꿈꾸는 라이언

정보처리기사 필기

핵심 요약 노트

꿈꾸는 라이언

목 차

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

폭포수	선형 순차적 개발 / 고전적, 전통적 개발 모형 / Step-by-Step
HIPO (Hierarchy Input Process Output)	하향식 설계 방식 / 가시적, 총체적, 세부적 다이어그램으로 구성 기능과 자료의 의존 관계 동시 표현 / 이해 쉽고 유지보수 간단
프로토타입	고객의 need 파악 위해 견본/시제품을 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 중시) 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 (Bottum-up) : 객체 지향 / 최하위 모듈 먼저 설계 후 이들을 결합하고 검사
→ 인터페이스 구조 구조 변경 시 상위 모듈도 같이 변경 필요하여 기능 추가 어려움

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

■ 스크럼 (Scrum) 기법

제품 책임자 (Product Owner / PO)	. 요구사항이 담긴 백로그(Back log)를 작성 및 우선순위 지정 . 요구사항을 책임지고 의사결정
스크럼 마스터 (Scrum Master / SM)	. 원활한 스크럼 위해 객관적 시각으로 조언 및 가이드 제시 . 진행사항 점검 및 장애요소 논의 후 해결
개발팀 (Development Team / DT)	. PO와 SM을 제외한 모든 팀원 . 백로그에 스토리 추가 가능하나 우선순위는 지정 불가

※ 스크럼 개발 프로세스

스프린트 계획 회의 → 스프린트 → 일일 스크럼 회의 → 스프린트 검토 회의 → 스프린트 회고
개발 기간 2~4주 단기간 목표

■ 익스트림 프로그래밍 (eXtreme Programming, XP)

- 고객의 요구사항을 유연하게 대응하기 위해 고객 참여와 신속한 개발 과정을 반복
- 핵심 가치 : 용기 / 단순성 / 의사소통 / 피드백 / 존중

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

- 기본 원리 : 전체 팀 / 소규모 릴리즈 / 테스트 주도 개발 / 계속적인 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)

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

▶ 요구사항의 유형

기능적 요구	실제 시스템 수행에 필요한 요구사항 *ex. 금융 시스템은 조회/인출/입금/송금 기능이 있어야 한다.*
비기능적 요구	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 기법 : 분류 / 개념 모델링 / 할당 / 협상 / 정형 분석

▶ 요구사항 확인 기법

요구사항 검토	문서화된 요구사항을 확인 (일반적 방법)
프로토타이핑	요구사항이 반영된 프로토타입 지속 제작 (피드백 후 반복 제작)
모델 검증	요구사항 분석 단계에서 개발된 모델이 충족되는지 검증
인수 테스트	사용자 측면에서 실제 사용 환경 내 요구사항이 충족되는지 검증

구조 / 행동 / 그룹 / 주해

■ UML (Unified Modeling Language) 구성요소 : 사물 / 관계 / 다이어그램
→ 고객/개발자 간 원활한 의사소통을 위해 표준화한 대표적 객체지향 모델링 언어

연관 관계 (Association)	2개 이상의 사물이 서로 관련
집합 관계	하나의 사물이 다른 사물에 포함
포함 관계	집합 관계의 특수 형태
일반화 관계 (Generalization)	한 사물이 다른 사물에 비해 일반/구체적인지 표현
의존 관계 (Dependency)	사물 간 서로에게 영향을 주는 관계
실체화 관계 (Realization)	한 객체가 다른 객체에게 오퍼레이션을 수행하도록 지정 / 서로를 그룹화할 수 있는 관계

구조, 정적 다이어그램 (클래컴배복패)	클래스 (Class) / 객체 (Object) / 컴포넌트 (Component) / 배치 (Deployment) / 복합체 (Composite) / 패키지 (Package)
행위, 동적 다이어그램 (유시커상활타상)	. **유스케이스 (Use case)** : 사용자의 요구를 분석 . **순차 (Sequence)** : 시스템/객체들이 주고받는 메시지 표현 → 구성 항목: 액터* / 객체 / 생명선 / 메시지 / 제어 삼각형 . 커뮤니케이션 (Communication) : 객체들 간의 연관까지 표현 . 상태 (State) : 상태가 어떻게 변화하는지 표현 . 활동 (Activity) : 처리의 흐름을 순서에 따라 표현 . 타이밍 : 객체 상태 변화와 시간 제약을 명시적으로 표현 . 상호작용 개요 : 상호작용 다이어그램 간 제어 흐름 표현

