**<1과목: 소프트웨어 설계>**

UML 다이어그램 중 순차 다이어그램에 대한 설명

객체 간의 동적 상호작용을 시간 개념을 중심으로 모델링

일반적으로 다이어그램의 수직방향이 시간의 흐름을 나타냄

회귀메시지, 제어블록 등으로 구성

시간의 흐름에 따라 객체들이 주고받는 메시지의 전달 과정을 강조

교류 다이어그램의 한 종류로 볼 수 있다

틀린 것:

주로 시스템의 정적 측면을 모델링하기 위해 사용

동적 다이어그램보다는 정적 다이어그램에 가깝다

메시지 지향 미들웨어(message-oriented middleware, MOM)에 대한 설명

독립적인 애플리케이션을 하나의 통합된 시스템으로 묶기 위한 역할

송신측과 수신측의 연결 시 메시지 큐를 활용하는 방법이 있다

상이한 애플리케이션 간 통신을 비동기 방식으로 지원

틀린 것: 느리고 안정적인 응답보다는 즉각적인 응답이 필요한 온라인 업무에 적합

익스트림 프로그래밍에 대한 설명

4대 원칙: 의사소통, 단순성, 피드백, 용기

5대 원칙: 의사소통, 단순성, 피드백, 용기, 존중

Pair Programming(PP): 개발자 공동 2명 작업

Collective Ownership(CO): 공동소유, 누구든지 수정 가능

Continuous Integration(CI): 지속적 통합, 컴포넌트 또는 모듈 단위로 나누어 개발된 코드들은 하나의 작업이 끝날 때 마다 지속적으로 통합되고 동시에 테스트함

계발계획수립(PG), 짧은 배포 주기(SR), 문장형 시스템 아키텍쳐(META), 단순설계(SD), 테스트, 중복제거, Refactoring, 고객상주, 코드표준의사소통

소규모 개발조직이 불확실하고 변경이 많은 요구를 접하였을 때 적절

익스트림 프로그래밍을 구동시키는 원리는 상식적인 원리와 경험을 최대로 끌어올리는 것

구체적인 실천방법을 정의, 개발 문서보다 소스코드에 중점

사용자의 요구사항은 언제든지 변할 수 있다

고객과 직접 대면하며 요구사항을 이야기하기 위해 사용자 스토리(User Story)를 활용할 수 있다

기존의 방법론에 비해 실용성(Pragmatism)을 강조한 것이라고 볼 수 있다

틀린 것:

대표적인 구조적 방법론 중 하나

빠른 개발을 위해 테스트를 수행하지 않는다

유스케이스의 구성요소 간의 관계

연관: Use Case와 Actor와의 관계

확장: 기본 Use Case 수행 시 특별한 조건을 만족할 때 수행할 Use Case

일반화: 하위 Use Case/Action이 상위 Use Case/Actor에게 기능/역할을 상속받음

포함: 시스템의 기능이 별도의 기능을 포함

그룹화: 여러 개의 Use Case를 단순화하는 방법

요구사항 분석에서 비기능적 요구에 대한 설명

‘차량 대여 시스템이 제공하는 모든 화면이 3초 이내에 사용자에게 보여야 한다’는 비기능적

틀린 것:

시스템의 처리량(Throughput), 반응시간 등의 성능 요구나 품질 요구는 비기능적 x

시스템 구축과 관련된 안전, 보안에 대한 요구사항들은 비기능적 x

‘금융 시스템은 조회, 인출, 입금, 송금의 기능이 있어야 한다’는 비기능적 요구

정보공학 방법론에서 데이터베이스 설계의 표현으로 사용하는 모델링 언어

Entity-Relationship Diagram

틀린 것:

Package Diagram

State Transition Diagram

Deployment Diagram

미들웨어에 대한 설명

운영체제와 해당 운영체제에서 실행되는 응용 프로그램 사이에서 운영체제가 제공하는 서비스 외에 추가 서비스를 제공하는 소프트웨어, 미들웨어는 표준화된 인터페이스를 제공함으로써 시스템 간의 데이터 교환에 일관성 보장

분산 컴퓨팅 환경에서 서로 다른 기종 간의 하드웨어나 프로토콜, 통신환경 등을 연결하여 응용프로그램과 운영환경 간에 원만한 통신이 이루어질 수 있게 서비스를 제공하는 소프트웨어

여러 운영체제에서 응용 프로그램들 사이에 위치한 소프트웨어

소프트웨어 컴포넌트를 연결하기 위한 준비된 인프라 구조를 제공

여러 컴포넌트를 1대 1, 1대 다, 다대 다 등 여러가지 형태로 연결이 가능

틀린 것: 미들웨어의 서비스 이용을 위해 사용자가 정보교환방법 등의 내부 동작을 쉽게 확인할 수 있어야 한다

UI의 설계지침

이해하기 편하고 쉽게 사용할 수 있는 환경을 제공해야 한다

주요 기능을 메인화면에 노출하여 조작이 쉽도록 하여야 한다

사용자의 직무, 연령, 성별 등 다양한 계층을 수용하여야 한다

틀린 것: 치명적인 오류에 대한 부정적인 사항은 사용자가 인지할 수 없도록 한다

객체지향 개념에서 다형성(Polymorphism)과 관련된 설명

다형성은 현재코드를 변경하지 않고 새로운 클래스를 쉽게 추가할 수 있다

다형성이란 여러가지 형태를 가지고 있다는 의미로, 여러 형태를 받아들일 수 있는 특징

메소드 오버라이딩은 상위클래스에서 정의한 일반 메소드의 구현을 하위 클래스에서 무시하고 재정의 할 수 있다

틀린 것: 메소드 오버로딩의 경우 매개변수타입은 동일, 메소드명을 다르게 함으로써 구현, 구분할 수 있다

소프트웨어 개발영역을 결정하는 요소

인터페이스:

소프트웨어에 의해 간접적으로 제어되는 장치와 소프트웨어를 실행하는 하드웨어

기존의 소프트웨어와 새로운 소프트웨어를 연결하는 소프트웨어

순서적 연산에 의해 소프트웨어를 실행하는 절차

객체에 대한 설명

객체는 상태, 동작, 고유 식별자를 가진 모든것이라 할 수 있다

객체는 필요한 자료 구조와 이에 수행되는 함수들을 가진 하나의 독립된 존재

객체의 상태는 속성값에 의해 정의

틀린 것: 객체는 공통속성을 공유하는 클래스들의 집합

속성과 관련된 연산(Operation)을 클래스 안에 묶어서 하나로 취급하는 것을 의미하는 객체지향 개념

Encapsulation

틀린 것:

