시험에 나오는 것만 공부한다!

2024 시나공

기출문제집

정보처리기사

필기

17 응집도, 결합도

20,21 패턴

길벗알앤디 지음
(강윤석, 김용갑, 김우경, 김종일)

길벗

1과목 소프트웨어 설계

핵심 001 소프트웨어 생명 주기 (Software Life Cycle)

2400101

소프트웨어 생명 주기는 소프트웨어 개발 방법론의 바탕이 되는 것으로, 소프트웨어를 개발하기 위해 정의하고 운용, 유지보수 등의 과정을 각 단계별로 나눈 것이다.

- 소프트웨어 생명 주기는 소프트웨어 개발 단계와 각 단계별 주요 활동, 그리고 활동의 결과에 대한 산출물로 표현한다. 소프트웨어 수명 주기라고도 한다.
- 소프트웨어 생명 주기를 표현하는 형태를 소프트웨어 생명 주기 모형이라고 하며, 소프트웨어 프로세스 모형 또는 소프트웨어 공학 패러다임이라고도 한다.

21.3, 20.8

핵심 002 소프트웨어 공학

2400131

소프트웨어 공학의 개념

소프트웨어 공학(SE; Software Engineering)은 소프트웨어의 위기를 극복하기 위한 방안으로 연구된 학문이며 여러 가지 방법론과 도구, 관리 기법들을 통하여 소프트웨어의 품질과 생산성 향상을 목적으로 한다.

소프트웨어 공학의 기본 원칙

- 현대적인 프로그래밍 기술을 계속적으로 적용해야 한다.
- 개발된 소프트웨어의 품질이 유지되도록 지속적으로 검증해야 한다.
- 소프트웨어 개발 관련 사항 및 결과에 대한 명확한 기록을 유지해야 한다.

21.8, 21.3, 20.9, 20.8, 20.6

핵심 003 폭포수 모형 (Waterfall Model)

2400102

폭포수 모형은 폭포에서 한번 떨어진 물은 거슬러 올라갈 수 없듯이 소프트웨어 개발도 이전 단계로 돌아갈 수 없다는 전제하에 각 단계를 확실히 매듭짓고 그 결과를 철저하게 검토하여 승인 과정을 거친 후에 다음 단계를 진행하는 개발 방법론이다.

- 폭포수 모형은 소프트웨어 공학에서 가장 오래되고 가장 폭넓게 사용된 전통적인 소프트웨어 생명 주기 모형으로, 고전적 생명 주기 모형이라고도 한다.
- 소프트웨어 개발 과정의 한 단계가 끝나야만 다음 단계로 넘어갈 수 있는 선형 순차적 모형이다.
- 모형을 적용한 경험과 성공 사례가 많다.
- 각 단계가 끝난 후에는 다음 단계를 수행하기 위한 결과물이 명확하게 산출되어야 한다.

타당성 검토 → 계획 → 요구 분석 → 설계 → 구현 (코딩) → 시험 (검사) → 유지 보수

프로토타입 모형은 사용자의 요구사항을 명확히 파악하기 위해

23.2, 22.7, 22.3, 21.8, 21.3, 20.9, 20.8, 20.6

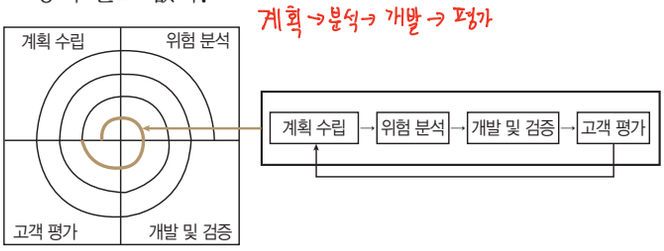
핵심 004 나선형 모형(Spiral Model, 점진적 모형)

2400104

나선형 모형은 보헴(Boehm)이 제안한 것으로, 폭포수 모형과 프로토타입 모형의 장점에 위험 분석 기능을 추가한 모형이다.

- 나선을 따라 돌듯이 여러 번의 소프트웨어 개발 과정을 거쳐 점진적으로 완벽한 최종 소프트웨어를 개발하는 것으로, 점진적 모형이라고도 한다.
- 소프트웨어를 개발하면서 발생할 수 있는 위험을 관리하고 최소화하는 것을 목적으로 한다.
- 점진적으로 개발 과정이 반복되므로 누락되거나 추가된 요구사항을 첨가할 수 있고, 정밀하며, 유지보수 과정이 필요 없다.

계획 → 분석 → 개발 → 평가



21.5, 20.9

핵심 005 애자일 모형(Agile Model)

2400105

애자일은 '민첩한', '기민한'이라는 의미로, 고객의 요구사항 변화에 유연하게 대응할 수 있도록 일정한 주기를 반복하면서 개발과정을 진행한다.

- 애자일 모형은 어느 특정 개발 방법론이 아니라 좋은 것을 빠르고 낭비 없게 만들기 위해 고객과의 소통에 초점을 맞춘 방법론을 통칭한다.
- 애자일 모형은 기업 활동 전반에 걸쳐 사용된다.
- 애자일 모형을 기반으로 하는 소프트웨어 개발 모형에는 스크럼(Scrum), XP(eXtreme Programming), 칸반(Kanban), Lean, 크리스탈(Crystal), ASD(Adaptive Software Development), 기능 중심 개발(FDD; Feature Driven Development), DSDM(Dynamic System Development Method), DAD(Disciplined Agile Delivery) 등이 있다.

주기마다 생성되는 결과물에 대해 고객의 평가와 요구를 적극 수용함
(폭포수 모형과 대조적)

23.7, 22.3, 21.8, 21.4, 21.3, 20.8

핵심 006 애자일 개발 4가지 핵심 가치

2400132

1. 프로세스와 도구보다는 개인과 상호작용에 더 가치를 둔다.
2. 방대한 문서보다는 실행되는 SW에 더 가치를 둔다.
3. 계약 협상보다는 고객과 협업에 더 가치를 둔다.
4. 계획을 따르기 보다는 변화에 반응하는 것에 더 가치를 둔다.

23.2, 22.3

핵심 007 스크럼의 개요

2400201

스크럼이란 럭비에서 반칙으로 경기가 중단된 경우 양 팀의 선수들이 럭비공을 가운데 두고 상대팀을 밀치기 위해 서로 대치해 있는 대형을 말한다. 스크럼은 이처럼 팀이 중심이 되어 개발의 효율성을 높인다는 의미가 내포된 용어이다.

- 스크럼은 팀원 스스로가 스크럼 팀을 구성(self-organizing)해야 하며, 개발 작업에 관한 모든 것을 스스로 해결(cross-functional)할 수 있어야 한다.
- 스크럼 팀은 제품 책임자, 스크럼 마스터, 개발팀으로 구성된다.

제품 책임자(PO; Product Owner)

- 이해관계자들 중 개발될 제품에 대한 이해도가 높고, 요구사항을 책임지고 의사 결정할 사람으로 선정하는데, 주로 개발 의뢰자나 사용자가 담당한다.
- 이해관계자들의 의견을 종합하여 제품에 대한 요구사항을 작성하는 주체다.
- 제품에 대한 테스트를 수행하면서 주기적으로 요구사항의 우선순위를 갱신한다.

스크럼 마스터(SM; Scrum Master)

- 스크럼 팀이 스크럼을 잘 수행할 수 있도록 객관적인 시각에서 조언을 해주는 가이드 역할을 수행한다. 팀원들을 통제하는 것이 목표가 아니다.
- 일일 스크럼 회의를 주관하여 진행 사항을 점검하고, 개발 과정에서 발생된 장애 요소를 공론화하여 처리한다.

개발팀(DT; Development Team)

- 제품 책임자와 스크럼 마스터를 제외한 모든 팀원으로, 개발자 외에도 디자이너, 테스터 등 제품 개발을 위해 참여하는 모든 사람이 대상이 된다.
- 보통 최대 인원은 7~8명이 적당하다.

23.2, 22.3

핵심 008 스크럼 개발 프로세스

2400202

제품 백로그 (Product Backlog)	제품 개발에 필요한 모든 요구사항(User Story)을 우선순위에 따라 나열한 목록
스프린트 계획 회의 (Sprint Planning Meeting)	제품 백로그 중 이번 스프린트에서 수행할 작업을 대상으로 단기 일정을 수립하는 것
스프린트(Sprint)	• 실제 개발 작업을 진행하는 과정으로, 보통 2 ~ 4주 정도의 기간 내에서 진행함 • 스프린트 백로그에 작성된 태스크를 대상으로 속도(Velocity)를 추정한 후 개발 담당자에게 할당함
일일 스크럼 회의 (Daily Scrum Meeting)	• 모든 팀원이 매일 약속된 시간에 약 15분 정도의 짧은 시간동안 진행 상황을 점검함 • 회의는 보통 서서 진행하며, 남은 작업 시간은 소멸 차트(Burn-down Chart)에 표시함
스프린트 검토 회의 (Sprint Review)	부분 또는 전체 완성 제품이 요구사항에 잘 부합되는지 사용자가 포함된 참석자 앞에서 테스팅을 수행함
스프린트 회고 (Sprint Retrospective)	스프린트 주기를 되돌아보며 정해놓은 규칙을 잘 준수했는지, 개선할 점은 없는지 등을 확인하고 기록함

시험에 나오는 것만 공부한다!

23.7, 23.5, 22.7, 22.4, 21.8, 20.9, 20.6

2400301

핵심 009 XP(eXtreme Programming)

XP(eXtreme Programming)는 수시로 발생하는 고객의 요구사항에 유연하게 대응하기 위해 고객의 참여와 개발 과정의 반복을 극대화하여 개발 생산성을 향상시키는 방법이다.

- XP는 짧고 반복적인 개발 주기, 단순한 설계, 고객의 적극적인 참여를 통해 소프트웨어를 빠르게 개발하는 것을 목적으로 한다.
- 릴리즈의 기간을 짧게 반복하면서 고객의 요구사항 반영에 대한 가시성을 높인다.
- XP의 5가지 핵심 가치 : 의사소통(Communication), 단순성(Simplicity), 용기(Courage), 존중(Respect), 피드백(Feedback)

23.5, 22.4, 20.9

2400331

핵심 010 XP의 주요 실천 방법 (Practice)

Pair Programming (짝 프로그래밍)	다른 사람과 함께 프로그래밍을 수행함으로써 개발에 대한 책임을 공동으로 나눠 갖는 환경을 조성함
Collective Ownership (공동 코드 소유)	개발 코드에 대한 권한과 책임을 공동으로 소유함
Test-Driven Development (테스트 주도 개발)	• 개발자가 실제 코드를 작성하기 전에 테스트 케이스를 먼저 작성하므로 자신이 무엇을 해야할지를 정확히 파악함 • 테스트가 지속적으로 진행될 수 있도록 자동화된 테스팅 도구(구조, 프레임워크)를 사용함
Whole Team (전체 팀)	개발에 참여하는 모든 구성원(고객 포함)들은 각자 자신의 역할이 있고 그 역할에 대한 책임을 가져야 함
Continuous Integration (계속적인 통합)	모듈 단위로 나눠서 개발된 코드들은 하나의 작업이 마무리될 때마다 지속적으로 통합됨
Design Improvement (디자인 개선) 또는 Refactoring(리팩토링)	프로그램 기능의 변경 없이, 단순화, 유연성 강화 등을 통해 시스템을 재구성함
Small Releases (소규모 릴리즈)	릴리즈 기간을 짧게 반복함으로써 고객의 요구 변화에 신속히 대응할 수 있음

21.3

2400401

핵심 011 현행 시스템 파악

단계	현행 시스템	내용
1단계	시스템 구성 파악	조직의 주요 업무를 담당하는 기간 업무와 이를 지원하는 지원 업무로 구분하여 기술함
	시스템 기능 파악	현재 제공하는 기능들을 주요 기능과 하부 기능, 세부 기능으로 구분하여 계층형으로 표시함
	시스템 인터페이스 파악	단위 업무 시스템 간에 주고받는 데이터의 종류, 형식, 프로토콜, 연계 유형, 주기 등을 명시함
2단계	아키텍처 구성 파악	최상위 수준에서 계층별로 표현한 아키텍처 구성도를 작성함
	소프트웨어 구성 파악	소프트웨어들의 제품명, 용도, 라이선스 적용 방식, 라이선스 수 등을 명시함
3단계	하드웨어 구성 파악	단위 업무 시스템들이 운용되는 서버의 주요 사양과 수량, 그리고 서버의 이중화의 적용 여부를 명시함
	네트워크 구성 파악	서버의 위치, 서버 간의 네트워크 연결 방식을 네트워크 구성도로 작성함

2459901

핵심 012 운영체제 (OS, Operating System)

운영체제는 컴퓨터 시스템의 자원들을 효율적으로 관리하며, 사용자가 컴퓨터를 편리하고 효율적으로 사용할 수 있도록 환경을 제공하는 소프트웨어이다.

