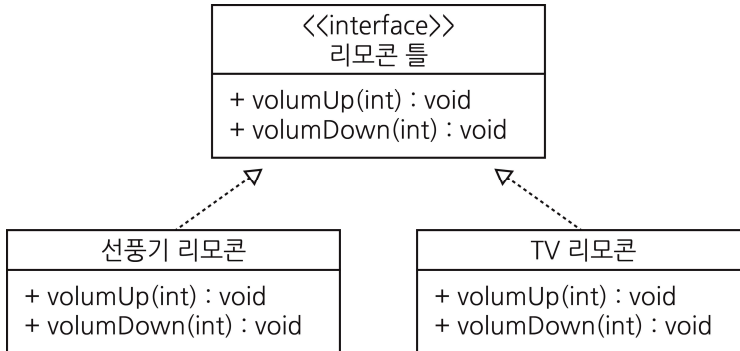
3) 의존관계(Dependency)

- 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 클래스가 다른 클래스를 오퍼레이션의 매개변수로 사용하는 경우



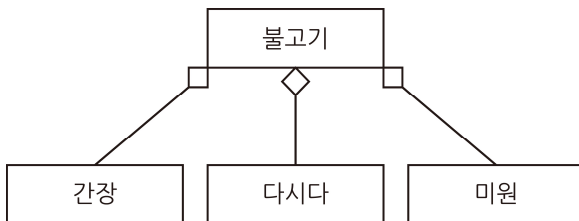
4) 실체화 관계(Realization)

- 인터페이스를 구현받아 추상 메서드를 오버라이딩하는 것을 의미



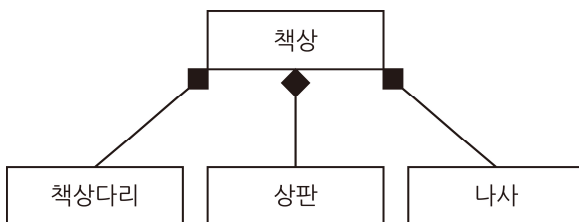
5) 집합 관계 - 집약관계(Aggregation)

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



6) 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)

1) 구조 다이어그램

종류	설명
클래스 다이어그램	- 클래스의 속성과 클래스 사이의 관계를 표현
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
복합체 다이어그램	- 클래스나 컴포넌트가 복합구조를 가질 시 그 내부 구조를 표현
패키지 다이어그램	- 유스케이스나 클래스 등 모델 요소들을 그룹화한 패키지들의 관계 표현

2) 행위 다이어그램

종류	설명
유스케이스 다이어그램	- 사용자의 요구를 분석하여 기능 모델링 작업에 사용됨
시퀀스 다이어그램	- 특정 행동이 어떠한 순서로 어떤 객체와 상호작용하는지 표현
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 제약을 명시적으로 표현

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

(2) UX(User eXperience) 개념

- 사용자가 컴퓨터, 웹 사이트, 시스템 등 정보기기의 UI를 직/간접적으로 이용하여 경험한 모든 것

(3) UI 유형

1) CLI(Command Line Interface)

- 사용자가 컴퓨터 자판을 이용해 명령을 입력하여 컴퓨터를 조작하는 시스템

2) GUI(Graphical User Interface)

- 그래픽과 텍스트로 이루어져 있어, 사용자의 입력이나 출력이 마우스 등을 통해 이루어짐

3) NUI(Natural User Interface)

- 특별한 하드웨어 없이 인간의 자연스러운 움직임을 인식하여 정보를 제공

4) OUI(Organic User Interface)

- 현실의 모든 것이 입출력 장치가 됨

5) AUI(Auditory User Interface)

- 보다 나은 사용자 경험을 제공하기 위해 만들어진 임베디드 사운드

2. UI 설계

(1) UI 요구사항 구분

- 기능적 요구사항
- 비기능적 요구사항

(2) 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 수용해야 한다.

(3) UI 설계 도구

1) 와이어프레임(Wireframe)

- 선(Wire)으로 틀(Frame)을 잡는다는 뜻

2) 스토리보드

- 개발 후 완성된 콘텐츠의 최종 결과를 예상할 수 있는 기초 문서

3) 프로토타입

- 실제 서비스와 흡사한 모형을 만드는 작업

4) 목업(Mockup)

- 와이어프레임보다 좀 더 실제 화면과 유사하게 만든 정적인 형태의 모형

5) 유스케이스

- 사용자 측면의 요구사항

3. 감성공학

(1) 감성공학의 개념

- 인간의 심상을 구체적인 물리적 설계 요소로 번역하여 이를 실현하는 기술
- 요소화 → 형상화 → 구현 → 생산

(2) 제품과 관련된 인간의 감성

- 감각적 감성, 기능적 감성, 문화적 감성

(3) 감성공학의 접근 방법

1) 1류 접근 방법

- 인간의 어휘를 이용하여 제품에 대한 이미지를 조사하고 제품 디자인 요소와 연계시킴

2) 2류 접근 방법

- 문화적 감성의 일부를 반영한 개념

3) 3류 접근 방법

- 특정 시제품을 사용하여 감각 척도를 계측하고, 정량화된 값을 환산

Section 2. UI 구현

1. 화면 레이아웃 구성

(1) 레이아웃(Layout)의 개념

- 특정 공간에 여러 구성 요소를 보기 좋게 효과적으로 배치하는 작업

(2) HTML5

- 웹 페이지의 기본 구조를 담당

(3) CSS(Cascading Style Sheet)

- 색상이나 크기, 이미지 크기나 위치, 배치 방법 등 웹 문서의 디자인 요소를 담당

(4) JavaScript

- 웹 문서에 삽입해서 사용하는 스크립트 언어

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 애플리케이션이 설치된 하드웨어
- Apache Web Server, IIS, Nginx, GWS 등

(2) 웹 애플리케이션 서버(WAS)

- 동적인 웹 서비스를 제공하기 위한 미들웨어가 설치된 하드웨어
- WebLogic, WebSpere, 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 : Random, Least loaded, Round Robin

(6) CDN(Content Delivery Network)

