20. 통합 구현
단위 모듈 구현
단위 모듈

소프트웨어 구현에 필요한 다양한 동작 중 한 가지 동작을 수행하는 기능을 모듈로 구현한 것을 의미한다.
사용자 또는 다른 모듈로부터 값을 전달받아 시작되는 작은 프로그램이다.
독립적인 컴파일이 가능하며, 다른 모듈에 호출되거나 삽입될 수 있다.
두 개의 단위 모듈이 합쳐지면 두 개의 기능들을 같은 모듈로 구현할 수 있다.
종류 : 화면, DB 접근, 인터페이스, 비즈니스 트랜잭션, 데이터 암호화 등
단위 기능 명세서

큰 규모의 시스템을 분해하여 단위 기능별로 계층적으로 구조화하고, 단순하게 추상화한 문서이다.
모듈화의 원리

소프트웨어 개발에 있어 기능을 나누고 추상화하여 소프트웨어의 성능을 향상시키고 유지보수를 효과적으로 구현하기 위한 기법을 의미한다.
종류
분할과 지배(Dibide Conquer) : 복잡한 문제를 분해, 모듈 단위로 문제를 해결한다.
정보 은폐(Information Hiding) : 어렵거나 변경 가능성이 있는 모듈을 타 모듈로부터 은폐시킨다.
자료 추상화(Data Abstraction) : 함수 내에 자료 구조의 표현 명세를 은폐, 자료와 자료에 적용 가능한 오퍼레이션을 함께 정의한다.
모듈의 독립성(Module Independence) : 낮은 결합도, 높은 응집도를 갖도록 한다.
단위 모듈 테스트

프로그램의 단위 기능을 구현하는 모듈이 정해진 기능을 정확히 수행하는지 검증하는 것이다.
화이트박스 테스트와 블랙박스 테스트 기법이 있다.
구현 단계의 작업 절차

코딩 계획 => 코딩 => 컴파일 => 테스트

통합 개발 환경
IDE(Integrated Development Environment)

C++, JAVA 등의 언어를 이용한 소프트웨어 개발 단계에서 패키지 인크루딩, 소스 코드 편집, 컴파일, 디버깅, 바이너리 배포 등 모든 작업을 통합 지원한다.
편집기, 컴파일러, 디버거 등의 다양한 도구를 하나의 인터페이스로 통합하여 제공한다.
오류 체크를 시각화하여 확인 및 수정을 쉽도록 지원한다.
컴파일에 필요한 외부 추가 기능을 연계하여 개발의 편의성을 높였다.
종류 : 이클립스, 비주얼 스튜디오, XCode, 안드로이드 스튜디오, IDEA, VSC 등

빌드 자동화 도구

소스 코드 컴파일 후 다수의 연관된 모듈을 묶어 실행 파일로 만든다.
소프트웨어 개발자가 반복 작업해야 하는 코딩을 잘 짜여진 프로세스를 통해 자동으로 실행하여, 신뢰성 있는 결과물을 생산해 낼 수 있는 작업 방식 및 방법이다.
소스 코드를 컴파일, 테스트, 정적 분석 등을 실시하여 실행 가능한 애플리케이션으로 자동 생성하는 프로그램이며, 지속해서 증가하는 라이브러리의 자동 추가 및 관리(전처리, Preprocessing)를 지원한다.
최근에는 오픈소스인 Gradle 이 등장했으며, 구글이 안드로이드의 기본 빌드 시스템으로 Gradle 을 선택하면서 사용자가 급증하였다.
기능 : 코드 컴파일, 컴포넌트 패키징, 파일 조작, 개발 테스트 실행, 버전 관리 도구 통합, 문서 생성, 배포 기능, 코드 품질 분석
프로세스 : 컴파일 => 패키징 => 단위 테스트 => 정적 분석 => 리포팅 => 배포 => 최종 빌드
종류 : Gradle, Jenkins, Makefile, Ant, Maven 등

Ant

아파치 소프트웨어 재단에서 개발. XML 기반 빌드 스크립트를 사용한다.
정해진 규칙이 없고, 절차적이다.(명확한 빌드 절차 정의가 필요).
생명주기를 갖지 않아 각 Target 에 대한 의존관계와 작업을 정의해 주어야 한다.
유연성이 높으나 프로젝트가 복잡해지는 경우 Build 과정의 이해가 어려워진다.
XML, Remote Repository 를 가져올 수 없고 스크립트의 재사용이 어렵다.

Maven

프로젝트에 필요한 모든 종속성(Dependency)을 리스트의 형태로 Maven 에 알려서 종속성을 관리한다. 사용성이 좋지만 맞춤화된 로직 실행이 어렵다.
XML, Repository 를 가져올 수 있지만 라이브러리가 서로 종속할 경우 XML 이 복잡해진다.
'Jar', 'Class Path'를 선언만 하면 직접 다운로드할 필요가 없이 Repository 에서 계층적인 데이터를 표현하기에는 좋지만, 플로우나 조건부 상황을 표현하기 어렵다.

Gradle

JVM 기반의 빌드 도구이며, Ant 와 Maven 의 단점을 보완한 오픈소시 기반의 Build 자동화 도구로 프로젝트 시작 시 설정에 드는 시간을 절약할 수 있다.

한스도커를 중심으로 6 인의 개발자가 공동 개발하였다.
Maven 처럼 종속성을 활용하여 Groovy 기반 스크립트를 사용한다.
Maven 처럼 Groovy 를 기반으로 제작된 DSL(Domain Specicfic Language)을 스크립트 언어로 사용하는 오픈 소스 형태의 자동화 도구이다.
안드로이드 앱 개발 환경에서 사용된다.
if, else, for 등의 로직 구현이 가능하고, xml 을 사용하지 않아 간결하고 빠른 성능을 제공한다.
유연성과 확장성을 제공하며 하나의 Repository 내에 멀티 프로젝트를 구성할 수 있다.
스트립트는 Project 와 Tasks 두 가지 개념으로 구성된다.
실행할 처리 명령들을 모아 태스크로 만든 후 태스크 단위로 실행한다.
Jenkins
Java 기반의 오픈소스 형태의 빌드 자동화 도구로 쉽게 설치 가능하다.
서버 기반의 도구로서 클라이언트의 요청을 처리하기 위해 서버에서 실행되는 서블릿 실행과 생명주기를 관리하는 서블릿 컨테이너에서 실행된다.
Web UI 를 지원하고, SVN, Git 등의 대부분 형상 관리 도구와 연동 가능하다.
분산된 다수의 컴퓨터를 이용하여 분산 빌드, 테스트가 가능하다.
RSS, E-mail 을 통하여 빌드 실패 내역을 실시간 통지가 가능하다.
21. 제품 소프트웨어 패키징
애플리케이션 패키징(배포)
애플리케이션 패키징의 개념

개발이 완료된 소프트웨어를 고객에 인도하기 위해 패키징하고, 설치 메뉴얼, 사용 메뉴얼 등을 작성하는 일련의 배포용 설치 파일을 만드는 작업을 의미한다.
패키징 시 고려사항

사용자 시스템의 환경, 직관적 UI, 관리 서비스 형태 제공, 패키징 변경 및 개선 관리를 통한 안정적 배포
패키징 프로세스

기능 식별
입/출력 데이터를 식별하고 전체적인 기능 정의 및 데이터 흐름을 식별한다.
기능 단위 및 출력에 대하여 상세 정의한다.
모듈화
모듈화를 위하여 모듈 간 결합도와 응집도를 분석한다.
분류할 수 있는 기능 단위 및 서비스 단위를 모듈 별로 분류한다.
공유 가능한 기능과 재활용 기능을 분류한다.
빌드 진행

신규 개발 소스 및 컴파일 결과물을 준비한다.
정상적으로 빌드되는 기능 단위 및 서비스를 분류한다.
빌드 도구를 선별하여 선택하고, 해당 빌드 도구를 이용하여 빌드를 수행한다.
컴파일 회의 에디터 등의 관련 도구 기능을 확인한다.
사용자 환경 분석
고객의 편의를 위하여 최소 사용자 환경 사전을 정의한다.
다양한 사용자 환경 테스트를 수행한다.
패키지 적용 시험
실 사용자 환경에서의 패키징 적용을 테스트한다.
사용자 관점에서 UI 및 시스템상의 편의성을 점검한다.
패키징 변경 개선
사용자 관점에서 패키징 적용 시 개선점을 도출하여 서비스 가능한 수준의 개선 후 개선 버전을 다시 패키징한다.
제품 소프트웨어의 패키징 도구
패키징 도구

소프트웨어 배포를 목적으로 패키징 시에 지적 재산권을 보호하고, 관리하는 기능을 제공하는 도구이다.
소프트웨어의 안전한 유통 그리고 배포를 도와주는 솔루션이다.
패키징 도구는 불법 복제로부터 디지털 콘텐츠의 지적 재산권을 보호해 주는 사용 권한 제어 기술, 패키징 기술, 라이센스 관리, 권한 통제 기술 등을 포함한다.
패키징 도구 활용 시 고려사항

사용자에게 배포되는 소프트웨어임을 고려하여 반드시 내부 콘텐츠에 대한 암호화 및 보안을 고려한다.
다양한 이기종 콘텐츠 및 단말기 간 DRM 연동을 고려한다.
사용자 편의성을 위한 복잡성 및 비효율성 문제를 고려한다.
반드시 내부 콘텐츠에 대한 암호화 및 보안을 고려한다.
제품 소프트웨어에 적합한 암호화 알고리즘을 적용하여 범용성에 지장이 없도록 고려한다.
패키징 도구의 구성 요소

구성 특징
암호화
(Encryptoin) 콘텐츠 및 라이센스를 암호화하고, 전자 서명을 할 수 있는 기술이다.
ex. PKI, Symmetric/Asymmetric Encryption, Digital Signature
키 관리
(Key Management) - 콘텐츠를 암호화한 키에 대한 저장 및 배포 기술이다.

- 관리 방식 : 분산형, 중앙 집중형

암호화 파일 생성

(Packager) 콘텐츠를 암호화된 콘텐츠로 생성하기 위한 기술이다.
ex. Pre-packaging, On-the-fly Packaging

식별 기술

(Identificatoin) 콘텐츠에 대해 식별하고 체계화하는 기술이다.
ex. DOI, URI

저작권 표현

(Right Expression) 저작권의 라이센스 내용을 표현하는 기술이다.
ex. XrML/MPGE-21 REL, ODRL

정책 관리

(Policy Management) 라이센스 발급 및 사용에 대한 정책 표현 및 관리 기술이다.
ex. XML, Contents Management System

크랙 방지

(Tamper Resistance) 크랙에 의한 콘텐츠 사용 방지 기술이다.(Code Obfuscation, Kernel Debugger Detection, Module Certification)
ex. Secure DB, Secure Time Management, Encryption

인증

(Authentication) 라이센스 발급 및 사용의 기준이 되는 사용자 인증 기술이다.
ex. User/Device Authentication, SSO, Digital Certificate

모니터링 도구와 협업 도구

애플리케이션 모니터링 도구(APM : Application Performance Management)

응용 소프트웨어의 성능과 서비스 이용성을 감시하고 관리하는 데 초점을 둔 도구이다. 애플리케이션의 안정적인 시스템 운영을 위한 도구로써 부하량, 접속자 파알, 장애 진단, 통계, 분석 등을 목적으로 하는 성능 모니터링 제품으로 정의할 수도 있다.
애플리케이션 모니터링 도구의 기능

애플리케이션 변경 관리 : 애플리케이션 간의 종속 관계를 모니터링, 애플리케이션의 변경이 있을 경우 변경의 영향도 파악에 활용, ChangeMiner
애플리케이션 성능 관리 : 애플리케이션 서버로 유입되는 트랜잭션 수량, 처리 시간, 응답 시간 등을 모니터링, Jeniffer, Nmon

협업 도구

소프트웨어 개발 과정에서 이해관계자 간의 지속적 의견 조율을 수행하기 위한 도구이다.
분류 : 문서 공유, 소스 공유, 아이디어 공유, 디자인 공유, 일정 관리, 프로젝트 관리, 마인드맵

22. 제품 소프트웨어 저작권

제품 소프트웨어 저작권 보호

DRM(Digital Rights Management)

디지털 콘텐츠의 생성에서부터 실제 사용자까지 모든 유통 과정에 걸쳐 콘텐츠를 안전하게 관리 및 보호하고 허가된 사용자만이 접근할 수 있도록 제한하는 기술이다.
컴퓨터 소프트웨어는 무한 복제가 가능하고 원본과 복사본이 동일하게 배포될 가능성이 커이를 방지하기 위한 기술적인 방법을 통칭한다.
DRM 의 기술적 요구사항 : 지속적 보호(Persistent Protection), 이용 편리성(Easy to User), 유연성(Flexibility), 통합의 용이성(Seamless) 등의 4 가지로 분류할 수 있다.
DRM 의 특성

