

2과목 <소프트웨어 개발>

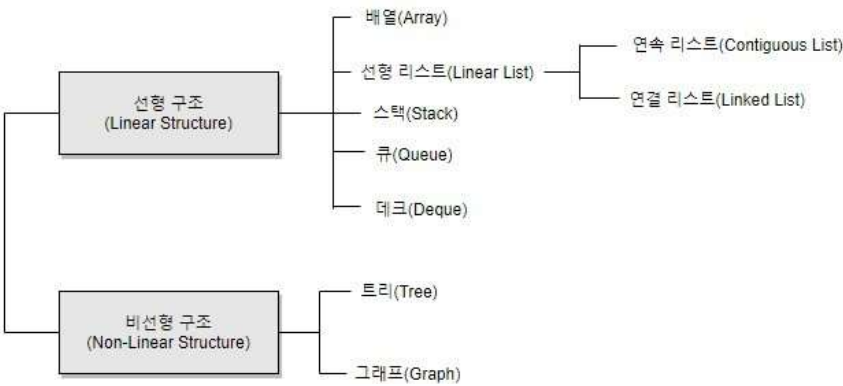
1장 데이터 입·출력 구현

섹션 36 자료 구조

자료구조의 정의 : 효율적인 프로그램을 작성할 때 가장 우선적인 고려사항은 저장 공간의 효율성과 실행시간의 신속성이다. 자료 구조는 프로그램 내에서 사용하기 위한 자료를 기억장치 공간 내에 저장하는 방법과 저장된 그룹 내에 존재하는 자료 간의 관계, 처리 방법 등을 연구 분석하는 것을 말한다.

- 자료 구조는 자료의 표현과 그것과 관련된 연산이다.
- 자료 구조는 일련의 자료들을 조직하고 구조화 하는 것이다.
- 어떠한 자료 구조에서도 필요한 모든 연산들을 처리할 수 있다.
- 자료 구조에 따라 프로그램 실행시간이 달라진다.

자료구조의 분류



1. 배열 : 동일한 자료의 데이터들이 같은 크기로 나열되어 순서를 갖고 있는 집합
- 정적인 자료 구조로 기억장소의 추가가 어렵고, 데이터 삭제 시 데이터가 저장되어 있던 기억장소는 빈공간으로 남아있어 메모리의 낭비가 심함
 - 첨자(Index)를 이용하여 데이터에 접근
 - 반복적인 데이터 처리 작업에 적합한 구조
 - 데이터마다 동일한 이름의 변수를 사용하여 처리가 간편
 - 사용한 첨자의 개수에 따라 n차원 배열이라 함

2. 선형 리스트 (Linear list) : 일정한 순서에 의해 나열된 구조
- 배열을 이용한 연속 리스트와 포인터를 이용하는 연결 리스트로 구분됨

- (1) 연속 리스트 (Contiguous List)
- 배열과 같이 연속되는 기억장소에 저장되는 구조
 - 기억장소를 연속적으로 배정받기 때문에 기억장소 이용 효율은 밀도가 1로서 가장 좋음
 - 중간에 데이터를 삽입하기 위해서는 연속된 빈 공간이 있어야 하며, 삽입·삭제 시에는 자료의 이동이 필요함

- (2) 연결 리스트 (Linked List)
- 자료들을 반드시 연속적으로 배열시키지는 않고 임의의 기억공간에 기억시키되, 자료 항목의 순서에 따라 노드의 포인터 부분을 이용하여 서로 연결시킨 자료 구조.
 - 노드의 삽입·삭제 작업이 용이하다
 - 기억 공간이 연속적이지 않아도 저장할 수 있음
 - 연결을 위한 링크(포인터) 부분이 필요하기 때문에 순차 리스트에 비해 기억 공간의 이용 효율이 좋지 않다.
 - 연결을 위한 포인터를 찾는 시간이 필요하기 때문에 접근 속도가 느림
 - 중간 노드 연결이 끊어지면 그 다음 노드를 찾기 힘들
 - 노드란 다음 노드를 가르키는 포인터이다.

3. 스택 (Stack) : 리스트의 한쪽 끝으로만 자료의 삽입, 삭제 작업이 이루어지는 자료 구조.
- 스택은 후입선출 방식으로 자료를 처리
 - 스택 응용 분야 : 함수 호출 순서 제어, 인터럽트 처리, 수식 계산 및 수식 표현, 컴파일러 이용한 번역, 프로그램 호출 시 복귀 및 서브루틴 복귀 주소 저장 등
 - 스택의 모든 기억 공간이 꽉 채워져 있는 상태에서 데이터가 삽입되면 오버플로우가 발생, 더 이상 삭제할 데이터가 없는 상태에서 삭제하면 언더플로우가 발생
 - ° TOP : 스택으로 할당된 기억 공간에 가장 마지막(위에) 삽입된 자료가 있는 위치
 - ° Bottom : 스택의 가장 마지막
 - ° M : 스택의 크기
 - ° X : 스택의 이름 등
 - 스택은 후입선출이므로 TOP이 스택의 포인터가 된다.

4. 큐 : 리스트의 한쪽에서는 삽입 작업이 이루어지고 다른 한쪽에서는 삭제 작업이 이루어지도록 구성된 자료 구조

- 선입선출 구조

- 큐는 시작과 끝을 표시하는 두 개의 포인터가 있다.

- ° 프런트 (F, Front) 포인터 : 가장 먼저 삽입된 자료의 기억 공간을 가리키는 포인터. 즉 삭제 작업시 가장 먼저 삭제되는 곳

- ° 리어 (R, Rear) 포인터 : 가장 마지막에 삽입된 자료가 위치한 공간을 가리키는 포인터

- 큐는 운영체제 작업 스케줄러에서 사용

5. 데크(Deque) 데크는 큐처럼 비슷하지만 삽입과 삭제가 리스트 양쪽 끝에서 발생할 수 있는 자료 구조

6. 그래프 : 정점 V(vertex)와 간선 E(edge)의 두 집합으로 이루어짐

- 간선의 방향성 유무에 따라 방향 그래프와 무방향 그래프로 구분된다.

- 통신망, 교통망, 이항관계, 연립방정식 무향선분 해법 등에 사용 됨

※ 트리는 사이클이 없는 그래프

섹션 37 트리 (Tree)

트리의 개요 : 트리는 정점(Node)와 선분(Branch)를 이용하여 사이클을 이루지 않도록 구성된 그래프의 특수한 형태

- 하나의 기억공간을 노드라하고, 노드와 노드를 연결하는 선을 링크라 함

- 가족의 계보(족보) , 조직도 등을 표현하기에 적합

- 노드의 종류

1. 근 노드 (Root Node) : 제일 최상위에 있는 노드

2. 디그리(Degree, 차수) : 각 노드에서 뻗어 나온 가지의 수

3. 단말 노드 (Terminal) = 잎 노드 (Leaf) : 자식이 없는 노드. 즉, 디그리가 0인 노드.

4. 자식 노드 : 어떤 한 노드에 연결된 다음 레벨의 노드

5. 부모 노드 : 어떤 한 노드에 연결된 이전 레벨의 노드

6. 형제 노드 : 동일한 레벨에서 동일한 부모를 갖는 노드들

7. 트리의 디그리 : 노드들의 디그리 중에서 가장 많은 수

8. 깊이 : 가장 높은 레벨

트리의 운행법 : 트리를 구성하는 각 노드들을 찾아가는 방법

- 이진 트리를 운행하는 방법은 산술식의 표기법과 연관성을 찾는다.

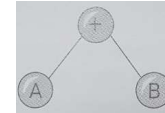
1. Preorder 운행 : Root → left → right 순으로 운행

2. Inorder 운행 : Left → Root → Right 순으로 운행

3. Postorder 운행 : Left → Right → Root 순으로 운행

트리를 또 여러개의 서브트리 단위로 나누어서 순서를 가짐

수식의 표기법 : 산술식을 계산하기 위해 기억공간에 기억시키는 방법으로 이진 트리를 많이 사용한다. 이진 트리로 만들어진 수식을 프리오더, 인오더, 포스트오더로 운행하면 각각 전위, 중위, 후위 표기법이 된다.



전위 표기법(PreFix) : 연산자 → left → Right : +AB

중위 표기법(InFix) : Left → 연산자 → Right : A+B

후위 표기법(PostFix) : Left → Right → 연산자 : AB+

※ Infix 표기를 PostFix나 PreFix로 바꾸기 : Postfix나 Prefix는 스택을 이용하여 처리하므로, Infix는 Postfix나 Prefix로 바꾸어 처리한다.

예) Infix → prefix : $X = A / B * (C + D) + E$

1. 우선순위로 괄호를 묶음 : $(X = (((A / B) * (C + D)) + E))$

2. 각 괄호를 변환 (연산자를 왼쪽으로) 순서대로 하면 $=X+*/AB+CDE$ 가 됨

※infix → postfix로 가면 연산자를 오른쪽으로 하면 됨 : $XAB/CD+*E+=$

거꾸로 prefix나 postfix를 prefix로 변환하기

예) postfix → infix : $ABC-/DEF+*+$

1. 연산이 뒤로간것에서 가운데로 바꾸는 것이므로 각 연산자를 앞 2개씩 묶는다

→ $(A(BC-))/(D(EF+)*)+$: 밖에서부터 천천히 풀면 $A/(B-C)+D*(E+F)$ 가 됨

prefix → infix 는 연산이 앞으로가므로 뒤 2개씩 묶으면 됨

섹션 38 정렬(Sort)

1. 삽입 정렬 : 가장 간단한 정렬 방식으로 이미 순서화된 파일에 새로운 하나의 레코드를 순서에 맞게 삽입시켜 정렬한다

- 두 번째 키와 첫 번째 키를 비교해 순서대로 나열(1회전)하고, 이어서 세 번째 키를 첫 번째, 두 번째 키와 비교해 순서대로 나열(2회전)하고, 계속해서 n 번째 키를 앞의 $n-1$ 개의 키와 비교하여 알맞은 순서에 삽입하여 정렬하는 방식이다
- 평균과 최악 모두 수행 시간 복잡도는 $O(2^n)$ 이다.

2. 쉘 정렬 : 쉘 정렬은 삽입 정렬(Insertion Sort)을 확장한 개념이다.

- 입력 파일을 어떤 매개변수의 값으로 서브파일을 구성하고, 각 서브파일을 Insertion 정렬 방식으로 순서 배열하는 과정을 반복하는 정렬 방식(보통 $h = \sqrt[3]{n}$), 즉 임의의 레코드 키와 h 값만큼 떨어진 곳의 레코드 키를 비교하여 순서화되어 있지 않으면 서로 교환하는 것을 반복하는 정렬 방식이다.
- 입력 파일이 부분적으로 정렬되어 있는 경우에 유리한 방식이다.
- 평균 수행 시간 복잡도는 $O(n^{1.5})$ 이고, 최악의 수행 시간 복잡도는 $O(n^2)$ 이다.

3. 선택 정렬(Selection Sort) : n 개의 레코드 중에서 최소값을 찾아 첫 번째 레코드 위치에 놓고, 나머지 ($n-1$)개 중에서 다시 최소값을 찾아 두 번째 레코드 위치에 놓는 방식을 반복하여 정렬하는 방식

- 평균과 최악 모두 수행 시간 복잡도는 $O(n^2)$ 이다.
- 처음 n 개에서 하나 택해서 맨 앞이랑 바꿈. 그 다음 $n-1$ 개에서 하나 택해서 2 번째꺼라 바꾸는 방식이다.

4. 버블 정렬 (Bubble Sort) : 주어진 파일에서 인접한 두 개의 레코드 키 값을 비교하여 그 크기에따라 레코드 위치를 서로 교환하는 정렬 방식이다.

- 계속 정렬 여부를 플래그비트 (f)로 결정한다.
- 평균과 최악 모두 수행 시간 복잡도는 $O(n^2)$ 이다.
- 1회전에서 1,2를 비교.. 2,3비교.. 3,4 비교...
- 2회전에서도 똑같이 반복함

5. 퀵 정렬 (Quick Sort) : 퀵 정렬은 레코드의 많은 자료 이동을 없애고 하나의 파일을 부분적으로 나누어 가면서 정렬하는 방법으로 키를 기준으로 작은 값을 왼쪽

에, 큰 값은 오른쪽 서브파일로 분해시키는 방식으로 정렬한다.

- 위치에 관계없이 임의의 키를 분할 원소로 사용할 수 있다.
- 정렬 방식 중에서 가장 빠른 방식이다.
- 프로그램에서 되부름을 이용하기 때문에 스택이 필요하다
- 분할(Divide)과 정복(Conquer)을 통해 자료를 정렬한다.
 1. 분할 : 기준값이 피벗을 중심으로 정렬할 자료들을 2개의 부분집합으로 나눔
 2. 정복 : 부분집합의 원소들 중 피벗보다 작은 원소들은 왼쪽, 피벗보다 큰 원소들은 오른쪽 부분집합으로 정렬한다.
- 부분집합의 크기가 더 이상 나누어질 수 없을때까지 분할과 정복을 반복 수행
- 평균 수행 시간 복잡도는 $O(n \log_2 n)$ 이고, 최악의 수행 시간 복잡도는 $O(n^2)$

6. 힙 정렬 (Heap Sort) : 전이진 트리(Complete Binary Tree)를 이용한 정렬 방식이다.

- 구성된 전이진 트리를 Heap Tree로 변환하여 정렬한다.
- 평균과 최악 모두 시간 복잡도는 $O(n \log_2 n)$ 이다.

7. 2-Way 합병 정렬 (Merge Sort) : 이미 정렬되어 있는 두 개의 파일을 한 개의 파일로 합병하는 정렬 방식이다.

- 두 개의 키들을 한 쌍으로 하여, 각 쌍에 대하여 순를 정한다.
- 순서대로 정렬된 각 쌍의 키들을 합병하여 하나의 정렬된 서브리스트로 만든다
- 위 과정에서 정렬된 서브리스트들을 하나의 정렬된 파일이 될 때까지 반복한다.
- 평균과 최악 모두 시간 복잡도는 $O(n \log_2 n)$ 이다.

8. 기수 정렬 (Radix Sort = Bucket Sort) : 큐를 이용하여 자릿수별로 정렬하는 방식이다.

- 레코드의 키 값을 분석하여 같은 수 또는 같은 문자끼리 그 순서에 맞는 버킷에 분배 하였다가 버킷의 순서대로 레코드를 꺼내어 정렬한다.
- 평균과 최악 모두 시간 복잡도는 $O(dn)$ 이다.

섹션 39 검색 - 이분 검색 / 해싱

이분 검색 (이진 검색) : 전체 파일을 두 개의 서브파일로 분리해가면서 Key 레코드를 검색하는 방법

- 이분 검색은 반드시 순서화된 파일이어야 검색할 수 있다.
- 찾고자 하는 Key 값을 파일의 중간 레코드 Key 값과 비교하면서 검색한다.
- 비교 횟수를 거듭할 때마다 검색 대상이 되는 데이터의 수가 절반으로 줄어듦으로 탐색 효율이 좋고 탐색 시간이 적게 소요된다.
- 중간 레코드 번호 $M = (F+L)/2$ (F: 첫 번째 레코드 번호, L : 마지막)

해싱 : 해시 테이블이라는 기억공간을 할당하고, 해시 함수를 이용하여 레코드 키에 대한 해시 테이블 내의 홈 주소를 계산한 후 주어진 레코드를 해당 기억장소에 저장하거나 검색 작업을 수행하는 방식이다.

해시 테이블 : 레코드를 한 개 이상 보관할 수 있는 버킷들로 구성된 기억공간으로, 보조기억장치에 구성할 수도 있고 주기억장치에 구성할 수도 있다.

버킷 (Bucket)	하나의 주소를 갖는 파일의 한 구역 버킷의 크기는 같은 주소에 포함될 수 있는 레코드의 수이다.
슬롯 (Slot)	한 개의 레코드를 저장할 수 있는 공간으로 n개의 슬롯이 모여 하나의 버킷을 구성한다.
Collision (충돌현상)	서로 다른 두 개 이상의 레코드가 같은 주소를 갖는 현상이다.
Synonym	충돌로 인해 같은 Home Address를 갖는 레코드들의 집합
Overflow	계산된 Home Address의 Bucket 내에 저장할 기억공간이 없는 상태로 , 버킷을 구성하는 Slot이 여러개일 때 Collision은 발생하도 Overflow는 발생하지 않을 수 있다.

해싱 함수

제산법 (Division)	레코드의 키(K)를 해시표(Hash Table)의 크기보다 큰 수 중에서 가장 작은 소수(Prime, Q)로 나눈 나머지를 홈 주소로 삼는 방식. 즉 $h(K) = K \bmod Q$
제곱법 (Mid-Square)	레코드 키 값(K)를 제곱한 후 그 중간 부분의 값을 홈 주소로 삼는 방식
폴딩법 (Folding)	레코드 키 값(K)를 여러부분으로 나눈 후 각 부분의 값을 더하거나 XOR한 값을 홈 주소로 삼는 방식
기수 변환법 (Radix)	키 숫자의 진수를 다른 진수로 변환시켜 주소 크기를 초과한 높은 자릿수는 절단하고, 이를 다시 주소 범위에 맞게 조정하는 방법

대수적 코딩법 (Algebraic Coding)	키 값을 이루고 있는 각 자리의 비트 수를 한 다항식의 계수로 간주하고, 이 다항식을 해시표의 크기에 의해 정의된 다항식으로 나누어 얻은 나머지 다항식의 계수를 홈 주소로 삼는 방식
숫자 분석법 (Digit Analysis, 계수 분석법)	키 값을 이루는 숫자의 분포를 분석하여 비교적 고른 자리를 필요한 만큼 택해서 홈주소로 삼는 방식
무작위법 (Random)	난수를 발생시켜 나온 값을 홈 주소로 삼는 방식

섹션 40 데이터베이스 개요

데이터저장소 : 소프트웨어 개발 과정에서 다루어야 할 데이터들을 논리적인 구조로 조직화하거나, 물리적인 공간에 구축한 것을 의미한다.