※ 액터 : 시스템과 상호작용하는 사람이나 다른 시스템에 의한 역할
- 사용자 액터 : 기능을 요구하는 대상이나 시스템의 수행결과를 통보받는 사용자
- 시스템 액터 : 본 시스템과 데이터를 주고 받는 연동 시스템

■ User Interface (UI) : 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 ***인터렉션** : UI를 통해 시스템을 사용하는 일련의 상호작용 (동적효과)
Mockup	실제 화면과 유사한 정적인 형태 모형
Use case	사용자 측면 요구사항 및 목표를 다이어그램으로 표현

- UI 프로토타입의 장/단점

장점	. 사용자를 설득하고 이해시키기 쉬움 / 사전에 오류 예방 가능 . 개발 시간 단축 (요구사항 및 기능의 불일치 방지)
단점	. 반복적 개선/보완 작업으로 작업 시간 증가, 자원 소모 . 부분적 프로토타이핑 시 중요 작업 누락 및 생략 가능

■ 소프트웨어 아키텍처

1. 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 ▲

2. 추상화 (Abstraction) : 불필요한 부분은 생략하고 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

3. 단계적 분해 (Stepwise refinement) : 하향식 설계 전략 (by Niklaus Wirth)
- 추상화의 반복에 의한 세분화 / 세부 내역은 가능한 뒤로 미루어 진행

4. 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

※ 소프트웨어 아키텍처 설계 과정
① 설계 목표 설정 → ② 시스템 타입 결정 → ③ 아키텍처 패턴 적용
→ ④ 서브시스템 구체화 → ⑤ 검토

※ 소프트웨어 아키텍처의 품질 속성
- 성능 / 보안 / 가용성 / 기능성 / 변경 용이성 / 확장성 / 배치성 / 안정성

■ 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 제외 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 쉘 – 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 (멀티스레딩 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 찾음
Interpreter	특정 언어로 작성된 프로그램 코드를 해석하는 컴포넌트 설계

EAI (Enterprise Application Integration) : 기업 응용 프로그램을 하나로 통합
FEP (Front-End Processor) : 입력 데이터를 프로세스가 처리하기 전에 미리 처리하여 프로세스 처리 시간을 줄여주는 프로그램

■ 객체 지향 (Object-oriented)
- 객체와 속성, 클래스와 멤버, 전체와 부분 등으로 나누어 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 수상화 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) : 상위 클래스의 메서드를 하위 클래스에서 동일하게 재정의
오버로딩 (Overloading) : 메서드 이름은 동일하나 매개변수 개수 또는 타입을 다르게 지정

※ 분석 방법론
. 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)
객체 모델링 (Object) → **객**체 다이어그램 / 객체 들간의 관계 규정
동적 모델링 (Dynamic) → **상**태 다이어그램 / 시스템 행위 기술
기능 모델링 (Function) → **자**료 흐름도 / 다수의 프로세스들 간의 처리 과정 표현

※ 데이터/자료 흐름도 (DFD, Date flow diagram) 구성요소 :
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

※ 5대 설계 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.

■ CASE (Computer-Aided Software Engineering)
- 소프트웨어 개발 시 요구분석/설계/구현/검사 과정을 자동화(표준화)해주는 작업
- 1980년대 소개된 이후 90년대부터 자주 사용
- 장점 : 비용 절감 / 모듈 재사용성 향상 / SW 품질 및 생산성 향상 / 유지보수 간편
- 지원 기능 : SW 생명 주기 전체 단계를 연결 / 자동화 통합 도구 제공 /
그래픽 기능 제공 (문서화, 명세화 도움) / 자료 흐름도 작성
모델의 오류 검증 / 모델 간 모순검사 / 원시코드 생성

■ 데이터베이스 관리 시스템 (DBMS)
- 사용자 요구에 따라 정보 생성하고, 데이터베이스를 관리해주는 소프트웨어
- 분석 시 고려사항 : 가용성 (무결성) / 성능 (효율성) / 상호 호환성 (일관성) /
기술지원 / 회복 / 보안 / 데이터베이스 확장 / 구축 비용

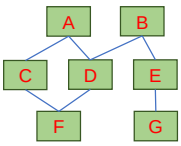
■ **요구사항 검증 (Requirements Validation)**

- 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)