거래 투명성 : 저작권자와 콘텐츠 유통업자 사이의 거래 구조 투명성 제공
사용규칙 제공 : 사용 가능 횟수, 유효기간, 사용 환경 등을 정의 가능, 다양한 비즈니스 모델 구성 및 콘텐츠 소비 형태 통제 제공
자유료운 상거래 제공 : 이메일, 디지털 미디어, 네트워크 등을 통한 자유로운 상거래 제공, 허가받은 사용자는 별도의 비밀키를 이용하여 대상 콘텐츠를 복호화하고 허가된 권한으로 사용 가능
DRM 기술 요소

요소 기술 설명 방식
암호화
(Encryption) 콘텐츠 및 라이센스를 암호화하고, 전자서명을 할 수 있는 기술 PKI, Encryption Digital Signature
키 관리
(Key Management) 콘텐츠를 암호화한 키에 대한 저징 및 배포 기술 Centralized, Enveloping
암호화 파일 생성
(Packager) 콘텐츠를 암호화된 콘텐츠로 생성하기 위한 기술 Pre-Packaging, On-the-fly Packaging
식별 기술
(Identification) 콘텐츠에 대한 식별체계 표현 기술 DOI, URI
저작권 표현
(Right Expression) 라이센스의 내용 표현 기술 ODRL,XrML/MPGE-21 REL
정책 관리
(Policy Management) 라이센스 발급 및 사용에 대한 정책 표현 및 관리 기술 XML, Contents Management System
크랙 방지

(Temper Resistance) 크랙에 의한 콘텐츠 사용 방지 기술Secure DB, Secure Time Management, Encryption
인증
(Authentication) 라이센스 발급 및 사용의 기준이 되는 사용자 인증 기술 SSO, ID/PW, 디지털 인증, 이메일 인증
인터페이스 - 상이한 DRM 플랫폼 간의 상호 호환성
- 인터페이스 및 인증 기술 IPMP
이벤트 보고
(Event Reporting)콘텐츠의 사용이 적절하게 이루어지고 있는지 모니터링 기술로 불법유통이 탐지되었을 때 이동 경로를 추적에 활용
사용 권한
(Permission) 콘텐츠의 사용에 대한 권한을 관리하는 기술 요소 퍼미션(렌더, 트랜스포트, 데리버티브)
DRM의 유통 과정

콘텐츠 제공자가 라이센스 등록
콘텐츠 소비자가 라이센스 요청
요금 지불
라이센스 발급
콘텐츠 다운로드
구성 설명
콘텐츠 제공자 콘텐츠를 제공하는 저작권자
콘텐츠 분배자 쇼핑몰 등으로 암호화된 콘텐츠 제공
패키저 콘텐츠를 메타 데이터와 함께 배포 가능한 단위로 묶는 기능
보안 컨테이너 원본을 안전하게 유통하려는 전자적 보안 장치
DRM Controller 배포된 콘텐츠의 이용 권한을 통제
Clearing House - 키 관리 및 라이센스 발급 관리
- 디지털 저작권의 이용 생태계를 관리 및 감독하기 위한 제 3의 운영 주체로서 디지털 저작물의 이용 내역을 근거로 신뢰할 수 있는 저작권료의 정산 및 분배가 이루어지는 곳
디지털 콘텐츠의 사용 권한(Permission) 유형

유형 설명
렌더 퍼미션 사용자에게 콘텐츠가 표현되고 이용되는 권리 형태를 정의
ex. 문서(뷰, 프린트 권한 제어), 동영상(플레이 권한 제어)
트랜스포트 퍼미션 사용자들 간에 권리 교환이 이루어지는 권리 형태를 정의
ex. 카피(copy), 무브(move), 론(loan)

데리버티프 퍼미션 콘텐츠의 추출 변형이 가능한 권한
ex. 익스트랙트(Extract), 임베드(Embed), 에디트(Edit) 등
23. 제품 소프트웨어 메뉴얼 작성
소프트웨어 메뉴얼
소프트웨어 메뉴얼

제품 소프트웨어 개발 단계부터 적용한 기준이나 패키징 이후 설치와 사용자 측면의 주요 내용 등을 기록한 문서로 설치 메뉴얼과 사용자 메뉴얼로 구분된다.
소프트웨어 설치 메뉴얼

소프트웨어 실사용자가 제품을 최초 설치 시 참조하는 메뉴얼이며, 제품 소프트웨어 소개, 설치 파일, 설치 절차 등이 포함된다.
설치 과정에서 표시될 수 있는 예외 상황에 관련 내용을 별도로 구분하여 설명한다.
설치 시작부터 완료할 때까지의 전 과정을 빠짐없이 순서대로 설명한다.
설치 메뉴얼은 사용자 기준으로 작성한다.
설치 메뉴얼에는 목차, 개요, 기본사항 등이 기본적으로 포함되어야 한다.
소프트웨어 설치 메뉴얼 구성

목차 및 개요
작성하는 메뉴얼 전체 내용을 순서대로 요약하여 작성한다.
설치 메뉴얼의 주요 특징, 구성과 설치 방법, 순서 등에 관해 기술한다.
문서 이력 정보
메뉴얼 변경 이력에 대한 정보를 버전별, 시간순으로 작성한다.
설치 메뉴얼 주석
주의 사항 : 사용자가 제품 설치 시 반드시 숙지해야 하는 중요한 정보 주석으로 안내를 작성한다.
참고 사항 : 설치 관련하여 영향을 미치는 특별한 사용자 환경 및 상황에 대한 내용 주석으로 안내를 작성한다.
설치 도구의 구성
exe/dll/ini/chm 등 해당 설치 관련 파일에 대한 안내를 작성한다.
폴더 및 설치 프로그램 실행 파일에 대한 안내를 작성한다.
설치 위치 지정
설치 폴더와 설치 프로그램 실행 파일을 설정한다.
소프트웨어 설치 메뉴얼 기본 사항

제품 소프트웨어 개요, 설치 관련 파일, 설치 아이콘, 프로그램 삭제, 관련 추가 정보
소프트웨어 설치 환경 체크 항목

사용자 환경, 설치 시 실행 중인 다른 프로그램 종료 확인, 업그레이드 버전 존재 여부 확인, 백업 폴더 확인
소프트웨어 설치 메뉴얼 작성 프로세스

기능 식별
Main Funciton 등
UI 분류
화면 단위 UI
설치 파일/백업 파일 확인
실행, 환경, log, 백업
uninstall 절차
uninstall.exe 원복 절차
이상 Case 확인
Case 유형 및 Message
최종 메뉴얼 적용
Return 값, 정상 Message
소프트웨어 사용자 메뉴얼

사용자 메뉴얼

소프트웨어 설치와 사용에 필요한 제반 절차 및 환경 등 전체 내용을 포함하는 메뉴얼을 작성하며, 제품 소프트웨어에 대한 패치 개발과 업그레이들르 위해 버전 관리를 수행한다.
소프트웨어 사용 방법을 기술하며 패키지의 기능, 패키지의 인터페이스, 포함하고 있는 메서드나 오퍼레이션과 메서드의 파라미터 등의 설명이 포함돼야 한다.
사용자 메뉴얼의 구성

사용자 화면 및 UI
주의 사항 : 사용자가 반드시 숙지해야 하는 중요정보를 작성한다.
참고 사항 : 특별한 사용자 환경 및 상황에 대한 예외사항을 작성한다.
주요 기능 분류
설명할 기능을 포함할 화면을 스크린샷하여 작성한다.
동작하는 기능을 화면의 순서대로 차례로 분류하여 작성한다.
동작하는 기능을 화면의 순서대로 차례로 분류하여 작성한다.
기능 동작 시 참고 사항, 주의 사항 등을 메모로 추가한다.
응용 프로그램/설정
제품 실행 시 영향을 받거나 주는 소프트웨어에 대하여 설명한다.

동작 시 사전에 실행해야 할 소프트웨어가 있다면 기술한다.
동작에 필요한 기본 설정(Settings)과 기본 설정값을 안내한다.
장치 연동
제품 소프트웨어가 Embedded(장치 내에 내장) 관련된 제품일 경우에 해당 장치에 어떤 것이 있는지와 연동되는 장치에는 무엇이 있는지 설명한다.
Network 환경
제품 소프트웨어와 관련한 Network 정보를 표시(Status)하고, Network 에 정상 연결되었는지, 이를 위한 관련 설정값은 무엇이 있는지 설명한다.
Profile 설명
제품 소프트웨어 구동 시 체크하는 환경 파일이므로 환경 파일의 경로 변경, 이동을 금지하는 안내를 설명한다.
구동 시 필요한 필수 파일의 내용을 간략히 설명한다.
고객 지원 방법
설치 및 사용에 관련된 기술적 지원을 받을 수 있는 유선, 이메일, 홈페이지 등의 정보를 기재한다.
준수 정보 및 제한 보증
시리얼 코드를 불법 등록 사용하지 못하도록 준수사항을 안내한다.
저작권자의 지적 재산권, 허가권, 통신 규격, 개발 언어, 연동 프로그램, 문서 효력 등의 정보를 안내한다.
사용자 메뉴얼 작성 프로세스

작성 지침 정의 => 구성 요소 정의 => 구성 요소별 내용 작성 => 사용자 메뉴얼 검포

소프트웨어 국제 표준 품질 특성

ISO/IEC 9126
Information Technology-Software Quality Characteristices and Metrucs
소프트웨어 품질 특성과 척도에 관한 지침이다.
고객 관점에서 소프트웨어에 관한 품질 특성과 품질 부 특성을 정의한다.
ISO/IEC 12199
ISO/IEC 9126 의 품질 모델을 따르며 패키지 소프트웨어의 일반적인 제품 품질 요구사항 및 테스트를 위한 국제 표준이다.
제품 설명서, 사용자 문서 및 프로그램으로 구분하여 각각 품질 요구사항을 규정하고 있다.
ISO.IEC 15504
소프트웨어 프로세스를 평가하고 개선함으로써 품질 및 생산성을 높이고자 하는 표준이다.
평가 수준에 따라 개발 기관의 능력 레벨을 In-complete, Performed, Managed Established, Predictable, Optimizing level 등 6 단계로 구분하고 있다.
ISO 9901

Quality Systems-Model for Quality Assurance in Design, Development, Production, Installation and Servicing

설계, 개발, 생산, 설치 및 서비스 과정에 대한 품질 보증 모델이다.

공급자와 구매자 각각의 관리 책임을 명시하고 있으며 운영 중인 품질 시스템이 이 표준에 적합할 경우 품질 인증을 부여할 수 있도록 한다.

소프트웨어 품질 목표(Software Quality and Goals)

운영 특성

정확성

사용자의 요구 기능을 충족시키는 정도

신뢰성

주어진 시간 동안 주어진 기능을 오류 없이 수행하는 정도

사용 용이성

사용에 필요한 노력을 최소화하고 쉽게 사용할 수 있는 정도

적절한 사용자 인터페이스와 문서를 가지고 있는 정도

효율성

명시된 조건 하에서 소프트웨어 제품의 일정한 성능과 자원 소요량의 관계에 관한 속성 즉 요구되는 기능을 수행하기 위해 필요한 자원의 소요 정도

무결성

허용되지 않는 사용이나 자료의 변경을 제어하는 정도

변경 수용 특성

이식성

다양한 하드웨어 환경에서도 운용 가능하도록 쉽게 수정될 수 있는 정도

상호운용성

다른 소프트웨어와 정보를 교환할 수 있는 정도

재사용성

전체나 일부 소프트웨어를 다른 목적으로 사용할 수 있는가 하는 정도

유지보수성

사용자의 기능 변경의 필요성을 만족하기 위하여 소프트웨어를 진화하는 것이 가능한 정도

유연성

소프트웨어를 얼마만큼 쉽게 수정할 수 있는가의 정도

시험 역량

의도된 기능을 수행하도록 보장하기 위해 프로그램을 시험할 수 있는 정도.

소프트웨어 품질 측정 시 관점별 분류

사용자 관점 : 제품의 신뢰성, 효율성, 사용 용이성, 간결성 등

개발자 관점 : 검증 가능성, 유지보수성, 이식성, 무결성, 사용성 등

프로젝트 관리자 관점 : 프로세스의 생산성과 제어 용이성
릴리즈 노트 작성하기
릴리즈 노트(Release note)

애플리케이션 최종 사용자인 고객에게 제공하는 잘 정리된 배포 정보 문서이다.
애플리케이션 릴리즈 노트에는 상세 서비스를 포함하여 수정/변경된 정보를 담고 있는 문서이다.
사용자에게 최종 배포된 릴리즈 노트를 보면 테스트가 어떻게 진행됐는지, 개발팀의 제공 사양을 얼마나 준수했는지를 확인해 볼 수 있다.
전체적인 버전 관리 및 릴리즈 정보를 체계적으로 관리할 수 있다.
릴리즈 노트는 현재 시제로 개발팀에서 직접 작성하여야 하며, 명확하고 정확하며 완전한 정보를 제공해야 한다.
개발자와 테스터가 함께 협업해야 하고 최초 및 변경, 개선 항목까지 연결되어 다음 항목에 대한 정보들이 릴리즈 노트를 통해 작성되어야 한다.
릴리즈 노트 작성 항목

헤더(Header) : 문서명, 제품명, 배포 버전 번호, 릴리즈 날짜, 참고 날짜, 문서(릴리즈 노트) 버전 등
개요 : 제품 및 변경에 대한 정보를 간략하게 작성한다.
목적 : 제품의 버그 픽스(오류 수정)와 새로운 기능을 포함한 릴리즈의 새로운 사항의 나열과 더불어 릴리즈 노트의 목적에 대한 간략한 개요를 작성한다.
이슈 요약 : 문제가 되는 버그의 간단한 설명과 개선사항 항목을 요약하여 작성한다.
재현 항목 : 버그 발생을 재현하기 위한 절차이다.
수정 및 개선 내용 : 수정 및 개선 내용을 간략하게 서술한다.
최종 사용자 영향도 : 최종 사용자에게 필요한 조치로, 이 변경사항으로 인해 다른 기능이 영향을 받는지 간략히 서술한다.
노트 : 소프트웨어 및 하드웨어 설치 항목, 제품, 문서를 포함한 업그레이드 항목을 서술한다.
면책 조항 : 회사와 표준 제품과 관련된 메시지를 작성한다.(프리웨어, 불법 복제 금지 등)
연락 정보 : 사용자 지원 및 문의 관련한 연락처 정보를 작성한다.
릴리즈 노트 작성 순서

모듈 식별
모듈 및 빌드 수행 후 릴리즈 노트 기준의 항목을 순서대로 정리한다.
소스를 통하여 처리되는 입/출력 데이터의 형, 기능 정의, 데이터 흐름을 정리한다.