Inheritance: 상속

Class: 클래스

Association: 연관

소프트웨어 개발 방법론 중 애자일(Agile) 방법론

익스트림 프로그래밍(eXtreme Programming, XP), 스크럼(Scrum), 기능 주도 개발(Feature-Driven Development, FDD), 적응형 소프트웨어 개발(Adaptive Software Development, ASD), 익스트림 모델링, 크리스털 패밀리

프로세스와 도구 중심이 아닌 개개인과의 상호소통을 통해 의견을 수렴

협상과 계약보다는 고객과의 협력을 중시

문서 중심이 아닌 실행가능한 소프트웨어를 중시

소프트웨어 개발에 참여하는 구성원들 간의 의사소통 중시

환경 변화에 대한 즉시 대응

프로젝트 상황에 따른 주기적 조정

빠른 릴리즈를 통해 문제점을 빠르게 파악할 수 있다

진화하는 요구사항을 수용하는데 적합하다

틀린 것:

각 단계의 결과가 완전히 확인된 후 다음 단계 진행

변화에 대한 대응보다는 자세한 계획을 중심으로 소프트웨어를 개발

정확한 결과 도출을 통해 계획 수립과 문서화에 중점을 둔다

명백한 역할을 가지고 독립적으로 존재할 수 있는 시스템의 부분, 넓은 의미에서는 재사용되는 모든 단위, 인터페이스를 통해서만 접근 가능한 것

Component

틀린 것:

Model

Sheet

Cell

GoF(Gang of Four) 디자인 패턴을 목적에 따라 생성, 구조, 행동 패턴의 세 그룹으로 분류

생성: 객체의 생성, 인스턴스 과정을 추상화

Abstract Factory: 구체적인 클래스를 지정하지 않고 인터페이스를 통해 서로 연관되는 객체들을 그룹으로 표현

Builder: 복합 객체의 생성과 표현을 분리하여 동일한 생성 절차에서도 다른 표현 결과를 만들어낼 수 있음

Factory Method: 객체 생성을 서브 클래스로 위임하여 캡슐화, 상위 클래스에서 객체를 생성하는 인터페이스를 정의, 하위 클래스에서 인스턴스를 생성하도록 하는 방식

Prototype: 원본 객체를 복사함으로써 객체를 생성, Prototype을 먼저 생성하고 인스턴스를 복제하여 사용하는 구조

Singleton: 어떤 클래스의 인스턴스는 하나임을 보장하고 어디서든 참조할 수 있도록, 한 클래스에 한 객체만 존재

구조: 클래스나 객체들을 조합해 더 큰 구조로 만들 수 있게, 클래스 패턴; 상속을 통해 클래스나 인터페이스 합성, 객체 패턴; 객체를 합성하는 방법

Adapter: 클래스의 인터페이스를 다른 인터페이스로 변환하여 다른 클래스가 이용할 수 있도록 함, 기존에 구현되어 있는 클래스에 기능 발생 시 기존 클래스를 재사용할 수 있도록 중간에서 맞춰주는 역할, 인터페이스가 호환되지 않는 클래스들을 함께 이용할 수 있도록 타 클래스의 인터페이스를 기존 인터페이스에 덧씌움

Bridge: 구현부에서 추상층을 분리하여 각자 독립적으로 확장할 수 있게 함

Composite: 객체들의 관계를 트리 구조로 구성하여 복합 객체와 단일 객체를 구분없이 다룸

Decorator: 주어진 상황 및 용도에 따라 어떤 객체에 다른 객체를 덧붙이는 방식, 객체를 묶어 하나의 객체로 이용

Facade: 서브시스템에 있는 인터페이스 집합에 대해 하나의 통합된 인터페이스(Wrapper) 제공

Flyweight: 크기가 작은 여러 개의 객체를 매번 생성하지 않고 가능한 한 공유할 수 있도록 하여 메모리 절약

Proxy: 접근이 어려운 객체로의 접근을 제어하기 위해 객체의 Surrogate나 Placeholder를 제공

행동: 클래스나 객체들이 서로 상호작용하는 방법이나 어떤 태스크, 어떤 알고리즘을 어떤 객체에 할당하는 것이 좋을지, 교류 방법, 책임을 분산하는 방법에 대해 정의, 클래스 패턴; 상속을 통해 알고리즘과 제어 흐름을 기술, 객체 패턴; 하나의 작업을 수행하기 위해 객체 집합이 어떻게 협력하는지 기술

Chain of Responsibility: 요청받는 객체를 연쇄적으로 묶어 요청처리 객체를 만날 때까지 Chain을 따라 요청을 전달

Command: 요청을 객체의 형태로 캡슐화하여 재사용하거나 취소할 수 있도록 저장

Interpreter: 특정 언어의 문법 표현을 정의

Iterator: 내부를 노출하지 않고 접근이 잦은 객체의 원소를 순차적으로 접근할 수 있는 동일한 인터페이스 제공

Mediator: 한 집합에 속해있는 객체들의 상호작용을 캡슐화하여 새로운 객체로 정의, 객체간의 통제와 지시의 역할을 하는 중재자를 두어 객체지향의 목표를 달성하게 해준다

Memento: 객체가 특정 상태로 다시 되돌아올 수 있도록 내부 상태를 실체화

Observer: 객체 상태가 변할 때 관련 객체들이 그 변화를 통지받고 자동으로 갱신될 수 있게 함

State: 객체의 상태에 따라 동일한 동작을 다르게 처리해야할 때 사용

Strategy: 동일 계열의 알고리즘군을 정의하고 캡슐화하여 상호교환이 가능하도록 함

Template Method: 상위 클래스는 알고리즘의 골격만을 작성하고 구체적인 처리는 서브클래스로 위임

Visitor: 객체의 원소에 대해 수행할 연산을 분리하여 별도의 클래스로 구성

UI와 관련된 기본 개념 중 하나로, 시스템의 상태와 사용자의 지시에 대한 효과를 보여주어 사용자가 명령에 대한 진행 상황과 표시된 내용을 해석할 수 있도록 도와주는 것

Feedback

틀린 것:

Posture

Hash

Module

UI의 종류로 멀티터치, 동작 인식 등 사용자의 자연스러운 움직임을 인식하여 서로 주고받는 정보를 제공하는 것

NUI(Natural User Interface): 사용자의 말이나 행동으로 기기 조작

틀린 것:

GUI(Graphical User Interface): 사용자 편리를 위해 입출력 등의 기능을 알기 쉬운 아이콘, 그래픽으로 나타낸 것

OUI(Organic User Interface): 모든 사물과 사용자 간의 상호작용을 위한 인터페이스

CLI(Command Line Interface): DOS, Unix 등의 운영체제에서 사용, 정해진 명령문자열을 입력, 시스템 조작

VUI(Voice User Interface): 사람의 음성으로 기기를 조작하는 인터페이스

소프트웨어 모델링과 관련된 설명

구조적 방법론에서는 DFD(Data Flow Diagram), DD(Data Dictionary) 등을 사용하여 요구사항의 결과를 표현

객체지향 방법론에서는 UML표기법을 사용

소프트웨어 모델을 사용할 경우 개발될 소프트웨어에 대한 이해도 및 이해 당사자 간의 의사소통향상에 도움

개발될 시스템에 대하여 여러 분야의 엔지니어들이 공통된 개념을 공유하는데 도움을 준다

절차적인 프로그램을 위한 자료흐름도는 프로세스 위주의 모델링 방법이다

틀린 것:

모델링 작업의 결과물은 다른 모델링 작업에 영향을 줄 수 없다

유지보수 단계에서만 모델링 기법을 활용한다(모델링은 분석 및 설계 단계에서 제작, 개발 전과정에서 지속적 사용)

유스케이스 다이어그램에 관련된 내용

유스케이스는 시스템이 액터에게 제공해야 하는 기능, 시스템의 요구사항이자 기능을 의미

액터는 시스템 외부에서 시스템과 상호작용하는 사람 혹은 시스템을 말함(주; 사용자 액터, 부; 시스템 액터)

유스케이스는 사용자 측면에서의 요구사항으로, 사용자가 원하는 목표를 달성하기 위해 수행할 내용을 기술한다

시스템 액터는 다른 프로젝트에서 이미 개발, 사용되고 있으며, 본 시스템과 데이터를 주고받는 서로 연동될 시스템, 사용자 액터가 사용한 유스케이스를 처리해주는 외부 시스템, 시스템의 기능 수행을 위해서 연동이 되는 또 다른 시스템 액터를 의미

사용자 액터는 기능을 요구하는 대상이나 시스템의 수행결과를 통보받는 사용자 혹은 기능을 사용하게 될 대상으로 시스템이 제공해야 하는 기능인 유스케이스의 권한을 가지는 대상, 역할

액터가 인식할 수 없는 시스템 내부의 기능을 하나의 유스케이스로 파악해서는 안된다

틀린 것: 시스템과 상호작용하는 외부시스템은 액터로 파악해서는 안된다

소프트웨어 아키텍쳐 모델 중 MVC(Model-View-Controller)와 관련된 설명

UI를 담당하는 계층의 응집도를 높일 수 있고, 여러 개의 다른 UI를 만들어 그 사이의 결합도를 낮출 수 있다

View는 Model에 있는 데이터를 사용자 인터페이스에 보이는 역할을 담당

Controller는 Model에 명령을 보냄으로써 Model의 상태를 변경할 수 있다

틀린 것: Model은 View와 Controller 사이에서 전달자 역할, View마다 Model 서브시스템이 각각 하나씩 연결

UI 설계 시 오류 메시지나 경고에 관한 지침

메시지는 이해하기 쉬워야 한다

오류로부터 회복을 위한 구체적인 설명이 제공되어야 한다

오류로 인해 발생될 수 있는 부정적인 내용을 적극적으로 사용자들에게 알려야 한다

틀린 것: 소리나 색의 사용을 줄이고 텍스트로만 전달하도록 한다

소프트웨어 설계에서 요구사항 분석에 대한 설명

소프트웨어가 무엇을 해야하는가를 추적하여 요구사항 명세를 작성하는 작업이다

사용자의 요구를 추출하여 목표를 정하고 어떤 방식으로 해결할 것인지 결정하는 단계

소프트웨어 개발의 출발점이면서 실질적인 첫 번째 단계로 사용자의 요구에 대해 이해하는 단계

요구 추출(Requirement Elicitation)은 프로젝트 계획 단계에 정의한 문제의 범위 내 있는 사용자의 요구를 찾는 단계

도메인 분석(Domain Analysis)은 요구에 대한 정보를 수집하고 배경을 분석하여 이를 토대로 모델링을 하게 된다

틀린 것:

소프트웨어 시스템이 사용되는 동안 발견되는 오류를 정리하는 단계

기능적(Functional) 요구에서 시스템 구축에 대한 성능, 보안, 품질, 안정 등에 대한 요구사항을 도출한다

설계기법 중 하향식 설계 방법과 상향식 설계 방법에 대한 비교 설명

하향식 설계에서는 통합 검사 시 인터페이스가 이미 정의되어 있어 통합이 간단하다

하향식 설계에서 레벨이 낮은 데이터 구조의 세부 사항은 설계 초기 단계에서 필요하다

상향식 설계는 최하위 수준에서 각각의 모듈들을 설계하고 이러한 모듈이 완성되면 이들을 결합하여 검사한다

틀린 것: 상향식 설계에서는 인터페이스가 이미 성립되어 있지 않더라도 기능 추가가 쉽다

자료흐름도(DFD, Data Flow Diagram)

Process: 원

Data Flow: 화살표

Data Store: 평행선

Terminator: 사각형

자료 흐름 그래프 또는 버블 차트라고도 한다

구조적 분석 기법에 이용

시간 흐름을 명확하게 표현할 수 없다

자료에 대한 흐름을 표현, 구조적 분석 기법에 이용

소프트웨어 개발에 이용되는 모델에 대한 설명

모델은 개발 대상을 추상화하고 기호나 그림 등으로 시각적으로 표현

모델을 통해 소프트웨어에 대한 이해도를 향상시킬 수 있다

모델을 통해 이해 당사자 간의 의사소통이 향상된다

틀린 것: 모델을 통해 향후 개발될 시스템의 유추는 불가능하다

UML: 객체 지향 시스템을 개발할 때 산출물을 명세화, 시각화, 문서화하는데 사용된다. 개발하는 시스템을 이해하기 쉬운 형태로 표현하여 분석가, 의뢰인, 설계자가 효율적인 의사소통을 할 수 있게 해준다. 개발 방법론이나 개발 프로세스가 아니라 표준화된 모델링 언어이다

구성요소: 사물(Things), 관계(Relationship), 다이어그램(Diagram)