- 데이터저장소는 논리 데이터저장소와 물리 데이터저장소로 구분된다.
- 논리 데이터저장소는 데이터 및 데이터 간의 연관성, 제약조건을 식별하여 논리적인 구조로 조직화한 것을 말한다.
- 물리 데이터저장소는 논리 데이터저장소에 저장된 데이터와 구조들을 소프트웨어가 운용될 환경의 물리적 특성을 고려하여 하드웨어적인 저장장치에 저장한 것을 의미한다.
- 논리 데이터저장소를 거쳐 물리 데이터저장소를 구축하는 과정은 데이터베이스를 구축하는 과정과 동일하다.

데이터베이스 : 특정 조직의 업무를 수행하는 데 필요한 상호 관련된 데이터들의 모임으로 다음과 같이 정의할 수 있다.

1. 통합된 데이터 (Integrated Data) : 자료의 중복을 배제한 데이터의 모임
2. 저장된 데이터 (Stored Data) : 컴퓨터가 접근할 수 있는 저장 매체에 저장된 자료
3. 운영 데이터 (Operational Data) : 조직의 고유한 업무를 수행하는 데 존재 가치가 확실하고 없어서는 안 될 반드시 필요한 자료
4. 공용 데이터 (Shared Data) : 여러 응용 시스템들이 공동으로 소유하고 유지하는 자료

DBMS (DataBase Management System; 데이터베이스 관리 시스템) : 사용자와 데이터베이스 사이에서 사용자의 요구에 따라 정보를 생성해주고, 데이터베이스를 관리해 주는 소프트웨어

- DBMS는 기존의 파일 시스템이 갖는 데이터의 종속성과 중복성의 문제를 해결하기 위해 제안된 시스템으로, 모든 응용 프로그램들이 데이터베이스를 공유할 수 있도록 관리해 준다.

- DBMS는 데이터베이스의 구성, 접근 방법, 유지관리에 대한 모든 책임을 진다.
- DBMS의 필수 기능에는 정의, 조작, 제어 기능이 있다.

1. 정의 기능 : 모든 응용 프로그램들이 요구하는 데이터 구조를 지원하기 위해 데이터베이스에 저장될 데이터의 형(Type)과 구조에 대한 정의, 이용 방식, 제약 조건등을 명시하는 기능이다.

2. 조작 기능 : 데이터 검색, 갱신, 삽입, 삭제 등을 체계적으로 처리하기 위해 사용자와 데이터베이스 사이의 인터페이스 수단을 제공하는 기능이다.

3. 제어 기능 : 무결성, 권한검사, 병행 제어

- 데이터베이스를 접근하는 갱신, 삽입, 삭제 작업이 정확하게 수행되어 데이터의 무결성이 유지되도록 제어해야 한다.
- 정당한 사용자가 허락된 데이터만 접근할 수 있도록 보안을 유지하고 권한을 검사할 수 있어야 한다.
- 여러 사용자가 데이터베이스를 동시에 접근하여 데이터를 처리할 때 처리 결과가 항상 정확성을 유지하도록 병행 제어(Concurrency Control)을 할 수 있어야함.

DBMS의 장·단점

장점	단점
데이터의 논리적,물리적 독립성이 보장 데이터 중복을 피할 수 있어 기억 공간이 절약 저장된 자료를 공동으로 이용할 수 있음 데이터의 일관성 유지 데이터의 무결성 유지 보안을 유지할 수 있음 데이터 표준화 가능 데이터를 통합하여 관리할 수 있음 항상 최신의 데이터를 유지 데이터의 실시간 처리 가능	데이터베이스의 전문가가 부족 전산화 비용 증가 파일의 예비(BackUp)와 회복이 어려움 시스템이 복잡함 대용량 디스크로의 집중적인 Access로 과부하(Overhead) 발생

데이터의 독립성 : 종속성에 대비되는 말로 DBMS의 궁극적인 목표

1. 논리적 독립성 : 응용 프로그램과 데이터베이스를 독립시킴으로써, 데이터의 논리적 구조를 변경시키더라도 응용 프로그램은 변경되지 않음
2. 물리적 독립성 : 응용 프로그램과 보조기억장치 같은 물리적 장치를 독립시킴으로써, 데이터베이스 시스템의 성능 향상을 위해 새로운 디스크를 도입하더라도 응용 프로그램에는 영향을 주지 않고 데이터의 물리적 구조만을 변경

스키마(Schema) : 데이터베이스의 구조와 제약 조건에 관한 전반적인 명세를 기술한 메타데이터(meta-data)의 집합

- 스키마는 DB를 구성하는 데이터 개체(Entity), 속성(Attribute), 관계(Relationship) 및 데이터 조작 시 데이터 값들이 갖는 제약 조건 등에 관해 전반적용 정의함.
- 스키마는 사용자의 관점에 따라 외부, 개념, 내부로 나뉜다.

외부 스키마	사용자나 응용 프로그래머가 각 개인의 입장에서 필요로 하는 데이터베이스의 논리적 구조를 정의
개념 스키마	데이터베이스의 전체적인 논리적 구조로서, 모든 응용 프로그램이나 사용자들이 필요로 하는 데이터를 종합한 조직 전체의 데이터베이스로, 하나만 존재 개체 간의 관계와 제약 조건을 나타내고, 데이터베이스의 접근 권한, 보안 및 무결성 규칙에 관한 명세를 정의한다.
내부 스키마	물리적 저장장치의 입장에서 본 데이터베이스 구조로서, 실제로 DB에 저장될 레코드의 형식을 정의하고 저장 데이터 항목의 표현 방법, 내부 레코드의 물리적 순서등을 나타낸다.

섹션 41 데이터의 입출력 (C등급)

데이터 입·출력의 개요 : 소프트웨어의 기능 구현을 위해 데이터베이스에 데이터를 입력하거나 데이터베이스의 데이터를 출력하는 작업을 의미한다.

- 데이터 입·출력은 단순 입력과 출력뿐만 아니라 데이터를 조작하는 모든 행위를 의미하며, 이와 같은 작업을 위해 SQL(Structured Query Language)를 사용
- 데이터 입·출력을 소프트웨어에 구현하기 위해 개발 코드 내에 SQL 코드르 삽입하거나, 객체와 데이터를 연결하는 것을 데이터 접속(Data Mapping)이라 한다.
- SQL을 통한 데이터베이스의 조작을 수행할 때 하나의 논리적 기능을 수행하기 위한 작업의 단위 또는 한꺼번에 모두 수행되어야 할 일련의 연산을 트랜잭션(Transaction)이라 한다.

SQL(Structured Query Language) : SQL은 1974년 IBM연구소에서 개발한 SEQUEL에서 유래. 국제표준 데이터베이스 언어로, 많은 회사에서 관계형 데이터베이스(RDB)를 지원하는 언어로 채택하고 있다.

- 관계대수와 관계해석을 기초로 한 혼합 데이터 언어이다.
- 질의어지만 질의 기능만 있는 것이 아니라 데이터 구조의 정의, 데이터 조작, 데이터 제어 기능을 모두 갖추고 있음
- SQL은 데이터 정의어(DDL), 데이터 조작어(DML), 데이터 제어어(DCL)로 구분된다.

1. 데이터 정의어 (DDL; Data Define language) : SCHEMA, DOMAIN, TABLE, VIEW, INDEX를 정의하거나 변경 또는 삭제할 때 사용하는 언어

2. 데이터 조작어 (DML; Data Manipulation language) : 데이터베이스 사용자가 응용 프로그램이나 질의어를 통해서 저장된 데이터를 실질적으로 처리하는 데 사용되는 언어이다.

3. 데이터 제어어 (DCL; Data Control Language) : 데이터의 보안, 무결성, 회복, 병행 수행 제어 등을 정의하는데 사용되는 언어이다.

데이터 접속 (Data Mapping) : 소프트웨어의 기능 구현을 위해 프로그래밍 코드와 데이터베이스의 데이터를 연결(Mapping)하는 것을 말하며, 관련 기술로 SQL Mapping과 ORM이 있다.

- SQL Mapping : 프로그래밍 코드 내에 SQL을 직접 입력하여 DBMS의 데이터에 접근하는 기술로, 관련 프레임워크에는 JDBC, ODBC, MyBatis 등이 있다.
- ORM(Object-Relational Mapping) : 객체지향 프로그래밍의 객체와 관계형 데이터베이스의 데이터를 연결하는 기술로, 관련 프레임워크에는 JPA, Hibernate, Django 등이 있다.

트랜잭션 (Transaction) : 데이터베이스의 상태를 변환시키는 하나의 논리적 기능을 수행하기 위한 작업의 단위 또는 한꺼번에 모두 수행되어야 할 일련의 연산들을 의미한다.

-트랜잭션을 제어하기 위해서 사용하는 명령어를 TCL(Transaction Control Language)라고 하며, TCL의 종류에는 COMMIT, ROLLBACK, SAVEPOINT가 있다.

1. COMMIT : 트랜잭션 처리가 정상적으로 종료되어 트랜잭션이 수행한 변경 내용을 데이터베이스에 반영하는 명령어

2. ROLLBACK : 하나의 트랜잭션 처리가 비정상적으로 종료되어 데이터베이스의 일관성이 깨졌을 때 트랜잭션이 행한 모든 변경 작업을 취소하고 이전 상태로 되돌리는 연산

3. SAVEPOINT(=CHECKPOINT) : 트랜잭션 내에 ROLLBACK 할 위치인 저장점을 지정하는 명령어

섹션 42 절차형 SQL

절차형 SQL은 C, JAVA 등의 프로그래밍 언어와 같이 연속적인 실행이나 분기, 반복 등의 제어가 가능한 SQL을 말한다.

- 절차형 SQL은 일반적인 프로그래밍 언어에 비해 효율은 떨어지지만 단일 SQL 문장으로 처리하기 어려운 연속적인 작업들을 처리하는데 적합하다.
- 절차형 SQL을 활용하여 다양한 기능을 수행하는 저장 모듈을 생성할 수 있다.
- DBMS 엔진에서 직접 실행되기 때문에 입·출력 패킷이 적은 편이다.
- BEGIN~END 형식으로 작성되는 블록 구조로 되어 있어 기능별 모듈화 가능
- 절차형 SQL의 종류에는 프로시저, 트리거, 사용자 정의 함수가 있다.

1. 프로시저 (Procedure) : 특정 기능을 수행하는 일종의 트랜잭션 언어로, 호출을 통해 실행되어 미리 저장해 놓은 SQL 작업을 수행한다.

2. 트리거 (Trigger) : DB시스템에서 데이터의 입력, 갱신, 삭제 등의 이벤트가 발생할 때마다 관련 작업이 자동으로 수행된다.

3. 사용자 정의 함수 : 프로시저와 유사하게 SQL을 사용하여 일련의 작업을 연속적으로 처리하며, 종료 시 예약어 Return을 사용하여 처리 결과를 단일값으로 반환

절차형 SQL의 테스트와 디버깅 : 절차형 SQL은 디버깅을 통해 기능의 적합성 여부를 검증하고, 실행을 통해 결과를 확인하는 테스트 과정을 수행한다.

- 절차형 SQL은 테스트 전에 생성을 통해 구문 오류(Syntax Error)나 참조 오류의 존재 여부를 확인한다.
- 많은 코드로 구성된 절차형 SQL의 특성상 오류 및 경고 메시지가 상세히 출력되지 않으므로 SHOW 명령어를 통해 내용을 확인하고 문제를 수정함
- 정상적으로 생성된 절차형 SQL은 디버깅을 통해 로직을 검증하고, 결과를 통해 최종적으로 확인한다.
- 절차형 SQL의 디버깅은 실제로 DB에 변화를 줄 수 있는 삽입 및 변경 관련 SQL문을 주석으로 처리하고, 출력문을 이용하여 화면에 출력하여 확인한다.

쿼리 성능 최적화 : 데이터 입·출력 애플리케이션의 성능 향상을 위해 SQL 코드를 최적화 하는 것

- 쿼리 성능을 최적화 하기 전에 성능 측정 도구인 APM을 사용하여 최적화 할 쿼리를 선정해야한다.
- 최적화를 할 쿼리에 대해 옵티마이저가 수립한 실행 계획을 검토하고 SQL 코드와 인덱스를 재구성 한다.

2장 통합 구현

섹션 43 단위 모듈 구현 (C)

단위 모듈의 개요 : 소프트웨어 구현에 필요한 여러 동작 중 한 가지 동작을 tgod 하는 기능을 모듈로 구현한 것

- 단위 모듈로 구현되는 하나의 기능을 단위 기능이라 부른다.
- 단위 모듈은 사용자나 다른 모듈로부터 값을 전달받아 시작되는 작은 프로그램을 의미하기도 한다.
- 두 개의 단위 모듈이 합쳐질 경우 두 개의 기능을 구현할 수 있다.
- 단위 모듈의 구성 요소에는 처리문, 명령문, 데이터 구조 등이 있다
- 단위 모듈은 독립적인 컴파일이 가능하며, 다른 모듈에 호출되거나 삽입된다.
- 단위 모듈을 구현하기 위해서는 단위 기능 명세서를 작성한 후 입·출력 기능과 알고리즘을 구성해야 한다.
- (단위 기능 명세서 작성) -> (입·출력 기능 구현) -> (알고리즘 구현)

1. 단위 기능 명세서 작성 : 설계 과정에서 작성하는 기능 및 코드 명세서나 설계 지침과 같이 단위 기능을 명세화한 문서를 작성하는 것이다.

- 명세서를 작성하기 위해 복잡한 시스템을 단순히 구현하기 위한 추상화 작업이 필요하다.
- 이 단계에서 대형 시스템을 분해하여 단위 기능별로 구분하고, 각 기능들을 계층적으로 구성하는 구조화 과정을 거친다.
- 명세서 작성 시 모듈의 독립적인 운용과 한 모듈 내의 정보가 다른 모듈에게 영향을 주지 않도록 정보 은닉의 원리를 고려한다.

2. 입·출력 기능 구현 : 단위 기능 명세서에서 정의한 데이터 형식에 따라 입·출력 기능을 위한 알고리즘 및 데이터를 구현한다.

- 이 단계에서는 단위 모듈 간의 연동 또는 통신을 위한 입·출력 데이터를 구현
- 입·출력 기능 구현 시 UI인 CLI, GUI와의 연동을 고려한다
- 입·출력 기능 구현 시 네트워크나 외부 장치와의 입·출력은 무료로 공개되어 있는 Open Source API를 이용하면 간편하게 구현할 수 있다.

IPC (Inter-Process Communication) : 모듈 간 통신 방식을 구현하기 위해 사용되는 대표적인 프로그래밍 인터페이스 집합으로, 복수의 프로세스를 수행하며 이뤄지는 프로세스 간 통신까지 구현이 가능하다.

※IPC의 대표 메소드 5가지

Shared Memory	다수의 프로세스가 공유 가능한 메모리를 구성하여 프로세스 간 통신을 수행
Socket	네트워크 소켓을 이용하여 네트워크를 경유하는 프로세스들 간 통신을 수행
Semaphores	공유 자원에 대한 접근제어를 통해 프로세스 간 통신 수행
Pipes& named Pipes	Pipe라고 불리는 FIFO 형태로 구성된 메모리를 여러 프로세스가 공유하여 통신을 수행 한 프로세스가 Pipe를 이용 중이면 다른 프로세스는 접근X
Message Queueing	메시지가 발생하면 이를 전달하는 형태로 통신 수행

3. 알고리즘 구현 : 입·출력 데이터를 바탕으로 단위 기능별 요구사항을 구현 가능한 언어를 이용하여 모듈로 구현

- 이 단계에서는 구현된 단위 기능들이 사용자의 요구와 일치하는지 확인해야함
- 구현되는 모듈은 단위 기능의 종류에 따라 다음과 같이 구분된다.

디바이스 드라이버 모듈	하드웨어 주변 장치의 동작을 구현한 모듈
네트워크 모듈	네트워크 장비 및 데이터 통신을 위한 기능을 구현한 모듈
파일 모듈	컴퓨터 내부의 데이터 구조 영역에 접근하는 방법을 구현한 모듈
메모리 모듈	파일으르 프로세스의 가상 메모리에 매핑/해제하는 방법, 프로세스 사이의 통신 기능을 구현한 모듈
프로세스 모듈	하나의 프로세스 안에서 다른 프로세스를 생성하는 방법을 구현한 모듈

섹션 44 단위 모듈 테스트

단위 모듈 테스트 개요 : 프로그램 단위 기능을 구현하는 모듈이 정해진 기능을 정확히 수행하는지 검증하는 것이다.

- 간단히 단위 테스트라고도 하며, 화이트박스 테스트와 블랙박스 테스트 기법을 사용한다.
- 단위 모듈 테스트를 하기 위해서는 모듈을 단독적으로 실행할 수 있는 환경과 테스트에 필요한 데이터가 모두 준비되어야 한다.
- 모듈의 통합 이후에는 오랜 시간 추적해야 발견할 수 있는 에러들도 단위 모듈 테스트를 수행하면 쉽게 발견하고 수정할 수 있다
- 단위 모듈 테스트의 기준은 단위 모듈에 대한 코드이므로 시스템 수준의 오류는 잡아낼 수 없다

테스트 케이스(Test Case) : 구현된 소프트웨어가 사용자의 요구사항을 정확하게 준수했는지를 확인하기 위해 설계된 입력 값, 실행 조건, 기대 결과 등으로 구성된 테스트 항목에 대한 명세서로, 명세 기반 테스트의 설계 산출물에 해당된다.