- 컴퓨터 사용자와 컴퓨터 하드웨어 간의 인터페이스로서 동작하는 시스템 소프트웨어의 일종으로, 다른 응용 프로그램이 유용한 작업을 할 수 있도록 환경을 제공해준다.
- 컴퓨터 운영체제의 종류에는 Windows, UNIX, Linux, Mac OS 등이, 모바일 운영체제에는 iOS, Android 등이 있다.
- 운영체제 관련 요구사항 식별 시 고려사항
  - 가용성
  - 성능
  - 기술 지원
  - 주변 기기
  - 구축 비용

23.2, 20.6

핵심 013 데이터베이스 관리 시스템 (DBMS)

2459902

DBMS(DataBase Management System)는 사용자와 데이터베이스 사이에서 사용자의 요구에 따라 정보를 생성해 주고, 데이터베이스를 관리해 주는 소프트웨어이다.

- DBMS는 기존의 파일 시스템이 갖는 데이터의 종속성과 중복성의 문제를 해결하기 위해 제안된 시스템으로, 모든 응용 프로그램들이 데이터베이스를 공용할 수 있도록 관리해 준다.
- DBMS는 데이터베이스의 구성, 접근 방법, 유지관리에 대한 모든 책임을 진다.
- DBMS의 종류에는 Oracle, IBM DB2, Microsoft SQL Server, MySQL, SQLite, MongoDB, Redis 등이 있다.
- DBMS 관련 요구사항 식별 시 고려사항
  - 가용성
  - 성능
  - 기술 지원
  - 상호 호환성
  - 구축 비용

21.3

핵심 014 웹 애플리케이션 서버(WAS)

2400506

웹 애플리케이션 서버는 정적인 콘텐츠 처리를 하는 웹 서버와 달리 사용자의 요구에 따라 변하는 동적인 콘텐츠를 처리하기 위해 사용되는 미들웨어이다.

- 데이터 접근, 세션 관리, 트랜잭션 관리 등을 위한 라이브러리를 제공한다.
- 주로 데이터베이스 서버와 연동해서 사용한다.
- 웹 애플리케이션 서버의 종류에는 Tomcat, GlassFish, JBoss, Jetty, JEUS, Resin, WebLogic, WebSphere 등이 있다.

23.2, 21.8

핵심 015 요구사항 정의

2459903

요구사항은 소프트웨어가 어떤 문제를 해결하기 위해 제공하는 서비스에 대한 설명과 정상적으로 운영되는데 필요한 제약조건 등을 나타낸다.

요구사항의 유형

유형	내용
기능 요구사항 (Functional requirements)	• 시스템이 무엇을 하는지, 어떤 기능을 하는지에 대한 사항 • 시스템의 입력이나 출력으로 무엇이 포함되어야 하는지, 시스템이 어떤 데이터를 저장하거나 연산을 수행해야 하는지에 대한 사항 • 시스템이 반드시 수행해야 하는 기능 • 사용자가 시스템을 통해 제공받기를 원하는 기능
비기능 요구사항 (Non-functional requirements)	• 시스템 장비 구성 요구사항 : 하드웨어, 소프트웨어, 네트워크 등의 시스템 장비 구성에 대한 요구사항 • 성능 요구사항 : 처리 속도 및 시간, 처리량, 동적 · 정적 적용량, 가용성 등 성능에 대한 요구사항 • 인터페이스 요구사항 : 시스템 인터페이스와 사용자 인터페이스에 대한 요구사항으로 다른 소프트웨어, 하드웨어 및 통신 인터페이스, 다른 시스템과의 정보 교환에 사용되는 프로토콜과의 연계도 포함하여 기술 • 데이터 요구사항 : 초기 자료 구축 및 데이터 변환을 위한 대상, 방법, 보안이 필요한 데이터 등 데이터를 구축하기 위해 필요한 요구사항 • 테스트 요구사항 : 도입되는 장비의 성능 테스트(BMT)나 구축된 시스템이 제대로 운영되는지를 테스트하고 점검하기 위한 테스트 요구사항 • 보안 요구사항 : 시스템의 데이터 및 기능, 운영 접근을 통제하기 위한 요구사항 • 품질 요구사항 : 관리가 필요한 품질 항목, 품질 평가 대상에 대한 요구사항으로 가용성, 정합성, 상호 호환성, 대응성, 신뢰성, 사용성, 유지 · 관리성, 이식성, 확장성, 보안성 등으로 구분하여 기술 • 제약사항 : 시스템 설계, 구축, 운영과 관련하여 사전에 파악된 기술, 표준, 업무, 법 · 제도 등의 제약조건 • 프로젝트 관리 요구사항 : 프로젝트의 원활한 수행을 위한 관리 방법에 대한 요구사항 • 프로젝트 지원 요구사항 : 프로젝트의 원활한 수행을 위한 지원 사항이나 방안에 대한 요구사항

21.8, 21.5, 20.8

핵심 016 요구사항 개발 프로세스

2459904

요구사항 개발 프로세스는 개발 대상에 대한 요구사항을 체계적으로 도출하고 이를 분석한 후 분석 결과를 명세서(Specification Document)에 정리한 다음 마지막으로 이를 확인 및 검증하는 일련의 구조화된 활동이다.

도출(Elicitation) → 분석(Analysis) → 명세(Specification) → 확인(Validation)

요구사항 도출(Requirement Elicitation, 요구사항 수집)

요구사항 도출은 시스템, 사용자, 그리고 시스템 개발에 관련된 사람들이 서로 의견을 교환하여 요구사항이 어디에 있는지, 어떻게 수집할 것인지를 식별하고 이해하는 과정이다.

- 요구사항을 도출하는 주요 기법에는 청취와 인터뷰, 설문, 브레인스토밍, 워크샵, 프로토타이핑, 유스케이스 등이 있다.

요구사항 분석(Requirement Analysis)

요구사항 분석은 개발 대상에 대한 사용자의 요구사항 중 명확하지 않거나 모호하여 이해되지 않는 부분을 발견하고 이를 걸러내기 위한 과정이다.

- 요구사항 분석에는 자료 흐름도(DFD), 자료 사전(DD) 등의 도구가 사용된다.

요구사항 명세(Requirement Specification)

요구사항 명세는 분석된 요구사항을 바탕으로 모델을 작성하고 문서화하는 것을 의미한다.

- 구체적인 명세를 위해 소단위 명세서(Mini-Spec)가 사용될 수 있다.

요구사항 확인(Requirement Validation, 요구사항 검증)

요구사항 확인은 개발 자원을 요구사항에 할당하기 전에 요구사항 명세서가 정확하고 완전하게 작성되었는지를 검토하는 활동이다.

20.9

핵심 017 요구사항 명세 기법

2459905

구분	정형 명세 기법	비정형 명세 기법
기법	수학적 원리 기반, 모델 기반	상태/기능/객체 중심
작성 방법	수학적 기호, 정형화된 표기법	일반 명사, 동사 등의 자연어를 기반으로 서술 또는 다이어그램으로 작성
특징	• 요구사항을 정확하고 간결하게 표현할 수 있음 • 요구사항에 대한 결과가 작성자에 관계없이 일관성이 있으므로 완전성 검증이 가능함 • 표기법이 어려워 사용자가 이해하기 어려움	• 자연어의 사용으로 인해 요구사항에 대한 결과가 작성자에 따라 다를 수 있어 일관성이 떨어지고, 해석이 달라질 수 있음 • 내용의 이해가 쉬어 의사소통이 용이함
종류	VDM, Z, Petri-net, CSP 등	FSM, Decision Table, ER모델링, State Chart(SADT) 등

22.3, 21.8, 20.9, 20.6

핵심 018 요구사항 분석의 개요

2400701

요구사항 분석은 소프트웨어 개발의 실제적인 첫 단계로 개발 대상에 대한 사용자의 요구사항을 이해하고 문서화(명세화)하는 활동을 의미한다.

- 사용자 요구의 타당성을 조사하고 비용과 일정에 대한 제약을 설정한다.
- 사용자의 요구를 정확하게 추출하여 목표를 정하고, 어떤 방식으로 해결할 것인지를 결정한다.
- 요구사항 분석을 통한 결과는 소프트웨어 설계 단계에서 필요한 기본적인 자료가 되므로 사용자의 요구사항을 정확하고 일관성 있게 분석하여 문서화해야 한다.
- 소프트웨어 분석가에 의해 요구사항 분석이 수행되며, 이 작업 단계를 요구사항 분석 단계라고 한다.
- 요구사항 분석을 위해 UML(Unified Modeling Language), 자료 흐름도(DFD), 자료 사전(DD), 소단위 명세서(Mini-Spec.), 개체 관계도(ERD), 상태 전이도(STD), 제어 명세서 등의 도구를 이용한다.

23.7, 23.2, 22.7, 22.3, 20.9, 20.8, 20.6

2400703

핵심 019 자료 흐름도(DFD)

자료 흐름도(DFD; Data Flow Diagram)는 요구사항 분석에서 자료의 흐름 및 변환 과정과 기능을 도형 중심으로 기술하는 방법으로 자료 흐름 그래프, 버블 차트라고도 한다.

- 자료 흐름도에서는 자료의 흐름과 기능을 프로세스(Process), 자료 흐름(Flow), 자료 저장소(Data Store), 단말(Terminator)의 네 가지 기본 기호로 표시한다.

