





핵심 요약 노트 이론편

정보 처리 기사 실기

목 차

1. 소프트웨어 구축	1
2. 데이터베이스	6
3. 운영체제	9
4. 네트워크	11
5. 정보보안	14
6. 기타 용어	16
7. SQL문 활용	18

안녕하세요. **꿈꾸는라이언**입니다.


먼저 정처기 필기 합격을 진심으로 축하드립니다!    

이제 정처기 실기를 앞두고 이 요약노트를 찾으셨을텐데요,

처음 준비하시는 분들을 위해 **간단한 실기 공부 팁**을 공유드리려 해요~


꼭 읽어주시고 참고하셔서 우리 모두 정처기 1트 합격해봐요!

1. **실기도 기출이 가장 중요해요!** 물론 필기만큼이나 기출 출제율이 높은 건 아니에요. 그럼에도 기출이 아예 출제되지 않는 것도 아닙니다! 예를 들어 **최근 23년 3회 실기에서는 5문항이 이전 기출에서 유사하게 출제되었는데요**, 전체 20문항인 실기 시험을 생각하면 기출 연계율이 결코 낮은게 아니죠! 사실 기출에서 ‘단 한 문제’라도 출제된다면 수험생 입장에서는 너무나 감사한 일이고 당연히 기출을 우선으로 공부하셔야 해요. 20년도 개정 이후로 출제된 실기 기출이 지금까지 200 여 문항 정도가 있을텐데요. 개정 이후 전체 기출 정리가 시간 관계 상 **어렵다면 최신 기준 2~3년치 기출만이라도 정리하고 시험에 응시하는걸 적극 권장드려요!**
2. **프로그래밍 언어를 반드시 공부하셔야 돼요!** 20년에 출제된 예전 기출을 보시면 프로그래밍 언어는 4~5문제로 전체 20문항 중에서 그 비중이 적었어요. 실제 당시에는 프로그래밍 언어를 모르더라도 합격에는 문제가 없었죠. 그런데 안타깝지만 최신 기출 동향을 살펴보면 프로그래밍 언어 문제가 8~10문항으로 거의 절반에 달하고 있습니다! 이 말인 즉슨 프로그래밍 언어를 아예 모른다면 실기 합격에 필요한 60점을 물리적으로 받을 수 없어요. π 원래 해당 요약노트에도 프로그래밍 언어를 간단하게라도 요약하고자 했습니다만 최신 기출 문제를 보면 난이도가 점점 높아지고 있고 단순 암기가 아닌 종합적인 ‘이해’를 해야만 풀 수 있는 문제들이 자주 출제되고 있어 단 몇 페이지로 방대한 프로그래밍 언어를 요약하기가 매우 어려웠습니다. **그래서 정처기 실기 중 프로그래밍 언어 부분은 별도 기출풀이집을 제작해서 기출 풀이를 통한 이론 정리로 대체하려고 해요!** (지금 현업 개발자 지인과 함께 열심히 제작하고 있으니, 조금만 기다려주세요 $\pi\pi$) 정처기에 출제되는 프로그래밍 언어 관련해서는 사실 유튜브에만 검색하셔도 이론부터 실제 기출 풀이까지 상세한 강의를 쉽게 찾아볼 수 있으니 꿈꾸는라이언 요약노트와 함께 반드시 프로그래밍 언어도 같이 병행해서 공부 부탁드릴게요!
3. **해당 요약노트는 프로그래밍 언어를 제외한 나머지 이론 내용과 SQL문에 대한 내용을 요약한 자료예요!** 필기 요약노트를 먼저 보신 분들은 상당 부분 겹치는 내용이 많은 것을 아실텐데요, 그럴 수밖에 없는게 필기와 실기 이론 부분의 출제 범위가 다르지 않거든요. 다만 총 5과목으로 나뉘서 과목별로 출제되는 필기와 다르게 실기의 경우 실제 시험에서는 과목별로 나뉘서 출제되지 않습니다. 기출을 보시면 아시겠지만 시험 과목 순서와 상관없이 이론 내용들이 프로그래밍 언어와 함께 무작위로 출제되고 있어요. π 이로 인해 시험 범위 순서대로 암기하신 수험생 분들은 실제 시험 때 관련 키워드가 어느 과목의 무슨 파트에서 출제되었는지 혼란스러울 수 있고요. 따라서 정처기 출제 기준으로 공식 가이드된 12개 과목의 순서가 아닌 각 내용별 연관성을 고려하여 **총 7개의 파트**로 다시 나뉘 방대한 시험 범위를 암기하시는데 편하게 정리했어요!

정처기 자격증이 필요하신 수험생들의 빠른 1트 합격을 위해 합격에 필요한 핵심 키워드들을 놓치지 않으면서 비전공자인 저도 이해할 수 있도록 최대한 꼼꼼하게 요약해봤어요! 필기보다 실기가 아무래도 낯선 프로그래밍 언어로 익숙하지 않아 많이 힘드시겠지만 끝까지 포기하지 마시고 최선을 다하신다면 모두들 좋은 결과로 마무리하실거라 믿어요. 

이 <정처기 실기 요약노트 - 이론편>으로 공부하시면서 궁금하신 사항이나 의견 있으시면 언제든지 편하게 메일이나 구매하셨던 스토어 톡톡문의로 연락주시고요~ 문의주신 모든 분들께 제가 아는 선에서 성심껏 빠르게 답변해드릴게요!

그럼 정처기 합격을 시작으로 각자 계획하셨던 목표와 꿈 모두 이루시길 진심으로 바라겠습니다!

합격 미리 축하드립니다!! 