메인 함수 이외의 호출 함수를 정의하고 이에 대한 출력 값을 식별한다.(ex. I/O 데이터, Function Data Flow)
추가 개선 항목 식별
추가 개선에 따른 추가 항목을 식별하여 릴리즈 노트를 작성한다.
추가 개선에 대한 베타 버전을 이용 테스트 수행한다.
테스트 중 발생한 긴급 버그를 수정한다.
사용자 요청에 따른 추가 개선을 계획하고 수정한다.(ex. 베타 버전, 긴급 버그, 사용자 요청)
24. 형상 관리
형상 관리 도구
형상 관리(Configuration Management)

개발 단계에서 생성되는 모든 문서, 코드 등 소프트웨어의 변경사항을 체계적으로 관리하기 위하여 추적하고 통제하는 것이다.
작업 산출물을 형상 항목(Configuration Item)이라는 형태로 선정하고, 형상 항목 간의 변경사항 추적과 통제 정책을 수립하고 관리한다.
요구사항 변경 또는 오류로 지속해서 변화하는 자료이며, 이러한 변화를 이력화하여 유지보수성을 향상할 수 있다.
소프트웨어는 눈으로 확인할 수 있는 가시성이 없으므로 개발 과정의 진행 정도를 확인하는 도구로 사용된다.
단순 버전 관리 기반의 소프트웨어 운용을 좀 더 포괄적인 학술 분야의 형태로 넓히는 근간을 의미한다.
형상 관리 항목(Configuration Item)

개발 프로세스에서 생산되거나 사용되는 작업 산출물, 작업 산출물들의 집합체를 의미한다.
대표적인 소프트웨어 형상 항목 : 프로젝트 요구 분석서, 운영 및 설치 지침서, 요구사항 명세서, 설계/인터페이스 명세서, 테스트 설계서, 소프트웨어 품질보증, 형상 관리, V & V 계획서(확인 및 검증)와 같은 계획서, 코드 모듈(소스와 오브젝트 모두)
형상 관리 종류

형상 관리는 버전관리, 리비전 관리, 변경 관리, 빌드 관리, 이슈 관리 등을 모두 포함한다.

버전 관리

다양한 형상항목이 과거부터 현재에 이르기까지 요구사항 등의 변화에 따라 버전을 부여함으로써 이력을 관리하는 것이다.

버전을 통해 시간적인 변경사항과 해당 작업 담당자를 추적할 수 있다.
변경 관리

변경된 요구사항에 대하여, 비용 및 기간 등을 고려하고 타당성을 평가한다.
요구사항이 타당한 경우 제품 또는 산출물을 변경하고, 그렇지 않을 경우 변경을 거부하는 활동이다.
형상 관리 도구

소프트웨어 개발 생명주기 전반에 걸쳐 생성되는 소스 코드와 문서 등과 같은 산출물의 종합 및 변경 과정을 체계적으로 관리하고 유지하는 일련의 개발 관리 활동이다.
소프트웨어에 가시성과 추적 가능성을 부여하여 제품의 품질과 안전성을 높인다.
형상 식별, 형상 통제, 형상 상태 보고, 형상 감사를 통하여 변경사항을 관리한다.
이전 리비전이나 버전에 대한 정보에 접근 가능하여 배포본 관리에 유용하다.
불필요한 사용자의 소스 수정을 제한할 수 있다.
형상 관리의 필요성과 효과

형상관리의 필요성
이미 수정된 오류가 갑자기 다시 나타나거나, 사용하던 문서나 코드가 갑자기 사라지거나 찾을 수 없는 경우가 발생할 수 있다.
원시 코드와 실행 코드의 버전이 일치하지 않는다.
요구사항이 자주 변경되고, 변경이 어떤 결과를 가져올지 예측할 수 없다.
무엇을 변경해야 할지 막연하고, 따라서 변경에 대한 노력을 예측할 수 없다.
분산된 지역에서 소프트웨어를 병렬적으로 개발하기 어렵다.
제품 납기일을 맞추기가 어렵고, 프로젝트가 계획대로 잘 진행되고 있는지 모르겠다.
형상 관리의 효과
관리적 효과
표준 확입으로 전사적 IT 자원 관리가 쉬워, 기간별/팀별/업무별 산출물 현황 및 변경 이력 통계를 파악할 수 있다.
제품 개발 관련 산출물이 자동 생성되고 관리된다.
개발/유지보수 활동을 통합 관리할 수 있다.
변경 프로세스의 체계를 확립하고, 외주 개발 통제 및 현황 파악을 도와준다.
품질 향상 효과
산출물 버전 관리를 자동으로 생성 관리할 수 있어 결함 및 오류가 감소한다.
변경 프로그램의 이력 관리를 통하여 문제 파악 및 버그 수정이 쉬워지고, 변경 내용의 영향 분석이 쉬워진다.
형상 관리 절차

형상 관리는 최초 계획을 수립하고 형상 식별, 통제, 감사, 기록 및 보고와 같은 활동들을 통해 일련의 과정들을 거치게 된다.

형상 식별(Configuration Identification)

형상 관리의 가장 기본이 되는 활동으로 형상 관리 계획을 근거로 형상 관리의 대상이 무엇인지 식별하는 과정이다.

변경 추적성 부여와 대상 식별을 위해 ID 와 관리 번호를 할당한다.

형상 항목 대상 : 품질 관리 계획서, 품질 관리 메뉴얼, 요구사항 명세서, 설계/인터페이스 명세서, 테스트 설계서, 소스 코드

형상 통제

형상통제위원회 운영을 통하여 변경 통제가 이루어져야 한다.

요구사항 변경 요구를 관리하고, 변경 제어, 형상 관리 등의 통제를 지원하고 기준선에 대한 관리 및 형상 통제를 수행할 수 있다.

형상 보고 및 감사

기준선의 무결성 평가 단계로서 개발자, 유지보수 담당자가 아닌 제 3 자의 객관적인 확인 및 검증 과정을 통해 새로운 형상의 무결성을 확보하는 활동이다. -형상 감사 시 고려사항 - 명시된 변경이 정확하게 수정되었는가? - 기술 검토를 수행하였는가? - 개발 프로세스를 준수하였는가? - 변경 발생 시, 형상 관리 절차를 준수하였는가? - 변경에 대한 정보(변경일, 변경인, 변경사항)를 기록하였는가?

형상 기록/보고

소프트웨어 개발 상태에 대한 보고서를 제공하는 단계로 기준선에 대한 변경과 처리 과정에서의 변경을 상태 보고에 모두 기록한다.

기록/보고 항목 : 승인된 형상 리스트, 계획된 변경 상태, 승인된 변경의 구현 상태

형상 관리, 버전 관리, 변경 관리

형상 관리 (Configuration Management) 버전 관리 (Version Management) 변경 관리(Version Management)

버전, 변경 관리 개념을 포함하고, 프로젝트 진행 상황, 빌드와 릴리즈 퍼블리싱까지 모두 관리할 수 있는 통합 시스템이라고 할 수 있다. - 변경 이력을 추적 관리하는 가장 좋은 방법이 버전으로 구분하는 것이다.

- 사소한 체크인, 체크아웃부터 릴리즈, 퍼블리싱의 과정을 버전으로 관리한다. - 소스 코드의 변경 상황을 관리한다.

- 문서의 변경 이력과 복원 등의 기능이 제공된다.

버전 관리 도구

버전 관리 도구 구분

공유 폴더 방식

담당자 한 명이 공유 폴더 내 자료를 자신의 PC 로 복사한 후 컴파일하여 이상 유무를 확인하고, 파일의 오류가 확인되면, 해당 파일을 등록한 개발자에게 수정 의뢰한다.

개발자들은 매일 완료된 파일을 공유 폴더에 복사하여 관리한다.

파일에 이상이 없다면 다음날 각 개발자가 동작 여부를 다시 확인한다.

파일의 변경사항을 데이터베이스에 기록하여 관리한다.

종류 : SCCS, RCS, PVCS, QVCS

클라이언트/서버 방식

버전 관리 자료가 중앙 시스템(서버)에 저장되어 관리되는 방식이다.

서버의 자료를 개발자별로 자신의 PC(클라이언트)로 복사하여 작업 후 변경된 내용을 서버에 반영하고, 모든 버전 관리는 서버에서 수행하는 방식이다.

하나의 파일을 서로 다른 개발자가 작업할 경우 경고 메시지를 출력한다.

서버에 문제가 생기면, 서버가 복구되기 전까지 다른 개발자와의 협업 및 버전 관리 작업을 중단한다.

종류 : CVS, SVN(Subversion), CMVC, Perfoce, CVSNT, Clear Case

분산 저장소 방식

버전 관리 자료가 원격 저장소와 로컬 저장소에 함께 저장되어 관리된다.

로컬 저장소에서 버전 관리가 가능하므로 원격 저장소에 문제가 생겨도 로컬 저장소의 자료를 이용하여 작업할 수 있다.

개발자별로 원격 저장소의 자료를 각자의 로컬 저장소로 복사하여 작업 후 변경사항을 로컬 저장소에서 우선 적용하여 로컬 버전 관리가 가능하다.

개발 완료한 파일을 수정한 다음에 로컬 저장소에 먼저 커밋한 이후, 다시 원격 저장소에 반영하는 방식이다.

종류 : Git, Bazaar, Mercurial, TeamWare,

주요 버전 관리 도구

CVS(Concurrent Versions System)

동시 버전 시스템이다.

소프트웨어 프로젝트를 진행할 때, 파일로 이뤄진 모든 작업과 모든 변화를 추적하고, 여러 개발자가 협력하여 작업할 수 있게 한다.

오픈소스 프로젝트에서 널리 사용되었다.

최근에는 CVS 가 한계를 맞아 이를 대체하는 Subversion 이 개발되었다.

RCS(Revision Control System)

CVS 와의 차이점은 소스 파일의 수정을 한 사람만으로 제한한다.

다수의 사용자가 동시에 파일 수정을 할 수 없도록 파일 잠금 방식으로 버전을 관리하는 도구이다.

