시험에 나오는 것만 공부한다!

2024 시나공

기출문제집

정보처리기사

필기

16,17 테스트 (White & Black)

길벗알앤디 지음
(강윤석, 김용갑, 김우경, 김종일)

길벗

2과목 소프트웨어 개발

23.7, 22.3, 21.8, 21.3 2403602

핵심 073 자료 구조의 분류

- 선형 구조 (Linear Structure)
  - 배열(Array)
  - 선형 리스트(Linear List)
    - 연속 리스트 (Contiguous List)
    - 연결 리스트 (Linked List)
  - 스택(Stack)
  - 큐(Queue)
  - 데크(Deque)
- 비선형 구조 (Non-Linear Structure)
  - 트리(Tree)
  - 그래프(Graph)

22.7 2403604

핵심 074 선형 리스트(Linear List)

빈 공간 없이 차례차례 데이터가 저장된다.

연속 리스트(Contiguous List)

- 연속 리스트는 배열과 같이 연속되는 기억장소에 저장되는 자료 구조이다.
- 연속 리스트는 기억장소를 연속적으로 배정받기 때문에 기억장소 이용 효율은 밀도가 1로서 가장 좋다.
- 연속 리스트는 중간에 데이터를 삽입하기 위해서는 연속된 빈 공간이 있어야 하며, 삽입 · 삭제 시 자료의 이동이 필요하다.

연결 리스트(Linked List)

- 연결 리스트는 자료들을 반드시 연속적으로 배열시키지는 않고 임의의 기억공간에 기억시키되, 자료 항목의 순서에 따라 노드의 포인터 부분을 이용하여 서로 연결시킨 자료 구조이다.
- 연결 리스트는 노드의 삽입 · 삭제 작업이 용이하다.
- 기억 공간이 연속적으로 놓여 있지 않아도 저장할 수 있다.
- 연결 리스트는 연결을 위한 링크(포인터) 부분이 필요하기 때문에 순차 리스트에 비해 기억 공간의 이용 효율이 좋지 않다.
- 연결 리스트는 연결을 위한 포인터를 찾는 시간이 필요하기 때문에 접근 속도가 느리다.
- 연결 리스트는 중간 노드 연결이 끊어지면 그 다음 노드를 찾기 힘들다.

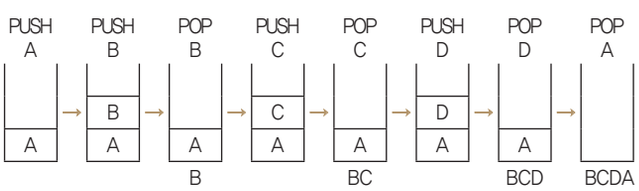
23.7, 23.2, 22.7, 22.4, 22.3, 21.8, 21.5, 21.3 2403605

핵심 075 스택(Stack)

스택은 리스트의 한쪽 끝으로만 자료의 삽입, 삭제 작업이 이루어지는 자료 구조이다.

- 스택은 가장 나중에 삽입된 자료가 가장 먼저 삭제되는 후입선출(LIFO; Last In First Out) 방식으로 자료를 처리한다.
- 스택의 응용 분야
  - 함수 호출의 순서 제어
  - 인터럽트의 처리
  - 수식 계산 및 수식 표기법
  - 컴파일러를 이용한 언어 번역
  - 부 프로그램 호출 시 복귀주소 저장
  - 서브루틴 호출 및 복귀 주소 저장
- 스택의 모든 기억 공간이 꽉 채워져 있는 상태에서 데이터가 삽입되면 오버플로(Overflow)가 발생하며, 더 이상 삭제할 데이터가 없는 상태에서 데이터를 삭제하면 언더플로(Underflow)가 발생한다.

예제 순서가 A, B, C, D로 정해진 입력 자료를 스택에 입력하였다가 B, C, D, A 순서로 출력하는 과정을 나열하시오.



21.3 2403606

핵심 076 큐(Queue)

큐는 리스트의 한쪽에서는 삽입 작업이 이루어지고 다른 한쪽에서는 삭제 작업이 이루어지도록 구성한 자료 구조이다.

- 큐는 가장 먼저 삽입된 자료가 가장 먼저 삭제되는 선입선출(FIFO; First In First Out) 방식으로 처리한다.
- 큐는 시작과 끝을 표시하는 두 개의 포인터가 있다.

데크(Deque)는 삽입과 삭제가 리스트의 양쪽 끝에서 발생할 수 있다.

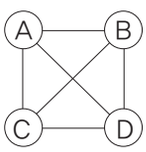
23.2, 20.9

핵심 077 방향/무방향 그래프의 최대 간선 수

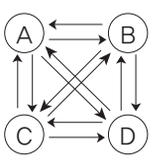
2403631

n개의 정점으로 구성된 무방향 그래프에서 최대 간선 수는 n(n−1)/2이고, 방향 그래프에서 최대 간선 수는 n(n−1)이다.

예 정점이 4개인 경우 무방향 그래프와 방향 그래프의 최대 간선 수는 다음과 같다.



- 무방향 그래프의 최대 간선 수 : 4(4−1)/2 = 6



- 방향 그래프의 최대 간선 수 : 4(4−1) = 12

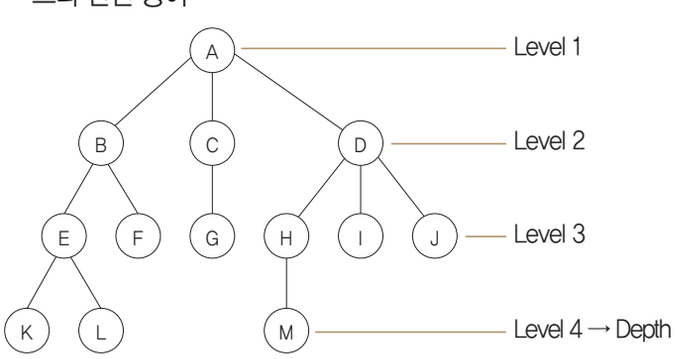
23.7, 23.2, 21.3, 20.8, 20.3

핵심 078 트리의 개요

2403701

트리는 정점(Node, 노드)과 선분(Branch, 가지)을 이용하여 사이클을 이루지 않도록 구성한 그래프(Graph)의 특수한 형태이다.

- 트리 관련 용어



– 노드(Node) : 트리의 기본 요소로서 자료 항목과 다른 항목에 대한 가지(Branch)를 합친 것

예 A, B, C, D, E, F, G, H, I, J, K, L, M

– 근 노드(Root Node) : 트리의 맨 위에 있는 노드

예 A

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

예 A = 3, B = 2, C = 1, D = 3

– 단말 노드(Terminal Node) = 잎 노드(Leaf Node) : 자식이 하나도 없는 노드, 즉 디그리가 0인 노드

예 K, L, F, G, M, I, J

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

예 D의 자식 노드 : H, I, J

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

예 E, F의 부모 노드 : B

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

예 H의 형제 노드 : I, J

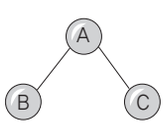
– 트리의 디그리 : 노드들의 디그리 중에서 가장 많은 수

예 노드 A나 D가 3개의 디그리를 가지므로 앞 트리의 디그리는 3이다.

23.5, 22.7, 22.4, 21.8, 21.3, 20.9, 20.8, 20.6

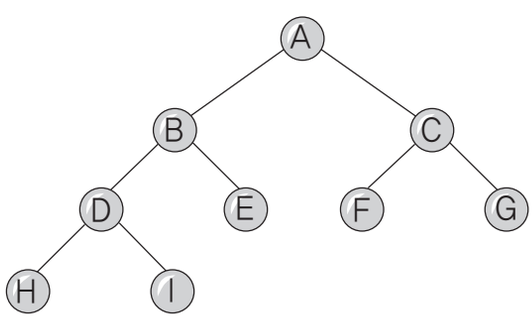
핵심 079 트리의 운행법

2403702



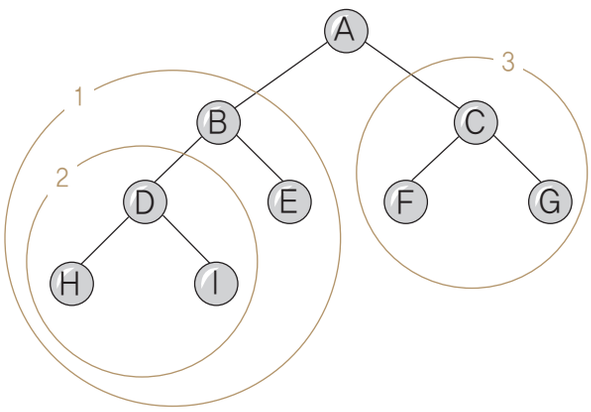
- Preorder 운행 : Root → Left → Right 순으로 운행.
  A, B, C
- Inorder 운행 : Left → Root → Right 순으로 운행.
  B, A, C
- Postorder 운행 : Left → Right → Root 순으로 운행.
  B, C, A

예제 다음 트리를 Inorder, Preorder, Postorder 방법으로 운행했을 때 각 노드를 방문한 순서는?



