<2 과목 소프트웨어 개발>

---

1. 꼭 알아야 할 키워드 = ____(밑줄)

2. # = 두음 암기 or 한 칸 띄어 쓴 건 산출물

3. 시나공 + 수제비 정리 (페이지 참고)

4. "Ctrl+F" 탐색 → 제목 활용하기

1 자료구조 ★★

p.162, 2-2

1) 자료 구조의 분류

▶ 선형 구조(Linear Structure)

- 배열(Array)
- 스택(Stack)
- 큐(Queue)
- 데크(Deque)
- 선형 리스트(Linear List) = 연속 리스트(순차적임), 연결 리스트(순차적이지 않음)

▶ 비선형 구조(Non-Linear Structure)

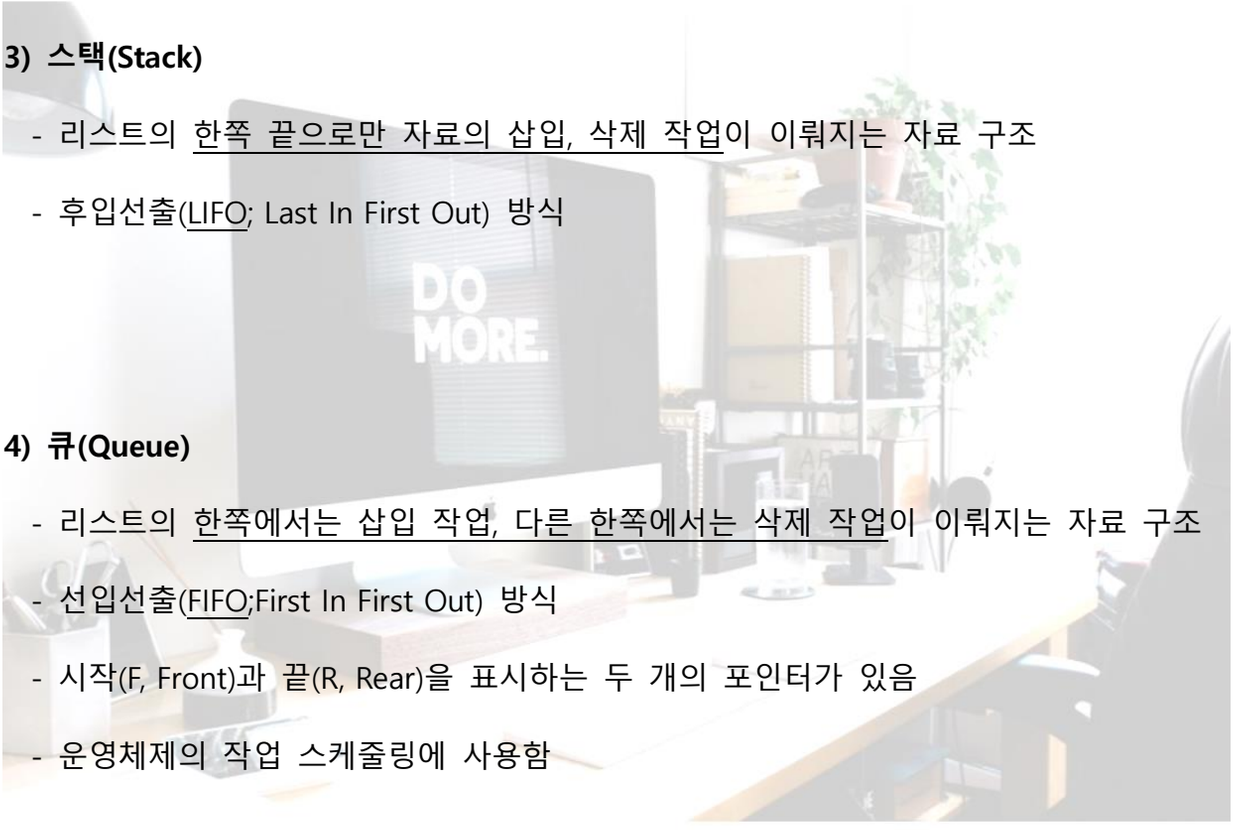
- 트리(Tree)
- 그래프(Graph)

2) 배열(Array)

- 정적인 자료 구조로 기억장소의 추가가 어렵고 메모리의 낭비가 발생함
- 첨자를 이용
- 반복적인 데이터 처리 작업에 적합한 구조
- 데이터마다 동일한 이름의 변수를 사용해 처리가 간편함

3) 스택(Stack)

- 리스트의 한쪽 끝으로만 자료의 삽입, 삭제 작업이 이뤄지는 자료 구조
- 후입선출(LIFO; Last In First Out) 방식



4) 큐(Queue)

- 리스트의 한쪽에서는 삽입 작업, 다른 한쪽에서는 삭제 작업이 이뤄지는 자료 구조
- 선입선출(FIFO;First In First Out) 방식
- 시작(F, Front)과 끝(R, Rear)을 표시하는 두 개의 포인터가 있음
- 운영체제의 작업 스케줄링에 사용함

5) 데크(Deque)

- 리스트의 양쪽 끝에서 삽입과 삭제작업을 할 수 있는 자료 구조

6) 선형 리스트(Linear List)

▶ 연속 리스트(Contiguous List)

- 배열과 같이 연속되는 기억장소에 저장되는 자료 구조
- 기억장소를 연속적으로 배정받아, 기억장소 이용 효율은 밀도가 1 로서 가장 좋음
- 중간에 데이터를 삽입하기 위해 연속된 빈 공간이 있어야함
- 삽입, 삭제 시 자료의 이동이 필요함

▶ 연결 리스트(Linked List)

- 자료들을 반드시 연속적으로 배열시키지 않고 임의의 기억공간을 기억시키되, 자료 항목의 순서에 따라 노드의 포인터 부분을 이용해 서로 연결시킨 자료 구조
- 노드의 삽입, 삭제 작업이 용이
- 기억공간이 연속적으로 놓여 있지 않아도 저장가능
- 연결을 위한 포인터가 필요하기 때문에 순차 리스트에 비해 기억 공간의 효율이 좋지 않음
- 연결을 위한 포인터를 찾는 시간이 필요하기 때문에 접근 속도가 느림
- 중간 노드 연결이 끊어지면 그 다음 노드를 찾기 힘듦

7) 트리(Tree) ★★

- 정점(Node, 노드)과 선분(Branch, 가지)을 이용해 사이클을 이루지 않도록 구성한 그래프(Graph)의 특수한 형태

▶ 노드(Node): 트리의 기본 요소, 자료 항목과 다른 항목에 대한 가지(Branch)를 합친 것

▶ 근 노드(Root Node): 트리의 맨 위에 있는 노드

▶ **디그리(Degree, 차수)**: 각 노드에서 뻗어 나온 가지의 수 ★

▶ **단말 노드(Terminal Node)**: 자식이 하나도 없는 노드, Degree 가 0 인 노드 ★

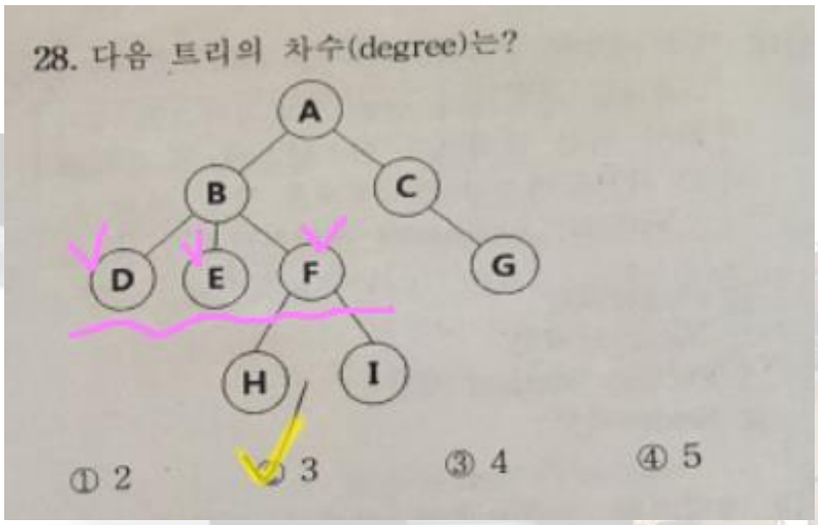
▶ 자식 노드(Son Node): 어떤 노드에 연결된 다음 레벨의 노드들

▶ 부모 노드(Parent Node): 어떤 노드에 연결된 이전 레벨의 노드들

▶ 형제 노드(Brother Node, Sibling): 동일한 부모를 갖는 노드들

▶ 트리의 디그리: 노드들의 디그리 중에서 가장 많은 수

차수(Degree)와 **단말(Terminal)의 수**를 물어보는 기출문제 多 __ **1, 2, 3 회 기출문제**



차수(Degree): 3 - D, E, F (B 의 자식 노드)

단말(Terminal): 5 - D, E, H, I, G (자식이 하나도 없는 노드)

8) 그래프(Graph)

▶ 방향 그래프

- 정점을 연결하는 선에 방향이 있는 그래프

- n 개의 정점으로 구성된 방향 그래프의 최대 간선 수 = n(n-1)

▶ 무방향 그래프

- 정점을 연결하는 선에 방향이 없는 그래프

- n 개의 정점으로 구성된 무방향 그래프의 최대 간선 수 = n(n-1)/2

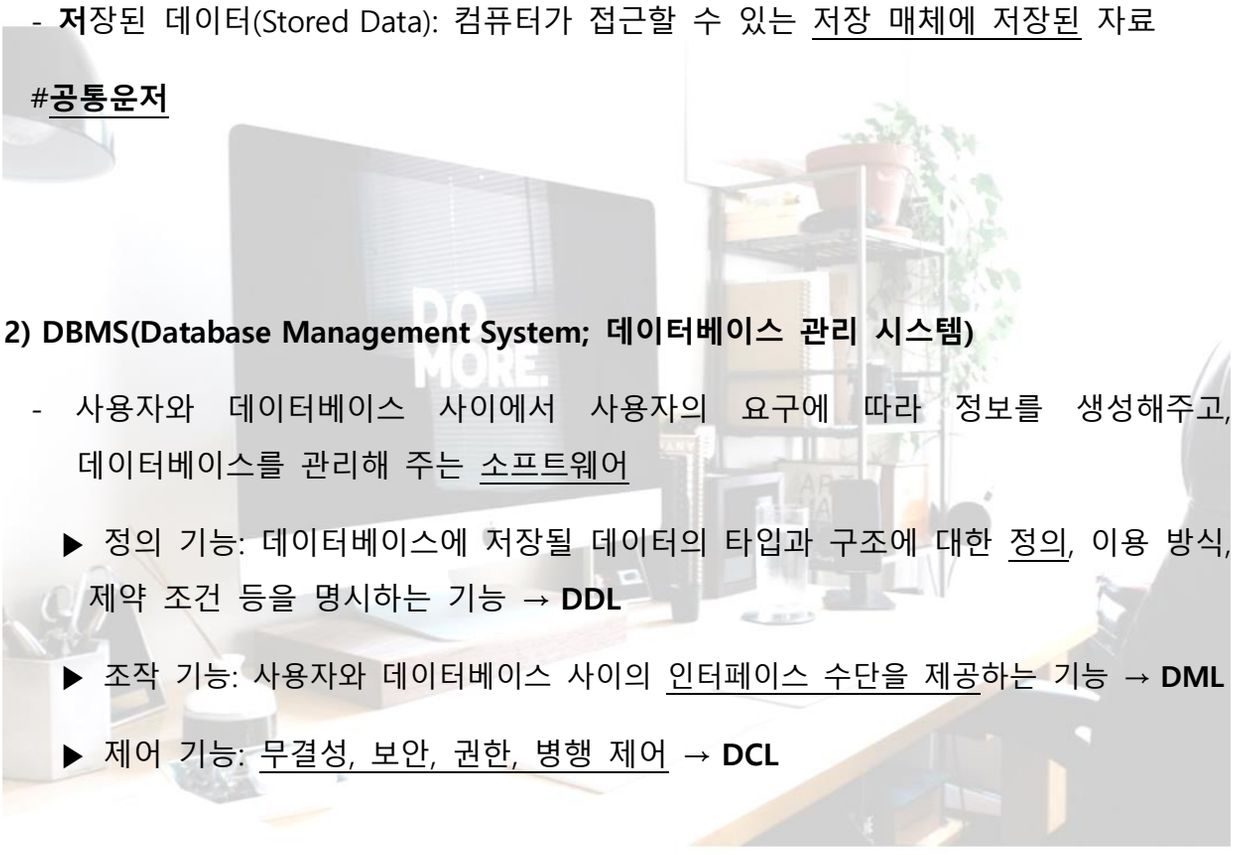
2 데이터베이스 / DBMS ★

p.168

1) 데이터베이스(Database) ★

- **공**용 데이터(Shared Data): 여러 응용 시스템들이 공동으로 소유하고 유지하는 자료
- **통**합된 데이터(Integrated Data): 자료의 중복을 최대로 배제한 데이터의 모임
- **운**영 데이터(Operational Data): 고유한 업무를 수행하는 데 없어서는 안 될 자료
- **저**장된 데이터(Stored Data): 컴퓨터가 접근할 수 있는 저장 매체에 저장된 자료

