1. 소프트웨어 공학의 개념

소프트웨어

소프트웨어(SW)의 개념

컴퓨터를 동작시키고 어떤 일의 처리 순서와 방법을 지시하는 명령어의 집합인 프로그램과 프로그램의 수행에 필요한 절차, 규칙, 관련 문서 등을 총칭한다.

프로그램(Program) : 컴퓨터를 통해 일련의 작업을 처리하기 위한 명령어와 관련된 데이터의 집합을 의미한다.

자료 구조(Data Structure): 컴퓨터 기억 장치 내에 자료의 표현, 처리, 저장 방법 등을 총칭하는 것으로 데이터 간의 논리적 관계나 처리 알고리즘을 의미한다.

문서(Paper) : 소프트웨어를 개발함에 있어서 사용자 설명서, 소프트웨어 요구분석서, 평가서, 명세서, 프로젝트 계획서, 검사 계획서 등을 의미한다.

소프트웨어의 특징

상품성 : 소프트웨어를 개발하면 상품이 되어 판매가 된다.

복잡성 : 개발하는 과정이 복잡하고 관리가 어렵다.

변경 가능성 : 프로그램을 일부 수정하여 업그레이드 및 오류 수정 등을 할 수 있다.

복제성 : 복제가 용이해 쉽게 복사, 유통이 가능하다.

시스템(System)의 개요와 기본 요소

시스템의 개요

컴퓨터로 처리 가능한 자료를 입력하고 저장, 처리, 가공해 출력할 수 있도록 설계/구현된 정보 체계를 의미한다.

하나의 목적을 위해 다양한 요소가 유기적으로 결합된 것을 의미한다.

기본 요소

입력, 처리, 출력, 제어, 피드백으로 구성된다.

소프트웨어 위기(Software Crisis)

컴퓨터의 발달 과정에서 소프트웨어의 개발 속도가 하드웨어의 개발 속도를 따라가지 못해 사용자들의 요구사항을 감당할 수 없는 문제가 발생함을 의미한다.

소프트웨어 위기의 원인

하드웨어 비용을 초과하는 개발 비용의 증가

개발 기간의 지연

개발 인력 부족 및 인건비 상승

성능 및 신뢰성 부족

유지보수의 어려움에 따른 엄청난 비용

소프트웨어 공학(Software Engineering)

소프트웨어 공학의 이해

경제적으로 신뢰도 높은 소프트웨어를 만들기 위한 방법, 도구와 절차들의 체계를 말한다.
IEEEE(전기/전자기술협회)는 소프트웨어의 개발, 운용, 유지보수 및 파기에 대한 체계적인 접근 방법이라 정의하였다.
소프트웨어 공학의 기본 원칙

현대적인 프로그래밍 기술을 적용해야 한다.
신뢰성이 높아야 한다.
사용의 편리성과 유지보수성이 높아야 한다.
지속적인 검증 시행을 해야 한다.
소프트웨어 공학 계층 구조

도구 : 프로세스와 방법을 처리하는 기능을 제공하는 것이다.
방법론 : 소프트웨어를 설계하는데 기술적인 방법을 제공하는 것이다.
프로세스 : 소프트웨어의 가장 기초가 되며 개발에 사용되는 방법론과 도구가 적용되는 순서를 의미한다.
소프트웨어 품질

사용자의 요구대로 만들어져야 한다.
유지보수가 쉬워야 한다.
에러를 최소화해야 한다.
초반에 정한 비용에 맞춰 개발해야 한다.
정확한 결과가 도출되어야 한다.
원하는 시간에 원하는 기능을 수행할 수 있어야 한다.
소프트웨어 공학의 목표

소프트웨어의 생산성과 품질을 향상시킨다.
최소의 비용으로 단기간에 시스템에 적합한 소프트웨어를 개발하는 것이 소프트웨어 공학의 궁극적 목적이다.
2. 재공학
재공학
소프트웨어 재공학(Software Reengineering)의 개념

소프트웨어 위기를 개발의 생산성이 아닌 유지보수의 생산성으로 해결하려는 방법을 의미한다.
현재의 시스템을 변경하거나 재구조화(Restructuring)하는 것이다.

재구조화는 재공학의 한 유형으로 사용자의 요구사항이나 기술적 설계의 변경 없이 프로그램을 개선하는 것이다.
소프트웨어 재공학 관점에서 가장 연관 깊은 유지보수 유형은 예방 유지보수(Preventive Maintenance)이다.
재공학의 장점, 목표, 과정

장점

개발 시간 및 비용 감소
품질 향상
생산성 향상
신뢰성 향상
구축 방법에 대한 지식의 공유
프로젝트 실패 위험 감소
목표

소프트웨어의 유지보수성 향상이 최우선
복잡한 시스템을 다루는 방법 구현, 다른 뷰의 생성, 잃어버린 정보의 복구 및 제거
재사용을 수월하게 하며 소프트웨어의 수명을 연장하기 위함.
과정

분석(Analysis) => 구성(Restructuring) => 역공학(Reverse Engineering) => 이식(Migration)
역공학
역공학의 개념

소프트웨어를 분석하여 소프트웨어 개발 과정과 데이터 처리 과정을 설명하는 분석 및 설계 정보를 재발견하거나 다시 만들어내는 작업이다.
역공학의 가장 간단하고 오래된 형태는 재문서화라고 할 수 있다.
CASE(Computer Aided Software Engineering)
CASE

소프트웨어 개발 과정에서 사용되는 요구분석, 설계, 구현, 검사 및 디버깅 과정을 컴퓨터와 전용 소프트웨어 도구를 사용하여 자동화하는 작업이다.
자료 흐름도 등의 다이어그램을 쉽게 작성하게 해주는 소프트웨어 CASE 도구이다.
작업 과정 및 데이터 공유를 통해 작업자 간의 커뮤니케이션을 증대한다.
CASE 가 제공하는 기능

개발을 신속하게 할 수 있고, 오류 수정이 쉬워 소프트웨어 품질이 향상된다.
소프트웨어 생명주기의 전체 단계를 연결해 주고 자동화시켜 주는 통합된 도구를 제공해주는 기술이다.
소프트웨어 시스템의 문서화 및 명세화를 위한 그래픽 기능을 제공한다.
소프트웨어 개발 단계의 표준화를 기할 수 있으며, 자료 흐름도 작성 기능을 제공한다.
모델들 사이의 모순 검사 기능을 제공하며 다양한 소프트웨어 개발 모형을 지원한다.
원천 기술 : 구조적 기법, 프로토타이핑 기술, 정보 저장소 기술
CASE 사용의 장점

소프트웨어 개발 기간 단축 및 개발 비용을 절약하여 소프트웨어 생산성을 향상시킨다.
자동화된 검사를 통해 소프트웨어 품질이 향상된다.
프로그램의 유지보수가 간편해지고 소프트웨어 모듈의 재사용성이 향상된다.
소프트웨어 개발 주기의 표준안 확입, 소프트웨어 개발 기법의 실용화, 문서화의 용이성 제공, 시스템 수정 및 유지보수 축소 등의 효과를 얻을 수 있다.
CASE 의 분류

상위(Upper) CASE : 요구분석 및 설계 단계 지원
하위(Lower) CASE : 소스 코드 작성, 테스트, 문서화 과정 지원
통합(Integrate) CASE : 소프트웨어 개발 주기 전체 과정 지원
요구사항 분석을 위한 CASE 도구
요구사항 분석을 위한 CASE

요구사항을 자동으로 분석하고, 요구사항 분석 명세서를 기술하도록 개발된 도구를 의미한다.
표준화와 보고를 통한 문서화 품질 개선, 변경이 주는 영향 추적의 용이성, 명세에 대한 유지보수 비용 축소, 교차 참조도와 보고서를 통한 결함, 생략, 불일치 등의 발견 용이성 등의 특징을 갖는다.
DB 가 모두에게 이용 가능하다는 점에서 분석자들 간의 적절한 조정 기능을 제공한다.
요구사항 분석을 위한 CASE 도구

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. 구조적 요구분석을 하기 위해 블록 다이어그램을 채택한 자동화 도구다.
REM(Software Requirements Engineering Methodolohy) = RSL/REVS : TRW 사가 우주 국방 시스템 그룹에 의해 실시간 처리 소프트웨어 시스템에서 요구사항을 명확히 기술하도록 할 목적으로 개발한 것으로, RSL 과 REVS 를 사용하는 자동화 도구이다.

RSL(Requirement Statement Language) : 요소, 속성, 관계, 구조들을 기술하는 요구사항 기술 언어이다.

REVS(Requirement Engineering and Validation System) : RSL 로 기술된 요구사항들을 자동으로 분석하여 요구사항 분석 명세서를 출력하는 요구사항 분석기이다.

3. 소프트웨어 개발 방법론

소프트웨어 설계 방법론

소프트웨어 생명주기(Software Life Cycle)

소프트웨어 제품의 개념 형성에서 시작하여 운용/유지보수에 이르기까지 변화의 모든 과정이다.

타당성 검토 => 개발 계획 => 요구사항 분석 => 설계 => 구현 => 테스트 => 운용 => 유지보수

폭포수 모형(Waterfall Model)의 개요

선형 순차적 모델이라고도 하며 Boehm 이 제시한 고전적 생명주기 모형으로, 소프트웨어 개발 과정의 각 단계가 순차적으로 진행되는 모형이다.

나선형 모형(Spiral Model)

Boehm 이 제시하였으며, 반복적인 작업을 수행하는 점증적 생명주기 모형이다.

점증적 모형, 집중적 모형이라고도 하며 유지보수 과정이 필요 없다.

소프트웨어 개발 중 발생할 수 있는 위험을 관리하고 최소화하는 것이 목적이다.

나선을 따라서 돌아가면서 각 개발 순서를 반복하여 수행하는 점진적 방식으로 누락된 요구사항을 추가할 수 있다.

하향식과 상향식 설계

하향식 설계 : 소프트웨어 설계 시 제일 상위에 있는 Main User Function 에서 시작하여 기능을 하위 기능들로 나눠 가면서 설계하는 방식이다.

상향식 설계 : 가장 기본적인 컴포넌트를 먼저 설계한 다음 이것을 사용하는 상위 수준의 컴포넌트를 설계하는 방식이다.

프로토타입 모형(Prototype Model)의 개요

실제 개발될 시스템의 견본(Prototype)을 미리 만들어 최종 결과물을 예측하는 모형이다.

개발이 완료되고 나서 사용을 하면 문제점을 알 수 있는 폭포수 모형의 단점을 보완하기 위한 모형으로 요구사항을 충실히 반영할 수 있다.

HIPO(Hierarchy Input Process Output)

입력, 처리, 출력으로 구성되는 시스템 분석 및 설계와 시스템 문서화용 기법이다.

일반적으로 가시적 도표(Visual Table of Contents), 총체적 다이어그램(Overview Diagram), 세부적 다이어그램(Detail Diagram)으로 구성된다.
구조도(가시적 도표, Visual Table of Contents), 개요, 도표(Index Diagram), 상세 도표(Detail Diagram)로 구성된다.
가시적 도표는 전체적인 기능과 흐름을 보여주는 구조이다.
기능과 자료의 의존 관계를 동시에 표현할 수 있다.
보기 쉽고 이해하기 쉬우며 유지보수가 쉽다.
하향식 소프트웨어 개발을 위한 문서화 도구이다.
V-모델

폭포수 모형에 시스템 검증과 테스트 작업을 강조한 모델이다.
세부적인 프로세스로 구성되어 있어서 신뢰도 높은 시스템 개발에 효과적이다.
개발 단계의 작업을 확인하기 위해 테스트 작업을 수행한다.
생명주기 초반부터 테스트 작업을 지원한다.
코드뿐만 아니라 요구사항과 설계 결과도 테스트할 수 있어야 한다.
폭포수 모형보다 반복과 재처리 과정이 명확하다.
테스트 작업을 단계별로 구분하므로 책임이 명확해진다.
애자일(Agile) 개발 방법론
애자일(Agile) 개발방법론

날렵한, 재빠른 이란 사전적 의미가 있다.
특정 방법론이 아닌 소프트웨어를 빠르고 낭비 없이 제작하기 위해 고객과의 협업에 초점을 두고 소프트웨어 개발 중 설계 변경에 신속히 대응하여 요구사항을 수용할 수 있다.
절차와 도구보다 개인과 소통을 중요시하고 고객과의 피드백을 중요하게 생각한다.
소프트웨어가 잘 실행되는데 가치를 두며, 소프트웨어 배포 시차를 최소화할 수 있다.
특징 : 짧은 릴리즈와 반복, 점증적 설계, 사용자 참여, 문서 최소화, 비공식적인 커뮤니케이션 변화
종류
익스트림 프로그래밍(XP, eXtreame Programming)
스크럼(SCRUM)
린(Lean)
DSDM(Dynamic System Development Method, 동적 시스템 개발 방법론)
FDD(Feature Driven Development, 기능 중심 개발)
Crystal
ASD(Adaptive Software Development, 적응형 소프트웨어 개발방법론)
DAD(Disciplined Agile Delivery, 학습 애자일 배포)

Agile 선언문

프로세스나 도구보다 개인과의 소통이 더 중요하다.
완벽한 문서보다 실행되는 소프트웨어가 더 중요하다.
계약 협상보다 고객과의 협업이 더 중요하다.
계획을 따르는 것보다 변경에 대한 응답이 더 중요하다.
XP(eXtreme Programming)
XP(eXtreme Programming)

1999 년 Kent Beck 이 제안하였으며, 개발 단계 중 요구사항이 시시각각 변동이 심한 경우 적합한 방법론이다.
요구에 맞는 양질의 소프트웨어를 신속하게 제공하는 것을 목표로 한다.
요구사항을 모두 정의해 놓고 작업을 진행하는 것이 아니라, 요구사항이 변경되는 것을 적용하는 방식으로 예측성보다는 적응성에 더 높은 가치를 부여한 방법이다.
고객의 참여와 개발 과정의 반복을 극대화하여 생산성을 향상하는 방법이다.
XP 핵심 가치

소통(Communication) : 개발자, 관리자, 고객 간의 원활한 소통을 지향한다.
단순성(Simplicity) : 부가적 기능 또는 미사용 구조와 알고리즘은 배제한다.
Feedback : 소프트웨어 개발에서 변화는 불가피하다. 이러한 변화는 지속적 테스트와 통합, 반복적 결함 수정 등 빠르게 피드백한다.
용기(Courage) : 고객 요구사항 변화에 능동적으로 대응한다.
존중(Respect) : 개발 팀원 간의 상호 존중을 기본으로 한다.
XP Process

용어 설명
User Story 일종의 요구사항으로 UML 의 유즈케이스와 같은 목적으로 생성되나, 형식이 없고 고객에 의해 작성된다는 것이 다르다.
Release Planning 몇 개의 스토리가 적용되어 부분적으로 기능이 완료된 제품을 제공하는 것으로 부분/전체 개발 완료 시점에 대한 일정을 수립한다.
iteration 하나의 릴리즈를 세분화한 단위이며 1~3 주 단위로 진행된다.
반복(iteration) 진행 중 새로운 스토리가 추가될 때 진행 중 반복(Iteration)이나 다음 반복에 추가될 수 있다.
Acceptance Test 릴리즈 단위의 개발이 구현되었을 때 진행하는 테스트로, 사용자 스토리에 작성된 요구사항을 확인하여 고객이 직접 테스트한다.
오류가 발견되면 다음 반복(Iteration)에 추가한다. 테스트 후 고객의 요구사항이 변경되거나 추가되면 중요도에 따라 우선순위가 변경될 수 있다.
완료 후 다음 반복(Iteration)을 진행한다.

Small Release 릴리즈 단위를 기능별로 세분화하면 고객의 반응을 기능별로 확인할 수 있다.
최종 완제품일 때 고객에 의한 최종 테스트 진행 후 고객에 제공한다.
XP 의 12 가지 실천사항(Practice)

구분 12 실천사항 설명
Fine Scale Feed-back Pair Programing(짝 프로그래밍) 두 사람이 짝이 되어 한 사람은 코딩을, 다른 사람은 검사를 수행하는 방식이다.
코드에 대한 책임을 공유하고, 비형식적인 검토를 수행할 수 있다.
코드 개선을 위한 리팩토링을 장려하며, 생산성이 떨어지지 않는다.
Planning Game 게임처럼 선수와 규칙, 목표를 두고 기획에 임한다.
Test Driven Development 실제 코드를 작성하기 전에 단위 테스트부터 작성 및 수행하며, 이를 기반으로 코드를 작성한다.
Whole Team 개발 효율을 위해 고객을 프로젝트 팀원으로 상주시킨다.
Continuous Process Continuous Integration 상시 빌드 및 배포를 할 수 있는 상태로 유지한다.
Design Improvement 기능 변경 없이 중복성/복잡성 제거, 커뮤니케이션 향상, 단순화, 유연성 등을 위한 재구성을 수행한다.
Small Releases 짧은 주기로 잦은 릴리즈를 함으로써 고객이 변경사항을 볼 수 있게 한다.
Shared Understanding Coding Standards 소스 코드 작성 포맷과 규칙들을 표준화된 관례에 따라 작성한다.
Collective Code Ownership 시스템에 있는 소스 코드는 팀의 모든 프로그래머가 누구든지 언제라도 수정할 수 있다.
Simple Design 가능한 가장 간결한 디자인 상태를 유지한다.
System Metaphor 최종적으로 개발되어야 할 시스템의 구조를 기술한다.
Programmer Welfare Sustainable Pace 일주일에 40 시간 이상 작업 금지. 2 주 연속 오버타임을 금지한다.
효과적인 프로젝트 관리를 위한 3 대 요소

사람(People) - 인적 자원
문제(Problem) - 문제 인식
프로세스(Process) - 작업 계획
4. SCRUM
SCRUM
SCRUM 개념과 특징

요구사항 변경에 신속하게 대처할 수 있는 반복적이고 점진적인 소규모 팀원 간 활발한 소통과 협동심이 필요한 팀 중심의 소프트웨어 개발 방법론이다.
신속하게 반복적으로 실제 작동하는 소프트웨어를 제공한다.
개발자들의 팀 구성과 각 구성원의 역할, 일정 결과물 및 그 외 규칙을 정하는 것을 말한다.
기능 개선점에 우선순위를 부여하고, 개발 주기 동안 실제 동작 가능한 결과를 제공한다.
개발 주기마다 적용된 기능이나 개선점의 리스트를 제공한다.
커뮤니케이션을 위하여 팀은 개방된 공간에서 개발하고, 매일 15 분 정도 회의를 한다.
팀원 스스로 팀을 구성해야 한다.(Self Organizing)
개발 작업에 관한 모든 것을 팀원 스스로 해결해야 한다.(Cross Functional)
SCRUM 기본 원리

기능 협업을 기준으로 배치된 팀은 스프린트 단위로 소프트웨어를 개발한다.
스프린트는 고정된 30 일의 반복이며, 스프린트를 시행하는 작업은 고정된다.
요구사항, 아키텍처, 설계가 프로젝트 전반에 걸쳐 잘 드러나야 한다.
정해진 시간을 철저히 지켜야 하며, 완료된 모든 작업은 제품 백로그에 기록된다.
가장 기본적인 정보 교환 수단은 일일 스탠드 업 미팅, 또는 일일 스크럼이다.
SCRUM 팀의 역할

담당자 역할
제품 책임자(Product Owner) - 개발 목표에 이해도가 높은 개발 의뢰자, 사용자가 담당한다.
- 제품 요구사항을 파악하여 기능 목록(Product Backlog)을 작성한다.
- 제품 테스트 수행 및 요구사항 우선순위를 갱신한다.
- 업무 관점에서 우선순위와 중요도를 표시하고 신규 항목을 추가한다.
- 스프린트 계획 수립까지만 임무를 수행한다.
- 스프린트가 시작되면 팀 운영에 관여하지 않는다.
스크럼 마스터 - 업무를 배분만 하고 일은 강요하지 않으며 팀을 스스로 조직하고 관리하도록 지원한다. 개발 과정 장애 요소를 찾아 제거한다.
- 개발 과정에서 스크럼의 원칙과 가치를 지키도록 지원한다.
SCRUM 과정
Product Backlog

제품 개발에 필요한 모든 요구사항(User Story)을 우선순위에 따라 나열한 목록이다.
개발 과정에서 새롭게 도출되는 요구사항으로 인해 지속해서 업데이트된다.
제품 백로그에 작성된 사용자 스토리를 기반으로 전체 일정 계획인 릴리즈 계획을 수립한다.