Preorder 운행법의 방문 순서

※ 서브 트리를 하나의 노드로 생각할 수 있도록 그림과 같이 서브트리 단위로 묶는다. Preorder, Inorder, Postorder 모두 공통으로 사용한다.



❶ Preorder는 Root → Left → Right이므로 A13이 된다.

❷ 1은 B2E이므로 AB2E3이 된다.

❸ 2는 DHI이므로 ABDHIE3이 된다.

❹ 3은 CFG이므로 ABDHIECFG가 된다.

- 방문 순서 : ABDHIECFG

Inorder 운행법의 방문 순서

❶ Inorder는 Left → Root → Right이므로 1A3이 된다.

❷ 1은 2BE이므로 2BEA3이 된다.

❸ 2는 HDI이므로 HDIBEA3이 된다.

❹ 3은 FCG이므로 HDIBEAFCG가 된다.

- 방문 순서 : HDIBEAFCG

Postorder

❶ Postorder는 Left → Right → Root이므로 13A가 된다.

❷ 1은 2EB이므로 2EB3A가 된다.

❸ 2는 HID이므로 HIDEB3A가 된다.

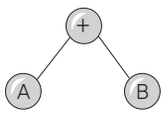
❹ 3은 FGC이므로 HIDEBFGCA가 된다.

- 방문 순서 : HIDEBFGCA

21.5, 21.3, 20.9

핵심 080 수식의 표기법

2403703



- 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.

예제 1 다음과 같이 Infix로 표기된 수식을 Prefix와 Postfix로 변환하시오. → 연산순위 괄 때기!

X = A / B * (C + D) + E

- Prefix로 변환하기

❶ 연산 우선순위에 따라 괄호로 묶는다.

(X = (((A / B) * (C + D)) + E))

❷ 연산자를 해당 괄호의 앞(왼쪽)으로 옮긴다.

X = (((A / B) * (C + D)) + E)) →

= (X + (*(/ (AB) + (CD)) E))

❸ 필요없는 괄호를 제거한다.

= X + * / A B + C D E

- Postfix로 변환하기

❶ 연산 우선순위에 따라 괄호로 묶는다.

(X = (((A / B) * (C + D)) + E))

❷ 연산자를 해당 괄호의 뒤(오른쪽)로 옮긴다.

(X = (((A / B) * (C + D)) + E)) →

(X (((A B) / (C D) +) * E) +) =

❸ 필요 없는 괄호를 제거한다.

X A B / C D + * E + =

Postfix나 Prefix로 표기된 수식을 Infix로 바꾸기

예제 2 다음과 같이 Postfix로 표기된 수식을 Infix로 변환하시오.

A B C − / D E F + ∗ +

Postfix는 Infix 표기법에서 연산자를 해당 피연산자 두 개의 뒤로 이동한 것이므로 연산자를 다시 해당 피연산자 두 개의 가운데로 옮기면 된다.

❶ 먼저 인접한 피연산자 두 개와 오른쪽의 연산자를 괄호로 묶는다.

((A (B C −) /) (D (E F +) ∗) +)

❷ 연산자를 해당 피연산자의 가운데로 이동시킨다.

((A (B C −) /) (D (E F +) ∗) +) →
((A / (B − C)) + (D ∗ (E + F)))

❸ 필요 없는 괄호를 제거한다.

((A / (B − C)) + (D ∗ (E + F))) →
A / (B − C) + D ∗ (E + F)

예제 3 다음과 같이 Prefix로 표기된 수식을 Infix로 변환하시오.

+ / A − B C ∗ D + E F

Prefix는 Infix 표기법에서 연산자를 해당 피연산자 두 개의 앞으로 이동한 것이므로 연산자를 다시 해당 피연산자 두 개의 가운데로 옮기면 된다.

❶ 먼저 인접한 피연산자 두 개와 왼쪽의 연산자를 괄호로 묶는다.

(+ (/ A (− B C)) (∗ D (+ E F)))

❷ 연산자를 해당 피연산자 사이로 이동시킨다.

(+ (/ A (− B C)) (∗ D (+ E F))) →
((A/(B−C)) + (D ∗ (E+F)))

❸ 필요 없는 괄호를 제거한다.

((A/(B−C)) + (D∗(E+F))) → A/(B−C)+D∗(E+F)

20.9

핵심 081 삽입 정렬(Insertion Sort)

2403801

O(n²)

예제 8, 5, 6, 2, 4를 삽입 정렬로 정렬하시오.

• **초기 상태 :** 8 5 6 2 4

• **1회전 :** 8 5 6 2 4 → 5 8 6 2 4

두 번째 값을 첫 번째 값과 비교하여 5를 첫 번째 자리에 삽입하고 8을 한 칸 뒤로 이동시킨다.

• **2회전 :** 5 8 6 2 4 → 5 6 8 2 4

세 번째 값을 첫 번째, 두 번째 값과 비교하여 6을 8자리에 삽입하고 8을 한 칸 뒤로 이동시킨다.

5 6 8 2 4 → 2 5 6 8 4

네 번째 값 2를 처음부터 비교하여 맨 처음에 삽입하고 나머지를 한 칸씩 뒤로 이동시킨다.

• **4회전 :** 2 5 6 8 4 → 2 4 5 6 8

다섯 번째 값 4를 처음부터 비교하여 5자리에 삽입하고 나머지를 한 칸씩 뒤로 이동시킨다.

22.7, 21.3, 20.8 1번째 자리부터 최댓값으로 저장

핵심 082 선택 정렬(Selection Sort)

2403803

O(n²)

예제 8, 5, 6, 2, 4를 선택 정렬로 정렬하시오.

• **초기 상태 :** 8 5 6 2 4

• **1회전 :** 5 8 6 2 4 → 5 8 6 2 4 → 2 8 6 5 4 → 2 8 6 5 4

• **2회전 :** 2 6 8 5 4 → 2 5 8 6 4 → 2 4 8 6 5

• **3회전 :** 2 4 6 8 5 → 2 4 5 8 6

• **4회전 :** 2 4 5 6 8

23.2, 22.4, 21.8, 21.5 마지막 자리부터 최대값으로 자리

핵심 083 버블 정렬(Bubble Sort)

2403804

예제 8, 5, 6, 2, 4를 버블 정렬로 정렬하시오. $O(n^2)$

- 초기 상태 : 8 5 6 2 4
- 1회전 : 5 8 6 2 4 → 5 6 8 2 4 → 5 6 2 8 4 → 5 6 2 4 8
- 2회전 : 5 6 2 4 8 → 5 2 6 4 8 → 5 2 4 6 8
- 3회전 : 2 5 4 6 8 → 2 4 5 6 8
- 4회전 : 2 4 5 6 8

23.5, 23.2, 22.3, 21.3

핵심 084 퀵 정렬(Quick Sort)

2403805

퀵 정렬은 레코드의 많은 자료 이동을 없애고 하나의 파일을 부분적으로 나누어 가면서 정렬하는 방법으로 키를 기준으로 작은 값은 왼쪽에, 큰 값은 오른쪽 서브파일로 분해시키는 방식으로 정렬한다.

- 분할(Divide)과 정복(Conquer)을 통해 자료를 정렬한다.
- 평균 수행 시간 복잡도는 $O(n\log_2 n)$이고, 최악의 수행 시간 복잡도는 $O(n^2)$이다.

23.5, 21.5

핵심 085 힙 정렬(Heap Sort)

2403806

힙 정렬은 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.

21.5

핵심 086 2-Way 합병 정렬 (Merge Sort)

2403807

2-Way 합병 정렬은 이미 정렬되어 있는 두 개의 파일을 한 개의 파일로 합병하는 정렬 방식이다.

- 평균과 최악 모두 시간 복잡도는 $O(n\log_2 n)$이다.

23.7, 22.4, 21.3

핵심 087 이분 검색

2403901

이분 검색(이진 검색, Binary Search)은 전체 파일을 두 개의 서브파일로 분리해 가면서 Key 레코드를 검색하는 방식이다.

- 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 $M = \frac{(F+L)}{2}$ (단, F : 첫 번째 레코드 번호, L : 마지막 레코드 번호)

22.7, 21.3, 20.9

핵심 088 해싱 함수 (Hashing Function)

2403902

- 제산법(Division) : 레코드 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 mod Q임
- 제곱법(Mid-Square) : 레코드 키 값(K)을 제곱한 후 그 중간 부분의 값을 홈 주소로 삼는 방식
- 폴딩법(Folding) : 레코드 키 값(K)을 여러 부분으로 나눈 후 각 부분의 값을 더하거나 XOR(배타적 논리합)한 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 홈 주소로 삼는 방식
- 무작위법(Random) : 난수(Random Number)를 발생시켜 나온 값을 홈 주소로 삼는 방식

체이닝(Chaining) : Collision이 발생하면 버킷에 할당된 연결리스트에 데이터를 저장하는 방법

23.2

핵심 089 DBMS(DataBase Management System; 데이터베이스 관리 시스템)

2404003

DBMS란 사용자와 데이터베이스 사이에서 사용자의 요구에 따라 정보를 생성해주고, 데이터베이스를 관리해 주는 소프트웨어이다.

