**<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

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

확장

일반화

틀린 것: 구체화

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

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) 방법론의 특징

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

Prototype: 원본 객체를 복사함으로써 객체를 생성

Singleton: 어떤 클래스의 인스턴스는 하나임을 보장하고 어디서든 참조할 수 있도록 함

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

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

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

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 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

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)의 각 요소별 표기 형태의 연결

Process: 원

Data Flow: 화살표

Data Store: 평행선

Terminator: 사각형

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

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

Rumbaugh Method: 객체지향 분석기법의 하나, 객체 모형, 동적 모형, 기능 모형의 3개 모형을 생성하는 방법

객체 모형: 객체 다이어그램

동적 모형: 상태 다이어그램

기능 모형: 자료 흐름도

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

Jacobson Method: Use Case를 강조하여 사용하는 분석방법

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

* 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, 점선 삼각 화살표

일반화 관계(generalization Relationship): 객체지향에서 상속관계, 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

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

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

틀린 것: 파이프 필터 아키텍쳐에서 데이터는 파이프를 통해 양방향으로 흐르며, 필터 이동 시 오버헤드가 발생하지 않는다(단방향, 양방향 모두 가능, 오버헤드 발생 가능)

파이프 필터: 데이터 통로인 파이프를 이용해 컴포넌트인 필터 간에 데이터를 전송하는 구조

**<2과목: 소프트웨어 개발>**

통합 테스트(Integration Test)와 관련한 설명

시스템을 구성하는 모듈의 인터페이스와 결합을 테스트

하향식 통합테스트의 경우 넓이우선(Breadth First) 방식으로 테스트를 할 모듈을 선택할 수 있다

모듈 간의 인터페이스와 시스템의 동작이 정상적으로 잘되고 있는지를 빨리 파악하고자 할 때 상향식보다는 하향식 통합 테스트를 사용하는 것이 좋다

모듈을 통합하는 과정에서 모듈 간의 호환성을 확인하기 위해 수행되는 테스트

틀린 것: 상향식 통합 테스트의 경우 시스템 구조도의 최상위에 있는 모듈을 먼저 구현하고 테스트한다

이진검색방법

중간값 설정 – 찾고자 하는 값과 크기 비교 – 중간값이 새로운 최대값 or 최소값이 됨 – 반복

소프트웨어 공학에서 워크스루(Walkthrough)에 대한 설명

사용사례를 확장하거나 명세하거나 설계 다이어그램, 원시코드, 테스트 케이스 등에 적용할 수 있다

복잡한 알고리즘 또는 반복, 실시간 동작, 병행 처리와 같은 기능이나 동작을 이해하려고 할 때 유용

단순한 테스트 케이스를 이용하여 프로덕트를 수작업으로 수행해 보는 것이다

틀린 것: 인스펙션(Inspection)과 동일한 의미를 가진다

형상관리: 소프트웨어 개발과정에서 소프트웨어의 변경사항을 관리하기 위해 개발된 일련의 활동을 뜻하는 것

형상 관리를 통해 가시성과 추적성을 보장함으로써 소프트웨어의 생산성과 품질을 높일 수 있다

형상관리 절차: 형상 식별 – 형상 통제 – 형상 감사 – 형상 기록/보고

형상 식별: 형상관리 계획을 근거로 형상관리의 대상이 무엇인지 식별, 관리 대상 구분, 관리 목록 번호 정의

형상관리 대상: 품질관리 계획서/매뉴얼, 요구사항 명세서, 설계/인터페이스 명세서, 테스트 설계서, 소스코드

형상통제: 소프트웨어 형상 변경 제안을 검토, 현재 소프트웨어 기준선(Baseline)에 반영하도록 통제, 형상통제 위원회(Configuration Control Board, CCB)의 승인을 통한 변경 통제가 이루어짐

형상 감사: 형상 관리 계획대로 형상관리가 진행되고 있는지, 형상 항목의 변경이 요구사항에 맞도록 제대로 이루어졌는지 등을 검토/승인하는 것, 개발자, 유지보수 담당자가 아닌 제 3자의 객관적인 확인 및 검증 과정을 통해 새로운 형상의 무결성을 확보하는 활동

형상 기록/보고: 소프트웨어 개발 상태에 대한 보고서를 제공하는 것, 베이스라인 산출물에 대한 변경과 처리 과정에서의 변경을 상태 보고에 모두 기록

테스트 케이스와 관련한 설명

프로그램에 결함이 있더라도 입력에 대해 정상적인 결과를 낼 수 있기 때문에 결함을 검사할 수 있는 테스트 케이스를 찾는 것이 중요

개발된 서비스가 정의된 요구사항을 준수하는지 확인하기 위한 입력값과 실행조건, 예상결과의 집합으로 볼수있다

테스트 케이스 실행이 통과되었는지 실패하였는지 판단하기 위한 기준을 테스트 오라클(Test Oracle)이라고 한다

틀린 것: 테스트의 목표 밑 테스트 방법을 결정하기 전에 테스트 케이스를 작성해야 한다

객체지향 개념을 활용한 소프트웨어 구현과 관련한 설명

객체(Object)란 필요한 자료구조와 수행되는 함수들을 가진 하나의 독립된 존재

상속(Inheritance)은 개별 클래스를 상속관계로 묶음으로써 클래스 간의 체계화된 전체구조를 파악하기 쉽다는 장점

같은 클래스에 속하는 개개의 객체이자 하나의 클래스에서 생성된 객체를 인스턴스(Instance)라고 한다

틀린 것: JAVA에서 정보은닉(Information Hiding)을 표기할 때 private의 의미는 ‘공개’이다

DRM(Digital Rights Management)과 관련한 설명

디지털 콘텐츠와 디바이스의 사용을 제한하기 위해 하드웨어 제조업자, 저작권자, 출판업자 등이 사용할 수 있는 접근 제어 기술을 의미

디지털 미디어의 생명 주기 동안 발생하는 사용권한관리, 과금, 유통 단계를 관리하는 기술로도 볼 수 있다

클리어링 하우스(Clearing House)는 사용자에게 콘텐츠 라이선스를 발급하고 권한을 부여해주는 시스템을 말한다

틀린 것: 원본을 안전하게 유통하기 위한 전자적 보안은 고려하지 않기 때문에 불법 유통과 복제의 방지가 불가능

위험 모니터링의 의미

위험 요소 징후들에 대하여 계속적으로 인지하는 것

틀린 것:

위험을 이해하는 것

첫번째 조치로 위험을 피할 수 있도록 하는 것

위험 발생 후 즉시 조치하는 것

동시에 소스를 수정하는 것을 방지, 다른방향으로 진행된 개발결과를 합치거나 변경내용을 추적할 수 있는 소프트웨어 버전관리도구

RCS(Revision Control System)

틀린 것:

RTS(Reliable Transfer Service)

RPC(Remote Procedure Call)

RVS(Relative Version System)

화이트박스 테스트와 관련한 설명

화이트박스 테스트의 이해를 위해 논리흐름도(Logic-Flow Diagram)을 이용할 수 있다

테스트 데이터를 이용해 실제 프로그램을 실행함으로써 오류를 찾는 동적테스트(Dynamic Test)에 해당한다

테스트 데이터를 선택하기 위하여 검증기준(Test Coverage)를 정한다

틀린 것: 프로그램의 구조를 고려하지 않기 때문에 테스트케이스는 프로그램 또는 모듈의 요구, 명세를 기초로 결정

알고리즘에 관련한 설명

주어진 작업을 수행하는 컴퓨터 명령어를 순서대로 나열한 것

검색(Searching)은 정렬되지 않는 데이터 혹은 정렬이 된 데이터 중에서 키값에 해당되는 데이터를 찾는 알고리즘

정렬(Sorting)은 흩어져있는 데이터를 키값을 이용하여 순서대로 열거하는 알고리즘

틀린 것: 선형검색은 검색을 수행하기 전에 반드시 데이터의 집합이 정렬되어 있어야 한다

버블 정렬

처음부터 순서대로 각 값과 바로 인접한 다음 값 사이의 크기비교를 통해 정렬

인스펙션(Inspection) 과정

계획 – 사전교육 – 준비 – 인스펙션 회의 – 수정 – 후속조치 or 계획

소프트웨어를 보다 쉽게 이해할 수 있고 적은 비용으로 수정할 수 있도록 겉으로 보이는 동작의 변화 없이 내부구조를 변경하는 것

Refactoring

틀린 것:

Architecting

Specification

Renewal

단위 테스트(Unit Test)와 관련한 설명

구현 단계에서 각 모듈의 개발을 완료한 후 개발자가 명세서의 내용대로 정확히 구현되었는지 테스트

모듈 내부의 구조를 구체적으로 볼 수 있는 구조적 테스트를 주로 시행

테스트할 모듈을 호출하는 모듈도 있고, 테스트할 모듈이 호출하는 모듈도 있다

개별 모듈을 시험하는 것으로 모듈이 정확하게 구현되었는지, 예정한 기능이 제대로 수행되는지를 점검

내부에 존재하는 논리적인 오류를 검출

틀린 것: 필요데이터를 인자를 통해 넘겨주고, 테스트 완료 후 그 결과값을 받는 역할을 하는 가상의 모듈을 테스트 스텁(Stub)이라고 한다

IDE(Integrated Development Environment) 도구의 각 기능에 대한 설명

Coding – 프로그래밍 언어를 가지고 컴퓨터 프로그램을 작성할 수 있는 환경을 제공

Debugging – 프로그램에서 발견되는 버그를 찾아 수정할 수 있는 기능

Deployment – 소프트웨어를 최종 사용자에게 전달하기 위한 기능

틀린 것: Compile – 저급언어의 프로그램을 고급언어 프로그램으로 변환하는 기능

트리구조 순회

전위 순회(Preorder Traversal): root –> left –> right

중위 순회(Inorder Traversal): left –> root -> right

후위 순회(Postorder Traversal): left -> right -> root

인터페이스 구현 시 사용하는 기술, 속성-값 쌍(Attribute-Value Pairs)으로 이루어진 데이터 오브젝트를 전달하기 위해 사용하는 개방형 표준 포맷