#공통운저



2) DBMS(Database Management System; 데이터베이스 관리 시스템)

- 사용자와 데이터베이스 사이에서 사용자의 요구에 따라 정보를 생성해주고, 데이터베이스를 관리해 주는 소프트웨어
  - ▶ 정의 기능: 데이터베이스에 저장될 데이터의 타입과 구조에 대한 정의, 이용 방식, 제약 조건 등을 명시하는 기능 → **DDL**
  - ▶ 조작 기능: 사용자와 데이터베이스 사이의 인터페이스 수단을 제공하는 기능 → **DML**
  - ▶ 제어 기능: 무결성, 보안, 권한, 병행 제어 → **DCL**

3 데이터 입, 출력 ★★

p.171

1) SQL(Structured Query Language)

- 1974 년 IBM 연구소에서 개발한 SEQUEL 에서 유래함
- 관계대수와 관계해석을 기초로 한 혼합 데이터 언어

▶ 데이터 정의어(DDL; Data Define Language): DOMAIN(**도**메인), SCHEMA(**스**키마), TABLE(**테**이블), VIEW(**뷰**), INDEX(**인**덱스)를 **정의**하거나 **변경** 또는 **삭제**할 때 사용하는 언어

#**도스테뷰인**

▶ 데이터 조작어(DML; Data Manipulation Language): SELECT(검색), INSERT(삽입), UPDATE(갱신), DELETE(삭제)로 저장된 데이터를 실질적으로 처리하는 데 사용하는 언어

▶ 데이터 제어어(DCL; Data Control Language): 데이터의 무결성, 보안, 회복, 병행 제어 등을 정의하는 데 사용되는 언어

2) 데이터 접속(Data Mapping) __ 2-18

- 소프트웨어의 기능 구현을 위해 프로그래밍 코드와 데이터베이스의 데이터를 연결(Mapping)하는 것을 말함

▶ **SQL Mapping**: 프로그래밍 코드 내 SQL 을 직접 입력해 DBMS 의 데이터에 접속하는 기술 ★

JDBC, ODBC, MyBatis

▶ **ORM**(Object-Relational Mapping): 객체(Object)와 관계형데이터베이스(RDB)의 데이터를 연결(Mapping)하는 기술 ★

JPA, Hibernate, Django

3) 트랜잭션(Transaction) ★★ _ 2-19

- 데이터베이스의 상태를 변환시키는 하나의 논리적 기능을 수행하기 위한 작업의 단위
- 한꺼번에 모두 수행되어야 할 일련의 연산들
  - ▶ COMMIT: 트랜잭션 처리가 정상적으로 종료되어 수행한 변경 내용을 DB 에 반영하는 명령어
  - ▶ ROLLBACK: 트랜잭션 처리가 비정상으로 종료되어 DB 의 길관성이 깨졌을 때 트랜잭션이 행한 모든 변경 작업을 취소하고 이전 상태로 되돌리는 연산
  - ▶ SAVEPOINT(=CHECKPOINT): 트랜잭션 내에서 ROLLBACK 할 위치인 저장점을 지정하는 명령어, 여러 개의 SAVEPOINT 지정 가능

원리	특징
원자성 (**A**tomicity)	트랜잭션 연산을 데이터베이스 모두에 반영 또는 반영하지 말아야 함(All or Nothing)
일관성 (**C**onsistency)	트랜잭션이 실행을 성공적으로 완료할 시 일관성 있는 데이터베이스 상태를 유지
독립성 (**I**solation, 격리성)	둘 이상 트랜잭션 동시 실행 시 한 개의 트랜잭션만 접근이 가능하여 간섭 불가
영속성 (**D**urability)	성공적으로 완료된 트랜잭션 결과는 영구적으로 반영됨

#**ACID**

4 절차형 SQL ★

p.173, 2-22

1) 개요

- C, JAVA 등의 프로그래밍 언어와 같이 연속적인 실행이나 분기, 반복 등의 제어가 가능한 SQL
- 일반적인 프로그래밍 언어에 비해 효율이 떨어짐
- 연속적인 작업들을 처리하는데 적합
- BEGIN ~ END 형식으로 작성되는 블록(Block) 구조로 기능별 모듈화 가능
  - ▶ **프로시저(Procedure)**: 호출을 통해 실행되어 미리 저장해 놓은 SQL 작업 수행, 처리 결과는 한 개 이상의 값 혹은 반환을 아예 하지 않음
  - ▶ **트리거(Trigger)**: 입력, 갱신, 삭제 등의 이벤트가 발생할 때마다 관력 작업을 자동 수행
  - ▶ **사용자 정의 함수**: 프로시저와 유사하게 SQL을 사용해 일련의 작업을 연속적으로 처리함, 종료 시 예약어 RETURN을 사용해 처리 결과를 단일값으로 반환

2) 테스트와 디버깅

- 테스트 전 구문 오류(Syntax Error)나 참조 오류의 존재 여부 확인
- 오류 및 경고 메시지가 상세히 출력되지 않으므로 SHOW 명령어를 통해 내용 확인
- 실제로 데이터베이스에 변화를 줄 수 있는 삽입 및 변경 관련 SQL 문을 주석으로 처리하고 디버깅 수행

3) 쿼리 성능 최적화

- 데이터 입, 출력 애플리케이션의 성능 향상을 위해 SQL 코드를 최적화하는 것
- 성능 측정 도구 APM(Application Performance Management/Monitoring)을 사용해 최적화할 쿼리를 선정
- 최적화할 쿼리에 대해 옵티마이저(Optimizer)가 수립한 실행 계획을 검토하고 SQL 코드와 인덱스 재구성

5 개발 지원 도구 ★★

p.186, 2-38

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

- 개발에 필요한 환경, 즉 편집기(Editor), 컴파일러(Compiler), 디버거(Debugger) 등의 다양한 툴을 하나의 인터페이스로 통합해 제공하는 것을 의미함

▶ 이클립스(Eclipse) ... IBM

▶ 비주얼 스튜디오(Visual Studio) ... Microsoft

▶ 엑스 코드(X Code) ... Apple

▶ 안드로이드 스튜디오(Android Studio) ... Google

▶ IDEA ... JetBrains

2) 빌드 자동화 도구 __ p.218, 2-70

- 소스 코드를 소프트웨어로 변환하는 과정에 필요한 전처리(Preprocessing), 컴파일(Complie) 등의 작업들을 수행하는 소프트웨어

▶ **Ant(Another Neat Tool)**

-아파치 소프트웨어 재단에서 개발한 소프트웨어

-자바 프로젝트의 공식적인 빌드 자동화 도구

-XML 기반의 빌드 스크립트를 사용

-정해진 규칙이나 표준이 없어 개발자가 모든 것을 정의

-스크립트의 재사용이 어려움

▶ **Maven**

-아파치 소프트웨어 재단에서 Ant 의 대안으로 개발

-규칙이나 표준이 존재해 예외 사항만 기록됨

-컴파일과 빌드를 동시에 수행할 수 있음

-의존성(Dependency)을 설정하여 라이브러리를 관리

▶ **Gradle ★**

-기존의 Ant 와 Maven 을 보완해 개발된 빌드 자동화 도구

-안드로이드 스튜디오(안드로이드 앱 개발)의 공식 빌드 도구

-Maven 과 동일하게 의존성(Dependency) 활용

-그루비(Groovy) 기반의 빌드 스크립트 사용

-플러그인을 설정하면, JAVA, C/C++, Python 등의 언어도 빌드 가능

-실행할 처리 명령들을 모아 태스크(Task)로 만든 후 태스크 단위로 실행

-이전에 사용했던 태스크를 재사용하거나 다른 시스템의 태스크를 공유할 수 있는 빌드 캐시 기능 지원 → 빌드의 속도 향상

▶ **Jenkins ★**

JAVA 기반의 오픈 소스 형태로 가장 많이 사용되는 빌드 자동화 도구

서블릿 컨테이너에서 실행되는 서버 기반 도구

SVN, Git 등 대부분의 형상 관리 도구와 연동 가능

친숙한 Web GUI 제공

여러 대의 컴퓨터를 이용한 분산 빌드나 테스트 가능

3) 기타 협업 도구(Groupware, 그룹웨어)

종류	내용
일정 관리 도구	구글 캘린더
프로젝트 관리 도구	트렐로(Trello), 지라(Jira)
정보 공유 및 커뮤니케이션 도구	슬랙(Slack), 잔디(Jandi), 태스크월드(Task world)
디자인 도구	스케치(Sketch), 제플린(Zeplin)
아이디어 공유 도구	에버노트(Evernote)
형상 관리 도구	깃허브(GitHub)

