**<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

탐색 효율이 좋고 탐색 시간이 적게 소요된다

검색할 데이터가 정렬되어 있어야 한다

비교 횟수를 거듭할 때 마다 검색 대상이 되는 데이터의 수가 절반으로 줄어든다

틀린 것: 피보나치 수열에 따라 다음에 비교할 대상을 선정, 검색한다(피보나치 검색)

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

유지 보수 단계 뿐만 아니라 개발 단계에도 적용할 수 있다

소프트웨어 개발의 전체 비용을 줄이고, 개발 과정의 여러 방해 요인이 최소화되도록 보증하는 것을 목적

형상 관리 도구: GIT, CVS, Subversion 등

틀린 것: Ant, Maven, Gradle(빌드 자동화 도구)

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, 프로젝트 계획, 분석서, 설계서, 프로그램, 테스트 케이스 등

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

구성 요소:

식별자: 항목 식별자, 일련번호

테스트 항목: 테스트 대상 – 모듈 또는 기능

입력 명세: 입력 데이터 또는 테스트 조건

출력 명세: 테스트 케이스 수행 시 예상되는 출력 결과

환경 설정: 필요한 하드웨어나 소프트웨어의 환경

특수 절차 요구: 테스트 케이스 수행 시 특별히 요구되는 절차

의존성 기술: 테스트 케이스 간의 의존성

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)

구성 요소: 컨텐츠 분배자(Contents Distributor), 패키저(Packager), 클리어링 하우스(Clearing House), DRM Controller

컨텐츠 분배자: 암호화된 컨텐츠를 유통하는 곳/사람

패키저: 컨텐츠를 메타데이터와 함께 배포 가능한 형태로 묶어 암호화

클리어링 하우스: 키 관리 및 라이센스 발급 관리, 중개 수행, 권한 부여 시스템

DRM 컨트롤러: 배포된 컨텐츠의 이용권한을 통제

기술 요소: 암호화, 키관리, 암호화 파일 생성, 식별 기술, 저작권 표현, 정책관리, 크랙방지, 인증

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

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, 다른 방향으로 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)

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

알고리즘 오류에 따른 원치 않는 결과 검출

탈출구가 없는 반복문의 사용 검출

틀린 계산 수식에 의한 잘못된 결과 검출

단위 테스트는 테스트 코드가 독립적이어야 함으로 모듈간의 상호작용은 해당되지 않는다(통합 테스트)

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, 공통된 코드 사용

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

단순성: 코드를 단순하게 작성, 프로그램을 최소 단위로 분리해 작업을 한 번에 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, 수식 계산 및 수식 표기법 등에 응용된다

입출력이 한쪽 끝으로만 제한된 리스트

LIFO 구조

더 이상 삭제할 데이터가 없는 상태에서 데이터를 삭제하면 언더플로가 발생

틀린 것: Head(Front)와 Tail(Rear)의 2개 포인터를 갖고 있다(Queue)

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 패키징

모듈별로 생성한 실행 파일들을 하나로 합쳐서 설치 파일을 만드는 것

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, 레코드의 많은 자료 이동을 없애고 하나의 파일을 부분적으로 나누어 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, 이미 순서화된 파일에 새로운 하나의 레코드를 순서에 맞게 삽입시켜 정렬, (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))

Radix Sort: 레코드의 키 값을 분석, 같은 값끼리 순서에 맞는 버킷에 분배, 버킷의 순서대로 레코드를 꺼내어 정렬

Shell Sort: 임의의 레코드 키와 매개변수(h)값 만큼 떨어진 곳의 레코드 키를 비교하여 서로 교환해가면서 정렬, 입력 파일을 매개변수값으로 서브 파일을 구성하고 각 서브파일을 삽입 정렬 방식으로 순서 배열 과정 반복 정렬