JSON

틀린 것:

HTML

AVPN

DOF

순서가 있는 리스트에서 데이터의 삽입(Push), 삭제(Pop)가 한 쪽 끝에서 일어나며 LIFO(Last-In-First-Out)인 자료구조

Stack

틀린 것:

Tree

Graph

Queue

단위테스트 도구

CppUnit: C++

Junit: JAVA

HttpUnit: HTML., JAVA

틀린 것: IgpUnit

클린 코드를 작성하기 위한 원칙

의존성: 다른 모듈에 미치는 영향을 최소화하도록 작성한다

가독성: 누구든지 읽기 쉽게 코드를 작성한다

중복성: 중복을 최소화할 수 있는 코드를 작성

추상화: 상위클래스; 프로그램 특성만 간단하게, 하위 클래스; 세부적 내용 구현

단순성: 코드를 단순하게 작성, 프로그램을 최소 단위로 분리해 작업을 한번에 하나씩 처리하도록

테스트 드라이버: 단위 테스트에서 테스트의 대상이 되는 하위 모듈을 호출하고, 파라미터를 전달, 결과를 받아오는 가상의 모듈, 상향식 테스트에 필요한 것

테스트 스텁: 테스트 대상 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구, 일시적으로 필요한 조건만을 가지고 있는 테스트용 모듈, 하향식 테스트에 사용

테스트 슈트: 테스트 대상 컴포넌트나 모듈, 시스템에 사용되는 테스트 케이스의 집합

테스트 케이스: 사용자의 요구사항을 정확히 준수했는지 확인하기 위한 입력 값, 실행조건, 기대 결과 등으로 만들어진 테스트 항목의 명세서

스택: 인터럽트 처리, 서브루틴 호출 작업 등에 응용된다

소프트웨어 모듈화의 장점

오류의 파급 효과를 최소화

모듈의 재사용 가능으로 개발과 유지보수가 용이

프로그램의 효율적인 관리가 가능

이해하기 쉬운 소프트웨어

틀린 것: 기능의 분리가 가능하여 인터페이스가 복잡하다

소프트웨어 프로젝트 관리에 대한 설명

주어진 기간 내에 최소의 비용으로 사용자를 만족시키는 시스템을 개발

프로젝트 관리는 소프트웨어 개발 계획을 세우고 분석, 설계, 구현 등의 작업을 통제하는 것으로 소프트웨어 생명 주기의 전 과정에 걸쳐 진행

소프트웨어 프로젝트를 성공적으로 수행하기 위해서는 수행할 작업의 범위, 필요한 자원, 수행 업무, 이정표, 비용 추진 일정들을 알아야함

소요인력은 최소화하는 것이 좋음

정형 기술 검토(FTR)의 지침

의제의 제한성: 의제를 제한

논쟁 반박의 제한성: 논쟁과 반박을 제한

문제 공개성: 문제 영역을 명확히 표현

참가 인원의 제한성: 참가자 수 제한

문서성: 발견된 오류는 문서화

안건 고수성: 안건을 세우면 고수

제품 검토의 집중성: 오류 검출에 초점을 두고 해결책을 나중으로 미룸

사전 준비성: 검토를 위한 자료를 사전에 배포하여 검토하도록 함

Migration: 소프트웨어 재공학의 주요 활동 중 기존 소프트웨어 시스템을 새로운 기술 또는 하드웨어 환경에서 사용할 수 있도록 변환하는 작업

Analysis: 기존 소프트웨어를 분석하여 재공학 대상을 선정하는 것

Restructuring: 기존 소프트웨어를 향상시키기 위하여 코드를 재구성하는 작업(기능과 외적 동작은 변하지 않음)

Reverse Engineering: 기존 소프트웨어를 분석하여 소스코드를 얻어내는 작업

정보시스템 개발 단계에서 프로그래밍 언어 선택 시 고려할 사항

친밀감

언어의 능력

처리의 효율성

개발 정보 시스템의 구조, 특성, 길이

사용자의 요구사항

컴파일러의 가용성

소프트웨어 패키징에 대한 설명

신규 및 변경 개발소스를 식별하고, 이를 모듈화하여 상용 제품으로 패키징한다

고객의 편의성을 위해 매뉴얼 및 버전 관리를 지속적으로 한다

범용 환경에서 사용이 가능하도록 일반적인 배포 형태로 패키징이 진행된다

소비자 중심 패키징

자료 구조 분류

선형구조: 리스트, 스택, 큐, 데크

비선형구조: 트리, 그래프

파일구조: 순차파일, 색인파일, 직접파일

Alien Code: 아주 오래되거나 참고문서 또는 개발자가 없어 유지보수 작업이 아주 어려운 프로그램

소프트웨어를 재사용함으로써 얻을 수 있는 이점

생산성 증가

프로젝트 문서, 구축 방법 공유, 실패 위험 감소

소프트웨어 품질 및 생산성 향상

개발 시간과 비용 단축

소프트웨어를 재사용함으로써 얻을 수 있는 문제점

시스템에 공통적으로 사용되는 요소 발견 필요

프로그램의 표준화 부족

새로운 개발 방법론 도입 어려움

재사용을 위한 관리 및 지원 부족

기존 소프트웨어에 재사용 소프트웨어를 추가하기 어려움

인터페이스 간의 통신을 위해 이용되는 데이터 포맷

JSON: 키-값 쌍 데이터 오브젝트를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷

XML: HTML의 단점 보완 인터넷 언어, SGML의 복잡한 단점을 개선한 특수 목적을 갖는 마크업 언어

YAML: 사람이 쉽게 읽을 수 있는 데이터 직렬화 양식

틀린 것:

AJAX: 자바스크립트를 사용하여 웹 서버와 클라이언트 간 비동기적으로 XML 데이터를 교환하고 조작하기 위한 웹기술, XMLHttpRequest 객체를 이용해 전체 페이지를 새로 로드하지 않고 필요한 부분만 로드한다

프로그램 설계도의 하나인 NS Chart에 대한 설명

논리의 기술에 중점을 두고 도형을 이용한 표현 방법

이해하기 쉽고 코드 변환이 용이하다

연속, 선택, 반복 등의 제어 논리 구조를 표현

기본 구조의 입구와 출구는 각 하나씩

전체적인 알고리즘을 일목요연하게 볼 수 있음

3가지 기본 구조만을 논리를 표현(표준화 가능)

Flow Chart의 최대 단점인 화살표가 표시되지 않음

Quick Sort: 분할 정복(Divide and Conquer)에 기반한 알고리즘으로 피벗(Pivot)을 사용하며 최악의 경우 n(n-1)/2 회(O(n^2))의 비교를 수행해야 하는 정렬

Selection Sort: 배열 내에서 최소값을 찾아 정렬되지 않은 맨 앞 값과 교환을 하며 정렬을 해 나가는 방법(O(n^2))

Bubble Sort: 왼쪽에서부터 두 데이터의 크기를 비교해서 자리를 바꾸는 방법(O(n^2))

Insert Sort: 한 개의 값을 추출, 앞쪽으로 비교해서 본인의 자리를 알맞게 찾아가게끔 하는 방법, (O(n) or O(n^2))

Merge Sort: 분할 정복에 기반한 알고리즘, 리스트를 1 이하인 상태까지 절반으로 자른 다음 다시 재귀적으로 Merge Sort을 이용해서 전체적인 리스트를 합병하는 방법(O(n log n))

Sort: 선택(Selection), 버블(Bubble), 삽입(Insert), 쉘(Shell), 퀵(Quick), 힙(heap), 병합(Merge), 기수(Radix)

화이트박스 검사 기법

기초 경로 검사

조건 검사

루프 검사

데이터 흐름 검사

블랙박스 검사 기법

동치(동등)분할 검사

경계값 분석

원인-효과 검사

오류예측 검사

비교 검사

소프트웨어 품질 관련 국제 표준인 ISO/IEC 25000에 관한 설명

소프트웨어 품질 평가를 위한 소프트웨어 품질평가 통합 모델 표준이다

System and Software Quality requirements and Evaluation으로 줄여서 SQuaRE라고도 한다

기존 소프트웨어 품질 평가 모델과 소프트웨어 평가 절차 모델인 ISO/IEC 9126과 ISO/IEC 14598을 통합

틀린 것: ISO/IEC 2501n에서는 소프트웨어의 내부 측정, 외부 측정, 사용품질 측정, 품질 측정 요소 등을 다룬다

코드 인스펙션에 관련한 설명

프로그램을 수행시켜보는 것 대신에 읽어보고 눈으로 확인하는 방법으로 볼 수 있다

코드 품질 향상 기법 중 하나이다

결함과 함께 코딩 표준 준수 여부, 효율성 등의 다른 품질 이슈를 검사하기도 한다

정적 테스트

표준이나 명세서에 서술한 내용과 비교하여 편차와 에러를 식별하기 위해 산출물을 근거로 수행하는 검사

Risk Analysis: 프로젝트에 내재된 위험 요소를 인식하고 그 영향을 분석하여 이를 관리하는 활동으로서, 프로젝트를 성공시키기 위하여 위험 요소를 사전에 예측, 대비하는 모든 기술과 활동을 포함

Critical Path Method(CPM, 임계 경로법): 네트워크 중심의 논리적 구성, 시간과 비용 문제를 취급, 프로젝트를 일정 기일 내에 완성시키고 해당 계획이 원가의 최소값에 의해 보증되는 등의 최적 스케줄을 구하는 관리 방법

Work Breakdown Structure(업무 분업 구조): 성과 목표 완전 달성을 위한 프로그램, 산업 관리 간접 부문의 기술 혁신형 업무-목표를 설정하여 소정 기간, 자원 내에서 달성하는 형태의 업무-를 효과적으로 수행하기 위한 수법

Waterfall Model(폭포수 모델): S/W 개발 생명주기에 기반하고 있는 소프트웨어 개발 기법 중 하나, 각 개발 단계를 확실히 매듭 짓고 다음 단계로 넘어간다는 의미