기 호	의 미	표기법	
		Yourdon/ DeMacro	Gane/ Sarson
프로세스 (Process)	• 자료를 변환시키는 시스템의 한 부분(처리 과정)을 나타내며 처리, 기능, 변환, 버블이라고도 함 • 원이나 둥근 사각형으로 표시하고 그 안에 프로세스 이름을 기입함	물품 확인	물품 확인
자료 흐름 (Data Flow)	• 자료의 이동(흐름)이나 연관관계를 나타냄 • 화살표 위에 자료의 이름을 기입함	물품 코드 →	
자료 저장소 (Data Store)	• 시스템에서의 자료 저장소(파일, 데이터베이스)를 나타냄 • 도형 안에 자료 저장소 이름을 기입함	물품대장	ID 물품대장
단말 (Terminator)	• 시스템과 교신하는 외부 개체로, 입력 데이터가 만들어지고 출력 데이터를 받음(정보의 생산자와 소비자) • 도형 안에 이름을 기입함	공장	

20.9, 20.8, 20.6

2400704

핵심 020 자료 사전

자료 사전(DD; Data Dictionary)은 자료 흐름도에 있는 자료를 더 자세히 정의하고 기록한 것이며, 이처럼 데이터를 설명하는 데이터를 데이터의 데이터 또는 메타 데이터(Meta Data)라고 한다.

기 호	의 미
=	자료의 정의 : ~로 구성되어 있다(is composed of)
+	자료의 연결 : 그리고(and)
( )	자료의 생략 : 생략 가능한 자료(Optional)
[|]	자료의 선택 : 또는(or)
{ }	자료의 반복 : Iteration of ① { }n : n번 이상 반복 ② { }n : 최대로 n번 반복 ③ { }nm : m 이상 n 이하로 반복
* *	자료의 설명 : 주석(Comment)

20.9

2400801

핵심 021 요구사항 분석을 위한 CASE (자동화 도구)

요구사항 분석을 위한 자동화 도구는 요구사항을 자동으로 분석하고, 요구사항 분석 명세서를 기술하도록 개발된 도구를 의미한다.

종류

- SADT(Structured Analysis and Design Technique)
  - SoftTech 사에서 개발한 것으로 시스템 정의, 소프트웨어 요구사항 분석, 시스템/소프트웨어 설계를 위해 널리 이용되어 온 구조적 분석 및 설계 도구이다.
- SREM(Software Requirements Engineering Methodology) = RSL/REVS
  - TRW 사가 우주 국방 시스템 그룹에 의해 실시간 처리 소프트웨어 시스템에서 요구사항을 명확히 기술하도록 할 목적으로 개발한 것으로, RSL과 REVS를 사용하는 자동화 도구이다.
  - RSL(Requirement Statement Language) : 요소, 속성, 관계, 구조들을 기술하는 요구사항 기술 언어
  - REVS(Requirement Engineering and Validation System) : RSL로 기술된 요구사항들을 자동으로 분석하여 요구사항 분석 명세서를 출력하는 요구사항 분석기

- PSL/PSA
  - 미시간 대학에서 개발한 것으로 PSL과 PSA를 사용하는 자동화 도구이다.
- TAGS(Technology for Automated Generation of Systems)
  - 시스템 공학 방법 응용에 대한 자동 접근 방법으로, 개발 주기의 전 과정에 이용할 수 있는 통합 자동화 도구이다.

23.7, 22.7, 20.6

2400802

핵심 022 HIPO

HIPO(Hierarchy Input Process Output)는 시스템의 분석 및 설계나 문서화할 때 사용되는 기법으로, 시스템 실행 과정인 입력, 처리, 출력의 기능을 나타낸다.

- 기본 시스템 모델은 입력, 처리, 출력으로 구성되며, 하향식 소프트웨어 개발을 위한 문서화 도구이다.
- 체계적인 문서 관리가 가능하다.
- 기호, 도표 등을 사용하므로 보기 쉽고 이해하기도 쉽다.
- 기능과 자료의 의존 관계를 동시에 표현할 수 있다.
- 변경, 유지보수가 용이하다.
- 시스템의 기능을 여러 개의 고유 모듈들로 분할하여 이들 간의 인터페이스를 계층 구조로 표현한 것을 HIPO Chart라고 한다.

HIPO Chart의 종류

- **가시적 도표(도식 목차)** : 시스템의 전체적인 기능과 흐름을 보여주는 계층(Tree) 구조도
- **총체적 도표(총괄도표, 개요 도표)** : 프로그램을 구성하는 기능을 기술한 것으로 입력, 처리, 출력에 대한 전반적인 정보를 제공하는 도표
- **세부적 도표(상세 도표)** : 총체적 도표에 표시된 기능을 구성하는 기본 요소들을 상세히 기술하는 도표

22.3, 20.9

2400901

핵심 023 UML(Unified Modeling Language)의 개요

UML은 시스템 분석, 설계, 구현 등 시스템 개발 과정에서 시스템 개발자와 고객 또는 개발자 상호간의 의사소통이 원활하게 이루어지도록 표준화한 대표적인 객체지향 모델링 언어이다.

- UML은 Rumbaugh(OMT), Booch, Jacobson 등의 객체지향 방법론의 장점을 통합하였으며, 객체 기술에 관한 국제표준화기구인 OMG(Object Management Group)에서 표준으로 지정하였다.
- UML을 이용하여 시스템의 구조를 표현하는 6개의 구조 다이어그램과 시스템의 동작을 표현하는 7개의 행위 다이어그램을 작성할 수 있다.
- 각각의 다이어그램은 사물과 사물 간의 관계를 용도에 맞게 표현한다.
- UML의 구성 요소에는 사물(Things), 관계(Relationships), 다이어그램(Diagram) 등이 있다.

23.5, 22.7, 21.8, 21.5, 20.8

2400903

핵심 024 관계(Relationships)

관계는 사물과 사물 사이의 연관성을 표현하는 것으로, 연관 관계, 집합 관계, 포함 관계, 일반화 관계, 의존 관계, 실체화 관계 등이 있다.

연관(Association) 관계

연관 관계는 2개 이상의 사물이 서로 관련되어 있음을 표현한다.

집합(Aggregation) 관계

집합 관계는 하나의 사물이 다른 사물에 포함되어 있는 관계를 표현한다.

포함(Composition) 관계

포함 관계는 집합 관계의 특수한 형태로, 포함하는 사물의 변화가 포함되는 사물에게 영향을 미치는 관계를 표현한다.

일반화(Generalization) 관계

일반화 관계는 하나의 사물이 다른 사물에 비해 더 일반적인지 구체적인지를 표현한다.

의존(Dependency) 관계

의존 관계는 연관 관계와 같이 사물 사이에 서로 연관은 있으나 필요에 의해 서로에게 영향을 주는 짧은 시간 동안만 연관을 유지하는 관계를 표현한다.

- 일반적으로 한 클래스가 다른 클래스를 오퍼레이션의 매개 변수로 사용하는 경우에 나타나는 관계이다.

실체화(Realization) 관계

실체화 관계는 사물이 할 수 있거나 해야 하는 기능(행위, 인터페이스)으로 서로를 그룹화 할 수 있는 관계를 표현한다.

23.7, 23.5, 23.2, 22.3, 21.8, 21.5, 21.3, 20.9, 20.8, 20.6

2400904

핵심 025 다이어그램(Diagram)

다이어그램은 사물과 관계를 도형으로 표현한 것이다.

- 여러 관점에서 시스템을 가시화한 뷰(View)를 제공함으로써 의사소통에 도움을 준다.
- 정적 모델링에서는 주로 구조적 다이어그램을 사용하고 동적 모델링에서는 주로 행위 다이어그램을 사용한다.
- 구조적(Structural) 다이어그램의 종류

클래스 다이어그램 (Class Diagram)	• 클래스와 클래스가 가지는 속성, 클래스 사이의 관계를 표현함 • 시스템의 구조를 파악하고 구조상의 문제점을 도출할 수 있음
객체 다이어그램 (Object Diagram)	• 클래스에 속한 사물(객체)들, 즉 인스턴스(Instance)를 특정 시점의 객체와 객체 사이의 관계로 표현함 • 럼바우(Rumbaugh) 객체지향 분석 기법에서 객체 모델링에 활용됨
컴포넌트 다이어그램 (Component Diagram)	• 실제 구현 모듈인 컴포넌트 간의 관계나 컴포넌트 간의 인터페이스를 표현함 • 구현 단계에서 사용되는 다이어그램
배치 다이어그램 (Deployment Diagram)	• 결과물, 프로세스, 컴포넌트 등 물리적 요소들의 위치를 표현함 • 노드와 의사소통(통신) 경로로 표현함 • 구현 단계에서 사용되는 다이어그램
복합체 구조 다이어그램 (Composite Structure Diagram)	클래스나 컴포넌트가 복합 구조를 갖는 경우 그 내부 구조를 표현함
패키지 다이어그램 (Package Diagram)	유스케이스나 클래스 등의 모델 요소들을 그룹화한 패키지들의 관계를 표현함

- 행위(Behavioral) 다이어그램의 종류

유스케이스 다이어그램 (Use Case Diagram)	• 사용자의 요구를 분석하는 것으로 기능 모델링 작업에 사용함 • 사용자(Actor)와 사용 사례(Use Case)로 구성되며, 사용 사례 간에는 여러 형태의 관계로 이루어짐
순차 다이어그램 (Sequence Diagram)	상호 작용하는 시스템이나 객체들이 주고받는 메시지를 표현함
커뮤니케이션 다이어그램 (Communication Diagram)	순차 다이어그램과 같이 동작에 참여하는 객체들이 주고받는 메시지를 표현하는데, 메시지뿐만 아니라 객체들 간의 연관까지 표현함
상태 다이어그램 (State Diagram)	• 하나의 객체가 자신이 속한 클래스의 상태 변화 혹은 다른 객체와의 상호 작용에 따라 상태가 어떻게 변화하는지를 표현함 • 럼바우(Rumbaugh) 객체지향 분석 기법에서 동적 모델링에 활용됨
활동 다이어그램 (Activity Diagram)	시스템이 어떤 기능을 수행하는지 객체의 처리 로직이나 조건에 따른 처리의 흐름을 순서에 따라 표현함
상호작용 개요 다이어그램 (Interaction Overview Diagram)	상호작용 다이어그램 간의 제어 흐름을 표현함
타이밍 다이어그램 (Timing Diagram)	객체 상태 변화와 시간 제약을 명시적으로 표현함

23.7, 22.7, 20.6

2400931

핵심 026 스테레오 타입(Stereotype)

스테레오 타입은 UML에서 표현하는 기본 기능 외에 추가적인 기능을 표현하기 위해 사용한다.

- 길러멧(Guilemet)이라고 부르는 겹화살괄호(《 》) 사이에 표현할 형태를 기술한다.
- 주로 표현되는 형태는 다음과 같다.

《include》	연결된 다른 UML 요소에 대해 포함 관계에 있는 경우
《extend》	연결된 다른 UML 요소에 대해 확장 관계에 있는 경우
《interface》	인터페이스를 정의하는 경우
《exception》	예외를 정의하는 경우
《constructor》	생성자 역할을 수행하는 경우

23.7, 23.5, 22.4, 21.5, 21.3

핵심 027 유스케이스(Use Case) 다이어그램

2401001

유스케이스 다이어그램은 개발될 시스템과 관련된 외부 요소들, 즉 사용자와 다른 외부 시스템들이 개발될 시스템을 이용해 수행할 수 있는 기능을 사용자의 관점(View)에서 표현한 것이다.

유스케이스 다이어그램의 구성 요소

유스케이스 다이어그램은 시스템, 액터, 유스케이스, 관계로 구성된다.