- 단위 모듈을 테스트하기 전에 테스트에 필요한 입력 데이터, 테스트 조건, 예상 결과 등을 모아 테스트 케이스를 만든다.
- 테스트케이스를 이용하지 않고 수행하는 직관적인 테스트는 특정 요소에 대한 검증이 누락되거나 불필요한 검증의 반복으로 인력/시간을 낭비할 수 있다.
- ISO/IEC/IEEE 29119-3 표준에 따른 테스트 케이스 구성 요소는 다음과 같다.

식별자(Identifier)	항목 식별자. 일련번호
테스트 항목(Test Item)	테스트 대상(모듈 또는 기능)
입력 명세 (Input Specification)	입력 데이터 또는 테스트 조건
출력 명세 (Output Specification)	테스트 케이스 수행 시 예상되는 출력 결과
환경 설정 (Environmental Needs)	필요한 하드웨어나 소프트웨어의 환경
특수 절차 요구 (Special Procedure Requirement)	테스트 케이스 수행 시 특별히 요구되는 절차
의존성 기술 (Inter-case Dependencied)	테스트 케이스 간의 의존성

테스트 프로세스 : 테스트를 위해 수행하는 모든 작업들이 테스트의 목적과 조건을 달성할 수 있도록 도와주는 과정이다.

- 테스트 프로세스 5단계

계획 및 제어 -> 분석 및 설계 단계 -> 구현 및 실현 단계 -> 평가 단계 -> 완료 단계

섹션 45 개발 지원 도구

통합 개발 환경 (IDE; Integrated Development Environment) : 코딩, 디버그, 컴파일, 배포(Deployment) 등 프로그램 개발과 관련된 모든 작업을 하나의 프로그램에서 처리할 수 있도록 제공하는 소프트웨어적인 개발 환경을 말한다.

- 기존 소프트웨어 개발에서는 편집기(Editor), 컴파일러,디버거 등 다양한 툴을 별도로 사용했으나 현재는 하나의 인터페이스로 통합하여 제공한다.
- IDE를 지원하는 도구는 플랫폼, 운영체제, 언어별로 다양하게 존재한다.

프로그램	개발사	플랫폼	운영체제	지원 언어
이클립스(Eclipse)	Eclipse Foundation, IBM	크로스 플랫폼	Windows, Linux, MacOS 등	Java, C, C++, PHP, JSP 등
비주얼 스튜디오 (Visual Studio)	Microsoft	Win32, Win64	Windows	Basic, C, C++, C#, .NET 등
엑스 코드(Xcode)	Apple	Mac, iPhone	MacOS, iOS	C, C++, C#, Java, AppleScript 등
안드로이드 스튜디오 (Android Studio)	Google	Android	Windows, Linux, MacOS	Java, C, C++
IDEA	JetBrains (이전 IntelliJ)	크로스 플랫폼	Windows, Linux, MacOS	Java, JSP, XML, Go, Kotlin, PHP 등

빌드 도구 : 빌드는 소스 코드 파일들을 컴퓨터에서 실행할 수 있는 제품 소프트웨어로 변환하는 과정 또는 결과물을 말한다.

- 빌드 도구는 소스 코드를 소프트웨어로 변환하는 과정에 필요한 전처리(Preprocessing), 컴파일 (Compile) 등의 작업들을 수행하는 소프트웨어를 말한다.
- 대표적인 도구로는 Ant, Maven, Gradle 등이 있다.

Ant (Another Neat Tool)	<ul style="list-style-type: none"> - 아파치 소프트웨어 제단에서 개발한 SW로, 자바 프로젝트의 공식적인 빌드 도구로 사용되고 있다. - XML 기반의 빌드 스크립트를 사용하며, 자유도와 유연성이 높아 복잡한 빌드 환경에도 대처가 가능하다 - 정해진 규칙이나 표준이 없어 개발자가 모든 것을 정의하며, 스크립트의 재사용이 어렵다.
Maven	<ul style="list-style-type: none"> - Ant와 동일하게 아파치 제단에서 만들었으며, Ant 대안으로 개발되었다. - 규칙이나 표준이 존재하여 예외 사항만 기록하면 되며, 컴파일과 빌드를 동시에 수행한다. - 의존성(Dependency)을 설정하여 라이브러리를 관리한다.
Gradle	<ul style="list-style-type: none"> - 기존의 Ant와 Maven을 보완하여 개발된 빌드 도구이다. - 한스 도커 외 6인이 모여 공동 개발하였다. - 안드로이드 스튜디오의 공식 빌드 도구로 채택된 소프트웨어다 - Maven처럼 의존성을 활용하며, 그루비(Groovy) 기반의 빌드 스크립트를 사용한다.

기타 협업 도구 : 개발에 참여하는 사람들이 서로 다른 작업 환경에서 프로젝트를 수행할 수 있도록 도와주는 도구(Tool)로, 협업 소프트웨어, 그룹웨어(Groupware) 등으로도 불린다.

프로젝트 및 일정 관리	전체 프로젝트와 개별 업무들의 진행상태, 일정 공유 구글 캘린더, 분더리스트, 트렐로, 지라, 플로우 등
정보 공유 및 커뮤니케이션	주제별로 구성원들을 지목하여 방을 만들고 정보 공유 및 대화 슬랙, 잔디, 태스크월드 등
디자인	디자이너가 설계한 UI나 이미지의 정보를 코드화하여 개발자에게 전달 스케치, 제플린 등
기타	아이디어 공유에 사용되는 에버노트 API를 문서화하여 개발자들 간 협업을 도와주는 스웨거 Git의 웹호스팅 서비스인 Github

3장 제품 소프트웨어 패키징

섹션 46 소프트웨어 패키징

개요 : 소프트웨어 패키징이란 모듈별로 생성한 실행 파일들을 묶어 배포용 설치

파일을 만드는 것

- 개발자가 아니라 사용자 중심으로 진행
- 소스 코드는 향후 관리를 고려하여 모듈화하여 패키징해야함
- 사용자가 소프트웨어를 사용하게 될 환경을 이해하고, 다양한 환경에서 SW를 손쉽게 사용할 수 있도록 일반적인 패포 형태로 패키징한다.

패키징 시 고려사항

- 사용자의 시스템 환경, 즉 OS, CPU, 메모리 등에 필요한 최소 환경을 정의
- UI는 사용자 눈으로 직접 확인할 수 있도록 시각적인 자료와 함께 제공하고 매뉴얼과 일치시켜 패키징한다.
- 소프트웨어는 단순히 패키징하여 배포하는 것으로 끝나는 것이 아니라 하드웨어와 함께 관리될 수 있도록 Managed Service 형태로 제공하는 것이 좋다.
- 사용자에게 배포되는 SW이므로 내부 콘텐츠에 대한 암호화 및 보안을 고려
- 다른 여러 콘텐츠 및 단말기 간 DRM 연동을 고려한다.
- 사용자의 편의성을 위한 복잡성 및 비효율성 문제 고려
- 제품 SW 종류에 적합한 암호화 알고리즘 적용

패키징 작업 순서

- 패키징 주기는 소프트웨어 개발 기법에 따라 달라지는데, 짧은 개발 주기를 반복하는 애자일 기법인 경우에는 보통 2주~4주 내에서 지정하며, 각 주기가 끝날 때마다 패키징을 수행
- 프로젝트 개발 과정에서 주기별로 패키징한 결과물은 테스트 서버에서 배포
- 마지막 개발 과정을 거쳐 최종 패키징한 결과물은 고객이 사용할 수 있도록 온라인/오프라인으로 배포한다.

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

섹션 47 릴리즈 노트 작성 (C등급)

개요 : 릴리즈 노트는 개발 과정에서 정리된 릴리즈 정보를 소프트웨어의 최종 사용자인 고객과 공유하기 위한 문서이다.

- 릴리즈 노트를 통해 테스트 진행 방법에 대한 결과와 소프트웨어 사양에 대한

개발팀의 정확한 준수 여부를 확인할 수 있다.

- 소프트웨어에 포함된 전체 기능, 서비스의 내용. 개선사항을 사용자와 공유
- 릴리즈 노트로 소프트웨어의 버전 관리나 릴리즈 정보를 체계적으로 관리
- 릴리즈 노트에 정리된 정보들은 철저한 테스트를 거친 것이며, 개발팀에서 제공하는 소프트웨어 사양에 대한 최종 승인을 얻은 후 문서화 되어 제공된다.

릴리즈 노트 초기 버전 작성 시 고려사항

- 릴리즈 노트는 정확하고 완전한 정보를 기반으로 개발팀에서 직접 8-제 시제로 작성해야 한다.
- 신규 소스, 빌드 등의 이력이 정확하게 관리되어 변경 또는 개선된 항목에 대한 이력 정보들도 작성되어야 한다.
- 릴리즈 노트 작성에 대한 표준 형식은 없지만 일반적으로 다음과 같은 항목이 포함된다.

항목	내용
Header (머릿말)	릴리즈 노트 이름, 소프트웨어 이름, 릴리즈 버전, 릴리즈 날짜, 릴리즈 노트 날짜, 릴리즈 노트 버전 등
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 참조사항 (예) 프리웨어, 불법 복제 금지 등
연락처	사용자 지원 및 문의 응대를 위한 연락처 정보

릴리즈 노트 추가 버전 작성 시 고려사항 : SW 테스트 과정에서 베타 버전이 출시되거나 긴급한 버그 수정, 업그레이드와 같은 자체 기능 향상, 사용자 요청 등의 특수한 상황이 발생하는 경우 릴리즈 노트를 추가로 작성한다.

- 중대한 오류가 발생하여 긴급하게 수정하는 경우 릴리즈 버전을 출시하고 버그

번호를 포함한 모든 수정된 내용을 담아 릴리즈 노트를 작성한다.

- 소프트웨어에 대한 기능 업그레이드 완료한 경우에는 릴리즈 버전을 출시하고 릴리즈노트를 작성한다.
- 사용자로부터 접수된 요구사항에 의해 추가나 수정된 경우 자체 기능 향상과는 다른 별도의 릴리즈 버전으로 출시하고 릴리즈 노트 작성

릴리즈 노트 작성 순서 : 모듈 식별 -> 릴리즈 정보 확인 -> 릴리즈 노트 개요 작성 -> 영향도 체크 -> 정식 릴리즈 노트 작성 -> 추가 개선 항목 식별

섹션 48 디지털 저작권 관리 (DRM)

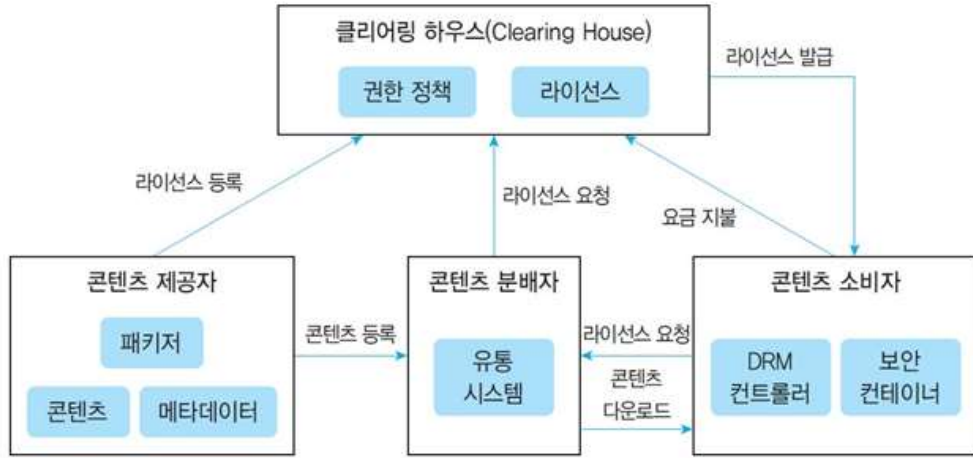
저작권은 소설, 시, 논문, 강연, 연술, 음악, 연극, 무용, 회화, 서예, 건축물, 사진, 영상, 지도, 도표, 컴퓨터 프로그램 저작물 등에 대하여 창작자가 가지는 배타적 독점적 권리로 타인의 침해를 받지 않을 고유한 권한이다.

- 컴퓨터 프로그램도 복사하기 쉬운 저작물에 대해 불법 복제 및 배포 등을 막기 위한 기술적인 방법을 통칭해 저작권 보호 기술이라 한다.

디지털 저작권 관리(DRM; Digital Right Management) : 저작권자가 배포한 디지털 콘텐츠가 저작권자가 의도한 용도로만 사용되도록 디지털 콘텐츠의 생성, 유통, 이용까지의 전 과정에 걸쳐 사용되는 디지털 콘텐츠 관리 및 보호 기술이다.

- 원본 콘텐츠가 아날로그인 경우에는 디지털로 변환 후 패키지(packager)에 의해 DRM 패키징을 수행한다.
- 콘텐츠의 크기에 따라 음원이나 문서와 같이 크기가 작은 경우에는 사용자가 콘텐츠를 요청하는 시점에서 실시간으로 패키징을 수행하고, 크기가 큰 경우에는 미리 패키징을 수행한 후 배포한다.
- 패키징을 수행하면 콘텐츠에는 암호화된 저작권자의 전자서명이 포함되고 저작권자가 설정한 라이선스 정보가 클리어링 하우스에 등록된다 (결제 이루어지는 곳에 저장되어 콘텐츠를 이용할 수 있도록 함)
- 종량제 방식을 적용한 소프트웨어인 경우 클리어링 하우스를 통해 서비스의 실제 사용량을 측정하여 이용한 만큼의 요금을 부과한다.

디지털 저작권 관리의 흐름 및 구성 요소



1. 클리어링 하우스 : 저작권에 대한 사용 권한, 라이선스 발급, 암호화된 키 관리, 사용량에 따른 결제 관리 등을 수행
2. 패키지 (Packager) : 콘텐츠를 메타 데이터와 함께 배포 가능한 형태로 묶어 암호화 하는 프로그램
3. 콘텐츠 분배자 : 암호화된 콘텐츠를 유통하는 곳이나 사람
4. 보안 컨테이너 : 콘텐츠 원본을 안전하게 유통하기 위한 전자적 보안 장치

디지털 저작권 관리의 기술 요소

구성 요소	설명
암호화	콘텐츠 및 라이선스를 암호화하고 전자 서명을 할 수 있는 기능
키 관리	콘텐츠를 암호화한 키에 대한 저장 및 분배 기술
암호화 파일 생성 (Packager)	콘텐츠를 암호화된 콘텐츠로 생성하기 위한 기술
식별 기술	콘텐츠에 대한 식별 체계 표현 기술
저작권 표현	라이선스의 내용 표현 기술
정책 관리	라이선스 발급 및 사용에 대한 정책 표현 및 관리 기술
크랙 방지 (Tamper Resistance)	크랙에 의한 콘텐츠 사용 방지 기술
인증 (Authentication)	라이선스 발급 및 사용의 기준이 되는 사용자 인증 기술

섹션 49 소프트웨어 설치 매뉴얼 작성

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

- 설치 매뉴얼은 사용자 기준으로 작성
- 설치 시작부터 완료할 때까지의 전 과정을 빠짐없이 순서대로 설명한다
- 설치 과정에서 표시될 수 있는 오류 메시지 및 예외 상황에 관한 내용을 별도로 분류하여 설명한다.
- 소프트웨어 설치 매뉴얼에는 목차 및 개요, 서문, 기본 사항 등이 기본적으로 포함되어야 한다.
- 소프트웨어 설치 매뉴얼의 목차에는 전체 설치 과정을 순서대로 요약한 후 관련 내용의 시작 페이지를 함께 기술한다.
- 설치 매뉴얼의 주요 특징, 구성과 설치 방법, 순서등의 내용을 기술한다.

1. 서문 : 서문에는 문서 이력, 설치 매뉴얼의 주석, 설치 도구의 구성, 설치 환경 체크 항목을 기술한다.

- 문서 이력은 버전 및 작성자 변경 내용 등을 기입
- 설치 매뉴얼의 주석 : 주의사항과 참고사항을 기술함
- 설치 도구의 구성 : exe, dll, ini, chm 등의 설치 관련 파일에 대해 설명.
- 설치 환경 체크 항목 : 사용자 환경, 응용 프로그램, 업그레이드 버전, 백업 폴더 확인 등을 기술함

2. 기본 사항

항목	설명
소프트웨어 개요	소프트웨어의 주요 기능 및 UI 설명 UI 및 화면 상의 버튼, 프레임 등을 그림으로 설명
설치 관련 파일	소프트웨어 설치에 필요한 파일 설명 exe, ini, log 등의 파일 설명
설치 아이콘	설치 아이콘 설명
프로그램 삭제	설치된 소프트웨어의 삭제 방법 설명
관련 추가 정보	소프트웨어 이외의 관련 설치 프로그램 정보 소프트웨어 제작사 등의 추가 정보 기술

설치 매뉴얼 작성 방법 생략 (p231~232)

설치 매뉴얼 작성 순서 :

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


섹션 50 소프트웨어 사용자 매뉴얼 작성

개요 : 사용자가 소프트웨어를 사용하는 과정에서 필요한 내용을 문서로 기록한 설명서와 안내서

- 사용자 매뉴얼은 사용자가 소프트웨어 사용에 필요한 절차, 환경 등의 제반 사항이 모두 포함되도록 작성한다
- SW 배포 후 발생할 수 있는 오류에 대한 패치나 기능에 대한 업그레이드를 위해 매뉴얼의 버전을 관리한다.
- 개별적으로 동작이 가능한 컴포넌트 단위로 매뉴얼을 작성한다
- 사용자 매뉴얼은 컴포넌트 명세서와 컴포넌트 구현 설계서를 토대로 작성한다

1. 서문 : 문서 이력, 사용자 매뉴얼의 주석, 기록 보관을 위해 필요한 내용 기술

2. 기본 사항 : 기본적으로 설명되어야 할 항목

항목	설명
소프트웨어 개요	 소프트웨어의 주요 기능 및 UI 설명 • UI 및 화면 상의 버튼, 프레임 등을 그림으로 설명
소프트웨어 사용 환경	• 소프트웨어 사용을 위한 최소 환경 설명 • CPU, 메모리 등의 PC 사양, 운영체제(OS) 버전 설명 • 최초 구동에 대한 설명 • 소프트웨어 사용 시 발생할 수 있는 프로그램 충돌이나 개인정보, 보안 등에 관한 주의사항을 설명함.
소프트웨어 관리	소프트웨어의 사용 종료 및 관리 등에 관한 내용 설명
모델, 버전별 특징	모델 및 버전별로 UI 및 기능의 차이점을 간략하게 요약함.
기능, 인터페이스의 특징	제품의 기능과 인터페이스의 특징을 간략하게 요약함.
소프트웨어 구동 환경	• 개발에 사용한 언어 및 호환 가능한 운영체제(OS)에 대해 설명함. • 설치 후 구동하기까지의 과정을 운영체제(OS)별로 설명함.