SubVersion(SVN)

CVS 보다 속도 개선, 저장 공간, 변경 관리 단위가 작업 모음 단위로 개선되었다.
2000 년부터 콜랩넷에서 개발되었다.
CVS 와 사용 방법이 유사해 CVS 사용자가 쉽게 도입해 사용할 수 있다.
아파치 최상위 프로젝트로서 전 세계 개발자 커뮤니티와 함께 개발되고 있다.
디렉토리, 파일을 자유롭게 이동해도 버전 관리가 가능하다.
repository(저장소) : 프로젝트의 파일 및 변경 정보가 저장되는 장소이다.
trunk : 메인 개발 소스. 개발 소스를 커밋했을 때 개발 소스가 모이는 곳이다.
branch : trunk 에서 분기된 개발 소스로 실험적인 기능을 추가하거나, 출시를 위한 안정화 버전 작업을 할 때 사용한다.
tag : 특정 시점에서 프로젝트의 스냅샷을 찍어 두는 것을 의미한다.
Bitkeeper
SVN 과 비슷한 중앙 통제 방식으로 대규모 프로젝트에서 빠른 속도를 내도록 개발된 버전 관리 도구이다.
Git
프로그램 등의 소스 코드 관리를 위한 분산 저장소 방식 시스템이다.
리누스 토르발스가 리눅스 커널 개발에 이용하려고 개발하였으며, 현재는 다른 곳에도 널리 사용되고 있다.
지역 저장소와 원격 저장소 2 개의 저장소가 존재한다.
지역 저장소에서 버전 관리가 진행되어 버전 관리가 빠르다.
깃의 작업 폴더는 모두 전체 기록과 각 기록을 추적할 수 있는 정보를 포함하고 있으며, 완전한 형태의 저장소이다.
네트워크에 접근하거나 중앙 서버에 의존하지 않는다.
Clear Case
복수 서버, 복수 클라이언트 구조이다.
서버 확장 요구가 있을 때 필요한 서버를 하나씩 추가할 수 있다.
컴포넌트 저장소(Repository)

인증을 받은 컴포넌트를 등록하는 저장소로 손쉽게 컴포넌트를 이용할 수 있다.
저장소는 컴포넌트의 최신 버전을 유지하고 있으며, 컴포넌트의 버전별 상태도 유지하고 관리함으로써 사용자가 컴포넌트 이용을 쉽게 한다.
Git 주요 명령어

init : 새로운 로컬 git 생성
add : 저장소(Staging Area)에 파일을 추가하기
commit : 작업 내역 지역 저장소에 저장하기
branch : 새로운 파생 저장소인 브랜치 생성
checkout : 선택한 브랜치로 이동하기

merge : 현재 브랜치와 지정한 브랜치를 병합하기
fetch : git 서버에서 코드를 받아오기
Pull : git 서버에서 최신 코드 받아와 병합하기
remote : 원격 저장소 추가하기
clone : 원격 저장소에 있는 프로젝트 복사하여 내려받기
SubVersion(SVN) 주요 명령어

import : 아무것도 없는 서버의 저장소에 맨 처음 소스 파일을 저장한다.
checkin : 체크아웃으로 가져온 파일을 수정 후 저장소(Repository)에 새로운 버전으로 갱신한다.
checkout : 타 개발자가 수정 작업을 위하여 저장소(Repository)에 저장된 파일을 자신의 작업 공간으로 인출한다.
commit : 체크인 시 이전 갱신 사항이 있는 경우 충돌(Conflict)이 있을 경우 알림을 표시하고 diff(코드 비교) 도구를 이용하여 수정한 뒤 commit 과정을 수행한다.
diff : 새로운 개발자가 추가된 파일의 수정 기록(Change Log)을 보면서 기존 개발자가 처음 추가한 파일과 이후 변경된 파일의 차이를 본다(Diff)
update : 저장소에 존재하는 최신 버전 자료와 자신의 작업 공간과 동기화한다.
branch : 주 저장소에서 파생된 프로젝트이다.
fork : 주 저장소에서 소프트웨어 소스 코드를 통째로 복사하여 독립적인 새로운 소프트웨어 개발을 허용하는 것으로, 제시된 라이센스 기준을 지켜야 한다.
update : commit 후 새로운 개발자가 자신의 작업 공간과 저장소를 동기화한다.
info : 지정된 파일에 대한 정보를 표시한다.
merge : 다른 디렉토리에서 작업된 버전 관리 내역을 기본 개발 작업과 병합한다.
25. 애플리케이션 테스트 관리
테스트 케이스
소프트웨어 테스트

소프트웨어 개발 단계에서 사용자 요구사항에 서술된 동작과 성능, 사용성, 안정성 등을 만족하는지 확인하기 위하여 소프트웨어의 결함을 찾아내는 활동으로 품질 향상, 오류 발견, 오류 예방 관점에서 수행하는 행동이다.
품질 향상 관점 : 반복적인 테스트를 거쳐 제품의 신뢰도를 향상하는 품질 보증 활동이다.
오류 발견 관점 : 잠재된 오류를 발견하고 이를 수정하여 올바를 프로그램을 개발하는 활동이다.
오류 예방 관점 : 코드 리뷰, 동료 검토, 인스펙션 등을 통해 오류를 사전에 발견하는 활동이다.
소프트웨어 테스트의 원리

테스팅은 결함이 존재함을 밝히는 활동이다.
소프트웨어의 잠재적인 결함을 줄일 수 있지만, 결함이 발견되지 않아도 결함이 없다고 증명할 수 없음을 나타낸다.
완벽한 테스팅은 불가능하다.
무한 경로, 무한 입력값, 무한 시간이 소요되어 완벽하게 테스트할 수 없으므로 리스크 분석과 우선순위를 토대로 테스트에 집중하는 것을 의미한다.
테스팅은 개발 초기에 시작해야 한다.
애플리케이션의 개발 단계에 테스트를 계획하고 SDLC(Software Development Life Cycle)의 각 단계에 맞춰 전략적으로 접근하는 것을 고려하라는 뜻이다.
결함 집중(Defect Clustering)
애플리케이션 결함의 대부분은 소수의 특정한 모듈에 집중되어 존재한다. 파레토 법칙이 좌우한다.
살충제 역설(Pesticide Paradox)
동일한 테스트 케이스로 반복 테스트 시 결함을 발견할 수 없으므로 주기적으로 테스트 케이스를 리뷰하고 개선해야 한다.
테스팅은 정황(Context)에 의존한다.
정황과 비즈니스 도메인에 따라 테스트를 다르게 수행하여야 한다.
오류 부재의 궤변(Absence of Errors Fallacy)
사용자의 요구사항을 만족하지 못하는 오류를 발견하고 그 오류를 제거하였다 해도, 해당 애플리케이션의 품질이 높다고 말할 수 없다.
파레토의 법칙(Law of Pareto)

80 대 20 법칙 또는 2:8 법칙이라고도 한다. 전체 결과의 80%가 전체 원인의 20%에서 일어나는 현상을 가리킨다. 예를 들어 20%의 VIP 고객이 백화점 전체 매출의 80%에 해당하는 만큼 쇼핑하는 현상을 의미한다.
테스트 케이스(Test Case)

구현된 소프트웨어가 사용자의 요구사항을 정확하게 준수했는지를 확인하기 위해 설계된 입력값, 실행 조건, 기대 결과 등으로 구성된 테스트 항목에 대한 명세서를 의미한다.
명세 기반 테스트의 설계 산출물이다. (명세 기반 테스트 : 테스트 수행의 증거로도 활용되며, 사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 구현하고 있는지 확인)
테스트 케이스를 설계 단계에 작성하면 테스트 시 오류를 방지하고, 테스트 수행에 있어 낭비를 줄일 수 있다.
표준 테스트 케이스 형식

테스트 계획 검토 및 자료 확보
위험 평가 및 우선 순위 결정
테스트 요구사항 정의
테스트 구조
설계 및 테스트 방법 결정
테스트 케이스 정의
테스트 케이스 타당성 확인 및 유지보수
테스트 케이스의 구성 요소(ISO/IEC/IEEE 29119-3)

식별자(Identifier)
테스트 항목(Test Item)
입력 명세(Input Specification)
출력 명세(Output Specification)
환경 설정(Environmental Needs)
특수 절차 요구(Special Procedure Requirement)
의존성 기술(Inter-case Dependencies)
테스트 프로세스(Test Process)

계획 및 제어 => 분석 및 설계 => 구현 및 실행 => 평가 => 완료

테스트 커버리지(Test Coverage)

테스트 수행 정도로서 구문 커버리지, 결정 커버리지, 조건 커버리지, 조건/결정 커버리지, 변경 조건/결정 커버리지, 다중 조건 커버리지로 구분한다.
테스트 오라클(Test Oracle)

테스트의 결과가 참인지 거짓인지를 판단하기 위해서 사전에 정의된 참(True) 값을 입력하여 비교하는 기법 및 활동을 말한다.
True 오라클
모든 입력값에 대하여 적합한 결과를 생성하여, 발생한 오류를 모두 검출 할 수 있는 오라클이다.
일관성 검사(Consistent) 오라클
애플리케이션 변경이 있을 때, 수행 전과 후의 결과값이 동일한지 확인하는 오라클이다.
샘플링(Sampling) 오라클
임의로 선정한 몇 개의 입력값에 대해서만 기대하는 결과를 제공한다.
휴리스틱(Heuristic) 오라클

샘플링 오라클을 개선한 오라클로, 임의 입력값에 대해 올바른 결과를 제공하고, 나머지 값들에 대해서는 휴리스틱(추정)으로 처리한다.
V 모델과 테스트
테스트 레벨

애플리케이션 개발 단계에 따라 단위 테스트, 통합 테스트, 시스템 테스트, 인수 테스트, 설치 테스트로 분류한다.
애플리케이션을 테스트하기 위한 총체적으로 관리하기 위한 테스트 활동의 묶음이다.
각각의 테스트 레벨은 서로 독립적, 각각 다른 테스트 계획과 전략이 필요하다.
시각에 따른 테스트

검증(Verification) 테스트 : 제품이 명세서대로 완성되었는지 검증하는 단계이다.
개발자의 시각에서 제품의 생산 과정을 테스트하는 것을 의미한다.
확인 테스트 : 사용자의 요구사항을 잘 수행하고 있는지 사용자의 시각에서 생산된 제품의 결과를 테스트하는 것을 의미한다.
테스트 케이스 자동 생성

자료 흐름도 => 테스트 경로 관리, 입력 도메인 분석 => 테스트 데이터 산출, 랜덤 테스트 => 무작위 값 입력, 신뢰성 검사

테스트 레벨의 종류

단위 테스트
개발자가 원시 코드를 대상으로 각각의 단위를 다른 부분과 연계되는 부분은 고려하지 않고 단위 자체에만 집중하여 테스트한다.
객체지향에서 클래스 테스팅이 여기에 해당한다.
통합 테스트
단위 테스트를 통과한 개발 소프트웨어/하드웨어 컴포넌트 간 인터페이스 및 연동 기능 등을 구조적으로 접근하여 테스트한다.
시스템 테스트
단위/통합 테스트가 가능한 완벽히 완료되어 기능상 문제가 없는 상태에서 실제 환경과 가능한 유사한 환경에서 진행된다.
시스템 성능과 관련된 요구사항이 완벽하게 수행되는지를 테스트하기 때문에 사전 요구사항이 명확해야 한다.
개발 조직과는 독립된 테스트 조직에서 수행한다.
인수 테스트

일반적인 테스트 레벨의 가장 마지막 상위 레벨로, SW 제품에 대한 요구사항이 제대로 이행되었는지 확인하는 단계이다.
테스팅 환경을 실 사용자 환경에서 진행하며 수행하는 주체가 사용자이다.
알파, 베타 테스트와 가장 밀접한 연관이 있다.
알파 테스트(Alpha Test)와 베타 테스트(Beta Test)

알파 테스트
개발자 관점에서 수행되며, 사용상의 문제를 반영되도록 하는 테스트이다.
개발자의 장소에서 사용자가 개발자 앞에서 행해지며, 오류와 사용상의 문제점을 사용자와 개발자가 함께 확인하면서 검사하는 기법이다.
개발자는 사용상의 문제를 기록하여 반영되도록 하는 테스트이다.
베타 테스트
선정된 다수의 사용자가 자신들의 시용 환경에서 일정 기간 사용하면서 테스트한다.
문제점이나 개선 사항 등을 기록하고 개발 조직에 통보하여 반영되도록 하는 테스트이다.
애플리케이션 테스트
정적 테스트

애플리케이션을 직접 실행하지 않고 명세서나 소스 코드를 대상으로 분석하는 테스트 방식이다.
소프트웨어 개발 초기에 결함 발견이 가능하여, 개발 비용을 낮출 수 있다.
종류 : Inspection, walk-through, Code Test, Orthogonal Array Testing, Prior Defect History Testing, Risk-Based Testing, Run Chart, Statistical Profile Testing
동적 테스트

애플리케이션을 직접 실행하여 오류를 찾는 테스트 방식이다.
소프트웨어 개발의 모든 단계에서 테스트를 수행한다.
종류 : 블랙박스 테스트(명세 기반), 화이트박스 테스트(구조 기반)
테스트 기반(Test Bases)에 따른 테스트

구분 설명
구조 기반 테스트 - 소프트웨어 내부의 구조(논리 흐름)에 따라 테스트 케이스를 작성하고 확인하는 테스트 방식이다.
- 종류 : 구문 기반, 조건 기반, 데이터 흐름
명세 기반 테스트 - 사용자의 요구사항에 대한 명세를 기반으로 테스트 케이스를 작성하고 확인하는 테스트 방식이다.

- 종류 : 동등 분할, 경계값 분석, 분류 트리, 상태 전이, 결정 테이블, 원인-결과, 조합 테스트, 시나리오, 오류 추정

경험 기반 테스트 - 테스터의 경험을 기반으로 수행하는 테스트 방식이다.

- 요구사항에 대한 명세가 미흡하거나 테스트 시간에 제약이 있는 경우 수행하면 효과적이다.
- 종류 : 에러 추정, 체크 리스트, 탐색적 테스팅

목적에 따른 테스트

구분	설명
성능 (Performance)	소프트웨어의 응답 시간, 처리량 등을 테스트한다.
회복 (Recovery)	소프트웨어에 고의로 부하를 가하여 실패하도록 유도하고 올바르게 복구되는지 테스트한다.
구조 (Structured)	소프트웨어 내부의 논리적인 경로, 소스 코드의 복잡도 등을 평가한다.
회귀 (Regression)	소프트웨어의 변경 또는 수정된 코드에 새로운 결함이 없음을 확인한다.
안전 (security)	소프트웨어가 불법적인 침입으로부터 시스템을 보호할 수 있는지 확인한다.
강도 (Stress)	소프트웨어에 과도하게 부하를 가하여도 소프트웨어가 정상적으로 실행되는지 확인한다.
병행(Parallel)	변경된 소프트웨어와 기존 소프트웨어에 동일한 데이터를 입력하여 두 결과를 비교 확인한다.