6 소프트웨어 패키징 ★

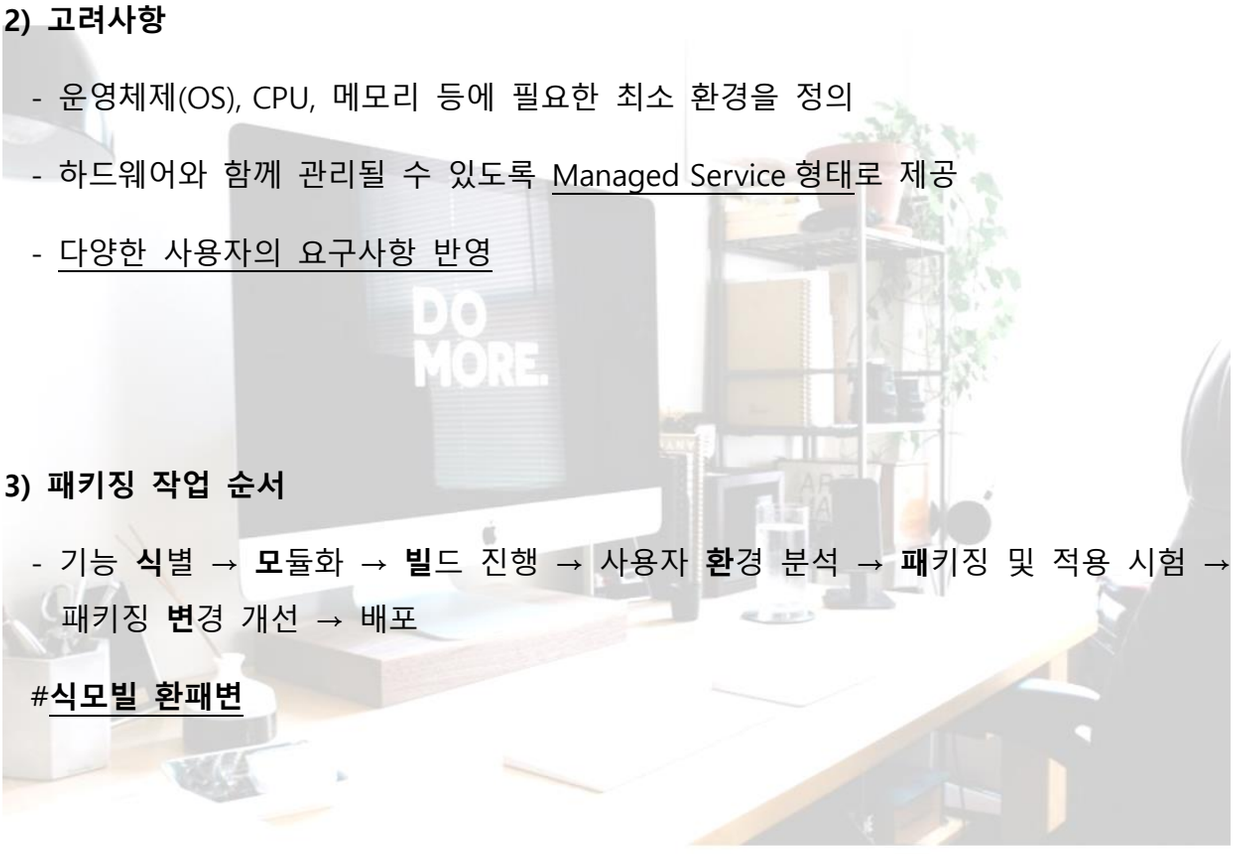
p.194, 2-44

1) 개요

- 모듈별로 생성한 실행 파일들을 묶어 배포용 설치 파일을 만드는 것
- 개발자가 아닌 사용자를 중심으로 진행

2) 고려사항

- 운영체제(OS), CPU, 메모리 등에 필요한 최소 환경을 정의
- 하드웨어와 함께 관리될 수 있도록 Managed Service 형태로 제공
- 다양한 사용자의 요구사항 반영



3) 패키징 작업 순서

- 기능 **식**별 → **모**듈화 → **빌**드 진행 → 사용자 **환**경 분석 → **패**키징 및 적용 시험 → 패키징 **변**경 개선 → 배포

#**식모빌 환패변**

4) 제품 소프트웨어 패키징 도구 활용 시 고려사항 ★ __ 20 년 1, 2, 3 회 기출문제

- 패키징 시 사용자에게 배포되는 SW 이므로 보안 고려
- 사용자 편의성을 위한 복잡성 및 비효율성 문제 고려
- 제품 SW 종류에 적합한 암호화 알고리즘 적용
- 다양한 이기종 연동 고려

7 릴리즈 노트 ★

p.196, 2-46

1) 릴리즈 노트(Release Note)의 개요

- 개발 과정에서 정리된 릴리즈 정보를 소프트웨어의 고객과 공유하기 위한 문서
- 개선된 작업이 있을 때마다 관련 내용을 릴리즈 노트에 담아 제공
- 개발팀에서 제공하는 소프트웨어 사양에 대한 최종 승인을 얻은 후 문서화되어 제공

2) 초기 버전 작성 시 고려사항

항목	내용
Header(머리말) ★	릴리즈 노트 이름, 소프트웨어 이름, 릴리즈 버전, 릴리즈 날짜, 릴리즈 노트 날짜, 릴리즈 노트 버전 등 (**20 년 1 회차 실기 기출문제**)
개요	소프트웨어 및 변경사항 전체에 대한 간략한 내용
목적	해당 릴리즈 버전에서의 새로운 기능이나 수정된 기능의 목록과 릴리즈 노트의 목적에 대한 간략한 개요
문제 요약	수정된 버그에 대한 간략한 설명 또는 릴리즈 추가 항목에 대한 요약
재현 항목	버그 발견에 대한 과정 설명
수정/개선 내용	버그를 수정/개선한 내용을 간단히 설명
사용자 영향도	사용자가 다른 기능들을 사용하는데 있어 해당 릴리즈 버전에서의 기능 변화가 미칠 수 있는 영향에 대한 설명
SW 지원 영향도	해당 릴리즈 버전에서의 기능 변화가 다른 응용 프로그램들을 지원하는 프로세스에 미칠 수 있는 영향에 대한 설명
노트	SW/HW 설치 항목, 업그레이드, 소프트웨어 문서화에 대한 참고 항목
면책 조항	회사 및 소프트웨어와 관련하여 참조할 사항 ex) 프리웨어, 불법 복제 금지 등
연락처	사용자 지원 및 문의 응대를 위한 연락처 정보

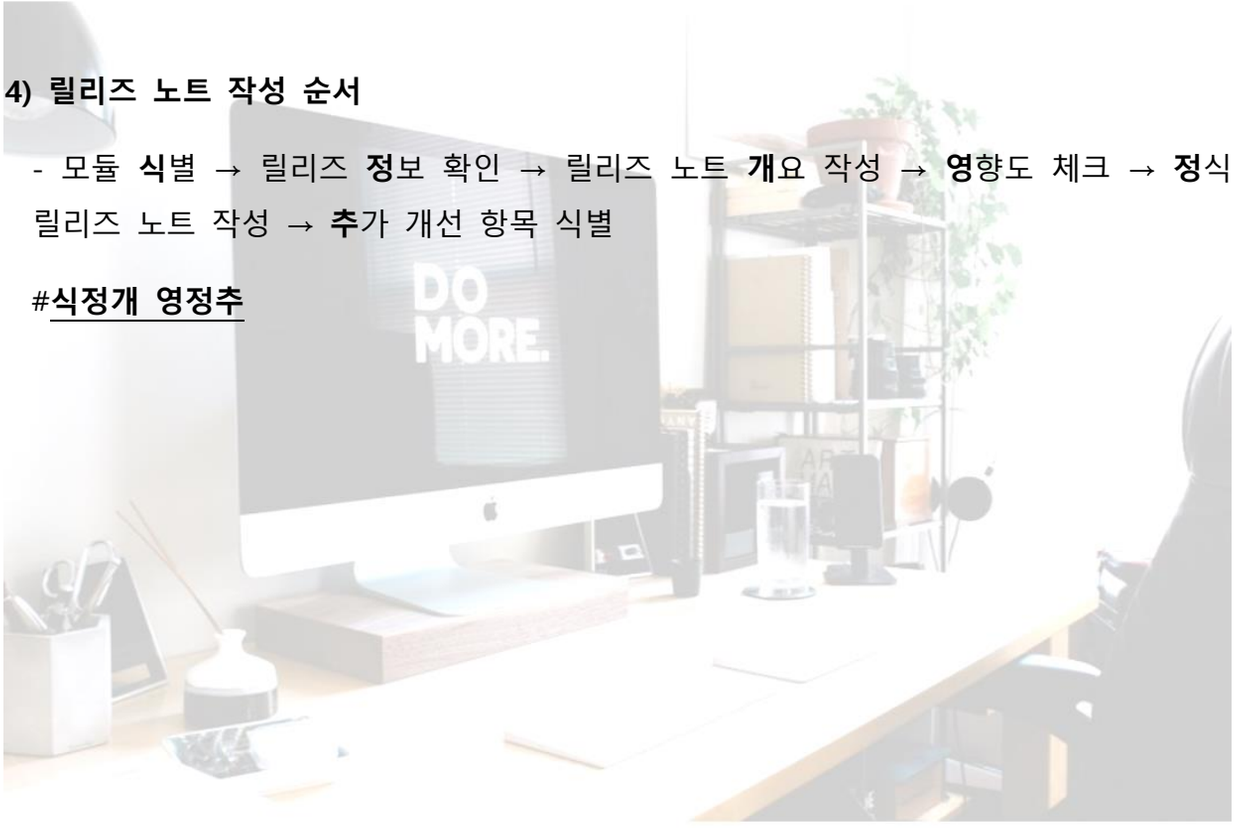
3) 추가 버전 작성 시 고려사항

- 베타 버전이 출시되거나 긴급한 버그 수정, 업그레이드와 같은 자체 기능 향상, 사용자 요청 등의 특수한 상황이 발생하는 경우 추가로 작성
- 버그 번호를 포함한 모든 수정된 내용을 담아 릴리즈 노트 작성
- 추가나 수정된 경우 자체 기능 향상과는 다른 별도의 릴리즈 버전 출시하고 릴리즈 노트 작성

4) 릴리즈 노트 작성 순서

- 모듈 **식**별 → 릴리즈 **정**보 확인 → 릴리즈 노트 **개**요 작성 → **영**향도 체크 → **정**식 릴리즈 노트 작성 → **추**가 개선 항목 식별

#식정개 영정추



8 디지털 저작권 관리 ★★

p.200, 2-51

- 디지털 콘텐츠의 전 과정에 걸쳐 사용되는 디지털 콘텐츠 관리 및 보호 기술

1) 디지털 저작권 관리(DRM; Digital Right Management)의 흐름 ★

- **콘**텐츠 제공자(Contents Provider): 콘텐츠를 제공하는 저작권자

  콘텐츠 분배자(Contents Provider): 암호화된 콘텐츠를 유통하는 곳이나 사람

  콘텐츠 소비자(Customer): 콘텐츠를 구매해서 사용하는 주체

- **패**키저(Packager): 콘텐츠를 메타 데이터와 함께 배포 가능한 형태로 묶어 암호화하는 프로그램

- **클**리어링 하우스(Clearing House): 저작권에 대한 사용 권한, 라이선스 발급, 사용량에 따른 결제관리 등을 수행하는 곳

- DRM **컨**트롤러(DRM Controller): 배포된 콘텐츠의 이용 권한을 통제하는 프로그램

- **보**안 컨테이너(Security Container): 콘텐츠 원본을 안전하게 유통하기 위한 전자적 보안 장치

#콘패클컨보

2) 디지털 저작권 관리의 기술 요소 ★ __ 2-110, 20 년 1, 2, 3 회 기출문제

- **암**호화(Encryption): 콘텐츠 및 라이선스를 암호화하고 전자서명을 할 수 있는 기술

- **키** 관리(Key Management): 콘텐츠를 암호화한 키에 대한 저장 및 분배 기술

- **식**별 기술(Identification): 콘텐츠에 대한 식별 체계 표현 기술

- **저**작권 표현(Right Expression): 라이선스의 내용 표현 기술

- 암호화 **파**일 생성(Packager): 콘텐츠를 암호화된 콘텐츠로 생성하기 위한 기술

- **정**책 관리(Policy Management): 라이선스 발급 및 사용에 대한 정책 표현 및 관리 기술

- **크**랙 방지(Tamper Resistance): 크랙에 의한 콘텐츠 사용 방지 기술

- **인**증(Authentication): 라이선스 발급 및 사용의 기준이 되는 사용자 인증 기술

#암키식저 파정크인

9 형상 관리 ★★

p.210, 2-40

1) 소프트웨어 패키징의 형상 관리(SCM; Software Configuration Management)