시스템 테스트의 종류: 요구사항, 무결성, 부피, 메모리 성능, 신뢰성, 부하, 수락, 회복, 안전, 구조, 회귀, 병행 등

강도(Stress) 테스트: 시스템에 과다 정보량을 부과하여 과부하 시에도 시스템이 정상적으로 작동되는지를 테스트

회복(Recovery) 테스트: 시스템에 고의로 실패를 유도하고 시스템이 정상적으로 복귀하는지 테스트

성능 테스트: 사용자의 이벤트에 시스템이 응답하는 시간, 특정 시간내에 처리하는 업무량, 사용자 요구에 시스템이 반응하는 속도 등을 테스트

안전(Security) 테스트: 부당하고 불법적인 침입을 시도하여 보안 시스템이 불법적인 침투를 잘 막아내는지 테스트

깊이 우선 탐색(Depth First Search): 최대한 깊이 탐색한 이후 더 이상 탐색할 것이 없다면 그 이전으로 돌아감

모듈(Module): 소프트웨어 구조를 이루며 다른 것들과 구별될 수 있는 독립적인 기능을 갖는 단위, 하나 또는 몇 개의 논리적인 기능을 수행하기 위한 명령어들의 집합, 서로 모여 하나의 완전한 프로그램으로 만들어질 수 있다

데이터(Data): 관찰하고 측정해서 얻은 Value

저장소: 데이터를 논리적 구조로 조직화, 혹은 물리적 공간에 구축한 것

오버플로우(Overflow): 스택 공간이 가득찼을 때 하나의 데이터를 더 넣으려고 하는 경우

언더플로우(Underflow): 스택 공간에 데이터가 없는데 프로그램이 스택에서 데이터를 꺼내려고 하는 경우

제품 소프트웨어의 사용자 매뉴얼 작성절차

컴포넌트 명세서, 구현 설계서: 설치와 사용에 필요한 제반 절차 및 환경 등 전체 내용을 포함하는 매뉴얼 작성

작성 지침 정의 – 사용 설명서 구성 요소 정의 – 구성 요소별 내용 작성 – 사용 설명서 검토

검증(Verification): 소프트웨어가 요구사항에 부합하게 구현되었음을 보장하는 활동, 개발자 기준

확인(Validation): 소프트웨어가 고객의 의도에 따라 구현되었음을 보장하는 활동, 사용자 기준

소프트웨어 테스트에서 검증과 확인을 구별하면 찾고자 하는 결함 유형을 명확하게 하는데 도움이 된다

검증은 소프트웨어 개발 과정을 테스트, 확인은 소프트웨어 결과를 테스트

검증은 작업 제품이 요구 명세의 기능, 비기능 요구사항을 얼마나 잘 준수하는지 측정하는 작업

시스템 테스트: ‘완전한 시스템’에 대해 수행하는 테스트, 기능적, 비기능적 요구사항을 만족하는지 확인

인수 테스트(Acceptance Test): 실제 환경에서 사용자가 참여하는 테스트, 시스템이 예상대로 동작하는지 사용자의 관점에서 확인

소스 코드 정적분석(Static Analysis)에 대한 설명

소스 코드를 실행시키지 않고 분석

코드에 있는 잠재적인 오류, 보안 취약점, 위험을 찾아내기 위한 활동

자료 흐름이나 논리 흐름을 분석하여 비정상적인 패턴을 찾을 수 있다

틀린 것: 하드웨어적인 방법으로만 코드 분석이 가능하다

Fault: 소프트웨어 개발 활동을 수행함에 있어 시스템이 고장(Failure)을 일으키게 하며, 오류(Error)가 있는 경우 발생, 오류 및 작동 실패의 원인, 개발자의 설계와 다르게 동작하거나 제작 의도와 다른 결과가 발생하는 것

코드의 간결성을 유지하기 위해 사용되는 지침

공백을 이용하여 실행문 그룹과 주석을 명확히 구분한다

복잡한 논리식과 산술식은 괄호와 들여쓰기(Indentation)를 통해 명확히 표현한다

빈 줄을 사용하여 선언부와 구현부를 구별한다

틀린 것: 한 줄에 최대한 많은 문장을 코딩한다

품질 목표 항목: 정확성, 신뢰성, 효율성, 무결성, 유지보수 용이성, 사용 용이성, 검사 용이성, 이식성, 상호 운용성, 유연성, 재사용성

Portability: 이식성, 하나 이상의 하드웨어 환경에서 운용되기 위해 쉽게 수정될 수 있는 시스템 능력

Efficiency: 효율성, 최소의 작업으로 요구되는 기능을 수행하는 정도

Usability: 사용 용이성, 소프트웨어를 쉽게 사용할 수 있는 정도

Correctness: 정확성, 사용자의 요구사항을 충족시키는 정도

트리 구조 검색 효율

최악의 경우

이진 탐색 트리: O(n)

2-3 트리: O(log 3n)

AVL 트리: O(log n)

레드 블랙 트리: O(log n)

테스트 케이스 자동 생성 도구를 이용하여 테스트 데이터를 찾아내는 방법

입력 도메인 분석

랜덤 테스트

자료 흐름도

기능 테스트

틀린 것: 스텁(하향식 테스트에 사용되는 테스트용 임시 모듈), 드라이버(상향식 테스트에 사용되는 테스트 가동기)

저작권 관리 구성 요소

컨텐츠 제공자: 컨텐츠를 제공하는 저작권자

컨텐츠 분배자: 암호화된 컨텐츠 제공

패키저(Packager): 컨텐츠를 메타 데이터와 함께 배포 가능한 단위로 묶는다

보안 컨테이너: 원본을 안전하게 유통하기 위한 전자적 보안장치

DRM 컨트롤러: 배포된 컨텐츠의 이용 권한을 통제

클리어링 하우스: 소비자와 유통업자 사이에 발생하는 거래에 대해 디지털 저작권 라이선싱을 중개하고 라이선스 발급을 수행하는 장소

**<3과목: 데이터베이스 구축>**

데이터베이스 정규화

데이터 구조의 안정성을 최대화

중복을 배제하여 삽입, 삭제, 갱신 이상의 발생을 방지

데이터 삽입 시 릴레이션을 재구성할 필요성을 줄임

틀린 것: 데이터베이스의 개념적 설계 단계 이전에 수행(논리적 설계 단계에서 수행)

1차 정규화: 각 행마다 열의 값이 1개씩만 있어야함, 원자값(Atomic value); 열의 값이 여러 개인 경우 행으로 쪼갠다

2차 정규화: 모든 열이 완전 함수적 종속을 만족, 기본키 중 특정 열에만 종속된 열(부분적 종속)이 없어야 한다

3차 정규화: 이행적 종속을 없애도록 테이블을 분해, A -> B, B -> C가 성립할 때 A -> C가 성립되는 것을 없앰

BCNF 정규화: 모든 결정자가 후보키가 되도록 테이블을 분해, 후보키; 최소성, 유일성 만족

4차 정규화: 함수 종속이 아닌 다치 종속을 제거(다치종속: 두 개의 독립된 속성이 1:N으로 대응하는 관계)

5차 정규화: 후보키를 통하지 않는 조인 종속을 제거(조인종속: 테이블에 포함된 필드들의 부분집합을 포함하는 테이블들을 JOIN하여 생성할 수 있는 것)

데이터베이스 트랜잭션 성질

트랜잭션: 데이터베이스 시스템에서 병행제어 밑 회복작업시 처리되는 작업의 논리적 단위

사용자가 시스템에 대한 서비스 요구시 시스템이 응답하기 위한 상태변환 과정의 작업 단위

Atomicity(원자성): 트랜잭션의 모든 연산들이 정상적으로 수행 완료되거나 아니면 전혀 어떠한 연산도 수행되지 않은 원래 상태가 되도록 해야 한다

Consistency(일관성): 트랜잭션이 그 실행을 성공적으로 완료하면 언제나 일관성 있는 데이터데이스 상태로 변환

시스템이 가지고 있는 고정요소는 수행 전과 수행 완료 후의 상태가 같아야 한다

Isolation(독립성, 격리성): 둘 이상의 트랜잭션이 동시에 병행 실행되는 경우 어느 하나의 트랜잭션 실행 중에 다른 트랜잭션의 연산이 끼어들 수 없다

수행중인 트랜잭션은 완전히 완료될 때 까지 다른 트랜잭션에서 수행결과를 참조할 수 없다

Durability(영속성, 지속성): 성공적으로 완료된 트랜잭션의 결과는 시스템이 고장나더라도 영구적으로 반영되야 한다

분산 데이터베이스 시스템과 관련한 설명

물리적으로 분산된 데이터베이스 시스템을 논리적으로 하나의 데이터베이스 시스템처럼 사용할 수 있도록 한 것

물리적으로 분산되어 지역별로 필요한 데이터를 처리할 수 있는 지역 컴퓨터(Local Computer)를 분산처리기(Distributed Processor)라고 한다

분산 데이터베이스 시스템을 위한 통신 네트워크 구조가 데이터 통신에 영향을 주므로 효율적으로 설계해야 한다

틀린 것: 데이터베이스가 분산되어 있음을 사용자가 인식할 수 있도록 분산투명성(Distribution Transparency)를 배제해야 한다

데이터베이스의 인덱스에 관련한 설명

문헌의 색인, 사전과 같이 데이터를 쉽고 빠르게 찾을 수 있도록 만든 데이터 구조

인덱스의 기본 목적은 검색 기능을 최적화 하는 것

B-트리 인덱스는 분기를 목적으로 하는 Branch Block을 가지고 있다(Ordered Index, 순서 인덱스)

Between 등 범위(Range) 검색에 활용될 수 있다

테이블에 붙여진 색인으로 데이터 검색 시 처리 속도 향상에 도움이 된다

대부분의 데이터베이스에서 테이블을 삭제하면 인덱스도 같이 삭제

틀린 것:

시스템이 자동으로 생성하여 사용자가 변경할 수 없다(기본키에 대한 인덱스만 자동 생성)