■ 소프트웨어 생명 주기 (Software Development Life Cycle, SDLC)

① 프로젝트 계획 ▶ ② 요구 분석 ▶ ③ 설계 ▶ ④ 구현 ▶ ⑤ 테스트 ▶ ⑥ 유지 보수

폭포수	선형 순차적 개발 / 고전적, 전통적 개발 모형 / Step-by-Step
프로토타입	고객의 need 파악 위해 전본/시제품 을 통해 최종 결과 예측 인터페이스 중심 / 요구사항 변경 용이
나선형 (Spiral)	폭포수 + 프로토타입 + 위험 분석 기능 추가 (위험 관리/최소화) 점진적 개발 과정 반복 / 정밀하며 유지보수 과정 필요 X ★ 계획 수립 → 위험 분석 → 개발 및 검증 → 고객 평가
애자일 (Agile)	일정한 짧은 주기(Sprint 또는 Iteration) 반복하며 개발 진행 → 고객 요구사항에 유연한 대응 (고객 소통/상호작용 중심) Ex. XP(eXtreme Programming), Scrum, FDD (기능중심), 린 (LEAN), DSDM (Dynamic System Development Method)

하향식 설계 (Top-down): **절차 지향** (순차적) / 최상위 컴포넌트 설계 후 하위 기능 부여
→ 테스트 초기부터 사용자에게 시스템 구조 제시 가능

상향식 설계 (Bottom-up): **객체 지향** / 최하위 모듈 먼저 설계 후 이들을 결합하고 검사
→ 인터페이스 구조 변경 시 상위 모듈도 같이 변경 필요하여 **기능 추가 어려움**

* **Component**: 명백한 역할을 가지며 재사용되는 모든 단위 / 인터페이스 통해 접근 가능

■ 익스트림 프로그래밍 (eXtreme Programming, XP)

- 고객의 요구사항을 유연하게 대응하기 위해 고객 참여와 신속한 개발 과정을 반복

- 5가지 핵심 가치: 용기 / 단순성 / 의사소통 / 피드백 / 존중

※ **피드백**: 시스템의 상태와 사용자의 지시에 대한 효과를 보여줘서 사용자가 명령에 대한 진행 상황과 표시된 내용을 해석할 수 있게 도와줌

- 기본 원리: 전체 팀 / 소규모 릴리즈 / 테스트 주도 개발 / 지속적인 통합 /

공동 소유권 (Collective ownership) / 짝(pair) 프로그래밍 /

디자인 개선 (리팩토링) / 애자일(Agile) 방법론 활용

상식적 원리 및 경험 추구, **개발 문서보단 소스코드에 중점** (문서화 X)

① 프로젝트 계획

▶ 하향식 비용 산정 기법

- 전문가 감정 기법: 외부 전문가에게 비용 산정 의뢰 (객관적)

- 델파이 기법: 한 명의 조정자와 여러 전문가의 의견을 종합하여 산정

▶ 상향식 비용 산정 기법

- 프로젝트 세부 작업 단위 별로 비용 정산 후 전체 비용을 산정하는 방법

[종류]

. **LOC (Source Line of Code)**: 코드 라인 총 수 / 생산성 / 개발 참여 인원 등으로 계산

$$\rightarrow \text{낙관치(a), 비관치(b), 기대치(c)를 측정/예측하여 비용 산정} = \frac{a + 4c + b}{6}$$

. **개발 단계별 인일 수 (Effort Per Task)**: LOC 기법 보완 / 생명 주기 각 단계별로 산정

▶ 수학적 비용 산정

① **COCOMO (Constructive Cost Model)**: 보험 (Boehm) 제안 / 원시코드 라인 수 기반

→ 비용 견적 강도 분석 및 비용 견적의 유연성이 높아 널리 통용됨

→ 같은 프로젝트라도 성격에 따라 비용이 다르게 산정

유형	조직형 (Organic)	중,소규모 SW용 / 5만 라인(50KDSI) 이하
	반분리형 (Semi-detached)	30만 라인 (300KDSI) 이하의 트랜잭션 처리 시스템
	내장형 (Embedded)	30만 라인 (300KDSI) 이상의 최대형 규모 SW 관리

② **PUTNAM**: SW 생명주기 전 과정에 사용될 **노력의 분포**를 이용한 비용 산정

Rayleigh Norden 곡선의 노력 분포도를 기초로 함

★ **SLIM**: Rayleigh-Norden 곡선 / Putnam 모형 기초로 개발된 자동화 추정 도구

③ **Function Point (FP)**: SW 기능 증대 요인에 **가중치 부여** 후 합산하여 기능점수 산출

→ SW 기능 증대 요인: 자료 입력 (입력 양식) / 정보 출력 (출력 보고서) /

명령어 (사용자 질의수) / 데이터 파일 / 인터페이스

★ **ESTIMACS**: FP 모형을 기반으로 하여 개발된 자동화 추정 도구

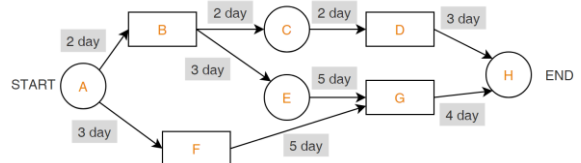
■ 개발 일정 산정

- **WBS (Work Breakdown Structure)**: 프로젝트 목표 달성을 위한 활동과 업무를 세분화
→ 전체 프로젝트를 분할 / 수행 업무 식별, 일정 및 비용

▶ 네트워크 차트

PERT (Program Evaluation and Review Technique)	. 프로젝트 작업 상호관계를 네트워크로 표현 . 원 노드(작업)와 간선(화살표)으로 구성 (@불확실한 상황) → 간선에는 각 작업별 낙관치/기대치/비관치를 기재
CPM (Critical Path method)	. 미국 Dupont 회사에서 화학공장 유지/관리 위해 개발 . 노드(작업) / 간선(작업 전후 의존 관계) / 박스(이정표) 구성 . 간선(화살표)의 흐름에 따라 작업 진행 (@확실한 상황)

[예시] CPM 네트워크 - **임계 경로**: 14일 (A-B-E-G-H) ← 경로상 가장 오래 걸리는 시간



▶ **간트 차트**: 각 작업의 시작/종료 일정을 막대 바(Bar) 도표를 이용하여 표현
시간선(Time-line) 차트 (수평 막대 길이 = 작업기간)
작업 경로는 표현 불가 / 계획 변화에 대한 적응성이 낮음

② 요구사항 분석

■ **요구사항**: 어떠한 문제를 해결하기 위해 필요한 조건 및 제약사항을 요구
소프트웨어 개발/유지 보수 과정에 필요한 기준과 근거 제공

▶ 요구사항의 유형

기능적 요구사항	실제 시스템 수행에 필요한 기능 관련 요구사항 ex. 금융 시스템은 조회/인출/입금/송금 기능이 있어야 한다.
비기능적 요구사항	성능, 보안, 품질, 안정성 등 실제 수행에 보조적인 요구사항 Ex. 모든 화면이 3초 이내에 사용자에게 보여야 한다.