- 형상 관리는 소프트웨어의 개발 과정에서 소프트웨어의 변경 사항을 관리하기 위해 개발된 일련의 활동임
- 소프트웨어 개발의 전 단계에 적용되는 활동이며, 유지보수 단계에서도 수행

2) 형상 관리의 중요성

- 소프트웨어의 변경 사항을 체계적으로 추적하고 통제할 수 있음
- 제품 소프트웨어에 대한 무절제한 변경 방지
- 진행 정도를 확인하기 위한 기준으로 사용될 수 있음

3) 형상 관리 기능

- 형상 **식**별: 형상 관리 대상에 이름과 관리 번호를 부여하고, 계층(Tree) 구조로 구분하여 수정 및 추적이 용이하도록 하는 작업
- 형상 **통**제(변경 관리): 식별된 형상 항목에 대한 변경 요구를 검토하여 현재의 기준선(베이스 라인, Base line)이 잘 반영될 수 있도록 조정하는 작업
- 형상 **감**사: 기준선(베이스 라인)의 무결성을 평가하기 위해 확인, 검증, 검열 과정을 통해 공식적으로 승인하는 작업

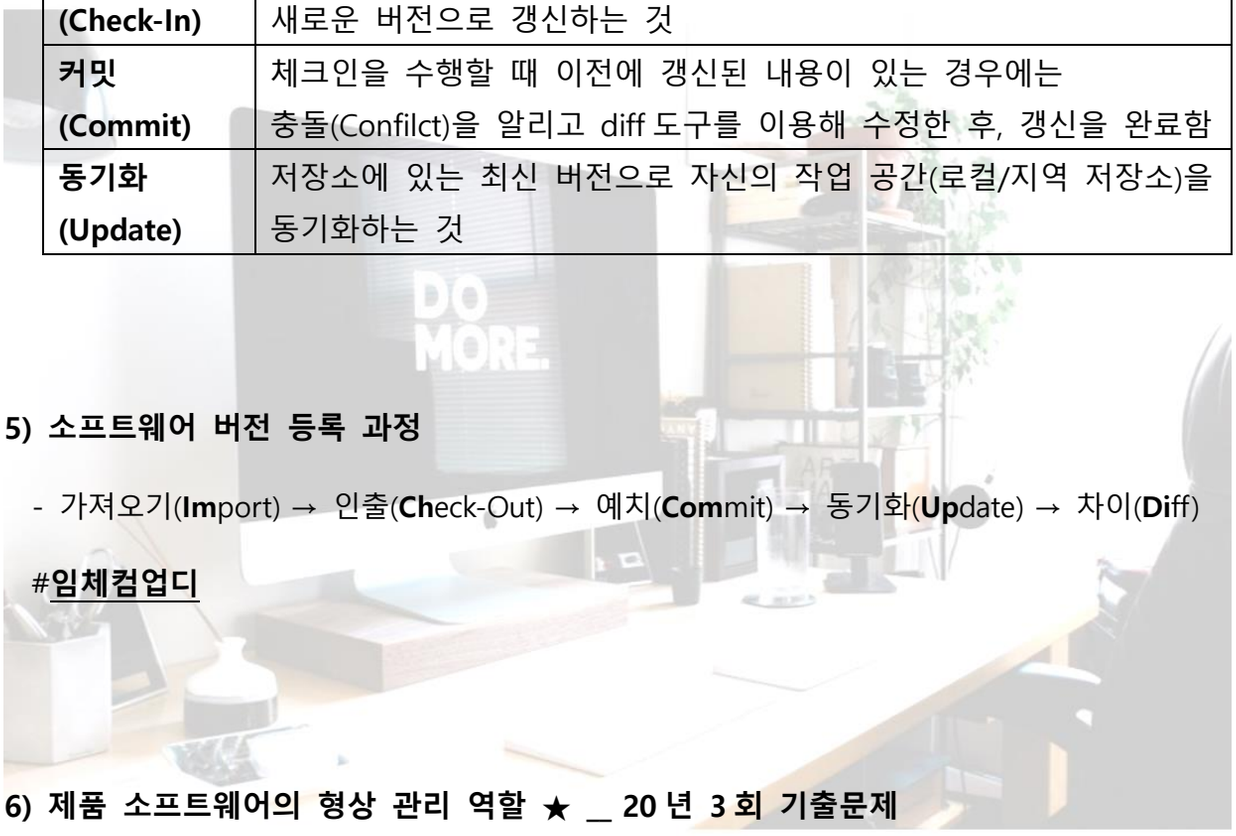
- 형상 **기**록(상태 보고): 형상의 식별, 통제, 감사 작업의 결과를 기록, 관리하고 보고서를 작성하는 작업

#**식통감기**

- 버전 제어: 소프트웨어 업그레이드나 유지 보수 과정에서 생성된 다른 버전의 형상 항목을 관리하고, 이를 위해 특정 절차와 도구(Tool)를 결합시키는 작업

4) 소프트웨어 버전 등록 관련 주요 용어

명령어	설명
저장소 (Repository)	최신 버전의 파일들과 변경 내역에 대한 정보들이 저장되어 있는 곳
가져오기 (Import)	버전 관리가 되고 있지 않은 아무것도 없는 저장소(Repository)에 처음으로 파일을 복사하는 것
체크아웃 (Check-Out)	프로그램을 수정하기 위해 저장소(Repository)에서 파일을 받아오는 것
체크인 (Check-In)	체크아웃 한 파일의 수정을 완료한 후, 저장소(Repository)의 파일을 새로운 버전으로 갱신하는 것
커밋 (Commit)	체크인을 수행할 때 이전에 갱신된 내용이 있는 경우에는 충돌(Confilct)을 알리고 diff 도구를 이용해 수정한 후, 갱신을 완료함
동기화 (Update)	저장소에 있는 최신 버전으로 자신의 작업 공간(로컬/지역 저장소)을 동기화하는 것



5) 소프트웨어 버전 등록 과정

- 가져오기(**Im**port) → 인출(**Ch**eck-Out) → 예치(**Com**mit) → 동기화(**Up**date) → 차이(**Di**ff)

#**임체컴업디**

6) 제품 소프트웨어의 형상 관리 역할 ★ _ 20 년 3 회 기출문제

- 형상 관리를 통해 이전 리비전이나 버전에 대한 정보에 접근 가능하여 배포본 관리에 유용

- 불필요한 사용자의 소스 수정 제한
- 동일한 프로젝트에 대해 여러 개발자 동시 개발 가능

10 버전 관리 도구 ★★

p.213, 2-67

1) 공유 폴더 방식

- 버전 관리 자료가 로컬 컴퓨터의 공유 폴더에 저장되어 관리되는 방식
- 개발자들은 개발이 완료된 파일을 약속된 공유 폴더에 매일 복사함
- 담당자는 공유 폴더의 파일을 자기 PC 로 복사해 컴파일 한 후 이상 유무 확인
- 파일의 변경 사항을 데이터베이스에 기록하며 관리

SCCS, RCS, PVCS, QVCS

2) 클라이언트/서버 방식

- 버전 관리 자료가 중앙 시스템(서버)에 저장되어 관리되는 방식
- 서버의 자료를 개발자별로 자신의 PC(클라이언트)로 복사해 작업한 후 변경된 내용을 중앙 서버에 반영
- 모든 버전 관리는 서버에서 수행됨
- 하나의 파일을 서로 다른 개발자가 작업할 경우 경고 메시지 출력
- 서버에 문제가 생기면 다른 개발자와의 협업 및 버전 관리 작업은 중단됨

CVS, SVN(Subversion)

3) 분산 저장소 방식

- 하나의 원격 저장소와 분산된 개발자 PC 의 로컬 저장소에 함께 저장되어 관리되는 방식

- 개발자별로 원격 저장소의 자료를 자신의 로컬 저장소로 복사해 작업한 후 변경 된 내용을 로컬 저장소에서 우선 반영(Commit)한 다음 이를 원격 저장소에 반영(Push)
- 원격 저장소에 문제가 생겨도 로컬 저장소의 자료를 이용해 작업 가능
- 로컬 저장소에서 작업을 수행할 수 있어 처리속도가 빠름

Git, Bitkeeper

4) SVN(Subversion)

- CVS 를 개선한 것으로 아파치 소프트웨어 재단에서 2000 년 발표함
- 모든 개발 작업은 trunk 디렉터리에서 수행되며, 추가 작업은 branches 디렉터리 안에 별도의 디렉터리를 만들어 작업을 완료한 후 trunk 디렉터리와 병합(merge)
- 커밋(Commit)할 때마다 리비전(Revision)이 1 씩 증가
- 서버는 주로 유닉스(UNIX) 사용
- 오픈 소스로 무료사용 가능
- CVS 의 단점이었던 파일이나 디렉터리의 이름 변경, 이동 등이 가능

명령어	의미
add	새로운 파일이나 디렉터리를 버전 관리 대상으로 등록
commit	버전 관리 대상으로 등록된 클라이언트의 소스 파일을 서버의 소스 파일에 적용
update	서버의 최신 commit 이력을 클라이언트의 소스파일에 적용
checkout	버전 관리 정보와 소스 파일을 서버에서 클라이언트로 받아옴
lock/unlock	서버의 소스 파일이나 디렉터리를 잠그거나 해제
import	아무것도 없는 서버의 저장소에 맨 처음 소스 파일을 저장하는 명령으로 한 번 사용 후 다시 사용하지 않음
export	버전 관리에 대한 정보를 제외한 순수한 소스 파일만을 서버에서 받아옴
info	지정한 파일에 대한 위치나 마지막 수정 일자 등에 대한 정보를 표시
diff	지정된 파일이나 경로에 대해 이전 리비전과의 차이를 표시
merge	다른 디렉터리에서 작업된 버전 관리 내역을 기본 개발 작업과 병합

5) Git(깃) ★

- 리누스 토발즈(Linus Torvalds)가 2005 년 리눅스 커널 개발에 사용할 관리 도구로 개발한 이후 주니오 하마노(Junio Hamano)에 의해 유지 보수되고 있음
- 원격 저장소는 여러 사람들이 협업을 위해 버전을 공동 관리하는 곳으로, 자신의 버전 관리 내역을 반영(Push)하거나 다른 개발자의 변경 내용을 가져올 때(Fetch) 사용
- 로컬 저장소는 개발자들이 본인의 실제 개발을 진행하는 장소로 버전 관리가 수행됨
- 브랜치(Branch)를 이용하면 기본 버전 관리 틀에 영향을 주지 않으면서 다양한 형태의 기능 테스팅 가능
- 파일의 변화를 스냅샷(Snapshot)으로 저장
- 스냅샷은 이전 스냅샷의 포인터를 가지므로 버전의 흐름 파악 가능

명령어	의미
add	작업 내역을 로컬 저장소에 저장하기 위해 스테이징 영역에 추가함, '--all' 옵션으로 작업 디렉터리의 모든 파일을 스테이징 영역에 추가 가능
commit	작업 내역을 로컬 저장소에 저장
branch	새로운 브랜치를 생성, 최초로 commit 을 하면 마스터(master) 브랜치가 생성됨, commit 할 때마다 해당 브랜치는 가장 최근의 commit 한 내용을 가리킴, '-d'옵션으로 브랜치 삭제 가능
checkout	지정한 브랜치로 이동
merge	지정한 브랜치의 변경 내역을 현재 HEAD 포인터가 가리키는 브랜치에 반영함으로써 두 브랜치를 병합
init	로컬 저장소를 생성
remote add	원격 저장소에 연결
push	로컬 저장소의 변경 내역을 원격 저장소에 반영
fetch	원격 저장소의 변경 이력만을 로컬 저장소로 가져와 반영
clone	원격 저장소의 전체 내용을 로컬 저장소로 복제
fork	지정한 원격 저장소의 내용을 자신의 원격 저장소로 복제

git init → git remote add → git add -all → git commit → git push

11 애플리케이션 테스트 ★★

p.224, 2-78

1) 애플리케이션 테스트의 개념

- 애플리케이션에 잠재되어 있는 결함을 찾아내는 일련의 행위 또는 절차
- 개발된 소프트웨어가 고객의 요구사항을 만족시키는지 확인(Validation)
- 소프트웨어가 기능을 정확히 수행하는지 검증(Verification)