기능적 모델은 사용자 측면에서 본 시스템의 기능, Use Case Diagram을 사용

정적 모델은 객체, 속성, 연관관계, 오퍼레이션의 시스템의 구조를 나타내며, Class Diagram 사용

동적 모델은 시스템의 내부 동작을 말하며 Sequence Diagram, State Diagram, Activity Diagram을 사용

정적 다이어그램(구조적 다이어그램): 클래스, 객체, 컴포넌트, 배치, 복합체, 패키지 등

동적 다이어그램(행위적 다이어그램): 유스케이스, 순차, 커뮤니케이션, 상태, 활동, 타이밍, 상호작용 등

Class 다이어그램: 시스템의 구조적인 모습을 그리는 다이어그램, 시스템 내 클래스의 정적 구조 표현, 클래스-클래스, 클래스의 속성 사이의 관계

Component 다이어그램: 소프트웨어 구조가 그리는 다이어그램

Deployment 다이어그램: 기업 환경의 구성과 컴포넌트들 간의 관계를 그린 다이어그램

Activity 다이어그램: 업무의 흐름을 모델링하거나 객체의 생명 주기를 표현하는 다이어그램, 시스템이 어떤 기능을 수행하는지 객체의 처리 로직이나 조건에 따른 처리의 흐름을 순서에 따라 표현

Sequence 다이어그램: 객체 간의 메시지 전달을 시간적 흐름에서 분석하는 다이어그램

Collaboration 다이어그램: 객체와 객체가 주고받는 메시지 중심의 작성 동적 다이어그램

State 다이어그램: 객체가 가진 상태와 상태 변환을 나타냄, 하나의 객체가 자신이 속한 클래스의 상태 변화 혹은 다른 객체와의 상호 작용에 따라 상태가 어떻게 변화하는지를 표현

Mockup

디자인, 사용방법설명, 평가 등을 위해 실제 화면과 유사하게 만든 정적인 형태의 모형

시각적으로만 구성 요소를 배치하는 것으로 일반적으로 실제로 구현되지는 않음

애자일 기법 중 스크럼(Scrum)과 관련된 용어

스크럼 마스터는 스크럼 프로세스를 따르고, 팀이 스크럼을 효과적으로 활용할 수 있도록 보장하는 역할

제품 백로그는 스크럼 팀이 해결해야 하는 목록으로 소프트웨어 요구사항, 아키텍쳐 정의 등이 포함될 수 있다

속도(Velocity)는 한 번의 스프린트에서 한 팀이 어느 정도의 제품 백로그를 감당할 수 있는지에 대한 추정치

틀린 것: 스프린트(Sprint)는 하나의 완성된 최종 결과물을 만들기 위한 주기로 3달 이상의 장기간으로 결정(세분화)

UML 다이어그램 중 정적 다이어그램

컴포넌트, 클래스, 객체, 복합구조, 배치, 패키지

틀린 것: 순차(시퀀스) 다이어그램(동적 다이어그램), 유스케이스, 상태, 활동, 통신, 상호작용, 타이밍

LOC기법에 의하여 예측된 총 라인 수가 36000, 프로그래머 6, 평균 생산성이 월간 300, 개발 소요기간

36000 / (6 \* 300) = 20 개월

클래스 설계원칙

개방-폐쇄의 원칙: 클래스는 확장에 대해 열려 있어야 하며 변경에 대해 닫혀 있어야 한다

단일 책임 원칙: 하나의 객체는 하나의 동작만의 책임을 가짐

리스코프 교체의 원칙: 특정 메소드가 상위 타입을 인자로 사용할 때 그 타입의 하위 타입도 문제없이 작동해야 함

의존관계 역전의 원칙: 상위 계층이 하위 계층에 의존하는 전통적인 의존관계를 반전(역전)시킴으로써 상위 계층이 하위 계층의 구현으로부터 독립되게 할 수 있음

아키텍쳐 설계과정

설계 목표 설정 – 시스템 타입 결정 – 스타일 적용 및 커스터마이즈 – 서브시스템의 기능, 인터페이스 동작 완성 – 아키텍쳐 설계 검토

UI를 설계할 경우 고려해야할 가이드라인

효율성을 높이게 설계해야 한다

발생하는 오류를 쉽게 수정할 수 있어야 한다

사용자에게 피드백을 제공해야 한다

틀린 것: 심미성을 사용성보다 우선하여 설계해야 한다

디자인 패턴: 소프트웨어 설계에서 자주 발생하는 문제에 대한 일반적이고 반복적인 해결방법, 각 모듈의 세분화된 역할이나 모듈들 간의 인터페이스와 같은 코드를 작성하는 수준의 세부적인 구현 방안을 설계할 때 참조할 수 있는 전형적인 해결 방식 또는 예제

장점

소프트웨어 코드의 품질을 향상시킬 수 있다

개발자들 사이의 의사소통을 원활하게 할 수 있다(공통 언어 사용)

소프트웨어의 품질과 생산성을 향상시킬 수 있다

향후 변화에 대한 대비 가능, 유지 보수 용이

소프트웨어 구조 파악이 용이

객체지향 설계 및 구현의 생산성을 높이는데 적합

재사용을 위한 개발 시간이 단축

틀린 것:

개발 프로세스를 무시할 수 있다

절차형 언어와 함께 이용될 때 효율이 극대화(객체 지향 설계/ 구현에 많이 사용)

Rumbaugh Method: 객체지향 분석기법의 하나, 소프트웨어 구성요소를 그래픽 표기법으로 이용한 모델링, 객체 모형, 동적 모형, 기능 모형의 3개 모형을 생성하는 방법

객체 모형: 객체 다이어그램, 정보 모델링, 시스템에서 요구되는 객체를 찾아내어 속성과 연산 식별 및 객체들 간의 관계를 규정

동적 모형: 상태 다이어그램, 시스템의 행위를 기술

기능 모형: 자료 흐름도, 다수의 프로세스들 간의 자료 흐름을 중심으로 처리 과정 표현

Booch Method: 미시적 개발 프로세스와 거시적 개발 프로세스를 모두 사용하는 분석방법, 클래스와 객체들을 분석 및 식별하고 클래스의 속성과 연산을 정의

Jacobson Method: Use Case를 사용하는 분석, 사용자, 외부 시스템, 다른 요소들이 시스템과 상호작용하는 방법

Coad Yourdon Method: E-R 다이어그램을 사용하여 객체의 행위를 모델링하며 객체 식별, 구조 식별, 주제 정의, 속성과 인스턴스 연결 정의, 연산과 메시지 연결 정의 등의 과정으로 구성하는 기법

