1과목 〈소프트웨어 설계〉

1장

섹션 1 소프트웨어 생명 주기

소프트웨어 생명 주기(소프트웨어 프로세스 모형, 소프트웨어 공학 패러다임) : 소프트웨어를 개발하기 위해 정의하고 운용, 유지보수 등의 과정을 각 단계별로 나눈 것. 즉, 소프트웨어의 품질이 항상 최상이 되도록 관리하기 위해 단계로 나눈 것.

종류

1. 폭포수 모형

: - 소프트웨어 개발에서 이전 단계로 돌아갈 수 없다는 전제하에 각 단계를 매듭짓고 그 결과를 철저히 검토하여 승인 후 다음 단계로 진행

- 전통적이며 고전적인 생명 주기 모형. 성공적 사례 많음
- 선형적이며, 매뉴얼 작성 필요.
- 타당성 검토-〉 계획 -〉 요구분석 -〉 설계 -〉 구현 -〉 시험 -〉 유지보수

2. 프로토타입 모형

: - 사용자의 요구사항을 정확히 파악하기 위해 실제 개발될 소프트웨어에 대한 견본(시제)품을 만들어 최종 결과물을 예측하는 모형

- 시스템과 인터페이스에 중점을 두어 개발
- 개발이 완료된 시점에서 오류가 발견되는 폭포수 모형의 단점을 보완한 모형
- 요구 수집 -〉 빠른 설계 -〉 프로토 타입 구축 -〉 고객평가 -〉 프로토 타입 조정 -〉 구현

3. 나선형 모형 (Spiral Model, 점진적 모형)

: - 보헴이 제안한 모형으로, 폭포수·프로토타입 모형의 장점에 위험 분석 기능을 추가한 모형

- 소프트웨어를 개발하면서 발생할 수 있는 위험을 관리하고 최소화 하는 것이 목적
- 계획 수립 -〉 위험 분석 -〉 개발 및 검증 -〉 평가 ---〉 다시 반복
- 계속 반복하므로 요구사항을 추가하고, 정밀하며, 유지보수가 필요없음

4. 애자일 모형(Agile Model(민첩한, 기민한)

: - 고객의 요구사항에 대응하기위해 일정 주기를 반복하는 개발 과정

- 어떤 특정 개발 방법론이 아닌 좋은 것을 빠르고 낭비 없게 만들기 위해 고객과에 소통에 초점을 맞춘 방법론.

- 기업 전반에서 사용
- 스프린트 또는 이터레이션이라는 짧은 개발 주기를 반복, 한 주기마다 고객 평가
- 소규모 프로젝트 , 고도로 숙련된 개발자. 급변하는 요구사항에 적합.
- 요규사항이 급변하므로 돌아갈 수 없는 폭포수와 대조적

애자일 선언 (Agile Manifesto)

: 애자일 개발의 핵심과치와 실행 지침이 있음

핵심가치 :

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.

폭포수 모형과 애자일 모형의 비교

구분	폭포수 모형	애자일
새로운 요구사항 반영	어려움	지속적으로 반영
고객과의 의사소통	적음	지속적임
테스트	마지막에 모든 기능 테스트	반복되는 일정 주기가 끝날때 마다 테스트
개발 중심	계획, 문서(매뉴얼)	고객

섹션 2 스크럽(Scrum) 기법

스크럽의 개요

: 럭비에서 반칙으로 경기 중단시 가운데에서 공을 두고 서로 상대를 밀치기 위해 서로 대치해 있는 대형에서 유래. 즉, 팀이 중심이 되어 개발의 효율성을 높인다는 의미가 내포

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

1. 제품 책임자(PO; Product Owner)
 - 이해 관계자들 중 개발될 제품의 이해도가 가장 높고 , 오ㅛ구사항을 책임지고 결정하는 사람. 주로 개발 의뢰자나 사용자가 담당.
 - 요구사항을 종합하고 우선순위 (백로그)를 지정 및 갱신
2. 스크럼 마스터(SM; Scrum Master)
 - 스크럼 팀이 스크럼을 잘 수 행할 수 있도록 객관적인 시각에서 조언을 해주는 가이드 역할을 수행. 팀원통제 X
 - 일일 스크럼 회의 주관하여 진행 상태 점검, 개발 과정에서 발생된 장애 요소를 공론화 하여 처리.
3. 개발팀(DT; Development Team)
 - 제품 책임자와 스크럼 마스터를 제외한 모든 팀원으로, 개발자가 아니더라도 디자이너, 테스터등 제품 개발을 위해 참여하는 모든 사람이 대상
 - 보통 최대 인원은 7~8명이 적당.

스크럼 개발 프로세스
 1. 제품 백로그
  : 제품 개발에 필요한 모든 요구사항 수집 및 우선순위에 따라 지정.
  : 지속적 업데이트. 제품 백로그에 기록된 스토리 기반으로 릴리즈 계획 수립
 2. 스프린트 계획 회의
  : 제품 백로그 중 이번 스프린트 에서 수행할 작업을 대상으로 단기 일정 수립
  : 각 요구사항을 작업 단위로 분할 후 각 개별자가 수행할 목록인 스프린트 백로그 작성
 3. 스프린트 (Sprint)
  : 실제 개발 진행. 보통 2주~4주
  : 스프린트 백로그에서 작성된 태스크 마다의 속도를 추정한 후 개발자에게 할당
  : 진행 상황에 따라 To Do/ In Progress / Done 으로 나뉨
 4. 일일 스크럼 회의
  : 모든 팀원이 매일 짧은 시간동안 진행상황 점검
  : 보통 서서 진행. 남은 작업 시간은 소멸 차트(Burn-down Chart)에 표시
  : 스크럼 마스터가 발견된 장애요소를 해결하도록 도움
 5. 스프린트 검토 회의
  : 부분 또는 전체 완성 제품이 요구사항에 잘 부합하는지 사용자 앞에서 테스팅
  : 제품 책임자는 개선할 사항에 대한 피드백 정리 후 다음 스프린트에 반영되도록 제품 백로그 업데이트
 6. 스프린트 회고(Sprint Retrospective)
  : 스프린트 주기를 되돌아 보면서 잘 진행 되었는지, 개선할 점이 없는지 확인 및 기록.

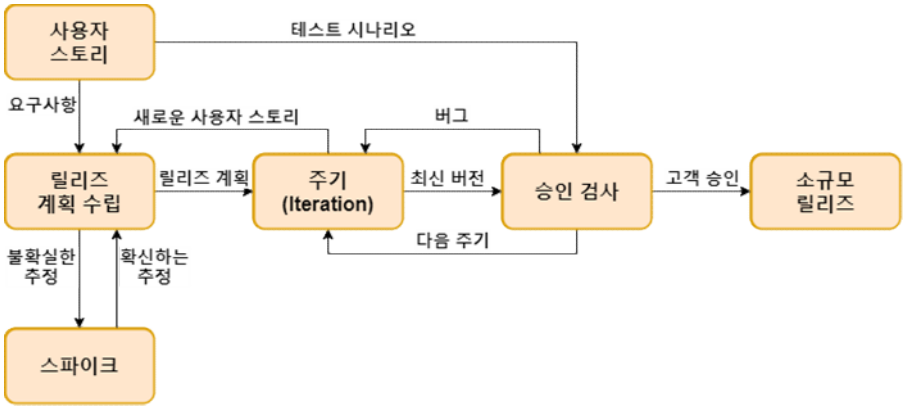
섹션 3 XP(eXtreme Programming) 기법

XP(eXtreme Programming)는 수시로 발생하는 고객의 요구사항에 유연하게 대응하기 위해 고객의 참여와 개발 과정의 반복을 극대화하여 개발 생산성을 향상시키는 방법.
: 기존의 방법론에 비해 실용성을 강조
: 짧고 반복적인 개발 주기, 단순한 설계, 고객의 적극적인 참여를 통해 소프트웨어를 빠르게 개발하는 것을 목적
: 릴리즈의 기간을 짧게 해 요구사항 반영에 대한 가시성을 높임
: 릴리즈 테스트마다 고객 참여로 동작을 확인
: 비교적 소규모 인원 개발 프로젝트에 효과적.

XP의 5가지 핵심가치

의사소통, 단순성, 용기, 존중, 피드백

XP개발 프로세스



XP의 주요 실천 방법(Practice), (기본원리)

실천방법	내용
Pair Programming (짝 프로그래밍)	두 사람이 하나의 작업 프로그래밍을 수행함으로써 개발에 대한 책임을 공동으로 나눠 갖는 환경 조성
Collective Ownership (코드 공동 소유)	개발 코드에 대한 권한과 책임을 공동으로 소유
Test-Driven Dev. (테스트 주도 개발)	· 개발자가 실제 코드를 작성하기 전에 테스트 케이스를 먼저 작성하므로 자신이 무엇을 해야할지 정확히 파악 · 테스트가 지속적으로 진행될 수 있도록 자동화된 테스팅 도구(구조,프레임워크)를 사용
Whole Team (전체 팀)	개발에 참여하는 모든 구성원(고객 포함)들은 자신이 역할이 있고, 그 역할에 책임을 지님.
Continuous Integration (계속적인 통합)	모듈 단위로 나눠서 개발된 코드들은 하나의 작업이 마무리될 때마다 지속적으로 통합
Desing Improvement (디자인 개선)// Refactoring(리팩토링)	프로그램의 기능의 번경 없이 단순화, 유연성 강화등을 통해 시스템을 재구성
Small Releases (소규모 릴리즈)	릴리즈 기간을 짧게 반복함으로써 고객의 요구 변화에 신속히 대응

섹션 4 현행 시스템 파악

현행 시스템 파악 절차

: 새로 개발하려는 시스템의 개발 범위를 명확히 설정하기 위해 현행 시스템의 구성과 제공 기능, 시스템 간의 전달 정보, 사용되는 기술 요소, 소프트웨어, 하드웨어, 그리고 네트워크의 구성등을 파악한다.

1단계	· 시스템 구성 파악 · 시스템 기능 파악 · 시스템 인터페이스 파악
2단계	· 아키텍처 구성 파악 · 소프트웨어 구성 파악
3단계	· 하드웨어 구성 파악 · 네트워크 구성 파악

1. 시스템 구성 파악

: 현행 시스템의 구성은 조직의 주요 업무를 담당하는 기간 업무와 이를 지원하는 지원 업무로 구분하여 기술.

2. 시스템 기능 파악

: 현행 시스템의 기능은 단위 업무 시스템이 현재 제공하는 기능들을 주요 기능과 하부기능, 세부 기능으로 구분하여 계층형(level)로 표시.

3. 시스템 인터페이스 파악

: 현행 시스템의 인터페이스에는 단위 업무 시스템 간에 주고받는 데이터의 종류, 형식, 프로토콜, 연계 유형, 주기 등을 명시

: 데이터를 어떤 형식으로 주고받는지 통신 규역은 무엇인지 연계 유형이 뭔지

4.아키텍처 구성 파악

: 현행 시스템의 아키텍처 구성은 기간 업무 수행에 어떠한 기술 요소들이 사용되는지 최상위 수준에서 계층별로 표현한 아키텍처 구성도로 작성한다.

: 아키텍처가 단위 업무 시스템별로 다른 경우에는 가장 핵심이 되는 기간업무 처리 시스템을 기준으로 표현

5. 소프트웨어 구성 파악

: 소프트웨어 구성에는 단위 업무시스템별로 업무 처리를 위해 설치되어 있는 소프트웨어의 제품명 , 용도, 라이선스 적용 방식, 라이선스 수 등을 명시.

:이는 시스템 구축비용을 확인에 필요

6.하드웨어 구성 파악

: 단위 업무 시스템들이 운용되는 서버의 주요 사양과 수량, 그리고 **이중화**의 적용 여부를 명시한다.

7. 네트워크 구성 파악

: 업무 시스템들의 네트워크 구성을 파악할 수 있도록 서버의 위치, 서버 간의 네트워크 연결 방식을 네트워크 구성도로 작성.

: 장애 시 복구하기 위한 용도 및 물리적인 위치 관계 파악에 용이

섹션 5 개발 기술 환경 파악

개발 기술 환경이란 개발하고자 하는 소프트웨어와 관련된 운영체제, 데이터베이스 관리 시스템, 미들웨어 등을 선정할 때 고려해야 할 사항 및 유의사항 제시

1. 운영 체제 관련 요구사항 식별 시 고려사항
 : 가용성, 성능, 기술 지원, 주변 기기, 구축 비용
2. 데이터베이스 관리 시스템(DBMS)
 : 데이터베이스 관리 및 사용자에 요구에 따른 정보 생성해주는 소프트웨어
 : 가용성, 성능, 기술지원, 상호 호환성, 구축 비용
 : Redis, MySQL, Microsoft SQL Server, IBM DB2, SQLite, MongoDB,Oracle 등이 있음
3. 웹 애플리케이션 서버 (WAS)
 : 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 라이브러리 제공
 : 데이터베이스 서버와 연동해서 사용
 : GlassFish, JBoss, Jetty, JEUS, Resin, WebLogic, WebSphere, Tomcat
 : 가용성, 성능, 기술지원, 구축비용
4. 오픈소스 사용시 고려사항
 : 라이선스 종류, 사용자 수, 기술의 지속 가능성 고려

섹션 6 요구사항 정의

요구사항의 개념 및 특징

:요구사항은 소프트웨어가 어떤 문제를 해결하기 위해 제공하는 서비스에 대한 설명과 정상적으로 운영되는데 필요한 제약조건등을 나타냄

: 소프트웨어 개발이나 유지 보수 과정에서 필요한 기준과 근거를 제공

: 소프트웨어의 전반적인 내용을 확인할 수 있게 하여 이해관계자들 간의 의사소통 원활하게 함

: 요구사항이 제대로 정의되어야, 이후 과정의 목표와 계획을 수립할 수 있음

요구사항의 유형

: 기술 내용에 따라 **기능/비기능** 으로 구분하고, 기술 관점과 대상의 범위에 따라 **시스템/사용자**로 나눌 수 있음

기능 요구사항	· 시스템이 무엇을 하는지, 어떤 기능을 하는지에 대한 사항 · 시스템의 입출력으로 무엇이 포함되는지, 어떤 데이터를 저장하거나 연산을 수행해야 하는지 · 시스템이 반드시 수행해야 하는 기능. · 사용자가 시스템을 통해 제공받기를 원하는 기능
비기능 요구사항	· 시스템 장비 구성 요구사항 : 하드웨어, 소프트웨어, 네트워크 등 · 성능 요구사항 : 처리 속도 및 시간, 처리량, 동·정적 적용량, 가용성 · 인터페이스 요구사항 : 시스템 인터페이스와 사용자 인터페이스 및 다른 시스템과의 정보 교환에 사용하는 프로토콜과의 연계도 포함하는 기술 · 데이터 요구사항 : 초기 자료 구축 및 데이터 변환을 위한 대상 방법, 보안등 · 테스트 요구사항 : 장비 성능 테스트 · 보안 요구사항 : 시스템의 데이터 및 기능, 운영 접근 통제 · 품질 요구사항 : 관리가 필요한 품질항목, 품질 평가 대상에 대한 요구사항 · 제약사항 · 프로젝트 관리 & 지원 요구사항
사용자 요구사항	· 사용자의 관점에서 본 시스템이 제공해야 할 요구사항 · 사용자를 위한 것으로 친숙한 표현으로 이해하기 쉽게 작성됨
시스템 요구사항	· 개발자 관점에서 본 시스템 전체가 사용자와 다른 시스템에 제공해야 할 요구사항 · 사용자 요구사항에 비해 전문적이고 기술적인 용어로 표현 · 소프트웨어 요구사항

요구사항 개발 프로세스

개발 대상에 대한 요구사항을 체계적으로 도출하고 이를 분석한 후 분석 결과를 명세서에 정리한 다음 마지막으로 이를 확인 및 검증하는 일련의 구조화된 활동

: 이 프로세스가 진행되기 전에 개발 프로세스가 비즈니스 목적에 부합하는지, 예산은 적정한지 등에 대한 정보를 수집, 평가한 보고서를 토대로 타당성 조사가 선행되어야 함.