Sprint

작은 단위의 개발 업무를 단기간에 전력 질주하여 개발한다는 의미로 반복 주기(2~4 주)마다 이해 관계자에게 일의 진척도를 보고한다.
Sprint Planning Meeting

Product Backlog(제품 기능 목록)에서 진행할 항목을 선택한다.
선택한 Sprint 에 대한 단기 일정을 수립하고, 요구사항을 개발자들이 나눠 작업할 수 있도록 Task 단위로 나눈다.
개발자별로 Sprint Backlog 를 작성하고 결과물에 대한 반복 완료 시 모습을 결정한다.
수행에 필요한 요구사항을 SCRUM Master 에게 보고하여 이해관계자로부터 지원을 받는다.
Daily SCRUM MEETING

매일 약속된 시간에 짧은 시간 동안(약 15 분) 서서 진행 상황만 점검한다.
한 사람씩 어제 한 일과 오늘 할 일을 이야기하고 스프린트 작업 목록을 잘 개발하고 있는지 확인한 뒤 완료된 세부 작업 항목을 완료 상태로 옮겨 스프린트 현황판에 갱신한다.
스크럼 마스터는 방해 요소를 찾아 해결하고 잔여 작업 시간을 소멸 차트(Burn down Chart)에 기록한다.
Finished Work

모든 스프린트 주기가 완료되면 제품 기능 목록(Product Backlog)의 개발 목표물이 완성된다.
스프린트 리뷰(Sprint Review)

스프린트 검토 회의(Sprint Review)에 개발자와 사용자가 같이 참석한다.
하나의 스프린트 반복 주기 (2~4 주)가 끝나면 실행 가능한 제품이 생성되며 이에 대해 검토하며, 검토는 가능한 4 시간 안에 마무리한다.
개선해야 할 사항에 대하여 제품 책임자(Product Owner)는 피드백을 정리하고 제품 백로그(Product Backlog)를 작성하여 다음 스프린트에 적용한다.
스프린트 회고(Sprint Retrospective)

스프린트에서 수행한 활동과 결과물을 살표본다.
개선점이 없는지 살펴보고 문제점을 기록하는 정도로 진행한다.
팀의 단점을 찾기보다는 강점을 찾아 팀 능력을 극대화한다.
개발 추정 속도와 실제 작업 속도를 비교하고 차이가 있다면 이유를 분석해본다.
5. 현행 시스템 분석
현행 시스템 분석
현행 시스템 분석의 정의와 목적

현행 시스템이 어떤 하위 시스템으로 구성되어 있는지 파악하는 절차를 의미한다.
현행 시스템의 제공 기능과 타 시스템과의 정보 교환 분석을 파악한다.
현행 시스템의 기술 요소와 소프트웨어, 하드웨어를 파악한다.
목적 : 개발 시스템의 개발 범위를 확인하고 이행 방향성을 설정한다.
현행 시스템 파악 절차

1 단계(시스템 구성 파악 - 시스템 기능 파악 - 시스템 인터페이스 현황 파악)
2 단계(아키텍처 파악 - 소프트웨어 구성 파악)
3 단계(시스템 하드웨어 현황 파악 - 네트워크 구성 파악)
시스템 아키텍처

시스템 내의 상위 시스템과 하위 시스템들이 어떠한 관계로 상호작용하는지 각각의 동작 원리와 구성을 표현한 것이다.
단위 업무 시스템별로 아키텍처가 다른 경우 핵심 기간 업무 처리 시스템을 기준으로 한다.
시스템의 전체 구조, 행위, 그리고 행위 원리를 나타내며 시스템이 어떻게 작동하는지 설명하는 틀이다.
시스템의 목적 달성을 위해 시스템에 구성된 각 컴포넌트를 식별하고 각 컴포넌트의 상호작용을 통하여 어떻게 정보가 교환되는지 설명한다.
시스템 아키텍처 ↔ 소프트웨어 아키텍처 => 소프트웨어 상세 설계

시스템 및 인터페이스 현황 파악
시스템 구성 파악

조직 내의 주요 업무를 기간 업무와 지원 업무로 구분하여 기술한다.
모든 단위 업무를 파악할 수 있도록 하며, 시스템 내의 명칭, 기능 등 주요 기능을 명시한다.
시스템 구성 현황 작성 예
구분 시스템명 시스템 내용
기간 업무 단위 A 업무 기간 단위 업무 A 처리를 위한 A1, A2 등의 기능을 제공
단위 B 업무 기간 단위 업무 B 처리를 위한 B1, B2 등의 기능을 제공
지원 업무 지원 C 업무 지원 업무 C 처리를 위한 C1, C2 등의 기능을 제공
시스템 기능 파악

단위 업무 시스템이 현재 제공하고 있는 기능을 주요 기능과 하부 기능으로 구분하여 계층형으로 표시한다.
시스템 기능 파악 예
시스템명 기능 L1 기능 L2 기능 L3

A 단위 업무 시스템 기능 1 하부 기능 11 세부 기능 111
세부 기능 112
하부 기능 12 세부 기능 121
세부 기능 122
기능 2 하부 기능 21 세부 기능 211
세부 기능 212
인터페이스 현황 파악

현행 시스템의 단위 업무 시스템이 타 단위 업무 시스템과 서로 주고받는 데이터의 연계 유형, 데이터 형식과 종류, 프로토콜 및 주기 등을 명시한다.
데이터 형식 ex. XML, 고정 Format, 가변 Format
통신 규약 ex. TCP/IP, X.25
연계 유형 ex. EAI, FEP
인터페이스 현황 작성 예
송신 시스템 수신 시스템 연동 데이터 연동 형식 통신 규약 연계 유형 주기
A 단위 업무 시스템 대외 기관 시스템 C 연체 정보 XML TCP/IP EAI 1 시간
A 단위 업무 시스템 대외 기관 시스템 D 신용 정보 XML X.25 FEP 수시
EAI(Enterprise Application Integration, 기업 애플리케이션 통합)

기업 내의 컴퓨터 애플리케이션들을 현대화하고, 통합하고, 조정하는 것을 목표로 세운 계획, 방법 및 도구 등을 의미한다.
FEP(Front-End Processor, 전위 처리기)

입력 데이터를 프로세서가 처리하기 전에 미리 처리하여 프로세서가 처리하는 시간을 줄여주는 프로그램이나 하드웨어이다.
여러 통신 라인을 중앙 컴퓨터에 연결하고 터미널의 메시지(Message)가 보낼 상태로 있는지 받을 상태로 있는지 검색한다. 통신 라인의 에러를 검출한다.
소프트웨어, 하드웨어, 네트워크 현황 파악
소프트웨어 구성 파악

시스템 내의 단위 업무 시스템의 업무 처리용 소프트웨어의 품명, 용도, 라이센스 적용 방식, 라이센스 수를 명시한다.
시스템 구축 시 많은 예산 비중을 차지하므로 라이센스 적용 방식과 보유한 라이센스 수량 파악이 중요하다.
라이센스 적용 방식 단위 : 사이트, 서버, 프로세서, 코어, 사용자 수
하드웨어 구성 파악

각 단위 업무 시스템의 서버 위치 및 주요 사양, 수량, 이중화 여부를 파악한다.
서버 사양 : CPU 처리 속도, 메모리 크기, 하드 디스크의 용량
서버 이중화 : 장애 시 서비스의 지속 유지를 위해 운영
기간 업무의 장애 대응 정책에 따라 필요 여부가 달라진다.
현행 시스템에 이중화가 적용되어 있다면 대부분 목표 시스템도 이중화가 요구되므로 그에 따른 기술 난이도, 비용 증가 가능성을 파악한다.
네트워크 구성 파악

현행 업무 처리 시스템의 네트워크 구성 형태를 그림으로 표현한다.
장애 발생 시 추적 및 대응 등의 다양한 용도로 활용된다.
서버의 위치, 서버 간 연결 방식 등을 파악한다.
물리적인 위치 관계, 조직 내 보안 취약성 분석 및 대응 방안을 파악한다.
개발 기술 환경 분석

개발 대상 시스템의 플랫폼, OS, DBMS, MiddleWare 등을 분석한다.
플랫폼(Platform)
플랫폼

응용 소프트웨어 + (하드웨어 + 시스템 소프트웨어)
다양한 애플리케이션이 작동하는 기본이 되는 운영체제 소프트웨어를 의미한다.
종류 : JAVA 플랫폼, .NET 플랫폼, IOS, Android, Windows
기능 : 개발/운영/유지보수 비용의 감소, 생산성 향상, 동일 플랫폼 간의 네트워크 효과
플랫폼 성능 특성 분석

현행 시스템의 사용자가 요구사항을 통하여 시스템 성능상의 문제를 요구할 경우 플랫폼 성능 분석을 통하여 사용자가 느끼는 속도를 파악하고 개선 방향을 제시할 수 있다.
특성 분석 항목 : 응답 시간(Response Time), 가용성(Availavilty), 사용률(Utilization)
현행 시스템의 OS 분석 항목 및 고려사항

분석 항목 : 현재 정보 시스템의 OS 종류와 버전, 패치 일자, 백업 주기 분석
고려사항 : 가용성, 성능, 기술 지원, 주변기기, 구축 비용(TCO)
현행 환경 분석 과정에서 라이센스의 종류, 사용자 수, 기술의 지속 가능성 등을 고려해야 한다.
메모리 누수 : 실행 SW 가 정상 종료되지 않고 남아있는 증상

TCO(Total Cost of Ownership) : 일정 기간 자산 획득에 필요한 직/간접적인 총비용으로 HW,SW 구매 비용, 운영 교육, 기술 지원, 유지보수, 손실 에너지 사용 비용 등이 있다.
오픈 라이센스 종류

소스 코드가 공개되어 누구나 특별한 제한 없이 소스를 사용할 수 있으며 대표적으로 Linux 가 있다.
GNU(GNU's Not Unix) : 컴퓨터 프로그램은 물론 모든 관련 정보를 돈으로 주고 구매하는 것을 반대하는 것이 기본 이념이다.
GNU GPLv1(General Public License) : 소스 코드를 공개하지 않으면서 바이너리만 배포하는 것을 금지하며, 사용할 수 있는 쉬운 소스 코드를 같이 배포해야 한다.
BSD(Berkeley Software Distributtion) : 아무나 제작할 수 있고, 수정한 것을 제한 없이 배포할 수 있다. 단, 수정본의 재배포는 의무적인 사항이 아니다. 공개하지 않아도 되는 상용 소프트웨어에서도 사용할 수 있다.
Apache 2.0 : Apache 재단 소유의 SW 적용을 위해 제공하는 라이센스이다. 소스 코드 수정 배포 시 Apache 2.0 을 포함해야 한다. (ex. Android, HADOOP - HADOOP 은 다수의 저렴한 컴퓨터를 하나처럼 묶어서 대량의 데이터를 처리하는 기술이다.)
플랫폼 성능 특성 분석 방법

기능 테스트(Performance Test) : 현재 시스템의 플랫폼을 평가할 수 있는 기능 테스트를 수행한다.
사용자 인터뷰 : 사용자를 대상으로 현행 플랫폼 기능의 불편함을 인터뷰한다.
문서 점검 : 플랫폼과 유사한 플랫폼의 기능 자료를 분석한다.
현행 시스템 DBMS 분석
DBMS(DataBase Management System)

종속성과 중복성의 문제를 해결하기 위해서 제안된 시스템이다.
응용 프로그램과 데이터의 중재자로서 모든 응용 프로그램들이 데이터베이스를 공유할 수 있도록 관리한다.
데이터베이스의 구성, 접근 방법, 관리 유지에 대한 모든 책임을 진다.
종류 : Oracle, IBM DB2, Microsoft SQL Server, MySQL, SQ Lite, MongoDB, Redis
현행 시스템 DBMS 분석

DBMS 의 종류, 버전, 구성 방식, 저장 용량, 백업 주기, 벤더의 유지보수 여부 가능성을 분석한다.
테이블 수량, 데이터 증가 추이, 백업 방식 등을 분석한다.

논리/물리 테이블의 구조도를 파악하여 각 테이블의 정규화 정도, 조인 난이도, 각종 프로시저, 트리거 등을 분석한다.
DBMS 분석 시 고려사항

구분 설명
가용성 장시간 운영 시 장애 발생 가능성, 패치 설치를 위한 재 기동 시간과 이중화 및 복제 지원, 백업 및 복구 편의성 등을 고려한다.
성능 대규모 데이터 처리 성능(분할 테이블의 지원 여부), 대량 거래 처리 성능 및 다양한 튜닝 옵션 지원, 비용 기반 최적화 지원 및 설정의 최소화 등을 고려한다.
기술 지원 제조업체의 안정적인 기술 지원, 같은 DBMS 사용자들 간의 정보 공유 여부와 오픈소스 여부 등을 고려한다.
상호 호환성 설치 가능한 운영체제 종류를 파악하여 다양한 운영체제에서 지원되는지 확인한다. JDBC, ODBC 등 상호 호환성이 좋은 제품을 선택한다.
구축 비용 라이센스 정책 및 비용, 유지 또는 관리 비용, 총 소유 비용(TCO)을 고려한다.
6. 요구사항 개발
요구사항 개발
요구공학(Requirements Engineering)

소프트웨어 개발 시 사용자 요구가 정확히 반영된 시스템 개발을 위하여 사용자의 요구를 추출, 분석, 명세, 검증, 관리하는 구조화된 활동 집합이다.
요구사항을 정의하고, 문서로 만들고 관리하는 프로세스를 의미한다.
효과적인 의사소통을 통하여 공통 이해를 설정하며, 불필요한 비용 절감, 요구사항 변경 추적이 가능해진다.
분석 결과의 문서화를 통해 향수 유지보수에 유용하게 활용할 수 있다.
자료 흐름도, 자료 사전 등이 효과적으로 이용될 수 있으며, 더 구체적인 명세를 위해 소단위 명세서(Mini-Spec)가 활용될 수 있다.
요구공학의 목적

소프트웨어 개발 시 이해관계자 사이의 원활한 의사소통 수단을 제공한다.
요구사항 누락 방지, 상호 이해 오류 등의 제거로 경제성을 제공한다.
요구사항 변경 이력 관리를 통하여 개발 비용 및 시간을 절약할 수 있다.
비용과 일정에 대한 제약설정과 타당성 조사, 요구사항 정의 문서화 등을 수행한다.
요구공학(개발) 프로세스

요구사항을 명확히 분석하여 검증하는 진행 순서를 의미한다.
개발 대상에 대한 요구사항을 체계적으로 도출한다.

도출된 요구사항을 분석하여 분석 결과를 명세서에 정리한다.
정리된 명세서를 마지막으로 확인, 검증하는 일련의 단계를 말한다.
경제성, 기술성, 적법성, 대안성 등 타당성 조사가 선행되어야 한다.
SWEBOK 에 따른 요구사항 개발 프로세스

도출(Elicitation) => 분석(Analysis) => 명세(Specification) =>
확인(Validation)
요구사항 도출(Requirement Elicitation)

소프트웨어가 해결해야 할 문제를 이해하는 첫 번째 단계이다.
현재의 상태를 파악하고 문제를 정의한 후 문제 해결과 목표를 명확히 도출하는 단계이다.
요구사항의 위치와 수집 방법과 관련되어 있다.
이해관계자(Stakeholder)가 식별되며, 개발팀과 고객 사이의 관계가 만들어지는
단계이며, 다양한 이해관계자와 효율적인 의사소통이 중요하다.
요구사항 도출 기법 : 고객의 발표, 문서 조사, 설문, 업무 절차 및 양식 조사,
브레인스토밍, 워크숍, 인터뷰, 관찰 및 모델의 프로토타이핑, Use Case, 벤치마킹,
BPR(업무 재설계), RFP(제안요청서)
요구사항 분석(Requirement Analysis)

소프트웨어가 환경과 어떻게 상호작용하는지 이해하고, 사용자의 요구사항을 걸러 내기
위한 과정을 통하여 요구사항을 도출하고, 요구사항 정의를 문서화하는 과정이다.
(사용자의 요구사항은 구조화와 열거가 어려워, 명확하지 못하거나 모호한 부분이 많다.)
도출된 사항을 분석하여 소프트웨어 개발 범위를 파악하고 개발 비용, 일정에 대한 제약을
설정하고 타당성 조사를 수행한다.
요구사항 간 상충하는 것을 해결하고, 소프트웨어의 범위(비용과 일정)를 파악하고 타당성
조사를 시행한다.
요구사항 기술 시 요구사항 확인, 요구사항 구현의 검증, 비용 추정 등의 작업이
가능하도록 충분하고 정확하게 기술한다.
요구분석을 위한 기법 : 사용자 의견 청취, 사용자 인터뷰, 현재 사용 중인 각종 문서
분석과 중재, 관찰 및 모델 작성 기술, 설문 조사를 통한 의견을 수렴한다.
요구사항 분석 수행 단계

문제 인식 : 인터뷰, 설문 조사 등 도구를 활용하여 요구사항을 파악하는 과정이다.
전개 : 파악한 문제를 자세히 조사하는 작업이다.
평가와 종합 : 요구들을 다이어그램이나 자동화 도구를 이용하여 종합하는 과정이다.
검토 : 요구분석 작업의 내용을 검토, 재정리하는 과정이다.
문서화 : 요구사항 분석 내용을 문서로 만드는 단계이다.