Wirfs-Brock Method: 분석과 설계 간의 구분이 없고, 고객 명세서를 평가해서 설계 작업까지 연속적으로 수행

FEP: 입력되는 데이터를 컴퓨터의 프로세서가 처리하기 전에 미리 처리하여 프로세서가 처리하는 시간을 줄여주는 프로그램, 하드웨어

EAI: Enterprise Application Integration, 기업용 응용 프로그램의 구조적 통합 방안

GPL: General Public License, 자유 소프트웨어 라이선스

Duplexing: 이중화, 데이터베이스의 회복 기법 중 가장 간단한 것

Class: 객체 지향 개념 중 하나 이상의 유사한 객체들을 묶어 공통된 특성을 표현한 데이터 추상화

Method: 클래스로부터 생성된 객체를 사용하는 방법, 객체가 메시지를 받아 실행해야 할 객체의 구체적인 연산

Message: 객체 간 상호 작용을 하기 위한 수단, 객체에게 어떤 행위를 하도록 지시하는 방법

Field: SQL에서 열 또는 속성이라고 불리는 것

요구사항 검증(Requirements Validation)과 관련한 설명

요구사항이 고객이 정말 원하는 시스템을 제대로 정의하고 있는지 점검하는 과정

개발 완료 이후에 문제점이 발견될 경우 막대한 재작업 비용이 들 수 있기 때문에 요구사항 검증은 매우 중요하다

시스템을 변경하여 요구사항 문제를 수정하는 비용은 설계 및 코딩 오류에 비하여 비용이 많이 소요된다. (시스템 설계, 구현 변경 및 다시 테스트를 해야 하기 때문)

요구사항이 실제 요구를 반영하는지, 문서상의 요구사항은 서로 상충되지 않는지 등을 점검한다

틀린 것: 요구사항 검증 과정을 통해 모든 요구사항 문제를 발견할 수 있다

요구사항 체크리스트:

유효성(Validity): 고객의 필요를 충족하는 기능을 제공하는지, 요구한 것이 맞는지

일관성(Consistency): 충돌하는 요구사항이 존재하는지(모순되는 제약조건 등)

완결성(Completeness): 고객이 요구한 모든 기능이 포함되었는지

현실성(Realism): 예산과 기술적으로 실행 가능한지(일정 포함)

검증 가능성(Verifiability): 만들고 난 뒤 요구사항들을 검증할 수 있는지(요구사항과 일치 여부)

연관 관계(Association Relationship): 한 클래스에서 타 클래스를 멤버 변수로 갖고 있는 경우, ‘has-a’ 관계

* Aggression: 해당 클래스 외부에서 생성된 타 클래스의 인스턴스를 멤버 변수로 포함, 실선 흰색 다이아
* Composition: 한 클래스 내부에서 타 클래스의 인스턴스를 생성하여 멤버 변수로 포함, 실선 검정 다이아

의존 관계(Dependency Relationship): UML 모델에서 한 사물의 명세가 바뀌면 다른 사물에 영향을 줌, 일반적으로 한 클래스가 다른 클래스를 오퍼레이션의 매개변수로 사용하는 경우의 관계, 실선 화살표

실체화 관계(Realization Relationship): 인터페이스와 실제 구현된 일반 클래스 간의 관계, 한 객체가 다른 객체에 의해 오퍼레이션을 수행하도록 지정, 점선 삼각 화살표

일반화 관계(Generalization Relationship): 객체지향에서 상속관계, 일반화된 사물과 좀 더 특수화된 사물 사이의 관계, ‘is-a’ 관계, 실선 삼각 화살표

소프트웨어 설계에서 사용되는 대표적인 추상화(Abstraction) 기법

문제의 전체적이고 포괄적인 개념을 설계한 후 차례로 세분화하여 구체화시켜 나가는 것

자료 추상화: 데이터의 세부적인 속성이나 용도를 정의하지 않고, 데이터 구조를 대표할 수 있는 표현으로 대체

제어 추상화: 제어의 정확한 메커니즘을 정의하지 않고 원하는 효과를 정하는데 이용하는 방법, 분기를 생각

과정 추상화: 자세한 단계는 고려하지 않고, 상위 수준에서 수행 흐름만 먼저 설계

틀린 것: 강도 추상화

객체지향 설계에서 정보 은닉(Information Hiding)과 관련한 설명

다른 객체에게 자신의 정보를 숨기고 자신의 연산만을 통해 접근을 허용하는 것

클래스 외부에서 특정 정보에 접근을 막는다는 의미

필요하지 않은 정보는 접근할 수 없도록 하여 한 모듈 또는 하부 시스템이 다른 모듈의 구현에 영향을 받지 않게 설계되는 것을 의미

모듈들 사이의 독립성을 유지시키는데 도움이 된다

설계에서 은닉되어야 할 기본 정보로는 IP주소와 같은 물리적 코드, 상세 데이터 구조 등이 있다

틀린 것: 모듈 내부 자료 구조와 접근 동작들에만 수정을 국한하기 때문에 요구사항 등 변화에 따른 수정이 불가능

장점:

기능의 교체나 변경에 대한 유연성을 제공(객체 간의 구체적인 결합도를 약화)

동일한 타입의 다른 구현 객체들의 교체로 동적 기능 변경 가능

구체적인 구현이 없는 상태(인터페이스)로도 정확한 연동 코드의 생성 가능

모듈화하여 코드의 가독성 증가

개발 기간 단축

기능적 요구사항

시스템이 수행해야 하는 행위들을 구체화한 것

시스템에서 제공해야 할 기능을 정의한 것

입력 기능, 출력 기능, 데이터베이스 기능, 통신 기능 등

비기능적 요구사항

시스템이 가져야 하는 기능 이외의 요구사항

시스템의 전체적인 품질이나 고려해야 하는 제약사항 등

사용 용이성, 효율성, 신뢰성, 이식성, 유연성, 확장성 등

성능적인 면: 응답 속도, 자원 사용량 등

보안 측면: 침입 대응, 침입 탐지, 사용자 인증, 권한 부여 등

클래스 다이어그램의 요소

Operation: 클래스의 동작, 클래스에 속하는 객체에 대하여 적용될 메서드를 정의, UML에서는 동작에 대한 인터페이스를 지칭한다고 볼 수 있다