26. 테스트 시나리오와 테스트

테스트 시나리오

테스트 시나리오

테스트 케이스를 적용하는 순서에 따라 여러 테스트 케이스의 집합으로서, 테스트 케이스의 동작 순서를 기술한 문서이며 테스트를 위한 절차를 정리한 문서이다.

테스트 순서에 대한 구체적인 절차, 사전 조건, 입력 데이터 등을 정리하여, 테스트 항목을 빠짐없이 수행할 수 있도록 한다.

테스트 시나리오 작성 시 유의점

테스트 항목을 시스템별, 모듈별, 항목별 테스트 시나리오를 분리하여 작성한다.
고객의 요구사항과 설계 문서 등을 토대로 테스트 시나리오를 작성한다.
테스트 항목은 식별자 번호, 순서 번호, 테스트 데이터, 테스트 케이스, 예상 결과, 확인 등의 항목을 포함하여 작성한다.
테스트 환경 구축

개발된 응용 소프트웨어가 실제 운영 시스템에서 정상적으로 작동하는지 테스트할 수 있도록 하기 위하여 실제 운영 시스템과 동일 또는 유사한 사양의 하드웨어, 소프트웨어, 네트워크 등의 시설을 구축하는 활동이다.
테스트 환경 구축의 유형

하드웨어 기반 : 서버 장비(WAS, DBMS), 클라이언트 장비, 네트워크 장비 등의 장비를 설치하는 작업이다.
소프트웨어 기반 : 구축된 하드웨어 환경에 테스트할 응용 소프트웨어를 설치하고 필요한 데이터를 구축하는 작업이다.
가상 시스템 기반 : 물리적으로 개발 환경 및 운영 환경과 별개로 독립된 테스트 환경을 구축하기 힘든 경우에는 가상 머신(Virtual Machine) 기반의 서버 또는 클라우드 환경을 이용하여 테스트 환경을 구축하고, 네트워크는 VLAN 과 같은 기법을 이용하여 논리적 분할 환경을 구축할 수 있다.
테스트 기법
화이트박스 테스트(white Box Test)

모듈의 원시 코드를 오픈시킨 상태에서 코드의 논리적 모든 경로를 테스트하는 방법이다.
Source Code 의 모든 문장을 한 번 이상 수행하여 모듈 안의 작동을 직접 관찰할 수 있다.
산출물의 기능별로 적절한 프로그램의 제어 구조에 따라 선택, 반복 등의 부분들을 수행함으로써 논리적 경로를 점검한다.
화이트박스 테스트 종류

기초 경로 검사(Base Path Testing)
TomMcCabe 가 제안한 대표적 화이트박스 테스트 기법
테스트 케이스 설계자가 절차적 설계의 논리적 복잡성을 측정할 수 있게 한다.
측정 결과는 실행 경로의 기초를 정의하는데 지침으로 사용된다.
기초 경로(Basis Path) : 제어 흐름 그래프를 분석하여 선형 독립 실행 경로 집합을 찾는다. McCabe 의 순환 복잡도를 사용하여 선형 독립 경로 수를 결정한 다음 얻어진 각 경로에 대한 테스트 사례를 생성한다.

제어 구조 검사
조건 검사 : 프로그램 모듈 내에 있는 논리적 조건을 테스트하는 테스트 케이스 설계 기법이다.
루프 검사 : 프로그램의 반복 구조에 초점을 맞춰 실시하는 테스트 케이스 설계 기법이다.
데이터 흐름 검사 : 프로그램에서 변수의 정의와 변수 사용의 위치에 초점을 맞춰 실시하는 테스트 케이스 설계 기법이다.
화이트박스 테스트 검증 기준

문장 검증 기준 : 소스 코드의 모든 구문이 한 번 이상 수행된다.
분기 검증 기준 : 소스 코드의 모든 조건문이 한 번 이상 수행된다.
조건 검증 기준 : 소스 코드의 모든 조건문에 대해 조건이 True 인 경우와 False 인 경우가 한 번 이상 수행된다.
분기/조건 기준 : 소스 코드의 모든 조건문과 각 조건문에 포함된 개별 조건식의 결과가 True 인 경우와 False 인 경우 한 번 이상 수행된다.
블랙박스 테스트(Black Box Test)

블랙박스 테스트는 소프트웨어가 수행할 특정 기능을 알기 위해 각 기능이 완전히 작동되는 것을 입증하는 테스트로 기능 테스트라고도 한다.
요구사항 명세를 보면서 테스트하며, 주로 구현된 기능을 테스트한다.
소프트웨어 인터페이스에서 실시되는 테스트이다.
블랙박스 테스트 종류

동치 분할 검사(Equivalence Partitioning)
입력 자료에 초점을 맞춰 테스트 케이스를 만들고 검사하는 방법이다.
입력 조건에 타당한 입력 자료와 그렇지 않은 자료의 개수를 균등하게 분할해 테스트 케이스를 설정한다.
원인-효과 그래프 검사(Cause and Effect Graphing)
입력 데이터 간의 관계와 출력에 영향을 미치는 상황을 체계적으로 분석한다.
효용성이 높은 테스트 케이스를 선정해 검사한다.
오류 예측 검사(Error Forecast)
과거의 경험이나 감각으로 테스트하는 기법이다.
다른 테스트 기법으로는 찾기 어려운 오류를 찾아내는 보충적 검사 기법이다.
비교 검사(Comparision Testing)
동일한 테스트 자료를 여러 버전의 프로그램에 입력하고 동일한 결과가 출력되는지 테스트하는 기법이다.
경계값 분석(Boundary Value Analysis)

입력 자료에만 치중한 동치 분할 기법을 보완한 기법이다.
입력 조건 경계값에서 오류 발생 확률이 크다는 것을 활용하여 경계값을 테스트 케이스로 선정해 검사한다.
대표적인 명세 기반 기법(Specification based Technique)이다.
27. 테스트 커버리지
테스트 커버리지
테스트 커버리지

주어진 테스트 케이스에 의해 수행되는 소프트웨어의 테스트 범위를 측정하는 테스트 품질 측정 기준이며, 테스트의 정확성과 신뢰성을 향상시키는 역할을 한다.
기능 기반
테스트 대상 애플리케이션의 전체 기능을 모수로 설정하고, 실제 테스트가 수행된 기능의 수를 측정하는 방법이다.
기능 기반 테스트 커버리지는 100% 달성을 목표로 하며, 일반적으로 UI 가 많은 시스템의 경우 화면 수를 모수로 사용할 수도 있다.
Line Coverage
애플리케이션 전체 소스 코드의 Line 수를 모수로 테스트 시나리오가 수행한 소스 코드의 Line 수를 측정하는 방법이다.
단위 테스트에서는 이 라인 커버리지를 척도로 삼기도 한다.
Code Coverage
소프트웨어 테스트 충분성 지표 중 하나로 소스 코드의 구문, 조건, 결정 등의 구조 코드 자체가 얼마나 테스트 되었는지를 측정하는 방법이다.
Statement Coverage(구문)
코드 구조 내의 모든 구문에 대해 한 번 이상 수행하는 테스트 커버리지를 말한다.
Condition Coverage(조건)
결정 포인트 내의 모든 개별 조건식에 대해 수행하는 테스트 커버리지를 말한다.
Decision Coverage(결정)
결정 포인트 내의 모든 분기문에 대해 수행하는 테스트 커버리지를 말한다.
Modified Condition/Decision Coverage
조건과 결정을 복합적으로 고려한 측정 방법이며, 결정 포인트 내의 다른 개별적인 조건식 결과에 상관없이 독립적으로 전체 조건식의 결과에 영향을 주는 테스트 커버리지를 말한다.
테스트 커버리지 유형

기능 기반 커버리지, 라인 커버리지, 코드 커버리지(구분, 결정, 조건, 변경 조건/결정)
테스트 자동화
테스트 자동화 도구

애플리케이션 개발 중 반복되는 다양한 테스트 과정을 HW/SW 적으로 자동화 도구를 사용하고 일관성 및 생산성을 향상시키는 도구이다.
테스트 관리, 소스 코드 리뷰 및 인스펙션, 테스트 설계 및 개발, 테스트 수행 등 테스트에 포함되는 다양한 과정을 자동으로 지원하는 도구이다.
테스트 자동화 수행 시 고려사항

모든 과정이 아닌 그때그때 맞는 적절한 도구를 선택
자동화 도구를 고려하여 프로젝트 일정 계획
프로젝트 초기에 테스트 엔지니어 투입 시기 계획
테스트 자동화 도구의 유형

정적 분석 도구
프로그램을 실행하지 않고 소스 코드 분석을 통해 결함을 발견하는 도구이다.
코딩 표준, 코딩 스타일, 코딩 복잡도, 남은 결함 등을 발견하기 위해 사용한다.
테스트 실행 도구
스크립트 언어를 사용하여 테스트를 실행하는 방법으로서 테스트 데이터와 수행 방법 등이 포함된 스크립트를 작성한 후 실행한다.
데이터 주도 접근 방식
테스트 데이터를 스프레드시트 문서에 저장하고 실행하는 방식으로 다양한 테스트 데이터를 동일한 테스트 케이스로 반복하여 실행할 수 있다.
새로운 데이터의 경우 미리 작성된 스크립트에 테스트를 추가하여 테스트를 진행할 수 있다.
키워드 주도 접근 방식
테스트를 수행할 동작을 나타내는 키워드와 테스트 데이터를 스프레드시트 문서에 저장하여 실행하는 방식이다.
키워드를 이용하여 테스트를 정의할 수 있다.
성능 테스트 도구
애플리케이션의 처리량, 응답 시간, 경과 시간, 자원 사용률에 대해 가상의 사용자를 생성하고 테스트를 수행함으로써 성능 목표를 달성하였는지를 확인하는 테스트 자동화 도구이다.
테스트 하네스(Test Harness) 도구 구성 요소

테스트 드라이버(Test Driver)
하위->상위 모듈로 통합하면서 테스트하는 상향식 테스트에서 사용된다.
테스트 대상을 제어하고 동작시키는데 사용되는 도구를 의미한다.

시스템 및 컴포넌트를 시험하는 환경의 일부분으로 시험을 지원하는 목적 하에 생성된 코드와 데이터이다.
테스트 스텁(Test Stub)
상위->하위 모듈 방향으로 통합 테스트를 진행하는 하향식 테스트에서 사용한다.
상위 모듈에서 하위 모듈로의 테스트를 진행하는 과정 중 하위 시스템 컴포넌트의 개발이 완료되지 않은 상황에서 시스템 테스트를 진행하기 위하여 임시로 생성된 가상의 더미 컴포넌트(Dummy Component)이다.
테스트 슈트(Test Suites)
일정한 순서에 의하여 수행될 개별 테스트들의 집합, 또는 패키지이다.
슈트는 응용 분야나 우선순위, 내용에 연관된다.
테스트 케이스(Test Case)
요구에 맞게 개발되었는지 확인하기 위하여 테스트할 입력과 예상 결과를 정의한 것이다.
테스트 자동화를 도입하면 테스트 케이스는 데이터 레코드로 저장될 수 있고 테스트 스크립트로 정의할 수 있다.
테스트 스크립트(Test Script)
테스트 케이스를 수행하여 그 결과를 보고할 목적으로 명령어 또는 이벤트 중심의 스크립트 언어로 작성한 파일로 수행경로에 영향을 미칠 논리 조건들을 포함하고 있다.
테스트 수행 단계별 테스트 자동화 도구

테스트 계획 단계 : 요구사항 관리 도구
테스트 분석 및 설계 단계 : 테스트 케이스 생성 도구
테스트 수행 단계 : 테스트 자동화/정적 분석/동적 분석/성능 테스트/모니터링 도구
테스트 관리 단계 : 커버리지 분석/형상 관리/결함 추적 및 관리 도구
28. 통합 테스트
통합 테스트
단위 테스트

소프트웨어 최소 기능 단위인 모듈, 컴포넌트를 테스트하는 것으로 사용자의 요구사항을 기반으로 한 기능 테스트를 제일 먼저 수행한다.
인터페이스, 자료 구조, 독립적 기초 경로, 오류 처리 경로, 결제 조건 등을 테스트한다.
구조 기반 테스트와 명세 기반 테스트로 분류할 수 있으나 주로 구조 기반 테스트를 시행한다.
통합 테스트(Integration Test)

각 모듈 간을 결합하여 시스템을 완성시키는 과정에서 모듈 간 인터페이스 혹은 통합된 컴포넌트 간 상호작용 오류 및 결함을 찾아 해결하기 위한 테스트 기법이다.