: 이 요구사항 개발은 요구공학의 한 요소이다.

: 도출 -〉 분석 -〉 명세 -〉 확인

요구공학

: 무엇을 개발해야 하는지 요구사항을 정의하고, 분석 및 관리하는 프로세스를 연구하는 학문.

1. 요구사항 도출 (Requriement Elicitation, 요구사항 수집)

: 시스템, 사용자,그리고 시스템 개발에 관련된 사람들이서로 의견을 교환하여 요구사항이어디에 있는지, 어떻게수집할 것인지를 식별하고 이해하는 과정이다.

- 요구사항 도출은 소프트웨어가 해결해야 할 문제를 이해하는 첫 번째 단계
- 이 단계에서 개발자와 고객 사이의 관계가 만들어지고 이해관계가 식별된다.
- 요구사항 도출은 소프트웨어 개발 생명 주기 동안 지속적으로 반복된다.
- 요구사항을 도출하는 주요 기법에는 청취와 인터뷰, 설문, 브레인스토밍, 워크샵, 프로토타이핑, 유스케이스 등이 있다.

2. 요구사항 분석

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

- 사용자 요구사항의 타당성을 조사하고 비용과 일정에 대한 제약을 설정
- 내용이 중복되거나 하나로 통합되어야 하는 등 서로 상충되는 요구사항이 있으면 이를 중재.
- 도출된 요구사항들을 토대로 소프트웨어의 범위와 주변 환경을 이해
- 요구사항 분석에는 자료 흐름도(DFD), 자료 사전(DD)등의 도구가 사용됨

3. 요구사항 명세 (Requirement Analysis)

: 요구사항 명세는 분석된 요구사항을 바탕으로 모델을 작성하고 문서화하는 것

- 문서화 시 기능 요구사항을 빠짐없이 완전하고 명확하게 기술하며, 비기능 요구사항은 필요한 것만 기재해야 한다.
- 사용자가 이해하기 쉽고, 개발자가 효과적으로 설계할 수 있도록 구성해야 한다.
- 설계 과정에서 잘못된 부분이 확인될 경우 그 내용을 요구사항 정의서에서 추적할 수 있어야함.
- 구체적인 명세를 위해 소단위 명세서(Mini-Spec)가 사용될 수 있다.

4. 요구사항 확인(Requirement Validation, 요구사항 검증)

: 개발 자원을 요구사항에 할당하기 전에 요구사항 명세서가 정확하고 완전하게 작성되었는지 검토하는 활동

- 요구사항이 실제 요구를 반영하는, 상충하는 요구들은 없는지 확인 및 검증, 점검을 해야한다.
- 이 검증 과정을 통해 모든 문제를 확인할 수 있는 것은 아니다.
- 일반적으로 요구사항 관리 도구를 이용하여 요구사항 정의 문서들에 대해 항상 관리를 수행한다.

〈요구사항 명세 기법〉

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	1. 요구사항을 정확하고 간결히 표현 2. 요구사항에 대한 결과가 작성자에 관계없이 일관성이 있으므로 완전성 검증이 가능함 3. 표기법이 어려워 사용자가 이해하기 어려움	1. 자연어의 사용으로 인해 요구사항에 대한 결과가 작성자에 따라 다를 수 있어 일관성이 떨어지고, 해석이 달라질 수 있음 2. 내용의 이해가 쉬어 의사소통이 용이
종류	VDM, Z, Petri-net, CSP 등	FSM, Decision Table, ER모델링, State Chart(SADT)등

섹션 7 요구사항 분석

요구사항 분석의 개요

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

- 사용자 요구의 타당성을 조사하고, 비용과 일정에 대한 제약 설정
- 사용자 요구를 정확히 추출하여 목표를 정하고, 어떤 방식으로 해결할지 결정
- 사용자 요구사항을 정확하고 일관성 있게 분석하여 문서화 필요
- UML(Unified Modeling Language) , 자료 흐름도(DFD), 자료사전(DD), 소단위 명세서(Mini - Spec.), 개체 관계도(ERD), 상태 전이도(STD), 제어 명세서

구조적 분석 기법

: 자료의 흐름과 처리를 중심으로 하는 요구사항 분석 방법

- 도형중심의 분석용 도구와 분석 절차를 이용하여 사용자의 요구사항을 파악하고 문서화 한다
- 도형을 이용하므로 분석가와 사용자 간의 대화가 용이
- 하양식 방법을 사용하여 시스템 세분화. 분석의 중복 배제

- 사용자의 요구사항을 논리적으로 표현하여, 전체 시스템을 일관성 있게 이해

자료 흐름도 (DFD : Data Flow Diagram)

: 요구사항 분석에서 자료의 흐름 및 변환 과정과 긴능을 도형 중심으로 기술하는 방법으로 자료 흐름 그래프, 버블 차트라고도 한다.

- 시스템 안의 프로세스와 자료 저장소 사이에 자료의 흐름을 나타내는 그래프로 자료흐름과 처리를 중심으로 하는 구조적 부넉 기법에 이용된다
- 단계적으로 세분화 하고, 처리를 거쳐 변환될 때마다 새로운 이름이 부여, 처리는 입력 자료가 발생하면 기능을 수행한 후 출력 자료를 산출
- 다음과 같이 4가지 기호로 표시

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

자료 사전 (DD)

: 자료흐름도에 있는 자료를 더 자세히 정의하고 기록한 것이며, 이처럼 데이터를 설명하는 데이터를 데이터의 데이터 또는 메타 데이터라고 한다.

- 정보를 체계적이고 죅적으로 모아 편리하게 이용가능

기호	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)
* *	자료의 설명 : 주석(Comment)

섹션 8 요구사항 분석 CASE와 HIPO

요구사항 분석을 위한 CASE(자동화 도구)

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

이점 - 표준화와 보고를 통한 문서화 품질 개선, 데이텁베이스가 모두에게 이용 가능하다는 점에서 분석자들 간의 적절한 조정. 명세에 대한 유지보수 비용 축소 등

종류 - SADT , SREM, PSL/PSA, TAGS, EPOS 등

1. 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 도구 , 블록 다이어그램을 채택
2. SREM (Software Requirements Engineering Methodology) =RSL/REVS : TRW사가 개발한 것으로 RSL과 REVS를 사용하는 자동화 도구.

- RSL(Requirement Statement Language) : 요소, 속성, 관계, 구조들을 기술하는 요구사항 기술 언어
- REVS(Requirement Engineering and Validation System) : RSL로 기술된 요구사항들을 자동으로 분석하여 명세서로 출력하는 요구사항 분석기

3. PSL/PSA : 미시간 대학에서 개발
   - PSL (Problem Statement Language) : 문제(요구사항) 기술 언어
   - PSA (Problem Statement Analyzer) : PSL로 기술한 요구사항을 자동으로 분석하여 다양한 보고서를 출력하는 문제 분석기
4. TAGS (Technology of Automated Gemeration of Systems) : 시스템 공학 방법 으용에 대한 자동 접근 방법, 개발 주기의 전 과정에 이용할 수 있는 통합 자동화 도구

-구성 : IORL, 요구사항 분석과 IORL 처리를 위한 도구, 기초적인 TAGS 방법론

-IORL : 요구사항 명세 언어

HIPO (Hierarchy Input Process Output)

: 시스템 분석 및 설계나 문서화할 때 사용되는 기법으로, 시스템 실행 과정인 입출력, 처리 기능을 나타낸다.

- 하양식 소프트웨어 개발을 위한 문서화 도구
- 기호, 도표 등을 사용하여 이해하기 쉽고 체계적.

- 변경, 유지보수 용이
- 시스템의 기능을 여러개의 고유 모듈들로 분할하여 이들 간의 인터페이스를 계층 구조로 표현한 것을 HIPO Chart라 함

HIPO Chart 종류
- 가시적 도표(도식 목차) : 시스템의 전체적인 기능과 흐름을 보여주는 계층 구조
- 총체적 도표(총괄/개요 도표) : 프로그램을 구성하는 기능을 기술로 입출력, 처리에 대한 전반적인 정보 제공하는 도표
- 세부적 도표(상세 도표) : 총체적 도표에 표시된 기능을 구성하는 기본 요소들을 상세히 기술

섹션 9 UML (Unified Modeling Language)

UML의 개요

: 시스템 분석, 설계 구현 등 개발 과정에서 시스템 개발자와 고객 또는 상호간의 의사소통이 원활하게 이루어지도록 표준화한 대표적인 객체지향 모델링 언어

- UML을 이용하여 시스템의 구조를 표현하는 6개의 구조 다이어그램과 7개의 행위 다이어그램을 작성 가능
- UML의 구성요소에는 사물(Things), 관계 (Relationships), 다이어그램 등 있음

사물(Things) : 모델을 구성하는 가장 중요한 기본 요소로, 다이어그램 안에서 관계가 형성될 수 있는 대상들을 말한다.

사물	내용
구조 사물 (Structural Things)	• 시스템의 개념적, 물리적 요소를 표현 • 클래스(Class), 유스케이스(Use Case), 컴포넌트(Component), 노드(Node) 등
행동 사물 (Behavioral Things)	• 시간과 공간에 따른 요소들의 행위를 표현 • 상호작용(Interaction), 상태 머신(State Machine) 등
그룹 사물 (Grouping Things)	• 요소들을 그룹으로 묶어서 표현 • 패키지(Package)
주해 사물 (Annotation Things)	• 부가적인 설명이나 제약조건 등을 표현 • 노트(Note)

※ 컴포넌트 : 문서, 라이브러리등 모듈화된 자원

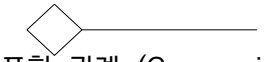
관계(Relationships)

: 사물과 사물 사이의 연관성을 표현하는 것으로 연관, 집합, 포함, 일반화, 의존, 실체화 관계로 총 6가지가 있다.

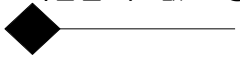
1. 연관 관계 : 2개 이상의 사물이 서로 관련 되어 있음 실선인 화살표를 사용하며, 서로에게 영향을 주는 양방향 관계인 경우 화살표를 생략하고 실선으로만 연결. 연관에 참여하는 객체의 개수를 의미하는 다중도를 선 위에 표기

2. 집합 관계 (Aggregation(소유)) : 하나의 사물이 다른 사물에 포함되어 있는 관계. 포함하는 쪽(전체)과 포함되는 쪽(부분)은 서로 **독립적**이다. 포함되는 쪽에서 포함하는 쪽으로 속이 빈마름모를 연결하여 표현

예) 컴퓨터와 프린터 : 프린터는 컴퓨터가 있어야 사용가능하나 다른 컴퓨터에서도 연결해 사용할 수 있으므로 독립적. 즉 집합 관계



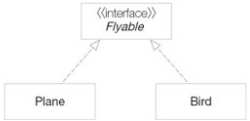
3. 포함 관계 (Composition) : 집합 관계의 특수한 형태로, 포함하는 사물의 변화가 포함되는 사물에게 영향을 미치는 관계를 표현한다. 포함하는 쪽과 포함되는쪽은 서로 독립될 수 없고 생명주기를 함께 함.



4. 일반화 관계 : 하나의 사물이 다른 사물에 비해 더 일반적인지, 구체적인지 표현. 일반적인 개념을 상위(부모), 보다 구체적인 개념을 하위(자식)이라 부른다. 하위에서 상위를 향하는 속이 빈 화살표를 연결하여 표현

5. 의존 관계 : 연관 관계와 같이 사물 사이에 서로 연관은 있으나 서로 필요에 의해 서로에게 영향을 주는 짧은 시간 동안만 연관을 유지하는 관계. 하나의 사물과 다른 사물이 소유관계는 아니나 하나의 변화가 다른 하나에 영향을 미침. 영향을 주는 사물이 영향을 받는 사물쪽으로 점선 화살표로 연결

6. 실체화 관계 : 사물이 할 수 있거나 해야 하는 기능(오퍼레이션, 인터페이스)으로 서로를 그룹화 할 수 있는 관계를 표현한다. 다시 말해 어떤 행동(의미적)으로 구분하는 것과 같음. 사물에서 기능 쪽으로 속이 빈 점선 화살표를 연결하여 표현한다.



다이어 그램

: 사물과 관계를 도형으로 표현한 것.

: 여러 관점에서 시스템을 가시화한 뷰(View)를 제공함으로써 의사소통에 도움

: 정적 모델링에서는 주로 구조적 다이어그램을 사용하고 동적 모델링에서는 주로 행위 다이어그램을 사용함

구조적 다이어그램

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.