- DBMS의 필수 기능

정의 기능	모든 응용 프로그램들이 요구하는 데이터 구조를 지원하기 위해 데이터베이스에 저장될 데이터의 형(Type)과 구조에 대한 정의, 이용 방식, 제약 조건 등을 명시하는 기능
조작 기능	데이터 검색, 갱신, 삽입, 삭제 등을 체계적으로 처리하기 위해 사용자와 데이터베이스 사이의 인터페이스 수단을 제공하는 기능
제어 기능	데이터베이스를 접근하는 갱신, 삽입, 삭제 작업이 정확하게 수행되어 데이터의 무결성이 유지되도록 제어해야 함

2404004

핵심 090 DBMS의 장 · 단점

장점	• 데이터의 논리적, 물리적 독립성이 보장됨 • 데이터의 중복을 피할 수 있어 기억 공간이 절약됨 • 저장된 자료를 공동으로 이용할 수 있음 • 데이터의 일관성을 유지할 수 있음 • 데이터의 무결성을 유지할 수 있음 • 보안을 유지할 수 있음 • 데이터를 표준화할 수 있음 • 데이터를 통합하여 관리할 수 있음 • 항상 최신의 데이터를 유지함 • 데이터의 실시간 처리가 가능함
단점	• 데이터베이스의 전문가가 부족함 • 전산화 비용이 증가함 • 대용량 디스크로의 집중적인 Access로 과부하(Overhead)가 발생함 • 파일의 예비(Backup)와 회복(Recovery)이 어려움 • 시스템이 복잡함

21.3, 20.9

핵심 091 스키마

구조라고 생각하면 됨

2404005

스키마(Schema)는 데이터베이스의 구조와 제약 조건에 관한 전반적인 명세(Specification)를 기술(Description)한 메타데이터(Meta-Data)의 집합이다.

- 스키마는 사용자의 관점에 따라 외부 스키마, 개념 스키마, 내부 스키마로 나누어진다.

외부 스키마	사용자나 응용 프로그래머가 각 개인의 입장에서 필요로 하는 데이터베이스의 논리적 구조를 정의한 것
개념 스키마	데이터베이스의 전체적인 논리적 구조로서, 모든 응용 프로그램이나 사용자들이 필요로 하는 데이터를 종합한 조직 전체의 데이터베이스로 하나만 존재함
내부 스키마	물리적 저장장치의 입장에서 본 데이터베이스 구조로서, 실제로 데이터베이스에 저장될 레코드의 형식을 정의하고 저장 데이터 항목의 표현 방법, 내부 레코드의 물리적 순서 등을 나타냄

스키마: DB의 구조와 제약조건에 관한 전반적인 명세를 기술한 것

21.5

핵심 092 절차형 SQL의 테스트와 디버깅

2404202

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

- 테스트와 디버깅의 목적 : 테스트(Test)를 통해 오류를 발견한 후 디버깅(Debugging)을 통해 오류가 발생한 소스 코드를 추적하며 수정함

핵심 093 단위 모듈(Unit Module)의 개요

2404301

단위 모듈은 소프트웨어 구현에 필요한 여러 동작 중 한 가지 동작을 수행하는 기능을 모듈로 구현한 것이다.

- 단위 모듈로 구현되는 하나의 기능을 단위 기능이라고 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 삽입되기도 한다.
- 단위 모듈을 구현하기 위해서는 단위 기능 명세서를 작성한 후 입 · 출력 기능과 알고리즘을 구현해야 한다.

단위 기능 명세서 작성 → 입 · 출력 기능 구현 → 알고리즘 구현

핵심 094 IPC(Inter-Process Communication)

2404331

IPC는 모듈 간 통신 방식을 구현하기 위해 사용되는 대표적인 프로그래밍 인터페이스 집합으로, 복수의 프로세스를 수행하며 이뤄지는 프로세스 간 통신까지 구현이 가능하다.

- IPC의 대표 메소드 5가지

공유 메모리	Shared Memory	다수의 프로세스가 공유 가능한 메모리를 구성하여 프로세스 간 통신을 수행
소켓	Socket	네트워크 소켓을 이용하여 네트워크를 경유하는 프로세스들 간 통신을 수행
세마포어	Semaphores	공유 자원에 대한 접근 제어를 통해 프로세스 간 통신을 수행
파이프	Pipes&named Pipes	• 'Pipe'라고 불리는 선입선출 형태로 구성된 메모리를 여러 프로세스가 공유하여 통신을 수행 • 하나의 프로세스가 Pipe를 이용 중이라면 다른 프로세스는 접근할 수 없음
메시지 큐	Message Queueing	메시지가 발생하면 이를 전달하는 형태로 프로세스 간 통신을 수행

핵심 095 단위 모듈 테스트의 개요

2404401

단위 모듈 테스트는 프로그램의 단위 기능을 구현하는 모듈이 정해진 기능을 정확히 수행하는지 검증하는 것이다.

- 단위 모듈 테스트는 단위 테스트(Unit Test)라고도 하며, 화이트박스 테스트와 블랙박스 테스트 기법을 사용한다.
- 단위 모듈 테스트를 수행하기 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.

21.3

핵심 096 테스트 케이스(Test Case)

2404402

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.

- 단위 모듈을 테스트하기 전에 테스트에 필요한 입력 데이터, 테스트 조건, 예상 결과 등을 모아 테스트 케이스를 만든다.
- 테스트 케이스를 이용하지 않고 수행하는 직관적인 테스트는 특정 요소에 대한 검증이 누락되거나 불필요한 검증의 반복으로 인해 인력과 시간을 낭비할 수 있다.
- ISO/IEC/IEEE 29119-3 표준에 따른 테스트 케이스의 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 Dependencies) : 테스트 케이스 간의 의존성

23.7, 23.2, 22.4

핵심 097 통합 개발 환경(IDE; Integrated Development Environment)

2404501

통합 개발 환경은 코딩, 디버그, 컴파일, 배포 등 프로그램 개발과 관련된 모든 작업을 하나의 프로그램에서 처리할 수 있도록 제공하는 소프트웨어적인 개발 환경을 말한다.

- 기존 소프트웨어 개발에서는 편집기(Editor), 컴파일러(Compiler), 디버거(Debugger) 등의 다양한 툴을 별도로 사용했으나 현재는 하나의 인터페이스로 통합하여 제공한다.
- 통합 개발 환경 도구는 통합 개발 환경을 제공하는 소프트웨어를 의미한다.
- 통합 개발 환경을 지원하는 도구는 플랫폼, 운영체제, 언어별로 다양하게 존재한다.
- **통합 개발 환경 도구의 대표적인 기능**

코딩 (Coding)	C, JAVA 등의 프로그래밍 언어로 프로그램을 작성하는 기능
컴파일 (Compile)	개발자가 작성한 고급 언어로 된 프로그램을 컴퓨터가 이해할 수 있는 목적 프로그램으로 번역하여 컴퓨터에서 실행 가능한 형태로 변환하는 기능
디버깅 (Debugging)	소프트웨어나 하드웨어의 오류나 잘못된 동작, 즉 버그(Bug)를 찾아 수정하는 기능
배포 (Deployment)	소프트웨어를 사용자에게 전달하는 기능

23.5, 22.3

핵심 098 빌드 도구

2404502

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	아파치 소프트웨어 재단(Apache Software Foundation)에서 개발한 소프트웨어로, 자바 프로젝트의 공식적인 빌드 도구로 사용되고 있음
Maven	Ant와 동일한 아파치 소프트웨어 재단에서 개발된 것으로, Ant의 대안으로 개발되었음
Gradle	기존의 Ant와 Maven을 보완하여 개발된 빌드 도구로, 한스 도커(Hans Dockter) 외 6인의 개발자가 모여 공동 개발하였음

23.2, 22.7, 22.3, 21.5

핵심 099 소프트웨어 패키징의 개요

2404601

소프트웨어 패키징이란 모듈별로 생성한 실행 파일들을 묶어 배포용 설치 파일을 만드는 것을 말한다.

- 개발자가 아니라 사용자를 중심으로 진행한다.
- 소스 코드는 향후 관리를 고려하여 모듈화하여 패키징한다.
- 사용자가 소프트웨어를 사용하게 될 환경을 이해하여, 다양한 환경에서 소프트웨어를 손쉽게 사용할 수 있도록 일반적인 배포 형태로 패키징한다.

20.9, 20.8, 20.6

핵심 100 패키징 시 고려사항

2404602

- 사용자의 시스템 환경, 즉 운영체제(OS), CPU, 메모리 등에 필요한 최소 환경을 정의한다.
- UI(User Interface)는 사용자가 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 소프트웨어이므로 내부 콘텐츠에 대한 암호화 및 보안을 고려한다.
- 다른 여러 콘텐츠 및 단말기 간 DRM(디지털 저작권 관리) 연동을 고려한다.
- 사용자의 편의성을 위한 복잡성 및 비효율성 문제를 고려한다.
- 제품 소프트웨어 종류에 적합한 암호화 알고리즘을 적용한다.