시스템(System) / 시스템 범위 (System Scope)	시스템 내부에서 수행되는 기능들을 외부 시스템과 구분하기 위해 시스템 내부의 유스케이스들을 사각형으로 묶어 시스템의 범위를 표현함
액터(Actor)	• 시스템과 상호작용을 하는 모든 외부 요소로, 사람이나 외부 시스템을 의미함 • 주액터 : 시스템을 사용함으로써 이득을 얻는 대상으로, 주로 사람이 해당함 • 부액터 : 주액터의 목적 달성을 위해 시스템에 서비스를 제공하는 외부 시스템으로, 조직이나 기관 등이 될 수 있음
유스케이스 (Use Case)	사용자가 보는 관점에서 시스템이 액터에게 제공하는 서비스 또는 기능을 표현한 것
관계 (Relationship)	유스케이스 다이이그램에서 관계는 액터와 유스케이스, 유스케이스와 유스케이스 사이에서 나타날 수 있으며, 연관 관계, 포함 관계, 확장 관계, 일반화 관계를 표현할 수 있음

21.8, 21.3

핵심 028 클래스(Class) 다이어그램

2401002

클래스 다이어그램은 시스템을 구성하는 클래스, 클래스의 특성인 속성과 오퍼레이션, 속성과 오퍼레이션에 대한 제약조건, 클래스 사이의 관계를 표현한 것이다.

클래스 다이어그램의 구성 요소

클래스 다이어그램은 클래스, 제약조건, 관계 등으로 구성된다.

클래스(Class)	• 클래스는 각각의 객체들이 갖는 속성과 오퍼레이션(동작)을 표현함 • 일반적으로 3개의 구획(Compartment)으로 나눠 클래스의 이름, 속성, 오퍼레이션을 표기함 • 속성(Attribute) : 클래스의 상태나 정보를 표현함 • 오퍼레이션(Operation) : 클래스가 수행할 수 있는 동작으로, 함수(메소드, Method)라고도 함
제약조건	속성에 입력될 값에 대한 제약조건이나 오퍼레이션 수행 전후에 지정해야 할 조건이 있다면 이를 적음
관계 (Relationships)	• 관계는 클래스와 클래스 사이의 연관성을 표현함 • 클래스 다이어그램에 표현하는 관계에는 연관 관계, 집합 관계, 포함 관계, 일반화 관계, 의존 관계가 있음

23.2, 22.7, 22.4, 21.8, 20.8

핵심 029 순차(Sequence) 다이어그램

2401003

순차 다이어그램은 시스템이나 객체들이 메시지를 주고받으며 시간의 흐름에 따라 상호 작용하는 과정을 액터, 객체, 메시지 등의 요소를 사용하여 그림으로 표현한 것이다.

순차 다이어그램의 구성 요소

순차 다이어그램은 액터, 객체, 생명선, 실행, 메시지 등으로 구성된다.

액터 (Actor)	시스템으로부터 서비스를 요청하는 외부 요소로, 사람이나 외부 시스템을 의미함
객체 (Object)	메시지를 주고받는 주체
생명선 (Lifeline)	객체가 메모리에 존재하는 기간으로, 객체 아래쪽에 점선을 그어 표현함
실행 상자 (Active Box)	객체가 메시지를 주고받으며 구동되고 있음을 표현함
메시지 (Message)	객체가 상호 작용을 위해 주고받는 메시지

21.5

핵심 030 사용자 인터페이스(UI)의 특징

2401102

- 사용자의 만족도에 가장 큰 영향을 미치는 중요한 요소로, 소프트웨어 영역 중 변경이 가장 많이 발생한다.
- 사용자의 편리성과 가독성을 높임으로써 작업 시간을 단축시키고 업무에 대한 이해도를 높여준다.
- 최소한의 노력으로 원하는 결과를 얻을 수 있게 한다.
- 사용자 중심으로 설계되어 사용자 중심의 상호 작용이 되도록 한다.
- 수행 결과의 오류를 줄인다.

- 사용자의 막연한 작업 기능에 대해 구체적인 방법을 제시해 준다.
- 정보 제공자와 공급자 간의 매개 역할을 수행한다.
- 사용자 인터페이스를 설계하기 위해서는 소프트웨어 아키텍처를 반드시 숙지해야 한다.

23.7, 23.5, 22.7, 22.4, 21.8

2401103

핵심 031 사용자 인터페이스의 구분

- CLI(Command Line Interface) : 명령과 출력이 텍스트 형태로 이뤄지는 인터페이스
- GUI(Graphical User Interface) : 아이콘이나 메뉴를 마우스로 선택하여 작업을 수행하는 그래픽 환경의 인터페이스
- NUI(Natural User Interface) : 사용자의 말이나 행동으로 기기를 조작하는 인터페이스
- VUI(Voice User Interface) : 사람의 음성으로 기기를 조작하는 인터페이스
- OUI(Organic User Interface) : 모든 사물과 사용자 간의 상호작용을 위한 인터페이스로, 소프트웨어가 아닌 하드웨어 분야에서 사물 인터넷(Internet of Things), 가상현실(Virtual Reality), 증강현실(Augmented Reality), 혼합현실(Mixed Reality) 등과 함께 대두되고 있음

Pan (누른 채 계속 움직임)
Flick (빠르게 스크롤)
Pinch (두 손가락으로 좁히기/넓히기)

23.7, 20.8, 20.6

2401104

핵심 032 사용자 인터페이스의 기본 원칙

- 직관성 : 누구나 쉽게 이해하고 사용할 수 있어야 함
- 유효성 : 사용자의 목적을 정확하고 완벽하게 달성해야 함
- 학습성 : 누구나 쉽게 배우고 익힐 수 있어야 함
- 유연성 : 사용자의 요구사항을 최대한 수용하고 실수를 최소화해야 함

22.4, 21.8, 20.8, 20.6

2401105

핵심 033 사용자 인터페이스의 설계 지침

- 사용자 중심 : 사용자가 쉽게 이해하고 편리하게 사용할 수 있는 환경을 제공하며, 실사용자에 대한 이해가 바탕이 되어야 함
- 사용성 : 사용자가 소프트웨어를 얼마나 빠르고 쉽게 이해할 수 있는지, 얼마나 편리하고 효율적으로 사용할 수 있는지를 말하는 것으로, 사용자 인터페이스 설계 시 가장 우선적으로 고려해야 함
- 심미성 : 디자인적으로 완성도 높게 글꼴이나 색상을 적용하고 그래픽 요소를 배치하여 가독성을 높일 수 있도록 설계해야 함
- 오류 발생 해결 : 오류가 발생하면 사용자가 쉽게 인지할 수 있도록 설계해야 함

20.9

2401106

핵심 034 사용자 인터페이스 개발 시스템의 기능

- 사용자의 입력을 검증할 수 있어야 한다.
- 에러 처리와 그와 관련된 에러 메시지를 표시할 수 있어야 한다.
- 도움과 프롬프트(Prompt)를 제공해야 한다.

23.5, 23.2, 22.3

2459906

핵심 035 UI 설계 도구

UI 설계 도구

UI 설계 도구는 사용자의 요구사항에 맞게 UI의 화면 구조나 화면 배치 등을 설계할 때 사용하는 도구로, 종류에는 와이어프레임, 목업, 스토리보드, 프로토타입, 유스케이스 등이 있다.

와이어프레임(Wireframe)

- 와이어프레임은 기획 단계의 초기에 제작하는 것으로, 페이지에 대한 개략적인 레이아웃이나 UI 요소 등에 대한 뼈대를 설계하는 단계이다.
- 개발자나 디자이너 등이 레이아웃을 협의하거나 현재 진행 상태 등을 공유하기 위해 와이어프레임을 사용한다.
- 와이어프레임 툴 : 손그림, 파워포인트, 키노트, 스케치, 일러스트, 포토샵 등

목업(Mockup)

- 목업은 디자인, 사용 방법 설명, 평가 등을 위해 와이어프레임보다 좀 더 실제 화면과 유사하게 만든 정적인 형태의 모형이다.
- 시각적으로만 구성 요소를 배치하는 것으로 실제로 구현되지는 않는다.
- **목업 툴** : 파워 목업, 발사믹 목업 등

스토리보드(Story Board)

- 스토리보드는 와이어프레임에 콘텐츠에 대한 설명, 페이지 간 이동 흐름 등을 추가한 문서이다.
- 디자이너와 개발자가 최종적으로 참고하는 작업 지침서로, 정책, 프로세스, 콘텐츠 구성, 와이어프레임, 기능 정의 등 서비스 구축을 위한 모든 정보가 들어 있다.
- **스토리보드 툴** : 파워포인트, 키노트, 스케치, Axure 등

프로토타입(Prototype)

- 프로토타입은 와이어프레임이나 스토리보드 등에 인터랙션을 적용함으로써 실제 구현된 것처럼 테스트가 가능한 동적인 형태의 모형이다.
- 프로토타입은 사용성 테스트나 작업자 간 서비스 이해를 위해 작성하는 샘플이다.

유스케이스(Use Case)

- 유스케이스는 사용자 측면에서의 요구사항으로, 사용자가 원하는 목표를 달성하기 위해 수행할 내용을 기술한다.
- 사용자의 요구사항을 빠르게 파악함으로써 프로젝트의 초기에 시스템의 기능적인 요구를 결정하고 그 결과를 문서화할 수 있다.

21.8, 21.3, 20.8, 20.6

핵심 036 품질 요구사항

2459907

소프트웨어의 품질은 소프트웨어의 기능, 성능, 만족도 등 소프트웨어에 대한 요구사항이 얼마나 충족하는가를 나타내는 소프트웨어 특성의 총체이다.

- 소프트웨어의 품질은 사용자의 요구사항을 충족시킴으로써 확립된다.
- ISO/IEC 9126 : 소프트웨어의 품질 특성과 평가를 위한 표준 지침으로서 국제 표준으로 널리 사용됨
- ISO/IEC 25010 : 소프트웨어 제품에 대한 국제 표준으로, 2011년에 ISO/IEC 9126을 개정하여 만들었음
- ISO/IEC 12119 : ISO/IEC 9126을 준수한 품질 표준으로, 테스트 절차를 포함하여 규정함
- ISO/IEC 14598 : 소프트웨어 품질의 측정과 평가에 필요 절차를 규정한 표준으로, 개발자, 구매자, 평가자 별로 수행해야 할 제품 평가 활동을 규정함

기능성 (Functionality)	• 소프트웨어가 사용자의 요구사항을 정확하게 만족하는 기능을 제공하는지 여부를 나타냄 • 하위 특성 : 적절성/적합성(Suitability), 정밀성/정확성(Accuracy), 상호 운용성(Interoperability), 보안성(Security), 준수성(Compliance)
신뢰성 (Reliability)	• 소프트웨어가 요구된 기능을 정확하고 일관되게 오류 없이 수행할 수 있는 정도를 나타냄 • 하위 특성 : 성숙성(Maturity), 고장 허용성(Fault Tolerance), 회복성(Recoverability)
사용성 (Usability)	• 사용자와 컴퓨터 사이에 발생하는 어떠한 행위에 대하여 사용자가 정확하게 이해하고 사용하며, 향후 다시 사용하고 싶은 정도를 나타냄 • 하위 특성 : 이해성(Understandability), 학습성(Learnability), 운용성(Operability), 친밀성(Attractiveness)
효율성 (Efficiency)	• 사용자가 요구하는 기능을 할당된 시간 동안 한정된 자원으로 얼마나 빨리 처리할 수 있는지 정도를 나타냄 • 하위 특성 : 시간 효율성(Time Behaviour), 자원 효율성(Resource Behaviour)
유지 보수성 (Maintainability)	• 환경의 변화 또는 새로운 요구사항이 발생했을 때 소프트웨어를 개선하거나 확장할 수 있는 정도를 나타냄 • 하위 특성 : 분석성(Analyzability), 변경성(Changeability), 안정성(Stability), 시험성(Testability)
이식성 (Portability)	• 소프트웨어가 다른 환경에서도 얼마나 쉽게 적용할 수 있는지 정도를 나타냄 • 하위 특성 : 적용성(Adaptability), 설치성(Installability), 대체성(Replaceability), 공존성(Co-existence)

21.3

핵심 037 UI 요소

2401931

- 체크 박스(Check Box) : 여러 개의 선택 상황에서 1개 이상의 값을 선택할 수 있는 버튼임
- 라디오 버튼(Radio Button) : 여러 항목 중 하나만 선택할 수 있는 버튼임
- 텍스트 박스(Text Box) : 사용자가 데이터를 입력하고 수정할 수 있는 상자임
- 콤보 상자(Combo Box) : 이미 지정된 목록 상자에 내용을 표시하여 선택하거나 새로 입력할 수 있는 상자임
- 목록 상자(List Box) : 콤보 상자와 같이 목록을 표시하지만 새로운 내용을 입력할 수 없는 상자임

20.9

핵심 038 상위 설계와 하위 설계

2402131

소프트웨어 개발의 설계 단계는 크게 상위 설계와 하위 설계로 구분할 수 있다.