2) 애플리케이션 테스트의 기본 원리 ★ __ 20 년 1, 2 회 기출문제

종류	설명
테스팅은 **결**함이 존재함을 밝히는 것	결함을 줄일 순 있지만, 결함이 없다고는 증명할 수 없음
완벽한 테스팅은 불가능	무한 경로, 무한 입력 값으로 인한 어려움
개발 **초**기에 테스팅 시작	테스팅 기간 단축, 재작업 감소로 개발 기간 단축 및 결함 예방
결함 **집**중	20%의 모듈에서 80%의 결함 발견, 파레토(Pareto) 법칙 ★
살충제 패러독스 (20 년 1 회차 실기 기출문제)	동일한 테스트 케이스에 의한 반복적 테스트는 새로운 버그를 찾지 못함 ★
테스팅은 **정**황에 의존적	소프트웨어 성격에 맞게 테스트 실시
오류-부재의 궤변	요구사항을 충족시켜주지 못한다면, 결함이 없다고 해도 품질이 높다 볼 수 없음

#**결완초집 살정오**

12 애플리케이션 테스트의 분류 ★★

p.227, 2-33

1) 프로그램 실행 여부에 따른 테스트

- 정적 테스트: 프로그램을 실행하지 않고 명세서나 소스 코드를 대상으로 분석하는 테스트
  # 워크 스루, 인스펙션, 코드 검사

- 동적 테스트: 프로그램을 실행하여 오류를 찾는 테스트
  # 화이트박스 테스트, 블랙박스 테스트

★ 테스트 = 검사

2) 테스트 기반에 따른 테스트

- **명**세 기반 테스트: 사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 만들어 구현하고 있는지 확인하는 테스트
  # 동등 분할, 경계값 분석(블랙박스 테스트)

- **구**조 기반 테스트: 소프트웨어 내부의 논리 흐름에 따라 테스트 케이스를 작성하고 확인하는 테스트
  # 구문 기반, 결정 기반, 조건 기반(화이트박스 테스트)

- **경**험 기반 테스트: 테스터의 경험을 기반으로 수행하는 테스트
  # 에러 추정, 체크 리스트, 탐색적 테스팅

#**명구경**

3) 시각에 따른 테스트

- 검증(Verification) 테스트: 개발자의 시각에서 제품의 생산 과정을 테스트하는 것

단위 테스트, **통**합 테스트, **시**스템 테스트

- 확인(Validation) 테스트: 사용자의 시각에서 생산된 제품의 결과를 테스트하는 것

인수 테스트(알파 테스트, 베타 테스트) ★ __ **1, 2, 3 회 기출문제**

4) 목적에 따른 테스트

종류	설명
회복(Recovery) 테스트	시스템에 여러가지 결함을 주어 실패하도록 한 후 올바르게 복구되는지를 확인하는 테스트
안전(Security) 테스트	시스템 보호 도구가 불법적인 침입으로부터 시스템을 보호할 수 있는지를 확인하는 테스트
강도(Stress) 테스트	과부하 시에도 소프트웨어가 정상적으로 실행되는지 확인하는 테스트
성능(Performance) 테스트	실시간 성능이나 전체적인 효율성을 진단하는 테스트
구조(Structure) 테스트	소프트웨어 내부의 논리적인 경로, 소스 코드의 복잡도 등을 평가하는 테스트
회귀(Regression) 테스트	소프트웨어의 변경 또는 수정된 코드에 새로운 결함이 없음을 확인하는 테스트 ★
병행(Parallel) 테스트	변경된 소프트웨어와 기존 소프트웨어에 동일한 데이터를 입력하여 결과를 비교하는 테스트

5) 테스트 커버리지 유형 __ 2-34

기법	설명
구문 커버리지	프로그램 내 모든 문장을 적어도 한 번 이상 실행하는 것을 기준으로 수행하는 테스트 커버리지
결정 커버리지	결정 조건 내 전체 조건식이 최소한 참/거짓 한 번의 값을 가지도록 측정하는 테스트 커버리지
조건 커버리지	전체 조건식 결과와 관계없이 관계없이 각 개별 조건식이 참/거짓 한 번 모두 갖도록 개별 조건식을 조합하는 테스트 커버리지
조건/결정 커버리지	전체 조건식이 참/거짓 한 번씩 가지면서, 개별 조건식이 참/거짓 모두 한 번씩 갖도록 조합하는 테스트 커버리지
변경/조건 결정 커버리지	각 개별 조건식이 다른 개별 조건식의 영향을 받지 않고 전체 조건식의 결과에 독립적으로 영향을 주도록 함으로써 조건/결정 커버리지를 향상시킨 테스트 커버리지
다중 조건 커버리지	결정 조건 내 모든 개발 조건식의 모든 가능한 조합을 100% 보장하는 테스트 커버리지

#**구결조 조변다**

13 화이트박스 테스트, 블랙박스 테스트 ★★★

p.229, 2-33, 2-77

1) 화이트박스 테스트(White Box Test) __ 20 년 1, 2, 3 회 기출문제

- 모듈 안의 내용(작동)을 직접 볼 수 있음
- 내부의 **논리적인** 모든 경로를 테스트해 테스트 케이스를 설계
- 소스 코드(Source Code)의 모든 문장을 한번 이상 수행함으로써 진행됨
- 선택, 반복 등의 부분들을 수행함으로써 **논리적** 경로 점검

종류	설명
기초 경로 검사 (Base Path Testing)	대표적인 화이트박스 테스트 기법 테스트 측정 결과는 실행 경로의 기초를 정의하는 지침으로 사용
제어 구조 검사	▶ **조**건 검사(Condition Testing): 논리적 조건을 테스트하는 기법 ▶ **루**프 검사(Loop Testing): 반복(Loop) 구조에 맞춰 테스트하는 기법 ▶ 데이터 **흐**름 검사(Data Flow Testing): 프로그램에서 변수의 정의와 변수 사용의 위치에 초점을 맞춰 테스트하는 기법

#**기조루흐**

2) 블랙박스(Black Box Test) __ 20 년 1, 2, 3 회 기출문제

- 모듈 안에서 어떤 일(작동)이 일어나는지 알 수 없음
- 소프트웨어가 수행할 특정 기능을 알기 위해 각 기능이 완전히 작동되는 것을 입증하는 테스트로 기능 테스트라고도 함
- 소프트웨어 인터페이스에서 실시되는 테스트

종류	설명
동치 분할 검사 (Equivalence Partitioning Testing)	프로그램의 입력 조건에 타당한 입력 자료와 타당하지 않은 입력 자료의 개수를 균등하게 해 테스트 케이스를 정하고, 해당 입력 자료에 맞는 결과가 출력되는지 확인하는 기법(동등 분할 기법)
경계값 분석 (Boundary Value Analysis)	입력 조건의 중간값보다 경계값에서 오류가 발생될 확률이 높다는 점을 이용해 입력 조건의 경계값을 테스트 케이스로 선정해 검사하는 기법
원인-효과 그래프 검사 (Cause-Effect Graphing Testing)	입력 데이터 간의 관계와 출력에 영향을 미치는 상황을 체계적으로 분석한 다음 효용성이 높은 테스트 케이스를 선정해 검사하는 기법
비교 검사 (Comparison Testing)	여러 버전의 프로그램에 동일한 테스트 자료를 제공해 동일한 결과가 출력되는지 테스트하는 기법
오류 예측 검사 (Error Guessing)	다른 블랙박스 테스트 기법으로 찾아낼 수 없는 오류를 찾아내는 일력의 보충적 검사 기법(데이터 확인 검사)

#**동경원비오**

14 개발 단계에 따른 애플리케이션 테스트 ★

p.232

#**단통시인**

1) 단위 테스트(Unit Test)

- 코딩 직후 최소 단위인 모듈이나 컴포넌트에 초점을 맞춰 테스트 하는 것
- 사용자의 요구사항을 기반으로 한 기능성 테스트를 최우선으로 수행
- 명세 기반 테스트, 구조 기반 테스트 중 주로 구조 기반 테스트를 시행함

2) 통합 테스트(Integration Test) _ 2-87

- 단위 테스트가 완료된 모듈들을 결합하여 하나의 시스템으로 완성시키는 과정에서의 테스트를 의미
- 모듈 간 또는 통합된 컴포넌트 간의 상호 작용 오류 검사

빅뱅 테스트, **상**향식 테스트(클러스터, Cluster/드라이버, Driver), **하**향식 테스트(스텁, Stub)

3) 시스템 테스트(System Test)

- 개발된 소프트웨어가 컴퓨터 시스템에서 완벽하게 수행되는가를 점검하는 테스트
- 실제 사용 환경과 유사하게 만든 테스트 환경에서 테스트 수행해야 함
- 기능적 요구사항(블랙박스 테스트), 비기능적 요구사항(화이트박스 테스트) 구분

4) 인수 테스트(Acceptance Test) ★

- 개발한 소프트웨어가 사용자의 요구사항을 충족하는지에 중점을 두는 테스트

종류	설명
알파 테스트	통제된 환경에서 사용자가 개발자와 함께 확인하면서 행하는 테스트 기법
베타 테스트	통제되지 않은 환경에서 여러 명의 사용자가 행하는 테스트 기법 (게임 베타 테스터)

15 통합 테스트 ★★

p.235, 2-87

1) 상향식 통합 테스트(Bottom Up Integration Test)

- 프로그램의 하위 모듈에서 상위 모듈 방향으로 통합하면서 테스트하는 기법
- 하나의 주요 제어 모듈과 관련된 종속 모듈의 그룹인 **클러스터(Cluster)** 필요

▶ 하위 모듈들을 클러스터(Cluster)로 결합 → 더미 모듈인 **드라이버(Driver)** 작성 → 통합된 클러스터 단위로 테스트 → 테스트 완료 후 클러스터는 프로그램 구조의 상위로 이동해 결합하고 드라이버는 실제 모듈로 대체됨 ★

2) 하향식 통합 테스트(Top Down Integration Test) __ 20년 1, 2, 3회 기출문제

- 프로그램의 상위 모듈에서 하위 모듈 방향으로 통합하면서 테스트하는 기법
- 깊이 우선 통합법, 넓이 우선 통합법 사용
- 테스트 초기부터 사용자에게 시스템 구조를 보여줄 수 있음
- 상위 모듈에서는 테스트 케이스 사용하기 어려움

▶ 주요 제어 모듈은 작성된 프로그램을 사용, 주요 제어 모듈의 종속 모듈은 스텁(Stub)으로 대체 → 깊이 우선 또는 넓이 우선 등의 통합 방식에 따라 하위 모듈인 스텁(Stub)들이 한 번에 하나씩 실제 모듈로 교체됨 → 모듈이 통합될 때마다 테스트 실시 → 새로운 오류가 발생하지 않음을 보증하기 위해 회귀 테스트 실시