- 용량이 큰 콘텐츠를 빠른 속도로 제공하기 위해 사용자와 가까운 곳에 분산되어 있는 데이터 저장 서버

2. 개발 소프트웨어 환경

(1) 시스템 소프트웨어

- 운영체제(OS, Operating System)
- JVM(Java Virtual Machine)
- Web Server
- WAS(Web Application Server)
- DBMS(DataBase Management System)

(2) 개발 소프트웨어

- 요구사항 관리 도구
- 설계/모델링 도구
- 구현도구
- 테스트 도구
- 형상관리 도구

3. IDE(Integrated Development Environment) 도구

- 소프트웨어 개발에 필요한 많은 도구의 기능을 하나로 묶어 활용하는 소프트웨어

4. 협업 도구

- 여러 사용자가 각기 별개의 작업 환경에서 통합된 하나의 프로젝트를 동시에 수행할 수 있도록 도와주는 소프트웨어

5. 형상 관리 도구

(1) 형상 관리 도구의 개념

- 소프트웨어 생명주기 동안 발생하는 변경사항을 통제하기 위한 관리 방법

(2) 변경 관리/버전 관리/형상 관리

1) 변경 관리

- 소스의 변경 사항을 관리

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

(3) 형상 관리 절차

1) 형상 식별

- 형상 관리의 시작으로 시스템을 구성하는 요소들 중 형상 관리의 대상들을 구분하고 관리 목록의 번호를 정의하여 부여한다.

2) 형상 통제

- 소프트웨어 형상 변경 요청을 검토하고 승인하여 현재의 베이스라인에 반영될 수 있도록 통제
- 형상통제가 이루어지기 위해서는 형상 통제 위원회(CCB, Configuration Control Board)의 승인을 통한 변경 통제가 이루어져야 한다.

3) 형상 감사

- 형상 항목의 변경이 계획에 따라 제대로 이뤄졌는지를 검토하고 승인

4) 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 유형

1) 공유 폴더 방식(RCS, SCCS)

- 개발이 완료된 파일을 매일 약속된 위치의 공유 폴더에 복사

2) 클라이언트/서버 방식(CVS, SVN)

- 중앙에 버전 관리 시스템이 항상 동작

3) 분산 저장소 방식(Git)

- 로컬 저장소와 원격 저장소 구조

(2) 버전 관리 도구별 특징

1) CVS

- 오랜 기간 사용된 형상 관리 도구로, 다양한 운영체제를 지원

2) SVN

- CVS의 단점을 보완하기 위해 만들어졌다.
- 최초 1회에 한해 파일 원본을 저장하고, 그 이후에는 실제 파일이 아닌 원본과 차이점을 저장하는 방식

3) Git

- 리눅스 토발즈가 리눅스 커널의 개발을 위해 만들었다.
- 원격 Repository에 장애가 있어도 버전 관리가 가능하다.

4) Clear Case

- IBM에서 개발된 유료 버전의 형상 관리 툴
- 서버가 부족할 때 서버를 하나씩 늘려 확장할 수 있다.

5) BitKeeper

- SVN과 비슷한 중앙 통제 방식으로 대규모 프로젝트에서 빠른 속도를 내도록 개발된 버전관리 도구

6) RCS(Revision Control System)

- 소스 파일의 수정을 한 사람만으로 제한하여 다수의 사람이 파일의 수정을 동시에 할 수 없도록 파일을 잠금 처리하는 방식으로 버전 컨트롤을 수행

(3) 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

7. 빌드 도구

(1) 빌드의 개념

- 소스코드 파일들을 컴퓨터에서 실행할 수 있는 소프트웨어로 변환하는 일련의 과정

(2) 빌드 자동화 도구 종류

1) Make

- 유닉스 계열 운영체제에서 주로 사용되는 프로그램 빌드 도구이다.

2) Ant

- Java 기반의 빌드 도구로 다른 빌드 도구보다 역사가 오래되었다.

3) Maven

- 프로젝트에 필요한 모든 의존성(Dependency)을 리스트 형태로 Maven에게 알려 관리할 수 있도록 돕는 방식이다.

4) Jenkins

- Java 기반의 오픈소스로, 소프트웨어 개발 시 지속적 통합(Continuous Integration) 서비스를 제공하는 툴

5) Gradle

- Groovy를 기반으로 한 오픈 소스 형태의 자동화 도구로 안드로이드 앱 개발 환경에서 사용

Section 2. 개발 프레임워크

1. 프레임워크의 개념

- 소프트웨어 개발을 위해 전체적인 틀을 제공하는 반제품 형태의 소프트웨어

2. 프레임워크의 특징

- 모듈화
- 재사용성
- 확장성
- 제어의 역흐름

3. 프레임워크의 구분

구분	종류
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)

- 프로그램 언어가 사용되는 모듈을 모아둔 것

5. API(Application Programming Interface)

- 소프트웨어에 서비스를 제공하는 프로그램

Section 3. 모듈 구현

1. 단위 모듈 구현

- (1) 효과적인 모듈화
 - 결합도를 낮추고 응집도를 높여 모듈의 독립성을 높임
 - FAN-OUT 최소화, FAN-IN 증가
- (2) 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
- (3) 단위 모듈 작성 원칙
 - 정확성, 명확성, 완전성, 일관성, 추적성

2. 결합도

- (1) 결합도(Coupling)의 개념
 - 두 모듈 사이의 연관 관계
 - 결합도가 낮을수록 잘 설계된 모듈이다.
- (2) 결합도 유형
 - 자료 결합도(Data Coupling)
 - 스탬프 결합도(Stamp Coupling)
 - 제어 결합도(Control Coupling)
 - 외부 결합도(External Coupling)
 - 공통 결합도(Common Coupling)
 - 내용 결합도(Content Coupling)