비점진적 통합 방식
(빅뱅 통합) 점진적 통합 방식
(상향식/하향식)
- 모든 모듈이 결합된 프로그램 전체를 대상으로 테스트한다.
- 규모가 작은 소프트웨어에 적합하다.
- 오류 발견/장애 위치 파악 또는 수정이 어렵다. - 단계적으로 통합하며 테스트한다.
- 오류 수정이 쉽다.
- 인터페이스 관련 오류를 테스트할 수 있다.
통합 방식
하향식 통합

상위 컴포넌트를 테스트하고 점증적으로 하위 컴포넌트를 검사한다.
주요 제어 모듈 기준으로 아래로 통합하며 진행한다.
하위 컴포넌트 개발이 완료되지 않은 경우 스텁(Stub)을 사용하기도 한다.
우선 통합법, 깊이 우선 통합법, 넓이 우선 통합법 등이 있다.
하위 레벨 모듈들은 특정한 소프트웨어 부가 기능을 수행하는 클러스터들에 결합된다.
상향식 통합

프로그램 구조에서 최하위 레벨인 모듈을 구성하고 상위 모듈 방향으로 통합하며 검사한다.
가장 하위 단계의 모듈부터 수행되므로 스터브가 필요 없으나 하나의 주요 제어 모듈과 관련된 종속 모듈의 그룹인 클러스터가 필요하다.
빅뱅(BigBang) 통합

시스템을 구성하는 모듈을 각각 따로 구현하고 전체 시스템을 한 번에 테스트를 진행한다.
테스트를 위한 Driver 와 Stub 없이 실제 모듈들로 테스트를 진행한다.
단시간 테스트를 수행하나 결함의 격리가 어려운 방식이다.
샌드위치 통합

상향식과 하향식의 장점을 이용하는 방식(하향식 + 상향식)이다.
하위 프로젝트가 있는 대규모 프로젝트에 사용하는 방식이다.
병렬 테스트가 가능하고 시간 절약이 가능하다.
스텁(Stub)과 드라이버(Driver)의 필요성이 매우 높은 방식이며, 비용이 많이 들어간다.
통합 테스트 수행 방법 비교

구분 상향식 하향식 빅뱅(BigBang)
Driver/Stub 드라이버 스텁 실제 모듈로 테스트

수행 하위->상위 상위->하위 동시
장점 - 장애 위치 확인 용이
- 모든 모듈이 준비되어 있지 않아도 됨 - 장애 위치 확인 용이
- 초기 프로토 타입 가능 소규모 시스템에 단기간 테스트 가능
단점 - 초기 프로토타입 불가
- 중요한 모듈들이 마지막에 테스트 될 가능성이 있음 - 많은 스텁 필요
- 낮은 수준 모듈은 부적절한 테스트 가능성 있음- 장애 위치 확인 어려움
- 모든 모듈의 개발이 준비되어 있어야 함.
29. 결함 관리
결함 관리
결함

소프트웨어의 에러(Error), 결함(Defect), 결점(Fault), 버그(Bug),
실패(Failure)와 같은 용어가 사용되며 이러한 결함으로 인하여 설계와 다르게 동작하거나 다른 결과가 발생하는 것을 의미한다.
심각도별 분류
치명적(Critical) 결함
주요(Major) 결함
보통(Normal) 결함
경비한(Minor) 결함
단순(Simple) 결함
결함 우선순위 : 결정적, 높음, 보통, 낮음 또는 즉시 해결, 주의 요망, 대기, 개선 권고 순으로 표시하며, 결함의 심각도가 높다고 해서 반드시 우선순위가 높은 것은 아니다.
결함 유입별 분류

기획 시 유입되는 결함, 설계 시 유입되는 결함, 코딩 시 유입되는 결함, 테스트 부족으로 유입되는 결함 등으로 분류한다.
결함 분류

시스템 결함 : 주로 애플리케이션이나 데이터베이스 처리에서 발생된 결함이다.
기능 결함 : 애플리케이션의 기획, 설계, 업무 시나리오 등의 단계에서 유입된 결함이다.
GUI 결함 : 화면 설계에서 발생된 결함이다.
문서 결함 : 기획자, 사용자, 개발자 간 의사소통 및 기록이 원활하지 않아 발생된 결함이다.
결함 관리 프로세스

결함 관리 계획 : 결함 관리에 대한 일정, 인력, 업무 프로세스를 확보하여 계획을 수립

결함 기록 : 테스터는 발견된 결함에 대한 정보를 결함 관리 DB 에 기록
결함 검토 : 등록된 결함에 있어서 주요 내용을 검토하고, 결함을 수정할 개발자에게 전달
결함 수정 : 개발자는 할당된 결함 프로그램 수정
결함 재확인 : 테스터는 개발자가 수정한 내용을 확인하고 다시 테스트 수행
결함 상태 추적 및 모니터링 : 결함 관리 팀장은 결함 관리 DB 를 이용하여 대시보드 또는 게시판 형태의 서비스를 제공
최종 결함 분석 및 보고서 작성 : 발견된 결함과 관련된 내용과 이해관계자들의 의견이 반영된 보고서를 작성하고 결함 관리를 종료
결함 관리 도구 및 용어
결함 관리 도구

Mantis: 소프트웨어 설계 시 단위별 작업 내용을 기록할 수 있어 결함 및 이슈 관리, 추적을 지원하는 오픈소스 도구
Trac : 결함 추적 및 통합 관리를 지원하는 오픈소스 도구
Bugzilla : 결함을 지속적으로 관리하고 심각도와 우선순위를 지정할 수 있는 오픈소스 도구
Redmine : 프로젝트 관리 및 결함 추적 도구
Jira : 아틀래시안에서 제작한 PHP 로 개발된 결함 상태 관리 도구
Test Collab : 테스트 케이스를 관리하기 위한 간단하고 쉬운 인터페이스를 제공하며 Jira, Redmine, Asana, Mantis 등과 같은 버그 추적 도구와의 완벽한 통합 지원
결함 관련 용어

에러(Error) : 소프트웨어 개발 또는 유지보수 수행 중에 발생한 부정확한 결과로, 개발자의 실수로 발생한 오타, 개발 명세서의 잘못된 이해, 서브루틴의 기능 오해 등이 있다.
오류(Fault) : 프로그램 코드상에 존재하는 것으로 비정상적인 프로그램과 정상적인 프로그램 버전 간의 차이로 인하여 발생되며, 잘못된 연산자가 사용된 경우에 프로그램이 서브루틴으로부터의 에러 리턴을 점검하는 코드가 누락된 것을 말한다.
실패(Failure) : 정상적인 프로그램과 비정상적인 프로그램의 실행 결과의 차이를 의미하며, 프로그램 실행 중에 프로그램의 실제 실행 결과를 개발 명세서에 정의된 예상 결과와 비교함으로써 발견한다.
결함(Defect) : 버그, 에러, 오류, 실패, 프로그램 실행에 대한 문제점, 프로그램 개선사항 등의 전체를 포괄하는 용어이다.
결함 내성(Fault Tolerance)

시스템을 구성하는 부품의 일부에서 결함(Fault) 또는 고장(Failure)이 발생하여도 정상적 혹은 부분적으로 기능을 수행할 수 있는 내성을 의미한다.
고장 허용성이라고도 한다.

30. 애플리케이션 성능개선

애플리케이션 성능개선
성능 측정 지표

처리량(Throughput) : 주어진 시간에 처리할 수 있는 프로세스 처리 수
응답 시간(Response Time) : 데이터 입력 완료 시부터 응답 출력이 개시될 때까지의 시간
경과 시간(Turnaround Time) : 입력한 시점부터 그 결과의 출력이 완료할 때까지 걸리는 시간
자원 사용률(Resource Usage) : 프로세스 처리 중 사용하는 CPU 사용량, 메모리 사용량, 네트워크 사용량
유형별 성능 분석 도구

성능/부하/스트레스(Performance/Load/Stress) 정검 도구 : 측정 지표인 처리량, 응답 시간, 경과 시간 등을 점검하기 위해 가상의 시스템 부하나 스트레스를 통해 성능을 분석하는 도구이다.
모니터링(Monitoring) 도구 : 성능 모니터링, 성능 저하 원인 분석, 시스템 부하량 분석, 장애 진단, 사용자 분석, 용량 산정 등의 기능을 통하여 애플리케이션 실행 시 자원 사용량을 확인하고 분석 가능한 도구이다.
애플리케이션 성능 저하 원인
데이터베이스 연결 및 쿼리 실행 시 발생되는 성능 저하 원인

DB Lock
과도한 데이터 조회/업데이트/인덱스 생성 시 발생한다.
Lock 의 해제 시까지 대기하거나 처리되지 못하고 종료된다.
불필요한 DB Fetch
필요한 데이터보다 많은 대량의 데이터 요청이 들어올 경우 발생한다.
결과 세트에서 마지막 위치로 커서를 옮기는 작업이 빈번한 경우 응답 시간 저하 현상이 발생한다.
연결 누수(Connection Leak)
DB 연결과 관련한 JDBC 객체를 사용 후 종료하지 않을 경우 발생한다.
부적절한 Connection Pool Size
커넥션 풀 크기가 너무 작거나 크게 설정한 경우 발생한다.
기타

트랜잭션이 Commit 되지 않고 커넥션 풀에 반환되거나, 잘못 작성된 코드로 인해 불필요한 commit 이 자주 발생하는 경우 발생한다.
내부 로직으로 인한 성능 저하 원인

웹 애플리케이션의 인터넷 접속 불량이나 대량의 파일로 인해 부하가 발생하는 경우이다.
정상적으로 처리되지 않은 오류 처리로 인한 부하나 트랜잭션이 수행되는 동안 외부 트랜잭션(외부 호출)이 장시간 수행되거나, 타임아웃이 일어나는 경우이다.
잘못된 환경 설정이나 네트워크 문제로 인한 성능 저하 원인

환경 설정으로 인한 성능 저하 : Thread pool, Heap Memoty 의 크기를 너무 작게 설정하면 Heap Memory Full 현상이 발생한다.
네트워크 장비로 인한 성능 저하 : 라우터, L4 스위치 등 네트워크 관련 장비 간 데이터 전송 실패 또는 전송 지연에 따른 데이터 손실이 발생한다.
알고리즘
알고리즘

주어진 과제를 해결하기 위한 방법과 절차를 의미한다.
알고리즘은 자연어, 의사코드(Pseudocode), 순서도, 프로그래밍 언어를 이용하여 표현 가능하다.
알고리즘 설계 기법

분할 정복법(Divide & Conquer)
제시된 문제를 분할이 불가할 때까지 나누고, 각 과제를 풀면서 다시 병합해 문제의 답을 얻는 Top-down 방식이다.
분할(Divide) : 정복이 필요한 과제를 분할이 가능한 부분까지 분할하고,
정복(Conquer) : 1 에서 분할된 하위 과제들을 모두 해결(정복)한다.
결합(Combine) : 그리고 2 에서 정복된 해답을 모두 취합(결합)한다.
ex. Quick sort, Merge sort 알고리즘
동적 계획법(Dynamic Programming)
주어진 문제를 해결하기 위해 부분 문제에 대한 답을 계속적으로 활용해 나가는 Bottom-up 방식이다.
부분 문제로 분리
가장 낮은 단계의 부분 문제 해답 계산
이 부분 문제의 해답을 이용해 상위 부분 문제를 해결
이전 단계의 해답을 활용하기 위해 반드시 기억할 수 있는 저장소가 필요하기 때문에 속도는 빠르지만, 공간 복잡도가 커지는 단점이 있다.

ex. 플로이드 알고리즘, 피보나치 수열 알고리즘(재귀 호출(동적 계획법) 뿐만 아니라 분할 정복법을 통해서도 구현 가능)

탐욕법(Greedy Method)

국소적인 관점에서 최적의 해결 방법을 구하는 기법으로 최적의 해결 방법을 구하지는 못하나 동적 계획법보다 효율적이라고 할 수 있다.

ex. 크루스칼 알로리즘, 다익스트라 알고리즘

백트래킹(Back tracking)

어떤 문제의 최적해를 구하기 위해 모든 가능성을 찾아가는 방법이다.

N-Queen 문제 해결 시에 응용된다. -동적 계획법과 같이 기억할 저장소를 필요로 한다.

분기 한정법(Branch & Bound)

정해진 범위(Bound)를 벗어나는 값들은 가지치기(Branch)해가며 결과값을 추적해 나가는 방식이다.

ex. 최적 우선 탐색(Best First Search) 알고리즘, A* 알고리즘

근사 해법(Approximation Algorithm)

복잡도가 매우 높은 문제게 대해 가장 근사치의 값을 구하는 기법이다.- NP-Hard 문제를 해결하기 위해, 주어진 시간에 최적해를 가장 가까운 답을 찾는 결정성 알고리즘을 구현하는 기법이다.

시간 복잡도, 공간 복잡도, 정밀도를 척도로 평가된다.

ex. 근사 알고리즘

시간 복잡도에 따른 알고리즘

시간 복잡도는 알고리즘이 문제를 해결하기 위한 시간(연산)의 횟수를 말한다.

시간 복잡도를 고려하는 것을 최적화를 위해 필요하다.

알고리즘의 소요 시간에 대한 정확한 평가는 어려워 자료의 수 n 이 증가할 때 시간이 증가하는(Time Complexity) 대략적인 패턴을 의미한다.

시간 복잡도 Big-O 표기법

O(1) : 상수 시간의 복잡도를 의미하며 입력값 n 이 주어졌을 때, 문제를 해결하는데 오직 한 단계만 거친다.(해시 함수).