요구사항 분류

기술 내용에 따른 분류 : 기능 요구사항, 비기능 요구사항
기술 관점 및 대상에 따른 분류 : 시스템 요구사항, 사용자 요구사항.
요구사항 분류 기준

기능 요구사항, 비기능 요구사항을 구분하고 우선순위 여부를 확인한다.
요구사항이 하나 이상의 고수준 요구사항으로부터 유도된 것인지 확인한다.
이해관계자나 다른 원천(Source)으로부터 직접 발생한 것인지 확인한다.
요구사항이 제품에 관한 것인지 프로세스에 관한 것인지 확인하고 요구사항이 소프트웨어에 미치는 영향의 범위를 확인한다.
요구사항이 소프트웨어 생명주기 동안에 변경이 발생하는지 확인한다.
기능적 요구사항 VS 비기능적 요구사항

기능적 요구사항 비기능적 요구사항
시스템이 실제로 어떻게 동작하는지에 관점을 둔 요구사항 시스템 구축에 대한 성능, 보안, 품질, 안정성 등으로 실제 수행에 보조적인 요구사항
요구사항 명세(Requirement Specification)

시스템 정의, 시스템 요구사항, 소프트웨어 요구사항을 작성한다.
체계적으로 검토, 평가, 승인될 수 있도록 문서로 만드는 것을 의미한다.
기능 요구사항은 빠지는 부분 없이 명확하게 기술한다.
설계 과정의 오류사항을 추적할 수 있어야 한다.
비기능 요구사항은 필요한 것만 명확하게 기술한다.
개발자가 효과적으로 설계할 수 있고 사용자가 쉽게 이해할 수 있도록 한다.
요구사항 명세 기법

구분 정형 명세 비정형 명세
기법 수학적 기반/모델링 기반 - 상태/기능/객체 중심 명세 기법
- 자연어 기반
종류 - Z,VDM
- Petri-Net(모형 기반)
- LOTOS(대수적 방법)
- CSP, CCS - FSM(Finite State Machine)
- Decision Table, ER 모델링
- State Chart(SADT)
- UseCase
- 사용자 기반 모델링

장점 - 시스템 요구 특성을 정확하고 간결하게 명세할 수 있다.
- 명세/구현의 일치성 - 명세 작성 이해 용이
- 의사전달 방법 다양성
단점 - 낮은 이해도
- 이해관계자의 부담 가중 - 불충분한 명세 기능
- 모호성
요구사항 명세 속성

정확성 : 요구사항은 정확해야 한다.
명확성 : 단 한 가지로만 해설되어야 한다.
완전성 : 모든 것이 표현(기능+비기능) 가능해야 한다.
일관성 : 요구사항 간 충돌이 없어야 한다.
수정 용이성 : 요구사항 변경이 가능해야 한다.
추적성 : RFP, 제안서를 통해 추적 가능해야 한다.
요구사항 확인(Requirement Validation)

요구사항 분석 단계를 거쳐 문서로 만들어진 내용을 확인하고 검증하는 단계이다.
일반적으로 요구사항 관리 도구를 이용하여 이해관계자들이 문서를 검토해야 하고,
요구사항 정의 문서들에 대해 형상 관리를 한다.
회사의 표준에 적합하고 이해할 수 있고, 일관성이 있고, 완전한지 검증한다.
요구분석가가 요구사항을 이해했는지 확인(Validation)이 필요하다.
리소스가 요구사항에 할당되기 전에 문제를 파악하기 위하여 다음과 같은 검증을 수행한다.
(표준에 적합한가, 이해 가능한가, 일관성 있는가, 완전한가)
요구사항 관리 도구의 필요성 : 요구사항 변경으로 인한 비용 편익 분석, 요구사항 변경의 추적, 요구사항 변경에 따른 영향 평가
형상관리(Configuration Management)

애플리케이션 개발 단계에서 도출되는 프로그램, 문서, 데이터 등의 모든 자료를 형상 단위라고 하며, 이러한 자료의 변경을 관리함으로써 애플리케이션 버전 관리 등을 하는 활동이다.
요구사항 할당(Requirement Allocation)

요구사항을 만족시키기 위한 아키텍처 구성 요소를 식별하는 활동이다.
식별된 타 구성 요소와 상호작용 여부 분석을 통하여 추가 요구사항을 발견할 수 있다.
정형 분석(Formal Analysis)

구문(Syntax)과 형식적으로 정의된 의미(Semantics)를 지닌 언어로 요구사항을 표현한다.
정확하고 명확하게 표현하여 오해를 최소화할 수 있다.
요구사항 분석의 마지막 단계에서 이루어진다.
요구사항 확인 기법의
요구사항 확인 기법의 종류

프로토타이핑(Prototyping), 모델 검증(Model Verification), 요구사항 검토(Requirement Reviews), 인수 테스트(Acceptance Tests)
프로토타이핑(Prototyping)

도출된 요구사항을 토대로 프로토타입(시제품)을 제작하여 대상 시스템과 비교하면서 개발 중에 도출되는 추가 요구사항을 지속해서 재작성하는 과정이다.
새로운 요구사항을 도출하기 위한 수단이다.
소프트웨어 엔지니어 관점에서 요구사항을 확인하기 위한 수단으로 많이 사용되고, 실제 구현 전에 잘못된 요구사항을 적용하는 자원 낭비를 방지할 수 있다.
절차 : 요구사항 분석 단계 => 프로토타입 설계 단계 => 프로토타입 개발 단계 => 고객의 평가 단계 => 프로토타입 정제 단계 => 완제품 생산 단계
장점 단점
- 분석가의 가정을 파악하고 잘못되었을 때 유용한 피드백을 제공한다.
- 문서나 그래픽 모델보다 프로토타입으로 이해하기 쉬워 사용자와 개발자 사이의 의사소통에 도움이 된다.
- 요구사항의 가변성이 프로토타이핑 이후에 급격히 감소한다.
- 빠르게 제작할 수 있으며, 반복 제작을 통하여 발전된 결과를 가져올 수 있다. - 사용자의 관심이 핵심 기능에서 멀어질 수 있으며, 프로토타입의 디자인이나 품질 문제로 집중될 수 있다.
- 프로토타입 수행 비용이 발생한다.
- 전체 범위 중 일부 대상 범위만 프로토타입을 제작하면 사용성이 과대하게 평가될 수 있다.
모델 검증(Model Certification)

분석 단계에서 개발된 모델의 품질을 검증ㅈ한다.
정적 분석(Static Analysis) : 객체 모델에서 객체들 사이에 존재하는 Communication Path(의사소통 경로)를 검증하기 위해 사용한다. 명세의 일관성과 정확성을 확인 분석하는 도구이다.
동적 문석(Dynamic Analysis) : 직접 실행을 통하여 모델을 검증하는 방식이다.

인수 테스트(Acceptance Tests)

최종 제품이 설계 시 제시한 요구사항을 만족하는지 확인하는 단계이다.
인수 시 각 요구사항의 확인 절차를 계획해야 한다.
종류 : 계약 인수 테스트, 규정 인수 테스트, 알파 검사, 베타 검사, 사용자 인수 테스트, 운영 인수 테스트
7. UML
개념 모델링(Conceptual Modeling)
개념 모델링

요구사항을 이해하기 쉽도록 실 세계의 상황을 단순화하여 개념적으로 표현한 것을 모델이라고 하고, 이렇게 표현된 모델을 생성해 나가는 과정을 개념 모델링이라고 한다.
모델은 문제가 발생하는 상황에 대한 이해를 증진하고 해결책을 설명하므로 소프트웨어 요구사항 분석의 핵심이라 할 수 있다.
개발 대상 도메인의 엔티티(Entity)들과 그들의 관계 및 종속성을 반영한다.
요구사항별로 관점이 다르므로 개념 모델도 다양하게 표현되어야 한다.
대부분 UML(Unified Modeling Language)을 사용한다.
종류 : Use Case Diagram, Data Flow Model, State Model, Goal-Based Model, User Interactions, Object Model, Data Model
UML(Unified Modeling Language)
UML

객체지향 소프트웨어 개발 과정에서 시스템 분석, 설계, 구현 등의 산출물을 명세화, 시각화, 문서화 할 때 사용하는 모델링 기술과 방법론을 통합하여 만든 범용 모델링 언어이다.
Rumbaugh 의 OMT 방법론과 Booch 의 Booch 방법론, Jacobson 의 OOSE 방법론을 통합하여 만든 모델링 개념의 공통 집합으로 객체지향 분석 및 설계 방법론의 표준 지정을 목표로 제안된 모델링 언어이다.
OMG(Object Management Group)에서 표준화 공고 후 IMB, HP, Microsoft, Oracle 등이 참여하여 1997.1 버전 1.0 을 Release 하였다.
럼바우(Rumbaugh) 객체지향 분석 기법

소프트웨어 구성 요소를 그래픽으로 모형화하였다.
객체 모델링 기법이라고도 한다.
객체 모델링 : 정보 모델링이라고도 한다. 시스템에서 요구되는 객체를 찾아내어 속성과 연산 식별 및 객체들 간의 관계를 규정하여 객체를 다이어그램으로 표시한다.(OMT : Object Modeling Technique)

동적 모델링 : 제어 흐름, 상호작용, 동작 순서 등의 상태를 시간 흐름에 따라 상태 다이어그램으로 표시한다.
기능 모델링 : 여러 프로세스 간의 자료 흐름을 표시한다. 어떤 데이터를 입력하여 어떤 결과를 가져올 수 있을지를 표현한다.
UML 의 특성

비주얼화 : 소프트웨어 구성 요소 간의 관계 및 상호작용을 시각화한 것이다.
문서화 : 소프트웨어 생명주기의 중요한 작업을 추적하고 문서화할 수 있다. 개발 프로세스 및 언어와 무관하게 개발자 간의 의사소통 도구를 제공한다.
명세화 : 분석, 설계, 구현의 완벽한 모델을 제공한다. 분석 단계-기능 모델, 설계 단계-동작 수준 모델, 구현 단계-상호작용 모델 수준으로 명세화할 수 있다. 단순 표기법이 아닌 구현에 필요한 개발적 요소 및 기능에 대한 명세를 제공한다.
구축 : 객체지향 언어와 호환되는 프로그래밍 언어는 아니지만, 모델이 객체지향 언어로 매핑될 수 있다.
UML 소프트웨어에 대한 관점

기능적 관점 : 사용자 측면에서 본 소프트웨어의 기능을 나타낸다. 사용 사례 모델링이라고도 한다. 요구분석 단계에서 사용한다. UML 에서는 Use Case Diagram 을 사용한다.
정적 관점 : 소프트웨어 내부의 구성 요소 사이의 구조적 관계를 나타낸다. 객체, 속성, 연관 관계, 오퍼레이션의 시스템 구조를 나타내며, UML 에서는 Class Diagram 을 사용한다. (ex. 클래스 사이의 관계, 클래스 구성과 패키지 사이의 관계)
동적 관점 : 시스템의 내부 동작을 말하며, UML 에서는 Sequence Diagram, State Diagram, Activity Diagram 을 사용한다.
UML 의 기본 구성

구성 설명
사물(Things)- 객체지향 모델을 구성하는 기본 요소이다.
- 객체 간의 관계 형성 대상이다.
관계(Relationship) - 객체 간의 연관성을 표현하는 것이다.
- 종류 : 연관, 집합 포함, 일반화, 의존, 실체화
다이어그램(Diagram) - 객체의 관계를 도식화한 것이다.
- 다양한 관점에서 의사소통할 수 있도록 View 를 제공한다.
- 정적 모델 : 구조 다이어그램
- 동적 모델 : 행위 다이어그램
스테레토 타입

UML 에서 제공하는 기본 요소 외에 추가적인 확장 요소를 표현할 때 사용한다.
UML 확장 모델에서 스테레오 타입 객체를 표현할 때 사용하는 기호는 쌍 꺾쇠와 비슷하게 생긴 길러멧(Guillemet) <<>>이며, 길러멧 안에 확장 요소를 적는다.
UML 접근 제어자

접근제어자	표기	설명
Publice	+	어떤 클래스의 객체에서든 접근 가능하다.
Private	-	해당 클래스로 생성된 객체만 접근 가능하다.
Protected	#	해당 클래스와 동일 패키지에 있거나 상속 관계에 있는 하위 클래스의 객체들만 접근 가능하다.
Package	~	동일 패키지에 있는 클래스의 객체들만 접근 가능하다.

연관 관계 다중성 표현

표기	의미
1	1 개체 연결
* 또는 0..*	0 이거나 그 이상 객체 연결
1..*	1 이거나 1 이상 객체 연결
0..1	0 이거나 1 객체 연결
1,3,6	1 이거나 3 이거나 6 객체 연결
n	n 개 객체 연결
n..*	n 이거나 n 개 이상 객체 연결

UML 다이어그램의 분류
구조적 다이어그램(Structure Diagram)

정적이고, 구조 표현을 위한 다이어그램이다.

다이어그램 유형	목적
클래스 다이어그램(ClassDiagram)	시스템 내 클래스의 정적 구조를 표현하고 시스템을 구성하는 클래스들 사이의 관계를 표현한다.
객체 다이어그램(Object Diagram)	객체 정보를 보여준다.
복합체 구조 다이어그램(Composite Structure Diagram)	복합 구조의 클래스와 컴포넌트 내부 구조를 표현한다.
배치 다이어그램(Deployment Diagram)	소프트웨어, 하드웨어, 네트워크를 포함한 실행 시스템의 물리 구조를 표현한다.
컴포넌트 다이어그램(Component Diagram)	컴포넌트 구조 사이의 관계를 표현한다.
패키지 다이어그램(Package Diagram)	클래스나 유스케이스 등을 포함한 여러 모델 요소들을 그룹화해 패키지를 구성하고 패키지들 사이의 관계를 표현한다.

행위 다이어그램(Behavior Diagram)

동적이고 순차적인 표현을 위한 다이어그램이다.

종류

Use Case Diagram
Activity Diagram
Collaboration Diagram
State Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram

다이어그램 유형 목적

유스케이스 다이어그램(Use Case Diagram) 사용자 관점에서 시스템 행위를 표현한다.

활동 다이어그램(Activity Diagram) 업무 처리 과정이나 연산이 수행되는 과정을 표현한다.

상태 머신 다이어그램(State Machine Diagram) 객체의 생명주기를 표현한다. 동적 행위를 모델링하지만 특정 객체만을 다룬다. (ex. 실시간 임베디드 시스템, 게임, 프로토콜 설계에 이용)

Collaboration Diagram Sequence Diagram과 같으며, 모델링 공간에 제약이 없어 구조적인 면을 중시한다.

상호작용 다이어그램(Interaction Diagram)

순차 다이어그램(Sequence Diagram) - 시스템의 동작을 정형화하고 객체의 메시지 교환을 쉽게 표현하고 시간에 따른 메시지 발생 순서를 강조한다.

- 요소 : 생명선(Life Line), 통로(Gate), 상호작용(Interaction Fragment), 발생(Occurrence), 실행(Execution), 상태 불변(State Invariant), 상호작용(Interaction Use), 메시지(Messages), 활성(Activations), 객체(Entitiy), Actor

상호작용 개요 다이어그램 여러 상호작용 다이어그램 사이의 제어 흐름을 표현한다.

통신 다이어그램(Communication Diagram) 객체 사이의 관계를 중심으로 상호작용을 표현한다.

타이밍 다이어그램(Timing Diagram) 객체 상태 변화와 시간 제약을 명시적으로 표현한다.

클래스 다이어그램 관계 표현

Class Diagram

시스템을 구성하는 객체 간의 관계를 추상화한 모델을 논리적 구조로 표현한다.

객체지향 개발에서 공통으로 사용된다.
분석, 설계, 구현 단계 전반에 지속해서 사용된다.(사용: Operation : 클래스의 동작을 의미하며, 클래스에 속하는 객체에 대하여 적용될 메소드를 정의한 것)
UML 관계 표현

구성 표시 설명
단방향 연관 관계 —> 한쪽은 알지만 반대쪽은 상대방 존재를 모름
양방향 연관 관계 — 양쪽 클래스 객체들이 서로의 존재를 인식
의존 관계 ---> 연관 관계와 같지만 메소드를 사용할 때와 같이 매우 짧은 시간만 유지
일반화 관계 —▷ 객체지향에서 상속 관계(IS-A)를 표현하며, 한 클래스가 다른 클래스를 포함하는 상위 개념일 때 사용
집합/포함 관계 —◇ 클래스 사이 전체나 부분 이 같은 관계
—◆ 전체/부분 객체 라이프타임 의존적(전체 객체 삭제->부분 객체 삭제)
실체화 관계 ---▷ 책임 집합 인터페이스와 실제로 실현한 클래스들의 사이
UML 연관 관계(Association Relation)

한 사물의 객체가 다른 사물의 객체와 연결된 것을 표현한다.
두 클래스가 서로 연관이 있다면 A, B 객체를 서로 참조할 수 있음을 표현한다.
이름 : 관계의 의미를 표현하기 위해 이름을 가질 수 있다.
역할 : 수행하는 역할을 명시적으로 이름을 가질 수 있다.
UML 의존 관계(Dependency Relation)

연관 관계와 같지만 메소드를 사용할 때와 같이 매우 짧은 시간만 유지된다.
영향을 주는 객체(User)에서 영향을 받는 객체 방향으로 점선 화살표를 연결한다.
운전자—>자동차--->연료

운전자 & 자동차 = 연관 관계

자동차 & 연료 = 의존 관계

UML 일반화 관계(Generalization Relation)

객체지향에서 상속 관계(Is A Kind Of)를 표현한다.
한 클래스가 다른 클래스를 포함하는 상위 개념일 때 사용한다.(부모-자식)
UML 집합 관계

A 객체가 B 객체에 포함된 관계이다.

'부분'을 나타내는 객체를 다른 객체와 공유할 수 있다.
'전체' 클래스 방향에 빈 마름모로 표시하고, or 관계에 놓이면 선 사이를 점선으로 잇고 {or}를 표시한다.
ex. 학생(부분) —◇ 학교(전체) ◇— 교사(부분)

UML 포함 관계(Composition Relation)

부분 객체가 전체 객체에 속하는 강한 집합 연관의 관계를 표현하는 클래스이다.
'부분' 객체는 다른 객체와 공유 불가하고, '전체' 객체 방향에 채워진 마름모로 표시한다.
ex. 다리(부분) —◆ 책상(전체) ◆— 상판(부분)

UML 실체화 관계(Realization Relation)

인터페이스와 실제 구현된 일반 클래스 간의 관계로 존재하는 행동에 대한 구현을 표현한다.
한 객체가 다른 객체에게 오퍼레이션을 수행하도록 지정하는 의미적 관계이다.
ex. 미사일 —▷ 날다 ◁— 비행기

Use Case Diagram
Use Case Diagram 의 개념

객체지향 초반기 분석 작업에 작성되는 사용자의 요구를 기능적 측면에서 기술할 때 사용되는 도구로 Actor 와 User Case 로 구성된다.
얻어지는 결과는 개발 대상 시스템이 제공해야 하는 서비스 목록이 된다.
Use Case Diagram 요소

요소 설명
시스템 경계(System Boundary) - 시스템이 제공해야 하는 사례(Use Case)들의 범위가 된다.
- 큰 규모의 객체로 구현되는 존재이다.
액터(Actor) - 서비스를 이용하는 외부객체이다.
- 시스템이 특정한 사례(Use Case)를 실행하도록 요구할 수 있는 존재이다.
유스케이스(Use Case) - 시스템이 제공해야 하는 개별적인 서비스 기능이다.
- 서비스는 특정 클래스의 멤버 함수로 모델링된다.
접속 관계(communication Association) - 액터/유스케이스 또는 유스케이스/유케이스 사이에 연결되는 관계이다.
- 액터나 유스케이스가 다른 유스케이스의 서비스를 이용하는 상황을 표현한다.

사용 관계(Uses Association) 여러 개의 유스케이스에서 공통으로 수행해야 하는 기능을 모델링하기 위해 사용한다.
확장 관계(Extends Association) - 확장 기능 유스케이스와 확장 대상 유스케이스 사이에 형성되는 관계로, 해당 유스케이스에 부가적인 유스케이스를 실행할 수 있을 때의 관계이다.
- 확장 대상 유스케이스를 수행할 때 특정 조건에 따라 확장 기능 유스케이스를 수행하는 경우에 적용한다.
Use Case Diagram 작성단계

단계 식별
액터 식별 - 모든 사용자 역할과 상호작용하는 타 시스템을 식별한다.
- 정보를 주고받는 하드웨어 및 지능형 장치를 식별한다.
Use Case 식별 - 액터가 요구하는 서비스와 정보를 식별한다.
- 액터가 시스템과 상호작용하는 행위를 식별한다.
관계 정의 - 액터와 액터 그리고 액터와 유스케이스의 관계 분석을 정의한다.
- 유스케이스와 유스케이스 간의 관계 분석을 정의한다.
Use Case 구조화 - 두 개의 상위 Use Case 에 존재하는 공통 서비스를 추출한다.
- 추출된 서비스로 Use Case 를 정의한다.
- 추출된 서비를 사용하는 Use Case 와 관계를 정의한다.
- 조건에 따른 서비스 수행 부분을 분석하여 구조화한다.
8. 소프트웨어 아키텍쳐
소프트웨어 아키텍처
소프트웨어 아키텍처

개발 대상 시스템의 전반적인 구조를 체계적으로 설계하는 것이다.
다수의 이해관계자가 참여하는 복잡한 개발에서 상호이해, 타협, 의사소통을 체계적으로 접근하기 위한 것이다.
소프트웨어를 구성하는 컴포넌트들의 상호작용 및 관계, 각각의 특성을 기반으로 컴포넌트들이 상호 유기적으로 결합하는 소프트웨어의 여러 가지 원칙들의 집합이다.
소프트웨어 아키텍처 품질 요구사항

소프트웨어의 기능, 성능, 만족도 등의 요구사항이 얼마나 충족하는가를 나타내는 소프트웨어 특성의 핵심 집합이다.
사용자의 요구사항을 얼마나 충족시키느냐에 따라 확립된다.
ISO/IEC 9126 모델

소프트웨어 품질 특성과 평가를 위한 국제 표준이다.