3. 응집도

- (1) 응집도(Cohesion)의 개념
 - 모듈의 독립성을 나타내는 개념으로, 모듈 내부 구성요소 간 연관 정도
 - 응집도는 높을수록 좋고, 결합도는 낮을수록 이상적
- (2) 응집도 유형
 - 기능적 응집도(Functional Cohesion)
 - 순차적 응집도(Sequential Cohesion)
 - 통신적 응집도(Communication Cohesion)
 - 절차적 응집도(Procedural Cohesion)
 - 시간적 응집도(Temporal Cohesion)
 - 논리적 응집도(Logical Cohesion)
 - 우연적 응집도(Coincidental Cohesion)

4. 팬인(Fan-in), 팬아웃(Fan-out)

- 소프트웨어의 구성요소인 모듈을 계층적으로 분석하기 위해 활용
- 팬인(Fan-in) : 해당 모듈로 들어오는 상위 모듈 수
- 팬아웃(Fan-out) : 해당 모듈에서 호출하는 하위 모듈 수

5. 공통 모듈 구현

(1) 공통 모듈 구현요소

- DTO : 프로세스 사이에서 데이터를 전송하는 객체
- VO : 도메인에서 속성들을 묶어서 특정 값을 나타내는 객체
- DAO : 실질적으로 DB에 접근하는 객체
- Service : DAO 클래스를 호출하는 객체
- Controller : 비즈니스 로직을 수행하는 객체

(2) Annotation

- 자바코드에 주석처럼 달아 특수한 의미를 부여한다.

Section 4. 서버 프로그램 구현

1. 서버 프로그램 구현

(1) MVC 모델의 계층

- 1) 프레젠테이션 계층(Presentation Layer)
 - 사용자 인터페이스
- 2) 제어 계층(Control Layer)
 - 어떤 요청이 들어왔을 때 어떤 로직이 처리해야 하는지를 결정한다.
- 3) 비즈니스 로직 계층(Business Logic Layer)
 - 핵심 업무를 어떻게 처리하는지에 대한 방법을 구현하는 계층
- 4) 퍼시스턴스 계층(Persistence Layer)
 - 데이터 처리를 담당하는 계층
- 5) 도메인 모델 계층(Domain Model Layer)
 - 각 계층 사이에 전달되는 실질적인 비즈니스 객체

2. DBMS 접속기술

(1) DBMS 접속기술 종류

- 소켓통신
- Vender API
- JDBC(Java DataBase Connectivity)
- ODBC(Open DataBase Connectivity)

3. ORM(Object-Relational Mapping) 프레임워크

(1) ORM 프레임워크의 개념

- 객체와 관계형 데이터베이스의 데이터를 자동으로 매핑(연결)해 주는 것

(2) 매핑 기술 비교

1) SQL Mapper

- SQL을 명시하여 단순히 필드를 매핑시키는 것이 목적

2) OR Mapping(=ORM)

- 객체를 통해 간접적으로 데이터베이스를 다룬다.

4. 시큐어 코딩(Secure Coding)

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

- 오픈소스 웹 애플리케이션 보안 프로젝트
- OWASP Top 10
 - 웹 애플리케이션 취약점 중 빈도가 많이 발생하고, 보안상 영향을 줄 수 있는 10가지를 선정하여 발표

(2) 시큐어 코딩 가이드

1) 입력 데이터 검증 및 표현

- 프로그램 입력값에 대한 검증 누락 또는 부적절한 검증, 데이터 형식을 잘못 지정하여 발생하는 보안 약점
- 보안 약점 종류
 - SQL Injection
 - XSS(크로스 사이트 스크립트)
 - 자원 삽입
 - 위험한 형식 파일 업로드
 - 명령 삽입
 - 메모리 버퍼 오버프로

2) 보안 기능

- 보안 기능을 부적절하게 구현하는 경우 발생할 수 있는 보안 약점
- 보안 약점 종류
 - 적절한 인증 없이 중요기능 허용
 - 부적절한 인가
 - 취약한 암호화 알고리즘 사용
 - 하드코딩된 패스워드
 - 패스워드 평문 저장
 - 취약한 패스워드 허용

3) 시간 및 상태

- 하나 이상의 프로세스가 동작하는 환경에서 시간 및 상태를 부적절하게 관리하여 발생할 수 있는 보안 약점
- 보안 약점 종류
 - 경쟁 조건
 - 종료되지 않는 반복문 또는 재귀 함수

4) 에러 처리

- 에러를 처리하지 않거나 불충분하게 처리하여 에러 정보에 중요 정보가 포함될 때 발생할 수 있는 보안 약점
- 보안 약점 종류
 - 오류 메시지 정보 노출
 - 오류 상황 대응 부재
 - 부적절한 예외 처리

5) 코드 오류

- 개발자가 범할 수 있는 코딩 오류로 인해 유발되는 보안 약점
- 보안 약점 종류
 - 널 포인터 역참조
 - 부적절한 자원 해제
 - 해제된 자원 사용
 - 초기화되지 않은 변수 사용

6) 캡슐화

- 중요한 데이터 또는 기능을 불충분하게 캡슐화하거나 잘못 사용해 발생하는 보안 약점
- 보안 약점 종류
 - 잘못된 세션에 의한 정보 노출
 - 제거되지 않은 디버그 코드
 - 시스템 정보 노출
 - 잘못된 접근 지정자

7) API 오용

- 의도된 사용에 반하는 방법으로 API를 사용하거나 보안에 취약한 API를 사용하여 발생할 수 있는 보안 약점
- 보안 약점 종류
 - DNS에 의존한 보안 결정
 - 취약한 API 사용

Section 5. 배치 프로그램 구현

1. 배치 프로그램

(1) 배치의 개념

- 데이터를 일괄적으로 모아서 처리하는 대량의 작업을 처리

(2) 배치 프로그램의 필수 요소

- 대용량 데이터
- 자동화
- 견고함
- 안정성
- 성능

(3) 스케줄 관리 종류

- 크론탭(Crontab)
- Spring Batch
- Quartz Job Scheduler

06 인터페이스 구현

Section 1. 인터페이스 개요

1. 인터페이스 시스템

- (1) 인터페이스 시스템의 개념
 - 서로 다른 시스템, 장치 사이에서 정보나 신호를 주고받을 수 있도록 도움을 주는 시스템
- (2) 인터페이스 시스템 구성
 - 송신 시스템
 - 수신 시스템
 - 중계 서버

2. 연계 시스템 분류와 데이터 식별

- (1) 송수신 데이터 식별
 - 송수신 전문 구성

구성	설명
전문 공통부	인터페이스 표준 항목을 포함
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
- (2) 인터페이스 정의서
 - 데이터 송신 시스템과 수신 시스템 간의 속성과 제약조건 등을 상세히 포함한다.