▶ 요구사항 개발 프로세스 ★순서 중요

① 도출/추출	이해관계자들이 모여 요구사항 정의 (식별하고 이해하는 과정) Ex. 인터뷰, 설문, 브레인스토밍, 청취, 프로토타이핑, 유스케이스
② 분석	사용자 요구사항에 타당성 조사 / 비용 및 일정에 대한 제약 설정 Ex. 관찰, 개념 모델링, 정형 분석, 요구사항 정의 문서화
③ 명세	요구사항 체계적 분석 후 승인가능하도록 문서화
④ 확인/검증	요구사항 명세서가 정확하고 완전하게 작성되었는지 검토

▶ 요구사항 분석 도구

요구사항 분석 CASE (Computer Aided SW Engineering)	. SADT: SoftTech사에서 개발 / 구조적 분석 및 설계분석 . SREM: 실시간 처리 SW 시스템에서 요구사항 명확한 기술 목적 . PSL/PSA: 문제 기술언어 및 요구사항 분석 보고서 출력 . TAGS: 시스템 공학 방법 응용에 대한 자동 접근 방법
HIPO (Hierarchy Input Process Output)	하향식 설계 방식 / 가시적, 총체적, 세부적 다이어그램으로 구성 기능과 자료의 의존 관계 동시 표현 / 이해 쉽고 유지보수 간단

▶ 구조적 분석 모델

- **데이터/자료 흐름도 (DFD, Data flow diagram)**:

프로세스 (Process) / 자료 흐름 (Flow) / 자료 저장소 (Data store) / 단말 (Terminator)
원 화살표 평행선 사각형

→ 구조적 분석 기법에 이용 / 시간 흐름 명확한 표현 불가 / 버블(bubble) 차트

- **자료 사전 (DD, Data Dictionary)**: 자료 흐름도에 기재된 모든 자료의 상세 정의/설명

=	정의	[]	택일 / 선택	()	생략
+	구성	**	설명 / 주석	{ }	반복

- 소단위 명세서 / 개체 관계도 (ERD, Entity Relationship Diagram) / 상태 전이도



※ 객체지향 분석 모델

- . Booch (부치) : 미시적, 거시적 개발 프로세스를 모두 사용 (클래스/객체 분석 및 식별)
- . Jacobson (제이콥슨) : Use case를 사용 (사용자, 외부 시스템이 시스템과 상호작용)
- . Coad-Yourdon : E-R 다이어그램 사용 / 객체의 행위 모델링
- . Wirfs-Brock : 분석과 설계 구분 없으며 고객 명세서 평가 후 설계 작업까지 연속 수행
- . Rumbaugh (럼바우) : 가장 일반적으로 사용, 객체/동적/기능 모델로 구분
 - 객체 모델링 (Object) → 객체 다이어그램 / 객체들 간의 관계 규정/정의
 - 동적 모델링 (Dynamic) → 상태 다이어그램 / 시스템 동적인 행위 기술
 - 기능 모델링 (Function) → 자료 흐름도(DFD) / 다수의 프로세스들 간의 처리 과정 표현

▶ 요구사항 명세

정형 명세	수학적 원리 / 정확하고 간결한 요구사항 표현 가능 어려운 표기법으로 사용자 이해 어려움 (VDM, Z, Petri-net, CSP)
비정형 명세	자연어, 그림 중심 / 쉬운 자연어 사용으로 의사소통 용이하나 작성자에 따라 모호한 내용으로 일관성 떨어짐 (FSM, Decision Table, E-R 모델, State Chart)

③ 소프트웨어 설계

■ 소프트웨어 설계 원리

- 분할과 정복** : 여러 개의 작은 서브시스템으로 나눠서 각각을 완성
- 모듈화 (Modularity)** : 시스템 기능을 모듈 단위로 분류하여 성능/재사용성 향상
 - 모듈 크기 ▲ → 모듈 개수 ▼ → 모듈간 통합비용 ▼ (but, 모듈 당 개발 비용 ▲)
 - 모듈 크기 ▼ → 모듈 개수 ▲ → 모듈간 통합비용 ▲
- 추상화 (Abstraction)** : 불필요한 부분은 생각하고 필요한 부분만 강조해 모델화
 - 문제의 포괄적인 개념을 설계 후 차례로 세분화하여 구체화 진행
 - ① **과정 추상화** : 자세한 수행 과정 정의 X, 전반적인 흐름만 파악가능하게 설계
 - ② **데이터(자료) 추상화** : 데이터의 세부적 속성/용도 정의 X, 데이터 구조를 표현
 - ③ **제어 추상화** : 이벤트 발생의 정확한 절차/방법 정의 X, 대표 가능한 표현으로 대체
- 단계적 분해 (Stepwise refinement)** : 하향식 설계 전략 (by Niklaus Wirth)
 - 추상화의 반복에 의한 세분화 / 세부 내역은 가능한 뒤로 미루어 진행
- 정보 은닉 (Information Hiding)**
 - 한 모듈 내부에 포함된 절차/자료 관련 정보가 숨겨져 다른 모듈의 접근/변경 불가
 - 모듈을 독립적으로 수행할 수 있어 요구사항에 따라 수정/시험/유지보수가 용이함