인덱스의 추가, 삭제 명령어는 각각 ADD, DELETE이다

물리적 데이터베이스 구조의 기본 데이터 단위인 저장 레코드의 양식을 설계할 때 고려사항

데이터 타입

데이터 값의 분포

접근 빈도

틀린 것: 트랜잭션 모델링

SQL의 기능에 따른 분류 중에서 REVOKE문과 같이 데이터의 사용권한을 관리하는데 사용하는 언어

DCL(Data Control Language): 사용자관리, 사용자별 릴레이션 또는 데이터를 관리하고 접근하는 권한, Grant, Revoke

틀린 것:

DDL(Data Definition Language): 각 릴레이션을 정의, Create, Alter, Drop, Rename, Truncate

DML(Data Manipulaton Language): 데이터를 추가/수정/삭제, Select, Insert, Update, Delete

DUL(Data User Language): 없음

TCL(Transaction Control Language): 논리적인 작업의 단위를 묶어서 DML에 의해 조작된 결과를 작업단위 별로 제어

Commit, Rollback, Savepoint

데이터 사전에 대한 설명

시스템 카탈로그 밑 시스템 데이터베이스라고도 한다

데이터베이스에 대한 데이터인 메타데이터를 저장하고 있다

데이터 사전에 있는 데이터를 실제로 접근하는 데 필요한 위치정보는 데이터 디렉토리라는 곳에서 관리한다

틀린 것: 데이터 사전 역시 데이터베이스의 일종이므로 일반 사용자가 생성, 유지 및 수정할 수 있다

데이터베이스에서 릴레이션에 대한 설명

릴레이션: 관계형 데이터베이스에서 정보를 구분하여 저장하는 기본단위, DB 테이블

모든 튜플은 서로 다른 값을 가지고 있다

각 속성은 릴레이션 내에서 유일한 이름을 가진다

모든 속성값은 원자값(Atomic value)를 가진다

틀린 것: 하나의 릴레이션에서 튜플은 특정한 순서를 가진다

데이터베이스에서 뷰(View)에 대한 설명

뷰는 다른 뷰를 기반으로 새로운 뷰를 만들 수 있다

뷰는 일종의 가상테이블이며, update에는 제약이 따른다

뷰는 기본테이블을 만드는 것처럼 create view를 사용하여 만들 수 있다

DBA는 보안성 측면에서 뷰를 활용할 수 있다

사용자가 필요한 정보를 요구에 맞게 가공하여 뷰로 만들 수 있다

틀린 것: 뷰는 논리적으로 존재하는 기본테이블과 다르게 물리적으로만 존재하며 카탈로그에 저장된다

트랜잭션의 상태 중 트랜잭션의 마지막 연산이 실행된 직후의 상태로, 모든 연산의 처리는 끝났지만 트랜잭션이 수행한 최종 결과를 데이터베이스에 반영하지 않은 상태

Partially Committed

틀린 것:

활성화(Active) - Read/Write – 부분완료(Partially committed) – Permanent Change – 완료(committed)

| Failure | Failure

실패(Failed) – Rollback – 철회(Aborted)

Active: 트랜잭션이 작업을 시작하여 실행 중인 상태

Failed: 트랜잭션에 오류가 발생하여 실행이 중단된 상태

Aborted: 트랜잭션이 비정상적으로 종료되어 Rollback 연산을 수행한 상태

Committed: 트랜잭션이 성공적으로 종료되어 commit 연산을 실행한 후의 상태

키의 종류 중 유일성과 최소성을 만족하는 속성 또는 속성들의 집합

후보키(Candidate key): 릴레이션을 구성하는 속성들 중 튜플을 유일하게 식별하기 위해 사용되는 속성들의 부분집합

틀린 것:

Atomic key: 없음, Atomic 한 열에 하나의 데이터

Super key: 슈퍼키, 한 릴레이션 내에 있는 속성들의 집합으로 구성된 키, 릴레이션을 구성하는 모든 튜플 중 슈퍼키로 구성된 슈퍼키로 구성된 속성의 집합과 동일한 값은 나타내지 않는다. 릴레이션을 구성하는 모든 튜플에 대해 유일성은 만족, 최소성은 만족하지 못한다

Primary key: 기본키, 후보키 중 특별히 선정된 키, 중복된 값을 가질 수 없다. Null을 가질 수 없다

Alternate key: 대체키, 후보키 중 선정된 기본키를 제외한 나머지 후보키

Foreign key: 외래키, 다른 릴레이션의 기본키를 참조하는 속성 또는 속성들의 집합, 다른 릴레이션 간의 관계를 표현할 때 사용, 참조 릴레이션의 기본키와 동일한 키 속성을 가진다.

유일성(Unique): 하나의 키 값으로 하나의 튜플만을 유일하게 식별할 수 있어야 한다

최소성(Minimality): 키를 구성하는 속성 하나를 제거하면 유일하게 식별할 수 없도록 꼭 필요한 최소의 속성,

데이터베이스에서 개념적 설계 단계에 대한 설명

산출물로 E-R Diagram을 만들 수 있다

DBMS에 독립적인 개념 스키마를 설계한다:

DBMS: 데이터베이스 데이터의 집합, 저장, 관리 기능 제공 프로그램

스키마: 데이터베이스의 구조와 제약조건에 관한 전반적인 명세를 기술한 메타데이터의 집합,

Entity, Attribute, Relationship, Constraint

논리적 설계 단계의 앞 단계에서 수행된다

틀린 것: 트랜잭션 인터페이스를 설계 및 작성한다

테이블의 기본키로 지정된 속성에 관한 설명

Not null로 널 값을 가지지 않는다

릴레이션에서 튜플을 구별할 수 있다

외래키로 참조될 수 있다

틀린 것: 검색할 때 반드시 필요하다

데이터 모델의 구성요소 중 데이터 구조에 따라 개념 세계나 컴퓨터 세계에서 실제로 표현된 값들을 처리하는 작업

Operation: 데이터베이스에 저장된 실제 데이터를 처리하는 작업에 대한 명세, 데이터베이스를 조작하는 기본도구

틀린 것:

Relation: 개체 간의 관계 또는 속성 간의 관계

Data Structure: 논리적으로 표현된 개체 타입들 간의 관계로서 데이터 구조 밑 정적 성질을 표현

Constraint: 데이터베이스에 저장될 수 있는 실제 데이터의 논리적인 제약조건

무결성 제약조건 중 개체 무결성 제약조건에 대한 설명

무결성 제약조건: 데이터베이스의 정확성, 일관성을 보장하기 위해 저장, 삭제, 수정 등을 제약하기 위한 조건

개체 무결성: 각 릴레이션의 기본키를 구성하는 속성은 null, 중복된 값을 가질 수 없다

참조 무결성: 외래키 값은 null이거나 참조하는 릴레이션의 기본키 값과 동일해야 한다

도메인 무결성: 속성들의 값은 정의된 도메인에 속한 값이어야 한다

고유 무결성: 특정 속성에 대해 고유값을 가지도록 조건이 주어진 경우, 릴레이션의 각 튜플이 가지는 속성값들은 서로 달라야 한다

Null 무결성: 릴레이션의 특정 속성 값은 null이 될 수 없다

키 무결성: 각 릴레이션은 최소한 한 개 이상의 키가 존재해야 한다

기본키는 null값을 가져서는 안되며 릴레이션 내에 오직 하나의 값만 존재해야 한다

틀린 것:

릴레이션 내의 튜플들이 각 속성의 도메인에 정해진 값 만을 가져야한다 -> 도메인 무결성

자식 릴레이션의 외래키는 부모 릴레이션의 기본키와 도메인이 동일해야 한다 -> 참조 무결성

자식 릴레이션의 값이 변경될 때 부모 릴레이션의 제약을 받는다 -> 참조 무결성

관계 데이터 모델에서 릴레이션에 포함되어 있는 튜플의 수

Cardinality: 릴레이션 튜플의 개수

틀린 것:

Degree: 차수, 한 릴레이션 안에 있는 attribute(속성)의 수

Attribute: 속성, 개체의 항목, 고유한 이름

Cartesian product: 곱집합

테이블 생성 권한 부여

Grant Create Table To HRD;

데이터베이스 설계 단계 중 물리적 설계 시 고려사항

응답시간

트랜잭션 처리량

저장 공간의 효율화

틀린 것: 스키마의 평가 및 정제

Delete 명령에 대한 설명

테이블의 행을 삭제할 때 사용

SQL을 사용 용도에 따라 분류할 경우 DML에 해당

기본 사용 형식은 ‘Delete From 테이블 Where 조건;’ 이다

틀린 것: Where 조건절이 없는 Delete명령을 수행하면 Drop Table명령을 수행했을 때와 동일한 효과를 얻을 수 있다

E-R 모델 표기법

마름모: 관계

사각형: 개체

선: 관계 – 속성 연결

다중값: 겹친 원

∀: 관계 해석에서 ‘모든 것에 대하여’의 의미를 나타내는 논리, All

∃: 존재한다, Exist

∈: t가 r에 속함(t ∈ r), t; 원소, r; 집합

⊂: a 가 b의 부분집합이다(a ⊂ b), a,b; 집합

→: Y는 X에 함수적 종속이다(X → Y), X; 결정자, Y; 종속자

함수적 종속: X의 값을 알면 Y의 값을 바로 식별할 수 있고, X의 값에 의해 Y의 값이 달라지는 것

관계 대수식

∏이름(σ학과=’교육’(학생))

Select 이름 From 학생 Where 학과=’교육’;

∏: Select

σ: Where

Create Table 기능

속성의 Not null 여부 지정

기본키를 구성하는 속성 지정

Check 제약조건의 정의

틀린 것: 속성 타입 변경

SQL에 관련한 설명

데이터 정의어(Data Define Language, DDL)는 기본 테이블, 뷰 테이블, 또는 인덱스 등을 생성, 변경, 제거하는데 사용되는 명령어이다

Distinct를 활용하여 중복값을 제거할 수 있다

Join을 통해 여러 테이블의 레코드를 조합하여 표현할 수 있다

틀린 것: Revoke 키워드를 사용하여 열 이름을 다시 부여할 수 있다

SQL Union All: 각 쿼리의 모든 결과를 포함한 합집합(중복 제거 안함)