Heap Sort: 전이진 트리를 이용한 정렬

2-Way 합병 정렬: 이미 정렬되어 있는 두 개의 파일을 한 개의 파일로 합병하는 정렬 방식

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

O(nlog₂n): 선형 로그 복잡도; 로그, 변수에 비례 -> 퀵정렬, 병합정렬

O(n^2): 제곱에 비례 -> 버블정렬, 삽입정렬, 선택정렬(정렬된 데이터; 버블,삽입정렬 O(n))

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, 선형 순차적 모델, 고전적 생명주기 모형, 분석 – 설계 – 구현 – 테스트 – 유지보수 단계, 가장 오래된 모형, 요구사항의 변경이 어려움

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, 메모리와 실행 시간 같은 시스템 리소스의 최소 사용

Reliability: 신뢰성, 정해진 상황에서 언제든지 필요한 기능을 수행할 수 있는 시스템의 능력

Integrity: 무결성, 시스템이 프로그램이나 데이터에 대한 허용되지 않거나 잘못된 접근을 막는 정도, 무결성의 기본 개념에는 데이터의 적절한 접근을 보장할 뿐만 아니라 권한이 없는 사용자의 접근 제한 기능이 포함

Usability: 사용 용이성/유용성, 소프트웨어를 쉽게 사용할 수 있는 정도, 사용자가 시스템을 배우고 사용하는데 있어서의 용이함

Adaptability: 적응성, 시스템을 변경하지 않고 설계된 환경 외 다른 응용 분야나 환경에서도 사용될 수 있는 정도

Accuracy: 정밀성, 구성된 시스템에 오류가 없는 정도

Robustness: 견고성, 시스템이 잘못된 입력이나 악조건에서도 기능을 계속해서 수행할 수 있는 정도

Correctness: 정확성, 사용자의 요구사항을 충족시키는 정도, 시스템의 사양과 설계, 구현에 있어서 오류가 없는 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

힙 정렬(Heap Sort)에 대한 설명

정렬할 입력 레코드들로 힙을 구성하고 가장 큰 키 값을 갖는 루트 노드를 제거하는 과정을 반복하여 정렬

평균 수행 시간은 O(nlog₂n)이다

완전 이진트리(Complete Binary Tree)로 입력자료의 레코드를 구성한다

틀린 것: 최악의 수행시간은 O(2n^4)이다(O(nlogn))

살충제 패러독스: 동일한 테스트 케이스로 반복 실행하면 더 이상 새로운 결함을 발견할 수 없으므로 주기적으로 테스트 케이스를 점검하고 개선해야 한다

오류 부재의 궤변: 사용자의 요구사항을 만족하지 못한다면 오류를 발견하고 제거해도 품질이 높다고 말할 수 없다

결함 집중: 파레토 법칙이 좌우, 애플리케이션 결함의 대부분은 소수의 특정한 모듈에 집중되어 존재, 결함은 발생한 모듈에서 계속 추가로 발생할 가능성이 높다

소프트웨어 버전 등록 관련 주요 용어

저장소: 최신 버전의 파일들과 변경 내역에 대한 정보들이 저장되어 있는 곳

가져오기(Import): 버전 관리가 되고 있지 않은 아무것도 없는 저장소에 처음으로 파일을 복사

체크아웃(Check out): 프로그램을 수정하기 위해 저장소에서 파일을 받아옴, 소스 파일과 함께 버전 관리를 위한 파일들을 받아옴

체크인(Check in): 체크아웃한 파일의 수정을 완료한 후 저장소의 파일을 새로운 버전으로 갱신

커밋(Commit): 체크인을 수행할 때 이전에 갱신된 내용이 있는 경우에는 충돌을 알리고 diff 도구를 이용해 수정한 후 갱신 완료

동기화: 저장소에 있는 최신 버전으로 자신의 작업 공간을 동기화

틀린 것:

형상 감사(Configuration Audit): 기준선의 무결성을 평가하기 위해 확인, 검증, 검열 과정을 통해 공식적으로 승인하는 과정

롤백(Rollback): 데이터베이스에서 업데이트에 오류가 발생할 때, 이전 상태로 되돌리는 것, 후진 복귀

소프트웨어 테스트

블랙박스 테스트: 내부 구조나 작동 원리를 모르는 상태에서 입력에 따른 출력 결과를 테스트

* 블랙박스 테스트는 프로그램의 구조를 고려하지 않는다
* 비정상적인 자료를 입력해도 오류 처리를 수행하지 않는 경우 검출
* 정상적인 자료를 입력해도 요구된 기능이 제대로 수행되지 않는 경우 검출
* 경계값을 입력할 경우 요구된 출력 결과가 나오지 않는 경우 검출
* 틀린 것: 반복 조건을 만족하는데도 루프 내의 문장이 수행되지 않는 경우(화이트박스 테스트)
* 검사 기법: 동치(동등) 분할 검사, 경계값 분석, 원인-효과 검사, 오류 예측 검사, 비교 검사

화이트박스 테스트: 내부 구조와 동작을 검사하는 테스트 방식, 소프트웨어의 내부 소스를 테스트

* 화이트박스 테스트는 모듈의 논리적인 구조를 체계적으로 점검할 수 있다
* 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
* 화이트 박스 테스트에서 기본 경로는 싸이클을 최대한 한 번만 지나야 한다
* 화이트박스 테스트는 내부의 논리적인 모든 경로를 테스트해 테스트 케이스를 설계함
* 틀린 것: 프로그램의 구조를 고려하지 않기 때문에 테스트케이스는 프로그램 또는 모듈의 요구, 명세를 기초로 결정
* 검사 기법: 기초 경로 검사, 조건 검사, 루프 검사, 데이터 흐름 검사, 문장 검증, 선택 검증

테스트 케이스에는 일반적으로 시험 조건, 테스트 데이터, 예상 결과가 포함되어야 한다

틀린 것: 화이트박스 테스트에서 기본 경로(Basis Path)란 흐름 그래프의 시작 노드에서 종료 노드까지의 서로 독립된 경로로 싸이클을 허용하지 않는 경로를 말한다

성능 테스트 도구(Performance Test Tools): 애플리케이션의 처리량, 응답시간, 경과시간, 자원 사용률 등 성능 목표 달성 여부 확인

단위 테스트 종류: 명세 기반 테스트, 구조 기반 테스트

명세 기반 테스트: 주어진 명세를 빠짐없이 테스트 케이스로 구현하고 있는지 확인하는 테스트

구조 기반 테스트: 프로그램 내부 구조 및 복잡도를 검증하는 화이트박스 테스트 시행, 제어 흐름과 조건 결정 목적

분산 저장소 방식: 하나의 원격 저장소와 개발자 PC의 로컬 저장소에 저장, 자신의 로컬로 복사해 작업, 로컬 저장소에서 우선 반영(Commit), 그 후 원격 저장소에 반영(Push), 문제가 생겨도 로컬의 자료를 이용해 작업 가능, 로컬 저장소에서 작업 및 처리 속도 빠름

공유 폴더 방식: 로컬 컴퓨터의 공유 폴더에 저장되어 관리, 공유 폴더의 파일을 자기 PC로 복사 후 이상 유무 확인

클라이언트/서버 방식: 중앙 시스템(서버)에 저장되어 관리, 개발자별로 자신의 PC(클라이언트)로 복사, 모든 버전 관리는 서버에서 수행, 서버에 문제가 생기면 다른 개발자 협업 및 버전 관리 작업 중단

EAI(Enterprise Application Integration): 기업 내 상호 연동이 가능하게 해주는 솔루션, 비즈니스 간 통합 및 연계성을 증대시켜 각 시스템 간의 확장성을 높여줌

구축 유형: Point-to-Point, Hub & Spoke, Message Bus(ESB), Hybrid

Point-to-Point: 가장 기본적인 애플리케이션 통합 방식으로, 애플리케이션을 1: 1로 연결, 변경 및 재사용이 어려움

Hub & Spoke: 단일 접점인 허브를 통해 데이터를 전송하는 중앙 집중형 방식, 확장 및 유지 보수가 용이하지만 허브 장애 발생 시 시스템 전체에 영향