동적 다이어그램

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.
시퀀스 다이어그램 **(Sequence Diagram)**	상호 작용하는 시스템이나 객체들이 주고받는 메시지를 표현함.
커뮤니케이션 다이어그램 **(Communication Diagram)**	시퀀스 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 활용.
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.

스트레오 타입 (Stereotype) : UML에서 표현하는 기본 기능 외에 추가적인 기능을 표현하기 위해 사용. 길러멧(Guilemet)이라 부르는 〈〈 〉〉 사용.

<<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

섹션 10 주요 UML 다이어그램

유스케이스 다이어그램 : 개발될 시스템과 관련된 외부요소들, 즉 사용자와 다른 외부 시스템들이 개발될 시스템을 이용해 수행할 수 있는 기능을 사용자의 관점에서 표현한 것.

- 유스케이스 다이어그램은 시스템 범위, 액터, 유스케이스, 관계로 구성된다.

1. 시스템/ 시스템 범위 : 시스템 내부에서 수행되는 기능들을 외부 시스템과 구분하기 위 | 해 시스템 내부의 유스케이스들을 사각형으로 묶어 시스템의 범위를 표현함
2. 액터 : 시스템과 상호작용을 하는 모든 외부 요소로, 사람이나 외부 시스템을 의미. 주액터/부액터로 나뉨.
3. 유스케이스 : 사용자가 보는 관점에서 시스템이 액터에게 제공하는 기능을 표현
4. 관계 : 액터와 유스케이스, 유스케이스와 유스케이스 사이에서만 나타나며, 연관·포함·**확장**·일반화 관계를 표현할 수 있다.

클래스 다이어그램 : 시스템을 구성하는 클래스, 클래스의 특성인 속성과 오퍼레이션(함수), 제약조건, 클래스간의 관계를 포함한 것

- 클래스 다이어그램은 클래스, 제약조건, 관계를 포함한다.

1. 클래스 : 이름, 속성, 오퍼레이션을 표기
2. 제약조건 : 속성에 입력될 값에 대한 제약조건이나 오퍼레이션 수행 전후에 지정해야할 조건이 있으면 적는다.
3. 관계 : 클래스와 클래스 사이의 연관성. 연관·집합·포함·일반화·의존이 있음

접근제어자 : + public // - private // # protected // ~ package

순차 다이어 그램 : 시스템이나 객체들이 메시지를 주고받으며 시간의 흐름에 따라 상호 작용하는 과정을 액터, 객체, 메시지 등의 요소를 사용하여 그림으로 표현
- 각 동작에 참여하는 시스템이나 객체들의 수행 기간을 확인 가능
- 객체들을 기본 단위로 상호작용을 표현
- 주로 기능 모델링에서 작성한 유스케이스 명세서를 하나의 표현 범위로 하지만, 하나의 클래스에 포함된 오퍼레이션을 하나의 범위로 표현하기도 함
- 순차 다이어그램은 액터, 객체, 생명선, 실행, 메시지로 구성됨

1. 액터 : 시스템으로부터 서비스를 요청하는 외부 요소. 사람이나 외부 시스템
2. 객체 : 메시지를 주고받는 주체
3. 생명선 : 객체가 메모리에 존재하는 기간, 객체 아래쪽에 점선을 그어 표현
4. 실행 상자 : 객체가 메시지를 주고받으며 구동되고 있음을 표현
5. 메시지 : 객체가 상호 작용을 위해 주고받는 메시지

2장

섹션 11 사용자 인터페이스(UI ; User Interface)

사용자 인터페이스의 개요

: UI는 사용자와 시스템 간의 상호작용이 원활하게 이루어지도록 도와주는 장치나 소프트웨어를 의미한다.

: 초기의 UI는 단순히 사용자와 컴퓨터 간의 상호작용에만 국한되었으나, 점차 사용자가 수행할 작업을 구체화시키는 기능 위주로 변경되었고, 최근에는 정보 내용을 전달하기 위한 표현 방법으로 변경 되었다.

- 사용자 인터페이스의 세 가지 분야

1. 정보 제공과 전달을 위한 물리적 제어에 관한 분야
2. 콘텐츠의 상세적인 표현과 전체적인 구성에 관한 분야
3. 모든 사용자가 편리하고 간편하게 사용하도록 하는 기능에 관한 분야

UI의 특징

: 사용자의 만족도에 가장 큰 영향을 미치고, SW 영역 중 변경이 제일 많음

: 사용자의 편리성과 가독성을 높임으로써 작업 시간을 단축시키고 업무에 대한 이해도를 높여준다

: 최소한의 노력으로 원하는 결과 도출, 수행 결과의 오류를 줄임

: 사용자 중심으로 설계되어 사용자 중심의 상호 작용이 되도록 함

: 정보 제공자와 공급자간의 매개 역할

: UI를 설계하기 위해서는 소프트웨어 아키텍처를 반드시 숙지해야한다.

UI의 구분

: 상호작용의 수단 및 방식에 따라 구분함

1. CLI(Command Line Interface) : 명령과 출력이 텍스트 형태
2. GUI(Graphical User Interface) : 아이콘이나 메뉴를 마우스로 선택하여 작업
3. NUI(Natural User Interface) : 사용자의 말이나 행동으로 기기 조작
4. VUI(Voice User Interface) : 사람의 음성으로 기기를 조작
5. OUI(Organic User Interface) : 모든 사물과 사용자 간의 상호작용을 하기 위한 인터페이스로, 소프트웨어가 아닌 하드웨어 분야에서 사물 인터넷, 가상현실, 증강현실, 혼합현실등과 대두되고 있음

사용자 인터페이스의 기본 원칙

1. 직관성 : 누구나 쉽게 이해하고 사용할 수 있어야 함
2. 유효성 : 사용자의 목적을 정확하고 완벽하게 달성해야 한다.
3. 학습성 : 누구나 쉽게 배우고 익힐 수 있어야 함
4. 유연성 : 사용자의 요구사항을 최대한 수용하고 실수를 최소화

사용자 인터페이스 설계 지침

1. **사용자 중심** : 사용자가 쉽게 이해하고 편리하게 사용할 수 있는 환경 제공, 실사용자에 대한 이해가 바탕이 되어야함
2. **사용성** : 사용자가 얼마나 빠르고 쉽게 이해할 수 있는지, 얼마나 효율적으로 사용할 수 있는지. UI설계 시 가장 우선적으로 고려
3. 일관성 : 버튼이나 조작방법을 사용자가 기억하고 쉽게 습득하게 설계
4. 단순성 : 조작 방법을 단순화로 인지적 부담을 감소해야함
5. 결과 예측 가능 : 작동시킬 기능만 보고 결과를 예측해야함
6. 가시성 : 메인 화면에 주요 기능을 노출시켜 최대한 조작을 쉽게 설계해야함
7. **심미성** : 디자인적으로 완성도 높게 글꼴이나 색상 적용 및 그래픽 요소로 가독성을 높여야함
8. 표준화 :기능 구조와 디자인을 표준화 하여 한번 학습한 후에는 사용하기 쉽게
9. 접근성 : 사용자의 연령, 성별, 인종 등에 구애를 받지 아니해야함

10. 명확성 : 사용자가 개념적으로 쉽게 인지할 수 있도록 해야함
11. **오류 발생 해결** : 오류 발생시 사용자가 쉽게 인지하게 설계해야함

사용자 인터페이스 개발 시스템의 기능
1. 사용자의 입력을 검증할 수 있어야함
2. 에러 처리와 그와 관련된 에러 메시지를 표시할 수 있어야 함.
3. 도움과 프롬포트를 제공해야함

섹션 12 UI표준 및 지침 (C등급)

UI표준 및 지침
: UI표준 및 지침을 토대로 기술의 중립성(웹 표준), 보편적 표현 보장성(웹 접근성), 기능의 호환성(웹 호환성)이 고려되었는지 확인한다.
UI표준 – 전체 시스템에 포함된 모든 UI에 공통적으로 적용될 내용으로, 화면 구성이나 화면 이동 등이 포함된다.
UI지침 – UI 요구사항,구현 시 제약사항 등 UI 개발 과정에서 지켜야할 공통 조건

웹표준 : 웹에서 사용되는 규칙 또는 기술. 웹사이트 작성시 사용하는 HTML, JS등에 대한 규정, 웹 페이지가 다른 기종이나 플랫폼에서도 구현되도록 제작하는 기법.
웹 접근성 : 누구나, 어떠한 환경에서도 웹 사이트에서 제공하는 모든 정보를 접근하여 이용할 수 있도록 보장하는 것
웹 호환성 : 하드웨어나 소프트웨어 등이 다른 환경에서도 모든 이용자에게 동등한 서비스를 제공하는 것

한국형 웹 콘텐츠 접근성 지침
(KWCAG; Korean Web Content Accessibility Guidelines)
: 장애인과 비장애인이 동등하게 접근할 수 있는 웹콘텐츠의 제작 방법 제시
: 고려사항으로 인식의 용이성, 운용의 용이성, 이해의 용이성, 견고성이 있음

내비게이션 : 사용자가 사이트에서 원하는 정보를 빠르게 찾을 수 있도록 안내하는 갓. 주로 사이트 상단의 메뉴바라고 보면 된다.

전자정부 웹 표준 준수 지침
: 정부기관의 홈페이지 구축 시 반영해야할 최소한의 규약. 모든 사람이 시스템환경에 구애받지 않고, 정부기관의 홈페이지를 이용할 수 있도록 하기 위함
: 내용의 문법 준수, 내용과 표현의 분리, 동작의 기술 중립성 보장. 플러그인의 호환성, 콘텐츠의 보편적 표현, 운영체제에 독립적인 콘텐츠 제공, 부가 기능의 호환성 확보, 다양한 프로그램 등의 지침이 있음

섹션 13 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.
: UI설계도구로 작상된 결과물은 사용자의 요구사항이 실제 구현되었을 때 화면은 어떻게 구성되는지, 어떤 방식으로 수행되는지 등을 기획단계에서 미리 보여줌

1. 와이어프레임 : 기획 단계의 초기에 제작하는 것으로, 페이지에 대한 개략적인 레이아웃이나 UI요소 등에 대한 뼈대를 설계하는 단계
 - 툴 : 손그림, 파워포인트, 키노트, 스케치, 일러스트, 포토샵 등
2. 목업(Mockup) : 디자인, 사용 방법 설명, 평가 등을 위해 와이어프레임보다 좀 더 실제 화면과 유사하게 만든 **정적**인 형태의 모형. 실제로 구현 하지 않고, 구성요소를 시각적로만 배치
 - 툴 : 파워 목업, 발사믹 목업
3. 스토리보드(Story Board) : 와이어프레임에 콘텐츠에 대한 설명, 페이지 간 이동 흐름 등을 추가 하는 문서. 디자이너와 개발자가 최종적으로 참고하는 작업 지침서. 서비스 구축을 위한 모든 정보가 들어 있음. 디스크립션에 세부적인 것들 기입
 - 툴 : 파워포인트, 키노트, 스케치, Axure
4. 프로토타입 : 와이어프레임이나 스토리보드 등에 인터랙션을 적용시켜 실제로 구현된 것처럼 테스트가 가능한 동적인 형태의 모형. 작성 방법에 따라 페이퍼 프로토타입과, 디지털 프로토타입으로 나뉨
 - 툴 : HTML/CSS, Axure, Flinto, 카카오오븐, 네이버 프로토나우 등
5. 유스케이스 : 사용자의 측면에서의 요구사항으로, 사용자가 원하는 목표를 달성하기 위해 수행할 내용을 구조적으로 표현함. 빠르게 문서화할 수 있음

섹션 14 UI 요구사항 확인 (C등급)

UI 요구사항 확인

: 새로 개발할 시스템에 적용할 UI 관련 요구사항을 조사해서 작성하는 단계

순서 : 목표 정의 -〉 활동 사항 정의 -〉 UI 요구사항 작성

1. 목표 정의 : 사용자들을 대상으로 인터뷰 후 수렴된 요구사항 정의
2. 활동 사항 정의 : 위의 요구사항을 토대로 앞으로 해야 할 활동 사항 정의
3. UI 요구사항 작성 : 요구사항 요소들을 검토 및 분석하여 요구사항을 작성한다.
   이 안에서도 순서가 있는데, : 요구사항 요소 확인 -〉 정황 시나리오 작성 -〉 요구사항 작성

여기서 사용자 중심으로 한다는 것만 이해하자

섹션 15 품질 요구사항 (A)

: 소프트웨어의 품질은 소프트웨어의 기능, 성능, 만족도 등 소프트웨어에 대한 요구사항이 얼마나 충족하는가를 나타내는 소프트웨어 특성의 총체이다. SW의 품질은 사용자 요구사항을 충족시킴으로써 확립된다.