핵심 101 릴리즈 노트(Release Note)의 개요

2404701

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, 개선 사항 등을 사용자와 공유할 수 있다.
- 릴리즈 노트를 이용해 소프트웨어의 버전 관리나 릴리즈 정보를 체계적으로 관리할 수 있다.
- 릴리즈 노트는 소프트웨어의 초기 배포 시 또는 출시 후 개선 사항을 적용한 추가 배포 시에 제공한다.

핵심 102 릴리즈 노트 초기 버전 작성 시 고려사항

2404702

릴리즈 노트의 초기 버전은 다음의 사항을 고려하여 작성한다.

- 릴리즈 노트는 정확하고 완전한 정보를 기반으로 개발팀에서 직접 현재 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.
  - Header(머릿말)
  - 개요
  - 목적
  - 문제 요약
  - 재현 항목
  - 수정/개선 내용
  - 사용자 영향도
  - SW 지원 영향도
  - 노트
  - 면책 조항
  - 연락처

23.5, 22.4

핵심 103 디지털 저작권 관리(DRM; Digital Right Management)

2404802

디지털 저작권 관리는 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 변환한 후 패키저(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 하우스(Clearing House)에 등록된다.
- 사용자가 콘텐츠를 사용하기 위해서는 클리어링 하우스에 등록된 라이선스 정보를 통해 사용자 인증과 콘텐츠 사용 권한 소유 여부를 확인받아야 한다.
- 종량제 방식을 적용한 소프트웨어의 경우 클리어링 하우스를 통해 서비스의 실제 사용량을 측정하여 이용한 만큼의 요금을 부과한다.

22.4, 21.8, 21.5, 20.9

핵심 104 디지털 저작권 관리(DRM)의 구성 요소

2404803

- 클리어링 하우스(Clearing House) : 저작권에 대한 사용 권한, 라이선스 발급, 암호화된 키 관리, 사용량에 따른 결제 관리 등을 수행하는 곳
- 콘텐츠 제공자(Contents Provider) : 콘텐츠를 제공하는 저작권자
- 패키저(Packager) : 콘텐츠를 메타 데이터와 함께 배포 가능한 형태로 묶어 암호화하는 프로그램
- 콘텐츠 분배자(Contents Distributor) : 암호화된 콘텐츠를 유통하는 곳이나 사람
- 콘텐츠 소비자(Customer) : 콘텐츠를 구매해서 사용하는 주체
- DRM 컨트롤러(DRM Controller) : 배포된 콘텐츠의 이용 권한을 통제하는 프로그램
- 보안 컨테이너(Security Container) : 콘텐츠 원본을 안전하게 유통하기 위한 전자적 보안 장치

23.7, 23.2, 22.7, 21.3, 20.9, 20.8, 20.6

핵심 105 디지털 저작권 관리(DRM)의 기술 요소

2404804

- 암호화(Encryption) : 콘텐츠 및 라이선스를 암호화하고 전자 서명을 할 수 있는 기술
- 키 관리(Key Management) : 콘텐츠를 암호화한 키에 대한 저장 및 분배 기술
- 암호화 파일 생성(Packager) : 콘텐츠를 암호화된 콘텐츠로 생성하기 위한 기술
- 식별 기술(Identification) : 콘텐츠에 대한 식별 체계 표현 기술
- 저작권 표현(Right Expression) : 라이선스의 내용 표현 기술
- 정책 관리(Policy Management) : 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

20.9

핵심 106 소프트웨어 설치 매뉴얼의 개요

2404901

소프트웨어 설치 매뉴얼은 개발 초기에서부터 적용된 기준이나 사용자가 소프트웨어를 설치하는 과정에 필요한 내용을 기록한 설명서와 안내서이다.

- 설치 매뉴얼은 사용자 기준으로 작성한다.
- 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.
- 작성 순서 : 기능 식별 → UI 분류 → 설치 파일/백업 파일 확인 → Uninstall 절차 확인 → 이상 Case 확인 → 최종 매뉴얼 적용

23.5, 21.3

핵심 107 소프트웨어 설치 매뉴얼의 기본 사항

2404903

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 (Installation)	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

21.8

핵심 108 소프트웨어 사용자 매뉴얼의 개요

2405001

소프트웨어 사용자 매뉴얼은 사용자가 소프트웨어를 사용하는 과정에서 필요한 내용을 문서로 기록한 설명서와 안내서이다.

- 사용자 매뉴얼은 사용자가 소프트웨어 사용에 필요한 절차, 환경 등의 제반 사항이 모두 포함되도록 작성한다.
- 소프트웨어 배포 후 발생될 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.
- 사용자 매뉴얼에는 목차 및 개요, 서문, 기본 사항 등이 기본적으로 포함되어야 한다.
- 작성 순서 : 작성 지침 정의 → 사용자 매뉴얼 구성 요소 정의 → 구성 요소별 내용 작성 → 사용자 매뉴얼 검토

23.7, 22.7, 22.4, 21.8, 21.5, 21.3, 20.9, 20.6

핵심 109 소프트웨어 패키징의 형상 관리

2405101

형상 관리(SCM; Software Configuration Management)는 소프트웨어의 개발 과정에서 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 각종 정의서, 지침서, 분석서 등이 포함된다.
- 형상 관리를 통해 가시성과 추적성을 보장함으로써 소프트웨어의 생산성과 품질을 높일 수 있다.
- 대표적인 형상 관리 도구에는 Git, CVS, Subversion 등이 있다.

Git, CVS, SVN

20.8

핵심 110 형상 관리의 중요성

2405102

- 지속적인 소프트웨어의 변경 사항을 체계적으로 추적하고 통제할 수 있다.
- 제품 소프트웨어에 대한 무절제한 변경을 방지할 수 있다.
- 제품 소프트웨어에서 발견된 버그나 수정 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.

21.8, 21.3

핵심 111 형상 관리 기능

2405103

- 형상 식별 : 형상 관리 대상에 이름과 관리 번호를 부여하고, 계층(Tree) 구조로 구분하여 수정 및 추적이 용이하도록 하는 작업
- 버전 제어 : 소프트웨어 업그레이드나 유지 보수 과정에서 생성된 다른 버전의 형상 항목을 관리하고, 이를 위해 특정 절차와 도구(Tool)를 결합시키는 작업
- 형상 통제(변경 관리) : 식별된 형상 항목에 대한 변경 요구를 검토하여 현재의 기준선(Base Line)이 잘 반영될 수 있도록 조정하는 작업
- 형상 감사 : 기준선의 무결성을 평가하기 위해 확인, 검증, 검열 과정을 통해 공식적으로 승인하는 작업
- 형상 기록(상태 보고) : 형상의 식별, 통제, 감사 작업의 결과를 기록 · 관리하고 보고서를 작성하는 작업

23.2, 21.5, 20.8

핵심 112 소프트웨어의 버전 등록 관련 주요 기능

2405104

항목	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 복사함
체크아웃 (Check-Out)	• 프로그램을 수정하기 위해 저장소(Repository)에서 파일을 받아옴 • 소스 파일과 함께 버전 관리를 위한 파일들도 받아옴
체크인 (Check-In)	체크아웃 한 파일의 수정을 완료한 후 저장소(Repository)의 파일을 새로운 버전으로 갱신함
커밋 (Commit)	체크인을 수행할 때 이전에 갱신된 내용이 있는 경우에는 충돌(Conflict)을 알리고 diff 도구를 이용해 수정한 후 갱신을 완료함
동기화 (Update)	저장소에 있는 최신 버전으로 자신의 작업 공간을 동기화함

22.4

핵심 113 공유 폴더 방식

2405201

공유 폴더 방식은 버전 관리 자료가 로컬 컴퓨터의 공유 폴더에 저장되어 관리되는 방식으로, 다음과 같은 특징이 있다.

- 개발자들은 개발이 완료된 파일을 약속된 공유 폴더에 매일 복사한다.
- 담당자는 공유 폴더의 파일을 자기 PC로 복사한 후 컴파일 하여 이상 유무를 확인한다.
- 종류에는 SCCS, RCS, PVCS, QVCS 등이 있다.

핵심 114 클라이언트/서버 방식

2405202

클라이언트/서버 방식은 버전 관리 자료가 중앙 시스템(서버)에 저장되어 관리되는 방식으로, 다음과 같은 특징이 있다.

- 서버의 자료를 개발자별로 자신의 PC(클라이언트)로 복사하여 작업한 후 변경된 내용을 서버에 반영한다.
- 모든 버전 관리는 서버에서 수행된다.
- 종류에는 CVS, SVN(Subversion), CVSNT, Clear Case, CMVC, Perforce 등이 있다.

21.5

핵심 115 분산 저장소 방식

2405203

분산 저장소 방식은 버전 관리 자료가 하나의 원격 저장소와 분산된 개발자 PC의 로컬 저장소에 함께 저장되어 관리되는 방식으로, 다음과 같은 특징이 있다.

- 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, DCVS, Bazaar, Mercurial, TeamWare, Bitkeeper, Plastic SCM 등이 있다.

핵심 116 Subversion(서브버전, SVN)

2405204

Subversion은 CVS를 개선한 것으로, 아파치 소프트웨어 재단에서 2000년에 발표하였다.