사용자 매뉴얼 작성 방법 생략 (p235~236)

사용자 매뉴얼 작성 순서 : 작성 지침 정의 -> 사용자 매뉴얼 구성요소 정의-> 구성 요소별 내용 작성 -> 사용자 매뉴얼 검토

섹션 51 소프트웨어 버전 등록 (★)

소프트웨어 패키징의 형상 관리 : 형상 관리 (SCM; Software Configuration Management)는 소프트웨어의 변경 사항을 관리하기 위해 개발된 일련의 활동이다.

- 소프트웨어 변경의 원인을 알아내고 제어하며, 적절히 변경되고 있는지 확인하여 해당 담당자에게 통보한다.
- 형상 관리는 소프트웨어 개발의 전 단계에 적용되는 활동이며, 유지보수 단계에서도 수행된다.
- 형상 관리는 소프트웨어 개발의 전체 비용을 줄이고, 개발 과정의 여러 방해 요인이 최소화되도록 보증하는 것을 목적으로 한다.
- 관리 항목에는 소스 코드뿐만 아니라 프로젝트 계획, 분석서, 설계서, 프로그램, 테스트 케이스등이 포함된다.
- 형상관리를 통해 가시성과 추적성을 보장함으로써 소프트웨어의 생산성과 품질을 높일 수 있다.
- 대표적인 형상 관리 도구에는 Git, CVS, Subversion 등이 있다.

형상 관리의 중요성

- 지속적인 소프트웨어 변경 사항을 체계적으로 추적하고 통제할 수 있다.
- 제품 SW에 대한 무제한 변경을 방지할 수 있다.
- 제품 SW에서 발견된 버그나 수정 사항을 추적 사항을 추적할 수 있다
- 소프트웨어는 형태가 없어 가시성이 결핍되므로 진행 정도를 확인하기 위한 기준으로 사용될 수 있다.
- 소프트웨어의 배포본을 효율적으로 관리할 수 있다
- 소프트웨어를 여러 명의 개발자가 동시에 개발할 수 있다

형상 관리 기능

형상 관리는 품질 보증을 위한 중요한 요소로 다음 기능을 수행한다.

1. 형상 식별 : 형상 관리 대상에 이름과 관리 번호를 부여하고, 계층(Tree) 구조로 구분하여 수정 및 추적이 용이하도록 하는 작업
2. 버전 제어 : 소프트웨어 업그레이드나 유지 보수 과정에서 생성된 다른 버전의 형상 항목을 관리하고, 이를 위해 특정 절차와 도구를 결합시키는 작업
3. 형상 통제(변경 관리) : 식별된 형상 항목에 대한 변경 요구를 검토하여 현재의 기준선 (Base Line)이 잘 반영될 수 있도록 조정하는 작업

4. 형상 감사 : 기준선의 무결성을 평가하기 위해 확인, 검증, 검열 과정을 통해 공식적으로 승인하는 작업
5. 형상 기록(상태 보고) : 형상의 식별, 통제, 감사 작업의 결과를 기록·관리하고 보고서를 작성하는 작업

소프트웨어의 버전 등록 관련 주요 기능

: 소프트웨어 개발 과정에서 코드와 라이브러리, 관련 문서 등의 버전을 관리하기 위해 자료를 기록하고 갱신하는 과정에서 사용되는 주요용어와 의미는 다음과 같다.

항목	설명
저장소 (Repository)	°최신 버전의 파일들과 변경 내역에 대한 정보들이 저장되어 있는 곳이다.
가져오기 (Import)	°버전 관리가 되고 있지 않은 아무것도 없는 저장소에 처음으로 파일을 복사한다
체크아웃 (Check-out)	°프로그램을 수정하기 위해 저장소에서 파일을 받아온다. °소스 파일과 함께 버전 관리를 위한 파일도 받아온다.
체크인 (Check-in)	°체크아웃 한 파일의 수정을 완료한 후 저장소의 파일을 새로운 버전으로 갱신한다
커밋 (Commit)	°체크인을 수행할 때 이전에 갱신된 내용이 있는 경우에는 충돌(Conflict)을 알리고 diff 도구를 이용해 수정한 후 갱신을 완료
동기화 (Update)	°저장소에 있는 최신 버전으로 자신의 작업 공간을 동기화

※ diff 도구 : 비교 대상이 되는 파일의 소스 코드를 비교하여 서로 다른 부분을 찾아 표시해주는 기능

소프트웨어 버전 등록 과정

: 가져오기(Import) -> 인출(Check-out) -> 예치(Commit) -> 동기화 -> 차이(Diff)

마지막 차이는, 수정 기록을 확인하여 커밋들간의 차이를 확인한다.

섹션 52 소프트웨어 버전 관리 도구

1. 공유 폴더 방식 : 버전 관리 자료가 로컬 컴퓨터의 공유 폴더에 저장되어 관리되는 방식

- 개발자들은 개발이 완료된 파일을 약속된 공유 폴더에 매일 복사한다.
- 담당자는 공유 폴더의 파일을 자기 PC로 복사한 후 컴파일 하여 이상 유무를

확인한다.

- 이상 유무 확인 과정에서 파일의 오류가 확인되면, 해당 파일을 등록한 개발자에게 수정을 의뢰한다.
- 파일에 이상이 없으면 다음날 각 개발자들이 동작 여부를 다시 확인한다.
- 파일을 잘못 복사하거나 다른 위치로 복사하는 것에 대비하기 위해 파일의 변경 사항을 데이터베이스에 기록하여 관리한다.
- 종류에는 SCCS, RCS, PVCS, QVCS 등이 있다.

2. 클라이언트/ 서버 방식 : 버전 관리 자료가 중앙 시스템(서버)에 저장되어 관리되는 방식

- 서버의 자료를 개발자별로 자신의 PC(클라이언트)로 복사하여 작업한 후 변경된 내용을 서버에 반영한다.
- 모든 버전 관리는 서버에서 수행한다.
- 하나의 파일을 서로 다른 개발자가 작업할 경우 경고 메시지를 출력한다
- 서버에 문제가 생기면, 서버가 복구되기 전까지 다른 개발자와의 협업 및 버전 관리 작업은 중단된다.
- 종류에는 CVS, SVN(Subversion), CVSNT, Clear Case, CMVC, Perforce 등이 있다.

3. 분산 저장소 방식 : 버전 관리 자료가 하나의 원격 저장소와 분산된 개발자 PC의 로컬 저장소에 함께 저장되어 관리되는 방식

- 개발자별로 원격 저장소의 자료를 자신의 로컬 저장소로 복사하여 작업한 후 변경된 내용을 로컬 저장소에서 우선 반영(버전 관리)한 다음 이를 원격 저장소에 반영한다.
- 로컬 저장소에서 버전 관리가 가능하므로 원격 저장소에 문제가 생겨도 로컬 저장소의 자료를 이용하여 작업할 수 있다.
- 종류에는 Git, GNU arch, DVCs, Bazaar, Mercurial, TeamWare, Bitkeeper, Plastic SCM 등이 있다.

Subversion (서브버전, SVN) (클라이언트/서버 방식) : Subversion은 CVS를 개선한 것으로, 아파치 소프트웨어 재단에서 발표함

- 서버의 자료를 클라이언트로 복사해와 작업한 후 변경 내용을 서버에 반영(Com

mit)함

- 모든 개발 작업은 trunk 디렉터리에서 수행, 추가 작업은 branches 디렉터리에서 별도의 디렉터를 만들어 작업을 완료한 후 trunk와 병합
- 커밋시 리비전(Revision)이 1씩 증가
- 오픈 소스. 주요 명령어는 p242 확인

Git(깃) : 리누스 토발즈가 2005년 리눅스 커널 개발에 사용할 관리도구로 개발한 이후 주니어 하마노에 의해 유지 보수되고 있다.

- Git은 분산 버전 관리 시스템으로 2개의 저장소, 즉 지역(로컬) 저장소와 원격 저장소가 존재한다.
 - 지역 저장소는 개발자들이 실제로 개발을 진행하는 장소로, 버전 관리가 수행
 - 원격 저장소는 여러 사람들이 협업을 위해 버전을 공동 관리하는 곳으로, 자신의 버전 관리 내역을 반영하거나 다른 개발자의 변경 내용을 가져올 때 사용
 - 버전 관리가 로컬에서 진행되므로 버전 관리가 신속하게 처리되고, 원격 저장소나 네트워크에 문제가 생겨도 작업이 가능
 - 브랜치를 이용하면 기번 버전 관리 틀에 영향을 주지 않으면서 다양한 형태의 기능 테스트가 가능하다
 - 파일의 변화를 스냅샷으로 저장하는데, 스냅샷은 이전 스냅샷의 포인터를 가지므로 버전의 흐름을 파악할 수 있다.
 - 주요 명령어

명령어	의미
add	<ul style="list-style-type: none">• 작업 내역을 지역 저장소에 저장하기 위해 스테이징 영역(Staging Area)에 추가함.• '- -all' 옵션으로 작업 디렉터리의 모든 파일을 스테이징 영역에 추가할 수 있음.
commit	작업 내역을 지역 저장소에 저장함.
branch	<ul style="list-style-type: none">• 새로운 브랜치를 생성함.• 최초로 commit을 하면 마스터(master) 브랜치가 생성됨.• commit 할 때마다 해당 브랜치는 가장 최근의 commit 한 내용을 가리키게 됨.• '-d' 옵션으로 브랜치를 삭제할 수 있음.
checkout	<ul style="list-style-type: none">• 지정한 브랜치로 이동함.• 현재 작업 중인 브랜치는 HEAD 포인터가 가리키는데, checkout 명령을 통해 HEAD 포인터를 지정한 브랜치로 이동시킴.

merge	지정한 브랜치의 변경 내역을 현재 HEAD 포인터가 가리키는 브랜치에 반영함으로써 두 브랜치를 병합함.
init	지역 저장소를 생성함.
remote add	원격 저장소에 연결함.
push	로컬 저장소의 변경 내역을 원격 저장소에 반영함.
fetch	원격 저장소의 변경 이력만을 지역 저장소로 가져와 반영함.
clone	원격 저장소의 전체 내용을 지역 저장소로 복제함.
fork	지정한 원격 저장소의 내용을 자신의 원격 저장소로 복제함.

처음 작업을 한다고 하면 remote add -> add -> all -> commit -> push 함

섹션 53 빌드 자동화 도구

빌드란 소스 코드 파일들을 컴파일한 후 여러 개의 모듈을 묶어 실행 파일로 만드는 과정이며, 이러한 빌드를 포함하여 테스트 및 배포를 자동화하는 도구를 빌드 자동화 도구라 한다.

- 애자일 환경에서는 하나의 작업이 마무리될 때마다 모듈 단위로 나눠서 개발된 코드들이 지속적으로 통합되는데, 이러한 지속적인 통합 개발 환경에서 빌드 자동화 도구는 유용하게 활용된다.
- 빌드 자동화 도구에는 Ant, Make, Maven, Gradle, Jenkins 등이 있으며 Jenkins와 Gradle이 대표적이다.

Jenkins : Java 기반의 오픈 소스 형태로 가장 많이 사용

- 서블릿 컨테이너에서 실행되는 서버 기반 도구이다.
- SVN, Git 등 대부분의 형상 관리 도구와 연동이 가능함
- 친숙한 Web GUI 제공으로 사용이 쉽다
- 여러 대의 컴퓨터를 이용한 분산 빌드나 테스트가 가능하다

Gradle : Groovy 기반으로 한 오픈 소스 형태의 자동화 도구로, 주로 안드로이드 앱 개발 환경에서 사용된다.

- Java, C/C++, Python 등 언어도 빌드가 가능
- Groovy를 사용해서 만든 DSL(Domain Specific Language)을 스크립트 언어

- 로 사용한다.
- Gradle은 실행할 처리 명령들을 모아 태스크(task)로 만든 후 태스크 단위로 실행한다.
 - 이전에 사용했던 태스크를 재사용하거나 다른 시스템의 태스크를 공유할 수 있는 빌드 캐시 기능을 지원하므로 빌드의 속도를 향상시킬 수 있다

4장 애플리케이션 테스트 관리

섹션 54 애플리케이션 테스트

- 개념 : 애플리케이션에 잠재되어 있는 결함을 찾아내는 일련의 행위 또는 절차
- 개발된 소프트웨어가 고객의 요구사항을 만족시키는지 확인하고 소프트웨어가 기능을 정확히 수행하는 검증함
 - 애플리케이션 테스트를 실행하기 전에 개발한 소프트웨어의 유형을 분류하고 특성을 정리해서 중점적으로 테스트할 사항을 정리해야 함
 - 확인은 사용자 중심, 검증은 개발자 중심이다

소프트웨어의 분류 : 크게 상용 소프트웨어 , 서비스 제공 소프트웨어로 나뉨

상용 SW - 산업 범용 소프트웨어, 산업 특화 소프트웨어로 나뉨

서비스 제공 SW - 신규 개발 SW, 기능 개선 SW, 추가 개발 SW, 시스템 통합 SW로 나뉨

- 애플리케이션 테스트의 필요성
- 프로그램 실행 전 오류 발견 및 예방 가능
 - 제품의 신뢰도 향상
 - 새로운 오류의 유입도 방지 가능
 - 최소한의 시간과 노력으로 많은 결함을 찾을 수 있음

애플리케이션 테스트의 기본 원리

- 애플리케이션 테스트는 SW의 잠재적인 결함을 줄일 수 있지만 SW에 결함이 없다고는 증명할 수 없다. 즉 완벽한 SW 테스트는 없다
- 애플리케이션의 결함은 대부분 개발자의 특성이나 애플리케이션의 기능적 특징 때문에 특정 모듈에 집중되어 있다. 애플리케이션의 20%에 해당하는 코드에서 전체 80%의 결함이 발견된다고 하여 파레토 법칙을 적용하기도 한다.

- 애플리케이션 테스트에서는 동일한 테스트 케이스로 동일한 테스트를 반복하면 더 이상 결함이 발견되지 않는 살충제 패러독스(Pesticide Paradox) 현상이 발생한다. 이를 방지하기 위해 테스트 케이스도 지속적으로 보완 및 개선해야 한다.
- 애플리케이션 테스트는 SW의 트징 , 테스트 환경, 테스터 역량 등 정황(Context)에 따라 테스트 결과가 달라질 수 있으므로, 정황에 따라 테스트를 다르게 수행해야한다.
- 소프트웨어의 결함을 모두 제거해도사용자의 요구사항을 만족시키지 못하면 해당 소프트웨어는 품질이 높다고는 할 수 없다. 이것을 오류-부재의 궤변 (Absence of Errors Fallacy)라 한다.
- 테스트와 위험은 반비례한다. 테스트를 많이 반복하면 할수록 미래의 발생할 위험을 줄일 수 있다.
- 테스트는 작은 부분에서 시작하여 점점 확대하면서 진행해야한다.
- 테스트는 개발자와 관계없는 별도의 팀에서 수행해야한다.