ISO/IEC 9126 : 소프트웨어 품질 특성과 평가를 위한 표준 지침으로서 국제 표준으로 사용. SW 요구사항을 기술하거나 SW의 품질 평가에 사용

ISO/IEC 9126 품질 특성

: 기능성, 신뢰성, 사용성, 효율성, 유지 보수성, 이식성

1. **기능성** : 소프트웨어가 사용자의 요구사항을 정확하게 만족하는 기능을 제공하는지의 여부
   - 적절성/적합성 : 지정된 작업과 사용자의 목적 달성을 위한 적절한 기능 제공
   - 정밀성/정확성 : 사용자가 요구하는 결과를 정확히 산출
   - 상호 운용성 : 다른 시스템들과 서로 어울려 작업할 수 있는 능력
   - 보안성 : 정보에 대한 접근을 권한에 따라 허용하거나 차단할 수 있는 능력
   - 준수성 : 기능과 관련된 표준, 관례 및 규정을 준수할 수 있는 능력
2. **신뢰성** : 소프트웨어가 요구된 기능을 정확하고 일관되게 오류없이 수행할 수 있는 정도
   - 성숙성 : 결함으로 인한 고장을 피해갈 수 있는 능력
   - 고장 허용성 : 결함 또는 인터페이스 결여 시에도 규정된 성능 수준을 유지할 수 있는 능력
   - 회복성 : 고장 시 규정된 성능 수준까지 다시 회복하고 직접적으로 영향 받은 데이터를 복구할 수 있는 능력
3. **사용성** : 사용자와 컴퓨터 사이에 발생하는 어떠한 행위에 대하여 사용자가 쉽게 배우고 사용할 수 있으며, 향후 다시 사용하고 싶은 정도
   - 이해성 : SW의 적합성, 사용 방법 등을 사용자가 이해할 수 있는 능력
   - 학습성 : SW 어플리케이션을 학습할 수 있도록 하는 능력
   - 운용성 : 사용자가 SW를 운용하고 제어할 수 있도록 하는 능력
   - 친밀성 : 사용자가 SW를 다시 사용하고 싶어 하도록 하는 능력
4. **효율성 :** 사용자가 요구하는 기능을 할당된 시간 동안 한정된 자원으로 얼마나 빨리 처리할 수 있는지에 대한 정도
   - 시간 효율성 : 특정 기능을 수행할 때 적절한 반응 시간 및 처리 시간, 처리율을 제공할 수 있는 능력
   - 자원 효율성 : 특정 기능을 수행할 때 적절한 자원의 양과 종류를 제공할 수 있는 능력
5. **유지 보수성** : 환경의 변화 혹은 새로운 요구사항이 발생했을 때 소프트웨어를 개선하거나 확장할 수 있는 정도
   - 분석성 : 결함이나 고장의 원인, 수정된 부분들의 식별을 가능하게 하는 능력
   - 변경성 : 결함 제거 또는 환경 변화로 인한 수정 등을 최소화할 수 있는 능력
   - 안정성 : 변경으로 인한 예상치 못한 결과를 최소화 할 수 있는 능력
   - 시험성 : 소프트웨의 변경이 검증될 수 있는 능력
6. **이식성** : 소프트웨어가 다른 환경에서도 얼마나 쉽게 적용할 수 있는지의 정도
   - 적용성 :원래의 목적으로 제공되는 것 외에 다른 환경으로 변경될 수 있는 능력
   - 설치성 : 임의의 환경에 소프트웨어를 설치할 수 있는 능력
   - 대체성 : 동일한 환경에서 동일한 목적을 위해 다른 소프트웨어를 대신하여 사용될 수 있는 능력
   - 공존성 : 자원을 공유하는 환경에서 다른 SW와 공존할 수 있는 능력

각 품질 특성에는 각각에 관련된 얼마나 그 표준, 관례, 규정들을 준수할 수 있는지의 준수성이 포함되어 있음

ISO/IEC 12119 : 9126에서 테스트 절차를 포함한 표준
ISO/IEC 14598 : 소프트웨어 품질의 측정과 평가에 필요 절차를 규정한 표준으로, 개발자, 구매자, 평가자 별로 수행해야 할 제품 평가 활동을 규정함
ISO/IEC 25010 : 소프트웨어 제품에 대한 국제 표준으로 9126의 호환성과 보안성을 강화한 규정으로 2011년에 제작되었다. 이것의 품질 특성으로는 기능 적합성, 성능 효율성, 호환성, 사용성, 신뢰성, 보안성, 유지 보수성, 이식성이 있다.

섹션 16 UI 프로토타입 제작 및 검토 (C)

UI프로토타입
: 앞에서 배웠든 실제 동작하는 것처럼 만든 동적인 형태의 모형. 최대한 간단히 만들어야함. 일부 핵심적인 기능을 제공하나 최종 제품의 작동 방식을 동작하기 위한 기능은 반드시 포함되어야한다. 사용자의 요구사항에 따라 계속 개선 및 보완해야함. 프로토타이핑 및 테스트가 필요하므로 실사용자를 대상으로 테스트가 좋음

UI프로토타입 장단점
장점 - 사용자를 설득하고 이해시키기 쉬움
- 요구사항과 기능의 불일치 등으로 인한 혼선 예방으로 개발시간 단축 가능
- 사전에 오류 발견 가능
단점 - 사용자의 모든 요구를 반영하기 위한 반복적인 작업이 큰 자원 소모와 작업 시간을 늘릴 수 있음.
- 부분적으로 프로토타이핑을 진행하다보면 중요한 작업이 생략될 수 있음

UI프로토타입 종류 : 페이퍼 프로토타입, 디지털 프로토 타입
페이퍼 : - 아날로그적, 제작기간이 짧고, 업무협의가 빠른 경우 사용. 저렴함. 큰 요구사항이 필요하지 않을 때 사용.
-그러나 테스트에 부적합. 공유하기 힘듬
디지털 : 프로그램을 이용하여 작성하는 방법. 최종과 비슷하게 테스트 가능하고 재사용 가능. 수정하기 용이
- 그러나 프로토타입을 작성할 프로그램의 사용법을 알아야 함

UI 프로토타입 계획 및 작성 시 고려 사항 : 일반적으로 프로토타입의 개발 계획을 수립하는 과정과 프로토타입을 개발 후 결과를 보고하는 과정으로 진행 (책참고)
UI 프로토타입 제작 단계는 4가지가 있음
1단계 : 사용자의 요구사항 분석하는 단계. 정리 및 확정될때까지 수행
2단계 : 프로토타입을 손으로 그리거나 편집 도구를 이용해 작성. 핵심적인 기능만
3단계 : 요구사항을 잘 수행하고 있는 사용자가 직접 확인. 의견 제안
4단계 : 3단계에서 요청한 의견을 보완. 수정 및 합의 이 3~4단계를 반복

섹션 17 UI 설계서 작성 (C)

UI 설계서의 개요
:UI 설계서는 사용자의 요구사항을 바탕으로 UI 설계를 구체화하여 **작성하는 문서**로, 상세 설계 전에 대표적인 화면들을 설계한다.
- UI 설계서는 기획자, 개발자, 디자이너 등과의 원활한 의사소통을 위해 작성한다.
- UI 설계서 표지, UI 설계서 개정 이력, UI 요구사항 정의서, 시스템 구조, 사이트 맵, 프로세스 정의서, 화면 설계 순으로 작성한다.

※UI 화면 설계는 실제로 사용할 UI를 설계하는 것이고, UI설계서는 실제 사용할 UI화면을 설계하기 전에 사용자의 요구사항을 가시화하고 검증하기 위해 작성.

1. UI 설계서 표지 작성 : 다른 문서와 혼동되지 않도록 프로젝트 명 또는 시스템 명을 포함시켜 작성
2. UI설개서 개정 이력 작성 : UI설계서가 수정될때마다 변경내용 및 작성자 기입
3. UI 요구사항 정의서 : 사용자의 요구사항을 확인하고 정리한 문서로, 사용자 요구사항의 UI 적용 여부를 요구사항별로 표시함
4. 시스템 구조 작성 : 시스템 구조는 UI 요구사항과 UI 프로토타입에 기초하여 전체 시스템의 구조를 설계한 것으로 사용자의 요구사항이 어떻게 시스템에 적용되는 지 알 수 있음
5. 사이트 맵 작성 : 시스템 구조를 바탕으로 사이트에 표시할 콘텐츠를 한 눈에 알아 볼 수 있도록 메뉴별로 구분하여 설계한 것
- 일반적으로 테이블 형태로 되어 있고, 위에서 아래로 내려가며 정보를 찾을 수 있는 계층형으로 되어 있는 것이 보통. (시각적인 콘텐츠 모형)
- 사이트 맵을 작성한 후 사이트 맵의 상세 내용을 표 형태로 작성한다.
6. 프로세스 정의서 작성 : 사용자 관점에서 사용자가 요구하는 프로세스들을 작업

진행 순서에 맞춰 정리한 것으로 UI의 전체적인 흐름을 파악할 수 있다.
7. 화면 설계 : UI프로토타입과 UI 프로세스를 참고하여 필요한 화면을 페이지별로 설계한 것. 화면별 고유 ID를 부여하고 별도 표지를 작성
 - 주요 흐름을 스토리보드 형태로 작성

섹션 18 유용성 평가 (D)

UI의 유용성 평가

- 유용성은 사용자가 시스템을 통해 원하는 목표를 얼마나 효과적으로 달성할 수 있는 가에 대한 척도로, UI의 주된 목적은 유용성이 뛰어난 UI를 제작하는 것이다.
- 유용성 평가는 사용자 측면에서 복잡한 시스템을 얼마나 편리하게 사용할 수 있는지를 평가하는 것으로, 시스템의 문제점을 찾아내고 개선 방향을 제시하기 위한 조사 과정이다.
- 유용한 UI를 설계하기 위해서는 UI의 구조, 기능, 가치 등에 대해 사용자가 생각하는 사용자 모형과 시스템 설계자가 만들려고 하는 개발자 모형 간의 차이를 최소화해야 한다.

사용자 모형과 개발자 모형간의 차이가 발생하는 원인

- 실행 차 : 사용자가 원하는 목적과 실행 기능이 다르기 때문
- 평가 차 : 사용자가 원하는 목적과 실행 결과가 다르기 때문

실행 차를 줄이기 위한 UI 설계 원리 검토

1. 사용 의도 파악
2. 행위 순서 규정
3. 행위의 순서대로 실행 (흐름에 맞게)

평가 차를 줄이기 위한 UI 설계 원리 검토

1. 수행한 키 조작의 결과를 사용자가 빠르게 지각하도록 유도
2. 키 조작으로 변화된 시스템의 상태를 사용자가 쉽게 인지하도록 유도
3. 사용자가 가진 원래 의도와 시스템 결과 간의 유사 정도를 사용자가 쉽게 파악하도록 유도

섹션 19 UI 상세 설계

UI 시나리오 문서 개요

: UI 상세 설계는 UI 설계서를 바탕으로 실제 설계 및 구현을 위해 모든 화면에 대한 자세한 설계를 진행하는 단계로, UI 상세 설계를 할 때는 반드시 시나리오를 작성해야 한다.

- 사용자 인터페이스의 기능 구조, 대표 화면, 화면 간 인터렉션의 흐름, 다양한 상황에서의 예외 처리 등을 문서로 정리한 것이다.
- 사용자가 최종 목표 달성을 위한 방법이 순차적으로 묘사되어 있다.
- UI 설계자 또는 인터랙션 디자이너가 UI 시나리오 문서를 작성하면 그래픽 디자이너가 시나리오를 바탕으로 디자인을 하고 개발자가 UI를 구현한다.

UI 시나리오 문서 작성 원칙

- 개발자가 전체적인 UI의 기능과 작동 방식을 한눈에 이해할 수 있도록 구체적으로 작성한다. 보통 계층 구조 또는 flow 차트 표기법으로 작성한다.
- 모든 기능에 공통적으로 적용될 UI요소와 인터랙션을 일반 규칙으로 정의한다.
- 인터랙션의 흐름을 정의하며, 화면 간 인터랙션의 순서, 분기, 조건, 루프등을 명시한다.

UI 시나리오 문서 작성을 위한 일반 규칙

구분	설명
주요 키의 위치와 기능	화면상에 공통적으로 배치되는 주요 키의 위치와 기능을 설명한 것으로 여러 화면 간의 일관성을 보장하기 위한 것이다.
공통 UI 요소	체크 박스, 라디오 버튼, 스크롤바, 텍스트 입력 필드, 상하/좌우 휠, 모드 설정, 탭, 팝업 등의 각 UI 요소를 언제 사용하며 어떤 형태인지 정의하고 사용자의 조작에 어떻게 반응하는지 그 흐름을 상세하게 설명한 것이다.
기본 스크린 레이아웃 (Basic Screen Layouts)	여러 화면 내에 공통적으로 나타나는 Indicators, Titles, Ok/Back, Soft Key, Option, Functional Buttons 등의 위치와 속성을 정의한 것으로서 여러 기능들 간에 화면 레이아웃의 일관성을 보장하기 위한 것이다.
기본 인터랙션 규칙 (Basic Interaction Rules)	터치 제스처 등의 공통적으로 사용되는 조작의 방법, 홈 키의 동작 방식과 같은 운항 규칙, 실행, 이전, 다음, 삭제, 이동 등의 화면 전환 효과 등에 대해 기술한 것이다.
공통 단위 태스크 흐름 (Task Flows)	많은 기능들에 공통적으로 자주 나타나는 삭제, 검색, 매너 모드 상태에서의 소리 재생 등의 인터랙션 흐름을 설명한 것이다.
케이스 문서	다양한 상황에서의 공통적인 시스템 동작에 대해 정의한 문서이다. (ex. 사운드, 조명, 이벤트 케이스 등)

UI의 요소 : 체크 박스, 라디오 버튼(여러개 중에 선택하는 버튼), 텍스트 박스, 콤보상자, 목록상자

UI 시나리오 문서의 요건
1. 완전성 : 누락되지 않도록 상세히, 사용자의 태스크에 초점을 맞춰 기술
2. 일관성 : 요구사항 스타일 등이 모두 일관성을 유지
3. 이해성 : 누구나 쉽게 이해하게
4. 가독성 : 표준화된 템플릿 등을 활용하여 문서를 쉽게 읽을 수 있게
5. 수정 용이성 : 시나리오 수정이나 개선이 쉬워야함
6. 추적 용이성 : 변경 사항은 언제, 어떤 부분이, 왜 발생했는지 쉽게 추적할 수 있어야 한다.

UI 시나리오 문서로 인한 기대 효과
- 요구사항이나 의사소통에 대한 오류가 감소
- 개발 과정에서의 재작업이 감소, 혼선이 최소화
- 불필요한 기능 최소화
- 소프트웨어 개발 비용을 절감
- 개발 속도를 향상시킨다.

※인터랙션 디자이너는 시나리오 문서를 작성하는 사람임
※ 템플릿 - 형판, 형틀 : 화면의 기본적인 레이아웃을 의미

섹션 20 HCI / UX / 감성공학 (C)

HCI(Human Computer Interaction or Interface)
: 사람이 시스템을 보다 편리하고 안전하게 사용할 수 있도록 연구하고 개발하는 학문으로, 최종 목표는 시스템을 사용하는데 있어 최적의 사용자 경험(UX)를 만드는 것이다. 원래는 사람이 컴퓨터를 편리하게 사용하도록 만드는 학문이었으나, 대상이 컴퓨터뿐만 아니라 서비스, 디지털 콘텐츠 등으로, 사람도 개인뿐만 아니라 사회나 집단으로 확대되었다.
- HCI는 어떤 제품이 좋은 제품인지, 어떻게 하면 좋은 제품을 만들 수 있는지 등을 연구한다.