분산 데이터베이스(Distributed Database System)에 대한 설명

분산 DB는 논리적으로는 하나의 시스템에 속하지만 물리적으로는 여러 개의 컴퓨터 사이트에 분산되어 있다

위치 투명성, 중복 투명성, 병행 투명성, 장애 투명성을 목표로 한다

데이터베이스의 설계가 비교적 어렵고, 개발 비용과 처리 비용이 증가한다는 단점이 있다

틀린 것: 분산 데이터베이스 시스템의 주요 구성 요소는 분산 처리기, P2P 시스템, 단일 데이터베이스 등이 있다

(분산 데이터베이스 시스템의 주요 구성 요소는 분산처리기, 분산 데이터베이스, 통신네트워크, 분산 트랜잭션)

분산 데이터베이스 구조: 전역, 분할(단편화), 할당, 지역 스키마

Drop View V\_1 CASCADE;

CASCADE: 테이블이 다른 테이블과 참조 관계가 존재하는 경우 테이블 제거 또는 비활성화 할 수 없다, Cascade는 부모 테이블과 자식 테이블의 제약조건을 비활성화 시키기 위해 사용, 일시적으로 참조관계를 끊을 수 있다

데이터베이스에서 병행제어의 목적

시스템 활용도 최대화

사용자에 대한 응답시간 최소화

데이터베이스 일관성 유지

여러 사용자들의 데이터베이스 공동 사용을 최대화

틀린 것: 데이터베이스 공유 최소화

데이터베이스의 무결성 규정(Integrity Rule)과 관련한 설명

무결성 규정에는 데이터가 만족해야 될 제약조건, 규정을 참조할 때 사용하는 식별자 등의 요소가 포함

무결성 규정의 대상으로는 도메인, 키, 종속성 등이 있다

릴레이션 무결성 규정(Relation Integrity Rule)은 릴레이션을 조작하는 과정에서의 의미적 관계(Semantic Relationship)을 명세한 것

틀린 것: 정식으로 허가 받은 사용자가 아닌 불법 사용자에 의한 갱신으로부터 데이터베이스를 보호하기 위한 규정

무결성 규칙: 데이터베이스에서 무결성 규칙은 데이터 무결성을 지키기 위한 모든 제약사항을 뜻함

카티션 프로덕트(Cartesian Product): 두 테이블 곱집합, 튜플 수는 각 테이블의 곱, 열은 각 테이블의 열을 더함

물리적 데이터베이스 설계에 대한 설명

물리적 설계의 목적은 효율적인 방법으로 데이터를 저장하는 것

트랜잭션 처리량과 응답시간, 디스크 용량 등을 고려해야 한다

저장 레코드의 형식, 순서, 접근 경로와 같은 정보를 사용하여 설계한다

논리적 설계에 기반하여 설계한다

틀린 것: 트랜잭션의 인터페이스를 설계하며, 데이터 타입 및 데이터 타입들 간의 관계로 표현한다(논리적 설계에서 인터페이스를 사용자의 입장에서 편리하도록 설계하는 것)

SQL에서 HAVING을 사용할 수 있는 절: GROUP BY 절, 그룹에 대한 조건을 지정

관계 데이터베이스에 있어서 관계 대수 연산

일반 집합 연산자:

합집합, 교집합, 차집합, 카티션 프로덕트

순수 관계 연산자:

Select, Project, Join, Division

로킹 단위(Locking Granularity)에 대한 설명

로킹 단위가 크면 병행성 수준이 낮아진다(로킹 큼 – 로크 수 적어짐 – 병행성 수준 낮아짐 – 병행 제어 기법 간단)

로킹: 병행 제어 기법의 종류 중 하나

로킹단위: 한번에 로킹할 수 있는 객체의 크기

직렬화 기법, 데이터베이스, 파일, 레코드 등은 로킹 단위가 될 수 있음

로킹 단위가 작아지면, 데이터베이스 공유도 증가, 로킹 오버헤드 증가, 로크의 수 증가, 병행성 수준 높아짐

관계 대수에 대한 설명

관계 대수: 관계형 DB에서 원하는 정보와 그 정보를 검색하기 위해서 어떻게 유도하는가를 기술하는 절차 언어

릴레이션 조작을 위한 연산의 집합으로 피연산자와 결과가 모두 릴레이션이다

일반 집합 연산과 순수 관계 연산으로 구분된다

질의에 대한 해를 구하기 위해 수행해야 할 연산의 순서를 명시한다

틀린 것: 원하는 릴레이션을 정의하는 방법을 제공하며 비절차적 언어이다

Anomaly(이상): 데이터의 중복으로 인하여 관계연산을 처리할 때 예기치 못한 곤란한 현상이 발생하는 것

삽입 이상, 갱신 이상, 삭제 이상

타임스탬프 기법: 동시성 제어를 위한 직렬화 기법으로 트랜잭션 간의 처리 순서를 미리 정하는 방법

로킹 기법: 같은 자원을 액세스하는 다중 트랜잭션 환경에서 DB의 일관성과 무결성을 유지하기 위해 트랜잭션의 순차적 진행을 보장하는 직렬화 기법

**<4과목: 프로그래밍 언어 활용>**

C 언어에서 문자열 처리 함수의 서식과 그 기능의 연결

strlen(s): s의 길이를 구한다

strcpy(s1, s2): s2를 s1으로 복사한다

strrev(s): s를 거꾸로 변환한다

틀린 것: strcmp(s1, s2): s1과 s2를 연결한다. 기능: 두 문자열을 비교한다

&&, || , !

and or 연산자, c언어는 0이 아닌 모든 값은 다 True, 1

IP 프로토콜에서 사용하는 필드와 해당 필드에 대한 설명

Header Length는 IP 프로토콜의 헤더 길이를 32비트 워드 단위로 표시

Time To Live는 송신 호스트가 패킷을 전송하기 전 네트워크에서 생존할 수 있는 시간을 지정한 것

Version Number는 IP 프로토콜의 버전번호를 나타낸다

틀린 것: Packet Length는 IP 헤더를 제외한 패킷 전체의 길이를 나타내며 최대크기는 2^32 -1 비트이다 (헤더 포함, 16비트, 최대값 2^16 - 1)

RIP 라우팅 프로토콜에 대한 설명

경로 선택 metric 홉 카운트(hop count)이다

최단 경로 탐색에 Bellman-Ford 알고리즘을 사용한다

각 라우터는 이웃 라우터들로부터 수신한 정보를 이용하여 라우팅 표를 갱신한다

틀린 것: 라우팅 프로토콜을 IGP와 EGP로 분류했을 때 EGP에 해당한다

HRN 스케줄링: 최소 작업 우선(Shortest-Job-First) 기법의 약점을 보완한 비선점 스케줄링 기법

우선순위 = (대기한 시간 + 서비스를 받을 시간) / 서비스를 받을 시간

FIFO: First-In-First-Out

RR: Round-Robin, 프로세스들 사이의 우선순위가 아닌, 순서대로 일정 시간단위로 CPU를 할당

MQ: 5개의 프로세스 특성에 따라 각각 특정 큐에 할당하여 각 큐에서 독자적인 스케줄링, 큐마다 우선순위

UNIX 운영체제에 관한 특징

하나 이상의 작업에 대하여 백그라운드에서 수행이 가능하다

트리 구조의 파일 시스템을 갖는다

이식성이 높으며 장치간의 호환성이 높다

틀린 것: Multi-User는 지원하지만 Multi-Tasking은 지원하지 않는다

UDP 프로토콜의 특징

비연결형 서비스를 제공

단순한 헤더 구조로 오버헤드가 적다

TCP와 같이 트랜스포트 계층에 존재한다

틀린 것: 주로 주소를 지정하고, 경로를 설정하는 기능을 한다

Python 데이터 타입 중 시퀀스(Sequence) 데이터 타입에 해당, 다양한 데이터타입들을 주어진 순서에 따라 저장할 수 있으나 저장된 내용을 변경할 수 없는 것

튜플(tuple) 타입

응집도: 모듈의 독립성, 모듈 내부 구성요소 간 연관 정도

우연적 < 논리적 < 시간적 < 절차적 < 통신적 < 순차적 < 기능적

우연적(Coincidental): 서로 간에 어떠한 의미 있는 연관 관계도 없는 기능 요소로 구성될 경우

논리적(Logical): 유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들이 한 모듈에서 처리되는 경우

시간적(Temporal): 연관된 기능이라기 보다는 특정 시간에 처리되어야 하는 활동들을 한 모듈에서 처리할 경우

절차적(Procedural): 모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성요소들이 그 기능을 순차적으로 수행할 경우

통신적(Communication): 동일한 입력과 출력을 사용하여 다른 기능을 수행하는 활동들이 모여 있을 경우

순차적(Sequential): 모듈 내에서 한 활동으로부터 나온 출력 값을 다른 활동이 사용할 경우

기능적(Functional): 모듈 내부의 모든 기능이 단일한 목적을 위해 수행되는 경우

IP 프로토콜의 주요 특징

패킷을 분할, 병합하는 기능을 수행하기도 한다

비연결형 서비스를 제공

Best Effort 원칙에 따른 전송 기능을 제공

틀린 것: 체크섬(Checksum) 기능을 데이터 체크섬만 제공한다 (헤더 체크섬만 제공, 데이터 체크섬 제공하지 않음)

LRU: Least-Recently-Used, 가장 최근에 사용하지 않은 페이지를 교체

FIFO: First-In-First-Out, 가장 먼저 메모리에 올라온 페이지를 가장 먼저 내보냄, 교체

OPT: Optimal, 앞으로 가장 오랫동안 사용하지 않을 페이지를 교체, 프로세스가 앞으로 사용할 페이지를 미리 알아야함, 구현 불가능

LFU: Least-Frequently-Used, 참조횟수가 가장 적은 페이지를 교체, 여러 개 LRU처럼 교체

MFU: Most-Frequently-Used, 참조횟수가 가장 많은 페이지를 교체

사용자 수준에서 지원되는 스레드가 커널에서 지원되는 스레드에 비해 가지는 장점

커널 모드로의 전환 없이 스레드 교환이 가능하므로 오버헤드가 줄어든다

틀린 것:

한 프로세스가 운영체제를 호출할 때 전체 프로세스가 대기할 필요가 없으므로 시스템성능을 높일 수 있다

동시에 여러 스레드가 커널에 접근할 수 있으므로 여러 스레드가 시스템 호출을 동시에 사용할 수 있다

각 스레드를 개별적으로 관리할 수 있으므로 스레드의 독립적인 스케줄링이 가능하다

한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 참조하는 경우의 결합도

결합도(Coupling): 모듈 내부가 아닌 외부의 모듈과의 연관도, 모듈 간의 상호의존성

자료 < 스탬프 < 제어 < 외부 < 공통 < 내용

자료(Data): 모듈 간의 인터페이스로 전달되는 파라미터를 통해서만 모듈 간의 상호작용이 일어나는 경우

스탬프(Stamp): 모듈 간의 인터페이스로 배열이나 객체, 구조 등이 전달되는 경우

제어(Control): 모듈이 다른 모듈의 내부 논리 조직을 제어하기 위한 목적으로 제어 신호를 이용하여 통신하는 경우

외부(External): 외부 모듈이 다수의 관련 기능을 가질 때 모듈의 구성요소들이 그 기능을 순차적으로 수행할 경우

공통(Common): 파라미터가 아닌 모듈 밖에서 선언되어 있는 전역변수를 참조하고 전역변수를 갱신하는 식으로 상호작용하는 경우

내용(Content): 다른 모듈 내부에 있는 변수나 기능을 다른 모듈에서 사용하는 경우

모듈화와 관련한 설명

시스템을 모듈로 분할하면 각각의 모듈을 별개로 만들고 수정할 수 있기 때문에 좋은 구조가 된다

모듈 간의 결합도가 약해야 독립적인 모듈이 될 수 있다

모듈 내 구성 요소들 간의 응집도가 강해야 좋은 모듈 설계이다

틀린 것: 응집도는 모듈과 모듈 사이의 상호의존 또는 연관 정도를 의미한다

IP 주소체계와 관련한 설명

IPv6는 주소 자동 설정(Auto Configuration) 기능을 통해 손쉽게 이용자의 단말을 네트워크에 접속시킬 수 있다

IPv4는 호스트 주소를 자동으로 설정하며 유니캐스트(Unicast)를 지원한다

IPv4는 클래스별로 네트워크와 호스트 주소의 길이가 다르다

틀린 것: IPv6의 패킷 헤더는 32 octet의 고정된 길이를 가진다

IPv4:

32비트 주소

유니캐스트 / 멀티캐스트 / 브로드캐스트 사용

IPv6:

128비트 주소

기존 IPv4의 주소 부족 문제를 해결하기 위해 개발

인증성 / 기밀성 / 무결성 지원(=보안성 강화)

유니캐스트 / 애니캐스트 / 멀티캐스트 사용

C 언어 2차원 배열

arr [2][3]

(\*p)[3] = NULL

\*(p[0]) = arr[0]

\*(p[1]) = arr[1]

\*(p[0] + 0) = arr[0][0]

\*(p[1] + 1) = arr[1][1]

\*(p[1] + 2) = arr[1][2]

OSI 7계층

1계층 물리 계층(Physical Layer): 전기적, 기계적, 기능적인 특성을 이용해서 통신 케이블로 데이터를 전송, 비트 단위

통신케이블, 리피터, 허브

2계층 데이터링크 계층(DataLink Layer): 물리 계층을 통해 송수신되는 정보의 오류와 흐름을 관리, 안전한 정보의 전달을 수행할 수 있도록 도와주는 역할, 에러 검출, 재전송, 흐름제어, 맥주소를 가지고 통신, 프레임 단위

브리지, 스위치

3계층 네트워크 계층(Network Layer): 데이터를 목적지까지 가장 안전하고 빠르게 전달하는 기능(라우팅), 경로 선택, 주소를 정하고 경로에 따라 패킷을 전달

4계층 전송 계층(Transport Layer): 통신을 활성화하기 위한 계층, TCP프로토콜 이용, 포트를 열어 응용프로그램들이 전송, 패킷 생성 및 전송

5계층 세션 계층(Session Layer): 데이터가 통신하기 위한 논리적인 연결, TCP/IP 세션을 만들고 없애는 책임

6계층 표현 계층(Presentation Layer): 데이터 표현이 상이한 응용 프로세서의 독립성을 제공, 사용자의 명령어를 완성 및 결과 표현, 포장, 압축, 암호화

7계층 응용 계층(Application Layer): 최종목적지 HTTP, SMTP, POP3, IMAP, Telnet 등 프로토콜, 응용서비스를 수행, 네트워크 소프트웨어 UI부분, 사용자의 입출력 부분

IPv6에 대한 특성

2^128개의 주소를 표현

등급별, 서비스별로 패킷을 구분할 수 있어 품질보장이 용이

확장기능을 통해 보안기능을 제공

틀린 것: 표시방법은 8비트씩 4부분의 10진수로 표시(16비트씩 8부분의 16진수로 표시)

JAVA의 Exception(예외)과 관련한 설명

오동작이나 결과에 악영향을 미칠 수 있는 실행 시간 동안에 발생한 오류

배열의 인덱스가 그 범위를 넘어서는 경우 발생하는 오류

존재하지 않는 파일을 읽으려고 하는 경우에 발생하는 오류

틀린 것: 문법 오류로 인해 발생하는 것(Syntax error, 컴파일 시 에러가 발생하므로 예외조건에 부합하지 않음)

ICMP(Internet Control Message Protocol): TCP/IP 계층 구조에서 IP의 동작 과정에서의 전송 오류가 발생하는 경우에 대비해 오류 정보를 전송하는 목적으로 사용하는 프로토콜

ARP(Address Resolution Protocol): IP 네트워크 상에서 IP주소를 MAC주소로 변환하는 프로토콜

PPP(Point-to-Point Protocol): 데이터링크 계층, 서로 다른 업체의 원격 액세스 소프트웨어들이 시리얼라인 상으로 연결하여 TCP/IP 프로토콜로 통신할 수 있도록 제정된 표준 규약

ECP(Error Checking Protocol): 데이터링크 계층 or 전송 계층

좋은 소프트웨어 설계를 위한 소프트웨어의 모듈간 결합도와 모듈 내 요소 간 응집도에 대한 설명

응집도는 높게, 결합도는 낮게 설계

Semaphore(세마포어): 임계 구역의 접근을 제어하는 상호배제 기법, 공유된 자원의 데이터 혹은 임계 영역 등에 따라 여러 프로세스 또는 스레드가 접근하는 것을 막아줌, 동기화 대상이 하나 이상

wait():

P(S): while S <= 0 do skip;

S := S – 1;

Signal():

V(S): S := S + 1;

Dekker Algorithm: 프로세스가 두 개일 때 상호 배제를 보장하는 최초의 알고리즘, flag와 turn 변수를 사용

Lamport Algorithm: 프로세스 n개의 상호 배제 문제를 해결한 알고리즘, 프로세스에게 고유한 번호를 부여하고 번호를 기준으로 우선순위를 정하여 우선순위가 높은 프로세스가 먼저 임계구역에 진입하도록 구현

Peterson Algorithm: 프로세스가 두 개일 때 상호 배제를 보장, 데커와 유사, 상대에게 진입기회를 양보, 더 간단

소프트웨어 개발에서 모듈이 되기 위한 주요 특징

다른 것들과 구별될 수 있는 독립적인 기능을 가진 단위(Unit)이다

독립적인 컴파일이 가능하다

유일한(Unique) 이름을 가져야 한다

모듈, 다른 프로그램에서 또 다른 모듈을 호출 가능

틀린 것: 다른 모듈에서의 접근이 불가능해야 한다

빈 기억공간의 크기가 20KB, 16KB, 8KB, 40KB일 때 기억장치 배치 전략으로 ‘Best Fit’을 사용하여 17KB의 프로그램을 적재할 경우 내부단편화(프로그램을 적재하고 나서 생기는 빈 공간)의 크기

20 – 17 = 3

16 – 17 = 불가

8 – 17 = 불가

40 – 17 = 23

3KB가 가장 작다

Best Fit: 가능한 공간 중 가장 작은 기억공간을 사용

First Fit: 가능한 공간 중 가장 첫번째 기억공간을 사용

Worst Fit: 가능한 공간 중 가장 큰 기억공간을 사용

개발 환경 구성을 위한 빌드(Build) 도구

Build: 작성된 소스 코드를 실제 기기에서 실행할 수 있는 소프트웨어로 변환하기 위한 과정

Ant: 아파치 재단에서 개발한 자바의 공식적인 빌드 도구

Maven: 아파치 재단에서 Ant 대안으로 개발한 빌드 도구

Gradle: Ant, Maven의 보완으로 개발된 빌드 도구, 안드로이드 스튜디오 주 빌드 도구

틀린 것: Kerberos; 컴퓨터 네트워크 인증 암호화 프로토콜

연산자 우선순위

1순위: ++(후위), --(후위), (), [], .(참조선택), ->(포인터)

2순위: !, ~, +, -, ++(전위), --(전위), (타입), \*(참조), &(주소), sizeof

3순위: \*(곱셈), /, %

4순위: +, -

5순위: <<, >>

6순위: <, <=, >, >=

7순위: ==, !=

8순위: &(비트 and)

9순위: ^(XOR)

10순위: |

11순위: &&(논리 and)

12순위: ||

13순위: ?:(삼항 조건 연산자)

14순위: 대입 연산자 및 복합 대입 연산자(+=, -=, etc)

15순위: ,

a=1, b=2

a < b + 2 && a << 1 <= b, (b+2)

a < 4 && a << 1 <= b, (a<<1)

a < 4 && 2 <= b, (a<4), (2<=b)

1 && 1

1

UNIX 시스템의 Shell의 주요 기능에 대한 설명

사용자 명령을 해석하고 커널로 전달하는 기능 제공

반복적인 명령 프로그램을 만드는 프로그래밍 기능을 제공

초기화 파일을 이용해 사용자 환경을 설정하는 기능을 제공

틀린 것: 쉘 프로그램 실행을 위해 프로세스와 메모리를 관리한다(프로세스, 메모리 관리는 커널의 기능)

오류 제어에 사용되는 자동 반복 요청방식(Automatic Repeat request, ARQ)