■ 아키텍처 패턴

Layer	시스템을 계층으로 구분/구성하는 고전적 방식 (OSI 참조 모델)
Client-server	하나의 서버 컴포넌트와 다수의 클라이언트 컴포넌트로 구성 클라이언트와 서버는 요청/응답 제의 시 서로 독립적 * 컴포넌트(Component) : 독립적 업무/기능 수행 위한 실행코드 기반 모듈
Pipe-Filter	데이터 스트림 절차의 각 단계를 필터 컴포넌트로 캡슐화 후 데이터 전송 / 재사용 및 확장 용이 / 필터 컴포넌트 재배치 가능 단방향으로 흐르며, 필터 이동 시 오버헤드 발생 변환, 버퍼링, 동기화 적용 (ex. UNIX 셸 - Shell)
Model-view Controller	모델 (Model) : 서브시스템의 핵심 기능 및 데이터 보관 뷰 (View) : 사용자에게 정보 표시 컨트롤러 (Controller) : 사용자로부터 받은 입력 처리 → 각 부분은 개별 컴포넌트로 분리되어 서로 영향 X → 하나의 모델 대상 다수 뷰 생성 ▶ 대화형 애플리케이션에 적합
Master-slave	마스터에서 슬레이브 컴포넌트로 작업 분할/분리/배포 후 슬레이브에서 처리된 결과물을 다시 돌려 받음 (병렬 컴퓨팅)
Broker	컴포넌트와 사용자를 연결 (분산 환경 시스템)
Peer-to-peer	피어를 한 컴포넌트로 산정 후 각 피어는 클라이언트가 될 수도, 서버가 될 수도 있음 (펄티스레드 방식)
Event-bus	소스가 특정 채널에 이벤트 메시지를 발행 시 해당 채널을 구독한 리스너들이 메시지를 받아 이벤트를 처리함
Blackboard	컴포넌트들이 검색을 통해 블랙보드에서 원하는 데이터 찾을
Interpreter	특정 언어로 작성된 프로그램 코드를 해석하는 컴포넌트 설계

■ UML (Unified Modeling Language) → 구성요소 : 사물, 관계, 다이어그램

- 고객/개발자 간 원활한 의사소통을 위해 표준화된 대표적 객체지향 모델링 언어
- Rumbaugh, Booch, Jacobson 등 객체지향 방법론의 장점 통합

※ 인터페이스 : 클래스/컴포넌트가 구현해야하는 오퍼레이션 세트를 정의하는 모델 요소

- 사물 (Things)** : 구조(개념, 물리적 요소) / 행동 / 그룹 / 주해(부가적 설명, 제약조건)
- 관계 (Relationship)**

연관 관계 (Association)	2개 이상의 사물이 서로 관련
집합 관계 (Aggregation)	하나의 사물이 다른 사물에 포함 (전체-부분 관계)
포함 관계 (Composition)	집합 관계 내 한 사물의 변화가 다른 사물에게 영향
일반화 관계 (Generalization)	한 사물이 다른 사물에 비해 일반/구체적인지 표현 (한 클래스가 다른 클래스를 포함하는 상위 개념일 때)
의존 관계 (Dependency)	사물 간 서로에게 영향을 주는 관계 (한 클래스가 다른 클래스의 기능을 사용할 때)
실체화 관계 (Realization)	한 객체가 다른 객체에게 오퍼레이션을 수행하도록 지정 / 서로를 그룹화할 수 있는 관계

3) 다이어그램 (Diagram)

구조, 정적 다이어그램 (클래스/컴포넌트)	클래스 (Class)	클래스 사이의 관계 및 속성 표현
	객체 (Object)	인스턴스를 객체와 객체 사이의 관계로 표현
	컴포넌트 (Component)	구현 모델인 컴포넌트 간의 관계 표현
	배치 (Deployment)	물리적 요소(HW/SW)의 위치/구조 표현
	복합체 구조 (Composite Structure)	클래스 및 컴포넌트의 복합체 내부 구조 표현
행위, 동적 다이어그램 (유시커상활타상)	패키지 (Package)	UML의 다양한 모델요소를 그룹화하여 묶음
	유스케이스 (Use case)	사용자의 요구를 분석 (사용자 관점) → 사용자(Actor) + 사용 사례 (Use Case)
	시퀀스 (Sequence)	시스템/객체들이 주고받는 메시지 표현 → 구성항목: 액터* / 객체 / 생명선 / 메시지 제어 삼각형
	커뮤니케이션 (Communication)	객체들이 주고받는 메시지와 객체 간의 연관관계까지 표현
	상태 (State)	다른 객체와의 상호작용에 따라 상태가 어떻게 변화하는지 표현
	활동 (Activity)	객체의 처리 로직 및 조건에 따른 처리의 흐름을 순서로 따라 표현
	타이밍 (Timing)	객체 상태 변화와 시간 제약 명시적으로 표현
	상호작용 개요 (Interaction Overview)	상호작용 다이어그램 간 제어 흐름 표현

▶ UI 설계 (User Interface) : 직관성, 유효성(사용자의 목적 달성), 학습성, 유연성

- 설계 지침 : 사용자 중심 / 일관성 / 단순성 / 결과 예측 / 가시성 / 표준화 / 접근성 / 명확성 / 오류 발생 해결

- CLI (Command Line), GUI (Graphical), NUI (Natural), VUI (Voice), OUI (Organic)

텍스트 그래픽 말/행동 음성 사물과 사용자 상호작용

- UI 설계 도구

Wireframe	기획 초기 단계에 대략적인 레이아웃을 설계
Story Board	최종적인 산출문서 (와이어프레임-UI, 콘텐츠 구성, 프로세스 등)
Prototype	와이어프레임 / 스토리보드에 인터랙션 적용 실제 구현된 것처럼 테스트가 가능한 동적인 형태 모형 *인터랙션 : UI를 통해 시스템을 사용하는 일련의 상호작용 (동적효과)
Mockup	실제 화면과 유사한 정적인 형태 모형
Use case	사용자 측면 요구사항 및 목표를 다이어그램으로 표현



④ 소프트웨어 구현

■ 소프트웨어 개발 프레임워크

- 개발해야 할 애플리케이션 일부분이 이미 내장된 클래스 라이브러리에 구현
- 동일 로직 반복 최소화 / 재사용성 확대 / 생산성 및 유지보수성 향상
- 모듈화 / 재사용성 / 확장성 / 제어의 역흐름 (프레임워크가 어플리케이션 흐름 제어)