- 클라이언트/서버 구조로, 서버(저장소, Repository)에는 최신 버전의 파일들과 변경 내역이 관리된다.
- 서버의 자료를 클라이언트로 복사해와 작업한 후 변경 내용을 서버에 반영(Commit)한다.
- 모든 개발 작업은 trunk 디렉터리에서 수행되며, 추가 작업은 branches 디렉터리 안에 별도의 디렉터리를 만들어 작업을 완료한 후 trunk 디렉터리와 병합(merge)한다.
- 커밋(Commit)할 때마다 리비전(Revision)이 1씩 증가한다.
- 클라이언트는 대부분의 운영체제에서 사용되지만, 서버는 주로 유닉스를 사용한다.
- 소스가 오픈되어 있어 무료로 사용할 수 있다.
- CVS의 단점이었던 파일이나 디렉터리의 이름 변경, 이동 등이 가능하다.

핵심 117 Git(깃)

2405205

Git은 리누스 토발즈(Linus Torvalds)가 2005년 리눅스 커널 개발에 사용할 관리 도구로 개발한 이후 주니오 하마노(Junio Hamano)에 의해 유지 보수되고 있다.

- Git은 분산 버전 관리 시스템으로 2개의 저장소, 즉 지역(로컬) 저장소와 원격 저장소가 존재한다.
- 지역 저장소는 개발자들이 실제 개발을 진행하는 장소로, 버전 관리가 수행된다.
- 원격 저장소는 여러 사람들이 협업을 위해 버전을 공동 관리하는 곳으로, 자신의 버전 관리 내역을 반영하거나 다른 개발자의 변경 내용을 가져올 때 사용한다.
- 버전 관리가 지역 저장소에서 진행되므로 버전 관리가 신속하게 처리되고, 원격 저장소나 네트워크에 문제가 있어도 작업이 가능하다.
- 브랜치를 이용하면 기본 버전 관리 틀에 영향을 주지 않으면서 다양한 형태의 기능 테스팅이 가능하다.
- 파일의 변화를 스냅샷(Snapshot)으로 저장하는데, 스냅샷은 이전 스냅샷의 포인터를 가지므로 버전의 흐름을 파악할 수 있다.

20.9

핵심 118 빌드 자동화 도구의 개념

2405301

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

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

20.9

핵심 119 Jenkins

2405302

Jenkins는 JAVA 기반의 오픈 소스 형태로, 가장 많이 사용되는 빌드 자동화 도구이다.

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

20.9

핵심 120 Gradle

2405303

Gradle은 Groovy를 기반으로 한 오픈 소스 형태의 자동화 도구로, 안드로이드 앱 개발 환경에서 사용된다.

- 안드로이드 뿐만 아니라 플러그인을 설정하면, JAVA, C/C++, Python 등의 언어도 빌드가 가능하다.
- Groovy를 사용해서 만든 DSL(Domain Specific Language)을 스크립트 언어로 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.

21.8

핵심 121 애플리케이션 테스트의 개념

2405401

애플리케이션 테스트는 애플리케이션에 잠재되어 있는 결함을 찾아내는 일련의 행위 또는 절차이다.

- 애플리케이션 테스트는 개발된 소프트웨어가 고객의 요구사항을 만족시키는지 확인(Validation)하고 소프트웨어가 기능을 정확히 수행하는지 검증(Verification)한다.

확인(Validation)	사용자의 입장에서 개발한 소프트웨어가 고객의 요구사항에 맞게 구현되었는지를 확인하는 것
검증(Verification)	개발자의 입장에서 개발한 소프트웨어가 명세서에 맞게 만들어졌는지를 점검하는 것

22.7, 21.5, 20.6

핵심 122 애플리케이션 테스트 관련 용어

2405403

결함 집중(Defect Clustering)

대부분의 결함이 소수의 특정 모듈에 집중해서 발생하는 것이다.

파레토 법칙(Pareto Principle)

상위 20% 사람들이 전체 부의 80%를 가지고 있다거나, 상위 20% 고객이 매출의 80%를 창출한다는 의미로, 이 법칙이 애플리케이션 테스트에도 적용된다는 것이다. 즉 테스트로 발견된 80%의 오류는 20%의 모듈에서 발견되므로 20%의 모듈을 집중적으로 테스트하여 효율적으로 오류를 찾자는 것이다.

살충제 패러독스 (Pesticide Paradox)

살충제를 지속적으로 뿌리면 벌레가 내성이 생겨서 죽지 않는 현상을 의미한다. 동일한 테스트 케이스로 동일한 테스트를 반복하면 더 이상 결함이 발견되지 않는 현상을 의미한다.

오류-부재의 궤변(Absence of Errors Fallacy)

소프트웨어의 결함을 모두 제거해도 사용자의 요구사항을 만족시키지 못하면 해당 소프트웨어는 품질이 높다고 말할 수 없다는 것을 의미한다.

핵심 123 프로그램 실행 여부에 따른 테스트

2405501

정적 테스트	• 프로그램을 실행하지 않고 명세서나 소스 코드를 대상으로 분석하는 테스트 • 소프트웨어 개발 초기에 결함을 발견할 수 있어 소프트웨어의 개발 비용을 낮추는데 도움이 됨 • 종류 : 워크스루, 인스펙션, 코드 검사 등
동적 테스트	• 프로그램을 실행하여 오류를 찾는 테스트로, 소프트웨어 개발의 모든 단계에서 테스트를 수행할 수 있음 • 종류 : 블랙박스 테스트, 화이트박스 테스트

핵심 124 테스트 기반(Test Bases)에 따른 테스트

2405502

명세 기반 테스트	• 사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 만들어 구현하고 있는지 확인하는 테스트 • 종류 : 동등 분할, 경계 값 분석 등
구조 기반 테스트	• 소프트웨어 내부의 논리 흐름에 따라 테스트 케이스를 작성하고 확인하는 테스트 • 종류 : 구문 기반, 결정 기반, 조건 기반 등
경험 기반 테스트	• 유사 소프트웨어나 기술 등에 대한 테스터의 경험을 기반으로 수행하는 테스트 • 경험 기반 테스트는 사용자의 요구사항에 대한 명세가 불충분하거나 테스트 시간에 제약이 있는 경우 수행하면 효과적 • 종류 : 에러 추정, 체크 리스트, 탐색적 테스팅

핵심 125 시각에 따른 테스트

2405503

검증 (Verification) 테스트	개발자의 시각에서 제품의 생산 과정을 테스트하는 것으로, 제품이 명세서대로 완성됐는지를 테스트함
확인 (Validation) 테스트	사용자의 시각에서 생산된 제품의 결과를 테스트하는 것으로, 사용자가 요구한대로 제품이 완성됐는지, 제품이 정상적으로 동작하는지를 테스트함

21.8

핵심 126 목적에 따른 테스트

2405504

회복 (Recovery) 테스트	시스템에 여러 가지 결함을 주어 실패하도록 한 후 올바르게 복구되는지를 확인하는 테스트
안전(Security) 테스트	시스템에 설치된 시스템 보호 도구가 불법적인 침입으로부터 시스템을 보호할 수 있는지를 확인하는 테스트
강도(Stress) 테스트	시스템에 과도한 정보량이나 빈도 등을 부과하여 과부하 시에도 소프트웨어가 정상적으로 실행되는지를 확인하는 테스트
성능 (Performance) 테스트	소프트웨어의 실시간 성능이나 전체적인 효율성을 진단하는 테스트로, 소프트웨어의 응답 시간, 처리량 등을 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

23.7, 22.7, 22.4, 21.5, 20.6

핵심 127 화이트박스 테스트 (White Box Test) 투명

2405601

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

- 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.

23.7, 21.5, 20.8, 20.6

핵심 128 화이트박스 테스트의 종류

2405602

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

22.4

핵심 129 화이트박스 테스트의 검증 기준

2405603

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)이라고도 불리며, 소스 코드의 모든 조건문에 대해 조건이 True인 경우와 False인 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/ Condition Coverage)	분기 검증 기준과 조건 검증 기준을 모두 만족하는 설계로, 조건문이 True인 경우와 False인 경우에 따라 조건 검증 기준의 입력 데이터를 구분하는 테스트 케이스 설계

23.7, 22.7, 21.5

핵심 130 블랙박스 테스트 (Black Box Test)

2405604

블랙박스 테스트는 소프트웨어가 수행할 특정 기능을 알기 위해서 각 기능이 완전히 작동되는 것을 입증하는 테스트로, 기능 테스트라고도 한다.

- 프로그램의 구조를 고려하지 않기 때문에 테스트 케이스는 프로그램 또는 모듈의 요구나 명세를 기초로 결정한다.
- 소프트웨어 인터페이스에서 실시되는 테스트이다.
- 부정확하거나 누락된 기능, 인터페이스 오류, 자료 구조나 외부 데이터베이스 접근에 따른 오류, 행위나 성능 오류, 초기화와 종료 오류 등을 발견하기 위해 사용되며, 테스트 과정의 후반부에 적용된다.