섹션 55 애플리케이션 테스트의 분류

1. 프로그램 실행 여부에 따른 테스트

정적 테스트	○프로그램을 실행하지 않고, 명세서나 소스 코드를 대상으로 분석하는 테스트 ○소프트웨어 개발 초기에 결함을 발견할 수 있어 소프트웨어의 개발 비용을 낮추는데 도움이 됨 ○ 워크스루, 인스펙션, 코드 검사
동적 테스트	○프로그램을 실행하여 오류를 찾는 테스트, 소프트웨어 개발의 모든 단계에서 테스트를 수행 가능 ○블랙박스 테스트, 화이트박스 테스트

2. 테스트 기반(Test Bases)에 따른 테스트 : 무엇을 기반으로 하는지

명세 기반 테스트	○사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 만들어 구현하고 있는지 확인하는 테스트 ○ 동등분할, 경계값 분석 등등
구조 기반 테스트	○소프트웨어 내부의 논리 흐름에 따라 테스트 케이스를 작성 후 확인 ○ 구문 기반, 결정 기반, 조건 기반 등등
경험 기반 테스트	○유사 소프트웨어나 기술 등에 대한 테스터의 경험을 기반으로 테스트 ○사용자의 요구사항에 대한 명세가 불충분하거나 테스트 시간에 제약이 있는 경우 수행하면 효과적이다

3. 시각에 따른 테스트 : 누구를 기준으로 하느냐에 따른 분류

검증(Verification) vs 확인(Validation)

검증 테스트	개발자의 시각에서 제품의 생산과정을 테스트하는 것으로, 제품이 명세서대로 완성되었는지 테스트
확인 테스트	사용자의 시각에서 생산된 제품의 결과를 테스트하는 것으로, 사용자가 요구한 대로 제품이 완성되었는지, 제품이 정상적으로 동작하는지 테스트

4. 목적에 따른 테스트 : 무엇을 목적으로 할지

회복 테스트	시스템에 여러 가지 결함을 주어 실패하도록 한 후 올바르게 복구되는지를 확인하는 테스트
안전 테스트	시스템에 설치된 시스템 보호 도구가 불법적인 침입으로부터 시스템을 보호할 수 있는지를 확인하는 테스트
강도(Stress) 테스트	시스템에 과도한 정보량이나 빈도를 부과하여 과부하 시에도 소프트웨어가 정상적으로 실행되는지를 검증하는 테스트
성능 테스트	소프트웨어의 실시간 성능이나 전체적인 효율성을 진단하는 테스트로, 소프트웨어의 응답시간, 처리량 등을 테스트
구조 테스트	소프트웨어 내부의 논리적인 경로, 소스 코드의 복잡도 등을 평가하는 테스트
회귀 테스트	소프트웨어의 변경 또는 수정된 코드에 새로운 결함이 없음을 확인하는 테스트
병행 테스트	변경된 소프트웨어와 기존 소프트웨어에 동일한 데이터를 입력하여 결과를 비교하는 테스트

섹션 56 테스트 기법에 따른 애플리케이션 테스트 (★)

화이트박스 테스트

: 모듈의 원시 코드를 오픈시킨 상태에서 원시 코드의 논리적인 모든 경로를 테스트하여 테스트 케이스를 설계하는 방법이다.

- 화이트박스 테스트는 설계된 절차에 초점을 둔 구조적 테스트로 프로시저 설계의 제어 구조를 사용하여 테스트 케이스를 설계함, 테스트 과정의 초기에 적용된다
- 모듈 안의 작동을 직접 관찰한다
- 원시 코드(모듈)의 모든 문장을 한 번 이상 실행함으로써 수행된다.
- 프로그램의 제어 구조에 따라 선택, 반복 등의 분기점 부분들을 수행함으로써 논리적 경로를 제어한다.

화이트박스 테스트의 종류

기초 경로 검사 (Base Path Testing)	○대표적인 화이트테스트 기법 ○테스트 케이스 설계자가 절차적 설계의 논리적 복잡성을 측정할 수 있게 해주는 테스트 기법으로, 테스트 측정 결과는 실행 경로의 기초를 정의하는데 지침으로 사용된다.
제어 구조 검사 (Control Structure Testing)	○조건 검사(Condition Testing) : 프로그램 모듈 내에 있는 논리적 조건을 테스트하는 테스트 케이스 설계 기법 ○루프검사 (Loop Testing) : 프로그램의 반복 구조에 초점을 맞춰 실시하는 테스트케이스 설계 방법 ○데이터 흐름 검사 (Data Flow Testing) : 프로그램에서 변수의 정의와 변수 사용의 위치에 초점을 맞춰 실시하는 테스트 케이스 설계 기법

화이트박스 테스트의 검증 기준 : 테스트 케이스들이 테스트에 얼마나 적절한지를 판단하는 기준

문장 검증 기준 (Statement Coverage)	소스 코드의 모든 구문이 한 번 이상 수행되도록 테스트 케이스 설계
분기 검증 기준 (Branch Coverage)	결정 검증 기준(Decision Coverage)라고도 불리며, 소스 코드의 모든 조건문에 대해 조건이 True인 경우와 False 경우가 한 번 이상 수행되도록 테스트 케이스 설계
조건 검증 기준 (Condition Coverage)	소스 코드의 조건문에 포함된 개별 조건식의 결과가 True인 경우와 False인 경우가 한 번 이상 수행되도록 테스트 케이스 설계
분기/조건 기준 (Branch/Contion Coverage)	분기 검증 기준과 조건 검증 기준을 모두 만족하는 설계로, 조건문인 True인 경우와 False인 경우에 따라 조건 검증 기준의 입력 데이터를 구분하는 테스트 케이스 설계

블랙 박스 테스트 : 소프트웨어가 수행할 특정 기능을 알기 위해 각 기능이 완전히 작동되는 것을 입증하는 테스트로, 기능 테스트라고도 한다.

- 프로그램의 구조를 고려하지 않으므로 테스트 케이스는 프로그램 혹은 모듈의 요구나 명세를 기초로 결정한다
- 소프트웨어 인터페이스에서 실시되는 케이스이다
- 부정확하거나 누락된 기능, 인터페이스 오류, 자료 구조나 외부 데이터베이스 접근에 따른 오류, 행위나 성능 오류, 초기화와 종료 오류등을 발견하기 위해 사용되

며, 테스트 과정의 후반부에 적용된다.

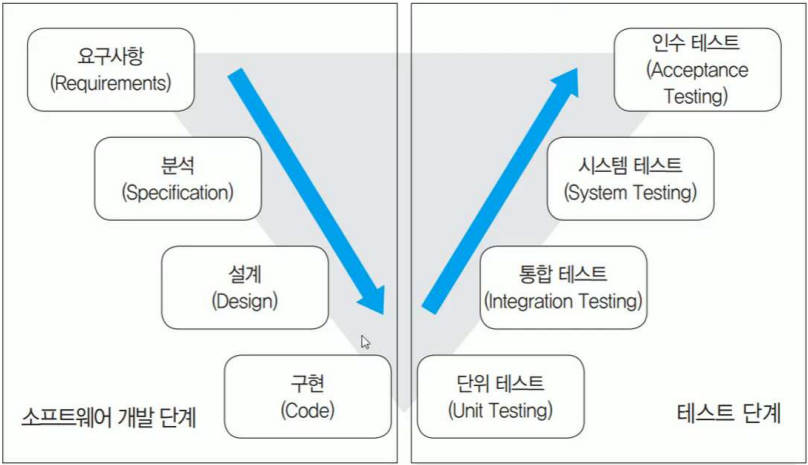
블랙박스 테스트의 종류

동치 분할 검사 (Equivalence Partitioning Testing)	○입력 자료에 초점을 맞춰 테스트 케이스 (동치 클래스)를 만들고 검사하는 방법으로 동등 분할 기법이라고도 한다. ○프로그램의 입력 조건에 타당한 입력 자료와 타당하지 않은 입력 자료의 개수를 균등하게 하여 테스트 케이스를 정하고, 해당 입력 자료에 맞는 결과가 출력되는지 확인하는 방법
경계값 분석 (Boundary Value Analysis)	○입력 자료에만 치중한 동치 분할 기법을 보완하기 위한 기법 ○입력 조건의 중간값보다 경계값에서 오류가 발생할 확률이 높다는 점을 이용하여 입력 조건의 경계값을 테스트 케이스로 선정하여 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 선정하여 검사하는 기법
오류 예측 검사 (Error Guessing)	○과거의 경험이나 확인자의 감각으로 테스트하는 기법 ○다른 블랙 박스 테스트 기법으로는 찾아낼 수 없는 오류를 찾아내는 일련의 보충적 검사 기법이며, 데이터 확인 검사라고도 한다.
비교 검사 (Comparison testing)	○여러 버전의 프로그램에 동일한 테스트 자료를 제공하여 동일한 결과가 출력되는지 테스트하는 기법

섹션 57 개발 단계에 따른 애플리케이션 테스트

애플리케이션 테스트는 소프트웨어의 개발 단계에 따라 단위 테스트, 통합 테스트, 시스템 테스트, 인수 테스트로 분류된다. 이렇게 분류된 것을 테스트 레벨이라 한다

- 애플리케이션 테스트는 소프트웨어의 개발 단계에서부터 테스트를 수행하므로 단순히 소프트웨어에 포함된 코드 상의 오류뿐만 아니라 요구 분석의 오류, 설계 인터페이스 오류 등도 발견할 수 있다.
- 애플리케이션 테스트와 소프트웨어 개발 단계를 연결하여 표현한 것을 V-모델이라 한다.



1. 단위 테스트 (Unit test) : 코딩 직후 SW 설계의 최소 단위인 모듈이나 컴포넌트에 초점을 맞춰 테스트하는 것이다.
- 단위 테스트에서는 인터페이스, 외부적 I/O, 자료 구조, 독립적 기초 경로, 오류 처리 경로, 경계 조건 등을 검사한다.
 - 사용자의 요구사항을 기반으로 한 가능성 테스트를 최우선으로 수행한다
 - 단위 테스트는 구조 기반 테스트와 명세 기반 테스트로 나뉘지만 주로 구조 기반 테스트를 시행한다.
 - 단위 테스트로 발견할 수 있는 오류 : 알고리즘 오류에 따른 원치 않은 결과, 탈출구가 없는 반복문의 사용, 틀린 계산 수식에 의한 잘못된 결과

방법	테스트 내용	테스트 목적
구조 기반 테스트	프로그램 내부 구조 및 복잡도 검증하는 화이트박스 테스트	제어 흐름, 조건 결정
명세기반 테스트	목적 및 실행 코드 기반의 블랙박스 테스트	동등 분할, 경계값 분석

2. 통합 테스트 (Integration Test) : 단위 테스트가 완료된 모듈들을 결합하여 하나의 시스템으로 완성시키는 과정에서의 테스트
- 모듈 간 또는 통합된 컴포넌트 간의 상호작용 오류를 검사함
3. 시스템 테스트 : 개발된 소프트웨어가 해당 컴퓨터 시스템에서 완벽히 수행되는지 점검하는 테스트

- 환경적인 장애 리스크를 최소화하기 위해 실제 사용 환경과 유사하게 만든 테스트 환경에서 테스트를 수행

- 시스템 테스트가 기능적 요구사항과 비기능적 요구사항으로 구분하여 테스트

방법	테스트 내용
기능적 요구사항	요구사항 명세서, 비즈니스 절차, 유스케이스 등 명세서 기반의 블랙박스 테스트
비기능적 요구사항	성능 테스트, 회복 테스트, 보안 테스트, 내부 시스템의 메뉴 구조, 웹 페이지의 네비게이션 등 구조적 요소에 대한 화이트박스 테스트

4. 인수 테스트 (Acceptance Test) : 개발한 소프트웨어가 사용자의 요구사항을 충족하는 지의 중점을 두고 테스트 하는 방법

- 인수 테스트는 개발한 sw를 사용자가 직접 테스트한다.
- 이 과정에서 문제가 없으면 사용자는 SW를 인수하게 되고, 프로젝트는 종료 됨
- 인수 테스트는 다음과 같은 6가지 종류로 구분해서 테스트한다.

테스트 종류	설명
사용자 인수 테스트	사용자가 시스템 사용의 적절성 여부를 확인
운영상의 인수 테스트	시스템 관리자가 시스템 인수 시 수행하는 테스트 기법으로, 백업/복원 시스템, 재난 복구, 사용자 관리, 정기 점검 등을 확인
계약 인수 테스트	계약상의 인수/검수 조건을 준수하는지 여부 확인
규정 인수 테스트	SW가 정부지침, 법규, 규정 등에 맞게 개발되었는지 확인
알파 테스트	○개발자의 장소에서 사용자가 개발자 앞에서 행하는 테스트 ○테스트는 통제된 환경에서 행해지며, 오류와 사용상의 문제점을 사용자와 개발자가 함께 확인하면서 기록
베타 테스트	○선정된 최종 사용자가 여러 명의 사용자 앞에서 행하는 테스트 기법으로, 필드 테스트(Field Testing)이라고도 한다. ○실업무를 가지고 사용자가 직접 테스트하는 것으로, 개발자에 의해 제어되지 않은 상태에서 테스트가 행해지며, 오류와 사용상의 문제점을 기록하고 개발자에게 주기적으로 보고

섹션 58 통합 테스트

위에서 설명했든 단위테스트가 끝난 모듈을 통합하는 과정에서 발생하는 상호작용 등의 오류 및 결함을 찾는 테스트 기법이다.

-통합 테스트 방법에는 비점진적 통합 방식과 점진적 통합 방식이 있다.

비점진적 통합 방식	○단계적으로 통합하는 절차 없이 모든 모듈이 미리 결하되어 있는 프로그램 전체를 테스트 하는 방법으로, 빅뱅 통합 테스트 방식이 있다. ○규모가 작은 소프트웨어에 유리하며 단시간 내에 테스트 가능 ○전체 프로그램을 대상으로 하기 때문에 오류 발견 및 장애 위치 파악이 어려움
점진적 통합 방식	○모듈 단위로 단계적으 통합하면서 테스트하는 방법. 하향식, 상향식, 혼합식 통합 방식이 있음 ○오류 수정이 용이, 인터페이스와 관련된 오류를 완전히 테스트할 가능성이 높음

1. 하향식 통합 테스트 : 프로그램의 상위 모듈에서 하위 모듈 방향으로 통합하면서 테스트

- 주로 제어 모듈을 기준으로 하여 아래로 통합하는데, 깊이 우선 통합법이나 넓이 우선 통합법을 사용함
- 테스트 초기부터 사용자에게 시스템 구조를 보여줄 수 있음
- 상위 모듈에서는 테스트 케이스를 사용하기 어려움
- 다음 절차로 구성

① 주요 제어 모듈은 작성된 프로그램을 사용학, 주요 제어 모듈의 종속모듈들은 스텝(Stub)(빈 모듈)으로 대체한다.

② 한 번에 하나씩 스텝 모듈을 실제 모듈로 교체

③ 모듈이 통합될 때마다(교체될 때마다) 테스트 실시

④ 새로운 오류가 발생하지 않음을 보증하기 위해 회귀 테스트를 실시

2. 상향식 통합 테스트 : 하위에서 상위 모듈로 통합

- 가장 하위 단계에서 통합 및 테스트를 수행하므로 스텝은 필요하지 않지만,하나의 주요 제어 모듈과 관련된 종속 모듈의 기roup인 클러스터(Cluster)가 필요하다.
- 상향식 통합 방법은 다음과 같은 절차로 수행

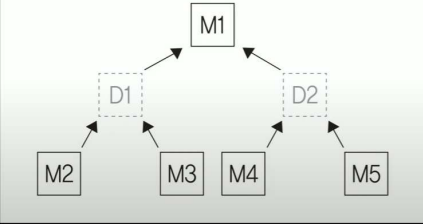
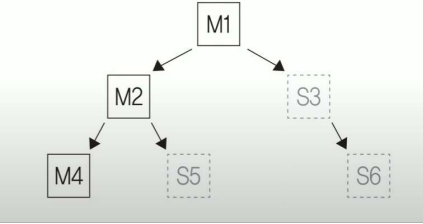
① 하위 모듈들을 클러스터로 결합

② 상위 모듈에서 데이터의 입·출력을 확인하기 위해 더미 모듈인 드라이버(Driver)를 작성

③ 통합된 클러스터 단위로 테스트

④ 테스트가 완료되면 클러스터는 프로그램 구조의 상위로 이동하여 결합하고 드라이버는 실제 모듈로 대체된다.

테스트 드라이버와 테스트 스텝의 차이

구분	드라이버	스텝
개념	테스트 대상의 하위 모듈을 호출하는 도구로, 매개 변수를 전달하고, 모듈 테스트 수행 후의 결과를 도출	제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구로, 일시적으로 필요한 조건만을 가지고 있는 시험용 모듈
필요시기	상위 모듈 없이 하위 모듈이 있는 경우 하위 모듈 구동	상위 모듈은 있으나 하위 모듈이 없는 경우 그 하위 모듈 대체
테스트 방식	상향식(Bottom up) 테스트	하향식(Top-down) 테스트
개념도		
공통점	소프트웨어 개발과 테스트를 병행할 경우 이용	
차이점	<ul style="list-style-type: none"> ○이미 존재하는 하위 모듈과 존재하지 않는 상위 모듈 간의 인터페이스 역할 ○소프트웨어 개발이 완료되면 드라이버는 본래의 모듈로 교체 	<ul style="list-style-type: none"> ○일시적으로 필요한 조건만을 가지고 임시로 제공되는 가짜모듈의 역할 ○시험용 모듈이기 때문에 일반적으로 드라이버보다 작성하기 쉬움

3. 혼합식 통합 테스트 : 하위 수준에서는 상향식 통합, 상위 수준에서는 하향식 통합을 사용하여 최적의 테스트를 지원하는 방식으로, 샌드위치식 통합 테스트 방법이라고도 한다.

4. 회귀 테스트(Regression Testing) : 회귀 테스트는 이미 테스트된 프로그램의 테스트를 반복하는 것으로, 통합 테스트로 인해 변경된 모듈이나 컴포넌트에 새로운 오류가 있는지 확인하는 테스트

- 회귀 테스트는 수정한 모듈이나 컴포넌트가 다른 부분에 영향을 미치는지, 오류가 생기지 않았는지 테스트하여 새로운 오류가 발생하지 않음을 보증하기 위해 반복 테스트한다.
- 회귀 테스트는 모든 테스트 케이스를 이용해 테스트하는 것이 좋지만, 시간과 비

이용이 많이 필요하므로, 기존 테스트 케이스 중 변경된 부분을 테스트할 수 있는 테스트 케이스만을 선정하여 수행

- 테스트 케이스 선정 방법

- : 모든 애플리케이션의 기능을 수행할 수 있는 대표적인 테스트 케이스를 선정
- : 애플리케이션 기능 변경에 따른 파급 효과를 분석하여 높은 부분이 포함된 테스트 케이스 선정
- : 실제 수정이 발생한 모듈 또는 컴포넌트에서 시행하는 테스트 케이스 선정

섹션 59 애플리케이션 테스트 프로세스 (C등급)

: 애플리케이션 테스트 프로세스는 개발된 소프트웨어가 사용자의 요구대로 만들어졌는지, 결함은 없는지 등을 테스트하는 절차로, 다음과 같은 순서로 진행된다.