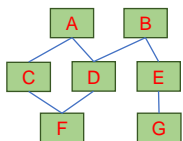
스프링 프레임워크 (Spring Framework)	. JAVA 플랫폼을 위한 오픈 소스 경량형 프레임워크 . 동적 웹 사이트 개발 / 전자정부 표준 프레임워크
전자정부 프레임워크	. 우리나라 공공부문 정보화 사업 시 효율적인 정보 시스템 구축 지원 위해 필요한 기능/아키텍처를 제공
닷넷 프레임워크 (.NET Framework)	. MS에서 개발한 Windows 프로그램 개발 . 공통 언어 런타임(CLR)이라는 가상 머신 상에서 작동

※ API (Application Programming Interface)

- 소프트웨어 간 인터페이스 (서로 다른 소프트웨어/서비스 간 상호 작용 용이)
- 운영체제 및 프로그래밍 언어의 기능을 프로그램에서 사용 가능하도록 구현
- 개발 비용 저감 / 중복 작업 최소화 / 유지 관리 용이 / 비즈니스 확장

▶ 팬인 (Fan-in): 자신을 사용하는 모듈의 수 → 팬인이 높으면 재사용 측면 설계 우수
but, 단일 장애점 발생 가능성 높아 집중적인 관리/테스트 필요

▶ 팬아웃 (Fan-out): 자신이 사용하는 모듈의 수 → 불필요한 호출 가능성 (단순화 필요)



Q. D의 팬인의 개수는? 2개 (A, B)

Q. D의 팬아웃의 개수는? 1개 (F)

■ 응집도 (Cohesion)

- 개발 모듈이 독립적인 기능으로 정의되어 있는 정도 / 응집도 ▲ → 품질 ▲

높은 응집도

기능적 (Function)	모듈 내부의 모든 기능 요소가 단일 문제와 연관되어 수행
순차적 (Sequential)	모듈 내 출력 데이터를 다음 활동의 입력 데이터로 사용
통신적 (Communication)	동일한 입/출력을 사용하여 서로 다른 기능을 수행
절차적 (Procedural)	모듈 내 구성 요소들이 다수 관련 기능을 순차적으로 수행
시간적 (Temporal)	특정 시간 내 처리되는 기능을 모아 하나의 모듈로 작성
논리적 (Logical)	유사한 성격의 처리 요소들로 하나의 모듈이 형성
우연적 (Coincidental)	각 구성 요소들이 서로 관련없는 요소로만 구성

낮은 응집도

■ 결합도 (Coupling) - 개별 모듈 간 상호 의존하는 정도 / 결합도 ▼ → 품질 ▲

높은 결합도

내용 (Content)	한 모듈이 다른 모듈의 내부 기능 및 자료를 직접 참조/수정
공통/공유 (Common)	공유되는 공통 데이터를 여러 모듈이 사용 (전역 변수 참조)
외부 (External)	한 모듈에서 선언한 데이터를 외부의 다른 모듈에서 참조
제어 (Control)	. 한 모듈이 다른 모듈의 상세한 처리 절차를 알고 있어 이를 통제하는 경우나 처리 기능이 두 모듈에 분리되어 설계 . 처리 대상 값뿐만 아니라 처리 방식의 제어 요소도 전달
스탬프 (Stamp)	. 두 모듈이 동일한 자료 구조(배열, 오브젝트)를 조회 . 자료 구조 및 포맷 변화 시 조회하고 있는 모든 모듈에 영향
자료 (Data)	모듈 간의 인터페이스가 자료 요소로만 구성

낮은 결합도

- ※ 모듈의 독립성: 모듈 간 과도한 상호작용을 막고 하나의 독립적 기능만을 수행
- 높은 독립성: 높은 응집도 / 낮은 결합도

■ 객체 지향 (Object-oriented): 객체와 속성, 클래스와 멤버, 전체와 부분으로 나뉘 분석

객체 (Object)	고유 식별자 / 하나의 독립된 존재 / 일정한 기억장소 보유 상태(state) = 객체가 가질 수 있는 조건, 속성 값에 의해 정의 행위(연산, Method) = 객체가 반응할 수 있는 메시지 집합
클래스 (Class)	공통 속성과 연산(행위)을 갖는 객체들의 집합 / 데이터 추상화 단위 ※ 인스턴스 (Instance): 클래스에 속한 각각의 객체 ※ Operation: 클래스의 동작 / 객체에 대해 적용될 메서드 정의
캡슐화 (Encapsulation)	데이터와 데이터 처리 함수를 하나로 묶음 세부 내용 은폐(정보 은닉) → 외부 접근 제한 결합도 낮음 / 재사용 용이 / 인터페이스 단순 / 오류 파급효과 낮음
상속 (Inheritance)	상위 클래스의 속성과 연산을 하위 클래스가 물려받는 것 ※ 다중 상속: 단일 클래스가 두 개 이상의 상위 클래스로부터 상속
다형성 (Polymorphism)	하나의 메시지에 각 객체 별 고유 특성에 따라 여러 형태의 응답

오버라이딩 (Overriding): 상위클래스로부터 상속받은 메서드를 하위클래스에서 재정의
→ 단, 메서드 이름 / 매개변수 / 반환 타입은 동일해야 함

오버로딩 (Overloading): 메서드 이름은 동일하나 매개변수 개수 또는 타입을 다르게 지정

※ 객체지향 설계 5대 원칙 (SOLID)

- . 단일 책임 원칙 (SRP, Single Responsibility Principle)
→ 모든 클래스/객체는 하나의 책임만 / 완전한 캡슐화
- . 개방 폐쇄의 원칙 (OCP, Open Closed Principle)
→ 확장에는 Open하고, 수정에는 Close되어야 한다.
- . 리스코프 교체 원칙 (LSP, Liskov Substitution Principle)
→ 상위 클래스의 행동 규약을 하위 클래스가 위반하면 안된다.
- . 인터페이스 분리 원칙 (ISP, Interface Segregation Principle)
→ 클라이언트가 비사용 메서드에 의존하지 않아야 한다.
- . 의존성 역전 원칙 (DIP, Dependency Inversion Principle)
→ 의존 관계 수립 시 변화하기 어려운 것에 의존해야 한다.
추상성이 높은 상위 클래스