내외부 품질 : 기능성, 신뢰성, 사용성, 효율성, 유지보수성, 이식성으로 구분된다.
사용 품질 : 효과성, 생산성, 안정성, 만족도
외부지표(External Metrics) : 실행 가능한 SW, 시스템을 시험, 운영 또는 관찰을 통하여 시스템을 구성하고 있는 일부분으로부터 추출된 소프트웨어 제품의 측정에 사용한다. 사용자, 평가자, 시험자 및 개발자에게 시험 수행이나 운영 중에 소프트웨어 제품에 대한 품질을 평가하는 항목이다.
내부 지표(Internal Metrics) : 설계, 코딩 도중에 실행할 수 없는 SW 제품(명세서, 원시 코드 등)에 대하여 적용하고 설계상 요구되는 외부 품질을 성취하기 위하여 ISO 9126-3 에 규정한다. 사용자, 평가자, 시험자 및 개발자가 소프트웨어 제품의 품질을 평가할 수 있도록 도움을 준다.
ISO/IEC 품질 특성
내/외부 품질
기능성
적합성
정확성
상호운용성
보안성
준수성
신뢰성
성숙성
결함 허용성
회복(복구)성
사용성
이해성
학습성
운용성
친밀성
준수성
효율성
시간 효율성
자원 활용성
준수성
유지보수성
분석성
변경성

안정성
시험성
준수성
이식성
적응성
설치성
공존성
대체성
준수성
사용 품질
효과성
생산성
안정성
만족도
ISO/IEC 25010

ISO/IEC 9126 에서 ISO/IEC 25010 으로 개정되어 특성 기준이 6 개에서 8 개로 증가하였다.
기존 : 기능성, 신뢰성, 사용성, 유지보수성, 이식성, 효율성
변경 : 기능 적합성, 실행 효율성, 호환성, 사용성, 신뢰성, 보안성, 유지보수성, 이식성, 부특성 일부가 증가
UI 표준을 위한 환경 분석
사용자 경향 분석

기존/현존 UI 경향을 숙지하고 현재 UI 단점을 작성한다.
사용자의 요구사항을 파악하고, 쉽게 이해 가능한 기능 위주로 기술 영역을 정의한다.
기능 및 설계 분석

기능 조작성 분석 : 사용자 편의를 위한 조작에 관한 분석을 확인한다. (ex. 스크롤바 지원 가능 여부, 마우스 조작 시 동선 확인)
오류방지 분석 : 조작 시 오류에 대해 예상 가능한지 확인한다. (ex. 의도치 않는 페이지 이동, 기능 버튼의 명확한 구분 가능한지 확인, 기능 버튼 이름이 사용자 조작과 일치하는지 확인)
최소한의 조작으로 업무 처리 가능한 형태 분석 : 작업 흐름에 가장 적합한 레이아웃인지 확인한다. (ex. 기능 특성에 맞는 UI 확인 및 조작 단계 최소화와 동선 단순 여부 확인)

UI 의 정보 전달력 확인 : 중요정보 인지, 쉽게 전달 가능한지, 정보 제공의 간결성, 명확성을 확인하고 정보 제공 방식의 일관성, 사용자 이해성 확인, 상호연관성 높은 정보인지 확인한다. (ex. 오류 발생 시 해결 방법 접근 용이성 확인)
요구사항 요소

구분 설명
데이터 요구 - 사용자 요구 모델과 객체들의 핵심 특성에 기반하여 데이터 객체를 정리한다.
- 인터페이스에 영향을 줄 수 있으니 초기에 확인한다.
(ex. Email 메시지 속성 : 제목, 송신자, 송신일, 참조인, 답변 등)
기능 요구 - 동사형으로 사용자의 목적 달성을 위해 실행해야 할 기능을 설명한다.
- 기능 요구 목록으로 정리한다.
- 최대한 철저하게 작성해야 한다.
제품, 서비스 품질 - 감성 품질과 데이터/기능 요구 외 제품 품질, 서비스 품질을 고려한다.
- 시스템 처리 능력 등 정량화 가능한 요구사항을 확인한다.
제약 사항 비용, 데드라인, 시스템 준수에 필요한 규제 등 사전에 제약 사항의 변경 여부를 확인한다.
정황 시나리오 작성

개발하는 서비스의 초기 모양을 상상하는 단계이다.
사용자 관점에서 작성하며 요구사항 정의에서 가장 기초적인 시나리오를 의미한다.
높은 수준과 낙관적인 상황에서 이상적 시스템 동작에 초점을 둔다.
육하원칙을 따르고 사용자가 주로 사용하는 기능 기반에서 작성한다.
간단명료하게 작성하여 정확하게 전달하고, 같은 동작 기능은 하나의 시나리오에 통합한다.
외부 전문가, 경험자에게 검토를 의뢰하도록 한다.
정황 시나리오 작성 예
정황 시나리오 요구사항
- 사원은 출근하여 시스템에 로그인하고 오늘 업무를 확인한다.
- 어제 요청한 결제가 승인되었는지 확인한다. - 로그인하면 맨 위 화면에 오늘 업무가 표시되어야 한다.
- 결제 요청 내역에 결제 승인 여부가 확인될 수 있도록 승인 내역은 다른 색을 이용한다.
9. UI 표준 및 지침
UI 표준 및 지침
UI(User Interface)

인간, 디지털 기기, 소프트웨어 사이에서 의사소통할 수 있도록 만들어진 매개체이다.

인간과 컴퓨터의 상호작용(HCI)에 필요한 화상, 문자, 소리, 수단(장치)을 의미한다.
UI 분야

표현에 관한 분야 : 전체적인 구성과 콘텐츠의 상세 표현을 위한 분야이다.
정보 제공과 전달 분야 : 물리적 제어를 통한 정보 제공과 전달을 위한 분야이다.
기능 분야 : 기능적으로 사용자가 쉽고 간편하게 사용하도록 하는 분야이다.
UI 의 특징

실사용자의 만족도에 직접 영향을 준다.
적절한 UI 구성으로 편리성, 가독성, 동선의 축약 등으로 작업 시간을 줄일 수 있고 업무 효율을 높일 수 있다.
실사용자가 수행해야 할 기능을 구체적으로 제시한다.
UI 설계 전 소프트웨어 아키텍처를 우선 숙지하고 있어야 한다.
UI 개발 시스템이 가져야 할 기능

사용자 입력의 검증
에러 처리의 에러 메시지 처리
도움과 프롬프트(Prompt) 제공
프롬프트 : 사용자의 명령을 받아들일 준비가 되었음을 모니터에 나타내는 표시(커서)
UI 설계
UI 설계 원칙

직관성 : 누구나 쉽게 이해하고 사용할 수 있도록 한다.
유효성 : 사용자의 목적을 정확히 달성할 수 있도록 유용하고 효과적이어야 한다.
학습성 : 사용자가 쉽게 배우고 익힐 수 있어야 한다.
유연성 : 사용자의 요구를 최대한 수용하면서 오류를 최소화해야 한다.
UI 설계의 필요성

구현 대상 결과의 오류 최소화와 적은 노력으로 구현하는 결과를 얻을 수 있다.
막연한 작업 기능에 대하여 구체적 방법을 제시한다.
사용자 편의성을 높여 작업 시간 단축, 업무 이해도를 높인다.
정보 제공자/공급자 사이의 원활하고 쉬운 매개 임무를 수행한다.
UI 설계 지침

구분 설명
사용자 중심 실사용자의 이해를 바탕으로 쉽게 이해하고 쉽게 사용할 수 있는 환경을 제공한다.

일관성 사용자가 기억하기 쉽고 빠른 습득을 가능하도록 버튼이나 조작법을 제공한다.
단순성 인지적 부담을 줄이도록 조작 방법을 가장 간단히 작동하도록 한다.
가시성 주요 기능은 메인 화면에 배치하여 조작이 쉽게 한다.
표준화 기능 구조의 선행 학습 이후 쉽게 이용할 수 있도록 디자인을 표준화한다.
접근성 사용자의 직무, 성별, 나이 등 다양한 계층을 수용해야 한다.
결과 예측 가능 작동 대상 기능만 보고도 결과 예측이 가능해야 한다.
명확성 사용자 관점에서 개념적으로 쉽게 인지할 수 있어야 한다.
오류 발생 해결 오류가 발생하면 사용자가 상황을 정확히 인지할 수 있어야 한다.
UI 표준
UI 구현 표준

전체 시스템 개발 중에 개발자 간 협업을 통하여 각기 개발한 화면 간에 갖추어야 할 최소한의 UI 요소 및 배치 규칙 등의 규칙을 의미한다.
UI 에 공통으로 적용되어야 할 화면 구성, 화면 이동 등이 있다.
UI 구현 지침

소프트웨어 개발 시 효율적인 정보 전달이 가능하도록 UI 설계에서 지켜야 할 세부 사항을 규정하는 것이다.
UI 요구사항, 구현 제약 사항 등 UI 개발 과정에서 꼭 지켜야 할 공통 조건을 의미한다.
한국형 웹 콘텐츠 접근성 지침 2.1 4 가지 원칙

인식의 용이성 : 대체 텍스트, 멀티미디어 대체 수단, 명료성
운용의 용이성 : 입력 장치 접근성, 충분한 시간 제공, 광(光) 과민성 발작 예방, 쉬운 내비게이션
이해의 용이성 : 가독성, 예측 가능성, 콘텐츠의 논리성, 입력 도움
견고성 : 문법 준수, 웹 애플리케이션 접근성
UX(User eXperience)
UX 사용자 경험

사용자가 제품을 대상으로 직/간접적으로 사용하면서 느끼고 생각하게 되는 지각과 반응, 행동 등 모든 경험을 의미한다.
UI 는 사람과 시스템 간의 상호작용을 의미하지만, UX 는 제품과 서비스, 회사와 상호작용을 통해서 전체적인 느낌이나 경험을 말한다.
UX 에 영향을 주는 요소 : 성능, 시간
모바일 사용자 UX 설계 시 고려사항(행정안전부 고시)

시스템을 사용하는 대상, 환경, 목적, 빈도 등을 고려한다.

사용자가 직관적으로 서비스 이용 방법을 파악할 수 있도록 한다.
입력의 최소화, 자동 완성 기능을 제공한다.
사용자의 입력 실수를 수정할 수 있도록 되돌림 기능을 제공한다.
모바일 서비스의 특성에 적합한 디자인을 제공한다.
10. UI 설계
UI 설계 단계
UI 설계 단계

문제 정의 : 시스템의 목적과 해결해야 할 문제를 정의한다.
사용자 모델 정의 : 사용자 특성을 결정하고, 소프트웨어 작업 지식 정도에 따라 초보자, 중급자, 숙련자로 구분한다.
작업 분석 : 사용자의 특징을 세분화하고 수행되어야 할 작업을 정의한다.
컴퓨터 오브젝트 및 기능 정의 : 작업 분석을 통하여 어떤 사용자 인터페이스에 표현할지를 정의한다.
사용자 인터페이스 정의 : 모니터, 마우스, 키보드, 터치스크린 등 물리적 입/출력 장치 등 상호작용 오브젝트를 통하여 시스템 상태를 명확히 한다.
디자인 평가 : 사용자 능력, 지식에 적합한가? 사용자가 사용하기 편리한가? 등의 평가를 의미하며, 사용성 공학을 통하여 사용성 평가를 할 수 있다. 평가 방법론으로는 GOMS, Heuristics 등이 있다.
GOMS : 인간이 어떤 행위를 할지 예측하여 그 문제를 해결하는데 필요한 소요시간, 학습시간 등을 평가하기 위한 기법
Heuristics : 논리적 근거가 아닌 어림짐작을 통하여 답을 도출해 내는 방법
UI 상세 설계 단계

설계 실행사항
UI 메뉴 구조 설계 - 요구사항과 UI 표준 및 지침에 따라 사용자의 편의성을 고려한다.
- 요구사항 최종 확인, UI 설계서 표지 및 개정 이력을 작성한다.
- UI 구조 설계, 사용자 기반 메뉴 구조 설계 및 화면을 설계한다.
내/외부 화면과 폼 설계 - UI 요구사항과 UI 표준 지침에 따라 하위 시스템 단위를 설계한다. 실행 차를 최소화하기 위하여 UI 설계 원리 검토 => 행위 순서 검토 => 행위 순서대로 실행 검토한다.
- 평가 차를 줄이기 위한 UI 설계 원리를 검토한다.
UI 검토 수행 - UI 검토 보완을 위한 시뮬레이션 시연 구성원에는 컴퓨터 역할을 하기 위해 서류를 조작하는 사람, 전체적인 평가를 위한 평가 진행자, 관찰자가 있다. 이 평가 결과를 토대로 설계를 보완한다.
- UI 시연을 통한 사용성에 대한 검토 및 검증을 수행한다.

UI 상세 설계-시나리오 작성 원칙

UI 전체적 기능, 작동 방식을 개발자가 쉽게 이해할 수 있도록 구체적으로 작성한다.
대표 화면 레이아웃 및 하위 기능을 정의하고 Tree 구조나 Flowchart 표기법을 이용한다.
공통 적용이 가능한 UI 요소와 상호작용(Interaction)을 일반적인 규칙으로 정의한다.
상호작용의 흐름 및 순서, 분기, 조건, 루프를 명시한다.
예외 상황에 관한 사례를 정의하고 UI 시나리오 규칙을 지정한다.
기능별 상세 기능 시나리오를 정의하되 UI 일반 규칙을 지킨다.
시나리오 문서의 작성 요건 : 완전성, 일관성, 이해성, 가독성, 수정 용이성, 추적 용이성
UI 흐름 설계서 구성

UI 설계서 표지 : 프로젝트 이름, 시스템 이름을 포함하여 작성한다.
UI 설계서 개정 이력 : 처음 작성 시 '초안 작성'을 포함한다. 초기 버전은 1.0 으로 설정하고 완성 시 버전은 x.0 으로 바꾸어 설정한다.
UI 요구사항 정의

시스템 구조 : UI 프로토타입 재확인 후 UI 시스템 구조를 설계한다.
사이트맵 : UI 시스템 구조를 사이트맵 구조로 설계한다.
프로세스 정의 : 사용자 관점에서의 요구 프로세스 순서를 정리한다.
화면 설계 : UI 프로세스/프로토타입을 고려하여 페이지 별로 화면을 구성 및 설계한다.
UI 설계 도구
UI 설계에 도움을 주는 도구들

와이어 프레임(Wire Frame) : UI 중심의 화면 레이아웃을 선(Wire)을 이용하여 대략적으로 작성한다.
목업(Mockup) : 실물과 흡사한 정적인 모형을 의미한다.
프로토타입(Prototype) : Interaction(상호작용)이 결합하여 실제 작동하는 모형이다.
스토리보드(Storyboard) : 정책, 프로세스, 와이어 프레임, 설명이 모두 포함된 설계 문서이다.
와이어 프레임

기획 단계 초기에 작성하며, 구성할 화면의 대략적인 레이아웃이나 UI 요소 등의 틀을 설계하는 단계이다.
개발 관계자(디자이너, 개발자, 기획자) 사이의 레이아웃 협의, 현재 진행 상황 등을 공유할 때 사용한다.

툴 : 핸드라이팅, 파워포인트, 키노트, Sketch, Balsamiq, Mockup, Adobe Experience Design, 카카오 오븐

목업(Mockup)

와이어 프레임보더 좀 더 실제 제품과 유사하게 만들어지는 실물 크기의 정적 모형으로 시각적으로만 구현된다.

툴 : 카카오 오븐, Balsamiq Mockup, Power Mockup

스토리보드

UI/UX 구현에 수반되는 사용자와 작업, 인터페이스 간 상호작용을 시각화한 것이다.
개발자/디자이너와의 의사소통을 돕는 도구이다.
완성해야 할 서비스와 예상되는 사용자 경험을 미리 보기 위한 방법론이다.
작성 목적 : 설계에 필요한 조각을 모아 순서대로 놓고 배치해 보고 쌓아서 조립하는 과정으로 설계 단계에서 발생할 수 있는 문제를 미리 발견하고 대처하기 위한 과정이다.
작성 방법 : 상단/우측 => 제목, 작성자 기재, 좌측 => UI 화면, 우측 => Description
작성 단계 : 메뉴 구성도 만들기 => 스타일 확정하기 => 설계하기

UI 프로토타입

UI Prototype

도출된 요구사항을 토대로 프로토타입을 제작하여 대상 시스템과 비교하면서 개발 중에 도출되는 추가 요구사항을 지속해서 재작성하는 과정이다.
와이어 프레임, 스토리보드에 Interaction 을 적용한 것이다.
동적인 형태로 구현된 모형이다.

툴 : HTML/CSS, Axure, Invision Studio, 카카오 오븐, Flinto, 네이버 Proto Now

작성 방법에 따른 분류 : 디지털 프로토타입, 페이퍼 프로토타입

프로토타입의 장단점

장점 단점

- 사용자 설득과 이해가 쉽다.
- 개발 시간이 감소한다.
- 오류를 사전에 발견할 수 있다. - 수정이 많아지면 작업 시간이 늘어날 수 있다.
- 필요 이상으로 자원을 많이 소모한다.
- 정확한 문서 작업이 생략되는 문제가 발생할 수 있다.

프로토타입 작성 도구 및 방법

구분 방법 비고

Analog - 포스트잇, 칠판, 종이, 펜 등을 이용한다.

- 소규모 개발, 제작 비용과 기간이 적을 경우 이용한다.
- 빠른 업무 협의가 필요한 경우 이용한다. 손,펜
- 비용이 저렴하면서 즉시 변경이 가능하다.
- 회의 중 바로 작성할 수 있다.
- 상호 연관 관계가 복잡한 경우 표현이 어렵다.
- 공유가 어렵다. -

Digital Power Point, Acrobat, Invision, Marvel, Adobe Xd, Flinto, Priciple, Keynote, UX pin, HTML 등을 이용한다. Digital Tool

- 재사용성이 높지만 툴을 다룰 줄 아는 전문가가 필요하다.
- 목표 제품과 비슷하게 테스트할 수 있으며 수정이 수월하다. -

UI Prototype 작성 시 고려사항

프로토타입 계획작성, 프로토타입 범위 확인, 프로토타입 목표 확인, 프로토타입 기간 및 비용 확인, 프로토타입 산출물 확인, 프로토타입 유의사항 확인

UI Prototype 계획 시 고려사항

프로토타입 목표 확인, 프로토타입 환경 확인, 프로토타입 일정 확인, 프로토타입 범위 확인, 프로토타입 인원 확인, 프로토타입 아키텍쳐 검증 확인, 프로토타입 이슈 및 해결, 프로토타입 가이드 확정, 프로토타입 개발 생산성 확인, 프로토타입 결과 시연

UI 프로토타입 제작 단계

사용자 요구분석 => 프로토타입 작성 => 프로토타입 사용자 테스트 => 수정과 합의 단계

감성 공학

감성 공학

인간의 소망으로 이미지나 감성을 구체적 제품 설계를 통하여 실현해 내는 공학적 접근 방법으로 인간과 컴퓨터 간의 상호작용 즉 HCI(Human Computer Interaction or Interface) 설계에 인간의 특성, 감성 등의 정량적 측정과 평가를 통하여 제품 환경 설계에 반영하는 기술이다.

인간이 가지고 있는 소망으로서의 이미지나 감성을 구체적인 제품 설계로 실현해 내는 인문 사회 과학, 공학, 의학 등 여러 분야의 학문이 융합된 기술이다.

감각 및 생체계측, 센서, 인공지능 등의 생체 제어 기술 등을 통해 과학적으로 접근한다.

최종 목표는 감성 공학을 통하여 인간이 쉽고 편리하고 쾌적하게 시스템과 어우러지는 것이다.

1988 년 시드니 국제 학회에서 '감성 공학'으로 명명된다.

감성 공학 접근 방법

1 류(의미 미분법) : 인간의 감각, 감성을 표현하는 어휘(형용사)를 이용하여 제품에 대한 이미지는 조사/분석하고, 디자인 요소에 연계하는 접근 방법이다.
2 류 : 1 류와 기본 틀은 공유하고, 감성 어휘 수집의 전 단계에서 평가자들의 생활 양식을 추가하였다. 제품에 대한 기호 및 수용를 분석 대상의 소속 지역, 생활 양식, 의식 문화를 분석하는 접근 방법이며, 1 류와 함께 감성의 심리적 특성을 강조한다.
3 류 : 1 류의 감성 어휘 대신 평가자의 특정 시제품을 사용하여 자신의 감각 척도로 감성을 표출하는 방법이다. 평가자의 생리적 감각 계측을 통해서 그 객관성이 보완되고 정량화된 값으로 산출된다. 대상 제품의 물리적인 특성에 대하여 객관적인 지표와의 연관 분석을 통하여 제품 설계에 응용된다. 인간 감각 계측과 이의 활용이 강조된 접근 방법으로 감성의 생리적 특성을 중요시한다.
HCI(Human computer Interaction or Interface)

인간과 컴퓨터의 상호작용을 연구하여 어떻게 하면 좋은 제품을 만들 수 있는지를 연구한다.
HCI 목적

컴퓨터를 인간이 쉽게 사용할 수 있게 하여 상호작용(UX)을 개선하는 것이다.
컴퓨터의 도구로서 잠재력을 극대화해 인간의 의지를 더 자유롭게 한다.
인간의 창의력, 인간 사이의 의사소통과 협력을 증진하는데 있다.
감성 공학 요소

기초 기술, 구현 기술, 응용 기술
감성 공학 관련 기술

생체측정 기술, 인간 감성 특성 파악 기술, 감성 디자인 기술과 오감 센서 및 감성 처리 기술, 마이크로 기구 설계, 사용성 평가 기술 및 가상현실 기술
11. 소프트웨어 설계 모델링
소프트웨어의 설계(Design)
소프트웨어 설계 모델링

정의 : 요구사항(기능, 성능)을 만족하는 소프트웨어의 내부 구조 및 동적 행위들을 모델링하여 표현하고, 분석 검증하는 과정이며 이 과정에서 만들어지는 산출물을 의미한다.
목적 : "무엇을(What)" 으로부터 "어떻게(How)"로 관점을 전환하면서 최종 제작할 소프트웨어의 청사진을 만드는 것을 의미한다.
소프트웨어 설계

본격적인 프로그램의 구현에 들어가기 전에 소프트웨어를 구성하는 뼈대를 정의해 구현의 기반을 만드는 것을 의미하며 상위 설계(High-Level Design)와 하위 설계로 구분된다.