	상위 설계	하위 설계
별칭	아키텍처 설계, 예비 설계	모듈 설계, 상세 설계
설계 대상	시스템의 전체적인 구조	시스템의 내부 구조 및 행위
세부 목록	구조, DB, 인터페이스	컴포넌트, 자료 구조, 알고리즘

23.7, 22.3, 21.8

핵심 039 소프트웨어 아키텍처 설계의 기본 원리

2459908

모듈의 수가 증가하면 상대적으로 각 모듈의 크기가 작아진다.

모듈화 (Modularity)	모듈화란 소프트웨어의 성능을 향상시키거나 시스템의 수정 및 재사용, 유지 관리 등이 용이하도록 시스템의 기능들을 모듈 단위로 나누는 것을 의미함
추상화 (Abstraction)	• 추상화는 문제의 전체적이고 포괄적인 개념을 설계한 후 차례로 세분화하여 구체화시켜 나가는 것 • **추상화의 유형** : 과정 추상화, 데이터 추상화, 제어 추상화
단계적 분해 (Stepwise Refinement)	단계적 분해는 Niklaus Wirth에 의해 제안된 하향식 설계 전략으로, 문제를 상위의 중요 개념으로부터 하위의 개념으로 구체화시키는 분할 기법
정보 은닉 (Information Hiding)	정보 은닉은 한 모듈 내부에 포함된 절차와 자료들의 정보가 감추어져 다른 모듈이 접근하거나 변경하지 못하도록 하는 기법

21.5

핵심 040 소프트웨어 아키텍처의 품질 속성

2402106

소프트웨어 아키텍처의 품질 속성은 소프트웨어 아키텍처가 이해 관계자들이 요구하는 수준의 품질을 유지 및 보장할 수 있게 설계되었는지를 확인하기 위해 품질 평가 요소들을 시스템 측면, 비즈니스 측면, 아키텍처 측면으로 구분하여 구체화시켜 놓은 것이다.

- 시스템 측면 : 성능, 보안, 가용성, 기능성, 사용성, 변경 용이성, 확장성 등
- 비즈니스 측면 : 시장 적시성, 비용과 혜택, 예상 시스템 수명 등
- 아키텍처 측면 : 개념적 무결성, 정확성, 완결성, 구축 가능성 등

23.5, 23.2, 22.3

핵심 041 소프트웨어 아키텍처의 설계 과정

2402107

❶ 설계 목표 설정 : 시스템의 개발 방향을 명확히 하기 위해 설계에 영향을 주는 비즈니스 목표, 우선순위 등의 요구사항을 분석하여 전체 시스템의 설계 목표를 설정함

❷ 시스템 타입 결정 : 시스템과 서브시스템의 타입을 결정하고, 설계 목표와 함께 고려하여 아키텍처 패턴을 선택함

❸ 아키텍처 패턴 적용 : 아키텍처 패턴을 참조하여 시스템의 표준 아키텍처를 설계함

❹ 서브시스템 구체화 : 서브시스템의 기능 및 서브시스템 간의 상호작용을 위한 동작과 인터페이스를 정의함

❺ 검토 : 아키텍처가 설계 목표에 부합하는지, 요구사항이 잘 반영되었는지, 설계의 기본 원리를 만족하는지 등을 검토함

20.8

핵심 042 협약(Contract)에 의한 설계

2459909

컴포넌트를 설계할 때 클래스에 대한 여러 가정을 공유할 수 있도록 명세한 것으로, 소프트웨어 컴포넌트에 대한 정확한 인터페이스를 명세한다.

- 협약에 의한 설계 시 명세에 포함될 조건에는 선행 조건, 결과 조건, 불변 조건이 있다.

선행 조건 (Precondition)	오퍼레이션이 호출되기 전에 참이 되어야 할 조건
결과 조건 (Postcondition)	오퍼레이션이 수행된 후 만족되어야 할 조건
불변 조건 (Invariant)	오퍼레이션이 실행되는 동안 항상 만족되어야 할 조건

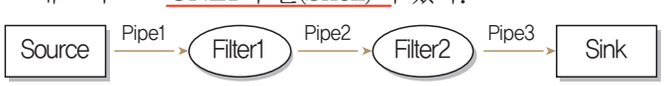
23.7, 22.7, 21.8, 21.5, 20.9

핵심 043 파이프-필터 패턴 (Pipe-Filter Pattern)

2402204

파이프-필터 패턴은 데이터 스트림 절차의 각 단계를 필터(Filter) 컴포넌트로 캡슐화하여 파이프(Pipe)를 통해 데이터를 전송하는 패턴이다.

- 필터 컴포넌트는 재사용성이 좋고, 추가가 쉬워 확장이 용이하다.
- 필터 컴포넌트들을 재배치하여 다양한 파이프라인을 구축하는 것이 가능하다.
- 파이프-필터 패턴은 데이터 변환, 버퍼링, 동기화 등에 주로 사용된다.
- 대표적으로 UNIX의 쉘(Shell)이 있다.



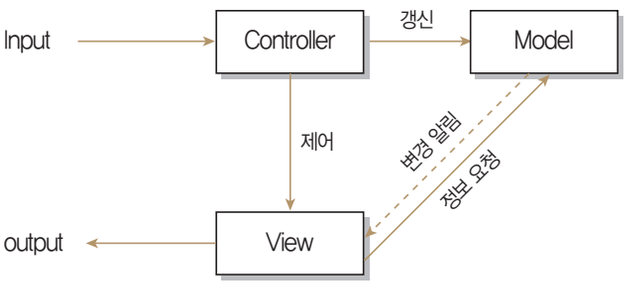
23.2, 22.4

핵심 044 모델-뷰-컨트롤러 패턴 (Model-View-Controller Pattern)

2402205

모델-뷰-컨트롤러 패턴은 서브시스템을 3개의 부분으로 구조화하는 패턴이며, 각 부분의 역할은 다음과 같다.

- 모델(Model) : 서브시스템의 핵심 기능과 데이터를 보관함
- 뷰(View) : 사용자에게 정보를 표시함
- 컨트롤러(Controller) : 사용자로부터 입력된 변경 요청을 처리하기 위해 모델에게 명령을 보냄



23.5, 21.8

핵심 045 기타 패턴

2402206

마스터-슬레이브 패턴 (Master-Slave Pattern)	• 마스터 컴포넌트에서 슬레이브 컴포넌트로 작업을 분할한 후, 슬레이브 컴포넌트에서 처리된 결과물을 다시 돌려받는 방식으로 작업을 수행하는 패턴 • 장애 허용 시스템과 병렬 컴퓨팅 시스템에서 주로 활용됨
브로커 패턴 (Broker Pattern)	• 사용자가 원하는 서비스와 특성을 브로커 컴포넌트에 요청하면 브로커 컴포넌트가 요청에 맞는 컴포넌트와 사용자를 연결해줌 • 분산 환경 시스템에서 주로 활용됨
피어-투-피어 패턴 (Peer-To-Peer Pattern)	피어(Peer)를 하나의 컴포넌트로 간주하며, 각 피어는 서비스를 호출하는 클라이언트가 될 수도, 서비스를 제공하는 서버가 될 수도 있는 패턴
이벤트-버스 패턴 (Event-Bus Pattern)	소스가 특정 채널에 이벤트 메시지를 발행(Publish)하면, 해당 채널을 구독(Subscribe)한 리스너들이 메시지를 받아 이벤트를 처리하는 방식
블랙보드 패턴 (Blackboard Pattern)	• 모든 컴포넌트들이 공유 데이터 저장소와 블랙보드 컴포넌트에 접근이 가능한 형태로, 컴포넌트들은 검색을 통해 블랙보드에서 원하는 데이터를 찾을 수 있음 • 음성 인식, 차량 식별, 신호 해석 등에 주로 활용됨
인터프리터 패턴 (Interpreter Pattern)	프로그램 코드의 각 라인을 수행하는 방법을 지정하고, 기호마다 클래스를 갖도록 구성됨

시험에 나오는 것만 공부한다!

23.7, 23.2, 22.7, 22.4, 21.5

핵심 046 객체(Object) = 인스턴스

2402302

객체는 데이터와 데이터를 처리하는 함수를 묶어 놓은(캡슐화한) 하나의 소프트웨어 모듈이다.

데이터	• 객체가 가지고 있는 정보로 속성이나 상태, 분류 등을 나타냄 • 속성(Attribute), 상태, 변수, 상수, 자료 구조라고도 함
함수	• 객체가 수행하는 기능으로 객체가 갖는 데이터(속성, 상태)를 처리하는 알고리즘 • 객체의 상태를 참조하거나 변경하는 수단이 되는 것으로 메소드(Method, 행위), 서비스(Service), 동작(Operation), 연산이라고도 함

- 객체의 특성
  - 객체는 독립적으로 식별 가능한 이름을 가지고 있다.
  - 객체가 가질 수 있는 조건을 상태(State)라고 하는데, 일반적으로 상태는 시간에 따라 변한다.
  - 객체와 객체는 상호 연관성에 의한 관계가 형성된다.
  - 객체가 반응할 수 있는 메시지(Message)의 집합을 행위라고 하며, 객체는 행위의 특징을 나타낼 수 있다.
  - 객체는 일정한 기억장소를 가지고 있다.
- 객체의 메소드는 다른 객체로부터 메시지를 받았을 때 정해진 기능을 수행한다.

23.5, 22.3, 21.5, 20.8, 20.6

핵심 047 클래스(Class)

2402303

클래스는 공통된 속성과 연산(행위)을 갖는 객체의 집합으로, 객체의 일반적인 타입(Type)을 의미한다.

- 클래스는 각각의 객체들이 갖는 속성과 연산을 정의하고 있는 틀이다.
- 클래스는 객체지향 프로그램에서 데이터를 추상화하는 단위이다.
- 클래스에 속한 각각의 객체를 인스턴스(Instance)라 하며, 클래스로부터 새로운 객체를 생성하는 것을 인스턴스화(Instantiation)라고 한다.