■ 디자인 패턴: GoF (Gang of Four) 처음 제안하여 구체화

- 서브시스템에 속하는 컴포넌트들과 그 관계를 설계하기 위한 참조 모델
- 객체 지향 프로그래밍 설계 시 자주 발생하는 문제에 대한 반복적 해결 방법
- cf) 아키텍처 패턴: 전체 시스템의 구조를 설계

[생성 패턴]

Abstract Factory	구체적인 클래스에 의존하지 않고, 서로 연관되거나 의존적인 객체들이 조합된 인터페이스 제공
Builder	객체 생성 단계를 캡슐화/분리하여 객체를 조립하여 생성 → 동일한 객체 생성 절차에서 서로 다른 표현 결과를 제공
Factory Method	상위클래스에서 객체 생성 인터페이스를 정의하지만, 인스턴스를 만드는 클래스는 서브 클래스에서 결정하도록 분리
Prototype	원본/원형 객체를 복제하는 방식으로 객체를 생성
Singleton	클래스에서 하나의 객체만 생성 가능하며, 해당 객체를 어디서든 참조할 수 있지만 여러 프로세스가 동시에 참조는 불가

[구조 패턴]

Adaptor	비호환 인터페이스에 호환성 부여하도록 변환
Bridge	구현부에서 추상층을 분리 후 각자 독립적으로 변형/확장 가능
Composite	트리 구조로 부분/전체 계층 표현, 복합/단일 객체를 구분없이 사용
Decorator	상속 사용없이 객체 간 결합을 통해 객체 기능을 동적으로 추가/확장
Façade	상위에 인터페이스 구성하여 서브클래스의 기능을 복잡하게 표현하지 않고 단순한 인터페이스로 구현
Flyweight	인스턴스를 공유하여 메모리 절약 (클래스 경량화)
Proxy	접근이 힘든 객체를 연결하는 인터페이스 역할 (대리 객체 수행)



[행위 패턴]

Chain of Responsibility	처리가능한 객체가 둘 이상 존재하여 한 객체 내 처리 불가 시 다음 객체로 이관
Command	요청 명령어들을 추상/구체 클래스로 분리 후 단순화/캡슐화
Interpreter	언어에 문법 표현 정의
Iterator	접근이 빈번한 객체에 대해 동일 인터페이스 사용
Mediator	객체들간 복잡한 상호작용을 캡슐화하여 객체로 정의 후 중재
Memento	객체를 이전의 특정 시점의 상태로 저장하고 복원 (캡슐화 유지)
Observer	한 객체 상태 변화 시 상속되어 있는 객체들에 변화 전달
State	객체의 상태에 따라 동일한 동작을 다르게 처리
Strategy	동일 계열 알고리즘을 개별적으로 캡슐화하여 상호 교환
Template Method	여러 클래스에서 공통 사용 메서드를 상위 클래스에서 정의하고, 하위 클래스마다 다르게 구현해야하는 세부 사항을 개별 구현
Visitor	각 클래스 데이터 구조로부터 처리/연산 기능을 분리하여 별도의 클래스를 만들고, 해당 클래스 메서드가 각 클래스를 돌아다니며 특정 작업을 수행 → 객체 구조 변경 X / 새로운 연산 기능만 추가

⑤ 소프트웨어 테스트

■ 애플리케이션 테스트 기본 원리

- 테스트는 기본적으로 결함이 존재함을 밝히는 것 (무결함을 증명할 수는 없음)
→ 완벽한 테스트는 근원적으로 불가능 (무한 경로, 무한 입력 불가)
- 결함 집중 : **파레토(Pareto) 법칙** - 20%의 모듈에서 전체 결함 80% 발생
- **살충제 패러독스** : 동일한 테스트 케이스에 의한 반복 테스트는 새로운 버그 발견 X
- 오류-부재의 궤변 : 결함이 없다 해도 사용자의 요구사항 미충족 시 품질 저하
- Brooks의 법칙 : 지연되는 프로젝트에 인력 추가 투입 시 더 지연

■ 애플리케이션 테스트 분류

① 프로그램 실행 여부

정적 테스트	프로그램 실행 X / 명세서, 소스 코드만 분석 Ex. 동료 검토, 워크 스루, 인스펙션, 코드 검사
동적 테스트	프로그램 실행 후 오류 검사 ex. 화이트/블랙박스 테스트

② 테스트 기반 테스트

명세 기반	사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 구현하는지 확인 → ex. 동등 분할 / 경계값 분석 (블랙박스)
구조 기반	SW 내부 논리 흐름에 따라 테스트 케이스 작성/확인 ex. 구문 기반 / 결정 기반 / 조건 기반 (화이트박스)
경험 기반	테스터의 경험을 기반으로 수행 ex. 에러 추정, 체크리스트, 탐색적 테스트

③ 목적 기반 테스트

회복 (Recovery)	시스템에 인위적 결함 부여 후 정상으로 회복되는 과정 확인
안전 (Security)	외부 불법 침입으로부터 시스템을 보호할 수 있는지 확인
강도 (Stress)	과부하시 SW 정상 구동 여부 확인
성능 (Performance)	실시간 성능 및 전체적인 효율성 진단 (응답 시간, 업무 처리량)
구조 (Structure)	SW 내부 논리적 경로 및 소스 코드 복잡도 평가
회귀 (Regression)	SW 내 변경 또는 수정된 코드에 새로운 결함이 없음을 확인
병행 (Parallel)	변경 및 기존 SW에 동일한 데이터 입력 후 결과 비교

④ 시각(관점) 기반 테스트

검증 (Verification)	개발자의 시각에서 제품의 생산 과정 테스트 Ex. 단위/통합/시스템 테스트
확인 (Validation)	사용자의 시각에서 생산된 제품의 결과 테스트 Ex. 인수 테스트 (알파 / 베타)