스텁(Stub) ★

3) 혼합식 통합 테스트

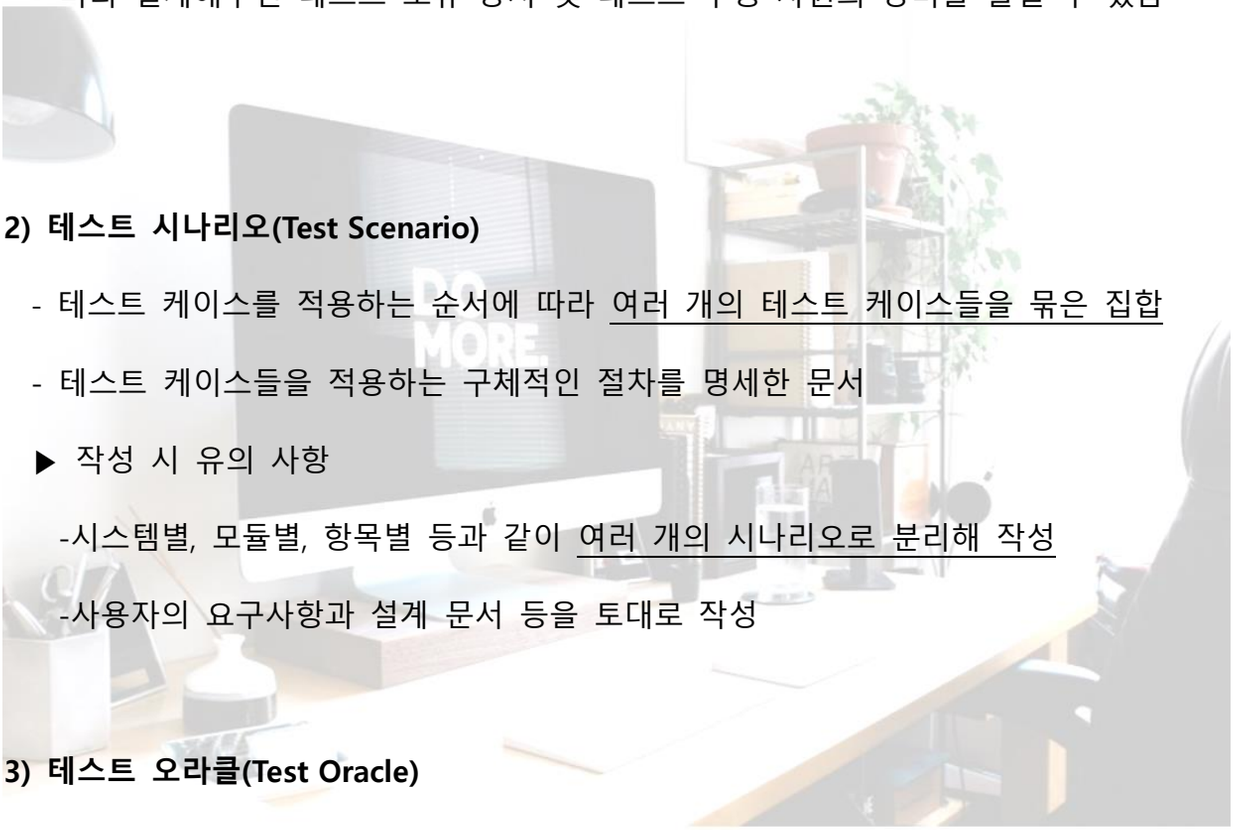
- 하위 수준에서는 상향식 통합, 상위 수준에서는 하향식 통합을 사용해 최적의 테스트를 지원하는 방식
- **샌드위치(Sandwich)**식 통합 테스트 방법

16 테스트 케이스 | 테스트 시나리오 | 테스트 오라클 | 테스트 하네스 ★★

p.242~248, 2-75

1) 테스트 케이스(Test Case)

- 구현된 소프트웨어가 사용자의 요구사항을 정확하게 준수했는지를 확인하기 위해 설계된 입력 값, 실행 조건, 기대 결과 등으로 구성된 테스트 항목에 대한 명세서
- 명세 기반 테스트(블랙박스 테스트)의 설계 산출물에 해당
- 미리 설계해두면 테스트 오류 방지 및 테스트 수행 자원의 낭비를 줄일 수 있음



2) 테스트 시나리오(Test Scenario)

- 테스트 케이스를 적용하는 순서에 따라 여러 개의 테스트 케이스들을 묶은 집합
- 테스트 케이스들을 적용하는 구체적인 절차를 명세한 문서

▶ 작성 시 유의 사항

-시스템별, 모듈별, 항목별 등과 같이 여러 개의 시나리오로 분리해 작성

-사용자의 요구사항과 설계 문서 등을 토대로 작성

3) 테스트 오라클(Test Oracle)

- 테스트 결과가 올바른지 판단하기 위해 사전에 정의된 참 값을 대입해 비교하는 활동

▶ 특징

-제한된 검증: 모든 테스트 케이스에 적용할 수 없음

-수학적 기법: 값을 수학적 기법을 이용해 구할 수 있음

-자동화 기능: 프로그램 실행, 결과 비교, 커버리지 측정 등을 자동화할 수 있음

종류	설명
참(True) 오라클	모든 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공하는 오라클, 발생된 모든 오류를 검출할 수 있음
샘플링(Sampling) 오라클	특정한 몇몇 테스트 케이스의 입력 값들에 대해서만 기대하는 결과를 제공하는 오라클
휴리스틱(Heuristic, 추정) 오라클	샘플링 오라클을 개선한 오라클, 특정 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공하고 나머지 입력 값들에 대해서는 추정으로 처리하는 오라클
일관성(Consistent) 검사 오라클	변경이 있을 때 테스트 케이스의 수행 전과 후의 결과 값이 동일한지를 확인하는 오라클

#**참샘휴일**

4) 테스트 하네스(Test Harness) ★ __ 2-86

구성 요소	설명
테스트 **드**라이버 (Test Driver)	테스트 대상의 하위 모듈을 호출하고 모듈 테스트 수행 후의 결과를 도출하는 도구
테스트 **스**텁 (Test Stub)	테스트 대상의 상위 모듈을 대신하는, 제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구
테스트 **슈**트 (Test Suites)	테스트 대상 컴포넌트나 모듈 등 시스템에 사용되는 테스트 케이스의 집합
테스트 **케**이스 (Test Case)	사용자의 요구사항을 정확하게 준수했는지 확인하기 위한 입력 값, 실행 조건, 기대 결과 등으로 만들어진 테스트 항목 명세서
테스트 **스**크립트 (Test Script)	자동화된 테스트 실행 절차에 대한 명세서
목 오브젝트 (Mock Object)	사전에 사용자의 행위를 조건부로 입력해 두면, 그 상황에 맞는 예정된 행위를 수행하는 객체

#**드스슈케스목**

17 결함 관리 ★

p.250, 2-81

1) 결함 상태 추적

종류	설명
결함 **분**포	모듈 또는 컴포넌트의 특정 속성에 해당하는 결함 수 측정
결함 **추**세	테스트 진행 시간에 따른 결함 수의 추이 분석 ex) 4 시간 동안 5 개 발견
결함 **에**이징 (Fault Aging)	특정 결함 상태로 지속되는 시간 측정 ex) 1 개의 결함이 30 분 동안 지속됨

#분추에

2) 결함 추적 순서

순서	설명
1. 결함 등록 (Open)	테스터와 품질 관리 담당자에 의해 발견된 결함이 등록된 상태
2. 결함 검토 (Reviewed)	등록된 결함을 테스터, 품질 관리 담당자, 프로그램 리더, 담당 모듈 개발자에 의해 검토된 상태
3. 결함 할당 (Assigned)	결함을 수정하기 위해 개발자와 문제 해결 담당자에게 결함이 할당된 상태
4. 결함 수정 (Resolved)	개발자가 결함 수정을 완료한 상태
5. 결함 조치 보류 (Deferred)	결함의 수정이 불가능해 연기된 상태
6. 결함 종료 (Closed)	결함이 해결되어 테스터와 품질 관리 담당자가 종료를 승인한 상태
7. 결함 해제 (Clarified)	종료 승인한 결함을 검토하여 결함이 아니라고 판명한 상태

3) 결함 심각도, 결함 우선순위

- 결함 심각도: 치명적(Critical) > 주요(Major) > 보통(Normal) > 경미(Minor) > 단순(Simple)
- 결함 우선순위: 치명적(Critical) > 높음(High) > 보통(Medium) > 낮음(Low)

18 애플리케이션 성능 분석 ★★

p.254~257

1) 애플리케이션 성능 ★★ __ 20 년 1, 2 회 기출문제

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

#**처응경자**

2) 애플리케이션 성능 저하 원인 분석

- DB 에 필요 이상의 많은 데이터를 요청한 경우
- 커넥션 풀(Connection Pool)의 크기를 너무 작거나 크게 설정한 경우
- JDBC 나 ODBC 같은 미들웨어를 사용한 후 종료하지 않아 연결 누수가 발생한 경우
- 대량의 파일을 업로드하거나 다운로드해 처리 시간이 길어진 경우

3) 소스 코드 최적화

- 클린 코드(Clean Code) 작성 원칙

#**가단의중추** - **가**독성, **단**순성, **의**존성 배제, **중**복성 최소화, **추**상화

4) 소스 코드 품질분석 도구의 종류 ★ __ 20 년 1, 2 회 기출문제

- 정적 분석 도구: **p**md, **c**ppcheck, **c**heckstyle, **S**onarQube, **c**cm, **c**obertuna
- 동적 분석 도구: **A**valanche, **V**algrind

19 모듈 연계 ★★

p.267, 2-103

1) EAI(Enterprise Application Integration) _ 20 년 1, 2 회 기출문제

- 기업 내 각종 애플리케이션 및 플랫폼 간의 정보 전달, 연계, 통합 등 상호 연동이 가능하게 해주는 솔루션

유형	기능
포인트 투 포인트 (Point to Point)	점 대 점으로 연결하는 방식, 변경 및 재사용이 어려움
허브 앤 스포크 (Hub & Spoke)	단일 접점인 허브(Hub) 시스템을 통해 데이터를 전송하는 중앙 집중형 방식, 확장 및 유지보수가 용이하지만 허브 장애 발생 시 시스템 전체에 영향을 미침
메시지 버스 (Message Bus, ESB 방식)	애플리케이션 사이에 미들웨어를 둬 처리하는 방식, 확장성이 뛰어나며 대용량 처리가 가능
하이브리드 (Hybrid)	Hub & Spoke(그룹 내)와 Message Bus(그룹 간)의 혼합 방식, 데이터 병목 현상을 최소화할 수 있음

#**포허메하** ★

2) ESB(Enterprise Service Bus)

- 애플리케이션 간 연계, 데이터 변환, 웹 서비스 지원 등 표준 기반의 인터페이스를 제공하는 솔루션
- 애플리케이션 통합 측면에서 EAI 와 유사하지만 애플리케이션 보다는 서비스 중심의 통합을 지향
- 결합도(Coupling)를 약하게(Loosely) 유지함
- 관리 및 보안 유지가 쉽고, 높은 수준의 품질 지원이 가능

20 인터페이스 구현 | 인터페이스 보안 ★★

p.276~281, 2-98, 2-112

1) 데이터 통신을 이용한 인터페이스 구현 _ 20 년 1, 2 회 기출문제

- 애플리케이션 영역에서 인터페이스 형식에 맞춘 데이터 포맷을 인터페이스 대상으로 전송하고 이를 수신 측에서 파싱(Parsing)해 해석하는 방식
- 주로 JSON 이나 XML 형식의 데이터 포맷을 사용해 인터페이스를 구현