$O(\log_2 n)$: 로그 시간의 복잡도를 의미하며 입력값 n 이 주어졌을 때, 문제를 해결하는데 필요한 단계들이 연산마다 특정 요인에 의해 줄어든다.(이진 탐색)

$O(n\log_2 n)$: 선형 로그 시간의 복잡도를 의미하며, 문제 해결을 위한 단계 수는 n log2 n 번의 수행 시간을 갖는다.(퀵 정렬, 병합 정렬)

O(n) : 선형 시간의 복잡도를 의미하며 문제를 해결하기 위한 단계의 수와 입력값 n 이 1:1 관계이다.(순차 탐색)

$O(n^2)$: 제곱 시간의 복잡도를 의미하며 문제를 해결하기 위한 단계의 수는 입력값 n 의 제곱근이다.(버블 정렬, 삽입 정렬, 선택 정렬)

O(Cⁿ) : 지수 시간의 복잡도를 의미하며 문제를 해결하기 위한 단계의 수는 주어진 상수값 C 의 n 제곱이다.
McCabe 순환 복잡도(Cyclomatic)
순환 복잡도

프로그램의 이해 난이도는 제어 흐름 난이도의 복잡도에 따라 결정되며, 복잡도를 싸이클로메틱 개수에 의해서 산정하는 방법이다.
싸이클로메틱의 개수와 원시 프로그램 오류의 개수는 밀접한 관계가 있다.
최대 10 을 넘지 않도록 하며 넘으면 이를 분해하도록 한다.
복잡도 계산 방식

복잡도 = 화살표 수 - 노드 수 + 2(제어 흐름 그래프를 통해 파악)
복잡도 = 영역 수(폐 구간) + 1(제어 흐름 그래프를 통해 파악)
복잡도 = 의사 결정수 + 조건 수 + 1(프로그램 코드상에서 파악, 제어 흐름도를 그리기 어려운 경우)
V(G) = E - N + 2

E : 화살표 수, N 은 노드 수(점)

해싱 함수의 종류

제산 방법(Division Method) : 나머지 연산자(%)를 사용하여 테이블 주소를 계산하는 방법.
중간 제곱 방법(Mid-Square Method) : 레코드 키값을 제곱한 후에 결과값의 중간 부분에 있는 몇 비트를 선택하여 해시 테이블의 홈주소로 사용하는 방법이다.
중첩 방법(Folding Method, 폴딩) : 해싱 함수 중 레코드 키를 여러 부분으로 나누고, 나눈 부분의 각 숫자를 더하거나 XOR 한 값을 홈주소로 삼는 방법이다.
기수 변환 방법(Radix Conversion Method) : 레코드 키를 구성하는 수들이 모든 키들 내에서 자리별로 어떤 분포인지를 조사하여 비교적 고른 분포를 나타내는 자릿수를 필요한 만큼 선택, 레코드의 홈주소로 사용하는 방법이다.
무작위 방법(Random Method) : 난수를 발생시킨 후 난수를 이용해 각 키의 홈주소를 산출하는 방법이다.
계수 분석 방법(Digit Analysis Method) : 레코드 키를 구성하는 수들이 모든 키들 내에서 자리별로 어떤 분포인지를 조사하여 비교적 고른 분포를 나타내는 자릿수를 필요한 만큼 선택, 레코드의 홈주소로 사용하는 방법이다.
동의어(Synonym) : 해싱에서 동일한 홈주소로 인하여 충돌이 일어난 레코드들의 집합이다.

31. 소스 코드 최적화
소스 코드 최적화
소스 코드 최적화

읽기 쉽고 변경 및 추가가 쉬운 클린 코드를 작성하는 것을 의미한다.
소스 코드 품질을 위해 기본적으로 지킬 원칙과 기준을 정의하고 있다.
나쁜 코드
다른 개발자가 로직을 이해하기 어렵게 작성된 코드
변수/메서드에 대한 명칭을 알 수 없는 코드이다. -동일한 처리 로직이 중복되게 작성된 코드이다.
스파게티 코드
유형 : 오염, 문서 부족, 의미 없는 이름, 높은 결합도, 아키텍처 침식
클린 코드
깔끔하게 잘 정리된 코드
중복 코드 제거로 애플리케이션의 설계가 개선된다.
가독성이 높아 애플리케이션의 기능에 대해 쉽게 이해할 수 있다.
버그를 찾기 쉬워지며, 프로그래밍 속도가 빨라진다.
클린 코드 최적화 원칙 : 가독성, 단순성, 의존성 배제, 중복성 취소화, 추상화
유형 : 보기 좋은 배치, 작은 함수, 분석 가능한 제어 흐름, 오류 처리, 간결한 주석, 의미 있는 이름
코드 간결성 유지 지침

공백을 이용하여 실행문 그룹과 주석을 명확히 구분하고, 복잡한 논리식과 산술식은 괄호와 들여쓰기(Indentation)를 통해 명확히 표현한다.
빈 줄을 사용하여 선언부와 구현부를 구별하고 한 줄에 되도록 적은 문장을 코딩한다.
클린 코드의 작성 원칙

가독성 : 이해하기 쉬운 용어를 사용하고 들여쓰기 등을 활용하여 코드를 쉽게 읽을 수 있도록 작성한다.
단순성 : 클래스/메서드/함수는 최소 단위로 분리해 한 번에 한 가지 기능만 처리한다.
의존성 배체 : 다른 모듈에 미치는 영향을 최소화하여 코드 변경 시 다른 부분에 영향이 없도록 작성한다.
중복성 최소화 : 중복된 코드는 삭제하여 공통된 코드로 사용한다.
추상화 : 상위 클래스/메서드/함수에서 간략하게 애플리케이션 특성을 나타내고, 상세 내용은 하위 클래스/메서드/함수에서 구현한다.
소스코드 최적화 유형

클래스 분할 배치
하나의 클래스는 하나의 역할만 수행하도록 응집도를 높이도록 한다.
모듈 크기를 작게 작성한다.
좋은 이름 사용
변수나 함수 이름은 Namming Rule 을 정의하여 기억하기 좋고, 발음이 쉽게 사용한다.
코딩 형식 준수
개념적 유사성 높은 종속 함수를 사용하여 논리적으로 코드를 라인별로 구분하여 가독성을 높이다.
호출하는 함수 앞쪽에, 호출되는 함수를 배치하고, 지역 변수는 각 함수 맨 처음에 선언한다.
느슨한 결합(Loosely Coupled)
클래스 간 의존성이 느슨하게 하기 위해 인터페이스 클래스를 이용하여 추상화된 자료 구조와 메서드를 구현한다.
적절한 주석
코드의 간단한 기능 안내 및 중요 코드를 표시할 때 적절히 사용한다.
소스 코드 품질 분석
소스 코드 품질 분석 도구

소스 코드의 코딩 스타일, 코드에 설정된 코딩 표준, 코드의 복잡도, 코드에 존재하는 메모리 누수 현상, 스레드 결함 등을 발견하기 위해 사용하는 분석 도구이다.
정적 분석 도구
잠재적인 실행 오류와 코딩 코딩 표준 위배 사항 등 보안 약점을 검출한다.
검출된 약점을 수정/보완하여 소프트웨어의 안정성을 강화하고 향후 발생하는 오류 수정 비용을 줄일 수 있다.
개발 초기의 결함을 찾을 때 사용하며, 개발 완료 시점에서는 개발된 소스 코드의 품질 검증을 위해 사용한다.
소스 코드에서 코딩의 복잡도, 모델 의존성, 불일치성 등을 분석할 수 있다.
기법
소스 코드 검증 : 검증 가이드라인을 통한 보안 조치
코드 리뷰 : 개발자가 작성하고 다른 개발자가 정해진 방법을 통해 검토하는 방법(동료 검토, 제 3 자 검토라고도 함)
리버스 엔지니어링 : 시스템의 기술적인 원리를 구조분석을 통해 발견하는 방법
종류
pmd, cppcheck, SonarQube, checkstyle, ccm, cobertura 등
동적 분석 도구

SW 소스 코드보다는 실행 과정에서의 다양한 입/출력 데이터의 변화 및 사용자 상호 작용에 따른 변화를 점검하는 분석 기법이다.
기법
디버깅 : 논리적인 오류(버그)를 찾아내는 테스트 과정
스트레스 테스트 : 결과 관찰을 위해 한계점에 이르는 테스트를 수반
모의 해킹 : 내부 또는 외부에서 실제 해커가 사용하는 해킹 도구와 기법 등을 이용하여 정보 시스템으로의 침투 가능성을 진단하는 선의의 해킹 기법
리버스 엔지니어링 : 동적 역공학 분석 툴을 이용하여 구조 분석
종류 : Avalanche, Valgrind, ValMeter 등
정적 분석과 동적 분석 기술의 비교

분류 정적 분석 동적 분석
대상 소스 코드 실제 애플리케이션
평가 기술 오염 분석, 패턴 비교 애플리케이션 실제 실행
단계 애플리케이션 개발 단계 애플리케이션 개발 완료 단계
32. 인터페이스 구현
인터페이스 확인
인터페이스 설계서(정의서)

시스템의 인터페이스 현황을 한눈에 확인하기 위하여, 이기종의 시스템 간 데이터 교환과 처리를 위해 사용되는 데이터뿐 아니라, 업무, 송/수신 시스템 등에 관한 상세 내용을 기술한 문서이다.

정적, 동적 모형을 통한 설계서, 일반적 형태의 설계서로 구분된다.

클래스 분할 배치

시각적인 다이어그램을 이용하여 정적, 동적 모형으로 각 시스템의 구성 요소를 표현한 문서이다.
각 인터페이스가 어느 부분에 속하는지 분석할 수 있다.
교환 트랜잭션 종류를 분석할 수 있다.
적절한 주석

개별 인터페이스의 상세 데이터 명세, 시스템 인터페이스 목록, 각 기능의 세부 인터페이스 정보를 정의한 문서이다.
시스템 인터페이스 설계서 : 시스템 인터페이스 목록을 만들고 각 인터페이스 목록에 대한 상세 데이터 명세를 정의하는 것이다.

상세 기능별 인터페이스 명세서 : 각 기능의 세부 인터페이스 정보를 정의한 문서이다.
내/외부 모듈 간 공통 기능, 데이터 인터페이스 확인 순서

인터페이스 설계서의 외부 및 내부 모듈의 기능을 확인한다.
인터페이스 설계서의 외부 및 내부 모듈을 기반으로 공통적으로 제공되는 기능과 각 데이터의 인터페이스를 확인한다.
모듈 연계
모듈 연계

시스템 인터페이스를 목적으로 내부 모듈-외부 모듈 또는 내부 모듈-내부 모듈 간 인터페이스를 위한 관계를 설정하는 것으로 EAI 와 ESB 방식이 있다.
EAI(Enterprise Application Integration)

기업 내부에서 운영되는 각종 플랫폼 및 애플리케이션 간의 정보 전달, 연계, 통합을 가능하게 해 주는 솔루션이다.
각 비즈니스 간 통합 및 연계성을 증대시켜 효율성을 높일 수 있다.
각 시스템 간의 확장성을 높여 줄 수 있다.
EAI 유형

Point-to-Point
애플리케이션을 중간 미들웨어 없이 Point to Point 로 연결하는 기본적인 통합 방식이다.
별도로 솔루션(미들웨어)을 구매하지 않고 구축할 수 있다.
상대적으로 저렴하게 구축 하능하지만 변경 및 재사용이 어렵다.
Hub & Spoke
단일 접점인 허브 시스템을 통해 데이터를 전송하는 중앙 집중형 방식으로 확장 및 유지보수가 수월하다.
중앙 허브에 장애가 발생하면 시스템 전체에 영향을 준다.
Message Bus
애플리케이션 사이에 미들웨어를 배치하여 처리하는 방식으로 확장성이 뛰어나다.
대용량 데이터 처리에 유리하다.
Hybrid
Hub & Spoke 와 Message Bus 의 혼합 방식이다.
그룹 내 : Hub & Spoke
그룹 간 : Message Bus
데이터 병목현상을 최소화할 수 있다.
필요한 경우 한 가지 방식으로 EAI 구현이 가능하다.
ESB(Enterprise Service Bus)

애플리케이션 간의 데이터 변환 및 연계 지원 등을 제공하는 인터페이스 제공 솔루션이다.
애플리케이션 간의 통합 관점으로 EAI 와 유사하다고 볼 수 있으나, 애플리케이션보다는 서비스 중심으로 통합을 지향하는 아키텍처 또는 기술을 의미한다.
범용적으로 사용하기 위해서는 애플리케이션과의 결합도를 약하게 유지해야 한다.
웹 서비스 중심으로 표준화된 데이터, 버스를 통해 이 기종 애플리케이션을 유연하게 통합하는 핵심 플랫폼(기술)이다.(Loosely-Coupled)
관리 및 보안이 쉽고 높은 수준의 품질 지원이 가능하다.
데이터 표준 확인

내/외부 모듈 간 데이터를 교환 시 데이터 표준을 정의하고 이를 관리하여야 한다.
기존 데이터 중 공통 영역을 추출하여 정의하는 경우와 인터페이스를 위해 다른 한쪽의 데이터 형식을 변환하는 경우가 있다.
JSON, DB, XML 등 다양한 표준으로 인터페이스 모듈을 표현할 수 없다.
절차 : 인터페이스 기능을 통해 인터페이스 데이터 표준을 확인 => 인터페이스 데이터 항목을 식별 => 데이터 표준을 최종 확인
인터페이스 기능 정의
인터페이스 기능 정의

인터페이스를 실제로 구현하기 위해 인터페이스 기능에 대한 구현 방법을 기능별로 기술하는 과정이다.
정의 순서 : 컴포넌트 명세서 확인 => 인터페이스 명세서 확인 => 일관된 인터페이스 기능 구현 정의 => 정의된 인터페이스 구현 정형화
모듈 세부 설계서

모듈 구성 요소와 세부적 동작 등을 정의한 설계서이다.
컴포넌트 명세서 인터페이스 명세서
- 내부 클래스 동작, 컴포넌트 개요, 인터페이스를 통해 외부와 통신하는 명세를 정의한다.
- 구성 : 컴포넌트 ID, 컴포넌트명, 컴포넌트 개요, 내부 클래스(ID, 클래스명, 설명), 인터페이스 클래스(ID, 인터페이스명, 오퍼레이션명, 구분) - 컴포넌트 명세서 항목 중 인터페이스 클래스의 세부 조건 및 기능 등을 정의한다.
- 구성 : 인터페이스 ID, 인터페이스명, 오퍼레이션명, 오퍼레이션 개요, 사전 조건, 사후 조건, 파라미터, 반환값
모듈 세부 설계서 확인

컴포넌트가 가지고 있는 주 기능은 컴포넌트 명세서(컴포넌트 개요, 내부 클래스의 클래스명, 설명 등)를 확인한다.