■ 화이트박스 테스트 (White Box Test)

- 모듈 안의 내용(작동) 직접 볼 수 있으며, 내부의 **논리적인** 모든 경로를 테스트
- 소스 코드의 모든 문장을 한 번 이상 수행 / **논리적 경로 점검** (선택, 반복 수행)
- 테스트 데이터 선택하기 위해 **검증 기준 커버리지(Coverage)** 정함

[화이트박스 테스트 검증 기준]

구문 커버리지 (Statement Coverage)	프로그램 내 모든 명령문 을 적어도 한 번 수행
결정(분기) 커버리지 (Branch Coverage)	프로그램 내 전체 결정문 이 적어도 한 번은 참/거짓 결과 수행
조건 커버리지 (Condition Coverage)	결정 명령문 내의 각 개별 조건식 이 적어도 한 번은 참/거짓 결과 수행
조건/결정 커버리지 (Condition/Branch)	전체 조건식뿐만 아니라 개별 조건식도 참 한 번 이상, 거짓 한 번 이상 결과 수행

[화이트박스 테스트 종류]

기초 경로 검사 (Base Path Testing)	. 대표적 화이트박스 테스트 기법 (동적 테스트) . 테스트 케이스 설계자가 절차적 설계의 논리적 복잡성을 측정할 수 있게 해주는 테스트 기법 . 측정 결과는 실행 경로의 기초를 정의하는 지침으로 사용
제어 구조 검사 (Control Structure Testing)	. 조건 검사 : 프로그램 모듈 내 논리적 조건 테스트 . 루프 검사 : 프로그램 반복(Loop) 구조 테스트 . 자료 흐름 검사 : 변수의 정의와 변수 사용의 위치 테스트

■ 블랙박스 테스트 (Black Box Test)

- 모듈 내부의 내용 알 수 없음 / 소프트웨어 인터페이스에서 실시되는 테스트
- SW 각 기능이 완전히 작동되는 것을 입증하는 테스트로 '**기능 테스트**' 라고 함

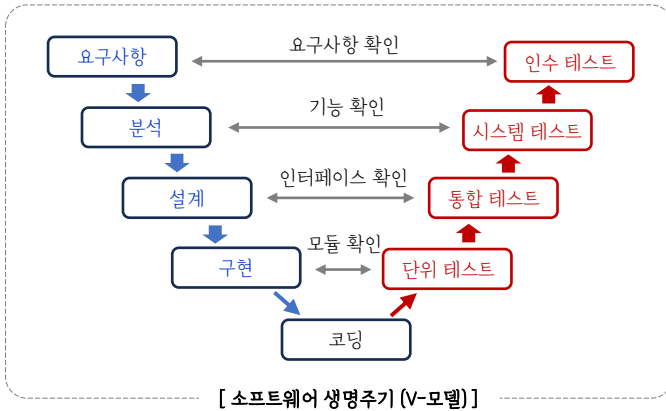
[블랙박스 테스트 종류]

동치 분할 검사 (Equivalence Partition)	. 프로그램 입력 조건에 타당한 입력 자료와 타당하지 않은 입력 자료의 개수를 균등하게 진행
경계값 분석 (Boundary Value)	. 입력 조건의 경계값을 테스트 케이스로 선정 (ex. 범위 구간의 양끝 : ~이상/이하/초과/미만)
원인-효과 그래프 검사 (Cause-Effect Graphing)	. 입력 데이터 간의 관계와 출력에 영향을 미치는 상황을 체계적으로 분석 후 효용성이 높은 테스트 케이스 선정
오류 예측 검사 (Error Guessing)	. 과거의 경험이나 확인자의 감각으로 테스트 진행
비교 검사 (Comparison)	. 여러 버전의 프로그램에 동일한 결과가 출력되는 확인

■ 요구사항 검증 (Requirements Validation)

- 실제로 고객이 원하는 시스템을 제대로 정의했는지 점검하는 과정
- 시스템 개발 완료 후 문제 발생 시 막대한 재작업 비용 발생하기에 검증 중요 (실제 요구사항 반영 여부 / 문서 상 요구사항이 서로 상충되지 않는지 점검)
 - ① 동료 검토 (Peer review) : 작성자가 내용 설명 후 동료들이 결함 검토
 - ② 워크 스루 (Walk through) : 요구사항 명세서 미리 배포 후 짧은 검토 회의 진행
 - ③ 인스펙션 (Inspection) : 작성자 제외한 다른 전문가들이 결함 검토

※ 검증 체크리스트 : 기능성 (Functionality) / 완전성 (Completeness) / 일관성 (Consistency) / 명확성 (Unambiguity) / 검증 가능성 (Verifiability) / 추적 가능성 (Traceability) / 변경 용이성 (Easily Changeable)



■ 개발단계에 따른 애플리케이션 테스트

→ 소프트웨어를 이루는 기본 단위 (독립적 기능)

① 단위 테스트 (Unit test): 최소 단위(모듈/컴포넌트) 기반 테스트

주로 구조 기반 테스트 진행 / 기능성 테스트 최우선

② 통합 테스트 (Integration test): 인터페이스 간 시스템이 정상 실행되는지 확인

- 단위 테스트 후 모듈을 통합하는 과정에서 발생하는 오류 및 결함을 찾는 테스트 기법

하향식 (Top-down)	. 상위 모듈에서 하위 모듈 방향으로 통합 . 깊이 우선(Depth first) 통합법 / 넓이 우선(Breadth first) 통합법 . 초기부터 사용자에게 시스템 구조 보여줌 . 스텝(Stub): 모듈의 기능을 단순히 수행하는 도구 (시험용 모듈)
상향식 (Bottom-up)	. 하위 모듈에서 상위 모듈 방향으로 통합 . 하나의 주요 제어 모듈과 관련된 종속 모듈의 그룹인 클러스터(Cluster)와 드라이버(Driver) 사용 / 스텝(Stub) 미사용