설계의 기본 원리

분할과 정복, 추상화, 단계적 분해, 모듈화, 정보 은닉
소프트웨어 설계 분류

설계
상위 설계
아키텍처 설계
데이터 설계
인터페이스 정의
사용자 인터페이스 설계
하위 설계
모듈 설계
자료 구조 설계
알고리즘 설계
상위 설계(High-Level Design)

아키텍처 설계(Architecture Design), 예비 설계(Preliminary Design)라고 하며 전체 골조(뼈대)를 세우는 단계이다.
아키텍처(구조) 설계 : 시스템의 전체적인 구조
데이터 설계 : 시스템에 필요한 정보를 자료 구조/데이터베이스 설계에 반영
시스템 분할 : 전체 시스템을 여러 개의 서브 시스템으로 분리
인터페이스 설계 : 시스템의 구조와 서브 시스템들 사이의 관계
사용자 인터페이스 설계 : 사용자와 시스템의 관계
하위 설계(Low_Level Design)

모듈 설계(Module Design), 상세 설계(Detail Design)라고 하며, 시스템 각 구성 요소들의 내부 구조, 동적 행위 등을 결정하여 각 구성 요소의 제어와 데이터 간의 연결에 대한 구체적인 정의를 하는 단계이다.
하위 설계 방법 : 절차 기반(Procedure Oriented), 자료 위주(Data Oriented), 객체 지향(Object Oriented) 설계 방법
소프트웨어 설계 대상

구분 설명
구조 모델링 - SW 를 구성하는 컴포넌트의 유형, 인터페이스, 내부 설계 구조 등 구조의 상호 연결 등의 구조를 모델링하는 것이다.
- 구성 요소에는 프로시저, 데이터 구조, 모듈, 파일 구조 등이 있다.

- 구성 요소들의 연결 구조, 포함 관계를 시스템 구조라 한다.
행위 모델링 - 소프트웨어의 구성 요소들 기능과 구성 요소들이 언제, 어떤 순서로 기능을 수행하고 상호작용하는지를 모델링하는 것이다.
- 시스템 각 구성 요소들의 기능적인 특성을 모델링하는 것이다.
- 입/출력 데이터, 데이터의 흐름과 변환, 데이터의 저장, 실행 경로, 상태 전이, 이벤트 발생순서 등이 행위 모형화에 속한다.
소프트웨어 설계 방법

구분 설명
구조적 설계 - 기능적 관점으로 소프트웨어에 요구된 기능이나 자료 처리 과정, 알고리즘 등을 중심으로 시스템을 나눠 설계하는 방식이다.
- 시스템의 각 모듈은 최상위 기능에서 하위 기능으로 하향적으로 세분화한다.
- Coad/Yourdon
자료 중심 설계 - 입/출력 자료의 구조를 파악하여 소프트웨워 자료 구조를 설계하는 방식이다.
- Jackson Warner - Orr
객체지향 설계 - 자료와 자료에 적용될 기능을 함께 묶어 추상화하는 개념이다.
- 시스템은 객체로 구성된다.
- Yourdon, Sheller/Meller, Rumbaugh, Booch
소프트웨어 구조도

소프트웨어의 구성 요소인 모듈 간의 계층적 구성을 나타낸 것이다.
프로그램 구조에서 사용되는 용어이다.
용어 정의
Fan-in 주어진 한 모듈을 제어하는 상위 모듈 수
Fan-out 주어진 한 모듈이 제어하는 하위 모듈 수
Depth 최상위 모듈에서 주어진 모듈까지의 깊이
Width 같은 등급(level)의 모듈 수
Super ordinate 다른 모듈을 제어하는 모듈
Subordinate 어떤 모듈에 의해 제어되는 모듈
ex. 모듈 F 에서의 Fan-in : 3, Fan-out : 2

Fan-in/Fan-out 을 분석하면 시스템 복잡도 파악이 가능하다.
Fan-in 이 높은 경우 Fan-out 이 높은 경우
- 재사용 측면에서 잘된 설계로 볼 수 있다.
- 시스템 구성 요소 중 일부가 동작하지 않으면 시스템이 중단되는 단일 장애 발생 가능성이 있다.

- 단일 장애 발생을 방지하기 위해 중점 관리가 필요하다. - 불필요한 타 모듈의 호출 여부를 확인한다.
- Fan-out 을 단순하게 설계할 수 있는지 검토한다.
복잡도 최적화를 위한 조건 : Fan-in 은 높이고 Fan-out 은 낮추도록 설계한다.
코드 설계의 개요
코드 설계

데이터의 사용 목적에 따라서 식별하고 분류, 배열하기 위하여 사용하는 숫자, 문자 혹은 기호를 코드라고 한다.
대량의 자료를 구별, 동질의 그룹으로 분류하고 순번으로 나열하며, 특정의 자료를 선별하거나 추출을 쉽게 하여 파일 시스템을 체계화한 것을 코드 설계라 한다.
코드 설계 순서 : 코드 대상 선정 => 코드화 목적 명확화 => 코드 부여 대상 수 확인 => 사용 범위 결정 => 사용 기간 결정 => 코드화 대상의 특성 분석 => 코드 부여 방식 결정 => 코드의 문서화
코드의 기능

코드의 기본적 기능 코드의 3 대 기능 코드의 부가적 기능
- 표준화 기능
- 간소화 기능- 분류 기능
- 식별 기능
- 배열 기능 -연상 기능
- 암호화 기능
- 오류 검출 기능
코드 설계 목적 및 특성

목적 특성
고유성 코드는 그 뜻이 1:1 로 확실히 대응할 수 있어야 한다.
분류 편리성 목적에 적합한 분류가 가능해야 한다.
배열의 효율성 바람직한 배열을 얻을 수 있어야 한다.
간결성 짧고 간결 명료해야 한다.
유지보수 편리성 유지 관리가 쉬워야 한다.
코드의 독립성 다른 코드 체계와 중복되지 않아야 한다.
코드의 편의성 이해가 쉽고, 사용하는데 편리해야 한다.
추가/삭제 편리성 추가와 삭제가 편리해야 한다.
코드 설계 시 고려사항

기계 처리의 적합성 사용의 편리성

컴퓨터의 처리에 적합하게 한다. 취급하기 쉽게 한다.
코드의 종류
순차 코드

코드화 대상 항목을 어떤 일정한 배열로 일련번호를 배당하는 코드로 확장성이 좋으며, 단순해서 이해하기 쉽고, 기억하기 쉽다.
항목 수가 적고, 변경이 적은 자료에 적합하며, 일정 순서대로 코드를 할당하므로 기억 공간 낭비가 적다.
누락된 번호를 삽입하기 어렵고 명확한 분류 기준이 없어 코드에 따라 분류가 어려워 융통성이 낮다.
블록 코드(Block Code, 구분 코드)

코드화 대상 항목에 미리 공통의 특성에 따라서 임의의 크기를 블록으로 구분하여 각 블록 안에서 일련번호를 배정하는 코드이다.
기계 처리가 어렵고 블록마다 여유 코드를 두어 코드의 추가를 쉽게 할 수 있지만, 여유 코드는 코드 낭비 요인이 된다.
그룹 분류식 코드(Group Classification Code)

코드화 대상 항목을 소정의 기준에 따라 대분류, 중분류, 소분류로 구분하고 순서대로 번호를 부여하는 코드이다.
분류 기준이 명확한 경우 이용도 높으며 기계 처리에 가장 접합하다.
여유 부분이 있어 자료 추가를 쉽게 처리할 수 있으나 자릿수가 길어질 수 있다.
10 진 분류 코드(Decimal Code)

좌측 부는 그룹 분류에 따르고 우측은 10 진수의 원칙에 따라 세분화하는 코드이다.
무한하게 확대할 수 있어 대량의 자료에 대한 삽입 및 추가가 쉽다.
자릿수가 많아지고 기계 처리에 불편하지만, 배열이나 집계가 쉽다.
주로 도서 분류 코드에 사용된다.
ex.

코드	의미
100	국문학
200	철학
300	정보학

표의 숫자 코드(Significant Digit Code, 유효 숫자 코드)

코드화 대상 항목의 길이, 넓이 부피, 무게 등을 나타내는 문자나 숫자, 기호를 그대로 코드로 사용하는 코드이다.

코드의 추가 및 삭제가 쉽다.
같은 코드를 반복 사용하므로 오류가 적다.
코드 의미
127-890-1245 두께 127mm, 폭 890mm, 길이 1,245mm 의 강판
연상 코드(Mnemonic Code. 기호 코드)

코드화 대상의 품목 명칭 일부를 약호 형태로 코드 속에 넣어 대상 항목을 쉽게 알 수 있는 코드이다.
코드 의미
TV-39-C TV 39 인치 컬러
코드의 오류 종류

오류 의미 예
필사 오류(Transcription Error) 입력 시 한 자리를 잘못 기록하는 오류
1234=>1237
전위 오류(Transposition Error) 입력 시 좌우 자리를 바꾸어 발생하는 오류
1234=>1243
이중 오류(Double Transposition Error) 전위 오류가 두 개 이상 발생하는 오류
1234=>2143
생략 오류(Missing Error) 입력 시 한 자리를 빼고 기록하는 오류 1234=>123
추가 오류(Addition Error) 입력 시 한 자리를 추가해서 기록하는 오류
1234=>12345
임의 오류(Random Error) 두 가지 이상의 오류가 결합해서 발생하는 오류
1234=>21345
구조적 개발 방법론
구조적 분석

자료(Data)의 흐름, 처리를 중심으로 한 요구분석 방법으로 전체 시스템의 일관성 있는 이해를 돕는 분석 도구로 모형화에 필요한 도구 제공 및 시스템을 나누어 분석할 수 있다.
정형화된 분석 절차에 따라 사용자 요구사항을 파악, 문서화하는 체계적 분석 방법으로 자료 흐름도, 자료 사전, 소단위 명세를 사용한다.
시스템 분할이 가능하며 하향식 분석 기법을 사용하고 분석자와 사용자 간의 의사소통을 돕는다.
구조적 설계의 특징과 기본 구조

특징 : 하향식 기법, 신뢰성 향상, 유연성 제공, 재사용 용이
기본 구조 : 순차(Sequence) 구조, 선택(Selection) 구조 = 조건(Condition) 구조, 반복(Repetition) 구조

구조적 분석 도구

자료 흐름도(DFD : Data Flow Diagram)

시스템 내의 모든 자료 흐름을 4 가지의 기본 기호(처리, 자료 흐름, 자료 저장소, 단말)로 기술하고 이런 자료 흐름을 중심으로 한 분석용 도구이다.
DFD 의 요소는 화살표, 원, 사각형, 직선(단선/이중선)으로 표시하고 구조적 분석 기법에 이용된다.
시스템이나 프로그램 간의 총체적인 데이터 흐름을 표시할 수 있으며, 기본적인 데이터 요소와 그들 사이의 데이터 흐름 형태로 기술된다.
다차원적이며 자료 흐름 그래프 또는 버블(Bubble) 차트라고도 한다.
그림 중심의 표현이고 하향식 분할 원리를 적용한다.
갱신하기 쉬워야 하며 이름의 중복을 제거하여 이름으로 정의를 쉽게 찾을 수 있도록 한다.
정의하는 방식이 명확해야 한다.

자료 흐름도(DFD) 작성 원칙

출력 자료 흐름은 입력 자료 흐름을 이용해 생성해야 한다.
입력, 출력 자료 자체에 대해서만 인지하고 자료의 위치나 방향은 알 필요가 없다.
자료 흐름 변환의 형태에는 본질 변환, 합성의 변환, 관점의 변환, 구성의 변환 등이 있다.
자료 보존의 원칙 : 출력 자료 흐름은 반드시 입력 자료 흐름을 이용해 새엇ㅇ한다.
최소 자료 입력의 원칙 : 출력 자료를 산출하는데 필요한 최소의 자료 흐름만 입력한다.
독립성의 원칙 : 프로세스는 오직 자신의 입력 자료와 출력 자료 자체에 대해서만 알면 된다.
지속성의 원칙 : 프로세스는 항상 수행하고 있어야 한다.
순차 처리의 원칙 : 입력 자료 흐름의 순서는 출력되는 자료 흐름에서도 지키도록 한다.
영구성의 원칙 : 자료 저장소의 자료는 입력으로 사용해도 삭제되지 않는다.

데이터(자료) 흐름도

구성 요소	의미	표기법
프로세스(Process)	자료를 변환시키는 시스템의 한 부분을 나타낸다.	원 안에 프로세스 이름
자료 흐름(Data Flow)	자료의 이동(흐름)을 나타낸다.	자료 이름 아래 —▶
자료 저장소(Data Store)	시스템에서의 자료 저장소(파일, 데이터베이스)를 나타낸다.	가로 수평선 사이 자료 저장소 이름
단말(Terminator)	- 자료의 발생지와 종착지를 나타낸다. - 시스템의 외부에 존재하는 사람이나 조직체이다.	직사각형 안에 단말 이름

소단위 명세서(Mini-Specification)

세분화된 자료 흐름도에서 최하위 단계 프로세스의 처리 절차를 설명한 것이다.
세분화된 자료 흐름도에서 최하위 단계 버블(프로세스)의 처리 절차를 기술한 것으로 프로세스 명세서라고도 한다.
분석가의 문서이며, 자료 흐름도(DFD)를 지원하기 위하여 작성한다.
서술 문장, 구조적 언어, 의사 결정 나무, 의사 결정 표(판단표), 그래프 등을 이용하여 기술한다.
구조적 언어, 의사 결정 나무, 의사 결정표

구조적 언어 : 자연어 일부분으로 한정된 단어와 문형, 제한된 구조를 사용하여 명세서를 작성하는데 이용하는 명세 언어이다.
의사 결정 나무 : 현재 상황과 목표와의 상호 관련성을 나무의 가지를 이용해 표현한 것으로 불확실한 상황에서의 의사결정을 위한 분석 방법이다.
의사 결정표(Decision Table) : 복잡한 의사결정 논리를 기술하는데 사용하며, 주로 자료 처리 분야에서 이용된다.
자료 사전(DD : Data Dictionary)

시스템과 관련된 모든 자료의 명세와 자료 석성을 파악할 수 있도록 조직화한 도구이다.
표기법
기호 의미 설명
= 자료의 정의 ~로 구성되어 있다.(is compose of).
+ 자료의 연결 그리고(and, along with)
( ) 자료의 생략 생략 가능한 자료(optional)
[|] 자료의 선택 다중 택일(selection), 또는(or)
{ } 자료의 반복(iteration of) { }n : n 의 위치가 중괄호 하단에 위치할 때는 최소 n 번 이상 반복
중괄호 상단에 위치할 때는 최대 n 번 이하 반복
위에 n 아래 m 인 경우 m 번 이상 n 번 이하 반복
* * 자료의 설명 주석(comment)
자료 사전의 역할

자료 흐름도에 기술한 모든 자료의 정의를 기술한 문서이다.
구조적 시스템 방법론에서 자료 흐름도, 소단위 명세서와 더불어 중요한 분석 문서 중 하나이다.
자료 사전 이해도를 높이고자 할 때는 하향식 분할 원칙에 맞추어 구성 요소를 재정의한다.
자료 사전에서 기술해야 할 자료

자료 흐름을 구성하는 자료 항목, 자료 저장소를 구성하는 자료 항목, 자료에 대한 의미, 자료 원소의 단위 및 값 등이 있다.

12. 모듈

모듈과 결합도, 응집도

모듈

전체 프로그램에서 어떠한 기능을 수행할 수 있는 실행 코드를 의미한다.
재사용이 가능하며 자체적으로 컴파일할 수 있다.
시스템 개발 시 기간과 노동력을 절감할 수 있다.
모듈의 독립성은 결합도와 응집도에 의해 측정된다.
서브루틴 = 서브 시스템 = 작업 단위
변수의 선언을 효율적으로 할 수 있어 기억 장치를 유용하게 사용할 수 있다.
모듈마다 사용할 변수를 정의하지 않고 상속하여 사용할 수 있다.
각 모듈의 기능이 서로 다른 모듈과의 과도한 상호 작용을 회피함으로서 이루어지는 것을 기능적 독립성이라 한다.

결합도(Coupling)

서로 다른 두 모듈 간의 상호 의존도로서 두 모듈 간의 기능적인 연관 정도를 나타낸다.
모듈 간의 결합도를 약하게 하면 모듈 독립성이 향상되어 시스템을 구현하고 유지보수 작업이 쉬워진다.
자료 결합도가 설계 품질이 가장 좋다.

결합도 수준 분류 특징

결합도 약함 자료 결합도(Data Coupling) - 모듈 간의 인터페이스가 자료 요소로만 구성된 경우로 다른 모듈에 영향을 주지 않는 가장 바람직한 결합도이다.
- 모듈 간의 내용을 전혀 알 필요가 없다.

스탬프 결합도(Stamp Coupling) - 두 모듈이 같은 자료 구조를 조회하는 경우의 결합도이며, 자료 구조의 어떠한 변화 즉 포맷이나 구조의 변화는 그것을 조회하는 모든 모듈 및 변화되는 필드를 실제로 조회하지 않는 모듈까지도 영향을 미치게 된다.
- 배열, 레코드, 구조 등이 모듈 간 인터페이스로 전달되는 경우와 관계된다.

결합도 보통 제어 결합도(Control Coupling) 어떤 모듈이 다른 모듈의 내부 논리 조작을 제어하기 위한 목적으로 제어 신호를 이용하여 통신하는 경우이며, 하위 모듈에서 상위 모듈로 제어 신호가 이동하여 상위 모듈에 처리 명령을 부여하는 권리 전도 현상이 발생하게 된다.

외부 결합도(External Coupling) 어떤 모듈에서 외부로 선언한 변수(데이터)를 다른 모듈에서 참조할 경우와 관계된다.