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
 - 2) EAI의 구축유형
 - Point-to-Point
 - Hub & Spoke
 - Message Bus(ESB 방식)
 - Hybrid

(2) ESB(Enterprise Service Bus)

- 다양한 시스템과 연동하기 위한 멀티 프로토콜 지원
- 버스를 통해 이기종 애플리케이션을 유연(loosely-coupled)하게 통합하는 핵심 플랫폼

2. 인터페이스 연계 기술

(1) Link

- 데이터베이스에 제공하는 DB Link 객체를 이용

(2) DB Connection

- DB Connection Pool을 생성하고 연계 프로그램에서 해당 DB Connection Pool명을 이용

(3) JDBC

- 수신 시스템의 프로그램에서 JDBC 드라이버를 이용하여 송신 시스템 DB와 연결

(4) API / OpenAPI

- 송신 시스템의 애플리케이션 프로그래밍 인터페이스 프로그램

(5) Web Service

- WSDL(Web Services Description Language), UDDI(Universal Description, Discovery and Integration), SOAP(Simple Object Access Protocol) 프로토콜을 이용하여 연계

(6) Hyper Link

- 웹 애플리케이션에서 하이퍼링크(Hyper Link) 이용

(7) Socket

- 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

(2) XML(eXtensible Markup Language)

- 웹에서 구조화한 문서를 표현하고 전송하도록 설계한 마크업 언어

(3) YAML(YAML Ain't Markup Language)

- 구성 파일 작성에 자주 사용되는 데이터 직렬화 언어

(4) CSV(Comma Separated Values)

- 몇 가지 필드를 쉼표(,)로 구분한 텍스트 데이터 및 텍스트 파일

4. 인터페이스 구현

(1) AJAX(Asynchronous JavaScript and XML)

1) AJAX의 개념

- 전체 페이지를 새로고침하지 않고, 페이지의 일부만을 변경할 수 있는 기법

2) 비동기 방식

- 웹페이지를 리로드하지 않고 데이터를 불러오는 방식

(2) SOAP(Simple Object Access Protocol)

1) SOAP의 구성

① SOAP(Simple Object Access Protocol)

- HTTP, HTTPS, SMTP 등을 사용하여 XML 기반의 메시지를 네트워크상에서 교환하는 형태의 프로토콜

② UDDI(Universal Description, Discovery and Integration)

- 인터넷에서 전 세계의 비즈니스 업체 목록에 자신의 목록을 등록하기 위한, XML기반의 규격

③ WSDL(Web Services Description Language)

- 웹 서비스 기술언어 또는 기술된 정의 파일의 총칭으로 XML로 기술

2) SOAP 보안 프로토콜

- SAML(인증/권한관리)
- XKMS(키관리)
- XACML(접근제어)

(3) REST

1) REST의 개념

- HTTP URI를 통해 자원을 명시하고, HTTP Method(POST, GET, PUT, DELETE)를 통해 해당 자원에 대한 CRUD Operation을 적용하는 것

2) REST 구성요소

① 자원(Resource), URI

- 서버에 존재하는 데이터의 총칭.

② 행위(Verb), Method

- 클라이언트는 URI를 이용해 자원을 지정하고 자원을 조작하기 위해 Method를 사용한다.

③ 표현(Representation)

- REST에서 하나의 자원은 JSON, XML, TEXT, RSS 등 여러 형태로 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) RESTful

- REST의 원리를 따르는 시스템

5. 인터페이스 보안

(1) 인터페이스 보안 기능 적용

1) 네트워크 영역

- IPSec, SSL, S-HTTP 등 다양한 방식으로 적용

2) 애플리케이션 영역

- 시큐어코딩 가이드를 참조하여 애플리케이션 코드상 보안 취약점을 보완하는 방향으로 보안 기능 적용

3) DB 영역

- DB, 스키마, 엔티티의 접근 권한 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

1) xUnit

- 다양한 언어를 지원하는 단위 테스트 프레임 워크

2) STAF

- 서비스 호출 및 컴포넌트 재사용 등 다양한 환경을 지원하는 테스트 프레임워크

3) FitNesse

- 웹 기반 테스트케이스 설계, 실행, 결과 확인 등을 지원하는 테스트 프레임워크

4) NTAF

- FitNesse의 장점과 STAF의 장점을 통합한 Naver의 테스트 자동화 프레임워크

5) Selenium

- 다양한 브라우저 및 개발 언어를 지원하는 웹 애플리케이션 테스트 프레임워크

6) watir

- Ruby를 사용하는 애플리케이션 테스트 프레임워크

(2) 인터페이스 구현 감시 도구

- APM(Application Performance Management)을 사용하여 동작상태 감시
- 종류 : 스카우터(Scouter), 제니퍼(Jennifer) 등

07 객체지향 구현

Section 1. 객체지향 설계

1. 객체지향(OOP, Object Oriented Programming)

(1) 객체지향 개념

- 현실 세계의 유형, 무형의 모든 대상을 객체(Object)로 나누고, 객체의 행동(Method)과 고유한 값(Attribute)을 정의하여 설계하는 방법

(2) 객체지향 구성요소

- 클래스(Class)
- 객체(Object)
- 속성(Attribute)
- 메서드(Method)
- 메시지(Message)

(3) 객체지향언어의 특징

1) 캡슐화(Encapsulation)

- 데이터(Attribute)와 데이터를 처리하는 행동(Method)을 하나로 묶은 것

2) 정보은닉(Information Hiding)

- 다른 객체에게 자신의 데이터를 숨기고, 자신이 정의한 행동만을 통하여 접근을 허용

3) 상속(Inheritance)

- 상위클래스(부모클래스)의 모든 데이터와 행동을 하위 클래스가 물려받는 것

4) 다형성(Polymorphism)

- 하나의 메시지에 대해 각 객체가 가지고 있는 여러 가지 방법으로 응답할 수 있는 개념

5) 추상화(Abstraction)

- 어떤 실체로부터 공통적인 부분들만 모아놓은 것

(4) 객체지향 설계원칙(SOLID)

1) 단일 책임 원칙(SRP, Single Responsibility Principle)

- 한 클래스는 하나의 책임만을 가져야 한다.