UX(User Experience)
: UX는 사용자가 시스템이나 서비스를 이용하면서 느끼고 생각하게 되는 총체적인 경험. 단순히 기능이나 절차상의 만족뿐만 아니라 사용자가 참여, 사용, 관찰하고, 상호 교감을 통해서 알 수 있는 가치 있는 경험
- 삶의 질을 향상시키는 하나의 방향으로 보는 새로운 개념
- 특징 : 주관성, 정황성, 총체성

감성공학 : 사용자가 제품을 사용한 경험을 통해 얻은 복합적인 감각
- 제품이나 작업환경을 사용자의 감성에 알맞도록 설계 및 제작하는 기술로, 인문사회과학, 공학, 의학 등 여러 분야의 학문이 공존하는 종합과학
- 감성공학은 HCI 설계에 인간의 특성과 감성을 반영하였음

감성공학의 요소 기술 : 기반 기술, 구현 기술 , 응용 기술

3장 - 애플리케이션 설계

섹션 21 소프트웨어 아키텍처

소프트웨어 아키텍처의 설계 : 소프트웨어 아키텍처는 소프트웨어의 골격이 되는 기본 구조 이자 소프트웨어를 구성하는 요소들 간의 관계를 표현하는 시스템의 구조 또는 구조체이다.
- 소프트웨어 개발 시 적용되는 원칙과 지침, 이해 관계자들의 의사소통 도구
- 사용자의 비기능적 요구사항으로 나타난 제약을 반영하고, 기능적 요구사항을 구현하는 방법을 찾는 해결 과정이다.
- 애플리케이션의 분할 방법과 분할된 모듈에 할당될 기능, 모듈 간의 인터페이스 등을 결정한다.
- 소프트웨어 아키텍처 설계의 기본 원리로는 모듈화, 추상화, 단계적 분해, 정보은닉이 있다.

상위설계와 하위 설계 : SW 개발의 설계 단계는 크게 상위/하위로 구분 가능

	상위 설계	하위 설계
별칭	아키텍처, 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

1. 모듈화 : 소프트웨어의 성능을 향상시키거나 시스템의 수정 및 재사용, 유지 관리 등이 용이하도록 시스템의 기능을 모듈 단위로 나누는 것
 - 자주 사용되는 계산식이나 사용자 인증과 같은 기능들은 공통 모듈로 구성하면 재사용성이 증가한다.
 - 모듈의 크기를 너무 작게 나누면 개수가 많아져 모듈간 통합 비용이 증가하고, 너무 크게 나누면, 각 모듈의 개발 비용이 많이 든다.
 - 모듈화를 통해 기능의 분리가 가능하여 인터페이스가 단순해지고, 오류 최소화

2. 추상화 : 문제의 전체적이고 포괄적인 개념을 설계한 후 차례로 세분화하여 구체화시켜 나가는 것.
 - 인간이 복잡한 문제를 다룰 때 가장 기본적으로 사용하는 방법으로, 완전한 시스템을 구축하기 전에 그 시스템과 유사한 모델을 만들어서 여러 가지 요인 테스트
 - 추상화는 최소 비용으로 실제 상황에 대처하고, 시스템 구조 및 구성을 파악함
 - **추상화의 유형 : 과정 추상화, 데이터 추상화, 제어 추상화**

3. 단계적 분해 (Stepwise Refinement) : NIklaus Wirth에 의해 제안된 하향식 설계 전략으로, 문제를 상위의 중요 개념으로부터 하위의 개념으로 구호화 시키는 분할 기법이다.
 - 추상화의 반복에 의해 세분화된다.
 - SW의 기능에서부터 시작되어 점차적으로 구체화하고, 알고리즘, 자료구조등 상세한 내역은 가능한 한 뒤로 미루어 진행한다.

4. 정보 은닉(Information Hiding) : 정보 은닉은 한 모듈 내부에 포함된 절차와 자료들의 정보가 감추어져 다른 모듈이 접근하거나 변경하지 못하도록 하는 기법이다.
 - 어떤 모듈이 소프트웨어 기능을 수행하는데 반드시 필요한 기능이 있어 정보 은닉된 모듈과 커뮤니케잇ㄴ할 필요가 있을 때는 필요한 정보만 인터페이스를 통해 주고 받는다.
 - 정보 은닉을 통해 모듈을 독립적으로 수할 수 있고, 하나의 모듈이 변경되더라도 다른 모듈에 영향을 주지 않으므로, 수정, 시험 유지보수에 용이

소프트웨어 아키텍처의 품질 속성 : SW아키텍처가 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 것

1. 시스템 측면 : 성능, 보안, 가용성, 기능성, 사용성, 변경 용이성, 확장성, 기타 속성(테스트 용이성, 배치성, 안정성 등)
2. 비즈니스 측면 : 시장 적시성, 비용과 혜택, 예상 시스템 수명, 기타속성(목표 시장, 공개 일정, 기존 시스템과의 통합)
3. 아키텍처 측면 : 개념적 무결성, 정확성/완결성, 구축 가능성, 기타 속성(변경성, 시험성, 적응성, 일치성, 대체성 등)

소프트웨어 아키텍처의 설계 과정
1. 설계 목표 설정 : 시스템의 개발 방향을 명확히 하기 위해 설계에 영향을 주는 비즈니스 목표,우선순위 등의 요구사항을 분석하여 전체 시스템의 설계목표 설정
2. 시스템 타입 결정: 시스템과 서브시스템의 타입 결정하고, 설계 목표와 함께 고려하여 아키텍처 패턴을 선택한다.
3. 아키텍처 패턴 적용 : 아키텍처 패턴을 참조하여 시스템의 표준 아키텍처를 설계
4. 서브시스템 구체화 : 서브시스템의 기능 및 서브시스템 간의 상호작용을 위한 동작과 인터페이스를 정의
5. 검토 : 아키텍처가 설계 목표에 부합하는지, 요구사항이 잘 반영되었는지, 설계의 기본 원리를 만족하는지 등을 검토

시스템 타입/ 협약에 의한 설계
시스템 타입 :
- 대화형 시스템 : 사용자 요구 발생시 시스템이 처리
- 이벤트 중심 시스템 : 외부 상태 변화에 따른 동작
- 변환형 시스템 : 데이터가 입력되면 정해진 작업을 수행하여 결과를 출력
- 객체 영속형 시스템 : 데이터베이스를 사용하여 파일을 효과적으로 저장, 검색, 갱신할 수 있는 시스템

협약(Contract)에 의한 설계 : 컴포넌트를 설계할 때 클래스에 대한 여러 가정을 공유할 수 있도록 명세한 것으로, 소프트웨어 컴포넌트에 대한 정확한 인터페이스를

명세. 명세에 포함될 조건에는 선행조건, 결과조건, 불변 조건이 있음
- 선행 조건 : 오퍼레이션이 호출되기 전에 참이 되어야 할 조건
- 결과 조건 : 오퍼레이션이 수행된 후 만족되어야 할 조건
- 불변 조건 : 오퍼레이션이 실행되는 동안 항상 만족되어야 할 조건

섹션 22 아키텍처 패턴

아키텍처 패턴의 개요
: 아키텍처를 설계할 때 참조할 수 있는 전형적인 해결 방식 또는 예제
- 아키텍처 패턴은 SW시스템의 구조를 구성하기 위한 기본적인 윤곽 제시
- 아키텍처 패턴에는 서브시스템들과 그 역할이 정의도이어 있으며, 서브시스템 사이의 관계와 여러 규칙, 지침 등이 포함되어 있음.
- 같을 말로 아키텍처 스타일 또는 표준 아키텍처라고 한다.

장점 - 시행착오를 줄여 개발 시간 단축, 고품질 소프트웨어 생산
- 검증된 구조 사용으로 안정적인 개발 가능
- 공통된 아키텍처 공유로 이해관계자들의 의사소통이 간편해짐
- 구조를 이해하기 쉬워져 손쉽게 유지보수 가능
- 시스템의 특성을 개발전에 예측하는 것이 가능

종류 : 레이어, 클라이언트-서버, 파이프-필터, MVC 패턴 등

1. 레이어 패턴
: 시스템을 계층으로 구분하여 구성하는 고전적인 방법
-각각의 서브시스템들이 계층 구조를 이루고, 하위 계층은 상위 계층에 대한 서브스 제공자가 되고, 상위 계층은 하위 계층의 클라이언트가 된다.
- 서로 마주보는 두 개의 계층 사이에만 상호작용이 이루어지고, 따라서 변경·수정이 용이
- 특별 계층만 교체하여 시스템 개선가능.
- 대표적으로 OSI 7계층 모델이 있음

2. 클라이언트 - 서버 패턴
: 하나의 서버 컴포넌트와 다수의 클라이언트 컴포넌트로 구성되는 패턴
- 이 패턴에서 사용자는 오직 클라이언트와만 의사소통을 한다. 즉, 사용자가 클라이언트를 통해서 서버에 요청하고 클라이언트가 응답을 받아 사용자에게 제공하는 방식으로 서비스를 제공한다.
- 서버는 클라이언트 요청에 대비해 항상 대기상태이어야 한다.
- 클라이언트나 서버는 요청 응답을 위해 동기화를 제외하고는 서로 독립적임

3. 파이프-필터 패턴
: 데이터 스트림 절차의 각 단계를 필터 컴포넌트로 캡슐화하여 파이프를 통해 데이터를 전송하는 패턴이다.
- 재사용성이 좋고, 추가가 쉬워 확장이 용이
- 필터 컴포넌트들을 재배치하여 다양한 파이프라인을 구축하는 것이 가능
- 이 패턴은 데이터 변환, 버퍼링, 동기화 등에 주로 사용된다.
- 필터 간 데이터 이동 시 데이터 변환으로 인한 오버헤드가 발생
- 대표적으로 UNIX의 쉘(Shell)이 있음

4. MVC 패턴 (모델 - 뷰 - 컨트롤러)
: 이 패턴은 서브시스템을 3가지의 부분으로 구조화 하는 패턴
모델 - 서브시스템의 핵심 기능과 데이터를 보관
뷰 - 사용자에게 정보를 표시
컨트롤러 - 사용자로부터 입력된 변경 요청을 처리하기 위해 모델에게 명령함
- MVC 패턴의 각 부분은 별도의 컴포넌트로 분리되어 있어 서로 영향을 받지 않고 개발 작업을 수행할 수 있음
- 여러개의 뷰를 만들 수 있으므로, 한 개의 모델에 대해 여러 개의 뷰를 필요로 하는 대화형 어플리케이션에 적합

5. 기타 패턴
마스터- 슬레이브 패턴 : 마스터 컴포넌트는 동일한 구조의 슬레이브 컴포넌트로 작업을 분할하고, 할당. 그 후 슬레이브에서 처리된 결과물을 다시 돌려받는 방식. 주로 장애 허용 시스템과 병렬 컴퓨터 시스템에서 활용
브로커 패턴 : 사용자가 원하는 서비스와 특성을 브로커 컴포넌트에 요청하면 브로커 컴포넌트가 요청에 맞는 컴포넌트와 사용자를 연결해주는 패턴. 주로 분산 환경 시스템에서 활용
피어-투-피어 패턴 : 피어를 하나의 컴포넌트로 간주하여 상황에 따라 각 피어가

서버의 역할 할 수 있고, 클라이언트의 역할을 할 수 있는 패턴. 전형적인 멀티스레딩 방식을 사용

이벤트- 버스 패턴 : 소스가 특정 채널에 이벤트 메시지를 발행하면, 해당 채널을 구독한 리스너들이 메시지를 받아 이벤트를 처리하는 방식. 여기서 4가지 컴포넌트가 있음 (소스, 리스너, 채널, 버스)

블랙보드 패턴 : 모든 컴포넌트들이 공유 데이터 저장소와 블랙보드 컴포넌트에 접근이 가능한 형태로, 컴포넌트들은 검색을 통해 블랙보드에서 원하는 데이터를 찾을 수 있음. 해결책이 명확하지 않은 문제를 처리하는데 유용한 패턴. 음성 인식, 차량 식별, 신호 해석에 쓰임

인터프리터 패턴 : 프로그램 코드의 각 라인을 수행하는 방법을 지정하고, 기호마다 클래스를 갖도록 구성. 특정 언어로 작성된 프로그램 코드를 해석하는 컴포넌트를 설계할 때 사용

섹션 23 객체 지향

객체지향의 개요 : 객체지향은 현실 세계의 개체를 기계의 부품처럼 하나의 객체로 만들어, 기계적인 부품들을 조립하여 제품을 만들 듯이 소프트웨어를 개발할 때에도 객체들을 조립해서 작성할 수 있는 기법

- 구조적 기법의 문제점으로 인한 소프트웨어 위기의 해결책
- 재사용 및 확장이 용이. 유지보수 쉬움
- 복잡한 구조를 단계적, 계층적으로 표현하고, 멀티미디어 데이터 및 병렬 처리 지원
- 주요 특성 : 객체, 클래스, 캡슐화, 상속, 다형성, 연관성

객체(Objcet) : 데이터와 데이터를 처리하는 함수를 묶어 놓은 (캡슐화한) 하나의 소프트웨어 모듈

- 데이터(속성, 상태 변수, 상수, 자료구조 등)와 함수로 이루어짐
- 객체가 반응할 수 있는 메시지의 집합을 행위라고 하며, 객체는 행위의 특징을 나타낼 수 있다.

클래스 : 클래스는 공통된 속성과 연산을 갖는 객체의 집합으로, 객체의 일반적인 타입을 의미한다.

- 클래스는 각각의 객체들이 갖는 속성과 연산을 정의하고 있는 틀이다.
- 클래스에 속한 각각의 객체를 인스턴스라 하며, 클래스로부터 새로운 객체를 생성하는 것을 인스턴스화라고 한다.
- 동일 클래스의 인스턴스는 동일한 속성을 갖으나, 그 속성의 정보는 서로 다를 수 있다.

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

- 캡슐화된 ㄴ객체는 인터페이스를 제외한 세부 내용이 은폐(정보 은닉)되어 외부의 접근이 제한적이기 때문에 외부 모듈의 변경으로 인한 파급 효과가 적다
- 재사용이 용이해진다.
- 객체들 간의 메시지를 주고 받을 때 상대 객체의 세부 내용은 알 필요가 없어 인터페이스가 단순해지고, 객체 간의 결합도가 낮아짐

상속 :이미 정의된 상위 클래스의 모든 속성과 연산을 하위 클래스가 물려 받는 것

- 상속을 이용하면 하위 클래스는 상위 클래스의 모든 속성과 과 연산을 자신의 클래스에 다시 정의하지 않고 물려받음
- 하위는 새로운 속성과 연산을 첨가할 수 있음
- 소프트웨어 재사용을 높이는 중요한 개념
- 다중상속 : 한 개의 클래스가 두 개 이사의 상위 클래스로부터 상속 받는 것

다형성 : 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 능력

- 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 것

연관성 : 두 개 이상의 객체(클래스)들이 상호 참조하는 관계를 말한다.