23.5, 22.7, 22.4, 21.5, 21.3, 20.9, 20.8

핵심 048 캡슐화(Encapsulation)

2402304

캡슐화는 데이터(속성)와 데이터를 처리하는 함수를 하나로 묶는 것을 의미한다.

- 캡슐화된 객체는 인터페이스를 제외한 세부 내용이 은폐(정보 은닉)되어 외부에서의 접근이 제한적이기 때문에 외부 모듈의 변경으로 인한 파급 효과가 적다.
- 캡슐화된 객체들은 재사용이 용이하다.
- 객체들 간의 메시지를 주고받을 때 상대 객체의 세부 내용은 알 필요가 없으므로 인터페이스가 단순해지고, 객체 간의 결합도가 낮아진다.

22.3, 21.8

핵심 049 상속(Inheritance)

2402305

상속은 이미 정의된 상위 클래스(부모 클래스)의 모든 속성과 연산을 하위 클래스(자식 클래스)가 물려받는 것이다.

- 상속을 이용하면 하위 클래스는 상위 클래스의 모든 속성과 연산을 자신의 클래스 내에서 다시 정의하지 않고서도 즉시 자신의 속성으로 사용할 수 있다.
- 하위 클래스는 상위 클래스로부터 상속받은 속성과 연산 외에 새로운 속성과 연산을 첨가하여 사용할 수 있다.

23.2, 22.4

핵심 050 다형성(Polymorphism)

2402306

다형성은 메시지에 의해 객체(클래스)가 연산을 수행하게 될 때 하나의 메시지에 대해 각각의 객체(클래스)가 가지고 있는 고유한 방법(특성)으로 응답할 수 있는 능력을 의미한다.

- 객체(클래스)들은 동일한 메소드명을 사용하며 같은 의미의 응답을 한다.
- 응용 프로그램 상에서 하나의 함수나 연산자가 두 개 이상의 서로 다른 클래스의 인스턴스들을 같은 클래스에 속한 인스턴스처럼 수행할 수 있도록 하는 것이다.

- 예1 '+' 연산자의 경우 숫자 클래스에서는 덧셈, 문자 클래스에서는 문자열의 연결 기능으로 사용된다.
- 예2 오버로딩(Overloading) 기능의 경우 메소드(Method)의 이름은 같지만 인수를 받는 자료형과 개수를 달리하여 여러 기능을 정의할 수 있다.
- 예3 오버라이딩(Overriding, 메소드 재정의) 기능의 경우 상위 클래스에서 정의한 메소드(Method)와 이름은 같지만 메소드 안의 실행 코드를 달리하여 자식 클래스에서 재정의해서 사용할 수 있다.

20.6

핵심 051 연관성(Relationship)

2402307

연관성은 두 개 이상의 객체(클래스)들이 상호 참조하는 관계를 말하며 종류는 다음과 같다.

종류	의미	특징
is member of	연관화 (Association)	2개 이상의 객체가 상호 관련되어 있음을 의미함
is instance of	분류화 (Classfication)	동일한 형의 특성을 갖는 객체들을 모아 구성하는 것
is part of	집단화 (Aggregation)	관련 있는 객체들을 묶어 하나의 상위 객체를 구성하는 것
is a	일반화 (Generalization)	공통적인 성질들로 추상화한 상위 객체를 구성하는 것
	특수화/상세화 (Specialization)	상위 객체를 구체화하여 하위 객체를 구성하는 것

21.3, 20.6

핵심 052 객체지향 분석의 방법론

2402402

객체지향 분석을 위한 여러 방법론이 제시되었으며 각 방법론은 다음과 같다.

- Rumbaugh(럼바우) 방법 : 가장 일반적으로 사용되는 방법으로 분석 활동을 객체 모델, 동적 모델, 기능 모델로 나누어 수행하는 방법
- Booch(부치) 방법 : 미시적(Micro) 개발 프로세스와 거시적(Macro) 개발 프로세스를 모두 사용하는 분석 방법으로, 클래스와 객체들을 분석 및 식별하고 클래스의 속성과 연산을 정의함
- Jacobson 방법 : Use Case를 강조하여 사용하는 분석 방법
- Coad와 Yourdon 방법 : E-R 다이어그램을 사용하여 객체의 행위를 모델링하며, 객체 식별, 구조 식별, 주제 정의, 속성과 인스턴스 연결 정의, 연산과 메시지 연결 정의 등의 과정으로 구성하는 기법
- Wirfs-Brock 방법 : 분석과 설계 간의 구분이 없고, 고객 명세서를 평가해서 설계 작업까지 연속적으로 수행하는 기법

23.7, 22.7, 22.3, 21.8, 21.5, 21.3, 20.9, 20.8, 20.6

핵심 053 럼바우(Rumbaugh)의 분석 기법

2402403

럼바우의 분석 기법은 모든 소프트웨어 구성 요소를 그래픽 표기법을 이용하여 모델링하는 기법으로, 객체 모델링 기법(OMT, Object-Modeling Technique)이라고도 한다.

- 분석 활동은 '객체 모델링 → 동적 모델링 → 기능 모델링' 순으로 통해 이루어진다.

객체 모델링 (Object Modeling)	정보 모델링이라고도 하며, 시스템에서 요구되는 객체를 찾아내어 속성과 연산 식별 및 객체들 간의 관계를 규정하여 객체 다이어그램으로 표시하는 것
동적 모델링 (Dynamic Modeling)	상태 다이어그램(상태도)을 이용하여 시간의 흐름에 따른 객체들 간의 제어 흐름, 상호 작용, 동작 순서 등의 동적인 행위를 표현하는 모델링
기능 모델링 (Functional Modeling)	자료 흐름도(DFD)를 이용하여 다수의 프로세스들 간의 자료 흐름을 중심으로 처리 과정을 표현한 모델링

23.5, 23.2, 22.7, 22.3, 20.9, 20.8

핵심 054 객체지향 설계 원칙

2402404

객체지향 설계 원칙은 시스템 변경이나 확장에 유연한 시스템을 설계하기 위해 지켜야 할 다섯 가지 원칙으로, 다섯 가지 원칙의 앞 글자를 따 SOLID 원칙이라고도 불린다.

단일 책임 원칙 (SRP, Single Responsibility Principle)	객체는 단 하나의 책임만 가져야 한다는 원칙
개방-폐쇄 원칙 (OCP, Open-Closed Principle)	기존의 코드를 변경하지 않고 기능을 추가할 수 있도록 설계해야 한다는 원칙
리스코프 치환 원칙 (LSP, Liskov Substitution Principle)	자식 클래스는 최소한 자신의 부모 클래스에서 가능한 행위는 수행할 수 있어야 한다는 설계 원칙
인터페이스 분리 원칙 (ISP, Interface Segregation Principle)	자신이 사용하지 않는 인터페이스와 의존 관계를 맺거나 영향을 받지 않아야 한다는 원칙
의존 역전 원칙 (DIP, Dependency Inversion Principle)	각 객체들 간의 의존 관계가 성립될 때, 추상성이 낮은 클래스보다 추상성이 높은 클래스와 의존 관계를 맺어야 한다는 원칙

23.7, 23.2, 22.7, 22.4, 21.5, 21.3, 20.9, 20.8, 20.6

2402502

핵심 055 결합도(Coupling)

결합도는 모듈 간에 상호 의존하는 정도 또는 두 모듈 사이의 연관 관계를 의미한다.

- 다양한 결합으로 모듈을 구성할 수 있으나 결합도가 약할수록 품질이 높고, 강할수록 품질이 낮다.
- 결합도가 강하면 시스템 구현 및 유지보수 작업이 어렵다.
- 결합도의 종류에는 자료 결합도, 스탬프 결합도, 제어 결합도, 외부 결합도, 공통 결합도, 내용 결합도가 있으며 결합도가 약함에서 강함순으로 정리하면 다음과 같다.

자료 결합도 (Data Coupling)	모듈 간의 인터페이스가 자료 요소로만 구성될 때의 결합도
스탬프(검인) 결합도 (Stamp Coupling)	모듈 간의 인터페이스로 배열이나 레코드 등의 자료 구조가 전달될 때의 결합도
제어 결합도 (Control Coupling)	어떤 모듈이 다른 모듈 내부의 논리적인 흐름을 제어하기 위해 제어 신호를 이용하여 통신하거나 제어 요소(Function Code, Switch, Tag, Flag)를 전달하는 결합도
외부 결합도 (External Coupling)	어떤 모듈에서 선언한 데이터(변수)를 외부의 다른 모듈에서 참조할 때의 결합도
공통(공유) 결합도 (Common Coupling)	공유되는 공통 데이터 영역을 여러 모듈이 사용할 때의 결합도
내용 결합도 (Content Coupling)	한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 직접 참조하거나 수정할 때의 결합도

23.5, 22.4, 21.8, 21.5, 21.3, 20.9, 20.8, 20.6

2402503

핵심 056 응집도(Cohesion)

응집도는 정보 은닉 개념을 확장한 것으로, 명령어나 호출문 등 모듈의 내부 요소들의 서로 관련되어 있는 정도, 즉 모듈이 독립적인 기능으로 정의되어 있는 정도를 의미한다.

- 다양한 기준으로 모듈을 구성할 수 있으나 응집도가 강할수록 품질이 높고, 약할수록 품질이 낮다.
- 응집도의 종류에는 기능적 응집도, 순차적 응집도, 교환(통신)적 응집도, 절차적 응집도, 시간적 응집도, 논리적 응집도, 우연적 응집도가 있으며 응집도가 강함에서 약함순으로 정리하면 다음과 같다.

기능적 응집도 (Functional Cohesion)	모듈 내부의 모든 기능 요소들이 단일 문제와 연관되어 수행될 경우의 응집도
순차적 응집도 (Sequential Cohesion)	모듈 내 하나의 활동으로부터 나온 출력 데이터를 그 다음 활동의 입력 데이터로 사용할 경우의 응집도
교환(통신)적 응집도 (Communication Cohesion)	동일한 입력과 출력을 사용하여 서로 다른 기능을 수행하는 구성 요소들이 모였을 경우의 응집도
절차적 응집도 (Procedural Cohesion)	모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성 요소들이 그 기능을 순차적으로 수행할 경우의 응집도
시간적 응집도 (Temporal Cohesion)	특정 시간에 처리되는 몇 개의 기능을 모아 하나의 모듈로 작성할 경우의 응집도
논리적 응집도 (Logical Cohesion)	유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들로 하나의 모듈이 형성되는 경우의 응집도
우연적 응집도 (Coincidental Cohesion)	모듈 내부의 각 구성 요소들이 서로 관련 없는 요소로만 구성된 경우의 응집도

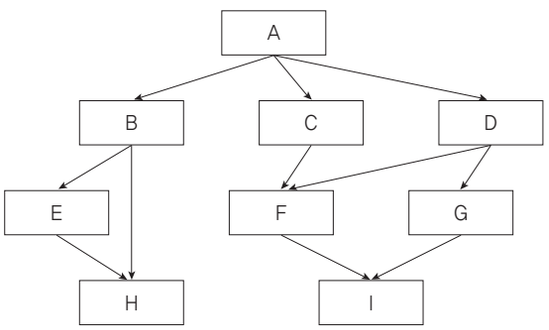
22.7, 21.3

핵심 057 팬인(Fan-In) / 팬아웃(Fan-Out)

2402504

- 팬인은 어떤 모듈을 제어(호출)하는 모듈의 수를 나타낸다.
- 팬아웃은 어떤 모듈에 의해 제어(호출)되는 모듈의 수를 나타낸다.