2) 개방 폐쇄 원칙(OCP, Open-Closed Principle)

- 확장에는 열려 있고, 수정에는 닫혀 있어야 한다.

3) 리스코프 치환 원칙(LSP, Liskov Substitution Principle)

- 자식 클래스는 언제나 자신의 부모 클래스를 대체할 수 있어야 한다.

4) 인터페이스 분리 원칙(ISP, Interface Segregation Principle)

- 자신이 사용하지 않는 인터페이스는 구현하지 말아야 한다.

5) 의존성 역전 원칙(DIP, Dependency Inversion Principle)

- 의존 관계를 맺을 때 자주 변화하는 것보다 변화가 거의 없는 것에 의존해야 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

(2) GoF 디자인 패턴

- Gof 디자인 패턴 분류

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

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

08 애플리케이션 테스트 관리

Section 1. 애플리케이션 테스트케이스 설계

1. 소프트웨어 테스트

(1) 소프트웨어 테스트의 개념

- 구현된 소프트웨어의 동작과 성능, 사용성, 안정성 등을 만족하기 위하여 소프트웨어의 결함을 찾아내는 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)
 - 파레토 법칙 : 전체 결과의 80%가 전체 원인의 20%에서 일어나는 현상
- 살충제 패러독스(Pesticide Paradox)
- 테스트는 정황(Context)에 의존한다.
- 오류-부재의 궤변(Absence of Errors Fallacy)

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

1) 참 오라클(True)

- 모든 테스트 케이스 입력값의 기대 결과를 확인

2) 샘플링 오라클(Sampling)

- 특정한 몇 개의 입력 값에 대해서만 기대하는 결과를 제공해 주는 오라클

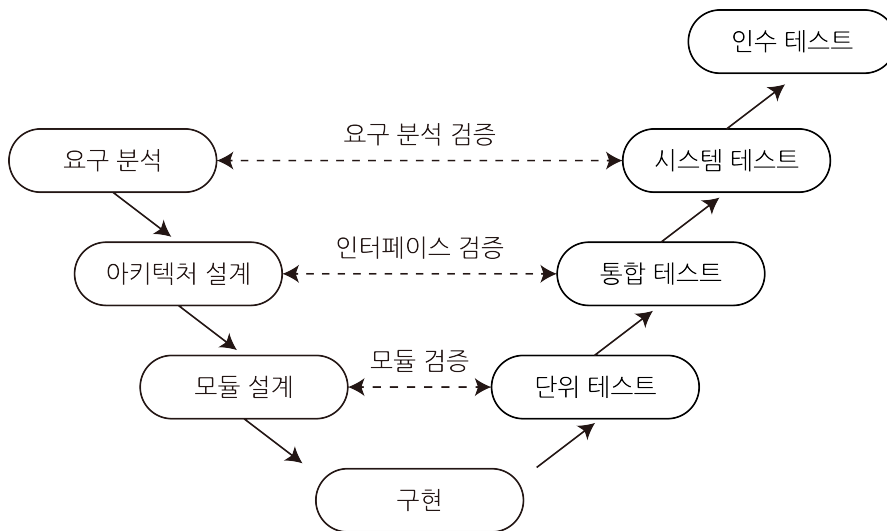
3) 휴리스틱 오라클(Heuristic)

- 특정 입력 값에 대해 올바른 결과를 제공하고, 나머지 값들에 대해서는 휴리스틱(추정)으로 처리하는 오라클

4) 일관성 검사 오라클(Consistent)

- 애플리케이션 변경이 있을 때, 수행 전과 후의 결과 값이 동일한지 확인하는 오라클

3. 테스트 레벨



4. 소프트웨어 테스트 기법

(1) 프로그램 실행 여부

- 정적 테스트
- 동적 테스트

(2) 테스트 기법

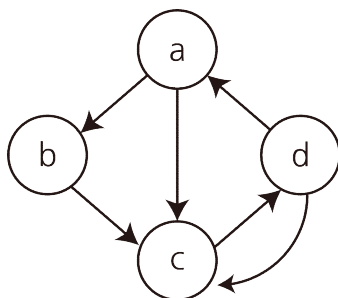
1) 화이트박스 테스트

- 소프트웨어의 내부 구조와 동작을 검사하는 테스트 방식

기법	설명
문장 검증	프로그램의 모든 문장을 한 번 수행하여 검증
선택(분기) 검증	선택하는 부분만 검증
경로 검증	수행 가능한 모든 경로 검사
조건 검증	조건이나 반복문 내 조건식을 검사

2) 기초 경로 검사(Basic Path Test)

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



3) 블랙박스 테스트

- 프로그램의 요구사항 명세를 보면서 테스트, 주로 구현된 기능을 테스트
- 블랙박스 테스트 기법

기법	설명
동등 분할 기법 (Equivalence Partitioning Testing)	- 입력 자료에 초점을 맞춰 테스트 케이스를 만들어 검사하는 방법
경계값 분석 (Boundary Value Analysis)	- 입력 값의 중간값보다 경계값에서 오류가 발생할 확률이 높다는 점을 이용해 입력 조건의 경계값을 테스트 케이스로 선정한다. - 80~90의 입력 조건이 있을 때, 79,80,81,89,90,91을 테스트 값으로 선정한다.
원인-효과 그래프 검사 (Cause-Effect Graphing Testing)	- 입력 데이터 간의 관계와 출력에 영향을 미치는 상황을 체계적으로 분석한 다음 효용성이 높은 테스트 케이스를 선정하여 검사하는 기법
오류 예측 검사 (Error Guessing)	- 과거의 경험이나 테스터의 감각으로 테스트하는 기법
비교 검사 (Comparison Testing)	- 여러 버전의 프로그램에 동일한 테스트 자료를 제공하여 동일한 결과가 출력되는지 테스트하는 기법이다.