▶ **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)**

■ **코드 (Code) 부여 체계**

순차/순서 코드 (Sequence)	최초의 자료부터 차례로 일련번호 부여 Ex) 1, 2, 3, 4, 5 …
블록 코드 (Block)	공통적인 블록으로 먼저 구분 후 각 블록 내 일련번호 부여 Ex) 101 ~ 150 : 중학생 / 201 ~ 250 : 고등학생
10진 코드 (Decimal)	0~9까지 10진 분할 후, 다시 추가 10진 분할 → 필요만큼 반복 1000: 음악 / 1100 : 국악 / 1110 : 거문고
그룹 분류 코드 (Group Classification)	대/중/소 분류 후 각 그룹 안에 일련번호 부여 Ex) 1-01-001 : 서울-초등-1학년
연상 코드 (mnemonic)	관계있는 숫자/문자/기호를 사용하여 코드 부여 Ex) S-03-13 : S 핸드폰 3월 13일 제조
표의 숫자 코드 (Significant digit)	물리적 수치를 그대로 코드에 적용 (길이, 넓이, 부피 등) Ex) 150-80-120 : 150x80x120 길이/폭/높이 서랍장
합성 코드 (Combined)	2개 이상의 코드를 조합 Ex) KE243: 대한항공 243기

■ **자료 사전 (Data Dictionary) 기호**

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

■ **미들웨어 (Middleware)**

- 운영체제와 애플리케이션 간에 중간 매개 역할 (사용자가 내부 동작 확인 필요 X)
- 클라이언트/서버 간 통신 담당, 시스템 간 표준화된 연결 지향 (데이터 교환 일관성)

DB (Database)	원격 데이터베이스와 연결 (2-Tier 아키텍처) → OCBC (MS) / IDAPI (볼랜드) / Glue (오라클)
RPC (Remote Procedure Call)	원격 프로시저를 로컬 프로시저처럼 호출 → Entera (이규브시스템스). ONC/RPC (OSF)
MOM (메시지 지향)	메시지 기반 비동기형 메시지 전달 (이기종 분산 데이터 시스템) 느린 대신 안정적 응답 필요 시 / 송수신측 간 메시지 큐 활용 → MQ (IBM) / Message Q (오라클) / JMS (JCP)
TP-Mornitor (Transaction Processing)	온라인 업무 처리 (항공기, 철도 예약) / 빠른 응답 속도 필요 시 트랜잭션 처리를 감시/제어 → Tuxedo (오라클) / Tmax
Legacyware	기존 앱에 새로운 업데이트 기능 부여
ORB (Object Request Broker)	객체 지향 / 코바(CORBA) 표준 스펙 구현 / 네트워크 호출 → Orbix (Micro Focus), CORBA (OMG)
WAS (Web Application Server)	동적 콘텐츠 처리 / 웹 환경 구현 적합 (클라이언트/서버 X) HTTP 세션 처리 기능 / 미션-크리티컬 기업 업무 포괄 JAVA, EJB 컴포넌트 기반 구현 가능 → Web Logic (오라클), WebSphere (IBM), JEUS, Tomcat

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

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

생성 패턴 (5개)	Abstract Factory : 연관 객체들을 그룹 생성 후 추상적 표현 Builder : 건축하듯 조립, 객체 생성 → 동일 객체 생성에도 다른 결과 Factory Method : 객체 생성을 서브클래스에서 결정 Prototype : 원본 객체를 복제 Singleton : 여러 프로세스가 하나의 객체를 동시에 참조 불가
구조 패턴 (7개)	Adaptor : 비호환 인터페이스에 호환성 부여하도록 변환 Bridge : 구현부에서 추상층 분리 후 독립적으로 확장하도록 구성 Composite : 복합, 단일 객체를 구분없이 사용 Decorator : 상속 사용 없이 객체 기능을 동적으로 확장 Façade : 상위 인터페이스 구성 후 서브클래스 기능을 간편하게 사용 Flyweight : 인스턴스 공유하여 메모리 절약 Proxy : 접근이 힘든 객체를 연결하는 인터페이스 역할
행위 패턴 (11개)	Chain of Responsibility : 한 객체 내 처리 불가 시 다음 객체로 이관 Command : 요청 명령어들을 추상/구체 클래스로 분리 후 단순화 Interpreter : 언어에 문법 표현 정의 Iterator : 동일 인터페이스 사용 Mediator : 객체들간 복잡한 상호작용을 캡슐화하여 객체로 정의 Memento : 객체를 특정 시점의 상태로 돌릴 수 있는 기능 제공 Observer : 한 객체 상태 변화 시 상속되어 있는 객체들에 변화 전달 State : 객체의 상태에 따라 동일 동작 다르게 처리 Strategy : 동일 계열 알고리즘을 개별적으로 캡슐화하여 상호 교환 Template Method : 서브클래스 내 공통 내용을 상위 클래스에 정의 Visitor : 처리 기능을 분리하여 별도의 클래스로 구성 분리된 처리 기능은 각 클래스를 방문하여 수행