예제 다음의 시스템 구조도에서 각 모듈의 팬인(Fan-In)과 팬아웃(Fan-Out)을 구하시오.



해설

팬인(Fan-In) : A는 0, B · C · D · E · G는 1, F · H · I는 2

팬아웃(Fan-Out) : H · I는 0, C · E · F · G는 1, B · D는 2, A는 3

22.3, 20.9

핵심 058 N-S 차트(Nassi-Schneiderman Chart)

2402531

N-S 차트는 논리의 기술에 중점을 둔 도형을 이용한 표현 방법으로 박스 다이어그램, Chapin Chart라고도 한다.

- 연속, 선택 및 다중 선택, 반복 등의 제어 논리 구조를 표현한다.
- GOTO나 화살표를 사용하지 않는다.
- 조건이 복합되어 있는 곳의 처리를 시각적으로 명확히 식별하는 데 적합하다.
- 선택과 반복 구조를 시각적으로 표현한다.
- 이해하기 쉽고, 코드 변환이 용이하다.
- 읽기는 쉽지만 작성하기가 어려우며, 임의로 제어를 전이하는 것이 불가능하다.
- 총체적인 구조 표현과 인터페이스를 나타내기가 어렵다.
- 단일 입구와 단일 출구로 표현한다.

20.6

핵심 059 공통 모듈의 개요

2402601

공통 모듈은 여러 프로그램에서 공통적으로 사용할 수 있는 모듈을 의미한다.

- 자주 사용되는 계산식이나 매번 필요한 사용자 인증과 같은 기능들이 공통 모듈로 구성될 수 있다.
- 모듈의 재사용성 확보와 중복 개발 회피를 위해 설계 과정에서 공통 부분을 식별하고 명세를 작성할 필요가 있다.
- 공통 모듈을 구현할 때는 다른 개발자들이 해당 기능을 명확히 이해할 수 있도록 다음의 명세 기법을 준수해야 한다.

정확성 (Correctness)	시스템 구현 시 해당 기능이 필요하다는 것을 알 수 있도록 정확히 작성함
명확성(Clarity)	해당 기능을 이해할 때 중의적으로 해석되지 않도록 명확하게 작성함
완전성 (Completeness)	시스템 구현을 위해 필요한 모든 것을 기술함
일관성 (Consistency)	공통 기능들 간 상호 충돌이 발생하지 않도록 작성함
추적성 (Traceability)	기능에 대한 요구사항의 출처, 관련 시스템 등의 관계를 파악할 수 있도록 작성함

22.4, 21.3, 20.9

핵심 060 재사용(Reuse)

2402602

재사용은 비용과 개발 시간을 절약하기 위해 이미 개발된 기능들을 파악하고 재구성하여 새로운 시스템 또는 기능 개발에 사용하기 적합하도록 최적화 시키는 작업이다.

- 재사용을 위해서는 누구나 이해할 수 있고 사용이 가능하도록 사용법을 공개해야 한다.
- 재사용되는 대상은 외부 모듈과의 결합도는 낮고, 응집도는 높아야 한다.
- 재사용 규모에 따른 분류

함수와 객체	클래스나 메소드 단위의 소스 코드를 재사용함
컴포넌트	• 독립적인 업무 또는 기능을 수행하는 실행 코드 기반으로 작성된 모듈임 • 컴포넌트 자체에 대한 수정 없이 인터페이스를 통해 통신하는 방식으로 재사용함
애플리케이션	공통된 기능들을 제공하는 애플리케이션을 공유하는 방식으로 재사용함

22.3, 21.3, 20.9, 20.8

2402603

핵심 061 효과적인 모듈 설계 방안

- 결합도는 줄이고 응집도는 높여서 모듈의 독립성과 재사용성을 높인다.
- 모듈의 제어 영역 안에서 그 모듈의 영향 영역을 유지시킨다.
- 복잡도와 중복성을 줄이고 일관성을 유지시킨다.
- 모듈의 기능은 예측이 가능해야 하며 지나치게 제한적이어서는 안 된다.
- 유지보수가 용이해야 한다.
- 모듈 크기는 시스템의 전반적인 기능과 구조를 이해하기 쉬운 크기로 분해한다.
- 효과적인 제어를 위해 모듈 간의 계층적 관계를 정의하는 자료가 제시되어야 한다.

20.8

2402701

핵심 062 코드(Code)의 개요

코드는 컴퓨터를 이용하여 자료를 처리하는 과정에서 분류 · 조합 및 집계를 용이하게 하고, 특정 자료의 추출을 쉽게 하기 위해서 사용하는 기호이다.

- 코드는 정보를 신속·정확·명료하게 전달할 수 있게 한다.
- 일반적인 코드의 예로 주민등록번호, 학번, 전화번호 등이 있다.
- 코드의 주요 기능에는 식별 기능, 분류 기능, 배열 기능, 표준화 기능, 간소화 기능이 있다.

식별 기능	데이터 간의 성격에 따라 구분이 가능함
분류 기능	특정 기준이나 동일한 유형에 해당하는 데이터를 그룹화 할 수 있음
배열 기능	의미를 부여하여 나열할 수 있음
표준화 기능	다양한 데이터를 기준에 맞추어 표현할 수 있음
간소화 기능	복잡한 데이터를 간소화할 수 있음

23.7, 23.2, 20.9, 20.6

2402702

핵심 063 코드의 종류

코드의 종류에는 다음과 같은 것들이 있다.

순차 코드 (Sequence Code)	자료의 발생 순서, 크기 순서 등 일정 기준에 따라서 최초의 자료부터 차례로 일련번호를 부여하는 방법으로, 순서 코드 또는 일련번호 코드라고도 함 예 1, 2, 3, 4, …
블록 코드 (Block Code)	코드화 대상 항목 중에서 공통성이 있는 것끼리 블록으로 구분하고, 각 블록 내에서 일련번호를 부여하는 방법으로, 구분 코드라고도 함 예 1001~1100 : 총무부, 1101~1200 : 영업부
10진 코드 (Decimal Code)	코드화 대상 항목을 0~9까지 10진 분할하고, 다시 그 각각에 대하여 10진 분할하는 방법을 필요한 만큼 반복하는 방법으로, 도서 분류식 코드라고도 함 예 1000 : 공학, 1100 : 소프트웨어 공학, 1110 : 소프트웨어 설계
그룹 분류 코드 (Group Classification Code)	코드화 대상 항목을 일정 기준에 따라 대분류, 중분류, 소분류 등으로 구분하고, 각 그룹 안에서 일련번호를 부여하는 방법 예 1-01-001 : 본사-총무부-인사계, 2-01-001 : 지사-총무부-인사계
연상 코드 (Mnemonic Code)	코드화 대상 항목의 명칭이나 약호와 관계있는 숫자나 문자, 기호를 이용하여 코드를 부여하는 방법 예 TV-40 : 40인치 TV, L-15-220 : 15W 220V의 램프
표의 숫자 코드 (Significant Digit Code)	코드화 대상 항목의 성질, 즉 길이, 넓이, 부피, 지름, 높이 등의 물리적 수치를 그대로 코드에 적용시키는 방법으로, 유효 숫자 코드라고도 함 예 120-720-1500 : 두께×폭×길이가 120×720×1500인 강판
합성 코드 (Combined Code)	필요한 기능을 하나의 코드로 수행하기 어려운 경우 2개 이상의 코드를 조합하여 만드는 방법 예 연상 코드 + 순차 코드 KE-711 : 대한항공 711기, AC-253 : 에어캐나다 253기

23.5, 22.3, 20.9, 20.8

핵심 064 디자인 패턴 (Design Pattern)의 개요

2402801

디자인 패턴은 각 모듈의 세분화된 역할이나 모듈들 간의 인터페이스와 같은 코드를 작성하는 수준의 세부적인 구현 방안을 설계할 때 참조할 수 있는 전형적인 해결 방식 또는 예제를 의미한다.

- 디자인 패턴은 문제 및 배경, 실제 적용된 사례, 재사용이 가능한 샘플 코드 등으로 구성되어 있다.
- '바퀴를 다시 발명하지 마라(Don't reinvent the wheel)'라는 말과 같이, 개발 과정 중에 문제가 발생하면 새로 해결책을 구상하는 것보다 문제에 해당하는 디자인 패턴을 참고하여 적용하는 것이 더 효율적이다.
- GoF의 디자인 패턴은 유형에 따라 생성 패턴 5개, 구조 패턴 7개, 행위 패턴 11개 총 23개의 패턴으로 구성된다.

21.3, 20.9

핵심 065 디자인 패턴 사용의 장 · 단점

2402802

- 범용적인 코딩 스타일로 인해 구조 파악이 용이하다.
- 객체지향 설계 및 구현의 생산성을 높이는 데 적합하다.
- 검증된 구조의 재사용을 통해 개발 시간과 비용이 절약된다.
- 초기 투자 비용이 부담될 수 있다.
- 개발자 간의 원활한 의사소통이 가능하다.
- 설계 변경 요청에 대한 유연한 대처가 가능하다.
- 객체지향을 기반으로 한 설계와 구현을 다루므로 다른 기반의 애플리케이션 개발에는 적합하지 않다.

23.7, 23.5, 23.2, 22.7, 22.3, 21.8, 21.5, 21.3, 20.8

핵심 066 생성 패턴 (Creational Pattern)

2402803

생성 패턴은 객체의 생성과 관련된 패턴으로 총 5개의 패턴이 있다.

- 생성 패턴은 객체의 생성과 참조 과정을 캡슐화 하여 객체가 생성되거나 변경되어도 프로그램의 구조에 영향을 크게 받지 않도록 하여 프로그램에 유연성을 더해준다.

추상 팩토리 (Abstract Factory)	• 구체적인 클래스에 의존하지 않고, 인터페이스를 통해 서로 연관 · 의존하는 객체들의 그룹으로 생성하여 추상적으로 표현함 • 연관된 서브 클래스를 묶어 한 번에 교체하는 것이 가능함
빌더 (Builder)	• 작게 분리된 인스턴스를 건축 하듯이 조합하여 객체를 생성함 • 객체의 생성 과정과 표현 방법을 분리하고 있어, 동일한 객체 생성에서도 서로 다른 결과를 만들어 낼 수 있음
팩토리 메소드 (Factory Method)	• 객체 생성을 서브 클래스에서 처리하도록 분리하여 캡슐화한 패턴 • 상위 클래스에서 인터페이스만 정의하고 실제 생성은 서브 클래스가 담당함 • 가상 생성자(Virtual Constructor) 패턴이라고도 함
프로토타입 (Prototype)	• 원본 객체를 복제하는 방법으로 객체를 생성하는 패턴 • 일반적인 방법으로 객체를 생성하며, 비용이 큰 경우 주로 이용함
싱글톤 (Singleton)	• 하나의 객체를 생성하면 생성된 객체를 어디서든 참조할 수 있지만, 여러 프로세스가 동시에 참조할 수는 없음 • 클래스 내에서 인스턴스가 하나뿐임을 보장하며, 불필요한 메모리 낭비를 최소화 할 수 있음

생성자가 여러 차례 호출되더라도 실제로 생성되는 객체는 하나이고 최초 생성 이후에 호출된 생성자는 최초의 생성자가 생성한 객체를 리턴한다.