(3) 테스트에 대한 시각

1) 검증(Verification)

- 소프트웨어의 개발 과정을 테스트

2) 확인(Validation)

- 완성된 소프트웨어의 결과를 테스트

(4) 테스트 목적

목적	설명
회복(Recovery)	- 시스템에 고의로 실패를 유도하고 시스템이 정상적으로 복구하는지 테스트
안전(Security)	- 불법적인 소프트웨어가 접근하여 시스템을 파괴하지 못하도록 소스코드 내의 보안적인 결함을 미리 점검하는 테스트
강도(Stress)	- 시스템에 과다 정보량을 부과하여 과부하 시에도 시스템이 정상적으로 작동되는지를 검증하는 테스트
성능(Performance)	- 시스템의 응답하는 시간, 처리량, 반응속도 등을 테스트
구조(Structure)	- 시스템의 내부 논리 경로, 소스코드의 복잡도를 평가하는 테스트
회귀(Regression)	- 변경 또는 수정된 코드에 대하여 새로운 결함 발견 여부를 평가하는 테스트
병행(Parallel)	- 변경된 시스템과 기존 시스템에 동일한 데이터를 입력 후 결과를 비교하는 테스트
A/B 테스트	- 기존 서비스 A와 새로 적용하고 싶은 서비스 B를 통계적인 방법으로 비교하여 새로운 서비스가 기존 서비스에 비해 정말 효과가 있는지 알아보는 테스트
스모크 테스트 (Smoke)	- 본격적인 테스트 수행에 앞서 테스트 환경의 테스트가 가능한지 여부를 판단하기 위해 간단하게 테스트

(5) 테스트 종류

1) 명세 기반 테스트

- 주어진 명세를 빠짐없이 테스트 케이스로 구현하고 있는지 확인하는 테스트

2) 구조 기반 테스트

- 소프트웨어 내부 논리 흐름에 따라 테스트 케이스를 작성하고 확인하는 테스트

3) 경험 기반 테스트

- 경험이 많은 테스터의 직관과 기술 능력을 기반으로 수행하는 테스트

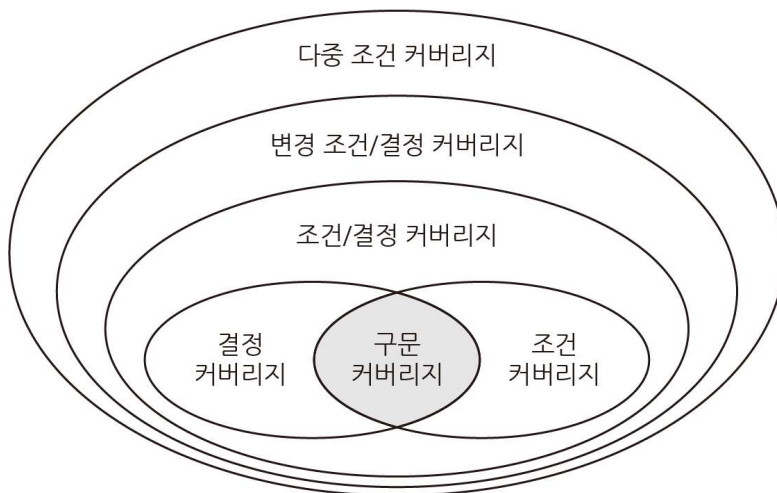
5. 테스트 커버리지

(1) 테스트 커버리지의 개념

- 주어진 테스트 케이스에 의해 수행되는 소프트웨어의 테스트 범위를 측정하는 테스트 품질 측정 기준
- 테스트를 얼마나 수행했는지 측정하는 기준

(2) 테스트 커버리지 유형

- 기능 기반 커버리지
- 라인 커버리지(Line Coverage)
- 코드 커버리지(Code Coverage)

**Section 2. 애플리케이션 통합 테스트****1. 결함관리 도구**

(1) 결함관리 프로세스

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

(2) 결함 추이 분석

- 결함 분포
- 결함 추세
- 결함 에이징

2. 테스트 자동화 도구

(1) 테스트 자동화 도구 유형

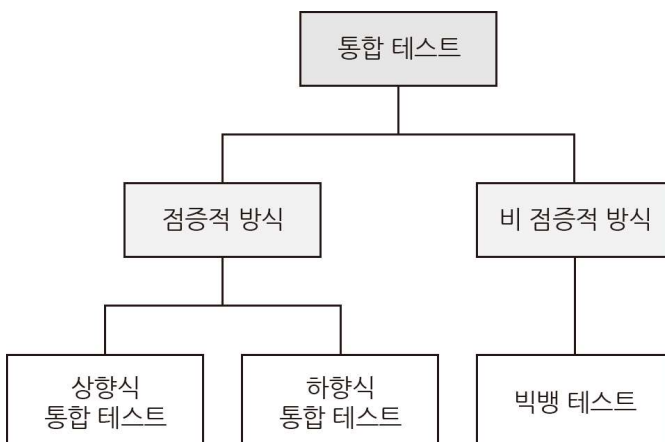
- 정적 분석 도구(Static Analysis Tools)
- 테스트 실행 도구(Test Execution Tools)
- 성능 테스트 도구(Performance Test Tools)
- 테스트 통제 도구(Test Control Tools)
- 테스트 장치(Test Harness)

(2) 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.
목 오브젝트 (Mock Object)	- 사용자의 행위를 조건부로 사전에 입력해 두면, 그 상황에 예정된 행위를 수행하는 객체를 말한다.