공통 결합도(Common Coupling) 여러 모듈이 공통 자료 영역을 사용하는 경우로 공통 데이터 영역 내용을 수정하면 이 데이터를 사용하는 모든 모듈에 영향을 준다.
결합도 강함 내용 결합도(Content Coupling) - 가장 강한 결합도를 가지고 있으며, 한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 조회하도록 설계되었을 경우와 관계된다.
- 한 모듈에서 다른 모듈의 내부로 제어 또는 이동된다.
- 한 모듈이 다른 모듈 내부 자료의 조회 또는 변경이 가능하다.
- 두 모듈이 같은 문자(Literals)의 공유가 가능하다.
응집도(Cohesion)

명령어, 명령어의 모임, 호출문, 특정 작업 수행 코드 등 모듈 안의 요소들이 서로 관련된 정도를 말한다.
구조적 설계에서 기능 수행 시 모듈 간 최소한의 상호작용을 하여 하나의 기능만을 수행하는 정도를 표현한다.
모듈이 독립적인 기능으로 구성됨의 정도를 의미한다.
응집도가 높다는 것은 필요한 요소들로 구성됨을 의미한다.
응집도가 낮다는 것은 요소 간의 관련성이 적음을 의미한다.
응집도 분류 특징
응집도 약함 우연적 응집도(Coincidental Cohesion) 모듈 내부의 각 기능 요소들이 서로 관련이 없는 요소로만 구성된 경우와 관계된다.
논리적 응집도(Logical Cohesion) 유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들로 하나의 모듈이 형성되는 경우와 관계된다.
시간적 응집도(Temporal Cohesion) 특정 시간에 처리되는 여러 기능을 모아 한 개의 모듈로 작성할 경우와 관계된다.
절차적 응집도(Procedural Cohesion) 모듀리 다수의 관련 기능을 가질 때 모듈 내부의 기능 요소들이 그 기능을 순차적으로 수행할 경우와 관계된다.
교환적 응집도(Communicational Cohesion) 같은 입력과 출력을 사용하는 소 작업이 모인 경우와 관계된다.
순차적 응집도(Sequential Cohesion) 한 모듈 내부의 한 기능 요소에 의한 출력 자료가 다음 기능 요소의 입력 자료로 제공되는 경우와 관계된다.
응집도 강함 기능적 응집도(Functional Cohesion) 모듈 내부의 모든 기능 요소들이 한 문제와 연관되어 수행되는 경우와 관계된다.
모듈 설계의 특징

바람직한 소프트웨어 설계는 응집도는 강하게, 결합도는 약하게 설계하여 모듈의 독립성을 확보할 수 있도록 한다.

유지보수가 수월해야 하며 복잡도와 중복을 피하며 입구와 출구는 하나씩 갖도록 한다.
모듈 간의 접속 관계를 분석하여 복잡도와 중복을 줄인다.
모듈 간의 효과적인 제어를 위해 설계에서 계층적 자료 조직이 제시되어야 한다.
적당한 모듈의 크기를 유지하고 모듈 간의 접속 관계를 분석하여 복잡도와 중복을 줄인다.
모듈의 크기가 작으면 모듈 개수가 증가하여 모듈 간 통합 비용이 증가하고, 모듈의 크기가 크면 단위 모듈 개발에 큰 비용과 시간이 소요된다.
모듈 독립성이 높다는 것은 단위 모듈을 변경하더라도 타 모듈에 영향이 적다는 의미이며, 오류 발견과 해결이 쉬워진다.
모듈과 컴포넌트
모듈 vs 컴포넌트

모듈 컴포넌트
- 자신만으로 동작할 수 있는 명령의 집합이다.
- 실제로 가장 맨 앞에 위치하는 구현된 단위이며, 자료 구조, 알고리즘 등 이를 제공하는 인터페이스이다.
- 정의하지 않는 이상 바로 재활용을 할 수 없다. - SW 시스템에서 독립적인 업무 또는 기능을 수행하는 모듈로 교체가 가능한 부품이다.
- 모듈화로 생산성을 향상했으나 모듈의 소스 코드 레벨의 재활용으로 인한 한계성을 극복하기 위하여 등장하였다.
- 인터페이스를 통해서 연결된다.
모듈, 컴포넌트, 서비스 특징 비교

구분 모듈 컴포넌트
주요 목적 소프트웨어 복잡도 해소 소프트웨어 재사용성 향상
재사용 단위 소스 코드 실행 코드
독립성 - 구현 언어 종속적
- 플랫폼에 종속적 - 구현 언어 종속적
- 동일 플랫폼 기반 개별적 연계
응용 단일 애플리케이션 분산 애플리케이션
중심사상 모듈화, 추상화 객체지향, CBD
호출 방법 함수 호출 구현 기술 인터페이스
서비스 특징 여러 모듈이 하나의 애플리케이션을 형성하는 계층 구조 다른 컴포넌트와 커뮤니케이션 네트워크를 이루면서 서비스
모듈 분할의 특징

설계의 질을 측정할 수 있고 유지보수가 쉽고 재사용성이 쉽다.

모듈 분할 시 영향을 주는 설계 형태 : 추상화(Abstraction), 모듈화(Modularity), 정보 은폐(Information Hiding), 복잡도(Complexity), 시스템 구조(System Structure)
재사용과 공통 모듈
재사용

검증된 기능을 파악하여 재구성하는 것을 의미한다.
모듈을 최적화하여 타 시스템에 적용하면 개발 비용과 기간을 낮출 수 있다.
생산성 및 소프트웨어의 품질이 향상된다.
재사용 시 해당 모듈은 외부 모듈과의 응집도는 높고, 결합도는 낮아야 한다.
기존 소프트웨어에 재사용 소프트웨어를 추가하기 어려운 문제점이 발생할 수 있다.
재사용 규모에 따른 구분

함수와 객체 : 클래스, 메서드 단위로 소스 코드 등을 재사용한다.
애플리케이션 : 공통 업무를 처리할 수 있도록 구현된 애플리케이션을 공유하여 재사용한다.
컴포넌트 : 컴포넌트 자체 수정 없이 인터페이스를 통하여 컴포넌트 단위로 재사용한다.
공통 모듈

각 서브 시스템에서 공통으로 사용하는 기능(날짜 처리 등)을 묶어 하나의 공통된 모듈로 개발한다.
모듈 재사용성을 높이고 중복 개발로 인한 낭비를 없애기 위해 설계 단계에서 공통 모듈을 분리한다.
같은 기능을 재사용함으로 기능에 대한 정합성 유지 및 중복 개발을 방지할 수 있다.
유지보수 단계에서도 모듈 변경을 통하여 관련된 시스템을 일괄 변경할 수 있다.
재사용 범위에 따른 분류 : 함수와 객체 재사용, 컴포넌트 재사용, 애플리케이션 재사용
공통 모듈 - 명세 기법

정확성 (Correctness)
명확성 (명료성, Clearity)
완전성 (Completeness)
일관성 (Consistency)
추적성 (Traceability)
실제 구현 시 꼭 필요한 기능인지 확인할 수 있도록 정확히 작성한다. 해당 기능에 대한 일관된 이해와 하나로 해석될 수 있도록 작성한다. 시스템 구현 시 필요한 것, 요구되는 것을 모두 작성한다. 공통 기능 간 서로 충돌이 발생하지 않도록 작성한다.

공통 기능에 대한 요구사항 출처, 관련 시스템이 유기적 관계 구분이 가능하도록 작성한다.
모듈 명세화 도구

흐름도(Flowchart), N-S 도표(Nassi-Schneiderman Chart), 의사 코드(Pseudo Code), 의사 결정표(Decision Table), 의사 결정도(Decision Diagram), PDL(Program Design Language), 상태 전이도(State Transition Diagram), 행위도(Action Diagram)
N-S 도표(Nassi-Schneiderman Chart)

구조적 프로그램의 순차, 선택, 반복의 구조를 사각형으로 도식화하여 알고리즘을 논리적 기술에 중점을 둔 도형식 표현 방법이다.
조건이 복합되어 있는 곳의 처리를 시각적으로 명확히 식별하는데 적합하다.
제어 구조 : 순차(연속, Sequence), 선택 및 다중 선택(IfThenElse, Case), 반복(Repeat~Until, While, For)
주로 박스다이어그램을 사용하여 논리적인 제어 구조로 흐름을 표현한다.
조건이 복합되어 있는 곳의 처리를 시각적으로 명확히 식별하는데 적합하다.
13. Software Architecture
소프트웨어 아키텍처(Software Architecture)
Software Architecture 의 개요

요구사항을 기반으로 개발 대상 소프트웨어의 기본 틀(뼈대)을 만드는 것이다.
다수의 이해관계자가 참여하는 복잡한 개발에서 상호 이해, 타협, 의사소통을 체계적으로 접근하기 위한 것이다.
전체 시스템의 전반적인 구조를 체계적으로 설계하는 것이다.
권형도(2004) : "소프트웨어를 구성하는 컴포넌트들의 상호작용 및 관계, 각각의 특성을 기반으로 컴포넌트들이 상호 유기적으로 결합하는 소프트웨어의 여러 가지 원칙들의 집합"이다.
역할 : 설계 및 구현을 위한 구조적/비구조적인 틀(Frame)을 제공한다.
Structure Frame : 시스템 개발을 위하여 결정된 컴포넌트의 구조 모델이다.
Non Structure frame : 해당 구조 모델 이외 다른 아키텍처 설계의 결정들이다.
Software Architecture 시스템 품질 속성 7

성능, 사용 운용성, 보안성, 시험 용이성, 가용성, 변경 용이성, 사용성
Software Architecture 특징

간략성 : 이해하고 추론할 수 있을 정도로 간결해야 한다.

추상화 : 시스템의 추상적인 표현을 사용한다.
가시성 : 시스템이 포함해야 하는 것들을 가시화해야 한다.
복잡도 관리 종류 : 과정 추상화, 데이터 추상화, 제어 추상화
Software Architecture 평가 기준

시스템은 어떻게 모듈로 구성되는가?
시스템은 실행 시에 어떻게 행동하고 연결되는가?
시스템은 어떻게 비 소프트웨어 구조(CPU, 파일 시스템, 네트워크, 개발팀 등)와 관계하고 있는가?
아키텍처 프레임워크 구성 요소

프레임워크(FrameWork) : 복잡한 소프트웨어 문제를 해결하거나 서술하는데 필요한 기본 구조를 제공함으로써 재사용이 가능하게 해준다.
요소 설명
Architecture Description(AD) - 아키텍처를 기록하기 위한 산출물이다.
- 하나의 AD 는 System 의 하나 이상의 View 로 구성한다.
이해관계자(Stakeholder) 소프트웨어 시스템 개발에 관련된 모든 사람과 조직을 의미하며, 고객, 개발자, 프로젝트 관리자 등을 포함한다.
관심사(Concerns) 같은 시스템에 대해 서로 다른 이해관계자의 의견이다.
ex. 사용자 입장 : 기본 기능 + 신뢰성/보안성 요구
관점(View point) 서로 다른 역할이나 책임으로 시스템이나 산출물에 대한 서로 다른 관점이다.
뷰(View) View: 이해관계자들과 이들이 가지는 생각이나 견해로부터 전체 시스템을 표현(4+1 View)한다.
소프트웨어 아키텍처 4+1 View Model

Kruchten 에 의해 Object 표기법을 사용하다가 1995 년 Booch 의 UML 이 정의되면서 Booch 표기법을 포함하여 4+1 이 되었다.
다양하고 동시적인 View 를 기반으로 소프트웨어 위주 시스템의 아키텍처를 묘사하는 View 모델이다.
복잡한 소프트웨어 아키텍처를 다양한 이해관계자들이 바라보는 관점으로, 다양한 측면을 고려하기 위하여 다양한 관점을 바탕으로 정의한 모델이다.
Logical View(분석 및 설계), Implementation View(프로그래머), Process View(시스템 통합자), Deployment View(시스템 엔지니어), Use Case View(사용자)의 5 계층으로 분류한 모델
소프트웨어 아키텍처 설계 원리

구분 설명
단순성 다양한 요소를 단순화하여 복잡성을 최소화한다.
효율성 활용 자원의 적절성과 효율성을 높인다.
분할, 계층화 다루기 쉬운 단위로 묶어서 계층화한다.
추상화 부가적인 기능이 아닌 핵심 기능 위주로 컴포넌트를 정의한다.
모듈화 내부 요소의 응집도를 높이고 각 모듈의 외부 결합도를 낮춘다.
소프트웨어 아키텍처 평가 방법론의 종류

방법 설명
SAAM - Software Architecture Analysis Method의 약어로, 최초 정리된 평가 방법이다.
- 다양한 수정 가능성(Modificability) 관점에서 아키텍처를 평가하고 분석하는 방법이다.
- 수정/변경에 필요한 자원을 가정하고 이를 기반으로 평가한다.
- ATAM에 비하여 상세하지는 않지만 보다 많은 영역에 적용할 수 있다.
ATAM - Architecture Trade off Analysis Method의 약어로 SAAM을 승계한 방법론이다.
- 아키텍처가 품질 속성을 만족하는지 판단하고, 어떻게 절충(TradeOff)하면서 상호작용하는지 분석하는 평가 방법이다.
- 모든 품질 속성을 평가하고, 관심 있는 모든 관련 당사자들이 참여한다.
- 정량적/정성적 분석/평가 수행하며, 민감점(Sensitivity Point)과 절충점(TradeOff Point)을 찾는 데 중점을 둔다.
CBAM - Cost Benefit Analysis Method의 약어로 ATAM에서 경제적인 부분을 보완한 형태이다.
- 소프트웨어 아키텍처를 ROI 관점에서 평가하며 시스템이 제공하는 품질에서 경제적 이득 측면을 고려한다.
- 비용, 이익을 기반으로 ROI를 계산하여 수익이 최대화되는 소프트웨어 아키텍처를 선정한다.
ARID - Active Review for intermediate Design, ATAM과 ADR(Active Design Review)를 혼합한 형태이다.
- 전체 아키텍처가 아닌 한 부분에 대한 품질 요소에 집중하여 평가를 진행한다.
14. 소프트웨어 아키텍처 패턴
소프트웨어 아키텍처 패턴
아키텍처 패턴

소프트웨어 아키텍처를 설계하는데 발생하는 문제점을 해결하기 위한 재사용 가능한 솔루션으로 디자인 패턴과 유사하나 더 큰 범위에 속한다.
종류

Layered, Client-Server, Mater-Slave, Pipe-Filter, Broker, Peer to Peer, Event-Bus, MVC(Model View Controller), Blackboard, Interpreter
장점

개발 시간 단축, 고품질 소프트웨어, 안정적 개발 가능, 개발 단계 관계자 간 의사소통이 간편함. 시스템 구조 이해도가 높아 유지보수에 유리하다.
아키텍처 패턴 = 아키텍처 스타일 = 표준 아키텍처
계층(Layered) 패턴

소프트웨어를 계층 단위(Unit)로 분할하며, N-tier 아키텍쳐 패턴이라고도 한다.
계층적으로 조직화할 수 있는 서비스로 구성된 애플리케이션에 적합하다.
전통적인 방법으로 층 내부의 응집도를 높이는 것이 중요하다.
모듈들의 응집된 딥합 계층 간의 관계는 사용 가능의 관계로 표현한다.
장점 : 정보은닉의 원칙 적용, 높은 이식성을 가진다.
단점 : 추가적인 실행 시 오버헤드(너무 많은 계층으로 성능 감소 발생)가 발생한다.
활용 : 일반적인 데스크톱 소프트웨어나 E-Commerce 웹 어플리케이션
4 계층
Presentation Layer = UI 계층(UI Layer)
Application Layer = 서비스 계층(Service Layer)
Business Logic Layer = 도메인 계층(Domain Layer)
Data access Layer = 영속 계층(Persistence Layer)
MVC(Model View Controller) 패턴

대화형 애플리케이션을 아래와 같이 3 부분으로 분류한다.
Model View Controller
핵심 기능 + 데이터 사용자에게 정보를 표시한다.(다수 뷰가 정의될 수 있다.)
      사용자로부터 입력을 처리한다.
장점 : 같은 모델에서 다수의 뷰를 생성할 수 있으며, 실행 시간에 동적으로 연결 및 해제할 수 있다.
단점 : 사용자 행동에 대한 불필요 업데이트가 발생할 수 있으며 복잡성이 증가할 수 있다.
활용 : 일반적인 웹 애플리케이션 설계 아키텍처, Django 나 Rails 와 같은 웹 프레임워크
클라이언트 서버(Client Server) 패턴

하나의 서버와 다수 클라이언트로 구성되며, 클라이언트가 서버에 서비스를 요청하면 커뮤니케이션이 이루어진다. 서버는 응답을 위해 항상 대기 중이어야 한다.
여러 컴포넌트에 걸쳐서 데이터와 데이터를 처리하는 애플리케이션에 적합하다.
장점 : 직접 데이터 분산, 위치 투명성을 제공한다.
단점 : 서비스와 서버의 이름을 관리하는 레지스터가 없어 이용 가능한 서비스 시간에 불편함을 초래한다.
활용 : 이메일, 문서 공유, 은행 등 온라인 애플리케이션
파이프 필터(Pipe-Filters)

데이터 흐름(Data Stream - 데이터 송/수신이나 처리의 연속적 흐름)을 생성하고 처리하는 시스템을 위한 구조이다.
필터는 파이프를 통해 받은 데이터를 변경시키고 그 결과를 파이프로 전송한다.
각 처리 과정은 필터 컴포넌트에서 이루어지며, 처리되는 데이터는 파이프를 통해 흐른다.
이 파이프는 버퍼링 또는 동기화 목적으로 사용될 수 있다.
컴파일러, 연속한 필터들은 어휘 분석, 파싱, 의미 분석 그리고 코드 생성을 수행한다.
장점 : 필터 교환과 재조합을 통해서 높은 유연성을 제공한다.
단점 : 상태정보 공유를 위해 비용이 소요되며 데이터 변환에 과부하가 걸릴 수 있다.
활용 : 컴파일러, 어휘 분석, 구문 분석, 의미 분석, 코드 생성
Peer To Peer

분산 컴퓨팅 애플리케이션 구축 시 유연성을 제공한다.
클라이언트/서버 스타일레 대칭적 특징을 추가한 형태이다.
Peer 가 하나의 컴포넌트로 대응되며 컴포넌트는 클라이언트, 서버 역할 모두 수행한다.
브로커(Broker)

Apache ActiveMQ, Apache Kafka, RabbitMQ, JBoss Messaging 과 같은 메시지 브로커 소프트웨어에 활용
컴포넌트가 컴퓨터와 사용자를 연결해 주는 역할을 하며, 분산 시스템에 주로 사용된다.
요청에 응답하는 컴포넌트들이 여러 개 존재할 때 적합하다.
블랙보드(Black Board)

결정적 해결 전략이 존재하지 않는 문제 해결에 적합하며 음성인식, 신호해석 등에 활용된다.
블랙보드의 데이터를 컴포넌트에서 검색을 통하여 찾을 수 있다.
장점 : 다양한 접근법, 유지보수성, 가변성, 재사용 가능한 지식 자원
단점 : 테스팅 어려움, 완전한 해결책을 보장하지 못함

이벤트 버스(Event - Bus)

이벤트 버스 : 이벤트 생성(소스), 이벤트 수행(리스너), 이벤트 통로(채널), 채널 관리(버스)
소스 이벤트가 메시지를 발행하면 해당 채널 구독자가 메시지 수신 후 해당 이벤트를 처리하는 방식으로 주로 이벤트를 처리하며 이벤트 소스, 이벤트 리스너(Event Listener), 채널, 이벤트 버스 등 4 가지 주요 컴포넌트들을 갖는다.
소스는 이벤트 버스를 통해 특정 채널로 메시지를 발행하고, 리스너는 특정 채널에서 메시지를 구독한다. 리스너는 이전에 구독한 채널에 발행된 메시지에 대해 알림을 받는다.
인터프리터(InterPreter)

SQL 과 같은 데이터베이스 쿼리 언어, 통신 프로토콜을 정의하기 위한 언어
특정 언어로 작성된 프로그램을 해석하는 컴포넌트를 설계할 때 사용된다.
주로 특정 언어로 작성된 문장 혹은 표현식이라고 하는 프로그램의 각 라인을 수행하는 방법을 지정한다. 기본 아이디어는 언어의 각 기호에 대해 클래스를 만드는 것이다.
15. 객체지향 설계
구조적, 절차적 프로그래밍과 객체지향
구조적 프로그래밍(Structured Programming)

프로그램의 이해가 쉽고 디버깅 작업이 쉽다.
한 개의 입구(입력)와 한 개의 출구(출력) 구조를 갖도록 한다.
GOTO(분기) 문은 사용하지 않는다.
구조적 프로그래밍의 기본 구조 : 순차(Sequence) 구조, 선택(Selection) 구조, 반복(Iteration) 구조
절차적 프로그래밍(Procedural Programming)

순서대로 일련의 명령어를 나열하여 프로그래밍한다.
Function 기반의 프로그래밍이며, 프로시저로써 Function 외에도 Subroutine 이 문법적으로 구현되어 있다.
절차형 언어의 경우 규모가 커지면 커질수록 함수가 기하급수적으로 늘어난다.
함수가 타 프로그램과 문제를 일으킬 수 있는 문제점을 가지고 있다.
프로그램과 별개로 데이터 취급이 되므로 완전하지 않고 현실 세계 문제를 프로그램으로 표현하는데 제약이 있다.
객체지향 프로그래밍(Object Oriented Programming)

컴퓨터 소프트웨어를 구조적인 코드 단위로 보는 것이 아니라 Object 단위로 구분하고 Object 간의 모음으로 설계하는 것이다.