* **JSON**(JavaScript Object Notation): 속성-값 쌍(Attribut-Value Pairs)으로 이뤄진 데이터 객체를 전달하기 위해 사람이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷 ★
* **XML**(eXtensible Markup Language): 특수한 목적을 갖는 마크업 언어를 만드는 데 사용되는 다목적 마크업 언어, 웹 페이지의 기본 형식인 HTML 의 문법이 각 웹 브라우저에서 상호 호환적이지 못하다는 문제와 SGML(Stand Generalized Markup Language)의 복잡함을 해결하기 위해 개발됨 ★

2) 인터페이스 엔터티를 이용한 인터페이스 구현

- 인터페이스가 필요한 시스템 사이에 별도의 인터페이스 엔터티로 상호 연계하는 방식
- 일반적으로 인터페이스 테이블을 엔터티로 활용
- 송, 수신 인터페이스 테이블의 구조는 상황에 따라 서로 다르게 설계할 수도 있음

3) 인터페이스 보안 기능 적용 _ 2-108

- 네트워크(**N**etwork), 애플리케이션(**A**pplication), 데이터베이스(**D**atabase) 영역

* **스니핑**(Sniffing): 네트워크의 중간에서 남의 패킷 정보를 도청하는 해킹 유형

* 소프트웨어 개발 보안(**시큐어 코딩**, Secure Coding): 소프트웨어 개발 과정에서 지켜야 할 일련의 보안 활동 ★

  ex) **입**력 데이터 검증 표현, **보**안 기능, **시**간 및 상태, **에**러 처리, **코**드 오류, **캡**슐화, **A**PI 오용

  #**입보시 에코캡아**

21 인터페이스 구현 검증 | 인터페이스 오류 확인 ★★

p.285~288, 2-119

1) 인터페이스 구현 검증 도구 ★ _ 20 년 1, 2 회 기출문제

도구	기능
xUnit	Java(Junit), C++(Cppunit), .Net(Nunit) 등 다양한 언어를 지원하는 단위 테스트 프레임워크
STAF	서비스 호출 및 컴포넌트 재사용 등 다양한 환경을 지원하는 테스트 프레임워크 ★
FitNesse	웹 기반 테스트케이스 설계, 실행, 결과 확인 등을 지원하는 테스트 프레임워크
NTAF	STAF 의 장점인 재사용 및 확장성과 FitNesse 의 장점인 협업 기능을 통합한 NHN(Naver)의 테스트 자동화 프레임워크
Selenium	다양한 브라우저 및 개발 언어를 지원하는 웹 애플리케이션 테스트 프레임워크
watir	Ruby 언어를 사용하는 애플리케이션 테스트 프레임워크

#**엑스피 엔셀와**

2) 인터페이스 오류 발생 즉시 확인

- 오류 메시지 **알람** 표시, 오류 **SMS** 발송, 오류 내역 **이메일** 발송

3) 인터페이스 오류 발생 주기적인 확인

오류 확인 방법	특징
인터페이스 오류 **로그** 확인	오류를 별도의 로그파일로 생성해 보관함, 자세한 오류 원인 및 내역을 확인할 수 있음
인터페이스 오류 **테이블** 확인	오류사항의 확인이 쉬워 관리가 용이함, 오류사항이 구체적이지 않아 별도의 분석이 필요
인터페이스 **감시(APM) 도구** 사용	스카우터(Scouter)나 제니퍼(Jennifer) 등의 인터페이스 감시 도구를 사용해 주기적 확인

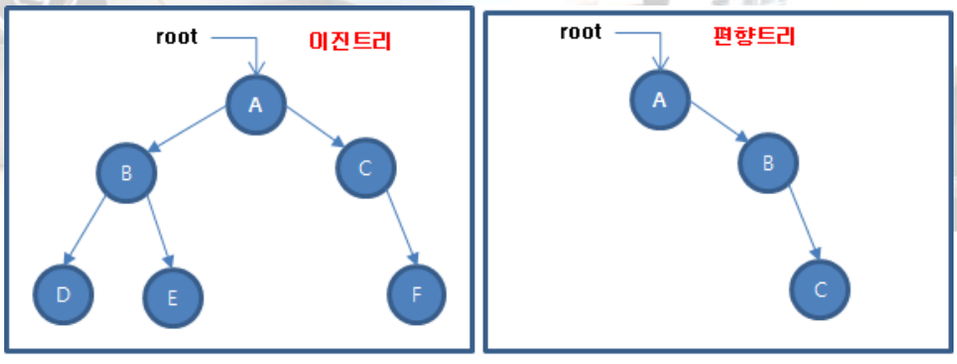
22 추가 정리, 수제비 및 기출문제 ★★★

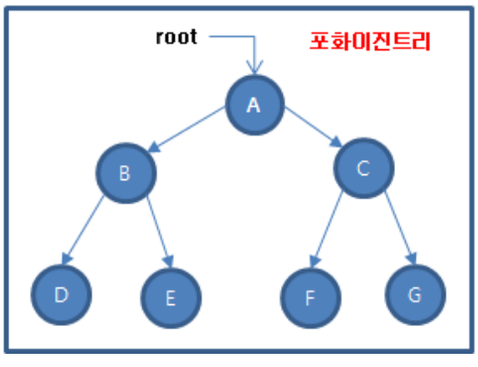
1) 트리 순회방법 ★ __ 2-6, 20 년 1, 2, 3 회 기출문제

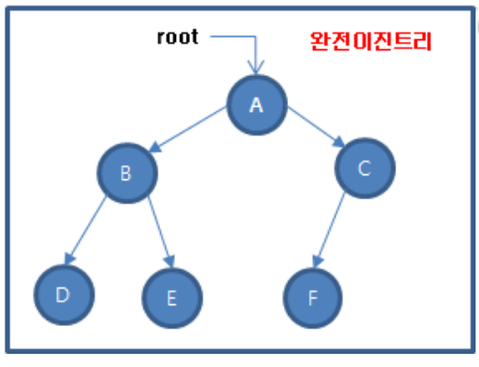
- **전**위 순회(Pre-Order Traversal): Root → Left → Right
- **중**위 순회(In-Order Traversal): Left → Root → Right
- **후**위 순회(Post-Order Traversal): Left → Right → Root

2) 이진 트리 __ 2-6

- 디그리(Degree, 차수)가 2 이하인 노드로 구성돼 자식이 둘 이하로 구성된 트리







3) 논리 데이터 저장소 __ 2-9

구조	설명
개체(Entity)	관리할 대상이 되는 실체
속성(Attribute)	관리할 정보의 구체적 항목
관계(Relationship)	개체 간의 대응 관계

#**개속관**

4) 물리 데이터 저장소 __ 2-13

▶ 논리 데이터 저장소에서 물리 데이터 저장소 모델로 변환하는 절차

단위 개체를 테이블로 변환 → 속성을 컬럼으로 변환 → UID(Unique Identifier)를 기본 키(Primary Key)로 변환 → 관계를 외래 키(Foreign Key)로 변환 → 컬럼 유형(Type)과 길이(Length) 정의 → 반정규화(De-normalization) 수행

5) 인덱스(Index) __ 2-15

- 분포도(Selectivity) 10~15% 이내

▶ 인덱스 컬럼 선정

-수정이 빈번하지 않는 "컬럼"

-ORDER BY, GROUP BY, UNION 이 빈번한 "컬럼"

-분포도가 좋은 컬럼은 단독 인덱스로 생성

-인덱스들이 자주 조합되어 사용되는 컬럼은 결합 인덱스로 생성

▶ 설계 시 고려사항

-지나치게 많은 인덱스는 오버헤드(Overhead) 발생

-인덱스만의 추가적인 저장 공간이 필요

-넓은 범위 인덱스 처리 시 오히려 전체 처리보다 많은 오버헤드를 발생시킴

6) 뷰(View) __ 2-16

- 기본 테이블로부터 유도된, 이름을 가지는 가상 테이블로 기본 테이블과 같은 형태의 구조를 사용하며, 조작도 기본 테이블과 거의 같음
- 가상 테이블이기 때문에 물리적으로 구현되어 있지 않지만 사용자에게 있는 것처럼 간주됨
- 데이터의 논리적 독립성을 제공할 수 있음
- 정의된 뷰로 다른 뷰를 정의할 수 있음
- 뷰가 정의된 기본 테이블이나 뷰를 삭제하면 그 테이블이나 뷰를 기초로 정의된 다른 뷰도 자동으로 삭제됨

속성	설명
REPLACE	뷰가 이미 존재하는 경우 재생성
FORCE	본 테이블의 존재 여부에 관계 없이 뷰 생성
NOFORCE	기본 테이블이 존재할 때만 뷰 생성
WITH CHECK OPTION	서브 쿼리 내의 조건을 만족하는 행만 변경
WITH READ ONLY	데이터 조작어(DML) 작업 불가

▶ 장점

-논리적 데이터 독립성 제공

-접근 제어를 통한 자동 보안 제공

▶ 단점

-독립적인 인덱스를 가질 수 없음

-뷰의 정의를 ALTER 로 변경할 수 없음 → DROP 하고 새로 CREATE 해야 함

-뷰로 구성된 내용에 대한 삽입, 삭제, 갱신, 연산에 제약이 따름

7) 클러스터(Cluster) __ 2-16

- 인덱스의 단점을 해결한 기법 → 분포도(Selectivity)가 넓을수록 오히려 유리함
- 분포도가 넓은 "테이블"의 클러스터링은 저장 공간의 절약이 가능
- 대량의 범위를 자주 액세스(조회)하는 경우 적용
- 인덱스를 사용한 처리 부담이 되는 넓은 분포도에 활용

▶ 클러스터 테이블 선정

-수정이 빈번하지 않는 "테이블"

-ORDER BY, GROUP BY, UNION 이 빈번한 "테이블"

-처리 범위가 넓어 문제가 발생하는 경우 단일 테이블 클러스터링

-조인이 많아 문제가 발생되는 경우는 다중 테이블 클러스터링

▶ 설계 시 고려사항

-조회 속도를 향상시켜주지만 입력, 수정, 삭제 시 성능이 저하됨(부하가 증가)

8) 파티션(Partition) __ 2-17, 20 년 3 회 기출문제

종류	설명
레인지 파티셔닝 (Range Partitioning, **범위분할**)	지정한 열의 값을 기준으로 분할 ex) 일별, 월별, 분기별 등
해시 파티셔닝 (Hash Partitioning, **해시분할**)	해시 함수에 따라 데이터 분할
리스트 파티셔닝 (List Partitioning)	미리 정해진 그룹핑 기준에 따라 분할
컴포지트 파티셔닝 (Composite Partitioning, **조합분할**)	범위분할 이후 해시 함수를 적용 ex) 범위분할 + 해시분할

#**레해리컴**

▶ 파티션의 장점

-**성**능 향상, **가**용성 향상, **백**업 가능, 경**합** 감소

#**성가백합**

9) PL/SQL __ 2-22

구성	설명
선언부 (Declare)	실행부에서 참조할 모든 변수, 상수, CURSOR, EXCEPTION 선언
실행부 (Begin/End)	BEGIN 과 END 사이에 기술되는 영역, 데이터를 처리할 SQL 문과 PL/SQL 블록을 기술
예외부 (Exception)	실행부에서 에러가 발생했을 때 문장 기술

#**선실예**

▶ 장점: **컴**파일 불필요, **모**듈화 기능, **절**차적 언어 사용, **에**러 처리

#**컴모절에**

▶ PL/SQL 을 활용한 저장형 객체 활용

-저장된 **프**로시저, 저장된 **함**수, 저장된 **패**키지, **트**리거(Trigger)