3. 통합 테스트

(1) 통합 테스트 수행 방법의 분류



Section 3. 애플리케이션 성능 개선

1. 애플리케이션 성능 저하 원인

- 데이터베이스 관련 성능 저하
- 내부 로직으로 인한 성능 저하 원인
- 외부 호출로 인한 성능 저하

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
- 경과 시간(Turnaround Time) : 애플리케이션에 요청을 전달한 시간부터 처리가 완료될 때까지 걸린 시간
- 자원 사용률(Resource Usage) : 작업을 처리하는 동안의 CPU, 메모리 네트워크 사용량 등의 자원 사용률

(2) 성능 분석 도구

1) JMeter

- HTTP, FTP 등 다양한 프로토콜을 지원하는 부하 테스트 도구

2) LoadUI

- 웹 서비스의 로드 테스트

3) OpenSTA

- HTTP, HTTPS 프로토콜에 대한 부하 테스트 및 생산품 모니터링 도구

(3) 모니터링 도구

1) Scouter

- 단일 뷰 통합/실시간 모니터링

2) NMon

- 리눅스 서버 자원에 대한 모니터링 도구

3) Zabbix

- 웹기반 서버, 서비스, 애플리케이션 모니터링 도구

4) Jeniffer

- 애플리케이션에서 서버로 유입되는 트랜잭션 수량, 처리시간, 응답시간, 자원 활용률 등을 모니터링

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

(1) FTR의 개념

- 소프트웨어 엔지니어가 수행하는 소프트웨어 품질보증 활동

(2) 검토지침

- 제작자가 아닌 제품의 검토에만 집중한다.
- 문제 영역을 명확히 표현한다.
- 제기된 모든 문제를 바로 해결하고자 하지 않는다.
- 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 의미 있는 교육을 행한다.
- 검토의 과정과 결과를 재검토한다.

4. 소스코드 품질 분석

(1) 동료 검토(Peer Review)

- 2~3명이 진행하는 리뷰의 형태

(2) 워크스루(Walkthrough)

- 계획된 개발자 검토 회의

(3) 인스펙션(Inspection)

- 공식적 검사 회의
- 계획 → 사전교육 → 준비 → 인스펙션 회의 → 수정 → 후속조치

5. 소스코드 품질 분석 도구

- 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 소스코드 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 소스코드 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 스멜(Code Smell)

- 컴퓨터 프로그래밍 코드에서 더 심각한 문제를 일으킬 가능성이 있는 프로그램 소스코드
- 코드 스멜 관련 용어
 - 스파게티 코드(Spaghetti Code), 외계인 코드

(2) 리팩토링

- 외부 동작을 바꾸지 않으면서 내부 구조를 개선하는 방법

(3) 클린코드

- 의존성을 최소로 하고 사람이 이해할 수 있는 가독성, 목적성이 뛰어난 명확한 코드
- 클린코드 작성 원칙
 - 가독성, 단순성, 의존성 배제, 중복성 최소화, 추상화

09 소프트웨어 유지보수

Section 1. 소프트웨어 유지보수

1. 유지보수의 구분

- (1) 수정 보수(Corrective Maintenance)
 - 소프트웨어 구축 시 테스트 단계에 미쳐 발견하지 못한 잠재적인 오류를 찾아 수정한다.
- (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
- (4) 예방 보수(Preventive Maintenance)
 - 장래에 유지보수성 또는 신뢰성을 보장하기 위해 선제적으로 하는 유지보수

2. 유지보수 관련 용어

- (1) 레거시 시스템(Legacy System)
 - 낡은 기술이나 방법론, 컴퓨터 시스템, 소프트웨어 등을 말한다.
- (2) 외계인 코드(Alien Code)
 - 아주 오래되거나 참고 문서 또는 개발자가 없어 유지보수 작업이 어려운 프로그램 코드
- (3) 스파게티 코드(Spaghetti Code)
 - 복잡한 프로그래밍 소스코드

10 제품 소프트웨어 패키징

Section 1. 국제 표준 제품 품질 특성

1. 제품 품질 국제 표준

(1) ISO/IEC 9126의 소프트웨어 품질 특성

품질 특성	설명
기능성 (Functionality)	- 요구사항에 만족하는 기능에 대한 제품의 능력 - 부특성 : 적합성, 정확성, 상호 운용성, 보안성, 준수성
신뢰성 (Reliability)	- 소프트웨어가 실행될 때, 성능 수준을 유지할 수 있는 제품의 능력 - 부특성 : 성숙성, 결함 허용성, 복구성
사용성 (Usability)	- 사용자가 쉽게 이해하고, 학습하고, 사용할 수 있는 제품의 능력 - 부특성 : 이해성, 학습성, 운영성, 선호도, 준수성
효율성 (Efficiency)	- 소프트웨어가 실행될 때, 효율적으로 자원(CPU, Memory)을 사용하는 제품의 능력 - 부특성 : 시간 반응성, 자원 활용성, 준수성
유지보수성 (Maintainability)	- 사용자의 추가 요구사항을 반영하여 제품을 변경하는 능력 - 부특성 : 분석성, 변경성, 안정성, 시험성, 준수성
이식성 (Portability)	- 현재 환경에서 다른 환경으로 이전될 수 있는 제품의 능력 - 부특성 : 적응성, 설치성, 공존성, 대체성, 준수성

(2) ISO/IEC 14598 평가 특성

평가 특성	설명
반복성 (Repetability)	- 동일 평가자, 동일 사양, 동일한 결과
재현성 (Reproducibility)	- 다른 평가자, 동일 사양, 비슷한 결과
공정성 (Impartiality)	- 평가가 특정 결과에 편향되지 않아야 한다.
객관성 (Objectivity)	- 평가 결과가 평가자의 감정이나 의견에 의해 영향을 받지 않아야 한다.