인터페이스에 필요한 기능을 각 모듈의 컴포넌트 명세서, 인터페이스 명세서를 통하여 분석한다.
인터페이스에 필요한 주 기능은 인터페이스 클래스를 통해 확인하고 인터페이스 명세서를 통해서 컴포넌트 명세서의 인터페이스 클래스에 작성된 인터페이스 세부 조건 및 기능을 확인한다.
인터페이스 구현
인터페이스 구현

송/수신 시스템 간의 데이터 교환 및 처리를 실현해주는 작업이다.
사전에 정의된 기능 구현을 분석하고 인터페이스를 구현한다.
인터페이스 기능 구현을 기반으로 인터페이스 구현 방법을 분석하고 분석된 인터페이스 구현 정의를 바탕으로 인터페이스를 구현한다.
데이터 통신을 이용한 인터페이스 구현

애플리케이션 영역에서 인터페이스 형식에 맞춘 데이터 포맷을 인터페이스 대상으로 전송하고 이를 수신 측에서 파싱하여 해석하는 방식이다.
AJAX(Asynchronous Javascript And Xml)

js 를 사용한 비동기 통신 기술로 클라이언트와 서버 간에 XML 데이터를 주고받는 기술이다.
브라우저가 가지고 있는 XMLHttpRequest 객체를 이용해서 전체 페이지를 새로 고치지 않고도 페이지의 일부만을 위한 데이터를 로드하는 기법이다.
JSON(Javascript Object Notation)

데이터 통신을 이용한 인터페이스 구현 방법이다.
웹과 컴퓨터 프로그램에서 용량이 적은 데이터를 교환하기 위해 데이터 객체를 속성/값의 쌍 형태로 표현하는 형식으로 JS 를 토대로 개발된 형식이다.
속성/값의 쌍(Attribute-Value Pairs)인 데이터 객체 전달을 위해 사람이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷으로 비동기 처리에 쓰이는 AJAX 에서 XML 을 대체하는 주요 데이터 포맷이다.
인터페이스 구현 검증
인터페이스 구현 검증

인터페이스 구현 및 감시 도구를 통해서 구현된 인터페이스의 동작 상태를 검증 및 감시(Monitoring)할 수 있다.
검증 순서 : 구현된 인터페이스 명세서를 참조하여 구현 검증에 필요한 감시 및 도구를 준비한 뒤 인터페이스 구현 검증을 위하여 외부 시스템과의 연계 모듈 상태를 확인한다.
인터페이스 구현 검증 도구의 종류

인터페이스 구현 검증을 위해서 단위 기능 및 시나리오에 기반한 통합 테스트가 필요하며, 테스트 자동화 도구를 이용하여 단위 및 통합 테스트의 효율성을 높일 수 있다.

Watir

Ruby 기반 웹 애플리케이션 테스트 프레임워크이다.
모든 언어 기반의 웹 애플리케이션 테스트와 브라우저 호환성을 테스트할 수 있다.
xUnit

java(Junit), C++(Cppunit), .Net(Nnuit) 등 다양한 언어를 지원하는 단위 프레임워크이다.
함수, 클래스 등 다른 구성 단위 테스트를 도와준다.
FitNesse

웹 기반 테스트 케이스 설계/실행/결과 확인 등을 지원하는 테스트 프레임워크이다.
테스트 케이스 테이블을 작성하면 자동으로 빠르고 쉽게 작성한 테스트를 수행할 수 있다.
STAF

서비스 호출, 컴포넌트 재사용 등 다양한 환경을 지원하는 테스트 프레임워크이다.
데몬을 사용하여 테스트 대상 분산 환경에서 대상 프로그램을 통해서 테스트를 수행하고 통합하는 자동화 검증 도구이다.
NTAF Naver

테스트 자동화 프레임워크이며, STAF 와 FitNesse 를 통합한다.
Selenium

다양한 브라우저 및 개발 언어를 지원하는 웹 애블리케이션 테스트 프레임워크이다.
테스트를 위한 스크립트 언어 습득 업이, 기능 테스트 작성을 위한 플레이백 도구를 제공한다.
인터페이스 구현 감시 도구

APM(Application Performance Management)을 사용하여 동작 상태를 감시할 수 있다.
데이터베이스, 웹 애플리케이션의 트랜잭션과 변수값, 호출 함수, 로그 및 시스템 부하 등 종합적인 정보를 조회하고 분석할 수 있다.
ex. 스카우터, 제니퍼 등
인터페이스 구현 검증 시 필요한 설계 산출물

데이터 전송 주기, 전송 포맷 등을 확인하여 송/수신 시스템에 데이터가 정확하게 전송되었는지 인터페이스 명세서를 중심으로 확인한다.
인터페이스 단위 테스트 케이스나 통합 테스트 케이스를 활용한다.

모듈 세부 설계서(컴포넌트 명세서, 인터페이스 명세서), 인터페이스 정의서, 동적/정적 모형 설계도, 식별된 인터페이스 기능 목록, 인터페이스 데이터 표준 정의서이다.

33. 인터페이스 보안

인터페이스 보안

인터페이스 보안

모듈 컴포넌트 간 데이터 교환 시 데이터 변조/탈취 및 인터페이스 모듈 자체의 보안 취약점이 존재할 수 있다.

데이터 통신 시 데이터 탈취 위협

스니핑(Sniffing) : 네트워크 주변을 지나다니는 패킷을 엿보면서 계정(ID)과 비밀번호를 알아내는 보안 위협이다.

스푸핑(Spoofing) : 일반 사용자가 인터넷상에서 통신하는 정보를 크래커의 사이트를 통하도록 하여 비밀번호를 알아내는 보안 위협이다.

데이터베이스 암호화

데이터베이스의 기밀성을 유지하기 위해 중요 민감 데이터는 암호화한다.

대칭 키, 해시, 비대칭키 알고리즘이 사용된다.

시큐어 코딩

OWASP(Open Web Application Security Project) Top 10 을 참고하여 KISA(한국 인터넷 진흥원)에서 SW 보안 약점 가이드를 발표하였다.

SW 보안 취약점, 약점 및 대응 방안이 구체적으로 서술되어 있으며 이를 바탕으로 시큐어 코딩을 하도록 한다.

네트워크 보안 적용

인터페이스 송/수신 간 중간자에 의한 데이터 탈취 또는 위변조를 방지하기 위해서 네트워크 트래픽에 대한 암호화 적용이 요구된다.

네트워크 구간의 암호화를 위해서는 인터페이스 아키텍처에 따라서 다양한 방식으로 보안 기능을 적용한다.

네트워크 구간 보안 기능 적용 시 고려사항

단계	고려사항	보안 기능 적용
Transport Layer Network 보안 (OSI7 계층 - 전송계층)	상대방 인증을 적용한다.	IPSec AH(Authentication Header)적용, IKE(Internet Key Ex-change) 프로토콜을 적용한다.
	데이터 기밀성 보장이 필요하다.	IPSec ESP(Encapsulation Security Payload)를 적용한다.
	End-to-End 보안을 적용한다.	IPSec Transport Mode 를 적용한다.
Application Layer Network 보안		

(OSI7 계층 - 응용계층) 서버만 공개키 인증서를 가지고 통신(위험 분산)한다.
SSL(Secure Socket Layer)의 서버 인증 상태를 운영한다.
연결 단위 외 메시지 단위로도 인증 및 암호화가 필요하다. S-HTTP 를 적용하여 메시지를 암호화한다.(상호 인증 필요, 성능 일부 저하됨).
데이터베이스 보안
데이터베이스 보안 적용

데이터베이스의 기밀성 유지를 위하여 중요하고 민감한 데이터는 암호화 기법을 활용하여 암호화하도록 한다.
데이터베이스의 접근 권한 및 SQL, 프로시져, 트리거 등 데이터베이스 동작 객체의 보안 취약점을 보완하도록 한다.
민감하고 중요한 데이터는 암호화와 익명화 등을 통하여 데이터 자체 보안 방법도 고려해야 한다.
영역 : 비인가자 접근 관리, 악의적 코드 삽입 금지, 민감 데이터 관리, 악의적 시도 시 에러 처리
데이터베이스 암호화 알고리즘

대칭키 알고리즘 : ARIA 128/129/256, SEED
해시 알고리즘 : SHA-256/384/512, HAS-160
비대칭키 알고리즘 : RSA, ECDSA, ECC
데이터베이스 암호화 기법

구분 API 방식 Filter(Plug-in) 방식 Hybrid 방식
개념 애플리케이션 레벨에서 암호 모듈(API)을 적용하는 방식이다. 데이터베이스 레벨의 확장성 프로시저 기능을 이용하여 DBMS 에 Plugin 또는 Snap-in 모듈 형식으로 작성하는 방식이다. API/Filter 방식을 결합하거나, Filter 방식에 추가로 SQL 문에 대한 최적화를 대행해 주는 어플라이언스를 제공하는 방식이다.
암호화/보안 방식 별도의 API 개발/통합 DB 내 설치/연동 어플라이언스/DB 내 설치
서버 성능 부하 애플리케이션 서버에서 암호화/복호화. 정책 관리, 키 관리를 하므로 부하가 발생한다. DB 서버에서 암호화, 복호화, 정책 관리 키 관리를 하므로 부하가 발생한다. DB 와 어플라이언스에서 부하가 분산된다.
시스템 통합 용이성 애플리케이션 개발 및 통합 기간이 필요하다. 애플리케이션 변경이 필요치 않아 용이성이 높다. 애플리케이션 변경이 필요치 않아 용이성이 높다.

관리 편의성 애플리케이션 변경 및 암호화 필드를 변경하는 유지보수가 필요하다.
관리자용 GUI 를 이용하여 DB 통합 관리가 가능하여 편의성이 높다. 관리자용 GUI 를 이용하여 DB 통합 관리가 가능하여 편의성이 높다.
중요도가 높거나 민감한 정보를 통신 채널을 통하여 전송 시에는 반드시 암/복호화 과정을 거치도록 한다.
IPSec, SSL/TLS 등 보안 채널을 활용하여 전송한다.
IPSec : 네트워크에서의 안전한 연결을 설정하기 위한 통신 규칙 또는 프로토콜 세트
SSL/TLS : 공개키 기반의 국제 인터넷 표준화 기구에서 표준으로 지정한 인터넷에서 정보를 암호화해서 수신하는 프로토콜
인터페이스 연계 테스트
연계 테스트

송/수신 시스템 간 구성 요소가 정상적으로 동작하는지 테스트하는 활동이다.
진행 순서 : 연계 테스트 케이스 작성 => 연계 테스트 환경 구축 => 연계 테스트 수행 => 연계 테스트 수행 결과 검증
연계 테스트 분류

소프트웨어 연계 테스트 구간 : 송신 시스템에서 연계 서버 또는 중계 서버를 거치고 수신 시스템까지 데이터가 전달되는가를 테스트한다.
소프트웨어 연계 단위 테스트 : 연계 자체만을 테스트한다. 송신 시스템에서 연계 데이터를 추출 및 생성하고 이를 연계 테이블로 생성한다. 연계 서버 또는 중계 서버가 있는 경우 연계 테이블 간 송/수신을 한다.
소프트웨어 연계 통합 테스트 : 연계 테스트보다 큰 통합 기능 테스트의 일부로서 연계 통합 테스트를 수행한다.