소프트웨어 내의 Object 는 서로 Message 를 주고받는다.
처리 요구를 받은 객체가 자기 자신 안에 있는 내용을 가지고 처리하는 방식이다.
프로그램이 단순화되고 생산성, 신뢰성이 높아져 대규모 개발에 많이 사용된다.
객체지향 구성 요소

구분 설명
Class - 유사한 객체를 정의한 집합으로 속성+행위를 정의한 것으로 일반적인 Type 을 의미한다.
- 기본적인 사용자 정의 데이터형이며, 데이터를 추상화하는 단위이다.
- 구조적 기법에서의 단위 테스트(Unit Test)와 같은 개념이다.
- 상위 클래스(부모 클래스, Super Class), 하위 클래스(자식 클래스, Sub Class)로 나뉜다.
Object - 데이터와 함수를 묶어 캡슐화하는 대상이 된다.
- Class 에 속한 Instance 를 Object 라고 한다.
- 하나의 소프트웨어 모듈로서 목적, 대상을 표현한다.
- 같은 클래스에 속한 각각의 객체를 instance 라고 한다.
Message Object 간에 서로 주고받는 통신을 의미한다.
Attribute : Object 가 가지고 있는 데이터 값
Method : Object 의 행위인 함수
Class = 틀 = Type
객체지향의 특징(캡상다추정)
캡슐화
(Encapsulation) 상속성
(Inheritance) 다형성
(Polymorphism) 추상화
(Abstraction) 정보은닉
(Information Hiding)
- 서로 관련성이 높은 데이터(속성)와 그와 관련된 기능(메서드, 함수)을 묶는 기법이다.
- 결합도가 낮아져 소프트웨어 개발에 있어 재사용성이 높아진다.
- 정보은닉을 통하여 타 객체와 메시지 교환 시 인터페이스가 단순해진다.
- 변경 발생 시 오류의 파급 효과가 적다.- 상위 클래스의 모든 속성, 연산을 하위 클래스가 재정의 없이 물려받아 사용하는 것이다.
- 상위 클래스는 추상적 성질을, 자식 클래스는 구체적 성질을 가진다.
- 하위 클래스는 상속받은 속성과 연산에 새로운 속성과 연산을 추가하여 사용할 수 있다.
- 다중 상속 : 다수 상위 클래스에서 속성과 연산을 물려받는 것이다. - 객체가 다양한 모양을 가지는 성질을 뜻한다.

- 오퍼레이션이나 속성의 이름이 하나 이상의 클래스에서 정의되고 각 클래스에서 다른 형태로 구현될 수 있는 개념이다.
- 속성이나 변수가 서로 다른 클래스에 속하는 객체를 지칭할 수 있는 성질이다. - 시스템 내의 공통 성질을 추출한 뒤 추상 클래스를 설정하는 기법이다.
- 현실 세계를 컴퓨터 시스템에 자연스럽게 표현할 수 있다.
- 종류 : 기능 추상화, 제어 추상화, 자료 추상화 - 객체 내부의 속성과 메서드를 숨기고 공개된 인터페이스를 통해서만 메시지를 주고받을 수 있도록 하는 것을 의미한다.
- 예기치 못한 Side Effect 를 줄이기 위해서 사용한다.

객체지향 기법에서의 관계성

is member of : 연관성(Association), 참조 및 이용 관계
is part of : 집단화(Aggregation), 객체 간의 구조적인 집약 관계
is a : 일반화(Generalization), 특수화(Specialization), 클래스 간의 개념적인 포함 관계

오버로딩(Overloading)

사전적 의미 : 과적, 과부하
한 클래스 내에서 같은 이름의 메서드를 사용하는 것이다.
같은 이름의 메서드를 여러 개 정의하면서 매개 변수의 유형과 개수가 달라지도록 하는 기술이다.

오버라이딩(Overriding)

사전적 의미 : 가장 우선되는, 최우선으로 되는, 다른 것보다 우선인
상속 관계의 두 클래스의 상위 클래스에서 정의한 메서드를 하위 클래스에서 변경(재정의)하는 것이다.
JAVA 언어에서는 static 메서드의 오버라이딩을 허용하지 않는다.
오버라이딩의 경우 하위 객체의 매개 변수 개수와 타입은 상위 객체와 같아야 한다.

오버로딩 vs 오버라이딩

구분	오버로딩 (OverLoading)	오버라이딩 (Overriding)
메서드 이름	한 클래스 내에서 같다.	상속 관계의 두 클래스 간 같다.
매개 변수 개수 / 매개 변수 타입	매개 변수 타입 또는 개수가 달라야 한다.	반드시 같아야 한다.
접근 제한	무관	범위는 같거나 커야 한다.
사용	같은 이름으로 메서드 중복 정의	자식 클래스에서 부모 클래스의 메서드 재정의

객체지향 설계 원칙(SOLID)

단일 책임의 원칙
(SRP : Single Responsibility Principle)
모든 클래스는 단일 목적으로 생성되고, 하나의 책임만 가져야 한다.
개방 - 폐쇄의 원칙
(OCP : Open Closed Principle)
소프트웨어 구성 요소는 확장에 대해서는 개방되어야 하나 수정에 대해서는 폐쇄적이어야 한다.
리스코프치환 원칙
(LSP : Liskov Substitution Principle)
부모 클래스가 들어갈 자리에 자식 클래스를 대체하여도 계획대로 작동해야 한다.
인터페이스 분리 원칙
(ISP : Interface Segregation Principle)
클라이언트는 자신이 사용하지 않는 메서드와 의존 관계를 맺으면 안 된다.
클라이언트가 사용하지 않는 인터페이스 때문에 영향을 받아서는 안 된다.
의존 역전 원칙
(DIP : Dependency Inversion Principle)
의존 관계를 맺으면 변하기 쉽고 변화 빈도가 높은 것보다 변하기 어렵고 변화 빈도가 낮은 것에 의존한다.
객체지향 개발 방법론
객체지향 개발 방법론

종류 설명 특징
Booch - OOD(Object Oriented Design)
- 설계 부분만 존재하며 문서화를 강조하여 다이어그램 기반으로 개발되었다. - 분석과 설계가 분리되지 않는다.
- 정적 모델과 동적 모델로 표현된다.
OOSE
(Jacobson) - Object Oriented SW Engineering
- Use Case 의 한 접근 방법
- Use Case 를 모든 모델의 근간으로 활용된다. - 분석, 설계 및 구현으로 구성된다.
- 기능적 요구사항 중심이다.
- 시스템 변화에 유연하다.
OMT
(Rumbaugh) - Object Modeling Technology
- 객체지향 분석, 시스템 설계, Object 설계/구현 4 단계로 구성된다.
- 객체 모델링 : 객체도를 이용하여 시스템의 정적 구조를 표현한다.
- 동적 모델링 : 상태 도를 이용하여 객체의 제어 흐름/상호 반응을 표현한다.

- 기능 모델링 : 자료 흐름도를 이용하여 데이터값의 변화 과정을 표현한다. - 복잡한 대형 개발 프로젝트에 유용하다.
- 기업 업무의 모델링에 있어 편리하고 사용자와 의사소통이 원활하다.
- Case 와 연동이 충실하다.
Coad 와 Yourdon 방법 객체지향 분석 방법론에서 E-R 다이어그램을 사용하여 객체의 행위를 모델링한다. 객체 식별, 구조 식별, 주체 정의, 속성 및 관계 정의, 서비스 정의 등의 과정으로 구성된다.
클래스 설계
클래스 설계

분석 단계 중 아직 확정되지 않은 클래스 내부 부분 중 구현에 필요한 중요한 사항을 결정하는 작업을 의미한다.
클래스의 서비스 인터페이스에 대한 정확한 정의, 메서드 내부의 로직 등 객체의 상태 변화와 오퍼레이션의 관계를 상세히 설계해야 하며, 클래스가 가지는 속성값에 따라 오퍼레이션 구현이 달라진다.
객체의 상태 변화 모델링은 필수이다.
클래스 인터페이스

관점에 따라 관심이 다르므로, 클래스 인터페이스가 중요하다.
관점이 다른 개발자들이 클래스 명세의 어떤 부분에 관심이 있는가?
클래스 구현 : 실제 설계로부터 클래스를 구현하려는 개발자
클래스 사용 : 구현된 클래스를 이용하여 다른 클래스를 개발하려는 개발자
클래스 확장 : 구현된 클래스를 확장하여 다른 클래스로 만들고자 하는 개발자
협약에 의한 설계(Design by Contract) 3 가지 타입

선행조건(Precondition) : 오퍼레이션이 호출되기 전에 참이 되어야 할 조건
결과조건(Postcondition) : 오퍼레이션이 수된 후 만족해야 하는 조건
불변조건(Invariant) : 클래스 내부가 실행되는 동안 항상 만족하여야 하는 조건
16. 디자인 패턴
디자인 패턴
디자인 패턴

자주 사용하는 설계 형태를 정형화하여 유형별로 설계 템플릿을 만들어 두고 소프트웨어 개발 중 나타나는 과제를 해결하기 위한 방법 중 한 가지다.
다양한 응용 소프트웨어 시스템들을 개발할 때 서로 간에 공통점이 있으며, 이러한 유사점을 패턴이라 한다.

개발자 간 원활한 의사소통, 소프트웨어 구조 파악 용이, 설계 변경에 대한 유연한 대처, 개발의 효율성, 유지보수성, 운용성 등 소프트웨어 품질 향상에 도움을 준다.
객체지향 프로그래밍 설계 시 유사한 상황에서 구조적인 문제를 해결할 수 있도록 방안을 제공해주며, Gof(Gang of Four) 분류가 가장 많이 사용된다.
디자인 패턴을 사용할 때의 장/단점

장점

개발자 간의 월활한 의사소통을 지원한다.
소프트웨어 구조 파악이 쉽다.
재사용을 통한 개발 시간을 단축할 수 있다.
설계 변경 요청에 대해 유연하게 대처할 수 있다.
객체지향 설계 및 구현의 생산성을 높이는 데 적합하다.
단점

객체지향 설계/구현 위주로 사용된다.
초기 투자 비용이 부담된다.
디자인 패턴의 구성 요소

필수 요소

패턴의 이름 : 패턴을 부를 때 사용하는 이름과 패턴의 유형
문제 및 배경 : 패턴이 사용되는 분야 또는 배경, 해결하는 문제를 의미
해법 : 패턴을 이루는 요소들, 관계, 협동(Collaboration) 과정
결과 : 패턴을 사용하면 얻게 되는 이점이나 영향
추가 요소

알려진 사례 : 간단한 적용 사례
샘플 코드 : 패턴이 적용된 원시 코드
원리, 정당성, 근거
GoF(Gangs of Four) 디자인 패턴

에릭 감마, 리처드 헬름, 랄프 존슨, 존 브리시데스가 제안
객체지향 설계 단계 중 재사용에 관한 유용한 설계를 디자인 패턴화하였다.
생성 패턴, 구조 패턴, 행위 패턴으로 분류한다.
생성 패턴

객체를 생성하는 것과 관련된 패턴이다.

객체의 생성과 변경이 전체 시스템에 미치는 영향은 최소화하도록 만들어주어 유연성을 높일 수 있고 코드를 유지하기가 쉬운 편이다.
객체의 생성과 참조 과정을 추상화함으로써 시스템을 개발할 때 부담을 덜어준다.
구성
Factory Method
상위 클래스에서 객체를 생성하는 인터페이스를 정의하고, 하위 클래스에서 인스턴스를 생성하도록 하는 방식이다.
Virtual-Constructor 패턴이라고도 한다.
Singleton
전역 변수를 사용하지 않고 객체를 하나만 생성하도록 한다.
생성된 객체를 어디에서든지 참조할 수 있도록 하는 패턴이다.
Prototype
prototype 을 먼저 생성하고 인스턴스를 복제하여 사용하는 구조이다.
일반적인 방법으로 객체를 생성한다.
비용이 많이 소요되는 경우 주로 사용한다.
Builder
작게 분리된 인스턴스를 조립하듯 조합하여 객체를 생성한다.
Abstraction Factory
구체적인 클래스에 의존하지 않고 서로 연관되거나 의존적인 객체들의 조합을 만드는 인터페이스를 제공하는 패턴이다.
관련된 서브 클래스를 그룹 지어 한 번에 교체할 수 있다.
구조 패턴

클래스나 객체를 조합해 더 큰 구조를 만드는 패턴이다.
복잡한 형태의 구조를 갖는 시스템을 개발하기 쉽게 만들어주는 패턴이다.
새로운 기능을 가진 복합 객체를 효과적으로 작성할 수 있다.
ex. 서로 다른 인터페이스를 지닌 2 개의 객체를 묶어 단일 인터페이스를 제공하거나 객체들을 서로 묶어 새로운 기능을 제공하는 패턴이다. 프로그램 내의 자료 구조나 인터페이스 구조 등 구조를 설계하는데 많이 활용된다.
구성
Adapter
클래스의 인터페이스를 사용자가 기대하는 다른 인터페이스로 변환하는 패턴으로, 호환성이 없는 인터페이스 때문에 함께 동작할 수 없는 클래스들이 함께 작동하도록 해준다.
기존에 구현되어 있는 클래스에 기능 발생 시 기존 클래스를 재사용할 수 있도록 중간에서 맞춰주는 역할을 한다.
Bridge
큰 클래스 또는 밀접하게 관련된 클래스들의 집합을 두 개의 개별 계층구조(추상화 및 구현)로 나눈 후 각각 독립적으로 개발할 수 있도록 하는 구조 디자인 패턴.

기능 클래스 계층과 구현 클래스 계층을 연결하고, 구현부에서 추상 계층을 분리하여 각자 독립적으로 변형할 수 있도록 해주는 패턴이다.

Composite(복합체)

객체들을 트리 구조들로 구성한 후, 이러한 구조들과 개별 객체들처럼 작업할 수 있도록 하는 구조 패턴

Decorator

객체들을 새로운 행동들을 포함한 특수 래퍼 객체들 내에 넣어서 위 행동들을 해당 객체들에 연결시키는 구조적 디자인 패턴

Facade(퍼사드)

라이브러리에 대한, 프레임워크에 대한 또는 다른 클래스들의 복잡한 집합에 대한 단순화된 인터페이스를 제공하는 구조적 디자인 패턴

Flyweight

각 객체에 모든 데이터를 유지하는 대신 여러 객체들 간에 상태의 공통 부분들을 공유하여 사용할 수 있는 RAM 에 더 많은 객체들을 포함할 수 있도록 하는 구조 디자인 패턴

Proxy

다른 객체에 대한 대체 또는 자리표시자를 제공할 수 있는 구조 디자인 패턴.
프록시는 원래 객체에 대한 접근을 제어하므로, 자신의 요청이 원래 객체에 전달되기 전 또는 후에 무언가를 수행할 수 있도록 함.

행위 패턴

반복적으로 사용되는 객체들의 상호작용을 패턴화한 것으로, 클래스나 객체들이 상호작용하는 방법과 책임을 분산하는 방법을 정의한다.
메시지 교환과 관련된 것으로, 객체 간의 행위나 알고리즘 등과 관련된 패턴을 말한다.

구성

Chain of Responsibility(책임 연쇄)

핸들러들의 체인(사슬)을 따라 요청을 전달할 수 있게 해주는 행동 디자인 패턴.
각 핸들러는 요청을 받으면 요청을 처리할지 아니면 체인의 다음 핸들러로 전달할지를 결정

Iterator(반복자)

컬렉션의 요소들의 기본 표현(리스트, 스택, 트리 등)을 노출하지 않고 그들을 하나씩 순회할 수 있도록 하는 행동 디자인 패턴.

Command(명령)

요청을 요청에 대한 모든 정보가 포함된 독립실행형 객체로 변환하는 행동 디자인 패턴.
이 변환은 다양한 요청들이 있는 메서드들을 인수화 할 수 있도록 하며, 요청의 실행을 지연 또는 대기열에 넣을 수 있도록 하고, 또 실행 취소할 수 있는 작업을 지원할 수 있도록 함.

Interpreter(해석자)

단순한 언어를 해석할 때 유용한 패턴

간단한 언어의 문법을 정의하고 해석하는 패턴
언어가 주어지면 해당 표현을 사용하여 언어로 문장을 해석하는 인터프리터를 사용하여 문법 표현을 정의하는 방법
Memento(기록)
메멘토 패턴은 객체를 이전 상태로 되돌릴 수 있는 기능을 제공하는 패턴
오리지네이터, 케어테이커, 메멘토 등 3 개의 객체로 구현된다.
오리지네이터: 내부 상태를 보유하고 있는 일부 객체
Observer(감시자)
여러 객체에 자신이 관찰 중인 객체에 발생하는 모든 이벤트에 대해 알리는 구독 매커니즘을 정의할 수 있도록 하는 행동 디자인 패턴(관찰자)
State(상태)
객체의 내부 상태가 변경될 때 해당 객체가 그의 행동을 변경할 수 있도록 하는 행동 디자인 패턴.
객체가 행동을 변경할 때 객체가 클래스를 변경한 것처럼 보일 수 있음
Strategy(전략)
패턴은 알고리즘들의 패밀리를 정의하고, 각 패밀리를 별도의 클래스에 넣은 후 그들의 객체들을 상호교환할 수 있도록 하는 행동 디자인 패턴
Visitor(방문자)
알고리즘들을 그들이 작동하는 객체들로부터 분리할 수 있도록 하는 행동 디자인 패턴
Template Method
부모 클래스에서 알고리즘의 골격을 정의하지만, 해당 알고리즘의 구조를 변경하지 않고 자식 클래스들이 알고리즘의 특정 단계들을 오버라이드(재정의)할 수 있도록 하는 행동 디자인 패턴.
Mediator(중재자)
객체 간의 통제와 지시의 역할을 하는 중재자를 두어 객체지향의 목표를 달성하게 해준다.
Virtual-Constructor 패턴이라고도 한다.
객체 간의 혼란스러운 의존 관계들을 줄일 수 있는 행동 디자인 패턴
패턴은 객체 간의 직접 통신을 제한하고 중재자 객체를 통해서만 협력하도록 함.
디자인 패턴 vs 아키텍처 패턴

아키텍처 패턴이 상위 설계에 이용된다.
아키텍쳐 패턴 : 시스템 전체 구조를 설계하기 위한 참조
디자인 패턴 : 서브 시스템 내 컴포넌트와 그들 간의 관계를 구성하기 위한 참조 모델
17. 인터페이스 요구사항 확인
인터페이스 요구사항
인터페이스 내/외부 요구사항

개발 대상 조직 내/외부의 시스템 연동을 통하여 상호작용을 위한 접속 방법, 규칙을 의미한다.
인터페이스 요구사항 구성

요구사항의 구성, 내/외부 인터페이스 이름, 연계 대상 시스템, 연계 범위 및 내용, 연계 방식, 송신 데이터, 인터페이스 주기, 기타 고려사항
인터페이스 요구사항의 분류

기능적 요구사항 : 소프트웨어가 내/외부 시스템 간의 연계를 통하여 수행될 기능과 관련하여 가져야 하는 기능적 속성에 대한 요구사항이다.
비기능적 요구사항 : 기능에 관련되지 않는 사항으로 기능 요구사항을 만족시키는 바탕에서 정상적으로 작동하기 위한 시스템 내/외부의 제약 조건을 의미한다.
인터페이스 요구사항 명세서

예시

요구사항 분류 시스템 인터페이스 요구사항
요구사항 번호 AAA-MOCK001
요구사항 명칭 소셜로그인 연동
요구사항 상세 설명 정의 네이버 아이디로 회원 연동
세부 내용 - 네이버 아이디 서비스인 네아로 API 를 이용하여 MOCK 사이트 회원 가입에 활용할 수 있도록 한다.
- MOCK 사이트 네아로 서비스 아이디가 네이버에 전달되고, API 를 통하여 회원 정보를 가져온다.
- 예상 트랜잭션 : 일 1,000 건
<추가 정의 내용>
- 네이버 회원 정보 중 이름, 닉네임, 이메일, 전화번호를 가져온다.
- 네아로 서비스를 통하여 회원 가입할 때 회원 아이디를 "NAVER_"로 시작하도록 하여 구분한다.
산출 정보 네아로 API 적용 설명서와 MOCK 사이트 회원 DB 구조
요구사항 출처 고객지원팀
관련 요구사항 AAA-MOCK001
인터페이스 요구사항의 분석 절차

요구사항 명세서에서 기능적인 요구사항과 비기능적인 요구사항을 명세하고 분류한 뒤 구체화하여 이해관계자와 공유하는 과정을 의미한다.