③ 시스템 테스트 (System test): 개발된 SW의 컴퓨터 시스템 내 작동여부 점검

실제 사용 환경과 유사한 테스트 환경 ▶ 기능적 및 비기능적 테스트 구분
블랙박스 화이트박스

④ 인수 테스트 (Acceptance test): 사용자의 요구사항 충족 여부 확인

- 알파 테스트: 통제된 환경에서 사용자가 개발자와 함께 확인

- 베타 테스트: 통제되지 않은 환경에서 개발자 없이 여러 명의 사용자가 검증

■ 테스트 관련 기타 용어

테스트 시나리오 (Test Scenario)	. 테스트 케이스 적용/동작 순서에 따라 여러 테스트 케이스를 묶은 집합
테스트 오라클 (Test Oracle)	. 테스트 결과의 참/거짓 판단 위해 사전에 정의된 참 값을 대입

. 참 오라클 (True): 모든 테스트 케이스의 입력값에 기대 결과 제공
. 샘플링 오라클 (Sampling): 특정 테스트 케이스 입력값에 기대 결과 제공
. 추정 오라클 (Heuristic): 특정 테스트 케이스 입력값에 기대 결과 제공
+ 나머지 입력값에 대해선 추정 결과 제공
. 일관성 오라클 (Consistent): 테스트 케이스의 수행 전/후의 결과값 동일 여부 확인

■ 테스트 하네스 (Test Harness)

테스트 드라이버 (Test Driver)	시험 대상의 하위 모듈 호출 / 모듈 테스트 수행 후의 결과 도출 → 상향식 통합 테스트에서 사용
테스트 스텝 (Test Stub)	제어 모듈이 호출하는 하위 모듈의 역할 단순 수행 → 하향식 테스트에 사용
테스트 스위트 (Test Suites)	시스템에 사용되는 테스트 케이스의 집합 (컴포넌트 / 모듈)
테스트 케이스 (Test Case)	사용자의 요구사항 준수 여부 확인 위해 설계된 테스트 항목 명세서 (입력값, 실행 조건, 기대 결과 등)
테스트 스크립트 (Test Script)	자동화된 테스트 실행 절차에 대한 명세서
목 오브젝트 (Mock Object)	사용자의 행위 조건부 입력 시 계획된 행위를 수행하는 객체

⑥ 소프트웨어 유지 보수

■ 애플리케이션 성능 개선

- 성능 분석 지표: 처리량 / 응답 시간 / 경과 시간 / 자원 사용률
- 소스코드 최적화를 통한 코드 스멜 제거 후 품질 개선

※ 코드 스멜 (Code Smell): 소스코드 내 존재하는 잠재적 문제점

예시) 중복 코드 / 긴 메서드 / 큰 클래스 / 클래스 동시 수정

→ 스파게티 코드: 소스코드 로직이 복잡하게 얽혀있는 구조

→ 외계인 코드: 오래되어 유지보수 작업이 어려운 코드

▶ 클린 코드 작성 원칙: 가독성 / 단순성 / 의존성 배제 / 중복성 최소화 / 추상화

▶ 소스 코드 품질 분석 도구

정적 분석 도구 (Static Analysis)	프로그램 실행 없이 코딩 표준/스타일/결함 등을 분석 → PMD, Checkstyle, SonarQube, Cppcheck, Ccm, Cobertura
동적 분석 도구 (Dynamic Analysis)	프로그램 실행하여 코드 내 메모리 누수 및 스레드 결함 발견 → Avalanche, Valgrind, Valance

■ 소프트웨어 형상 관리 (SCM: Software Configuration Management)

- 개발 과정에서 SW 변경사항을 관리하기 위한 일련의 활동 / 개발 전체 단계 적용

- 중요성: 변경사항 추적/통제, 무절제한 변경 방지, 개발 진행 이력 확인

- 형상 관리 역할: 배포본 관리 용이 / 불필요한 소스 수정 제한 / 여러 개발자 동시 개발

형상 식별	관리 대상에 이름/번호 부여 후 계층 구조로 구분 → 수정/추적 용이
형상 통제	식별된 형상 항목에 대한 변경 요구 검토 (기준선 반영될 수 있게)
형상 감사	기준선(Base line)의 무결성 평가 위해 확인/검증/검열 과정 진행
형상 기록	형상 식별/통제/감사 작업 결과를 기록/관리하고 보고서 작성

※ 제품 SW의 형상 관리 역할: 배포본 관리 용이 / 소스 수정 제한

동일 프로젝트에 대해 여러 개발자 참여 후 동시 개발 가능

■ 소프트웨어 버전 관리 도구

공유 폴더	클라이언트/서버	분산 저장소
공유 폴더에 복사	서버에서 버전 일괄 관리	원격 → 지역 저장소
RCS, SCCS, PVCS, QVCS	CVS, SVN, CVSNT, CMBC	Git, GNU arch, DCVS, Bitkeeper, Bazaar

※ 형상관리 도구

CVS	서버/클라이언트 구성, 다수의 인원이 동시 버전 관리 가능
SVN (Subversion)	CVS 개선 톨 / 모든 개발은 trunk 디렉터리에서 수행 Commit 수행 시 revision 1씩 증가
Git	서버(원격) 저장소와 개발자(지역) 저장소가 독립적 → Commit 실수 발생해도 서버에 영향 없음 (분산된 P2P 모델)

■ ISO 12207 - 소프트웨어 관련 생명주기

기본 생명주기	획득, 공급, 개발, 운영, 유지보수 프로세스
지원 생명주기	문서화, 형상관리, 품질보증, 검증, 확인, 합동검토, 감사
조직 생명주기	관리, 기반구조, 개선, 교육훈련

■ CMMI(Capability Maturity Model Integration): SW 개발 조직의 업무 능력 평가 모델

1) 초기	프로세스 X	작업자 능력에 따라 성공 여부 결정
2) 관리	규칙화 프로세스	특정한 프로젝트 내의 프로젝트 정의/수행
3) 정의	표준화 프로세스	조직의 표준 프로세스 활용 업무 수행
4) 정량적 관리	예측 가능 프로세스	프로젝트 정량적 관리/통제
5) 최적화	지속 개선 프로세스	프로세스 역량 향상 위한 지속적인 개선