23.5, 21.5, 21.3, 20.9, 20.8, 20.6

핵심 131 블랙박스 테스트의 종류

2405605

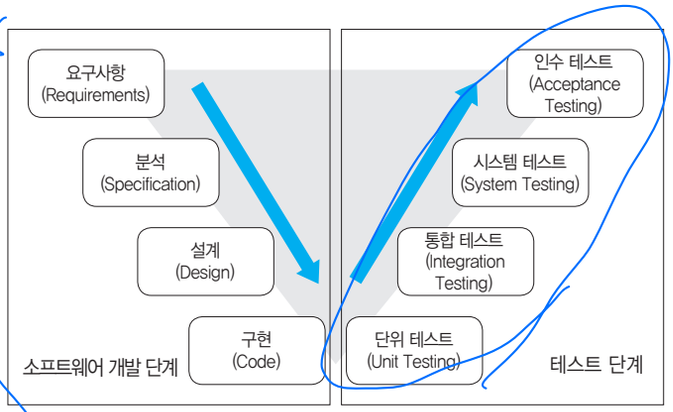
동치 분할 검사 (Equivalence Partitioning Testing, 동치 클래스 분해)	• 입력 자료에 초점을 맞춰 테스트 케이스(동치 클래스)를 만들고 검사하는 방법으로 동등 분할 기법이라고도 함 • 프로그램의 입력 조건에 타당한 입력 자료와 타당하지 않은 입력 자료의 개수를 균등하게 하여 테스트 케이스를 정하고, 해당 입력 자료에 맞는 결과가 출력되는지 확인하는 기법
경계값 분석 (Boundary Value Analysis)	• 입력 자료에만 치중한 동치 분할 기법을 보완하기 위한 기법 • 입력 조건의 중간값보다 경계값에서 오류가 발생될 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 함
비교 검사 (Comparison Testing)	여러 버전의 프로그램에 동일한 테스트 자료를 제공하여 동일한 결과가 출력되는지 테스트하는 기법

핵심 132 개발 단계에 따른 애플리케이션 테스트

2405701

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

- 애플리케이션 테스트와 소프트웨어 개발 단계를 연결하여 표현한 것을 V-모델이라 한다.



소프트웨어 생명 주기의 V-모델

23.5, 22.4, 21.8, 21.5

핵심 133 단위 테스트(Unit Test)

2405702

단위 테스트는 코딩 직후 소프트웨어 설계의 최소 단위인 모듈이나 컴포넌트에 초점을 맞춰 테스트하는 것이다.

- 단위 테스트에서는 인터페이스, 외부적 I/O, 자료 구조, 독립적 기초 경로, 오류처리 경로, 경계 조건 등을 검사한다.
- 단위 테스트는 사용자의 요구사항을 기반으로 한 기능성 테스트를 최우선으로 수행한다.
- 단위 테스트는 구조 기반 테스트와 명세 기반 테스트로 나뉘지만 주로 구조 기반 테스트를 시행한다.
- **단위 테스트로 발견 가능한 오류** : 알고리즘 오류에 따른 원치 않는 결과, 탈출구가 없는 반복문의 사용, 틀린 계산 수식에 의한 잘못된 결과

핵심 134 통합 테스트(Integration Test)

2405703

통합 테스트는 단위 테스트가 완료된 모듈들을 결합하여 하나의 시스템으로 완성시키는 과정에서의 테스트를 의미한다.

비점진적 통합 방식	• 단계적으로 통합하는 절차 없이 모든 모듈이 미리 결합되어 있는 프로그램 전체를 테스트하는 방법으로, 빅뱅 통합 테스트 방식 • 규모가 작은 소프트웨어에 유리하며 단시간 내에 테스트가 가능함 • 전체 프로그램을 대상으로 하기 때문에 오류 발견 및 장애 위치 파악 및 수정이 어려움
점진적 통합 방식	• 모듈 단위로 단계적으로 통합하면서 테스트하는 방법으로, 하향식, 상향식, 혼합식 통합 방식이 있음 • 오류 수정이 용이하고, 인터페이스와 연관된 오류를 완전히 테스트할 가능성이 높음

핵심 135 시스템 테스트(System Test)

2405704

시스템 테스트는 개발된 소프트웨어가 해당 컴퓨터 시스템에서 완벽하게 수행되는가를 점검하는 테스트이다.

- 환경적인 장애 리스크를 최소화하기 위해서는 실제 사용 환경과 유사하게 만든 테스트 환경에서 테스트를 수행해야 한다.
- 시스템 테스트는 기능적 요구사항과 비기능적 요구사항으로 구분하여 각각을 만족하는지 테스트한다.

23.7, 23.5, 21.3, 20.9, 20.8, 20.6

핵심 136 인수 테스트 (Acceptance Test)

2405705

인수 테스트는 개발한 소프트웨어가 사용자의 요구사항을 충족하는지에 중점을 두고 테스트하는 방법이다.

- 인수 테스트는 개발한 소프트웨어를 사용자가 직접 테스트한다.
- 인수 테스트에 문제가 없으면 사용자는 소프트웨어를 인수하게 되고, 프로젝트는 종료된다.
- 인수 테스트의 종류

알파 테스트	• 개발자의 장소에서 사용자가 개발자 앞에서 행하는 테스트 기법 • 테스트는 통제된 환경에서 행해지며, 오류와 사용상의 문제점을 사용자와 개발자가 함께 확인하면서 기록함
베타 테스트	• 선정된 최종 사용자가 여러 명의 사용자 앞에서 행하는 테스트 기법으로, 필드 테스팅(Field Testing)이라고도 불림 • 실업무를 가지고 사용자가 직접 테스트하는 것으로, 개발자에 의해 제어되지 않은 상태에서 테스트가 행해지며, 발견된 오류와 사용상의 문제점을 기록하고 개발자에게 주기적으로 보고함

22.4, 20.8

핵심 137 하향식 통합 테스트 (Top Down Integration Test)

2405802

하향식 통합 테스트는 프로그램의 상위 모듈에서 하위 모듈 방향으로 통합하면서 테스트하는 기법이다.

- 주요 제어 모듈을 기준으로 하여 아래 단계로 이동하면서 통합하는데, 이때 깊이 우선 통합법이나 넓이 우선 통합법을 사용한다.
- 테스트 초기부터 사용자에게 시스템 구조를 보여줄 수 있다.
- 상위 모듈에서는 테스트 케이스를 사용하기 어렵다.

22.4

핵심 138 상향식 통합 테스트 (Bottom Up Integration Test)

2405803

상향식 통합 테스트는 프로그램의 하위 모듈에서 상위 모듈 방향으로 통합하면서 테스트하는 기법이다.

- 가장 하위 단계의 모듈부터 통합 및 테스트가 수행되므로 스텁(Stub)은 필요하지 않지만, 하나의 주요 제어 모듈과 관련된 종속 모듈의 그룹인 클러스터(Cluster)가 필요하다.

22.7, 21.8, 21.3, 20.6

핵심 139 테스트 드라이버와 테스트 스텁의 차이점

2405831

구분	드라이버(Driver)	스텁(Stub)
개념	테스트 대상의 하위 모듈을 호출하는 도구로, 매개 변수(Parameter)를 전달하고, 모듈 테스트 수행 후의 결과를 도출함	제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구로, 일시적으로 필요한 조건만을 가지고 있는 시험용 모듈임
필요 시기	상위 모듈 없이 하위 모듈이 있는 경우 하위 모듈 구동	상위 모듈은 있지만 하위 모듈이 없는 경우 하위 모듈 대체
테스트 방식	상향식(Bottom Up) 테스트	하향식(Top-Down) 테스트
공통점	소프트웨어 개발과 테스트를 병행할 경우 이용	
차이점	• 이미 존재하는 하위 모듈과 존재하지 않는 상위 모듈 간의 인터페이스 역할을 함 • 소프트웨어 개발이 완료되면 드라이버는 본래의 모듈로 교체됨	• 일시적으로 필요한 조건만을 가지고 임시로 제공되는 가짜 모듈의 역할을 함 • 시험용 모듈이기 때문에 일반적으로 드라이버보다 작성하기 쉬움

데이터의 입출력을 확인하기 위해
더미 모듈인 Driver 생성

영어

핵심 140 회귀 테스팅 (Regression Testing)

2405805

회귀 테스트는 이미 테스트된 프로그램의 테스팅을 반복하는 것으로, 통합 테스트로 인해 변경된 모듈이나 컴포넌트에 새로운 오류가 있는지 확인하는 테스트이다.

- 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 테스팅하는 것이 가장 좋지만 시간과 비용이 많이 필요하므로 기존 테스트 케이스 중 변경된 부분을 테스트할 수 있는 테스트 케이스만을 선정하여 수행한다.

핵심 141 애플리케이션 테스트 프로세스

2405901

애플리케이션 테스트 프로세스는 개발된 소프트웨어가 사용자의 요구대로 만들어졌는지, 결함은 없는지 등을 테스트하는 절차이다.