23.7, 23.2, 22.4, 21.5

핵심 067 구조 패턴 (Structural Pattern)

2402804

구조 패턴은 클래스나 객체들을 조합하여 더 큰 구조로 만들 수 있게 해주는 패턴으로 총 7개의 패턴이 있다.

- 구조 패턴은 구조가 복잡한 시스템을 개발하기 쉽게 도와준다.

어댑터 (Adapter)	• 호환성이 없는 클래스들의 인터페이스를 다른 클래스가 이용할 수 있도록 변환해주는 패턴 • 기존의 클래스를 이용하고 싶지만 인터페이스가 일치하지 않을 때 이용함
브리지 (Bridge)	• 구현부에서 추상층을 분리하여, 서로가 독립적으로 확장할 수 있도록 구성한 패턴 • 기능과 구현을 두 개의 별도 클래스로 구현함
컴포지트 (Composite)	• 여러 객체를 가진 복합 객체와 단일 객체를 구분 없이 다루고자 할 때 사용하는 패턴 • 객체들을 트리 구조로 구성하여 디렉터리 안에 디렉터리가 있듯이 복합 객체 안에 복합 객체가 포함되는 구조를 구현할 수 있음

데코레이터 (Decorator)	• 객체 간의 결합을 통해 능동적으로 기능들을 확장할 수 있는 패턴 • 임의의 객체에 부가적인 기능을 추가하기 위해 다른 객체들을 덧붙이는 방식으로 구현함
퍼싸드 (Facade)	• 복잡한 서브 클래스들을 피해 더 상위에 인터페이스를 구성함으로써 서브 클래스들의 기능을 간편하게 사용할 수 있도록 하는 패턴 • 서브 클래스들 사이의 통합 인터페이스를 제공하는 Wrapper 객체가 필요함
플라이웨이트 (Flyweight)	• 인스턴스가 필요할 때마다 매번 생성하는 것이 아니고 가능한 한 공유해서 사용함으로써 메모리를 절약하는 패턴 • 다수의 유사 객체를 생성하거나 조작할 때 유용하게 사용할 수 있음
프록시 (Proxy)	• 접근이 어려운 객체와 여기에 연결하려는 객체 사이에서 인터페이스 역할을 수행하는 패턴 • 네트워크 연결, 메모리의 대용량 객체로의 접근 등에 주로 이용함

복잡한 시스템을 개발하기 쉽도록 클래스나 객체를 조합하는 패턴에 속함
"대리"라는 이름으로도 불림
비유 - 객체간의 복잡한 관계를 단순하게 정리
외부 - 객체의 세부인 내용을 숨기는 역할

23.5, 23.2, 21.8, 21.5, 20.8, 20.6

핵심 068 행위 패턴 (Behavioral Pattern)

2402805

행위 패턴은 클래스나 객체들이 서로 상호작용하는 방법이나 책임 분배 방법을 정의하는 패턴으로 총 11개의 패턴이 있다.

- 행위 패턴은 하나의 객체로 수행할 수 없는 작업을 여러 객체로 분배하면서 결합도를 최소화 할 수 있도록 도와준다.

책임 연쇄 (Chain of Responsi-bility)	• 요청을 처리할 수 있는 객체가 둘 이상 존재하여 한 객체가 처리하지 못하면 다음 객체로 넘어가는 형태의 패턴 • 요청을 처리할 수 있는 각 객체들이 고리(Chain)로 묶여 있어 요청이 해결될 때까지 고리를 따라 책임이 넘어감
커맨드 (Command)	• 요청을 객체의 형태로 캡슐화하여 재이용하거나 취소할 수 있도록 요청에 필요한 정보를 저장하거나 로그에 남기는 패턴 • 요청에 사용되는 각종 명령어들을 추상 클래스와 구체 클래스로 분리하여 단순화함
인터프리터 (Interpreter)	• 언어에 문법 표현을 정의하는 패턴 • SQL이나 통신 프로토콜과 같은 것을 개발할 때 사용함
반복자 (Iterator)	• 자료 구조와 같이 접근이 잦은 객체에 대해 동일한 인터페이스를 사용하도록 하는 패턴 • 내부 표현 방법의 노출 없이 순차적인 접근이 가능함
중재자 (Mediator)	• 수많은 객체들 간의 복잡한 상호작용(Interface)을 캡슐화하여 객체로 정의하는 패턴 • 객체 사이의 의존성을 줄여 결합도를 감소시킬 수 있음
메멘토 (Memento)	• 특정 시점에서의 객체 내부 상태를 객체화함으로써 이후 요청에 따라 객체를 해당 시점의 상태로 돌릴 수 있는 기능을 제공하는 패턴 • Ctrl+Z와 같은 되돌리기 기능을 개발할 때 주로 이용함
옵서버 (Observer)	1:다 대응 • 한 객체의 상태가 변화하면 객체에 상속되어 있는 다른 객체들에게 변화된 상태를 전달하는 패턴 • 주로 분산된 시스템 간에 이벤트를 생성 · 발행(Publish)하고, 이를 수신(Subscribe)해야 할 때 이용함
상태 (State)	• 객체의 상태에 따라 동일한 동작을 다르게 처리해야 할 때 사용하는 패턴 • 객체 상태를 캡슐화하고 이를 참조하는 방식으로 처리함
전략 (Strategy)	• 동일한 계열의 알고리즘들을 개별적으로 캡슐화하여 상호 교환할 수 있게 정의하는 패턴 • 클라이언트는 독립적으로 원하는 알고리즘을 선택하여 사용할 수 있으며, 클라이언트에 영향 없이 알고리즘의 변경이 가능함
템플릿 메소드 (Template Method)	• 상위 클래스에서 골격을 정의하고, 하위 클래스에서 세부 처리를 구체화하는 구조의 패턴 • 유사한 서브 클래스를 묶어 공통된 내용을 상위 클래스에서 정의함으로써 코드의 양을 줄이고 유지보수를 용이하게 해줌
방문자 (Visitor)	• 각 클래스들의 데이터 구조에서 처리 기능을 분리하여 별도의 클래스로 구성하는 패턴 • 분리된 처리 기능은 각 클래스를 방문(Visit)하여 수행함

- 호스트 객체의 내부 상태에 접근할 수 있는 방법을 제공하며
  호스트 객체에 연산을 추가할 수 있도록 한다.
- 함성 규모의 원소들과 상호 작용하는데 사용
  기존 코드를 변경하지 않고 새로운 기능을 추가

22.7, 22.4, 20.8, 20.6 2403005

핵심 069 요구사항 검증 방법

- 요구사항 검토(Requirements Review) : 요구사항 명세서의 오류 확인 및 표준 준수 여부 등의 결함 여부를 검토 담당자들이 수작업으로 분석하는 방법으로, 동료검토, 워크스루, 인스펙션 등이 있음

동료검토 (Peer Review)	요구사항 명세서 작성자가 명세서 내용을 직접 설명하고 동료들이 이를 들으면서 결함을 발견하는 형태의 검토 방법
워크스루 (Walk Through)	검토 회의 전에 요구사항 명세서를 미리 배포하여 사전 검토한 후에 짧은 검토 회의를 통해 결함을 발견하는 형태의 검토 방법
인스펙션 (Inspection)	요구사항 명세서 작성자를 제외한 다른 검토 전문가들이 요구사항 명세서를 확인하면서 결함을 발견하는 형태의 검토 방법

- 프로토타이핑(Prototyping) : 사용자의 요구사항을 정확히 파악하기 위해 실제 개발될 소프트웨어에 대한 견본품(Prototype)을 만들어 최종 결과물을 예측함
- 테스트 설계 : 요구사항은 테스트할 수 있도록 작성되어야 하며, 이를 위해 테스트 케이스(Test Case)를 생성하여 이후에 요구사항이 현실적으로 테스트 가능한지를 검토함
- CASE 도구 활용 : 일관성 분석(Consistency Analysis)을 통해 요구사항 변경사항의 추적 및 분석, 관리하고, 표준 준수 여부를 확인함

21.3 2403302

핵심 070 시스템 연계 기술

DB Link	DB에서 제공하는 DB Link 객체를 이용하는 방식
API/ Open API	송신 시스템의 데이터베이스(DB)에서 데이터를 읽어와 제공하는 애플리케이션 프로그래밍 인터페이스 프로그램
연계 솔루션	EAI 서버와 송 · 수신 시스템에 설치되는 클라이언트(Client)를 이용하는 방식
Socket	서버는 통신을 위한 소켓(Socket)을 생성하여 포트를 할당하고 클라이언트의 통신 요청 시 클라이언트와 연결하여 통신하는 네트워크 기술
Web Service	웹 서비스(Web Service)에서 WSDL과 UDDI, SOAP 프로토콜을 이용하여 연계하는 서비스

21.5 2459910

핵심 071 연계 매커니즘 구성요소

- 송신 시스템 : 연계 프로그램으로부터 생성된 데이터를 전송 형식에 맞게 인터페이스 테이블이나 파일(xml, csv, text 등)로 변환한 후 송신하는 시스템
- 수신 시스템 : 수신한 인터페이스 테이블이나 파일을 연계 프로그램에서 처리할 수 있는 형식으로 변환한 후 연계 프로그램에 반영하는 시스템
- 연계 서버 : 송 · 수신 시스템 사이에 위치하여 데이터의 송 · 수신 현황을 모니터링하는 역할을 수행함

23.7, 23.5, 23.2, 22.7, 22.4, 21.8, 21.3, 20.9, 20.8, 20.6 2459954

핵심 072 미들웨어(Middleware)

미들웨어는 미들(Middle)과 소프트웨어(Software)의 합성어로, 운영체제와 응용 프로그램, 또는 서버와 클라이언트 사이에서 다양한 서비스를 제공하는 소프트웨어이다.

DB (DataBase)	• DB는 데이터베이스 벤더에서 제공하는 클라이언트에서 원격의 데이터베이스와 연결하기 위한 미들웨어 • DB를 사용하여 시스템을 구축하는 경우 보통 2-Tier 아키텍처라고 함
RPC (Remote Procedure Call)	RPC(원격 프로시저 호출)는 응용 프로그램의 프로시저를 사용하여 원격 프로시저를 마치 로컬 프로시저처럼 호출하는 방식의 미들웨어
MOM (Message Oriented Middleware)	• MOM(메시지 지향 미들웨어)은 메시지 기반의 비동기형 메시지를 전달하는 방식의 미들웨어 • 온라인 업무보다는 이기종 분산 데이터 시스템의 데이터 동기를 위해 많이 사용됨
TP-Monitor (Transaction Processing Monitor)	• TP-Monitor(트랜잭션 처리 모니터)는 항공기나 철도 예약 업무 등과 같은 온라인 트랜잭션 업무에서 트랜잭션을 처리 및 감시하는 미들웨어 • 사용자 수가 증가해도 빠른 응답 속도를 유지해야 하는 업무에 주로 사용됨
ORB(Object Request Broker)	• ORB(객체 요청 브로커)는 객체 지향 미들웨어로 코바(CORBA) 표준 스펙을 구현한 미들웨어 • 최근에는 TP-Monitor의 장점인 트랜잭션 처리와 모니터링 등을 추가로 구현한 제품도 있음
WAS(Web Application Server)	• WAS(웹 애플리케이션 서버)는 정적인 콘텐츠를 처리하는 웹 서버와 달리 사용자의 요구에 따라 변하는 동적인 콘텐츠를 처리하기 위해 사용되는 미들웨어 • 클라이언트/서버 환경보다는 웹 환경을 구현하기 위한 미들웨어