테스트 계획 -> 테스트 분석 및 디자인 -> 테스트 케이스 및 시나리오 작성 -> 테스트 수행 -> 테스트 결과 평가 및 리포팅 -> 결함 추적 및 관리

애플리케이션 테스트를 마치면 테스트 계획서, 테스트 케이스, 테스트 시나리오, 테스트 결과서가 산출된다.

1. 테스트 케이스 : 사용자의 요구사항을 얼마나 준수하는지 확인하기 위한 입력값, 실행 조건, 기대 결과 등으로 만들어진 테스트 항목의 명세서
2. 테스트 시나리오 : 테스트를 수행할 여러 개의 테스트 케이스의 동작 순서를 기술한 문서

이정도 순서를 기억하자

섹션 60 테스트 케이스/ 테스트 시나리오/ 테스트 오라클

1. **테스트 케이스 (Test Case)** : 테스트 케이스는 구현된 소프트웨어가 사용자의 요구사항을 정확하게 준수했는지를 확인하기 위해 설계된 입력 값, 실행 조건, 기대 결과 등으로 구성된 테스트 항목에 대한 명세서로, 명세 기반 테스트의 설계 산출물에 해당된다.

- 테스트 케이스를 미리 설계하면 테스트 오류를 방지할 수 있고 테스트 수행에 필요한 인력, 시간 등의 낭비를 줄일 수 있다.
- 테스트 케이스는 테스트 목표와 방법을 설정한 후 작성한다

- 테스트 케이스는 테스트 목표와 방법을 설정한 후 작성한다.

테스트 케이스 작성순서

1. 테스트 계획 검토 및 자료 확보
2. 위험 평가 및 우선순위 결정
3. 테스트 요구사항 정의
4. 테스트 구조 설계 및 테스트 방법 결정
5. 테스트 케이스 정의
6. 테스트 케이스 타당성 확인 및 유지 보수

테스트 시나리오 : 테스트 케이스를 적용하는 순서에 따라 여러 개의 테스트 케이스들을 묶은 집합으로, 테스트 케이스들을 적용하는 구체적인 절차를 명세화한 문서

- 테스트 시나리오에는 테스트 순서에 대한 구체적인 절차, 사전 조건의 입력 데이터 등이 설정되어 있다
- 테스트 시나리오를 통해 테스트 순서를 미리 정함으로써 테스트 항목을 빠짐없이 수행할 수 있다.

테스트 시나리오 작성 시 유의 사항

- 테스트 시나리오는 시스템별, 모듈별, 항목별 등과 같이 여러개의 시나리오로 분리하여 작성해야 한다
- 유스 케이스간 업무 흐름이 정상적인지 테스트할 수 있도록 작성
- 개발된 모듈 또는 프로그램 간의 연계가 정상적으로 동작하는 지 테스트할 수 있도록 작성해야 한다.

테스트 오라클 : 테스트 오라클은 테스트 결과가 올바른지 판단하기 위해 사전에 정의된 참 값을 대입하여 비교하는 기법 및 활동을 말한다. 기준을 말한다.

- 결과를 판단하기 위해 테스트 케이스에 대한 예상 결과를 계산하거나 확인
- 테스트 오라클의 특징

제한된 검증	테스트 오라클을 모든 테스트 케이스에 적용할 수 없다
수학적 기법	테스트 오라클의 값을 수학적 기법을 이용해 구할 수 있다.
자동화 가능	테스트 대상 프로그램의 실행, 결과 비교, 커버리지 측정 등을 자동화할 수 있다.

테스트 오라클의 종류

참(true) 오라클	○모든 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공하는 오라클로, 발생한 모든 오류를 검출할 수 있다. ○주로 항공기, 은행, 발전소 소프트웨어 등 미션 크리티컬한 업무에 사용된다.
샘플링 (Sampling) 오라클	○특정한 테스트 케이스의 입력 값들에 대해서만 기대하는 결과를 제공하는 오라클이다.
추정(Heuristic) 오라클	○샘플링 오라클을 개선한 오라클로, 특정 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공하고, 나머지 입력 값들에 대해서는 추정으로 처리하는 오라클이다.
일관성(Consistent) 오라클	○애플리케이션의 변경이 있을 때, 테스트 케이스의 수행 전과 후의 결과 값이 동일한지를 확인하는 오라클이다.

※샘플링 & 추정 오라클은 주로 일반적인 업무, 게임, 오락 등에 사용

섹션 61 테스트 자동화 도구

테스트 자동화는 사람이 반복적으로 수행하던 테스트 절차를 스크립트 형태로 구현하는 자동화 도구를 적용함으로써 쉽고 효율적으로 테스트를 수행할 수 있도록 한 것이다.

- 테스트 자동화 도구를 사용함으로써 휴먼 에러를 줄이고 테스트의 정확성을 유지하면서 테스트의 품질을 향상시킬 수 있다.

장/단점

장점	○테스트 데이터의 재입력, 재구성 같은 반복적인 작업을 자동화함으로써 인력 및 시간을 줄일 수 있음 ○다중 플랫폼 호환성, 소프트웨어 구성, 기본 테스트 등 향상된 테스트 품질을 보장한다. ○사용자 요구사항 등을 일관성 있게 검증 ○테스트 결과에 대한 객관적인 평가 기준을 제공 ○테스트 결과를 그래프등 다양한 표시 형태로 제공 ○UI가 없는 서비스도 정밀 테스트가 가능
단점	○테스트 자동화 도구의 사용 방법에 대한 교육 및 학습 필요 ○자동화 도구를 프로세스 단계별로 적용하기 위한 시간, 비용, 노력이 필요 ○비공개 사용 도구의 경우 고가의 추가 비용이 필요하다

테스트 자동화 도구의 유형 : 테스트를 수행하는 유형에 따라 다음과 같이 분류

정적 분석 도구 (Static Analysis Tools)	○프로그램을 실행하지 않고 분석하는 도구로, 소스 코드에 대한 코딩 표준, 코딩 스타일, 코드 복잡도 및 남은 결함등을 발견하기 위해 사용된다. ○테스트를 수행하는 사람이 작성된 소스코드를 이해하고 있어야만 분석이 가능 하다
테스트 케이스 생성 도구 (Test Case Generation Tools)	○자료 흐름도 : 자료 원시 프로그램을 입력받아 파싱한 후 자료 흐름도를 작성함 ○기능 테스트 : 주어진 기능을 구동시키는 모든 가능한 상태를 파악하여 이에 대한 입력 작성 ○입력 도메인 분석 : 원시 코드의 내부를 참조하지 않고, 입력 변수의 도메인을 분석하여 테스트 데이터를 작성함 ○랜덤 테스트 : 입력 값을 무작위로 추출하여 테스트함
테스트 실행 도구 (Test Execution Tools)	○스크립트 언어를 사용하여 테스트를 실행하는 방법으로, 테스트 데이터와 테스트 수행 방법 등이 포함된 스크립트를 작성 후 실행한다 ○데이터 주도 접근 방식 : 스프레드시트에 테스트 데이터를 저장하고, 이를 읽어들이는 실행 방식. 다양한 테스트 데이터를 동일한 테스트 케이스로 반복하여 실행할 수 있고, 스크립트에 익숙치 않은 사람도 테스트 데이터만 추가하여 테스트 할 수 있음 ○키워드 주도 접근 방식 : 스프레드시트에 테스트를 수행할 동작을 나타내는 키워드와 테스트 데이터를 함께 저장하여 실행하는 방식. 키워드를 이용해 여러 가지 테스트 케이스를 정의할 수 있음
성능 테스트 도구 (Performance Test Tools)	○애플리케이션의 처리량, 응답 시간, 경과 시간, 자원 사용률 등을 인위적으로 적용한 가상의 사용자를 만들어 테스트를 수행함으로써 선의 목표 달성 여부를 확인 한다.
테스트 통제 도구 (Test Control Tools)	○테스트 계획 및 관리, 테스트 수행, 결함 관리 등을 수행하는 도구로, 종류에는 형상 관리 도구, 결함 추적/관리 도구 등이 있다.
테스트 하네스 도구 (Test Harness Tools)	○테스트 하네스는 애플리케이션의 컴포넌트 및 모듈을 테스트하는 환경의 일부분으로, 테스트를 지원하기 위해 생성

테스트 하네스의 구성 요소

1. 테스트 드라이버 (Test Driver) : 테스트 대상의 하위 모듈을 호출하고, 매개 변수를 전달하고, 모듈 테스트 수행 후의 결과를 도출하는 도구
2. 테스트 스텝 (Test Stub) : 제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구로, 일시적으로 필요한 조건만 가지고 있는 테스트용 모듈
3. 테스트 스위트 (Test Suites) : 테스트 대상 컴포넌트나 모듈, 시스템에 사용되는 테스트 케이스의 집합
4. 테스트 케이스 (Test Case) : 사용자의 요구사항을 정확하게 준수했는지 확인하기 위한 입력값, 실행 조건, 기대 결과 등으로 만들어진 테스트 항목의 명세서
5. 테스트 스크립트 (Test Script) : 자동화된 테스트 실행 절차에 대한 명세서
6. 목 오브젝트 (Mock Object) : 사전에 사용자의 행위를 조건부로 입력해 두면, 그 상황에 맞는 예정된 행위를 수행하는 객체

테스트 수행 단계별 테스트 자동화 도구

테스트 단계	자동화 도구	설명
테스트 계획	요구사항 관리	사용자의 요구사항 정의 및 변경 사항 등을 관리하는 도구
테스트 분석 / 설계	테스트 케이스 생성	테스트 기법에 따른 테스트 데이터 및 테스트 케이스 작성을 지원하는 도구
테스트 수행	테스트 자동화	테스트의 자동화를 도와주는 도구로 테스트의 효율성을 높임
	정적 분석	코딩 표준, 런타임 오류 등을 검증하는 도구
	동적 분석	대상 시스템의 시뮬레이션을 통해 오류를 검출하는 도구
	성능 테스트	가상의 사용자를 생성하여 시스템의 처리 능력을 측정하는 도구
	모니터링	CPU, Memory 등과 같은 시스템 자원의 상태 확인 및 분석을 지원하는 도구
테스트 관리	커버리지 분석	테스트 완료 후 테스트의 충분성 여부 검증을 지원하는 도구
	형상 관리	테스트 수행에 필요한 다양한 도구 및 데이터를 관리하는 도구
	결함 추적/관리	테스트 시 발생한 결함 추적 및 관리 활동을 지원하는 도구

섹션 62 결함 관리

결함이란 오류 발생, 작동 실패 등과 같이 소프트웨어가 개발자가 설계한 것과 다르게 동작하거나 다른 결과가 발생하는 것

- 사용자가 예상한 결과와 실행 결과 간의 차이나 업무 내용의 불일치 등으로 인해 변경이 필요한 부분도 모두 결함에 해당한다

결함 관리 프로세스 : 애플리케이션 테스트에서 발견된 결함을 처리하는 것으로, 처리 순서는 다음과 같다

1. 결함 관리 계획 : 전체 프로세스에 대한 결함 관리 일정, 인력, 업무 프로세스 등을 확보하여 계획을 수립한다.
2. 결함 기록 : 테스터는 발견된 결함을 결함 관리 DB에 등록한다
3. 결함 검토 : 테스터, 프로그램 리더, 품질 관리 (QA) 담당자 등은 등록된 결함을 검토하고 결함을 수정할 개발자에게 전달한다
4. 결함 수정 : 개발자는 전달받은 결함을 수정한다.
5. 결함 재확인 : 테스터는 개발자가 수정한 내용을 확인하고 다시 테스트한다
6. 결함 상태 추적 및 모니터링 활동 : 결함 관리 DB를 이용하여 프로젝트별 결함 유형, 발생률 등을 한눈에 볼 수 있는 대시보드 또는 게시판 형태의 서비스를 제공한다.
7. 최종 결함 분석 및 보고서 작성 : 발견된 결함에 대한 정보와 이해관계자들의 의견이 반영된 보고서를 작성하고 결함 관리를 종료한다.

2~6이 결함 관리 DB에서 결함 상태 추적 및 모니터링 한다.

결함 상태 추적 : 테스트에서 발견된 결함은 지속적으로 상태 변화를 추적하고 관리해야 한다.

- 발견된 결함에 대해 결함 관리 측정 지표의 속성 값들을 분석하여 향후 결함이 발견될 모듈 또는 컴포넌트를 추정할 수 있다
- 결함 관리 측정 지표

결함 분포	모듈 또는 컴포넌트의 특정 속성에 해당하는 결함 수 측정
결함 추세	테스트 진행 시간에 따른 결함 수의 추이 분석
결함 에이징	특정 결함 상태로 지속되는 시간 측정

결함 추적 순서 : 결함이 발견된 때부터 결함이 해결될 때까지 전 과정을 추적하는 것으로 순서는 다음과 같다.

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 : 개발자가 결함 수정을 완료한 상태
5. 결함 조치 보류 (Deferred) : 결함의 수정이 불가능해 연기된 상태로, 우선순위, 일정 등에 따라 재오픈을 준비중인 상태
6. 결함 종료(Closed) : 결함이 해결되어 테스터와 품질 관리 담당자가 종료를 승인한 상태
7. 결함 해제 (Clarified) : 테스터, 프로그램 리더, 품질 관리 담당자가 종료 승인한 결함을 검토하여 결함이 아니라고 판명한 상태

결함 분류 : 시스템 결함, 기능 결함, GUI 결함, 문서 결함 등이 있다

테스트 단계별 유입 결함 : 기획 시 유입되는 결함, 설계 시 유입되는 결함, 코딩 시 유입되는 결함, 테스트 부족으로 유입되는 결함

결함 심각도 : High / Medium / Low로 구분

High : 핵심 요구사항 미구현, 장시간 시스템 다운 과 같이 더 이상 프로세스를 진행할 수 없도록 만드는 결함

Medium : 부정확한 기능이나 데이터베이스 에러 등과 같이 시스템 흐름에 영향을 미치는 결함

Low : 부정확한 GUI나 메시지, 에러 메시지 미출력, 화면상 문법 오류 등과 같이 시스템에는 영향이 없는 결함

결함 우선순위 : 발견된 결함 처리에 대한 신속성을 나타내는 척도

- 심각도가 높으면 보통 우선순위가 높으나 항상 그렇지 않다
- 우선순위는 Critical , High, Medium, Low 또는 즉시 해결, 주의 요망, 대기, 개선 권고 등으로 분류된다.

결함 관리 도구 :SW에 발생한 결함을 체계적으로 관리할 수 있도록 도와주는 도구

1. Mantis : SW설계 시 단위별 작업 내용을 기록할 수 있어 결함 추적 & 관리
2. Trac : 결함 추적과 더불어 결함을 통합하여 관리
3. Redmine : 프로젝트 관리 및 결함 추적
4. Bugzilla : 결함 신고, 확인, 처리 등 결함을 지속적으로 관리하는 도구, 결함의 심각도와 우선순위 지정 가능

섹션 63 애플리케이션 성능 분석

애플리케이션 성능 : 사용자가 요구한 기능을 최소한의 자원을 사용하여 최대한 많은 기능을 신속하게 처리하는 정도를 나타낸다.

애플리케이션 성능 측정 지표

처리량 (Throughput)	일정 시간내에 애플리케이션이 처리하는 일의 양
응답 시간 (Response Time)	애플리케이션에 전달한 시간부터 응답이 도착할때까지 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 등 자원 사용률

- 애플리케이션 성능 분석 도구는 애플리케이션의 성능을 테스트하는 도구와 시스템을 모니터링하는 도구로 분류된다.

성능 테스트 도구 : 애플리케이션의 성능을 테스트하기 위해 애플리케이션에 부하나 스트레스를 가하면서 애플리케이션의 성능 지표를 점검하는 도구이다.

도구명	도구 설명	지원 환경
Jmeter	HTTP,FTP 등 다양한 프로토콜을 지원하는 부하 테스트 도구	Cross-Platform
LoadUI	서버 모니터링, Drag&Drop 등 사용자의 편리성이 강화된 부하 테스트 도구. HTTP,JDBC 등 다양한 프로토콜 지원	Cross-Platform
OpenSTA	HTTP/S 프로토콜에 대한 부하 테스트 및 생산품 모니터링 도구	Windows

시스템 모니터링 도구 : 애플리케이션이 실행되었을 때 시스템 자원의 사용량을 확인하고 분석하는 도구

- 성능 저하의 원인 분석, 시스템 부하량 분석, 사용자 분석 등 시스템을 안정적으로 운영할 수 있는 기능을 제공한다.

도구명	도구 설명	지원 환경
Scouter	단일 뷰 통합/실시간 모니터링, 튜닝에 최적화된 인프라 통합 모니터링 도구. 애플리케이션의 성능을 통제	Cross-Platform
Zabbix	웹기반 서버, 서비스, 애플리케이션 등의 모니터링 도구	Cross-Platform

애플리케이션 성능 저하 원인 분석 : 애플리케이션의 성능 저하 현상은 애플리케이션을 DB에 연결하기 위해 Connection 객체를 생성하거나 쿼리를 실행하는 애플리케이션 로직에서 많이 발생한다.

:다음은 애플리케이션의 성능 저하를 발생시키는 주요 요인이자.