Stop-and-Wait ARQ: 송신 측이 하나의 블록을 전송한 후 수신 측에서 에러 발생을 점검, 에러발생 유무 신호 기다림

Go-Back-N ARQ:

**<5과목: 정보시스템 구축관리>**

소프트웨어 개발에서 정보보안 3요소

기밀성: 인가된 사용자에 대해서만 자원 접근이 가능하다

무결성: 인가된 사용자에 대해서만 자원 수정이 가능하며 전송중인 정보는 수정되지 않는다

가용성: 인가된 사용자는 가지고 있는 권한 범위 내에서 언제든 자원 접근이 가능하다

틀린 것: 휘발성: 인가된 사용자가 수행한 데이터는 처리 완료 즉시 폐기되어야 한다

어떤 외부 컴퓨터가 접속되면 접속 인가 여부를 점검해서 인가된 경우에만 접속이 허용되고, 그 반대의 경우에는 거부할 수 있는 접근제어 유틸리티

TCP wrapper

Zing

기기를 키오스크에 갖다 대면 원하는 데이터를 바로 가져올 수 있는 기술로 10츠 이내 근접거리에서 기가급 속도로 데이터 전송이 가능한 초고속 근접무선통신(NFC: Near Field Communication)

취약점 관리를 위한 응용 프로그램의 보안 설정

실행 프로세스 권한 설정

운영체제의 접근 제한

운영체제의 정보 수집 제한

틀린 것: 서버 관리실 출입 통제

소프트웨어 개발 프레임워크와 관련한 설명

반제품 상태의 제품을 토대로 도메인별로 필요한 서비스 컴포넌트를 사용하여 재사용성 확대와 성능을 보장받을 수 있게 하는 개발 소프트웨어

설계 관점에 개발 방식을 패턴화 시키기 위한 노력의 결과물인 소프트웨어 디자인 패턴을 반제품 소프트웨어 상태로 집적화시킨 것으로 볼 수 있다

프레임워크의 동작 원리를 그 제어 흐름의 일반적인 프로그램 흐름과 반대로 동작한다고 해서 IoC(Inversion of Control)이라고 설명하기도 한다

틀린 것: 라이브러리와는 달리 사용자 코드에서 프레임워크를 호출해서 사용하고, 그에 대한 제어도 사용자 코드가 가지는 방식

클라우드 기반 HSM(Cloud-based Hardware Security Module)에 대한 설명

클라우드(데이터센터) 기반 암호화 키 생성, 처리, 저장 등을 하는 보안기기

국내에서는 공인인증제의 폐지와 전자서명법 개정을 추진하면서 클라우드 HSM용어가 자주 등장

클라우드에 인증서를 저장하므로 기존 HSM 기기나 휴대폰에 인증서를 저장해 다닐 필요가 없다

틀린 것: HW가 아닌 SW적으로만 구현되기 때문에 SW식 암호기술에 내재된 보안 취약점을 해결할 수 없다는 단점

Mesh Network

다른 국을 향하는 호출이 중계에 의하지 않고 직접 접속되는 그물 모양의 네트워크

통신량이 많은 비교적 소수의 국 사이에 구성될 경우 경제적이며 간편하지만, 다수의 국 사이에는 회선이 세분화되어 비경제적일 수도 있다

해당 형태의 무선 네트워크의 경우 대용량을 빠르고 안전하게 전달할 수 있어 행사장이나 군 등에서 많이 활용된다

물리적 위협으로 인한 문제

화재, 홍수 등 천재지변으로 인한 위협

하드웨어 파손, 고장으로 인한 장애

방화, 테러로 인한 하드웨어와 기록장치를 물리적으로 파괴하는 행위

틀린 것: 방화벽 설정의 잘못된 조작으로 인한 네트워크, 서버 보안 위협

악성코드의 유형 중 다른 컴퓨터의 취약점을 이용하여 스스로 전파하거나 메일로 전파되며 스스로를 증식하는 것

Worm

틀린 것:

Rogue Ware: 가짜 백신 프로그램, 거짓 내용으로 사용자를 속여 결제하게끔 유도

Adware: 팝업 광고 표시 프로그램

Reflection Attack: 동일한 프로토콜을 양방향으로 사용하는 시도 응답 인증 시스템을 공격하는 방법

Ping of Death: 시스템 공격 기법 중 하나로 허용범위 이상의 ICMP 패킷을 전송하여 대상 시스템의 네트워크를 마비

Session Hijacking: 공격 대상이 이미 시스템에 접속되어 세션에 연결되어 있는 상태를 가로채기 하는 공격

Piggyback attack: 다른 사용자의 합법적 연결에서 비활성 간격을 통해 시스템에 액세스하는 활성형태의 도청

XSS: Cross Site Scripting, 상대방의 브라우저에 스크립트가 실행되도록 해 사용자의 세션을 가로채거나, 웹사이트를 변조하거나, 악의적 콘텐츠를 삽입하거나, 피싱 공격을 진행하는 것

소프트웨어 개발 프레임워크

개발해야 할 애플리케이션의 일부분이 이미 내장된 클래스 라이브러리로 구현이 되어 있다

따라서 그 기반이 되는 이미 존재하는 부분을 확장 및 이용하는 것

Java 기반의 대표적인 소프트웨어로는 스프링(Spring)이 있다

대칭 암호 알고리즘, 비대칭 암호 알고리즘에 대한 설명

대칭 암호 알고리즘은 비교적 실행 속도가 빠르기 때문에 다양한 암호의 핵심 함수로 사용될 수 있다

비대칭 암호 알고리즘은 자신만이 보관하는 비밀키를 이용하여 인증, 전자서명 등에 적용이 가능하다

대표적인 대칭키 암호 알고리즘으로는 AES, IDEA 등이 있다

틀린 것: 대칭 암호 알고리즘은 비밀키 전달을 위한 키 교환이 필요하지 않아 암호화 및 복호화의 속도가 빠르다

월별(man-month) 생산성 측정

두 명의 개발자 5개월 10000라인 코드 개발 -> 10000 / (5 \* 2)

RBAC: Role-Based Access Control, 조직 내에서 직무, 직책 등 개인의 역할에 따라 권한을 결정, 부여하는 접근 정책

COCOMO(Constructive Cost Model) 모형의 특징

프로젝트를 완성하는데 필요한 man-month로 산정 결과를 나타낼 수 있다

보헴(Boehm)이 제안한 것으로 원시코드 라인 수에 의한 비용 산정 기법이다

비교적 작은 규모의 프로젝트 기록을 통계 분석하여 얻은 결과를 반영한 모델이며 중소 규모 소프트웨어 프로젝트 비용 추정에 적합하다

틀린 것: 프로젝트 개발 유형에 따라 object, dynamic, function의 3가지 모드로 구분한다

각 사용자 인증의 유형에 대한 설명

지식: 주체는 ‘그가 알고 있는 것’을 보여주며 예시로는 패스워드, PIN 등이 있다

소유: 주체는 ‘그가 가지고 있는 것’을 보여주며 예시로는 토큰, 스마트카드 등이 있다

행위: 주체는 ‘그가 하는 것’을 보여주며 예시로는 서명, 움직임, 음성 등이 있다

틀린 것: 존재: 주체는 ‘그를 대체하는 것’을 보여주며 예시로는 패턴, QR 등이 있다(‘그를 나타내는 것’, 홍채, 지문)

시스템의 사용자가 로그인하여 명령을 내리는 과정에 대한 시스템의 동작

Authentication(인증): 자신의 신원(Identity)을 시스템에 증명하는 과정, 아이디와 패스워드를 입력하는 과정이 가장 일반적인 예시

틀린 것: Authorization(허가)

SDN(Software Defined Networking)

네트워크를 제어부, 데이터 전달부로 분리하여 네트워크 관리자가 보다 효율적으로 네트워크를 제어, 관리할 수 있는 기술

기존의 라우터, 스위치 등과 같이 하드웨어에 의존하는 네트워크 체계에서 안정성, 속도, 보안 등을 소프트웨어로 제어, 관리하기 위해 개발됨

네트워크 장비의 펌웨어 업그레이드를 통해 사용자의 직접적인 데이터 전송 경로 관리가 가능하고, 기존 네트워크에는 영향을 주지 않으면서 특정 서비스의 전송경로 수정을 통하여 인터넷 상에서 발생하는 문제를 처리할 수 있음

프로젝트 일정 관리 시 사용하는 PERT 차트에 대한 설명

각 작업들이 언제 시작하고 언제 종료되는지에 대한 일정을 막대 도표를 이용하여 표시

시간선(Time-line) 차트라고도 한다

수평 막대의 길이는 각 작업의 기간을 나타낸다

틀린 것: 작업들 간의 상호 관련성, 결정경로, 경계시간, 자원할당 등을 제시한다

소프트웨어 생명주기 모델 중 나선형(Spiral Model)과 관련한 설명

소프트웨어 개발 프로세스를 위험 관리(Risk Management) 측면에서 본 모델

시스템을 여러 부분으로 나누어 여러 번의 개발 주기를 거치면서 시스템이 완성

요구사항이나 아키텍처를 이해하기 어렵다거나 중심이 되는 기술에 문제가 있는 경우 적합한 모델

틀린 것: 위험 분석(Risk Analysis)은 반복적인 개발 진행 후 주기의 마지막 단계에서 최종적으로 한 번 수행

고 가용성 솔루션(HACMP): 각 시스템 간에 공유 디스크를 중심으로 클러스터링으로 엮여 다수의 시스템을 동시에 연결할 수 있다

조직, 기업의 기간 업무 서버 등의 안정성을 높이기 위해 사용될 수 있다

여러 방식으로 구현, 2개의 서버를 연결하는 것으로 2개의 시스템이 각각 업무를 수행하는 구현 방식이 널리 사용

점대점 연결 방식(Point-to-Point Mode): 네트워크에 있어 물리적으로는 중계 장치를 통과하지 않고 한 지점에서 다른 지점으로 직접 가는 채널, 두 스테이션간을 별도의 회선을 사용하여 1 대 1로 연결, 회선 구성이 간단하고 대용량 전송에 유리, 별도의 회선과 포트에 따른 높은 설치비용