분산 시스템을 위한 마스터-슬레이브(Master-Slave) 아키텍쳐에 대한 설명

일반적으로 실시간 시스템에서 사용

마스터 프로세스는 일반적으로 연산, 통신, 조정을 책임진다

마스터 프로세스는 슬레이브 프로세스들을 제어할 수 있다

틀린 것: 슬레이브 프로세스는 데이터 수집 기능을 수행할 수 없다

마스터: 작업을 분리, 배포, 슬레이브가 반환한 결과로부터 최종 결과값을 계산한다

슬레이브: 요청 작업 처리, 마스터의 작업 요청을 처리하고 결과를 되돌려준다

요구사항 정의 및 분석 설계의 결과물을 표현하기 위한 모델링 과정에서 사용되는 다이어그램

Data Flow Diagram

UML Diagram

E-R Diagram

틀린 것: AVL Diagram; 이진 트리의 높낮이가 불규칙해지는 것을 보완하여 일정하게 처리하기 위한 이진 트리 모형

객체지향의 주요 개념에 대한 설명

객체는 실세계에 존재하거나 생각할 수 있는 것을 말한다

클래스는 하나 이상의 유사한 객체들을 묶어 공통된 특성을 표현한 것이다

다형성은 상속받은 여러 개의 하위 객체들이 다른 형태의 특성을 갖는 객체로 이용될 수 있는 성질이다

메시지: 객체의 행위를 표현

캡슐화: 추상화된 객체의 구현을 은닉, 시스템의 한 컴포넌트의 내부 구현이 다른 컴포넌트에 의존하지 않는 것, 객체 내부의 구현을 Client에게 보이지 않게 하는 것, 인터페이스가 단순화, 재사용성 증가, 오류 파급효과 감소

틀린 것: 캡슐화는 상위클래스에서 속성이나 연산을 전달받아 새로운 형태의 클래스로 확장하여 사용하는 것(상속)

UI에 대한 설명

사용자와 시스템이 정보를 주고받는 상호작용이 잘 이루어지도록 하는 장치나 소프트웨어

배우기가 용이하고 쉽게 사용할 수 있도록 만들어져야 한다

사용자의 요구사항이 UI에 반영될 수 있도록 구성해야 한다

틀린 것: 편리한 유지보수를 위해 개발자 중심으로 설계되어야 한다

GoF(Gang of Four) 디자인 패턴과 관련한 설명

디자인 패턴을 목적(Purpose)으로 분류할 때 생성, 구조, 행위로 분류할 수 있다

행위 패턴은 클래스나 객체들이 상호작용하는 방법과 책임을 분산하는 방법을 정의

Singleton 패턴은 특정 클래스의 인스턴스가 오직 하나임을 보장하고, 이 인스턴스에 대한 접근 방법을 제공한다

틀린 것: Strategy 패턴은 대표적인 구조 패턴으로 인스턴스를 복제하여 사용하는 구조를 말한다

객체지향 분석기법과 관련한 설명

소프트웨어를 개발하기 위한 비즈니스(업무)를 객체와 속성, 클래스와 멤버, 전체와 부분 등으로 나누어서 분석

동적 모델링 기법이 사용될 수 있다

데이터와 행위를 하나로 묶어 객체를 정의내리고 추상화시키는 작업이라 할 수 있다

코드 재사용에 의한 프로그램 생산성 향상 및 요구에 따른 시스템의 쉬운 변경이 가능하다

틀린 것: 기능 중심으로 시스템을 파악하며 순차적인 처리가 중요시되는 하향식(Top-Down) 방식으로 볼 수 있다

분산 시스템에서의 미들웨어(Middleware)와 관련한 설명

분산 시스템에서 다양한 부분을 관리하고 통신하며 데이터를 교환하게 해주는 소프트웨어

위치 투명성(Location Transparency)를 제공

분산 시스템의 여러 컴포넌트가 요구하는 재사용 가능한 서비스의 구현을 제공

틀린 것: 애플리케이션과 사용자 사이에서만 분산 서비스를 제공한다(프로그램-환경 간에서도 서비스 제공)

미들웨어: 복잡한 이기종 환경에서 응용프로그램과 운영환경 간에 원만한 통신을 이룰 수 있게 해주는 소프트웨어

소프트웨어 아키텍쳐와 관련한 설명

외부에서 인식할 수 있는 특성이 담긴 소프트웨어의 골격이 되는 기본 구조

데이터 중심 아키텍쳐는 공유 데이터저장소를 통해 접근자 간의 통신이 이루어지므로 각 접근자의 수정과 확장이 용이하다

이해 관계자들의 품질 요구사항을 반영하여 품질 속성을 결정한다

종류: 레이어 패턴, 클라이언트-서버 패턴, 파이프필터 패턴, MVC 패턴, 마스터-슬레이브 패턴, 피어-투-피어 패턴, 브로커 패턴, 이벤트-버스 패턴, 블랙보드 패턴, 인터프리터 패턴

레이어 패턴: 시스템을 계층으로 구분하여 구성, OSI계층모델

클라이언트-서버 패턴: 하나의 서버 컴포넌트와 다수의 클라이언트 컴포넌트로 구성되는 패턴

파이프필터 패턴: 데이터 스트림 절차의 각 단계를 필터 컴포넌트로 캡슐화, 파이프를 통해 데이터를 전송, UNIX 쉘

MVC 패턴: 서브시스템을 3개 부분(모델, 뷰, 컨트롤러)으로 구조화하는 패턴

시스템의 구성요소

입력(Input): 처리 방법, 처리할 데이터, 조건을 시스템에 투입하는 것

처리(Process): 입력된 데이터를 처리 방법과 조건에 따라 처리하는 것

출력(Output): 처리된 결과를 시스템에서 산출하는 것

제어(Control): 자료를 입력하여 출력될 때 까지의 처리 과정이 올바르게 진행되는지 감독하는 것

피드백(Feedback): 출력된 결과가 예정된 목표를 만족시키지 못할 경우 목표 달성을 위해 반복 처리하는 것

Maintenance는 유지보수로서 시스템 구성요소에는 포함되지 않음

요구사항 개발 프로세스의 순서

도출 – 분석 – 명세 – 확인

Instance: 객체지향 기법에서 같은 클래스에 속한 각각의 객체

Message: 객체에게 어떤 행위를 하도록 지시하는 명령

Method: 객체에 소속된 함수

Module: 실행코드와 객체들(함수, 클래스, 변수)의 묶음

Class: 객체를 정의해 놓은 것, 객체의 설계도, 틀

Package: 클래스를 묶어두는 물리적인 단위, 클래스들의 집합

Object: 실제로 존재하는 것, 클래스에 정의된 내용대로 메모리에 생성된 것

요구사항 분석이 어려운 이요

개발자와 사용자 간의 지식이나 표현의 차이가 커서 상호 이해가 쉽지 않다

사용자의 요구사항이 모호하고 불명확하다

소프트웨어 개발 과정 중에 요구사항이 계속 변할 수 있다

틀린 것: 사용자의 요구는 예외가 거의 없어 열거와 구조화가 어렵지 않다(예외가 많아 열거와 구조화가 어렵다)