- DB에 필요 이상의 많은 데이터를 요청한 경우
- 데이터베이스의 락(DB lock)이 해제되기를 기다리면서 애플리케이션이 대기하거나 타임아웃 된 경우
- 커넥션 풀(Connection Pool)의 크기를 너무 작거나 크게 설정한 경우
- JDBC나 ODBC 같은 미들웨어를 사용한 후 종료하지 않아 연결 누수(Connection Leak)가 발생한 경우
- 트랜잭션이 확정(Commit)되지 않고 커넥션 풀에 반환되거나, 잘못 작성된 코드로 인해 불필요한 Commit이 발생한 경우
- 인터넷 접속 불량으로 인해 서버 소켓에 쓰기는 지속되나 클라이언트에서 정상적인 읽기가 수행되지 않는 경우
- 대량의 파일을 업로드하거나 다운로드하여 처리 시간이 길어진 경우
- 트랜잭션 처리 중 외부 호출이 장시간 수행되거나 타임아웃된 경우
- 네트워크 관련 장비 간 데이터 전송이 실패하거나 전송 지연으로 인해 데이터 손실이 발생한 경우

섹션 64 복잡도

시스템이나 시스템 구성 요소 또는 소프트웨어의 복잡한 정도를 나타내는 말로, 시스템 또는 소프트웨어를 어느 정도의 수준까지 테스트해야 하는지 또는 개발하는데 어느 정도의 자원이 소요되는지 예측하는데 사용된다

- 시스템의 복잡도가 높으면 장애가 발생할 수 있으므로 정밀한 테스트를 통해 미리 오류를 제거할 필요가 있다
- 주요 복잡도 측정 방법에는 LOC(Line of Code), 순환 복잡도 (Cyclomatic Complexity) 등이 있다.

시간 복잡도 : 알고리즘의 실행시간, 즉 알고리즘을 수행하기 위해 프로세스가 수행하는 연산 횟수를 수치화한 것을 의미한다

- 시간 복잡도가 낮을수록 알고리즘의 실행시간이 짧고, 높을수록 실행시간이 김
- 시간 복잡도는 알고리즘의 실행시간이 하드웨어적 성능이나 프로그래밍 언어의

종류에 따라 달라지기 때문에 시간이 아닌 명령어의 실행 횟수를 표기하는데, 이러한 표기법을 점근 표기법이라 한다.

점근 표기법 종류

빅오 표기법 (Big-O Notation)	알고리즘의 실행시간이 최악일 때를 표기하는 방법이다. 입력값에 대해 알고리즘을 수행했을 때 명령어의 실행 횟수는 어떠한 경우에도 표기 수치보다 많을 수 없다.
세타 표기법 (Big-θ Notation)	알고리즘의 실행시간이 평균일 때를 표기하는 방법이다. 입력값에 대해 알고리즘을 수행했을 때 명령어의 실행 횟수는 평균적인 수치를 표기한다.
오메가 표기법 (Big-Ω Notation)	알고리즘의 실행시간이 최상일 때를 표기하는 방법이다. 입력값에 대해 알고리즘을 수행했을 때 명령어의 실행 횟수는 어떠한 경우에도 표기 수치보다 적을 수 없다.

빅오 표기법 : 신뢰성이 떨어지는 오메가 표기법이나 평가하기 까다로운 세타 표기법에 비해 성능을 예측하기 용이하여 주로 사용된다.

$O(1)$	입력값 n 에 관계 없이 일정하게 문제 해결에 하나의 단계만을 가짐 ex) 스택의 푸시와, 팝
$O(\log_2 n)$	문제 해결에 필요한 단계가 입력값 n 또는 조건에 의해 감소 ex) 이진 트리, 이진 검색 등
$O(n)$	문제 해결에 필요한 단계가 입력값 n 과 1:1의 관계를 가진다 ex) for문
$O(n \log_2 n)$	문제 해결에 필요한 단계가 $n(\log_2 n)$ 번 만큼 수행된다. ex) 힙 정렬, 2-Way 합병 정렬
$O(n^2)$	문제 해결에 필요한 단계가 입력값 n 의 제곱만큼 수행한다 ex) 삽입 정렬, 쉘 정렬, 선택 정렬, 버블 정렬, 퀵 정렬
$O(2^n)$	문제 해결에 필요한 단계가 2의 입력값 n 제곱만큼 수행된다 ex) 피보나치 수열

순환 복잡도 : 한 프로그램의 논리적인 복잡도를 측정하기 위한 소프트웨어의 복잡도를 측정하기 위한 소프트웨어 척도로, 맥케이브 순환도 (McCabe's Cyclomatic) 또는 맥케이브 복잡도 메트릭(McCabe's Complexity Metrix)라고도 하며, 제어 흐름도 이론에 기초를 준다.

- 순환 복잡도를 이용하여 계산된 값은 프로그램의 독립적인 경로의 수를 정의하고, 모든 경로가 한 번 이상 수행되었음을 보장하기 위해 행해지는 테스트 횟수의

상한선을 제공한다.

- 제어 흐름도 G 에서 순환 복잡도 $V(G)$ 는 다음과 같은 방법으로 계산할 수 있다.

방법 1 : 순환 복잡도는 제어 흐름도의 영역 수와 일치하므로 영역 수를 계산한다.
방법 2 : $V(G) = E - N + 2$: E 는 화살표 수, N 은 노드의 수

섹션 65 애플리케이션 성능 개선

소스 코드 최적화 : 소스 코드 최적화는 나쁜 코드를 배제하고, 클린 코드로 작성하는 것이다.

클린 코드(Clean Code) : 누구나 쉽게 이해하고 수정 및 추가할 수 있는 단순, 명료한 코드, 즉 잘 작성된 코드를 의미한다

나쁜 코드(Bad Code) :

- 프로그램의 로직이 복잡하고 이해하기 어려운 코드로, 스파게티 코드와 외계인 코드가 여기에 해당한다.
- 스파게티 코드(Spaghetti Code) : 코드의 로직이 서로 복잡하게 얽혀있는 코드
- 외계인 코드(Alien Code) : 아주 오래되거나 참고문서 또는 개발자가 없어 유지 보수 작업이 어려운 코드
- 나쁜 코드로 작성된 애플리케이션의 코드를 클린 코드로 수정하면 애플리케이션의 성능이 개선된다.

클린 코드 작성 원칙(★)

가독성	○누구든지 코드를 쉽게 읽을 수 있도록 작성한다. ○코드 작성 시 이해하기 쉬운 용어를 사용하거나 들여쓰기 기능 사용
단순성	○코드를 간단하게 작성한다. ○한 번에 한 가지를 처리하도록 코드를 작성하고 클래스/메소드/함수 등을 최소 단위로 분리한다.
의존성	○코드가 다른 모듈에 미치는 영향을 최소화한다.
배제	○코드 변경 시 다른 부분에 영향이 없도록 작성한다
중복성	○코드의 중복을 최소화한다.
최소화	○중복된 코드는 삭제하고 공통된 코드를 사용한다
추상화	○상위 클래스/메소드/함수에서는 간략하게 어플리케이션의 특성을 나타내고, 상세 내용은 하위 클래스/메소드/함수에서 구현한다.

소스코드 최적화 유형

- 1. 클래스 분할 배치 : 하나의 클래스는 하나의 역할만 수행하도록 응집도를 높이고, 크기를 작게 작성
- 2. 느슨한 결합(Loosely Coupled) : 인터페이스 클래스를 이용하여 추상화된 자료 구조와 메소드를 구현함으로써 클래스 간의 의존성을 최소화 한다.
- 3. 코딩 형식 준수 : 코드 작성 시 다음 형식을 준수한다.
 - 줄 바꿈 사용
 - 개념적 유사성이 높은 종속 함수 사용
 - 호출하는 함수는 선행치, 호출되는 함수는 후배치
 - 지역 변수는 각 함수의 맨 처음에 선언

- 4. 좋은 이름 사용 : 변수나 함수 등의 이름 기억하기 좋고, 발음이 쉽고, 접두어 사용 등 기본적인 이름 명명 규칙을 정의하고 규칙에 맞는 이름을 사용한다
- 5. 적절한 주석문 사용 : 소스 코드 작성 시 앞으로 해야할 일을 기록하거나 중요한 코드를 강조할 때 주석문을 사용

소스 코드 품질 분석 도구(★) : 소스 코드의 코딩 스타일, 코드에 설정된 코딩 표준, 코드의 복잡도, 코드에 존재하는 메모리 누수 현상, 스레드 결함등을 발견하기 위해 사용하는 분석도구로, 크게 정적 분석 도구와 동적 분석 도구로 나뉜다.

정적 분석 도구 :

- 작성한 소스 코드를 실행하지 않고 코딩 표준이나 코딩 스타일, 결함 등을 확인하는 코드 분석 도구이다.
- 비교적 애플리케이션 개발 초기의 결함을 찾는데 사용되고, 개발 완료 시점에서는 개발된 소스 코드의 품질을 검증하는 차원에서 사용된다
- 자료 흐름이나 논리 흐름을 분석하여 비정상적인 패턴을 찾을 수 있다.
- 동적 분석 도구로는 발견하기 어려운 결함을 찾아내고, 소스 코드에서 코딩의 복잡도, 모델 의존성, 불일치성 등을 분석할 수 있다.

동적 분석 도구 :

- 작성한 소스 코드를 실행한 코드에 존재하는 메모리 누수, 스레드 결함등을 분석하는 도구이다.

도구	설명	지원 환경
pmd	소스 코드에 대한 미사용 변수, 최적화되지 않은 코드 등 결함을 유발할 수 있는 코드를 검사한다.	Linux, windows
cppcheck	C/C++ 코드에 대한 메모리 누수, 오버플로우 분석	windows
SonarQube	중복 코드, 복잡도, 코딩 설계 등을 분석하는 소스 분석 통합 플랫폼	Cross-Platform
checkstyle	자바 코드에 대해 소스 코드 표준을 따르는 지 검사 다양한 개발 도구에 통합하여 사용 가능	Cross-Platform
ccm	다양한 언어의 코드 복잡도를 분석한다	Cross-Platform
cobertura	자바 언어의 소스 코드 복잡도 분석 및 테스트 커버리지를 측정한다	Cross-Platform

Avalanche	Valgrind 프레임워크 및 STP 기반으로 구현된다 프로그램에 대한 결함 및 취약점 등을 분석한다	Linux, Andoid
Valgrind	프로그램 내에 존재하는 메모리 및 스레드 결함을 분석한다.	Cross-Platform

5장 인터페이스 구현

섹션 66 모듈 간 공통 기능 및 데이터 인터페이스 확인 (C)

- 공통 기능은 모듈의 기능 중에서 공통적으로 제공되는 기능
- 데이터 인터페이스는 모듈 간 교환되는 데이터가 저장될 파라미터를 의미한다.
- 모듈 간 공통 기능 및 데이터 인터페이스는 인터페이스 설계서에서 정의한 모듈의 기능을 기반으로 확인한다
- 확인된 공통 기능 및 데이터 인터페이스는 모듈 간 연계가 필요한 인터페이스의 기능을 식별하는데 사용된다.
- 모듈 간 공통 기능 및 데이터 확인 순서
 - 1. 인터페이스 설계서를 통해 모듈별 기능 확인
 - 2. 외부 및 내부 모듈을 기반으로 공통적으로 제공되는 기능과 각 데이터의 인터페이스 확인

인터페이스 설계서

: 시스템 사이의 데이터 교환 및 처리를 위해 교환 데이터 및 관련 업무, 송·송신 시스템 등에 대한 내용을 정의한 문서이다,

- 인터페이스 설계서는 일반적인 형태의 설계서와 정적·동적 모형을 통한 설계서로 구분된다.

일반적인 인터페이스 설계서 : 시스템의 인터페이스 목록, 각 인터페이스의 상세 데이터 명세, 각 기능의 세부 인터페이스 정보를 정의한 문서이다,

- 이는 또 시스템 인터페이스 설계서와 상세 기능별 인터페이스 명세서로 구분된다,

○ 시스템 인터페이스 설계서 : 시스템 인터페이스 목록을 만들고 각 인터페이스 목록에 대한 상세 데이터 명세를 정의

○ 상세 기능별 인터페이스 명세서 : 각 기능의 세부 인터페이스 정보를 정의한 문서. 인터페이스를 통한 각 세부 기능의 개요, 세부 기능이 동작하기 전에 필요한 사전/사후 조건, 인터페이스 데이터, 호출 이후 결과를 확인하기 위한 반환값 등으로 구성된다.

정적·동적 모형을 통한 설계서 : 정적·동적 모형으로 각 시스템의 구성 요소를 표현한 다이어그램을 이용하여 만든 문서이다.

- 시스템을 구성하는 주요 구성 요소 간의 트랜잭션을 통해 해당 인터페이스가 시스템 어느 부분에 속하고, 해당 인터페이스를 통해 상호 교환되는 트랜잭션의 종류를 확인할 수 있다.

인터페이스 설계서별 모듈 기능 확인

: 인터페이스 설계서에서 정의한 모듈을 기반으로 각 모듈의 기능 확인

- 시스템 인터페이스 목록에서 송신 및 전달부분은 외부 모듈, 수신 부분은 내부 모듈에 해당된다,

- 상세 기능 인터페이스 명세서에서 오퍼레이션과 사전 조건은 외부 모듈, 사후 조건은 내부 모듈에 해당된다.

- 정적·동적 모형을 통한 인터페이스 설계에서 인터페이스 영역을 기준으로 상위 모듈, 하위 모듈이 내부모듈에 해당 된다.

모듈 간 공통 기능 및 데이터 인터페이스 확인 :

1. 내 · 외부 모듈 기능을 통해 공통적으로 제공되는 기능을 확인한다.

2. 내·외부 모듈 기능과 공통 기능을 기반으로 필요한 데이터 인터페이스 항목을 확

인한다.

섹션 67 모듈 연계를 위한 인터페이스 기능 식별

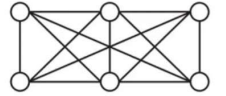
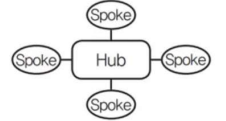
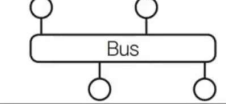
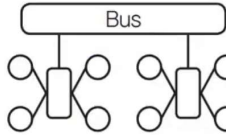
모듈 연계의 개요 : 모듈 연계는 내부 모듈과 외부 모듈 또는 내부 모듈 간 데이터 교환을 위해 관계를 설정하는 것으로, 대표적으로 EAI방법과 ESB 방식이 있다.

EAI(Enterprise Application Integration) :

- EAI는 기업 내 각종 애플리케이션 및 플랫폼 간의 정보 전달, 연계, 통합 상호 연동이 가능하게 해주는 솔루션이다.

- EAI는 비즈니스 간 통합 및 연계성을 증대시켜 효율성 및 각 시스템 간의 확장성(Determinacy)을 높여 준다.

- EAI 구축 유형은 다음과 같다.

유형	기능	모형
Point-to-Point	<ul style="list-style-type: none"> 가장 기본적인 애플리케이션 통합 방식 애플리케이션을 1:1로 연결함 변경 및 재사용이 어려움 	
Hub & Spoke	<ul style="list-style-type: none"> 단일 접점인 허브 시스템을 통해 데이터를 전송하는 중앙 집중형 방식 확장 및 유지 보수가 용이함 허브 장애 발생 시 시스템 전체에 영향을 미침 	
Message Bus (ESB 방식)	<ul style="list-style-type: none"> 애플리케이션 사이에 미들웨어를 두어 처리하는 방식 확장성이 뛰어나며 대용량 처리가 가능함 	
Hybrid	<ul style="list-style-type: none"> Hub & Spoke와 Message Bus의 혼합 방식 그룹 내에서는 Hub & Spoke 방식을, 그룹 간에는 Message Bus 방식을 사용함 필요한 경우 한 가지 방식으로 EAI 구현이 가능함 데이터 병목 현상을 최소화할 수 있음 	

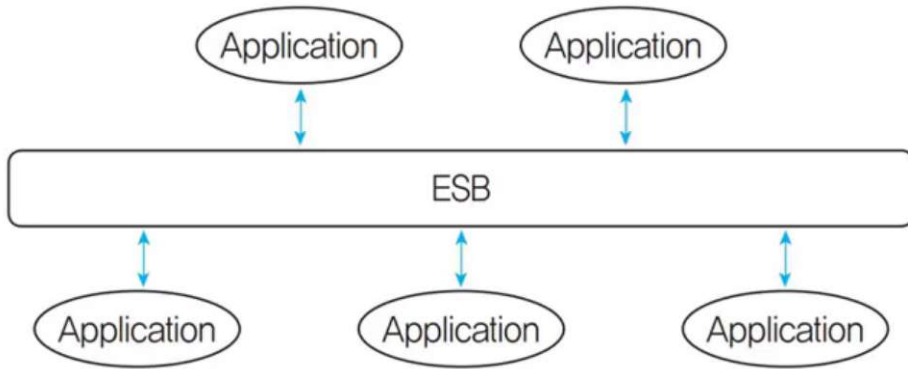
ESB(Enterprise Service Bus) : ESB는 애플리케이션 간 연계, 데이터 변환, 웹 서비스 지원 등 표준 기반의 인터페이스를 제공하는 솔루션이다.

- ESB는 애플리케이션 통합 측면에서 EAI와 유사하지만 애플리케이션 보다는 서비스 중심의 통합을 지향한다.

- ESB는 특정 서비스에 국한되지 않고, 범용적으로 사용하기 위하여 애플리케이션

선과의 결합도(Coupling)를 약하게 유지한다.

- 관리 및 보안 유지가 쉽고 높은 수준의 품질 지원이 가능하다.