#**프함패트**

10) 단위 모듈 구현의 원리 __ 2-32

원리	설명
정보 은닉 (Information Hiding)	어렵거나 변경 가능성이 있는 모듈을 타 모듈로부터 은폐
분할과 정복 (Divide & Conquer)	복잡한 문제를 분해, 모듈 단위로 문제 해결
데이터 **추**상화 (Data Abstraction)	각 모듈 자료 구조를 액세스하고 수정하는 함수내에 자료 구조의 표현 내역을 은폐
모듈 **독**립성 (Module Inpendency)	낮은 결합도와 높은 응집도

#**정분추독**

11) 알고리즘 설계 기법 ★ __ 2-92, 20 년 3 회 기출문제

기법	설명
분할과 정복 (Divide and Conquer)	문제를 나눌 수 없을 때까지 나누고, 각각을 풀면서 다시 병합해 문제의 답을 얻는 알고리즘
동적계획법 (Dynamic Programming)	어떤 문제를 풀기 위해 그 문제를 더 작은 문제의 연장선으로 생각하고, 과거에 구한 해를 활용하는 방식의 알고리즘
탐욕법 (Greedy)	결정을 해야 할 때마다 그 순간에 가장 좋다고 생각되는 것을 해답으로 선택하는 알고리즘
백트래킹 (Backtracking)	어떤 노드의 유망성 점검 후, 유망하지 않으면 그 노드의 부모 노드로 되돌아간 후 다른 자손 노드를 검색하는 알고리즘

#**분동탐백**

12) 시간 복잡도에 다른 알고리즘 ★★ __ 2-93, 20 년 1, 2 회 기출문제

복잡도	설명	대표 알고리즘
O(1)	상수형 복잡도 자료 크기 무관하게 항상 일정한 속도로 작동 ★	해시 함수 (Hash Function)
O(logN)	로그형 복잡도 문제를 해결하기 위한 단계의 수가 $\log_2 N$ 번만큼의 수행 시간을 가짐	이진 탐색 (Binary Search)
O(n)	선형 복잡도 입력 자료를 차례로 하나씩 모두 처리 수행 시간이 자료 크기와 직접적 관계로 변함 (정비례)	순차 탐색 (Sequential Search)
O(N logN)	선형 로그형 복잡도 문제를 해결하기 위한 단계의 수가 $N\log_2 N$ 번 만큼 수행 시간을 가짐	퀵 정렬 합병 정렬 ★
O(N^2)	제곱형 주요 처리 루프 구조가 2 중인 경우 N 의 크기가 작을 땐 N^2 이 $N\log_2 N$ 보다 느릴 수 있음	선택 정렬 버블 정렬 삽입 정렬 ★

13) SW 품질 측정을 위해 개발자 관점에서 고려해야 할 항목 _ 20 년 1, 2 회 기출문제

- **정확성, 무결성, 사용성** (O) / 간결성 (X)

14) 인터페이스 보안을 위해 네트워크 영역에 적용되는 솔루션 _ 1, 2, 3 회 기출문제

- IPSec, SSL, S-HTTP

15) 외계인코드(Alien Code) _ 개정 전 기출문제, 20 년 1, 2 회 기출문제

- 아주 오래되거나 참고문서 또는 개발자가 없어 유지보수 작업이 어려운 프로그램

16) IPC(Inter-Process Communication) _ p.181

- 모듈 간 통신 방식을 구현하기 위해 사용되는 대표적인 프로그래밍 인터페이스 집합으로, 복수의 프로세스를 수행하며 이뤄지는 프로세스 간 통신까지 구현 가능

대표적인 메소드	특징
Shared Memory	- 다수의 프로세스가 공유 가능한 메모리를 구성하여 프로세스 간 통신 수행
Socket	- 네트워크 소켓을 이용해 네트워크를 경유하는 프로세스들 간 통신 수행
Semaphores	- 공유 자원에 대한 접근제어를 통해 프로세스 간 통신 수행
Pipes & named Pipes	- 'Pipe'라고 불리는 FIFO 형태로 구성된 메모리를 여러 프로세스가 공유하여 통신 수행 ▶ 하나의 프로세스가 Pipe 를 이용 중이라면 다른 프로세스는 접근할 수 없음
Message Queueing	- 메시지가 발생하면 이를 전달하는 형태로 프로세스 간 통신 수행

#SSS PM

17) 정렬 알고리즘 ★ __ 개정 전 기출문제, 20 년 3 회 기출문제

37, 14, 17, 40 ,35

▶ 선택 정렬, PASS 3 (3 회전)

- **PASS 1** ▶ **37, 14**, 17, 40, 35 → **14, 37**, 17, 40, 35

  1 번째 37 를 제외한 14, 17, 40, 35 중 가장 작은 수(14)와 37 을 선택해서 바꿈

- **PASS 2** ▶ 14, **37, 17**, 40, 35 → 14, **17, 37**, 40, 35

  1, 2 번째 14, 37 를 제외한 17, 40, 35 중 가장 작은 수(17)와 37 을 선택해서 바꿈

- **PASS 3** ▶ 14, 17, **37**, 40, **35** → 14, 17, **35**, 40, **37** (정답)

  1, 2, 3 번째 14, 17, 37 를 제외한 40, 35 중 가장 작은 수(35)와 37 을 선택해서 바꿈

▶ 버블 정렬, PASS 1 (1 회전) __ 보통 실기 코딩문제로 나옴

- **PASS 1** ▶ **37, 14**, 17, 40, 35 → **14, 37**, 17, 40, 35

  1 번째 37 와 2 번째 14 를 비교해서 1 번째(37) > 2 번째(14)가 참이라면 바꿈

  ▶ 14, **37, 17**, 40, 35 → 14, **17, 37**, 40, 35 (정답)

  2 번째 37 와 3 번째 17 를 비교해서 2 번째(37) > 3 번째(17)가 참이라면 바꿈

  3 번째 37 와 4 번째 40 을 비교해서 3 번째(37) > 4 번째(40)가 거짓이므로 종료(끝)

▶ 삽입 정렬, PASS 4 (4 회전)

- **PASS 1** ▶ 37, **14**, 17, 40, 35 → **14**, 37, 17, 40, 35

  2 번째 14 를 앞의 1 번째 37 과 비교해서 수가 더 작다면 바꿔 삽입함

- **PASS 2** ▶ 14, 37, **17**, 40, 35 → 14, **17**, 37, 40, 35

  3 번째 17 을 앞의 2 번째 37, 1 번째 14 와 비교해서 수가 더 작다면 바꿔 삽입함

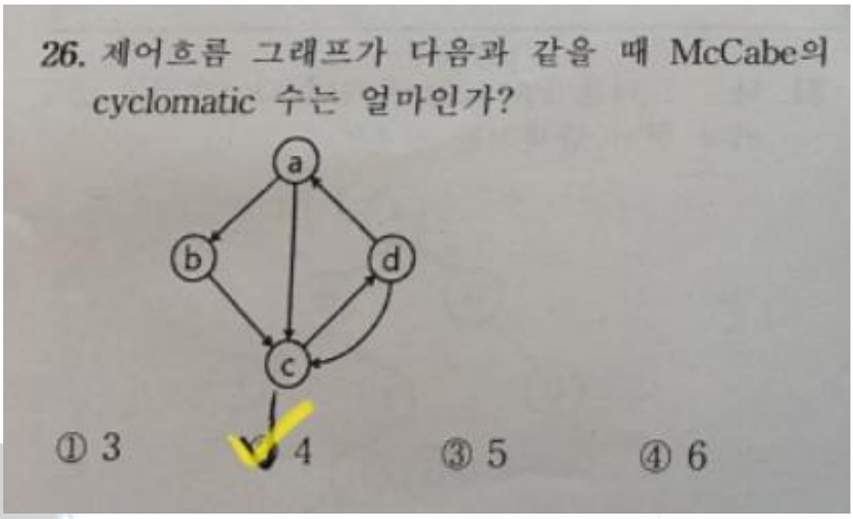
- **PASS 3** ▶ 14, 17, 37, **40**, 35

  4 번째 40 을 앞의 3 번째 37, 2 번째 17, 1 번째 14 비교해서 수가 더 작다면 바꿔 삽입함

- **PASS 4** ▶ 14, 17, 37, 40, **35** → 14, 17, **35**, 37, 40 (정답)

  5 번째 35 를 앞의 4 번째 40, 3 번째 37, 2 번째 17, 1 번째 14 와 비교해서 수가 더 작다면 바꿔 삽입함

18) McCabe 의 cyclomatic 수 __ 개정 전 기출문제, 20 년 3 회 기출문제



▶ **Edge(선) – Node(점) + 2**

→ 6 – 4(a, b, c, d) + 2 = **4**

19) 소프트웨어 재공학이 소프트웨어 재개발에 비해 갖는 장점 __ 20 년 3 회 기출문제

- 위험부담 감소, 비용 절감, 시스템 명세의 오류억제, 개발시간의 감소

20) 소프트웨어 품질 목표 __ 20 년 3 회 기출문제

구 분	품질 표준	의 미
소프트웨어 운영 특성	정확성(Correctness)	사용자의 요구 기능을 충족시키는 정도
	신뢰성(Reliability)	정확하고 일관된 결과를 얻기 위해 요구된 기능을 오류 없이 수행하는 정도
	효율성(Effeciency)	요구되는 기능을 수행하기 위한 필요한 자원의 소요 정도로, 소프트웨어가 자원을 쓸데없이 낭비하지 않아야 함
	무결성(Integrity)	허용되지 않는 사용이나 자료의 변경을 제어하는 정도
	사용 용이성(Usability)	사용에 필요한 노력을 최소화하고 쉽게 사용할 수 있는 정도로, 소프트웨어는 적절한 사용자 인터페이스와 문서를 가지고 있어야 함
소프트웨어 변경 수용 능력	유지보수성 (Maintainability)	변경 및 오류 사항의 교정에 대한 노력을 최소화하는 정도로, 사용자의 기능 변경의 필요성을 만족하기 위하여 소프트웨어를 진화하는 것이 가능해야 함
	유연성(Flexibility)	소프트웨어를 얼마만큼 쉽게 수정할 수 있는가 하는 정도
	시험 역량(Testability)	의도된 기능이 수행되도록 보장하기 위해 프로그램을 시험할 수 있는 정도
소프트웨어 적응 능력	이식성(Portability)	다양한 하드웨어 환경에서도 운용 가능하도록 쉽게 수정될 수 있는 정도
	재사용성(Reusability)	전체나 일부 소프트웨어를 다른 목적으로 사용할 수 있는가하는 정도
	상호 운용성 (Interoperability)	다른 소프트웨어와 정보를 교환할 수 있는 정도

21) 소프트웨어 공학의 기본 원칙 _ 20 년 3 회 기출문제

- 품질 높은 소프트웨어 상품 개발

- 지속적인 검증 시행

- 결과에 대한 명확한 기록 유지

22) AJAX(Asynchronous JavaScript and XML) ★★ _ 20 년 3 회 기출문제

- **JavaScript** 를 사용한 **비동기 통신기술**로 **클라이언트와 서버 간**에 **XML 데이터를 주고 받는 기술**

23) 외부 스키마, 내부 스키마, 개념 스키마 _ 개정 전 기출문제, 20 년 4 회 기출문제

- 외부 스키마(External Schema): 사용자의 관점에서 보여주는 데이터베이스 구조로 전체 데이터베이스의 일부이므로 서브 스키마로고도 함

- 내부 스키마(Internal Schema): 저장장치의 입장에서 데이터베이스 전체가 저장되는 방법을 명세한 것으로 단 하나만 존재함

- 개념 스키마(Conceptual Schema): 전체 사용자 또는 모든 응용 시스템이 필요한 데이터베이스 구조로 조직 전체의 데이터베이스로 단 하나만 존재함

24) 해싱함수 _ 20 년 4 회 기출문제

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