소프트웨어 아키텍쳐 설계에서 시스템 품질 속성

가용성(Availability), 변경용이성(Modifiability), 사용성(Usability), 성능, 보안성, 시험 용이성

연계 시스템 구성

송신 시스템, 수신 시스템, 중계 서버

송신 시스템: 연계할 데이터를 DB와 애플리케이션으로부터 연계 테이블 또는 파일 형태로 생성하여 송신

수신 시스템: 수신한 연계 테이블, 파일 데이터를 수신 시스템에서 관리하는 데이터 형식에 맞게 변환하여 DB에 저장하거나 애플리케이션에서 활용할 수 있도록 제공

중계 서버: 송/수신 시스템 사이에서 데이터를 송수신하고 연계 데이터의 송수신 현황을 모니터링함, 연계 데이터의 보안 강화 및 다중 플랫폼 지원 등이 가능

CASE(Computer-Aided Software Engineering)의 원천 기술

구조적 기법, 프로토타이핑 기술, 자동 프로그래밍 기술, 정보 저장소 기술, 분산 처리 기술

클라이언트 서버 구조: 컴포넌트가 다른 컴포넌트에게 서비스를 요청, 데이터가 여러 컴포넌트를 거치며 처리

계층 구조: 모듈들로 응집된 계층 단위로 SW를 구성, 계층 간에 사용 가능의 관계로 표현

MVC 구조: Model-View-Controller, 기능을 분리한 아키텍쳐

CASE: 시스템 개발 과정의 일부 또는 전체를 자동화 시킨 것, 소프트웨어 생명주기의 전체 단계를 연결해주고 자동화해주는 통합된 도구를 제공, 소프트웨어, 하드웨어, 데이터베이스, 테스트 등을 통합, 소프트웨어 개발 환경을 제공

소프트웨어 모듈의 재사용성 향상

자동화된 기법을 통해 소프트웨어 품질 향상

소프트웨어 유지보수를 간편하게 수행할 수 있다

일관성 분석을 통해 요구사항 변경사항의 추적 및 분석, 관리 표준 준수여부를 확인

상위 CASE: 요구 분석과 설계 단계를 지원

모델들 사이의 모순 검사 기능

모델의 오류 검증 기증

자료 흐름도 작성 기능

하위 CASE: 코드를 작성하고 테스트하며 문서화하는 과정 지원

원시 코드 생성 기능

통합 CASE: 소프트웨어 개발 주기 전체 과정을 지원

기능:

그래픽 기능: 차트와 다이어그램을 자동으로 생성

화면과 리포트 생성기, 데이터 사전, 분석과 검사 도구, 코드 생성기, 문서 생성기 등

SW 생명주기 전 단계의 연결, 다양한 소프트웨어 개발 모형 지원

요구사항 관리 도구의 필요성

요구사항 변경으로 인한 비용 편익 분석

요구사항 변경의 추적

요구사항 변경에 따른 영향 평가

틀린 것: 기존 시스템과 신규 시스템의 성능 비교

사용자 인터페이스(User Interface, UI)의 특징

구현하고자 하는 결과의 오류를 최소화

막연한 작업 기능에 대해 구체적인 방법을 제시

사용자 중심의 상호 작용이 되도록 함

틀린 것: 사용자의 편의성을 높임으로써 작업 시간을 증가시킨다(단축시킨다)

리눅스 명령어

ls: List, 디렉토리 목록 출력

cat: 파일 출력

pwd: Print Working Directory, 현재 디렉토리 출력

uname: 시스템 정보를 출력

시스템 연계 기술

DB 링크 기술: 데이터베이스에서 제공하는 DB링크 객체를 이용, 수신측에서 DB링크를 생성하고 송신측에서 해당 DB링크를 직접 참조하는 방식

DB 커넥션: 수신측의 WAS에서 송신측 데이터베이스로 연결하는 DB Connection Pool을 생성

API/OpenAPI: 송신측의 데이터베이스에서 데이터를 가져와 제공하는 응용 프로그래밍 인터페이스 프로그램

JDBC: 수신측의 프로그램에서 JDBC드라이버를 이용하여 송신 시스템 데이터베이스와 연결,

DBMS 유형, DBMS 서버 IP와 Port, DB Instance 정보가 필요

하이퍼링크: 웹 응용에서 하이퍼링크(Hyper Link)를 이용

소켓: 서버는 통신을 위한 Socket을 생성, Port를 할당, 클라이언트의 통신 요청 시 클라이언트와 연결하고 통신하는 내외부 송수신 연계기술, 네트워크 기술

RPC(Remote Procedure Call): 응용프로그램의 프로시저를 사용, 원격 프로시저를 로컬처럼 호출하는 방식의 미들웨어

WAS: 애플리케이션 수행 미들웨어

MOM: 메시지 지향 미들웨어

ORB: 네트워크 호출 미들웨어

현행 시스템 분석

플랫폼 기능 분석, 플랫폼 성능 특성 분석, 운영체제 분석, 네트워크 분석, DBMS분석, 비즈니스 융합 분석

요구사항 모델링

애자일 방법, 유스케이스 다이어그램, 시퀀스 다이어그램

틀린 것: 단계 다이어그램; 물리, 화학 등에서 사용하는 다이어그램

상향식 설계: 최하위 수준에서 각각의 모듈들을 설계하고 모듈이 완성되면 이들을 결합하여 검사

하향식 설계: 주어진 문제를 분석하여 모듈의 전체적인 구조와 데이터를 개괄적으로 설계하고 이를 기반으로 하위 레벨에서 점차 세부적인 기능을 중심으로 모듈을 설계

객체지향 설계 원칙(SOLID)

단일 책임 원칙(SRP, Single Responsibility Principle): 모든 클래스는 하나의 책임만 가지며, 클래스는 그 책임을 완전히 캡슐화 해야 함

개방 폐쇄의 원칙(OCP, Open-Closed Principle): 소프트웨어 개체(클래스, 모듈, 함수 등)는 확장에 대해 열려있어야 하고, 수정에 대해서는 닫혀있어야 한다