소프트웨어 개발 요구사항 목록에서 시스템 인터페이스와 관련된 요구사항을 선별하여
시스템 인터페이스 요구사항 명세를 작성한다.
시스템 인터페이스와 관련된 요구사항, 아키텍처 정의서, 현행 시스템의 대내외 연계
시스템 현황 등 관련 자료를 준비한다.
시스템 인터페이스 요구사항 명세서를 파악하여 기능적/비기능적 요구사항을 구분한다.
시스템 인터페이스 요구 명세서와 요구사항 목록, 기타 관련 자료를 비교 분석하여 내용을
추가, 수정하여 완성도를 높인다.
앞서 정리된 문서를 이해관계자와 공유한다.
인터페이스 요구사항 검증

인터페이스 설계 및 구현 전 사용자의 요구사항을 명세하고 그 명세가 완전한가를 검토하고
개발 범위를 설정하는 것이다.
인터페이스 요구 명세가 완전하지 않으면 설계 및 구현 단계에서 추가 수정하면
비경제적이다.
검증 절차 : 검토 계획 수립 => 검토, 오류 수정 => 베이스라인 설정
검토계획 수립

프로젝트 규모, 참여 인력, 기간 등을 고려하여 검토 기준 및 방법을 결정하는 단계이다.
품질 관리자, 인터페이스 분석가, 아키텍트, 사용자, 테스터 등 참여자를 선임한다.
완전성, 명확성, 일관성 검토 점검표를 작성한다.
요구사항 명세서, 요구사항 목록, 시스템 구성도, 현행 표준 등 자료를 준비한다.
요구사항 검토 일정을 확정한다.
검토 및 오류 수정

검토 계획 수립 단계에서 수집, 작성된 문서를 검토한다.
검토 중 오류 발생 시 수정할 수 있도록 오류 목록, 시정 조치서를 작성한다.
검토 결과를 관련자에게 전달하여 오류 수정 및 요구사항 승인 절차를 진행한다.
시정 조치가 완료되면 검토 작업을 마무리한ㄷ.ㅏ
베이스라인 설정

검토 및 수정 단계에서 검증된 요구사항을 프로젝트 관리자와 관련 결정자에게 승인받는다.
요구사항 명세서의 베이스라인을 설정한다.
인터페이스 요구사항 검증 방법

프로토타이핑, 테스트 설계, CASE 도구 활용, 요구사항 검토 등의 방법이 있다.
방법 설명

프로토타이핑 요구사항에 대한 이해를 위하여 기본적인 기능만 시제품으로 제공하여 사용자로부터 피드백을 받는 요구사항 분석 기법이다.
테스트 설계 Test Case를 생성하고, 요구사항이 현실적으로 테스트 가능한지 검토한다.
CASE - Computer Aid Software Engineering
- 일관성 분석을 통하여 요구사항 변경의 추적과 분석을 통하여 요구사항을 관리한다.
요구사항 검토
동료 검토 명제 작성자가 동료들에게 설명하고 동료들이 결함을 찾는 방법이다.
워크 스루
(Walk Through) 검토회의 전 명세서 배포 => 짧은 검토 회의 => 결함 발견
인스펙션(Inspection) 소프트웨어 요구, 설계, 원시 코드 등의 작성자 외의 다른 전문가 또는 팀이 검사하여 오류를 찾아내는 공식적 검토 방법이다.
18. 인터페이스 대상 식별
대상 식별
시스템 아키텍처 요구사항

하드웨어, 소프트웨어를 모두 포함하는 전체 시스템에 대한 논리적 기능 체계 그리고 그것을 실현하기 위한 구성 방식, 시스템 전체의 최적화를 목표로 한다.
요구사항과 시스템의 전체 생명주기를 고려한다.
시스템의 구성, 동작 원리를 정확하게 표현해야 하며 각 컴포넌트에 대한 설계, 구현을 지원하는 수준으로 자세히 기술한다.(IEEE 1474, TOGAF).
각 컴포넌트 사이의 상호작용, 외부 환경과의 관계를 기술한다.
시스템 인터페이스 식별

개발 대상 시스템과 연결된 연계 시스템 사이의 인터페이스를 식별하는 것이다.
시스템의 업무 정의서, 시스템 아키텍처 정의서, 유스케이스 정의서 등을 통하여 송신, 수신, 중계 시스템을 식별한다.
인터페이스 요구명세서, 인터페이스 요구사항 목록을 기반으로 개발 대상 시스템과 연계된 내/외부 시스템 사이의 인터페이스 목록을 작성한다.
Interface System Process

송신 시스템 <-> 중계 시스템(송신/수신, 암호화/ 복호화, 변환/맵핑, 응답 처리/완료 처리) <-> 수신 시스템

인터페이스 시스템의 구성

서로 다른 시스템 간의 연결을 의미하며, 송신, 수신, 중계 서버로 구성된다.
송신 시스템 : 연계할 데이터를 테이블, 파일 형태로 생성하고 전송하는 시스템이다.

수신 시스템 : 송신된 데이터를 수신 시스템에서 관리하는 형식의 데이터를 변환하여 DB 에 저장하거나 애플리케이션에 활용할 수 있도록 지원하는 시스템이다.
중계 시스템 : 송/수신 시스템 사이에서 데이터 송/수신 상태를 모니터링하는 시스템이다.
인터페이스 데이터 표준

시스템 사이에 상호 교환되는 데이터는 표준 형식을 정의하여 사용한다.
인터페이스 설계 단계에서 송/수신 시스템 사이의 전송 표준 항목, 업무 처리 데이터, 공통 코드 정보 등을 누락 없이 확인하여 명세서를 작성한다.
인터페이스 데이터 공통부/개별부/종료부로 구성된다.
공통부 : 인터페이스 표준 항목을 포함한다.
개별부 : 송/수신 시스템에서 업무 처리에 필요한 데이터를 포함한다.
종료부 : 전송 데이터의 끝을 표시하는 문자를 포함한다.
전문 공통부(고정 크기)

전문 길이 10 Byte
시스템 공통 246 Byte
거래 공통 256 Byte
전문 개별부(가변 크기)

데이터 n Byte
전문 종료부

전문 종료 2 Byte
인터페이스 상세 설계
내/외부 송/수신 방식

직접 연계 방식 : 중계 서버 또는 솔루션 사용 없이 송/수신 시스템이 직접 인터페이스하는 방식이다.
간접 연계 방식 : 연계 솔루션을 통하여 송/수신 엔진과 어댑터를 이용하여 인터페이스하는 방식이다.
간접 연계 vs 직접 연계 방식의 장단점

구분 설명
간접 장점 - 연계 서버를 활용하여 송/수신 처리 및 현황을 모니터링하고 통제하는 방식이다.
- 네트워크/프로토콜 등 서로 다른 환경을 갖는 시스템을 연계 통합할 수 있다.
- 인터페이스 변경 시 대처가 수월하다.

단점 - 연계 절차가 복잡하고 연계 서버 사용으로 인하여 성능 저하가 발생할 수 있다.
- 개발 및 테스트 기간이 많이 소요된다.
직접 장점 - 연계 절차가 없어 처리 속도가 빠르다.
- 구현이 단순하며 개발 비용과 개발 기간에서 경제적이다.
단점 - 송/수신 시스템 간 높은 결합도로 인하여 시스템 변경에 유연성이 떨어진다.
- 전사 시스템의 통합 환경을 구축하기 어렵다.
- 보안 처리와 업무 로직 구현을 인터페이스별로 작성해야 하는 불편함이 있다.
인터페이스 연계 기술

구분 설명
DB Link - DB 에서 제공하는 DB Link 객체를 이용하는 것이다.
- 수신 시스템의 DB 에서 송신 시스템에 접근 가능한 DB Link 를 생성한 뒤 송신 시스템에서 DB Link 로 직접 참조하여 연계하는 것이다.
DB Connection 수신 시스템 WAS 에서 송신 시스템으로 연결되는 DB Connection Pool 을 생성하고 프로그램 소스에서 이를 사용하는 것이다.
API/Open API 송신 시스템의 DB 에서 데이터를 읽어 제공하는 APPlication Programming interface 를 이용하는 것이다.
JDBC 수신 시스템의 프로그램에서 JDBC 드라이버를 이용하여 송신 시스템 DB 와 연결하여 사용하는 것이다.
Hyper Link 웹 애플리케이션에서 Hyper Link 를 사용하는 방식이다.
Socket 서버에서 통신을 위한 소켓을 생성, 포트를 할당한 뒤 클라이언트의 통신 요청 시 클라이언트와 연결하는 방식이다.
Web Service 웹 서비스에서 WSDL, UDDI, SOAP 프로토콜을 이용하는 방식이다.
연계 솔루션 실제 송/수신 처리와 진행 상황을 모니터링 및 통제하는 EAI 서버, 송/수신 시스템에 설치되는 어댑터(Client)를 이용하는 방식이다.
연계 기술
시스템 연계 기술

API(Application Programming Interface) : 프로그래밍을 통하여 프로그램을 작성하기 위한 일련의 부프로그램, 프로토콜 등을 정의하여 상호작용을 하기 위한 인터페이스 사양을 말한다.
WSDL(Web Services Description Language): 관련된 서식, 프로토콜 등을 웹 서비스를 통해 표준적인 방법으로 기술하고 게시하기 위한 언어이다.
UDDI(Universal Description, Descovery, and Integration) : 인터넷에서 전 세계 비즈니스 목록에 자신을 등재하기 위한 확장성 생성 언어(XML) 기반의 규격화된 레지스트리이다.

SOAP(Single Object Access Protocol) : 웹 서비스를 실제로 이용하기 위한 객체 간의 통신 규약이다.
인터페이스 송/수신 통신 유형

구분 설명
통신유형 단방향 데이터를 요청한 뒤 그에 대한 피드백이 필요 없는 경우이다.
동기 - 데이터를 요청한 뒤 그에 대한 피드백이 올 때까지 대기하는 방식이다.
- 거래량이 적고, 빠른 응답이 요구되는 경우이다.
비동기 - 데이터를 요청한 뒤 그에 대한 피드백이 올 때까지 타 적업을 처리한 뒤 해당 요청을 처리하는 방식이다.
- 거래량이 많거나 데이터 전송 시스템의 처리가 늦는 경우 사용한다.
처리유형 지연 처리 단위 처리 비용이 과다하게 발생하는 경우이다.
배치 처리 대량의 데이터를 한 번에 처리해야 하는 경우이다.
실시간 처리 요청을 즉시 처리해야 하는 경우이다.
인터페이스 데이터 명세화

인터페이스 요구사항 분석 과정에서 식별한 연계 정보에 해당하는 개체 정의서, 테이블 정의서, 코드 정의서 등을 분석하여 그에 요구되는 데이터 명세를 작성한다.
개체 정의서
데이터베이스 개념 모델링 단계에서 도출한 개체 타입, 속성, 식별자 등 개체에 관한 정보를 명세화한 자료이다.
테이블 정의서
데이터베이스 논리/물리 모델링 단계에서 작성하는 설계 산출물이다.
Table 의 속성명, 자료형, 길이, Key, Default 값, Index, 업무 규칙 등을 명세화한 자료이다.
코드 정의서
Code 는 전체 데이터베이스에 유일하게 정의되며 Code 의 명명 규칙 확정 및 그에 따른 어떤 코드를 사용할지 정한다.
송/수신 데이터 명세

송/수신 시스템의 테이블 정의서, 파일 레이아웃에서 연계하고자 하는 테이블이나 파일 단위로 명세를 작성한다.
송/수신데이터 항복의 데이터 타입, 길이, 필수 입력 여부, 식별자 여부를 정의한다.
코드성 데이터 항목은 공통 코드 여부인지 확인하고 코드값 범위를 정의한다.
법률적 근거 및 사내 보안 규정을 찾고하여 암호화 대상 칼럼을 선정하고, 해당 칼럼이 송/수신 데이터에 포함되었으면 암호화 적용 여부를 정의한다.

인터페이스 오류
오류 식별 및 처리 방안 명세화

내/외부 인터페이스 목록에 존재하는 각 인터페이스에 대해 발생 가능한 오류를 식별하고 오류 처리 방안을 명세화하는 것을 의미한다.
시스템 및 전송 오류 시 연계 프로그램 등에서 정의한 예외 상황과 대/내외 시스템 연계 시 발생할 수 있는 다양한 오류 상황을 식별 구분한다.
오류 발생 영역

송신 시스템(연계 프로그램과 중계 시스템(연계 서비스) 사이, 중계 시스템(연계 서비스)과 수신 시스템(연계 프로그램) 사이이다.)
인터페이스 오류 처리 유형

연계 시스템(서버)의 장애, 송신 시스템의 연계 프로그램 오류, 수신 시스템의 연계 프로그램 오류, 연계 데이터 자체 오류 등으로 구분할 수 있다.
송신 시스템의 연계 프로그램 오류는 연계 데이터를 생성/추출하는 과정 및 코드와 데이터를 변환하는 과정에서 발생할 수 있다.
수신 시스템의 연계 프로그램 오류는 운영 DB 에 데이터를 반영하고 코드 및 데이터를 변환하는 과정에서 발생할 수 있다.
연계 서버 : 연계 서버의 실행 여부, 송/수신 전송 형식 변환 등 연계 서버의 기능과 관련된 장애와 오류이다.
연계 데이터 : 연계 데이터값이 유효하지 않아 발생하는 오류이다.
송신 시스템 연계 프로그램 : 송신 데이터 추출을 위한 데이터베이스 접근 권한 오류와 데이터 변환 처리 오류이다.
수신 시스템 연계 프로그램 : 수신 데이터를 응용 데이터베이스에 반영하는 중에 발생하는 오류와 데이터 변환 시 발생하는 오류이다.
인터페이스 설계서
인터페이스 설계서의 정의

시스템 인터페이스 현황을 확인하기 위해서 시스템이 가지고 있는 인터페이스 목록과 상세 데이터 명세를 정의한 것이다.
인터페이스 목록과 인터페이스 정의서 작성을 통하여 구현된다.
내/외부의 모듈 간에 공통으로 제공되는 기능과 각 데이터의 인터페이스 확인에 활용된다.
송/수신 방법 및 송/수신 데이터 명세화 과정에서 작성된 산출물을 기반으로 작성한다.
초안 작성 후 인터페이스 시스템 정의서 내용과 비교하여 보완 및 수정을 진행한다.
인터페이스 목록 작성

인터페이스 목록은 연계 업무와 연계에 참여하는 송/수신 시스템의 정보나 연계 방식 그리고 통신 유형 등에 관한 정보를 포함해야 한다.
인터페이스 정의서 작성

데이터 송신 시스템, 데이터 수신 시스템 간의 데이터 저장소와 속성 등의 상세 내역을 포함한다.
인터페이스별로 시스템 간 연계를 유지하는데 필요한 데이터 항목 및 구현 요건 등을 기술하는 것이다.
19. 미들웨어 솔루션
미들웨어(Middle Ware)
미들웨서 솔루션의 정의

클라이언트와 서버 간의 통신을 담당하는 시스템 소프트웨어이다.
이기종 하드웨어, 소프트웨어, 네트워크, 프로토콜, PC 환경, 운영체제 환경 등에서 시스템 간의 표준화된 연결을 도와주는 소프트웨어이다.
표준화된 인터페이스를 통하여 시스템 간의 데이터 교환에 있어 일관성을 제공한다.
운영체제와 애플리케이션 사이에서 중간 매개 역할을 하는 다목적 소프트웨어이다.
애플리케이션에 운영체제가 제공하는 서비스를 추가 및 확장하여 제공하는 컴퓨터 소프트웨어이다.
표준화된 인터페이스를 제공하여 다양한 환경을 지원하여 체계가 다른 업무와 상호 연동이 가능하다.
분산된 업무를 동시에 처리 가능하여 자료의 일관성이 유지되어 부하의 분산이 가능하다.
미들웨어 솔루션의 유형

구분 특징
데이터베이스 - DB 제작사에서 제공하는 클라이언트와 데잍어베이스를 연결하기 위한 미들웨어이며, DB 사용 시스템 구축은 보통 2-Tier 아키텍처이다.
- 종류 : Oracle 의 Glue, Boland 의 IDAPI, MS 의 ODBC 등
TP-Monitor
(Transaction Processing Monitor) - 비즈니스의 요구사항을 해결하기 위하여 여러 소프트웨어 상호 간 혼합된 환경의 온라인 업무에서 세션, 시스템, 데이터베이스 사이의 트랜잭션을 감시하는 미들웨어이다.
- 분산 환경에서 분산 트랜잭션을 처리하며, 사용자 수가 증가해도 빠른 응답 속도를 보장해야 할 경우 사용한다.
- 통신 미들웨어 기능 외에 트랜잭션 협력 서비스, 안정적인 메시지 큐잉 시스템, 일의 흐름 관리와 개발의 통합적인 서비스들을 제공한다.
- 종류 : Oracle 의 tuxedo, Tmax 의 Tmax

ORB
(Object Request Broker) - 객체지향 미들웨어로 코바(CORBA) 표준 스펙을 구현한 미들웨어이다.
- 로컬 및 원격지에 있는 객체들 사이에 통신을 담당하는 핵심 기술이다.
- 인터페이스는 인터페이스 정의 언어인 IDL 을 사용한다.
- 하나의 객체와 다른 객체 사이의 인터페이스를 정의하게 된다.
- 최근에는 TP-Monitor 의 장점인 트랜잭션 처리와 모니터링 등을 추가로 구현 가능하다.
- 종류 : Micro Focus 의 Orbix, OMG 의 CORBA
RPC
(Remote Procedure Call) - 분산 처리 시스템을 구현하기 위해 응용 프로그램의 프로시저를 사용하여 원격 프로시저를 로컬 프로시저처럼 호출하는 방식이다.
- 종류 : OSF 의 ONC/RPC, 이큐브 시스템의 Entera
MOM
(Message Oriented Middleware) - 메시지를 기반으로 하는 비동기식 메시지 전달 보장 방식 미들웨어로 이기종의 분산 데이터베이스 시스템에서 데이터 동기화에 주로 사용한다.
- 종류 : Oracle 의 Message Q, JCP 의 JMS, MS 의 MSMQ
WAS
(Web Application Server) - 일반 웹 서버와 구별되며, 주로 DB 서버와 같이 동적 서버 콘텐츠를 수행하는데 사용한다.
- 동적인 웹 사이트, 웹 애플리케이션, 웹 서비스의 개발을 지원하기 위하여 설계된 미들웨어 소프트웨어이다.
- 서버 단에서 애플리케이션을 동작할 수 있도록 지원한다.
- 데이터 접근, 세션 관리, 트랜잭션 관리 등을 위한 라이브러리를 제공한다.
- HTTP 를 통한 사용자 컴퓨터나 장치에 Application 을 수행해주는 미들웨어이다.
- 선정 시 고려사항 : 가용성, 성능, 기술 지원, 구축 비용
- 종류 : RedHat 의 JBoss, Tmax 의 JEUS, Oracle 의 Weblogic, IBM 의 Websphere, GlasFish, Jetty, Resin, Tomcat
객체 트랜잭션 모니터
(OTM) - 전통적인 TP-MONITOR 의 기능과 ORBs 에 의해 제공되는 객체 기반 프로그램 인터페이스를 제공한다.
- 유연성 있는 통합적인 시스템 환경을 제공하는 새로운 형태의 미들웨어이다.
미들웨서 솔루션 분류

DB 미들웨어
(애플리케이션-TO-데이터 방식) 통신 미들웨어
(애플리케이션-TO-애플리케이션 방식)
ODBC

(Open Database Application Connectivity)
IDAP
(Intergranted Database Application Interface)
DRDA
(Distributed Relational Data Access)
OLEDB RPC
(Remote Procedure Call)
DCE
(Distributed Computing Environment)
MOM
(Message Orented Middleware)
ORB
(Object Request Broker)
OTM
(Object Transaction Monitor)