1. 연관화 : 2개 이상의 객체가 상호 관련되어 있음을 의미함 (is member of)
2. 분류화 : 동일한 형의 특성을 갖는 객체들을 모아 구성하는 것(is instance of)
3. 집단화 : 관련 있는 객체들을 묶어 하나의 상위 객체를 구성하는 것(is part of
4. 일반화 – 공통적인 성질들로 추상화한 상위 객체를 구성하는 것 (is a)
   특수화/상세화 – 상위 객체를 구체와하여 하위 객체를 구성하는 것 (is a)

섹션 24 객체지향 분석 및 설계

객체지향 분석(OOA: Object Oriented Analysis)의 개념

: 사용자의 요구사항을 분석하여 요구된 문제와 관련된 모든 클래스(객체), 이와 연관된 속성과 연산, 그들 간의 관계등을 정의하여 모델링하는 작업

- 소프트웨어를 개발하기 위한 비즈니스(업무)를 객체와 속성, 클래스와 멤버, 전체와 부분 등으로 나누어서 분석한다.
- 문제를 객체지향적으로 모형화하게 해줌
- 개게는 클래스로부터 인스턴스화하고, 이 클래스를 식별하는 것이 OOA의 주목적이다.

객체지향 분석의 방법론

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

럼바우의 분석 기법

: 럼바우의 분석 기법은 모든 SW 구성 요소를 그래픽 표기법을 이용하여 모델링하는 기법으로, 객체 모델링 기법(OMT; Objcet-Modling Technique)라고도 한다

- 분석 활동은 객체 모델링 -〉 동적 모델링 -〉 기능 모델링 순으로 이루어진다.

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

객체지향 설계 원칙

: 시스템 변경이나 확장에 유연한 시스템을 설계하기 위해 지켜야할 5가지 원칙 원칙들의 앞자리를 따 SOLID 원칙이라고 함

1. 단일 책임 원칙 (SRP; Single Responsibility Principle) : 객체는 단 하나의 책임만 지녀야한다는 원칙, 응집도는 높게, 결합도는 낮게 설계하는 것을 의미
2. 개방-폐쇄 원칙 (OCP; Open-Closed Principle) : 기존의 코드를 변경하지 않고 기능을 추가할 수 있도록 해야한다. 공통 인터페이스를 하나의 인터페이스로 묶어 캡슐화하는 것이 대표적
3. 리스코프 치환 원칙(LSP; Liskov Substitution Principle) : 자식 클래스는 최소한 자신의 부모 클래스에서 가능한 행위는 수행할 수 있어야한다는 설계 원칙. 자식 클래스는 반드시 부모 클래스에서 확장만 수행하도록 해야함
4. 인터페이스 분리 원칙(ISP; Interface Segregation Principle) : 자신이 사용하지 않는 인터페이스와 의존 관계를 맺거나 영향을 받지 않아야 한다는 원칙. 단일 책임 원칙이 객체가 갖는 하나의 책임이면, ISP는 인터페이스가 갖는 하나의 책임이다.
5. 의존 역전 원칙(DIP; Dependency Inversion Principle) : 각 객체들 간의 의존 관계가 성립될 때, 추상성이 낮은 클래스보다 추상성이 높은 클래스와 의존 관계를 맺어야한다는 원칙. 인터페이스를 활용하면 이 원칙은 준수됨

섹션 25 모듈

모듈의 개요

: 모듈화를 통해 분리된 시스템의 각 기능들로, 서브루틴, 서브시스템, 소프트웨어 내의 프로그램, 작업 단위 등과 같은 의미로사용된다.

- 모듈은 단독으로 컴파일 가능하며, 재사용이 가능하다.
- 모듈의 기능적 독립성은 SW를 구성하는 각 모듈의 기능이 서로 독립을 의미
- 독립성이 높은 모듈일수록 모듈을 수정하더라도 다른 모듈에 영향을 미치지 않고, 오류가 발생해도 쉽게 발견하고 해결 가능
- 모듈의 독립성은 결하보(Coupling)와 응집도(Cohesion)에 의해 측정되며, 독립성을 높이려면 결합도는 약하게, 응집도는 강하게, 크기는 작게 만들어야 함

결합도(Coupling)

: 모듈 간에 상호 의존하는 정도 또는 두 모듈 사이의 연관 관계를 의미

- 다양한 결합으로 모듈을 구성할 수 있으나 결합도가 약할수록 품질이 높고, 강할수록 품질이 낮다.
- 결합도가 높으면 구현 및 유지보수가 힘듬
- 결합도의 정도에 따라 다음과 같이 구분할 수 있음

자료 결합도 (Data Coupling)	• 모듈 간의 인터페이스가 자료 요소로만 구성될 때의 결합도. • 어떤 모듈이 다른 모듈을 호출하면서 매개 변수나 인수로 데이터를 넘겨주고, 호출 받은 모듈은 받은 데이터에 대한 처리 결과를 다시 돌려주는 방식. • 모듈 간의 내용을 전혀 알 필요가 없는 상태로서 한 모듈의 내용을 변경하더라도 다른 모듈에는 전혀 영향을 미치지 않는 가장 바람직한 결합도.
스탬프(검인) 결합도 (Stamp Coupling)	• 모듈 간의 인터페이스로 배열이나 레코드 등의 자료 구조가 전달될 때의 결합도. • 두 모듈이 동일한 자료 구조를 조회하는 경우의 결합도이며, 자료 구조의 어떠한 변화, 즉 포맷이나 구조의 변화는 그것을 조회하는 모든 모듈 및 변화되는 필드를 실제로 조회하지 않는 모듈에까지도 영향을 미치게 됨.
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. • 한 모듈이 다른 모듈의 상세한 처리 절차를 알고 있어 이를 통제하는 경우나 처리 기능이 두 모듈에 분리되어 설계된 경우에 발생함. • 하위 모듈에서 상위 모듈로 제어 신호가 이동하여 하위 모듈이 상위 모듈에게 처리 명령을 내리는 권리 전도현상이 발생하게 됨.
외부 결합도 (External Coupling)	• 어떤 모듈에서 선언한 데이터(변수)를 외부의 다른 모듈에서 참조할 때의 결합도. • 참조되는 데이터의 범위를 각 모듈에서 제한할 수 있음.
공통(공유) 결합도 (Common Coupling)	• 공유되는 공통 데이터 영역을 여러 모듈이 사용할 때의 결합도. • 공통 데이터 영역의 내용을 조금만 변경하더라도 이를 사용하는 모든 모듈에 영향을 미치므로 모듈의 독립성을 약하게 만듦.
내용 결합도 (Content Coupling)	• 한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 직접 참조하거나 수정 할 때의 결합도. • 한 모듈에서 다른 모듈의 내부로 제어가 이동하는 경우에도 내용 결합도에 해당됨.

응집도(Cohesion)

: 정보 은닉의 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
- 응집도도 정도에 따라 다음과 같이 나뉜다.

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

위로 갈수록 응집도가 강함

팬인(Fan-In) / 팬아웃(Fan-Out)
- 팬인은 어떤 모듈을 제어(호출)하는 모듈의 수를 나타냄
- 팬아웃은 어떤 모듈에 의해 제어(호출)되는 모듈의 수를 나타냄
- 팬인과 팬아웃을 분석하여 시스템의 복잡도를 알 수 있음
- 팬인이 높다는 것은 재사용 측면에서 잘 설계되었다고 볼 수 있으나, 단일 장애점(SPOF)이 발생할 수 있으므로, 중심적인 관리 및 테스트가 필요함
- 팬아웃이 높은 경우 불필요하게 다른 모듈을 호출하고 있지는 않은지, 단순화할 순 없는지 검토가 필요하다
- 시스템의 복잡도를 최소화하려면 팬인은 높게, 팬아웃은 낮게 설계해야한다.

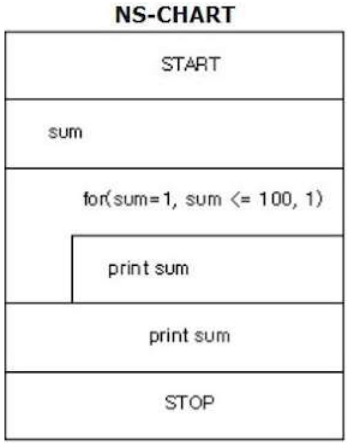
시스템 구조도에서 화살표가 뻗어나가는 방향이 팬아웃되는 모듈. 뻗어나온쪽이 팬인되는 모듈이다.

N-S 차트 : 논리의 기술에 중점을 둔 도형을 이용한 표현 방법으로 박스 다이어그램, Chapin Chart 라고도 한다.
- 연속, 선택 및 다중 선택, 반복 등의 제어 논리 구조를 표현
- GOTO나 화살표를 사용하지 않음
- 조건이 복합되어 있는 곳의 처리를 시각적으로 명확히 식별하는데 적합
- 선택과 반복 구조를 시각적으로 표현
- 이해하기 쉽고, 코드 변환 용이
- 읽기는 쉬어나 작성하기 어렵고, 임의로 제어를 전이하기 어려움
- 총체적인 구조 표현과 인터페이스를 나타내기가 어려움
- 단일 입구와 단일 출구로 표현

예시)

1부터 100의 합을 구하는 ns차트

위에서 아래로 순서가 이루어짐



섹션 26 공통 모듈

공통 모듈의 개요 : 공통 모듈은 ㅇ러 프로그램에서 공통적으로 사용할 수 있는 모듈을 말한다.
- 자주 사용되는 계산식이나 매번 필요한 사용자 인증과 같은 기능들이 공통 모듈로 구성될 수 있음
- 모듈의 재사용성 확보와 중복 개발 회피를 위해 설계 과정에서 공통 부분을 식별하고 명세를 작성할 필요가 있다.
- 다른 개발자들이 해당 모듈의 기능을 명확히 이해할 수 있도록 다음 명세 기법을 준수해야한다

1. 정확성 : 시스템 구현 시 해당 기능이 필요하다는 것을 알 수 있도록 정확히 작성
2. 명확성 : 해당 기능을 이해할 때 중의적으로 해석되지 않게 작성
3. 완전성 : 시스템 구현을 위해 필요한 모든 것을 기술한다
4. 일관성 : 공통 기능들 간 상호 충돌이 발생하지 않도록 작성한다.
5. 추적성 : 기능에 대한 요구사항의 출처, 관련 시스템 등의 관계를 파악할 수 있도록 작성한다.

재사용 : 비용과 개발 시간을 절약하기 위해 이미 개발된 기능들을 파악하고 재구성하여 새로운 시스템 또는 기능 개발에 사용하기 적합하도록 최적화 시키는 작업
- 재사용을 위해서는 누구나 이해할 수 있고 사용이 가능하도록 사용법 공개
- 재사용되는 대상은 외부 모듈과의 결합도는 낮고, 응집도는 높아야한다.(독립성이 높아야한다)
- 재사용 규모에 따른 분류

1. 함수와 객체 : 클래스나 메소드 단위의 소스 코드를 재사용한다.
2. 컴포넌트 : 독릭적인 업무 또는 기능을 수행하는 실행 코드 기반으로 작성된 모듈. 컴포넌트 자체에 대한 수정 없이 인터페이스를 통해 통신하는 방식으로 재사용한다.
3. 어플리케이션 : 공통된 기능들을 제공하는 애플리케이션을 공유하는 방식으로 재사용한다.

효과적인 모듈 설계 방안
- 결합도는 줄이고 응집도는 높여서 모듈의 독립성과 재사용성을 높임

- 모듈의 제어 영역 안에서 그 모듈의 영향 영역을 유지시킨다.
- 복잡도와 중복성을 줄이고 일관성 유지
- 모듈의 기능은 예측이 가능해야 하며 지나치게 제한적이어서는 안된다.
- 유지보수가 용이해야한다.
- 모듈 크기는 시스템의 전반적인 기능과 구조를 이해하기 쉬운 크기로 분해한다.
- 하나의 입구와 하나의 출구를 갖게 한다.
- 인덱스 번호나 기능 코드들이 전반적인 처리 구조에 예기치 못한 영향을 끼치지 않도록 모듈 인터페이스를 설계해야한다.
- 효과적인 제어를 위해 모듈 간의 계층적 관계를 정의하는 자료가 제시되어야함

섹션 27 코드

코드의 개요 : 컴퓨터를 이용하여 자료를 처리하는 과정에서 분류·조합 및 집계를 용이하게하고, 특정 자료의 추출을 쉽게 하기 위해서 사용되는 기호이다.

- 코드는 정보를 신속·정확·명료하게 전달할 수 있게 한다
- 코드는 일정한 규칙에 따라 작성되며, 정보 처리의 효율과 처리된 정보의 가치에 많은 영향을 미친다.
- 일반적인 코드의 예로 주민등록번호, 학번, 전화번호 등이 있다
- 코드의 주요 기능에는 식별기능, 분류기능, 배열기능, 표준화 기능, 간소화 기능

식별 기능	데이터 간의 성격에 따라 구분이 가능
분류 기능	특정 기준이나 동일한 유형에 해당하는 데이터를 그룹화 가능
배열 기능	의미를 부여하거나 나열할 수 있다.
표준화 기능	다양한 데이터를 기준에 맞추어 표현할 수 있다
간소화 기능	복잡한 데이터를 간소화할 수 있다.