- 순서

테스트 계획	프로젝트 계획서, 요구 명세서 등을 기반으로 테스트 목표를 정의하고 테스트 대상 및 범위를 결정함
테스트 분석 및 디자인	테스트의 목적과 원칙을 검토하고 사용자의 요구 사항을 분석함
테스트 케이스 및 시나리오 작성	테스트 케이스의 설계 기법에 따라 테스트 케이스를 작성하고 검토 및 확인한 후 테스트 시나리오를 작성함
테스트 수행	• 테스트 환경을 구축한 후 테스트를 수행함 • 테스트의 실행 결과를 측정하여 기록함
테스트 결과 평가 및 리포팅	테스트 결과를 비교 분석하여 테스트 결과서를 작성함
결함 추적 및 관리	테스트를 수행한 후 결함이 어디에서 발생했는지, 어떤 종류의 결함인지 등 결함을 추적하고 관리함

22.4

핵심 142 테스트 케이스(Test Case)

2406001

테스트 케이스는 구현된 소프트웨어가 사용자의 요구사항을 정확하게 준수했는지를 확인하기 위해 설계된 입력값, 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 시스템 설계 단계에서 작성하는 것이 가장 이상적이다.

핵심 143 테스트 시나리오 (Test Scenario)

2406003

테스트 시나리오는 테스트 케이스를 적용하는 순서에 따라 여러 개의 테스트 케이스들을 묶은 집합으로, 테스트 케이스들을 적용하는 구체적인 절차를 명세한 문서이다.

- 테스트 시나리오에는 테스트 순서에 대한 구체적인 절차, 사전 조건, 입력 데이터 등이 설정되어 있다.

23.2, 22.4, 20.9

핵심 144 테스트 오라클(Test Oracle)

2406005

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

- 테스트 오라클은 결과를 판단하기 위해 테스트 케이스에 대한 예상 결과를 계산하거나 확인한다.

22.7

핵심 145 테스트 오라클의 종류

2406006

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

23.2, 21.8, 21.5

핵심 146 테스트 자동화 도구

2406104

테스트 자동화의 개념

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

테스트 자동화 도구의 유형

정적 분석 도구 (Static Analysis Tools)	프로그램을 실행하지 않고 분석하는 도구로, 소스 코드에 대한 코딩 표준, 코딩 스타일, 코드 복잡도 및 남은 결함 등을 발견하기 위해 사용된다.
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 입력을 작성함 • 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 작성한 후 실행함 • **데이터 주도 접근 방식** : 스프레드시트에 테스트 데이터를 저장하고, 이를 읽어 실행하는 방식 • **키워드 주도 접근 방식** : 스프레드시트에 테스트를 수행할 동작을 나타내는 키워드와 테스트 데이터를 저장하여 실행하는 방식
성능 테스트 도구 (Performance Test Tools)	애플리케이션의 처리량, 응답 시간, 경과 시간, 자원 사용률 등을 인위적으로 적용한 가상의 사용자를 만들어 테스트를 수행함으로써 성능의 목표 달성 여부를 확인함
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 있음
테스트 하네스 도구 (Test Harness Tools)	• 테스트 하네스는 애플리케이션의 컴포넌트 및 모듈을 테스트하는 환경의 일부분으로, 테스트를 지원하기 위해 생성된 코드와 데이터를 의미함 • 테스트 하네스 도구는 테스트가 실행될 환경을 시뮬레이션 하여 컴포넌트 및 모듈이 정상적으로 테스트되도록 함

22.3

핵심 147 테스트 하네스(Test Harness)의 구성 요소

2406131

- 테스트 드라이버(Test Driver) : 테스트 대상의 하위 모듈을 호출하고, 매개변수(Parameter)를 전달하고, 모듈 테스트 수행 후의 결과를 도출하는 도구
- 테스트 스텁(Test Stub) : 제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구로, 일시적으로 필요한 조건만을 가지고 있는 테스트용 모듈
- 테스트 슈트(Test Suites) : 테스트 대상 컴포넌트나 모듈, 시스템에 사용되는 테스트 케이스의 집합
- 테스트 케이스(Test Case) : 사용자의 요구사항을 정확하게 준수했는지 확인하기 위한 입력 값, 실행 조건, 기대 결과 등으로 만들어진 테스트 항목의 명세서
- 테스트 스크립트(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

21.8

핵심 148 결함(Fault)

2406201

결함은 오류 발생, 작동 실패 등과 같이 소프트웨어가 개발자가 설계한 것과 다르게 동작하거나 다른 결과가 발생되는 것을 의미한다.

- 사용자가 예상한 결과와 실행 결과 간의 차이나 업무 내용과의 불일치 등으로 인해 변경이 필요한 부분도 모두 결함에 해당된다.

핵심 149 애플리케이션 성능 분석

2406301

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

- 애플리케이션 성능 측정 지표

처리량 (Throughput)	일정 시간 내에 애플리케이션이 처리하는 일의 양
응답 시간 (Response Time)	애플리케이션에 요청을 전달한 시간부터 응답이 도착할 때까지 걸린 시간
경과 시간 (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

23.7, 22.7, 20.6

핵심 150 빅오 표기법(Big-O Notation)

2406403

빅오 표기법은 알고리즘의 실행시간이 최악일 때를 표기하는 방법으로, 신뢰성이 떨어지는 오메가 표기법이나 평가하기 까다로운 세타 표기법에 비해 성능을 예측하기 용이하여 주로 사용된다.

- 일반적인 알고리즘에 대한 최악의 시간 복잡도를 빅오 표기법으로 표현하면 다음과 같다.

O(1)	입력값(n)에 관계 없이 일정하게 문제 해결에 하나의 단계만을 거침 예 스택의 삽입(Push), 삭제(Pop)
$O(\log_2 n)$	문제 해결에 필요한 단계가 입력값(n) 또는 조건에 의해 감소함 예 이진 트리(Binary Tree), 이진 검색(Binary Search)
O(n)	문제 해결에 필요한 단계가 입력값(n)과 1:1의 관계를 가짐 예 for문
$O(n\log_2 n)$	문제 해결에 필요한 단계가 $n(\log_2 n)$번만큼 수행됨 예 힙 정렬(Heap Sort), 2-Way 합병 정렬(Merge Sort)
$O(n^2)$	문제 해결에 필요한 단계가 입력값(n)의 제곱만큼 수행됨 예 삽입 정렬(Insertion Sort), 쉘 정렬(Shell Sort), 선택 정렬(Selection Sort), 버블 정렬(Bubble Sort), 퀵 정렬(Quick Sort)
$O(2^n)$	문제 해결에 필요한 단계가 2의 입력값(n) 제곱만큼 수행됨 예 피보나치 수열(Fibonacci Sequence)

23.2, 20.8

핵심 151 순환 복잡도

2406404

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

- 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 G에서 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 : E는 화살표 수, N은 노드의 수

22.7, 22.3, 21.8, 21.5, 20.9, 20.8, 20.6

핵심 152 소스 코드 최적화

2406501

소스 코드 최적화는 나쁜 코드(Bad Code)를 배제하고, 클린 코드(Clean Code)로 작성하는 것이다.

- 클린 코드(Clean Code) : 누구나 쉽게 이해하고 수정 및 추가할 수 있는 단순, 명료한 코드, 즉 잘 작성된 코드를 의미함
- 나쁜 코드(Bad Code)
  - 프로그램의 로직(Logic)이 복잡하고 이해하기 어려운 코드로, 스파게티 코드와 외계인 코드가 여기에 해당함
  - 스파게티 코드 : 코드의 로직이 서로 복잡하게 얽혀 있는 코드
  - 외계인 코드 : 아주 오래되거나 참고문서 또는 개발자가 없어 유지보수 작업이 어려운 코드
- 클린 코드 작성 원칙

가독성	• 누구든지 코드를 쉽게 읽을 수 있도록 작성함 • 코드 작성 시 이해하기 쉬운 용어를 사용하거나 들여쓰기 기능 등을 사용함
단순성	• 코드를 간단하게 작성함 • 한 번에 한 가지를 처리하도록 코드를 작성하고 클래스/메소드/함수 등을 최소 단위로 분리함
의존성 배제	• 코드가 다른 모듈에 미치는 영향을 최소화함 • 코드 변경 시 다른 부분에 영향이 없도록 작성함
중복성 최소화	• 코드의 중복을 최소화함 • 중복된 코드는 삭제하고 공통된 코드를 사용함
추상화	상위 클래스/메소드/함수에서는 간략하게 애플리케이션의 특성을 나타내고, 상세 내용은 하위 클래스/메소드/함수에서 구현함

23.7, 22.7, 21.8, 20.9, 20.6

핵심 153 소스 코드 품질 분석 도구

2406503

소스 코드 품질 분석 도구는 소스 코드의 코딩 스타일, 코드에 설정된 코딩 표준, 코드의 복잡도, 코드에 존재하는 메모리 누수 현상, 스레드 결함 등을 발견하기 위해 사용하는 분석 도구로, 크게 정적 분석 도구와 동적 분석 도구로 나뉜다.

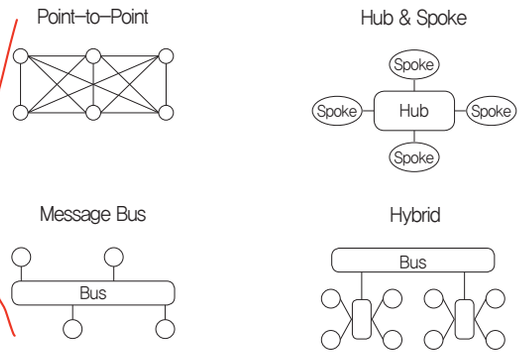
- 정적 분석 도구
  - 작성한 소스 코드를 실행하지 않고 코딩 표준이나 코딩 스타일, 결함 등을 확인하는 코드 분석 도구이다.