Message Bus: 애플리케이션 사이에 미들웨어를 두어 처리하는 방식, 확장성이 뛰어나며 대용량 처리가 가능

Hybrid: 그룹 내에서는 Hub & Spoke 방식, 그룹 간에는 Message 방식 사용, 데이터 병목 현상 최소화

인터페이스 구현 검증 도구

xUnit, STAF, Fitnesse, NTAF, Selenium, Watir

틀린 것: Foxbase; 프로그래밍 언어

공학적으로 잘된 소프트웨어(Well Engineered Software)

소프트웨어는 유지보수가 용이해야 한다

소프트웨어는 신뢰성이 높아야 한다

소프트웨어는 충분한 테스팅을 거쳐야 한다

틀린 것: 소프트웨어는 사용자 수준에 무관하게 일관된 인터페이스를 제공해야 한다

소프트웨어의 최종 소비자는 End User이기 때문에 End User의 요구사항을 최대한으로 반영해서 소프트웨어를 개발해야 한다, 소프트웨어의 인터페이스는 End User의 수준에 맞게 직관적이고 사용하기 쉽게 설계, 개발되어야 한다

테스트와 디버그의 목적:

Coding – Testing - Error 발생(==Bug 발견) – Error 수정(==Debugging) – Retesting

테스트는 오류를 찾는 작업이고 디버깅은 오류를 수정하는 작업이다

스택을 이용한 연산

재귀 호출, 후위 표현(Post-Fix Expression)의 연산, 깊이 우선 탐색

선택정렬: 큐를 이용해서 정렬, 가장 작은 값을 위치가 정렬되어지지 않은 최초의 값과 바꿈, 스택은 중간값에 대해 접근할 수 없으므로 선택정렬을 할 수 없음

구현 단계에서의 작업 철차

코딩 작업 계획 – 코딩 – 컴파일 – 코드 테스트

Tree: 그래프의 특수한 형태, 노드(Node)와 선분(Branch)으로 되어있고, 정점 사이에 사이클(Cycle)이 형성되어 있지 않으며, 자료 사이의 관계성이 계층 형식으로 나타나는 비선형 구조

UI 요소

체크박스: 여러 개의 선택 상황에서 1개 이상의 값을 선택할 수 있는 버튼

라디오 버튼: 여러 항목 중 하나만 선택할 수 있는 버튼

텍스트 박스: 사용자가 데이터를 입력하고 수정할 수 있는 상자

토글 버튼: On/Off와 같이 둘 중 하나의 값을 선택하는 버튼

신뢰성(Reliability): 소프트웨어가 요구된 기능을 정확하고 일관되게 오류없이 수행할 수 있는 정도

유지보수성(Maintainability): 환경 변화 또는 새로운 요구사항이 발생했을 때 SW를 개선, 확장할 수 있는 정도

가시성(Visibility): 대상을 확인할 수 있는 정도

재사용성(Reusability): 소프트웨어의 일부분을 다른 시스템에서 사용할 수 있는 정도

소프트웨어 설치 매뉴얼

소프트웨어 개요, 설치 관련 파일, 설치 아이콘, 프로그램 삭제, 관련 추가 정보

Chief Programmer Team: 효율성을 제고하기 위해 능력과 경험이 풍부한 책임 프로그램 작성자 중심 구성 개발팀

Hashing Function

제산법(Division), 제곱법(Mid-Square), 중첩법(Folding), 숫자분석법(Digit Analysis), 기수변환법, 무작위 방법

인수 테스트: 계약 인수 테스트, 규정 인수 테스트, 알파 테스트, 베타 테스트

알파 테스트: 개발자의 장소에서 사용자가 개발자 앞에서 행하는 테스트

베타 테스트: 선정된 최종 사용자가 여러 명의 사용자 앞에서 행하는 테스트 기법