코드의 종류

순차 코드 (Sequence)	자료의 발생 순서, 크기 순서 등 일정 기준에 따라서 최초의 자료부터 차례로 일련번호를 부여하는 방법. 순서 코드 또는 일련번호 코드라고도 한다. ex) 1, 2, 3, 4 ...
블록 코드 (Block)	코드화 대상 항목 중에서 공통성이 있는 것끼리 블록으로 구분하고, 각 블록 내에서 일련번호를 부여하는 방법으로, 구분 코드라고도 한다. ex)1001~1100: 총무부, 1101~1200 : 영업부
10진 코드 (Decimal)	코드화 대상 항목을 0~9까진 10진으로 분할하고, 다시 그 각각에 대하여 10진 분할 하는 방법을 필요한 만큼 반복하는 방법, 도서 분류식 코드라고도 한다. ex) 1000: 공학, 1100 :SW 공학
그룹 분류 코드(Group Classification)	코드화 대상 항목을 일정 기준에 따라 대분류, 중분류, 소분류 등으로 구분하고, 각 그룹 안에서 일련번호를 부여하는 방법 ex) 1-01-001 : 본사-총무부-인사계/ 2-01-001 : 지사-총무부 -인사계
연상 코드 Mnemonic	코드화 대상 항목의 명칭이나 약호와 관계있는 숫자나 문자, 기호를 이용하여 코드를 부여하는 방법 ex) TV-40 : tv 40인치
표의 숫자 코드 (Signigicant Digit)	코드화 대상 항목의 성질, 즉 길이, 넓이, 부피, 지름, 높이 등의 물리적 수치를 그대로 코드에 적용시키는 방법으로, 유효 숫자 코드라고도 한다. ex) 120-720-1500 : 두께X폭X길이인 강판
합성 코드 Combined	필요한 기능을 하나의 코드로 수행하기 어려운 경우 2개 이상의 코드를 조합하여 만드는 경우 ex) KE-711 : 대한 항공 711기

코드 부여 체계

: 이름만으로 개체(모듈, 컴포너느, 인터페이스)의 용도와 적용 범위를 알 수 있도록 코드를 부여하는 방식을 말한다.

- 코드 부여 체계는 각 개체에 유일한 코드를 부여하여 개체들의 식별 및 추출을 용이하게 함
- 코드를 부여하기 전에 각 단위 시스템의 고유한 코드와 개체를 나타내는코드 등이 정의되어야 한다.
- 코드 부여 체계를 담당하는 자는 코드의 자릿수와 구분자, 구조 등을 상세하게 명시해야 한다.

예)

자리수	구분자를 포함한 11자리
기본 구조	AAA-MOD-000
상세 구조	AAA • 영문 및 숫자 3자리 • 단위 시스템의 코드 3자리 • 전체 시스템의 경우 'PJC' 고정 000 • 숫자 3자리 • 순차적 일련번호 001~999 MOD • 영문 3자리 • 모듈은 MOD, 공통 모듈은 COM을 사용

예) 코드 부여 체계에 따른 코드 작성

- PJC-COM-003 : 전체 시스템 단위의 3번째 공통 모듈
- PY3-MOD-010 : PY3이라는 단위 시스템의 10번째 모듈

섹션 28 디자인 패턴

디자인 패턴의 개요 : 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 구성되 있다.
- 바퀴를 다시 발명하지 마라(Don't reinvent the wheel) 라는 말과 같이, 개발 중에 문제가 발생하면 새로 해결책을 구상하는 것보다 문제에 해당하는 다자인 패턴을 참고하여 적용하는 것이 더 효율적이다
- 디자인 패턴은 한 패턴에 변형을 가하거나 특정 요구사항을 반영하면 유사한 형태의 다른 디자인 패턴으로 변화되는 특징이있다
- 디자인 패턴은 1995년 GoF라 불리는 에릭 감마, 리차드 헬름, 랄프 존슨, 존 브리시디스가 처음으로 구체화 및 체계화하였다.
- GoF의 다자인 패턴은 수많은 디자인 패턴들 중 가장 일반적인 사례에 적용될 수 있는 패턴들을 분류하여 정리함으로써, 지금까지도 소프트웨어 공학이나 ㅎㄴ업에서 가장 많이 사용되는 디자인 패턴이다.
- GoF의 패턴은 유형에 따라 생성패턴 5개, 구조 패턴 7개, 행위 패턴 11개로 총 23개의 패턴으로 구성된다.

아키텍처 패턴 vs 디자인 패턴

- 이 두 패턴 모두 소프트웨어 설계를 위한 참조 모델이다.
- 그러나 아키텍처 패턴은 디자인 패턴보다 상위 수준의 설계에 사용된다.
- 아키텍처 패턴이 전체 시스템의 구조를 설계하기 위한 참조 모델이라면, 디자인 패턴은 서브시스템에 속하는 컴포넌트들과 그 관계를 설계하기 위한 참조 모델
- 몇몇 디자인 패턴은 특정 아키텍처 패턴을 구현하는데 유용하게 사용됨

디자인 패턴 사용의 장·단점

- 범용적인 코딩 스타일로 인해 구조 파악이 용이
- 객체지향 설계 및 구현의 생산성을 높이는 데 적합
- 검즈된 구조의 재사용을 통해 개발 시간과 비용이 절약
- 초기 투자 비용이 부담될 수 있음
- 개발자 간의 원활한 의사소통이 가능하다
- 실제 변경 요청에 대한 유연한 대처가 가능하다
- 객체지향을 기반으로 한 설계와 구현을 다루므로 다른 기반의 애플리케이션 케발에는 적합하지 않다.

생성 패턴 : 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 표현. 연관된 서브 클래스를 묶어 한번에 교체가 가능
빌더 (Builder)	작게 분리된 인스턴스를 건축 하듯이 조합하여 객체 생성. 객체의 생성과 표현 방법을 분리하고 있어, 동일한 객체 생성에도 서로 다른 결과를 만들어낼 수 있다.
팩토리 메소드 (Factory Method)	객체 생성을 서브 클래스에서 처리하도록 분리하여 캡슐화한 패턴. 상위 클래스에서 인터페이스만 정의하고 실제 생성은 서브 클래스가 담당. 가상 생성자 패턴이라고 함
프로토 타입	원본 객체를 복제하여 객체를 생성하는 패턴. 일반적인 방법으로 객체를 생성하며, 비용이 큰 경우 주로 이용
싱글톤 (Singleton)	하나의 객체를 생성하면 생성된 객체를 어디서든 참조할 수 있지만, 여러 프로세스가 동시 참조할 수 없음. 클래스 내에서 인스턴스가 오직 하나임을 보장하여 메모리 누수 최소화

구조 패턴 : 클래스나 객체를 조합하여 더 큰 구조로 만들 수 있게 해주는 패턴

어댑터 (Adapter)	호환성이 없는 클래스들의 인터페이스를 다른 클래스가 이용할 수 있도록 변환해주는 패턴. 기존의 클래스를 이용하고 싶지만 인터페이스가 일치하지 않을 때 이용
브리지 (Bridge)	구현부에서 추상층을 분리하여, 서로가 독립적으로 확장할 수 있도록 구성한 패턴. 기능과 구현을 두 별도의 클래스로 구현
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)	객체간의 결합을 통해 능동적으로 기능을 확장할 수 있는 패턴.임의에 객체에 부가적인 기능을 추가하기 위해 다른 객체들을 덧붙이는 방식으로 구현한다.

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 생성되는 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, 메모리 대용량 객체로의 접근 등에 주로 이용

행위패턴 (Behavioral Pattern)

책임 연쇄 (Chain of Responsibility)	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)형태로 묶여 있어 요청이 해결될 때까지 고리를 따라 책임이 넘어감
커맨드 (Command)	요청을 객체의 형태로 캡슐화하여 재이용하거나 취소할 수 있도록 요청에 필요한 정보를 저장하거나 로그에 남김. 요청에 사용되는 각종 명령어들을 추상 클래스와 구체 클래스로 분리하여 단순화함.
인터프리터 (Interpreter)	언어에 문법 표현을 정의하는 패턴. SQL이나 통신 프로토콜과 같은 것을 개발할 때 사용
반복자 (Iterator)	자료 구조와 같이 접근이 잦은 객체에 대해 동일한 인터페이스를 사용하도록 하는 패턴. 내부 표현 방법의 노출 없이 순차적인 접근이 가능
중재자 (Mediator)	수많은 객체들 간의 복잡한 상호작용을 캡슐화하여 객체로 정의하는 패턴. 객체 사이의 의존성을 줄여 결합도를 감소시킬 수 있음. 중재자는 객체 간의 통제나 지시의 역할을 수행
메멘토 (Memento)	특정 시점에서의 객체 내부 상태를 객체화함로써 이후 요청에 따라 객체를 해당 시점의 상태로 돌릴 수 있는 기능을 제공하는 패턴
옵서버 (Observer)	한 객체의 상태가 변화하면 객체에 상속되어 있는 다른 객체들에게 변화된 상태를 전달하는 패턴. 주로 분산된 시스템 간에 이벤트를 생성·발행하고, 이를 수신해야할 때 이용
상태 (State)	객체의 상태에 따라 동일한 동작을 다르게 처리할 때 사용하는 패턴. 객체 상태를 캡슐화하고 이를 참조하는 방식
전략 (Strategy)	동일한 곌ㄹ의 알고리즘들을 개별적으로 캡슐화하여 상호 교환할 수 있게 정의하는 패턴. 클라이언트는 독립적으로 원하는 알고리즘ㅇ르 선택해 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 함
방문자 (Visitor)	각 클래스들의 데이터 구조에서 처리 기능을 분리하여 별도의 클래스로 구성하는 패턴. 분리된 처리 기능은 각 클래스를 방문하여 수행

4장 – 인터페이스 설계

섹션 29 시스템 인터페이스 요구사항 분석 (C등급)

시스템 인터페이스는 독립적으로 떨어져 있는 시스템들끼리 서로 연동하여 상호작용하기 위한 접속 방법이나 규칙을 의미한다.

– 시스템 인터페이스 요구사항은 개발을 목표로 하는 시스템과 외부 시스템을 연동하는데 필요한 시스템 인터페이스에 대한 요구사항을 기술한 것

– 시스템 인터페이스 요구사항 명세서에는 인터페이스 이름, ds계 대상 시스템, 연계 범위 및 내용, 연계 방식, 송신 데이터, 인터페이스 주기. 기타 고려사항 등이 포함되어야 한다.

시스템 인터페어스 요구사항 분석

: 시스템 인터페이스 요구사항 분석은 요구사항 명세서에서 요구사항을 기능적 요구사항과 비기능적 요구사항으로 분류하고 조직화하여 요구사항 명세를 구체화하고 이를 이해관계자에게 전달하는 일련의 과정.

– 요구사항은 분석은 소프트웨어 요구사항 분석 기법을 적절히 이용한다.
– 요구사항의 분해가 필요한 경우 적절한 수준으로 세분화한다.
– 요구사항 분석 시 누락된 요구사항이나 제한조건을 추가
– 요구사항에 대한 상대적 중요도를 평가하여 우선순위 부여

시스템 인터페이스 요구사항 분석 절차

1. 소프트웨어 요구사항 목록에서 시스템 인터페이스 관련 요구사항으르 선별하여 별도로시스템 인터페이스 요구사항 목록을 만듬
2. 시스템 인터페이스와 관련된 요구사항 및 아키텍처 정의서, 현행 시스템의 대·내외 연계 시스템 현황 자료 등 시스템 인터페이스 요구사항과 관련된 자료를 준비한다.
3. 시스템인터페이스에 대한 요구사항 명세서를 확인하여 기능적인 요구살아과 비기능적인 요구사항으로 분류
4. 시스템 인터페이스 요구사항 명세서와 시스템 인터페이스 요구사항 목록 및 기타 관련 자료들을 비교하여 요구사항을 분석하고 내용을 추가하거나 수정한다
5. 추가·수정한 시스템

인터페이스 요구사항 명세서와 시스템 인터페이스 요구사항 목록을 관련 이해관계자에게 전달한다.

섹션 30 인터페이스 요구사항 검증

요구사항 검증은 인터페이스의 설계 및 구현 전에 사용자들의 요구사항이 요구사항 명세서에 정확하고 완전하게 기술되었는지 검토하고 개발 범위의 기준인 베이스라인을 설정하는 것

- 인터페이스의 설계 및 구현 중에 요구사항 명세서의 오류가 발견되어 이를 수정할 경우 많은 비용이 소모되므로 프로젝트에서 요구사항 검증은 매우 중요
- 요구사항 검토 계획 수립 -〉 검토 및 오류 수정 -〉 베이스라인 설정 순으로 수행한다.

1. **인터페이스 요구사항 검토 계획 수립** : 프로젝트 이해관계자들이 프로젝트 품질 관리 계획을 참조하여 다음과 같이 인터페이스 요구사항 검토 계획을 수립한다.
   - 검토 기준 및 방법, 참여자, 체크리스트 , 관련자료, 일정 등으로 계획 수립
2. **인터페이스 요구사항 검토 및 오류 수정** : 검토 체크리스트 항목에 ᄯᅡ라 인터페이스 요구사항 명세서를 검토하는 것.
   - 오류가 발견되면 오류 수정할 수 있도록 오류 목록과 시정 조치서 작성
   - 오류 수정과 요구사항 승인 절차를 진행할 수 있도록 요구사항 검토 결과를 검토 관련자들에게 전달
3. **인터페이스 요구사항 베이스라인 설정 :** 인터페이스 요구사항 검토를 통해 검증된 인터페이스 요구사항은 프로젝트 관리자의 주요 의사 결정자에게 승인들 맡음
   - 소프트웨어 설계 및 구현을 위해 요구사항 명세서의 베이스라인(기준)을 설정

요구사항 검증 방법 (중요!!)

: 인터페이스 요구사항 검증은 다음과 같은 검증 방법을 적절하게 이용하낟.

○요구사항 검토 : 요구사항 명세서의 오류 확인 및 표준 준수 여부 등의 결함 여부를 검토 담당자들이 수작업으로 분석하는 방법. 동료검토, 워크스루, 인스펙션이 있음

1. 동료 검토(Peer Review) : 요구사항 명세서 작성자가 명세서 내용을 직접 설명하고 동료들이 이를 들으면서 결함을 발견하는 방법
2. 워크 스루 (Walk Through) : 검토 회의 전에 요구사항 명세서를 미리 배포하여 사전 검토한 후 짧은 검토 회의를 통해 결함을 발견하는 방법
3. 인스펙션 (Inspection) : 요구사항 명세서 작성자를 제외한 다른 검토 전문가들이 명세서를 확인하여 결함을 발견하는 것

○프로토타이핑 : 사용자의 요구사항을 정확하게 파악하기 위해 실제 개발될 소프트웨어에 대한 프로토타입을 만들어 최종 결과물을 예측

○테스트 설계 : 요구사항은 테스트할 수 있도록 작성되어야 하며, 이를 위해 test case를 생성하여 이후에 요구사항이 현실적으로 테스트 가능한지 검토

○CASE (Computer Aided Software Engineering) 도구 활용 : 일관성 분석 (Consistency Analysis)를 통해 요구사항 변경사항의 추적 및 분석, 관리하고, 표준 준수 여부를 확인한다.