- 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.
- 종류 : pmd, cppcheck, SonarQube, checkstyle, ccm, cobertura 등

• 동적 분석 도구

- 작성한 소스 코드를 실행하여 코드에 존재하는 메모리 누수, 스레드 결함 등을 분석하는 도구이다.
- 종류 : Avalanche, Valgrind 등

보면서 이해



23.2, 22.7, 21.5, 20.9, 20.6

핵심 154 EAI(Enterprise Application Integration)

2459911

- EAI는 기업 내 각종 애플리케이션 및 플랫폼 간의 정보 전달, 연계, 통합 등 상호 연동이 가능하게 해주는 솔루션이다.
- EAI는 비즈니스 간 통합 및 연계성을 증대시켜 효율성 및 각 시스템 간의 확정성(Determinacy)을 높여 준다.
- EAI의 구축 유형은 다음과 같다.

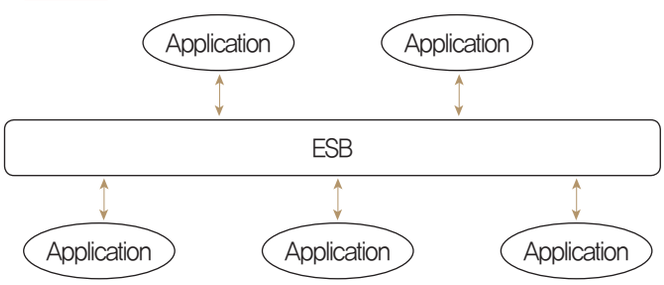
유형	기능
Point-to-Point	• 가장 기본적인 애플리케이션 통합 방식으로, 애플리케이션을 1:1로 연결함 • 변경 및 재사용이 어려움
Hub & Spoke	• 단일 접점인 허브 시스템을 통해 데이터를 전송하는 중앙 집중형 방식 • 확장 및 유지 보수가 용이하다. • 허브 장애 발생 시 시스템 전체에 영향을 미침
Message Bus (ESB 방식)	• 애플리케이션 사이에 미들웨어를 두어 처리하는 방식 • 확장성이 뛰어나며 대용량 처리가 가능함
Hybrid	• Hub & Spoke와 Message Bus의 혼합 방식 • 그룹 내에서는 Hub & Spoke 방식을, 그룹 간에는 Message Bus 방식을 사용함 • 필요한 경우 한 가지 방식으로 EAI 구현이 가능함 • 데이터 병목 현상을 최소화할 수 있음

핵심 155 ESB (Enterprise Service Bus)

2459912

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 애플리케이션과의 결합도(Coupling)를 약하게(Loosely) 유지한다.
- 관리 및 보안 유지가 쉽고, 높은 수준의 품질 지원이 가능하다.



23.5, 22.4, 20.6

핵심 156 JSON(JavaScript Object Notation)

2459913

JSON은 속성-값 쌍(Attribute-Value Pairs)으로 이루어진 데이터 객체를 전달하기 위해 사람이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷이다.

- 비동기 처리에 사용되는 AJAX에서 XML을 대체하여 사용되고 있다.

P272 스니핑
네트워크 상에 돌아다니는 패킷들의 내용을 엿보는 행위

핵심 157 XML(eXtensible Markup Language)

2459914

XML은 특수한 목적을 갖는 마크업 언어를 만드는 데 사용되는 다목적 마크업 언어이다.

- 웹 페이지의 기본 형식인 HTML의 문법이 각 웹 브라우저에서 상호 호환적이지 못하다는 문제와 SGML의 복잡함을 해결하기 위하여 개발되었다.

핵심 158 AJAX(Asynchronous JavaScript and XML)

19.8

2459915

AJAX는 자바 스크립트(JavaScript) 등을 이용하여 클라이언트와 서버 간에 XML 데이터를 교환 및 제어함으로써 이용자가 웹 페이지와 자유롭게 상호 작용할 수 있도록 하는 비동기 통신 기술을 의미한다.

핵심 159 인터페이스 보안 기능 적용

22.7, 21.5, 20.9, 20.8, 20.6

2407203

인터페이스 보안 기능은 일반적으로 네트워크, 애플리케이션, 데이터베이스 영역에 적용한다.

네트워크 영역	• 인터페이스 송 · 수신 간 스니핑(Sniffing) 등을 이용한 데이터 탈취 및 변조 위협을 방지하기 위해 네트워크 트래픽에 대한 암호화를 설정함 • 암호화는 인터페이스 아키텍처에 따라 IPSec, SSL, S-HTTP 등의 다양한 방식으로 적용함
애플리케이션 영역	소프트웨어 개발 보안 가이드를 참조하여 애플리케이션 코드 상의 보안 취약점을 보완하는 방향으로 애플리케이션 보안 기능을 적용함
데이터베이스 영역	데이터베이스, 스키마, 엔티티의 접근 권한과 프로시저(Procedure), 트리거(Trigger) 등 데이터베이스 동작 객체의 보안 취약점에 보안 기능을 적용함

※ IPsec(IP Security) : 네트워크 계층에서 IP 패킷 단위의 데이터 변조 방지 및 은닉 기능을 제공하는 프로토콜로, 암호화 수행시 양방향 암호화를 지원함

21.3, 20.6

핵심 160 데이터 무결성 검사 도구

2407204

- 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 이용하여 이를 감지할 수 있다.
- 해시(Hash) 함수를 이용하여 현재 파일 및 디렉토리의 상태를 DB에 저장한 후 감시하다가 현재 상태와 DB의 상태가 달라지면 관리자에게 변경 사실을 알려준다.
- 대표적인 데이터 무결성 검사 도구에는 Tripwire, AIDE, Samhain, Claymore, Slipwire, Fcheck 등이 있다.

23.7, 23.2, 22.7, 21.5, 20.9, 20.8

핵심 160 인터페이스 구현 검증 도구

2407402

- 인터페이스 구현을 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 여러 번 작성하지 않게 도와주고, 테스트마다 예상 결과를 기억할 필요가 없게 하는 자동화된 해법을 제공하는 단위 테스트 프레임워크 • Smalltalk에 처음 적용되어 SUnit이라는 이름이었으나 Java용의 JUnit, C++용의 CppUnit, .NET용의 NUnit, Http용의 HttpUnit 등 다양한 언어에 적용되면서 xUnit으로 통칭되고 있음
STAF	• 서비스 호출 및 컴포넌트 재사용 등 다양한 환경을 지원하는 테스트 프레임워크 • 크로스 플랫폼, 분산 소프트웨어 테스트 환경을 조성할 수 있도록 지원함 • 분산 소프트웨어의 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 애플리케이션 테스트 프레임워크
watir	Ruby를 사용하는 애플리케이션 테스트 프레임워크

핵심 162 APM(Application Performance Management/Monitoring)

2407431

APM은 애플리케이션의 성능 관리를 위해 접속자, 자원 현황, 트랜잭션 수행 내역, 장애 진단 등 다양한 모니터링 기능을 제공하는 도구를 의미한다.

- APM은 리소스 방식과 엔드투엔드(End-to-End)의 두 가지 유형이 있다.
  - 리소스 방식 : Nagios, Zabbix, Cacti 등
  - 엔드투엔드 방식 : VisualVM, 제니퍼, 스카우터 등

3과목 데이터베이스 구축

요개놈물구

핵심 163 데이터베이스 설계 순서

2407603

요구 조건 분석	요구 조건 명세서 작성
↓	
개념적 설계	개념 스키마, 트랜잭션 모델링, E-R 모델
↓	
논리적 설계	목표 DBMS에 맞는 논리 스키마 설계, 트랜잭션 인터페이스 설계
↓	
물리적 설계	목표 DBMS에 맞는 물리적 구조의 데이터로 변환
↓	
구현	목표 DBMS의 DDL(데이터 정의어)로 데이터베이스 생성, 트랜잭션 작성

23.2, 22.4

핵심 164 개념적 설계(정보 모델링, 개념화)

2407605

개념적 설계란 정보의 구조를 얻기 위하여 현실 세계의 무한성과 계속성을 이해하고, 다른 사람과 통신하기 위하여 현실 세계에 대한 인식을 추상적 개념으로 표현하는 과정이다.

- 개념적 설계 단계에서는 개념 스키마 모델링과 트랜잭션 모델링을 병행 수행한다.
- 개념적 설계 단계에서는 요구 분석 단계에서 나온 결과인 요구 조건 명세를 DBMS에 독립적인 E-R 다이어그램으로 작성한다.
- DBMS에 독립적인 개념 스키마를 설계한다.

22.7, 20.6

핵심 165 논리적 설계(데이터 모델링)

2407606

논리적 설계 단계란 현실 세계에서 발생하는 자료를 컴퓨터가 이해하고 처리할 수 있는 물리적 저장장치에 저장할 수 있도록 변환하기 위해 특정 DBMS가 지원하는 논리적 자료 구조로 변환(mapping)시키는 과정이다.