스턱스넷(Stuxnet): 웜 바이러스, 윈도우를 통해 감염, 지맨스산업의 SW 및 장비를 공격

루팅(Rooting): 모바일 기기에서 구동되는 안드로이드 운영체제에서 최상위 권한(루트 권한)을 얻음으로 해당 기기의 생산자 또는 판매자 측에서 걸어 놓은 제약을 해제하는 행위

Switch Jamming: 위조된 매체 접근 제어(MAC) 주소를 지속적으로 네트워크로 흘려보내 스위치 MAC 주소 테이블의 저장 기능을 혼란시켜 더미 허브(Dummy Hub)처럼 작동하게 하는 공격

Parsing: 하나의 프로그램을 런타임 환경이 실제로 실행할 수 있는 내부 포맷으로 분석하고 변환하는 것

LAN Tapping: LAN신호를 직접 자신에게 끌어오는 방식의 공격

FTP(SYN) Flooding: TCP의 3 Way Handshake 취약점을 이용한 DoS 공격으로 다량의 SYN패킷을 보내 백로그큐를 가득 채우는 공격, FTP 프로토콜을 사용한 서버에 다량의 SYN 패킷을 보내 마비시키는 것

DAS(Direct-Attached Storage, 직접 연결 저장장치): 하드디스크와 같은 데이터 저장장치를 호스트 버스 어댑터에 직접 연결하는 방식, 저장장치와 호스트 기기 사이에 네트워크 디바이스 없이 직접 연결하는 방식으로 구성

NAS(Network-Attached Storage, 네트워크 연결 저장장치)

NFC: 근거리 무선 통신

취약점 관리를 위해 일반적으로 수행하는 작업

무결성 검사

응용 프로그램의 보안 설정 및 패치(Patch) 적용

불필요한 서비스 및 악성 프로그램의 확인과 제거

틀린 것: 중단 프로세스 및 닫힌 포트 위주로 확인(활성화 프로세스와 열린 포트를 중심으로 확인)

소프트웨어 생명주기 모델 중 V 모델과 관련한 설명

Perry에 의해 제안, 세부적인 테스트 과정으로 구성되어 신뢰도 높은 시스템을 개발하는데 효과적

개발 작업과 검증 작업 사이의 관계를 명확히 들어내 놓은 폭포수 모델의 변형, 코딩 단계에서 위쪽으로 꺾여 V모양

폭포수 모델이 산출물 중심이라면 V 모델은 작업과 결과의 검증에 초점을 둔다

검증(Verification) 단계: 요구사항 분석 – 시스템 설계 – 아키텍쳐 설계 – 모듈 설계

틀린 것: 요구 분석 및 설계 단계를 거치지 않으며 항상 통합테스트를 중심으로 V 형태를 이룬다

블루투스 공격과 해당 공격에 대한 설명

블루프린팅(BluePrinting): 블루투스 공격 장치의 검색 활동을 의미

블루버그(BlueBug): 블루투스 장비 사이의 취약한 연결 관리를 악용한 공격

블루스나프(BlueSnarf): 블루투스의 취약점을 활용하여 장비의 파일에 접근하는 공격, OPP를 사용하여 정보를 열람

블루재킹(BlueJacking): 블루투스를 이용해 스팸처럼 명함을 익명으로 퍼뜨리는 것

OPP(Obex Push Protocol): 블루투스 장치끼리 인증 없이 정보를 간편하게 교환하기 위하여 개발됨

DoS(Denial of Service) 공격과 관련한 내용

Ping of Death 공격은 정상 크기보다 큰 ICMP 패킷을 작은 조각(Fragment)으로 쪼개어 공격 대상이 조각화 된 패킷을 처리하게 만드는 공격 방법

SYN Flooding은 존재하지 않는 클라이언트가 서버별로 한정된 접속 가능 공간에 접속한 것처럼 속여 다른 사용자가 서비스를 이용하지 못하게 하는 것

Land 공격은 패킷 전송 시 출발지 IP주소와 목적지 IP주소 값을 똑같이 만들어서 공격 대상에게 보내는 공격 방법

Smurf 공격은 브로드캐스트를 활용하여 공격 대상이 네트워크의 임의의 시스템에 패킷을 보내게 만드는 공격

Honeypot: 1990년대 David Clock이 처음 제안, 비정상적인 접근의 탐지를 위해 의도적으로 설치해 둔 시스템, 침입자를 속여 실제 공격당하는 것처럼 보여줌으로써 크래커를 추적 및 공격기법의 정보를 수집하는 역할, 쉽게 공격자에게 노출되어야 하며 쉽게 공격이 가능한 것처럼 취약해 보여야 한다

Hadoop: 오픈 소스를 기반으로 한 분산 컴퓨팅 플랫폼, 대형 스토리지, 빅데이터 관련

MapReduce: 대용량 데이터를 분산처리 하기위해 구글이 개발한 프로그래밍 모델, 임의의 순서로 분산처리 후 합침

Apache: W3C에서 사용하고 아파치 소프트웨어 재단에서 관리 및 운영하는 서버용 오픈소스 소프트웨어

Docker: 소프트웨어 컨테이너 안에 응용프로그램들을 배치시키는 일을 자동화 해 주는 오픈소스 소프트웨어

StackGuard: Stack 상에 일정한 주소 번지에 프로그램이 선언한 Canary를 심어 스택이 변조된 경우 Canary를 체크하여 프로그램을 비정상적으로 종료시키는 기법

Cipher Container: 자바에서 암호화 복호화 기능을 제공하는 컨테이너

Scytale: 암호화 기법으로 단순하게 문자열의 위치를 바꾸는 방법

간트 차트(Gantt Chart)에 대한 설명

프로젝트를 이루는 소작업별로 언제 시작되고 언제 끝나야 하는지를 한 눈에 볼 수 있도록 도와준다

자원 배치 계획에 유용하게 사용된다

CPM 네트워크로부터 만드는 것이 가능하다

틀린 것: 수평 막대의 길이는 각 작업에 필요한 인원수를 나타낸다(기간을 나타낸다)

Scrapy: Python 기반의 웹 크롤링 프레임워크, 가볍고 빠르고 확장성이 좋음

Li-fi: 스펙트럼의 빛을 이용한 5세대 이동 통신 기술

SBAS(위성항법보강시스템): GPS의 오차를 보정해 신뢰성과 안정성을 높인 기법

Secure 코딩에서 입력 데이터의 보안 약점과 관련한 설명

SQL 삽입: 사용자의 입력값 등 외부 입력값이 SQL 쿼리에 삽입되어 공격

크로스사이트 스크립트: 검증되지 않은 외부 입력값에 의해 브라우저에서 악의적인 코드가 실행

운영체제 명령어 삽입: 운영체제 명령어 파라미터 입력값이 적절한 사전검증을 거치지 않고 사용되어 공격자가 운영체제 명령어를 조작

틀린 것: 자원 삽입: 사용자가 내부 입력값을 통해 시스템 내에 사용이 불가능한 자원을 지속적으로 입력함으로써 시스템에 과부하 발생

(자원을 조작할 수 있는 문자열을 삽입하여 시스템이 보호하는 자원에 임의로 접근할 수 있는 취약점)

Windows 파일 시스템인 FAT과 비교했을 때의 NTFS의 특징

대용량 볼륨에 효율적

자동 압축 및 안정성

저용량 볼륨에서의 속도 저하

틀린 것: 보안에 취약

DES: 64비트 암호화 알고리즘, 7비트마다 오류 검출을 위한 정보가 1비트씩 들어가기 때문에 실질적으로 56 비트이다. 추가적으로 AES 암호화 알고리즘은 AES-128, AES-192, AES-256으로 나뉘며 숫자는 비트수이다.

리눅스에서 생성된 파일 권한이 644일 경우 umask 값

022

666에서 파일 권한 644를 빼면 022가 된다

wtmp: 리눅스 시스템에서 사용자의 성공한 로그인/아웃 정보 기록, 시스템의 종료/시작 시간 기록, var/log/wtmp

utmp: 현재 로그인 사용자의 상태 정보를 담고 있는 로그 파일

btmp: 실패한 로그인 정보를 담고 있는 로그 파일

last log: 마지막으로 성공한 로그인 정보를 담고 있는 로그 파일

상향식 비용 산정 기법 중 LOC(원시 코드 라인 수) 기법에서 예측치를 구하기 위해 사용하는 항목

낙관치

기대치

비관치

틀린 것: 모형치

DPI(Deep Packet Inspection): OSI 7 Layer 전 계층의 프로토콜과 패킷 내부의 컨텐츠를 파악하여 침입 시도, 해킹 등을 탐지하고 트래픽을 조정하기 위한 패킷 분석 기술, 네트워크에서 전송되는 패킷의 헤더와 페이로드 내 정보를 분석하는 컨텐츠 내용 분석 기술, 네트워크 보안, 관리, 컨텐츠 관리 등이 목적

PLCP(Physical Layer Convergence Procedure): 물리 계층 수렴 처리, 논리적인 802.11 MAC 부계층과 물리적인 특성을 연결하는 역할, 802.11 MAC 부계층이 물리적 특성에 관계없이 동작하도록 함

Traffic Distributor: 네트워크 통신 간에 트래픽을 분배해주는 솔루션

소프트웨어 개발 방법론의 테일러링(Tailoring)과 관련한 설명

프로젝트에 최적화된 개발 방법론을 적용하기 위해 절차, 산출물 등을 적절히 변경하는 활동

관리 측면에서의 목적 중 하나는 최단기간에 안정적인 프로젝트 진행을 위한 사전 위험을 식별, 제거하는 것

기술적 측면에서의 목적 중 하나는 프로젝트에 최적화된 기술 요소를 도입하여 프로젝트 특성에 맞는 최적의 기법과 도구를 사용하는 것이다

틀린 것: 프로젝트 수행 시 예상되는 변화를 배제하고 신속히 진행하여야 한다

Tailoring: 프로젝트 상황 특성에 맞게 정의된 소프트웨어 개발 방법론 절차, 사용 기법 등을 수정 및 보완하는 작업