인터페이스 요구사항 검증의 주요 항목

1. 완전성 : 사용자의 모든 요구사항이 누락되지 않고 완전하게 반영되어 있는가?
2. 일관성 : 요구사항이 모순되거나 충돌되는 점없이 일관성을 유지하나?
3. 명확성 : 모든 참여자가 요구사항을 명확히 이해할 수 있는가
4. 기능성 :요구사항이 어떻게 보다 무엇을 에 중점을 두고 있는가
5. 검증 가능성 : 요구사항이 사용자의 요구를 모두 만족하고, 개발된 소프트웨어가 사용자의 요구 내용과 일치하는 지를 검증할 수 있는가?
6. 추적 가능성 : 요구사항 명세서와 설계서를 추적할 수 있는가
7. 변경 용이성 : 요구사항 명세서의 변경이 쉽도록 작성되었는가?

섹션 31 인터페이스 시스템 식별 (C등급)

1. 개발 시스템 식별 : 개발 시스템을 식별하는 것은 인터페이스 관련 자료들을 기반으로 개발하고자 하는 시스템의 상세 식별 정보를 정의하고 목록을 작성하는 것.
2. 내·외부 시스템 식별 : 인터페이스 관련 자료들을 기반으로 개발할 시스템과 연계할 내·외부 세스템들의 상세 정보를 정의하고 목록을 작성하는 것
3. 내·외부 시스템 환경 및 관리 주체 식별 : 연계할 시스템 접속에 필요한 IP 또는 URL, Port 정보 등 시스템의 실제 운용 환경을 의미
4. 내·외부 시스템 네트워크 연결 정보 식별 : 시스템 로그인 및 DB 정보
5. 인터페이스 식별 : 인터페이스 요구사항 명세서와 인터페이스 요구사항 기록을 기반으로 개발할 시스템과 이와 연계할 내·외부 시스템 사이의 인터페이스를 식별하고 인터페이스 목록을 작성하는 것이다.
6. 인터페이스 시스템 식별 : 인터페이스 시스템을 식별하는 것은 인터페이스별로 인터페이스에 참여하는 시스템들을 송신/수신을 구분하여 작성하는 것이다.

섹션 32 송·수신 데이터 식별 (D등급)

식별 대상 데이터 : 식별 대상 데이터는 송·수신 시스템 사이에서 교환되는 데이터로, 규격화된 표준 형식에 따라 전송

- 교환되는 데이터의 종류에는 인터페이스 표준 항목, 송·수신 데이터 항목, 공통코드가 있다.

1. 인터페이스 표준 항목 : 송수신 시스템을 연계하는데 표준적으로 필요한 데이터를 의미한다. 시스템 공통부/ 거래 공통부로 나뉨

시스템 공통부	시스템 간 연동 시 필요한 공통 정보. 구성 정보에는 인터페이스 ID, 전송 시스템 정보, 서비스 코드 정보, 응답 결과 정보, 장애 정보 등이 있음
거래 공통부	시스템들이 연동된 후 송·수신 되는 데이터를 처리할 때 필요한 정보이다. 구성 정보에는 직원 정보, 승인자 정보, 기기 정보, 매체 정보

2. 송·수신 데이터 항목 : 송·수신 업무를 수행하는 데 사용하는 데이터. 전송되는 데이터 항복과 순서는 인터페이스별로 다르다.
3. 공통 코드 : 시스템이 공통적으로 사용하는 코드. 연계 시스템이나 SW에서 사용하는 상태 및 오류 코드와 같은 것들을 공통으로 관리

정보 흐름 식별 : 개발할 시스템과 내·외부 시스템 사이에서 전송되는 정보들의 방향성을 식별하는 것

- 개발할 시스템과 내·외부 시스템에 대한 각각의 인터페이스 목록을 확인하여 정보 흐름을 식별한다.
- 식별한 정보 흐름을 기반으로 송·수신 시스템 사이에서 교환되는 주요 데이터 항목이나 정보 그룹을 도출한다.

송·수신 데이터 식별 : 개발할 시스템과 연계할 내·외부 시스템 사이의 정보 흐름과 데이터베이스 산출물을 기반으로 송·수신 데이터를 식별한다.

- 송수신 데이터의 종류에는 인터페이스 표준 항목에 대한 데이터 항목과 코드성 데이터 항목이 있다.

1. 인터페이스 표준 항목과 송수신 데이터 항목 식별 : 송·수신 시스템 사이의 교환 범위를 확인하고 인터페이스 표준 항목에 대해 송·수신 데이터 항목을 식별한다.
2. 코드성 데이터 항목 식별 : 코드, 코드명, 코드설명 등의 코드 정보를 식별

- 코드성 데이터 항목에 대해 송신 시스템에서 사용하는 코드 정보와 수신 스스템에서 사용하는 코드 정보가 동일한 경우 공통 코드 정보를 확보하고, 다른 경우 매핑 필요 대상으로 분류하여 양쪽 시스템에서 사용하는 코드 정보를 확보한다.

섹션 33 인터페이스 방법 명세화

인터페이스 방법 명세화는 내·외부 시스템이 연계하여 작동할 때 인터페이스별 송·수신 방법, 송·수신 데이터, 오류 식별 및 처리 방안에 대한 내용을 문서로 명확하게 정리하는 방법이다.

- 인터페이스별로 송·수신을 명세화하기 위해서는 시스템 연계 기술, 인터페이스 통신 유형, 처리 유형, 발생 주기 등에 대한 정보가 필요하다.

시스템 연계 기술 : 개발할 시스템과 내·외부 시스템을 연계할 떄 사용되는 기술.

1. DB-Link : DB에서 제공하는 DB Link 객체를 이용하는 방식이다.
2. API/Open API : 송신 시스템의 데이터베이스에서 데이터를 읽어 와 제공하는 애플리케이션 프로그래밍 인터페이스 프로그램이다.
3. 연계 솔류션 : EAI 서버와 송·수신 시스템에 설치되는 클라이언트를 이용하는 방식
4. Socket : 서버는 통신을 위한 소켓을 생성하여 포트를 할당하고 클라이언트의

통신 요청 시 클라이언트와 연결하여 통신하는 네트워크 기술
5. Web Service : 웹 서비스에서 WSDL과 UDDI, SOAP 프로토콜을 이용하여 연계하는 서비스이다.

인터페이스 통신 유형 : 개발할 시스템과 내·외부 시스템 간 데이터를 송·수신하는 형태를 의미한다.
- 인터페이스 통신 유형에는 단방향, 동기 비동기 방식 등이 있다.
1. 단방향 : 시스템에서 거래만 요청하고 응답은 없는 방식
2. 동기 : 시스템에서 거래를 요청하고 응답이 올 때까지 대기하는 방식
3. 비동기 : 시스템에서 거래를 요청하고 다른 작업을 수행하다 응답이오면 처리하는 형식

인터페이스 처리 유형 : 송·수신 데이터를 어떤 형태로 처리할 것인지에 대한 방식
- 엄무의 성격과 송·수신 데이터 전송량을 고려하여 실시간, 지연처리, 배치 방식으로 구분한다.
1. 실시간 방식 : 사용자가 요청한 내용을 바로 처리
2. 지연 처리 방식 : 데이터를 매건 단위로 처리할 경우 비용이 많이 발생할 때 사용하는 방식
3. 대량의 데이터를 처리할 때 사용하는 방식

인터페이스 발생 주기 : 개발할 시스템과 내·외부 간 송·수신 데이터가 전송되어 인터페이스가 사용되는 주기.
- 업무의 성격과 송·수신 데이터 전송량을 고려하여 매일, 수시, 주1회등으로 구분

송·수신 방법 명세화
송·수신 방법 명세화는 내·외부 인터페이스 목록에 있는 각각의 인터페이스에 대해 연계 방식, 통신 및 처리 유형, 발생 주기 등의 송·수신 방법을 저의하고 명세를 작성하는 것이다.

송·수신 데이터 명세화
송·수신 데이터 명세화는 내·외부 인터페이스 목록에 있는 각각의 인터페이스에 대해 인터페이스 시 필요한 송·수신 데이터에 대한 명세를 작성하는 것이다.
- 인터페이스별로 테이블 정의서와 파일 레이아웃에서 연계하고자 하는 테이블 또는 파일 단위로 송·수신 데이터에 대한 명세를 작성한다.

오류 식별 및 처리 방안 명세화
오류 식별 및 처리 방안 명세화는 내·외부 인터페이스 목록에 있는 각각의 인터페이스에 대해 인터페이스 시 발생할 수 있는 오류를 식별하고 오류 처리 방안에 대한 명세를 작성하는 것이다.
- 시스템 및 전송 오류, 연계 프로그램 등에서 정의한 예외 상황 등 대·내외 시스템 연계 시 발생할 수 있는 다양한 오류 상황을 식별하고 분류한다.
- 오류 상황에 대해 오류 코드, 오류 메시지, 오류 설명, 해결 방법등을 명세화한다.

섹션 34 시스템 인터페이스 설계서 작성 (D등급)
시스템 인터페이스 설계서의 개요 : 시스템 인터페이스 설계서는 시스템의 인터페이스 현황을 확인하기 위해 시스템이 갖는 인터페이스 몰록과 각 인터페이스의 상세 데이터 명세를 작성한 것이다.
- 시스템 인터페이스 설계서는 시스템 인터페이스 목록과 시스템 인터페이스 정의서로 구성된다.

시스템 인터페이스 목록 작성 : 엄무 시스템과 내외부 시스템 간 데이터를 주고받는 경우에 사용하는 인터페이스에 대해 기술

시스템 인터페이스 정의서 작성 : 인터페이습ㄹ로 시스템 간의 연계를 위해 필요한 데이터 항목 및 구현 요건 등을 기술

섹션 35 미들웨어 솔루션 명세
미들웨어는 미들(Middle)과 소프트웨어의 합성어이다.
- 분산 컴퓨팅 환경에서 서로 다른 기종 간의 하드웨어나 프로토콜, 통신 환경 등을 연결할여 운영체제와 응용 프로그램, 또는 서버와 클라이언트 사이에서 원만한 통신이 이루어지도록 다양한 서비스를 제공한다
- 표준화된 인터페이스르 제공함으로ㅆ 시스템 간의 데이터 교환에 일관성을 보장한다.

- 위치 투명성을 제공한다. (시스템의 논리적인 명칭만으로 시스템에 액세스할 수 있는 것)
- 사용자가 미들웨어의 내부 동작을 확인하려면 별도의 응용 소프트웨어ㄴ를 사용해야한다.
- 시스템들을 1:1 , 1:N , N:M 등 여러 가지 형태로 연결할 수 있다.
- 종류 : DB, RPC, MOM, TP-Monitor, ORB, WAS

1. DB (DataBase) : 데이터베이스 벤더에서 제공하는 클라이언트에서 원격의 데이터베이스와 연동하기 위한 미들웨어이다.
- DB를 사용하여 시스템을 구축하는 경우 보통 2-Tier 아키텍처라 한다.
- 대표적인 종류에는 ODBC, IDAPI, Glue 등이 있다.

2. RPC(Remote Procedure Call; 원격 프로시저 호출) : 응용 프로그램의 프로시저를 이용하여 원격 프로시저를 마치 로컬 프로시저처럼 호출하는 방식의 미들웨어
- 종류는 ENtera. ONC/RPC 가 있음

3. MOM(Message Oriented Middleware ; 메시지 지향 미들웨어) : 메시지 기반의 비동기형 메시지를 전달하는 방식의 미들웨어.
- 온라인 업무보다는 이기종 분산 데이터 시스템의 데이터 동기를 위해 사용
- 서로 다른 플랫폼에서 독립적으로 실행되는 소프트웨어 간의 상호 작용을 통해 하나의 통합된 시스템처럼 동작한다.
- 대표적으로 MQ, Message Q, JMS 등이 있음

4. TP-Monitor(Transaction Processing Monitor; 트랜젝션 처리 모니터) : 항공기나 철도 예약 업무 등과 같은 온라인 트랜잭션 업무에서 트랜잭션을 처리 및 감시하는 미들웨어이다.
- 사용자 수가 증가해도 빠른 응답 속도를 유지해야 하는 업무에 주로 사용
- 종류에는 오라클의 tuxedo, tmax 등이 있다

5. ORB(Object Request Broker; 객체 요청 브로커) : 객체 지향 미들웨어로 코바(CORBA) 표준 스펙을 구현한 미들웨어이다.
- 최근에는 TP-Monitor의 장점인 트랜잭션 처리와 모니터링 등을 추가로 구현한 제품도 있다.
- 종류는 Orbix, CORBA 등이 있다.

6. WAS(Web Appication Server) : 정적인 콘텐츠를 처리하는 웹서버와 달리 사용자의 요구에 따라 변하는 동적인 콘텐츠를 처리하기 위해 사용되는 미들웨어
- 클라이언트/서버 환경 보다는 웹 환경을 구현하기 위한 미들웨어이다.
- HTTP 세션 처리를 위한 웹 서버 기능뿐만 아니라 미션-크리티컬한ㅇ 기업 업무까지 JAVA,EJB 컴포넌트 기반으로 구현이 가능하다.
- 종류에는 WebLogic, WebSphere 등이 있다.

미들웨어 솔루션 식별 : 개발 및 운영 환경에 사용될 미들웨어 솔루션을 확인하고 목록을 작성하는 것이다.
- 소프트웨어 아키텍처에서 정의한 아키텍처 구성 정보와 프로젝트에서 구매가 진행 중이거나 구매 예정인 소프트웨어 내역을 확인하여 개발 및 운영 환경에서 사용될 미들웨어 솔루션을 식별한다.
- 식별한 미들웨어 솔루션들에 대해 솔루션의 시스템, 구분, 솔루션명, 버전, 제조사등의 정보를 정리한 미들웨어 솔루션 목록을 작성한다.
- 작성된 미들웨어 솔루션 목록은 이해관계자 등에게 전달하여 오류 및 누락을 확인하고 수정한다.

미들웨어 솔루션 명세서 작성 : 미들웨서 솔루션 별로 관련 정보들을 상세하게 기술하는 것