(3) ISO/IEC 12119 구성요소

구성요소	설명
제품설명서	- 소프트웨어 패키지의 속성을 설명하는 문서
사용자문서	- 인쇄 또는 비인쇄 형태의 사용 가능한 전체 문서들의 집합
실행프로그램	- 요구사항이 명확하게 정의된 대상

(4) ISO/IEC 25000

- ISO/IEC 2500n(품질 일반 부분)
- ISO/IEC 2501n(품질 모델 부분) : ISO/IEC 9126-1 통합
- ISO/IEC 2502n(품질 측정 부분) : ISO/IEC 9126-2,3,4 통합
- ISO/IEC 2503n(품질 요구사항 부분)
- ISO/IEC 2504n(품질 평가 부분) : ISO/IEC 14598 통합

2. 프로세스 품질 국제 표준

(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)

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)

수준	단계	설명
1	초기 단계	- 관리가 안 되는 단계
2	반복 단계	- 프로세스의 반복
3	정의 단계	- 프로젝트의 관리
4	관리 단계	- 정량적 관리
5	최적화 단계	- 지속적 개선

(4) CMMi(Capability Maturity Model Integration)

수준	단계	설명
1	초기 단계	- 프로세스가 없는 상태
2	관리 단계	- 기본적인 프로세스
3	정의 단계	- 표준 프로세스
4	정량적 관리 단계	- 정량적 관리
5	최적화 단계	- 지속적인 개선

3. 서비스 관리 국제 표준

- ISO/IEC 20000
 - 고객에게 제공하는 IT서비스의 수준을 객관적으로 평가
 - IT 조직 기능에 부합되는 견고하고 통합화된 프로세스 프레임워크를 제공

Section 2. 제품 소프트웨어 패키징

1. 애플리케이션 패키징 순서

- 기능 식별
- 모듈화
- 빌드 진행
- 사용자 환경 분석
- 패키징 적용 시험
- 패키징 변경 개선

2. 애플리케이션 배포 도구

(1) CI/CD

1) CI(Continuous Integration)

- 지속적인 통합
- 여러 개발자들의 새로운 코드 변경 사항을 정기적으로 공유 레포지토리에 통합하는 것

2) CD(Continuous Delivery & Continuous Deployment)

- 지속적인 서비스 제공 혹은 지속적인 배포

3. 릴리즈 노트

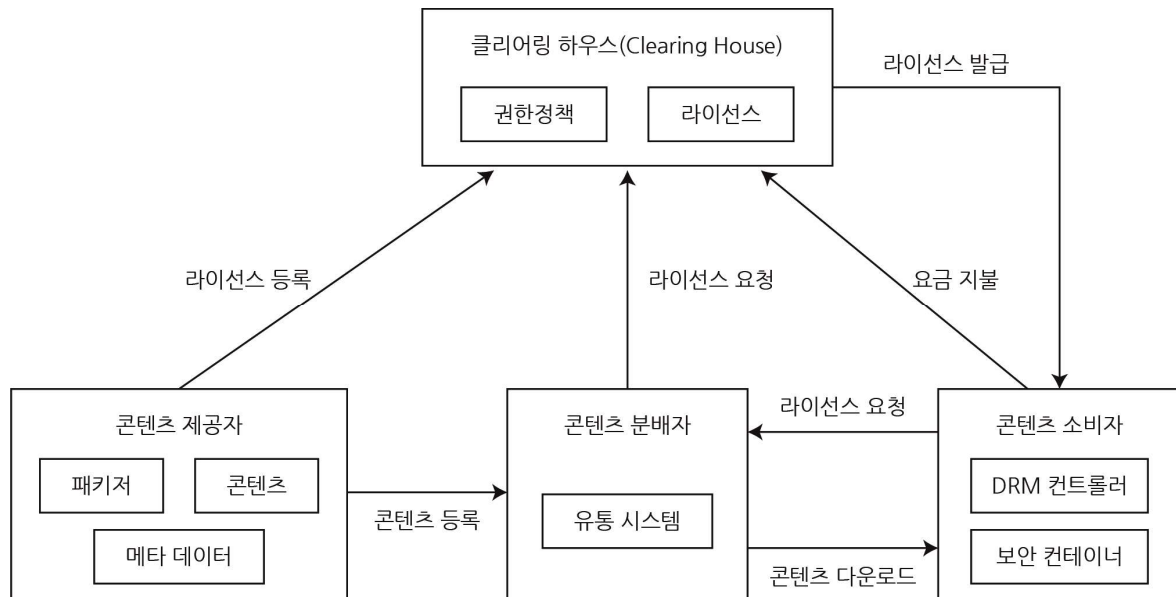
- 소프트웨어 제품과 함께 배포되는 문서들을 말한다.

4. DRM

(1) DRM(Digital Rights Management)의 개념

- 각종 디지털 콘텐츠의 불법적인 사용을 제한하고, 승인된 사용자의 콘텐츠 사용을 저작권 소유자의 의도에 따라 제어하는 기술

(2) DRM의 구성 및 흐름



(3) DRM 사용 규칙 제어 기술

1) 콘텐츠 식별 체계(Identification)

- 디지털 콘텐츠에 고유 식별 번호를 부여하여 관리하고 운영(DOI, URI)

2) 메타데이터(Meta Data)

- 콘텐츠에 관한 구조화된 데이터

3) 권리 표현 기술(Right Expression)

- 콘텐츠에 대한 규칙 설정
- XrML(eXtensible rights Mark-up Language) 기술이 대표적

4) 권리 표현 종류

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

(4) 저작권 보호 기술

1) 암호화 기술

- 특정키를 가진 사용자만이 해당 콘텐츠를 사용할 수 있도록 한다.

2) 위변조 방지(Tamper-Proofing)

- 콘텐츠에 승인되지 않은 조작이 가해졌을 때 위변조 사항을 감지하고, 오류를 발생시키는 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)

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

(5) DRM 구성요소

구성요소	설명
암호화 (Encryption)	- 콘텐츠 및 라이선스를 암호화하고, 전자 서명을 할 수 있는 기술
키 관리 (Key Manangement)	- 콘텐츠를 암호화한 키에 대한 저장 및 배포 기술
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
정책 관리 (Policy Management)	- 라이선스 발급 및 사용에 대한 정책표현 및 관리기술
크랙 방지 (Tamper Resistance)	- 크랙에 의한 콘텐츠 사용 방지 기술
인증 (Authentication)	- 라이선스 발급 및 사용의 기준이 되는 사용자 인증 기술