모듈 간 연계 기능 식별

- 모듈 간 공통 기능 및 데이터 인터페이스를 기반으로 모듈과 연계된 기능을 시나리오 형태로 구체화하여 식별한다
- 식별된 기능은 인터페이스 기능을 식별하는 데 사용된다.
- 외부/내부로 크게 나누고, 기능 별로 다시 나눠 시나리오 작성

모듈 간 인터페이스 기능 식별

- 식별된 모듈 간 관련 기능을 검토하여 인터페이스 동작에 필요한 기능 식별
- 식별된 인터페이스 기능은 인터페이스 기능 구현을 정의하는데 사용

섹션 68 모듈 간 인터페이스 데이터 표준 확인 (D)

인터페이스 데이터 표준은 모듈 간 인터페이스에 사용되는 데이터의 형식을 표준화하는 것이다.

- 인터페이스 데이터 표준은 기존의 데이터 중에서 공통 영역을 추출하거나 어느 한쪽의 데이터를 변환하여 정의한다.
- 확인된 인터페이스 데이터 표준은 인터페이스 기능 구현을 정의하는데 사용

1. 데이터 인터페이스 확인

- 데이터 표준을 위해 식별된 데이터 인터페이스에서 입·출력값의 의미와 데이터의 특성등을 구체적으로 확인한다.

- 확인된 데이터 인터페이스의 각 항목을 통해 데이터 표준을 확인한다

2. 인터페이스 기능 확인

- 데이터 표준을 위해 식별된 인터페이스 기능을 기반으로 인터페이스 기능 공부를 위해 필요한 데이터 항목을 확인한다,
- 확인된 데이터 항목과 데이터 인터페이스에서 확인된 데이터 표준에서 수정·추가·삭제될 항목이 있는지 확인 한다.

3. 인터페이스 데이터표준 확인

- 데이터 인터페이스에서 확인된 데이터 표준과 인터페이스 기능을 통해 확인된 데이터 항목들을 검토하여 최종적으로 데이터 표준을 확인한다.
- 확인된 데이터 표준은 항목별로 데이터 인터페이스와 인터페이스 기능 중 출처를 구분하여 기록한다.

섹션 69 인터페이스 기능 구현 정의 (C등급)

개요 : 인터페이스 기능 구현의 정의는 인터페이스를 실제로 구현하기 위해 인터페이스 기능에 얼마나 구현방법을 기능별로 기술한 것이다,

- 인터페이스 기능 구현 정의 순서

1. 컴포넌트 명세서를 확인한다.
2. 인터페이스 명세서를 확인한다.
3. 일관된 인터페이스 기능 구현을 정의한다.
4. 정의된 인터페이스 기능 구현을 정형화한다.

모듈 세부 설계서 : 모듈의 구성 요소와 세부적인 동작 등을 정의한 설계서

- 대표적인 모듈 세부 설계서에는 컴포넌트 명세서와 인터페이스 명세서가 있다.

컴포넌트 명세서 : 컴포넌트의 개요 및 내부 클래스의 동작, 인터페이스를 통해 외부와 통신하는 명세 등을 정의한 것

인터페이스 명세서 : 컴포넌트 명세서의 항목 중 인터페이스 클래스의 세부 조건 및 기능 등을 정의한 것

모듈 세부 설계서 확인

- 각 모듈의 컴포넌트 명세서와 인터페이스 명세서를 기반으로 인터페이스에 필요한 기능을 확인
- 컴포넌트 명세서의 컴포넌트 개요, 내부 클래스의 클래스 명과 설명 등을 통해 컴포넌트가 가지고 있는 주요 기능을 확인한다.
- 컴포넌트 명세서의 인터페이스 클래스를 통해 인터페이스에 필요한 주요 기능 확인
- 인터페이스 명세서를 통해 컴포넌트 명세서의 인터페이스 클래스에 명시된 인터페이스의 세부 조건 및 기능을 확인한다.

인터페이스 기능 구현 정의

- 인터페이스의 기능, 인터페이스 데이터 표준, 모듈 세부 설계서를 기반으로 일관성 있고 정형화된 인터페이스 기능 구현에 대해 정의한다.
- 일관성 있는 인터페이스 기능 구현 정의
- 정의된 인터페이스 기능 구현 정형화

p308

섹션 70 인터페이스 구현

인터페이스 구현은 송·수신 시스템 간의 데이터 교환 및 처리를 실현해 주는 작업

- 정의된 인터페이스 기능 구현을 기반으로 구현 방법 및 범 등을 고려하여 인터페이스 구현 방법을 분석한다.
- 분석된 인터페이스 구현 정의를 기반으로 인터페이스를 구현한다.
- 인터페이스를 구현하는 대표적인 방법에는 데이터 통신을 이용한 방법과 인터페이스 엔티티를 이용한 방법이 있다.

데이터 통신을 이용한 인터페이스 구현

: 애플리케이션 영역에서 인터페이스 형식에 맞춘 데이터 포맷을 인터페이스 대상으로 전송하고 이를 수신 측에서 파싱(Parsing)하여 해석하는 방식이다.

인터페이스 엔티티를 이용한 인터페이스 구현

: 인터페이스가 필요한 시스템 사이에 별도의 인터페이스 엔티티를 두어 상호 연계하는 법

- 일반적으로 인터페이스 테이블을 엔티티로 활용한다
- 인터페이스 테이블은 한 개 또는 송신 및 수신 인터페이스 테이블을 각각 두어 활용한다.
- 송신 및 수신 인터페이스 테이블의 구조는 대부분 같지만 상황에 따라 서로 다르게 설계할 수 있다.

JSON/XML/AJAX : 데이터 통신을 이용한 인터페이스 구현에 쓰이는 데이터 포맷

1. **JSON (JavaScript Object Nation)** : 속성-값 쌍으로 이루어진 데이터 객체를 전달하기 위해 사람이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷

- 비동기 처리에 사용되는 AJAX에서 XML을 대체하여 사용

2. **XML(eXtensible Markup Language)** : 특수한 목적을 갖는 마크업 언어를 만드는데 사용되는 다목적 마크업 언어

- 웹 페이지의 기본 형식인 HTML의 문법이 각 웹 브라우저에서 상호 호환적이지 못하는 문제와 SGML의 복잡함을 해결하기 위해 개발

3. **AJAX(Asynchronous JavaScript and XML)** : 자바스크립트를 이용하여 클라이언트와 서버 간에 XML 데이터를 교환 및 제어함으로써 이용자가 웹 페이지와 자유롭게 상호 작용할 수 있도록 하는 비동기 통신 기술

섹션 71 인터페이스 예외 처리 (C)

구현된 인터페이스가 동작하는 과정에서 기능상 예외 상황이 발생 했을 때 이를 처리하는 절차를 말한다.

-인터페이스 예외처리는 구현방법에 따라 데이터 통신을 이용한 방법과 인터페이스 엔티티를 이용하는 방법이 있다.

데이터 통신을 이용한 인터페이스 예외 처리 : JSON, XML 등 인터페이스 객체를 이용해 구현한 인터페이스 동작이 실패할 경우를 대비한 것으로, 인터페이스 객체의 송·수신 시 발생할 수 있는 예외 케이스를 정의하고 각 예외 케이스마다 예외 처리 방법을 기술한다.

- 시스템 환경, 송·수신 데이터, 프로그램 자체 원인 등으로 예외가 발생한다.

인터페이스 엔티티를 이용한 인터페이스 예외 처리 : 인터페이스 동작이 실패할 경우를 대비하여 해당 엔티티에 인터페이스의 실패 상황과 원인 등을 기록하고, 이에 대한 조치를 취할 수 있도록 사용자 및 관리자에서 알려주는 방식으로 예외 처리 방법을 정의한다.

- 송수신 테이블이나 인터페이스 테이블 통한 예외처리가 발생된다.

섹션 72 인터페이스 보안

- 인터페이스는 시스템 모듈 간 통신 및 정보 교환을 위한 통로로 사용되므로 충분한 보안 기능을 갖추지 않으면 시스템 모듈 전체에 악영향을 주는 보안 취약점이 될 수 있다.
- 인터페이스의 보안성 향상을 위해서는 인터페이스의 보안 취약점을 분석한 후 적절한 보안 기능을 적용한다.

인터페이스 보안 취약점 분석

- 인터페이스 기능이 수행되는 각 구간들의 구현 현황을 확인하고 각 구간에 어떤 보안 취약점이 있는지를 분석한다.
- 인터페이스 기능이 수행되는 각 구간들의 구현 현황은 송·수신 영역의 구현 기술 및 특징 등을 구체적으로 확인 한다
- 확인된 인터페이스 기능을 기반으로 송신 데이터 선택, 송신 객체 생성, 인터페이스 송·수신, 데이터 처리 결과 전송 등 영역별로 발생할 수 있는 보안 취약점을 시나리오 형태로 작성한다

인터페이스 보안 기능 적용

- 분석한 인터페이스 기능과 보안 취약점을 기반으로 인터페이스 보안 기능을 적용
- 인터페이스 보안 기능은 일반적으로 네트워크, 애플리케이션, 데이터베이스 영역에 적용한다.

1. 네트워크 영역

- 인터페이스 송·수신 간 스니핑(Sniffing) 등을 이용한 데이터 탈취 및 변조 위협을 방지하기 위해 네트워크 트래픽에 대한 암호화를 설정한다.
- 암호화는 인터페이스 아키텍처에 따라 IPSec, SSL, S-HTTP 등 다양한 방식으로 적용한다.

2. 애플리케이션 영역

- 소프트웨어 개발 보안 가이드를 참조하여 애플리케이션 코드 상의 보안 취약점을 보완하는 방향으로 애플리케이션 보안 기능을 적용한다.

3. 데이터베이스 영역

- 데이터베이스, 스키마, 엔티티의 접근 권한과 프로시저, 트리거 등 데이터베이스 동작 객체의 보안 취약점에 대해 보안 기능 적용
- 개인 정보나 업무상 민감한 데이터의 경우 암호화나 익명화 등 데이터 자체의 보안 방안도 고려한다.

데이터 무결성 검사 도구

- 데이터 무결성 검사 도구는 시스템 파일의 변경 유무를 확인하고, 파일이 변경되었을 경우 이를 관리자에게 알려주는 도구로, 인터페이스 보안 취약점을 분석하는데 사용된다.
- 크래커나 허가받지 않은 내부 사용자들이 시스템에 침입하면 백도어를 만들어 놓거나 시스템 파일을 변경하여 자신의 흔적을 감추는데, 무결성 검사 도구를 이용해 이를 감지할 수 있다.
- 해시 함수를 이용하여 현재 파일 및 디렉토리의 상태를 DB에 저장한 후 감시하다가 현재 상태와 DB의 상태가 달라지면 관리자에게 변경 사실을 알려줌
- 대표적인 도구로 Tripwire, AIDE, Samhain, Claymore, Slipwire, Fcheck 등이 있음

섹션 73 연계 테스트 (C) p317

: 연계 테스트는 구축된 연계 시스템가 연계 시스템의 구성 요소가 정상적으로 동작하는지 확인하는 활동이다.

- 연계 테스트는 연계 테스트 케이스 작성, 연계 테스트 환경 구축, 연계 테스트 수행, 연계 테스트 수행 결과 검증 순으로 진행된다.

※연계 시스템의 구성 요소에는 송·수신 모듈, 연계 서버, 모니터링 현황 등이 있음

1. 연계 테스트 케이스 작성 : 연계 시스템 간의 데이터 및 프로세스 흐름을 분석하여 필요한 테스트 항목을 도출하는 과정
2. 연계 테스트 환경 구축 : 테스트의 일정, 방법, 절차, 소요 시간 등을 송·수신 기관과의 협의를 통해 결정

- 3.연계 테스트 수행 : 연계 응용 프로그램을 실행하여 연계 테스트 케이스의 시험 항목 및 처리 절차 등을 실제로 진행
- 4.연계 테스트 수행 결과 검증 : 연계 테스트 케이스의 시험 항목 및 처리 절차를 수행한 결과가 예상 결과와 동일한지 확인

섹션 74 인터페이스 구현 검증 (☆)

- 인터페이스가 정상적으로 문제없이 작동하는 지 확인하는 것
- 인터페이스 구현 검증 도구와 감시 도구를 이용해 동작 상태를 확인

인터페이스 구현 검증 도구(☆)

- 인터페이스 구현 검증하기 위해서는 인터페이스 단위 기능과 시나리오 등을 기반으로 하는 통합 테스트가 필요하다.
- 통합 테스트는 다음과 같은 테스트 자동화 도구를 이용하면 효율적으로 수행할 수 있다.

도구	기능
xUnit	○같은 테스트 코드를 여러번 작성하지 않게 도와주고, 테스트마다 예상 결과를 기억할 필요가 없게 하는 자동화된 해법을 제공하는 단위 테스트 프레임워크이다. ○Smalltalk에 처음 적용되어 SUnit이라는 이름이었으나 Java용의 JUnit, C++용의 Cppunit, .NET용의 NUnit, Http용의 HttpUnit 등 다양한 언어에 적용되면서 xUnit으로 총칭하고 있다.
STAF	○서비스 호출 및 컴포넌트 재사용등 다양한 환경을 지원하는 테스트 프레임워크이다. ○크로스 플랫폼, 분산 소프트웨어 테스트 환경을 조성하도록 지원. ○분산 소프트웨어인 경우 각 분산 환경에 설치된 데몬이 프로그램 테스트에 대한 응답을 대신하며, 테스트가 완료되면 이를 통합하고 자동화하여 프로그램을 완성함
FitNesse	웹 기반 테스트케이스 설계, 실행, 결과 확인 등을 지원하는 테스트 프레임워크
NTAF	Fitnesse의 장점인 협업 기능과 STAF의 장점인 재사용 및 확장성을 통합한 NHN(Naver)의 테스트 자동화 프레임워크
Selenium	다양한 브라우저 및 개발 언어를 지원하는 웹 어플리케이션 테스트 프레임워크
watir	Ruby를 사용하는 애플리케이션 테스트 프리임워크

인터페이스 구현 감시 도구 (검증 도구와 감시 도구의 차이)

- 인터페이스 동작 상태는 APM을 사용하여 감시(Monitoring)할 수 있다.
- 애플리케이션 성능 관리 도구를 통해 데이터베이스와 웹 애플리케이션의 트랜잭션, 변수값, 호출 함수, 로그 및 시스템 부하 등 종합적인 정보를 조회하고 분석할 수 있다.
- 대표적인 애플리케이션 성능 관리 도구에는 스카우터(Scouter), 제니퍼(Jennifer)등이 있다.

- ※APM : (Applicdation Performance Management/Monitoring) : 애플리케이션의 성능관리를 위해 접속자, 자원 현황, 트랜잭션 수행 내역, 장애 진단 등 다양한 모니터링 기능을 제공하는 도구
- 리소스 방식 : Nagios, Zabbix, Cacti 등
 - End-to-End 방식 : VisualVM, 제니퍼, 스카우트

인터페이스 구현 검증 도구 및 감시 도구 선택

- 인터페이스 기능 구현 정의를 통해 구현된 인터페이스 명세서의 세부 기능을 참조하여 인터페이스의 정상적인 동작 여부를 확인하기 위한 검증 도구와 감시 도구의 요건을 분석한다.
- 분석이 끝나면 시장 및 솔루션 조사를 통해서 적절한 인터페이스 구현을 검증하고 감시하는데 필요한 인터페이스 구현 검증 도구와 감시 도구를 선택한다.

인터페이스 구현 검증 확인

- 도구를 이용해 외부 시스템과 연계 모듈의 동작 상태 확인
- 예측값과 실제 검증감시 일치하는지 확인
- 추가적으로 각 단계별 오류 처리도 적절하게 구현되어 있는지 확인

인터페이스 구현 감시 확인

- 도구를 이용해 외부 시스템과 연결 모듈이 서비스를 제공하는 동안 정상적으로 동작하는지 확인
- 인터페이스 동작 여부, 에러 발생 여부 등 감시 도구에서 제공해 주는 리포트를 활용

섹션 75 인터페이스 오류 확인 및 처리 보고서 작성 (D등급)

- 인터페이스는 독립적으로 떨어져 있는 시스템 간 연계를 위한 기능이므로 인터페이스에서 발생하는 오류는 대부분 중요한 오류이다.
- 인터페이스 오류 발생 시 사용자 또는 관리자는 오류사항을 확인하고 오류 처리 보고서를 작성하여 보고 체계에 따라 관리 조직에 보고해야 한다.
- 인터페이스 오류 확인 방법에는 오류 발생 즉시 확인하는 방법과 주기적인 확인 방법이 있다.

1. 인터페이스 오류 발생 즉시 확인

- 인터페이스 오류가 발생하면 오류 메시지를 표시하고 자동으로 SMS, 이메일을 발송하므로 즉시 오류 발생을 확인할 수 있다.
- 인터페이스 오류 발생을 즉시 처리하는 것은 가장 직관적인 방법이기 때문에 □ 가장 많이 사용되고 있다.

2. 주기적인 인터페이스 오류 발생 확인

- 시스템 관리자가 시스템의 현재 상태를 보여주는 시스템 로그나 인터페이스 오류 관련 테이블 등을 통해 주기적인 오류 발생 여부를 확인한다.
- 발생하는 오류에 대한 정보가 주기적으로 축적되면 오류의 재발을 방지할 수 있는 계획을 세울 수 있다.

인터페이스 오류 처리 보고서 작성 : 인터페이스 작동 시 발생하는 오류의 발생 및 종료 시점, 원인 및 증상, 처리사항 등을 정리한 문서

- 정형화된 형식이 없어 조직의 체계에 따라 작성
- 최초 발생 시/ 오류 처리 경과 시/ 완료 시에 따라 기록함