리스코프 교체(치환)의 원칙(LSP, Liskov Substitution Principle): 컴퓨터 프로그램에서 자료형 S가 자료형 T의 하위형이라면 필요한 프로그램의 속성(정확성, 수행하는 업무 등)의 변경 없이 자료형 T의 객체를 자료형 S의 객체로 교체할 수 있어야 한다, 상위 자료형을 하위 자료형으로 객체를 교체할 수 있어야 한다

인터페이스 분리 원칙(ISP, Interface Segregation Principle): 클라이언트는 자신이 이용하지 않는 메서드에 의존하지 않아야 한다, 클라이언트가 사용하지 않는 인터페이스 때문에 영향을 받아서는 안된다

의존성 역전 원칙(DIP, Dependency Inversion Principle): 의존 관계를 맺을 때 변화하기 쉬운 것 보다 변화하기 어려운 것에 의존해야 한다

파이프 필터 형태의 소프트웨어 아키텍쳐

파이프 필터: 파이프를 통해 받은 데이터를 변경, 결과를 파이프로 전송, 서브 시스템이 입력 데이터를 받아 처리하고 결과를 다른 시스템에 보내는 작업이 반복되는 아키텍쳐 스타일, 컴포넌트인 필터 간에 데이터를 전송하는 구조

서브시스템이 입력데이터를 받아 처리하고 결과를 다음 서브시스템으로 넘겨주는 과정을 반복

틀린 것:

데이터는 파이프를 통해 양방향으로 흐르며, 필터 이동 시 오버헤드가 발생하지 않는다(단/양방향 가능, 오버헤드 발생 가능)

노드와 간선으로 구성

계층 모델이라고도 한다

3개의 서브시스템(모델, 뷰, 제어)으로 구성

코드: 데이터를 사용 목적에 따라 식별, 분류, 배열하기 위하여 사용되는 숫자, 문자 또는 기호로 컴퓨터 처리에 효율적인 것을 선정

종류: 순차코드, 블록코드, 그룹 분류 코드, 표의 숫자 코드, 십진 분류 코드, 연상 코드, 약자 코드, 끝자리 분류 코드

순차코드: Sequence Code, 자료의 발생 순, 크기 순, 가나다 순 등 일정 순서대로 코드

블록코드: Block Code, 구분코드, 코드화 대상을 미리 파악하여 블록으로 구분한 후 그 안에서 순서대로 코드를 부여

그룹 분류 코드: Group Classification Code, 구분 코드를 세분화한 형태로 대분류, 중분류, 소분류 등 각 분류별로 자릿수를 구성

표의 숫자 코드(Significant Digit Code, 표현하려는 대상의 의미는 제외, 수치만을 모아 만든 것, 대상이 되는 물체의 중량, 면적, 크기 등을 직접 코드에 적용

십진 분류 코드(Decimal Classification Code, 코드화 대상물을 일정한 소속으로 구분하여 십진수 한 자리씩 구분하여 대분류하고, 같은 방법으로 중분류, 소분류한 코드

연상 코드: Mnemonic Code, 숫자나 문자를 조합해서 나타내는 것, 어떤 내용을 기억할 수 있도록 표시한 기호 코드

약자 코드: Letter Code, 일반적으로 사용해온 단위의 약자를 코드로 사용

끝자리 분류 코드: Final Digit Code, 다른 종류의 코드와 조합해서 사용, 코드의 끝에 붙여서 그 의미를 표현

소프트웨어

상위 설계: 아키텍쳐 설계, 데이터 설계, 시스템 분할, 인터페이스 정의, UI 설계

하위 설계: 모듈 설계, 인터페이스 설계

자료 사전(Data Dictionary)

=: 정의, ~로 구성되어 있다

+: 구성, 연결, 그리고

[ ]: 택일, 선택, 또는

{ }: 반복

( ): 생략 가능, 생략 가능한 자료

\*\*: 설명, 주석

소프트웨어의 사용자 인터페이스개발시스템(User Interface Development System)

사용자 입력의 검증

에러 처리와 에러 메시지 처리

도움과 프롬프트(Prompt) 제공

틀린 것: 소스 코드 분석 및 오류 복구(보통 백엔드에서 컴파일러가 하는 역할)

요구사항 명세 기법에 대한 설명

비정형 명세 기법은 사용자의 요구를 표현할 때 자연어를 기반으로 서술한다

정형 명세 기법은 사용자의 요구를 표현할 때 수학적인 원리와 표기법을 이용한다

정형 명세 기법은 비정형 명세 기법에 비해 표현이 간결하다

정형 명세법

수학적 기반/모델링 기반

Z, VDM, Petri-Net: 모형 기반

CSP, CCS, LOTOS: 대수적 방법

시스템 요구 특성이 정확하고 명세가 간결, 명세와 구현이 일치

이해도가 낮으며 이해관계자의 작성 부담 가중

비정형 명세법

상태, 기능, 객체 중심 명세법

FSM(Finite State Machine)

Decision Table, E-R 모델링

State Chart(SADT)

UseCase: 사용자 기반 모델링

명세 작성이 간편하고 의사 전달 방법이 다양

불충분한 명세 가능성, 모호성

틀린 것: 비정형 명세 기법은 사용자의 요구를 표현할 때 Z 비정형 명세기법을 사용한다

소프트웨어 개발 단계 요구 분석 과정

분석 결과의 문서화를 통해 향후 유지 보수에 유용하게 활용할 수 있다

자료 흐름도, 자료 사전 등이 효과적으로 이용될 수 있다

보다 구체적인 명세를 위해 소단위 명세서(Mini-Spec)가 활용될 수 있다

틀린 것: 개발 비용이 가장 많이 소요되는 단계이다(유지보수 단계가 가장 많이 소요)

미들웨어 솔루션: 컴퓨터-컴퓨터 간의 연결을 쉽고 안전하게 할 수 있도록 해주고, 이에 대한 관리를 도와주는 SW

유형: DB 미들웨어, 원격 프로시저 호출(RPC), 메시지 지향 미들웨어(MOM), 트랜잭션 처리(TP) 모니터, 레거시웨어(Legacyware), 객체기반(ORB) 미들웨어, WAS(Web Application Server)

바람직한 소프트웨어 설계 지침

적당한 모듈의 크기를 유지

모듈 간의 접속 관계를 분석, 복잡도와 중복을 줄인다

모듈 간의 결합도는 약할수록 바람직하다

모듈 간의 효과적인 제어를 위해 설계에서 계층적 자료 조직이 제시되어야 한다