**34. 자료 구조**

1. 자료 구조의 정의

효율적인 프로그램을 작성할 때 우선 고려사항 – 저장 공간의 효율성, 실행 시간의 신속성

자료구조 – 자료를 기억장치의 공간 내에 저장하는 방법, 자료 간의 관계, 처리 방법등의 연구

2. 자료 구조의 분류

선형 구조 – 배열, 선형 리스트(Linear List, 연속 리스트 / 연결 리스트), 스택, 큐, 데크(Deque)

비선형 구조 – 트리, 그래프

|  |  |
| --- | --- |
| 배열(Array) | 동일한 자료형의 데이터들이 같은 크기로 나열되어 순서를 갖고 있는 집합 |
| 선형 리스트  (Linear List) | 일정한 순서에 의해 나열된 자료구조  연속 리스트 – 배열, 효율↑ / 연결 리스트 - 포인터 |
| 스택(Stack) | 리스트의 한쪽으로만 자료의 삽입, 삭제 작업이 이루어짐(후입선출, LIFO)  TOP : 마지막 삽입 자료가 기억된 위치 Bottom : 밑바닥 |
| 큐(queue) | 삽입과 삭제가 서로 반대되는 방향에서 이루어지는 자료구조(선입선출, FIFO)  프런트(Front) 포인터 – 삭제 작업, 리어(Rear) 포인터 – 삽입 작업 |
| 트리(Tree) | 그래프의 특수한 형태, 사이클을 이루지 않도록 구성 / Node와 Link  노드, 근노드, 디그리, 단말 노드, 자식 노드, 부모 노드, 형제 노드, 트리의 디그리 |

**35 데이터저장소 / 데이터베이스 / DBMS**

1. 데이터 저장소

데이터들을 논리적인 구조로 조직화, 물리적인 공간에 구축한 것을 의미

논리 데이터저장소 – 데이터, 데이터간의 연관성, 제약 조건 식별 후 논리적인 구조로 조직화

물리 데이터저장소 – 저장된 데이터와 구조들을 소프트웨어가 운용될 환경의 물리적 특성을 고려하여 하드웨어적인 저장장치에 저장한 것을 의미

2. 데이터베이스

통합된 데이터(Integrated Data) : 자료의 중복을 배제한 데이터의 모임

저장된 데이터(Stored Data) : 컴퓨터가 접근할 수 있는 저장 매체에 저장된 자료

운영 데이터(Operational Data) : 조직의 고유한 업무를 수행하는 데 존재 가치가 확실, 필수 자료

공용 데이터(Shared Data) : 여러 응용 시스템들이 공동으로 소유하고 유지하는 자료

3. DBMS(Data Base Management System; 데이터베이스 관리 시스템)

사용자와 데이터베이스 사이에서 사용자의 요구에 따라 정보를 생성, DB 관리해 주는 소프트웨어

정의 기능 : 데이터의 형(Type), 구조에 대한 정의, 이용 방식, 제약 조건 명시 기능

조작 기능 : 데이터 검색, 갱신, 삽입, 삭제 등을 체계적으로 처리하기 위해 인터페이스 수단 제공

제어 기능 : 무결성 유지 제어, 보안 유지, 권한 검사, 병행 제어(Concurrency Control)

4. DBMS의 장 단점

|  |  |
| --- | --- |
| 장점 | 단점 |
| 논리적, 물리적 독립성 보장, 기억공간 절약  자료 공동 이용, 일관성/무결성/보안 유지  데이터 표준화/통합관리/최신 유지/실시간처리 | 전문가 없음, 전산화 비용 증가  파일의 백업과 회복이 어려움, 복잡한 시스템  디스크로의 집중적인 접근으로 과부하 발생 |

**36. 데이터 입 ∙ 출력**

1. 데이터 입 ∙ 출력의 개요

데이터베이스에 데이터를 입력하거나 출력하는 작업을 의미, 객체와 데이터를 연결 – 데이터접속

2. SQL(Structured Query Language)

관계대수와 관계해석을 기초로 한 혼합 데이터 언어, 질의어 이외의 기능도 갖추고 있음

데이터 정의어(DDL) : 스키마, 테이블, 도메인, 뷰, 목차 등을 정의, 변경, 삭제할 때 사용하는 언어

데이터 조작어(DML) : 저장된 데이터를 실질적으로 처리하는 데 사용되는 언어

데이터 제어어(DCL) : 데이터의 보안, 무결성, 회복, 병행 수행 제어 정의 언어

3. 데이터 접속(Data Mapping)

SQL Mapping : 코드 내에 SQL을 직접 입력하여 DBMS의 데이터에 데이터를 연결하는 것

ORM(Object-Relational Mapping) : 객체화 관계형 데이터베이스의 데이터를 연결 하는 기술

4. 트랜잭션(Transaction) – 논리적 기능을 수행하기 위한 작업의 단위, 일련의 연산들을 의미

TCL(Transaction Control Language) → COMMIT, ROLLBACK, SAVEPOINT(=CHECK POINT)

**37. 절차형 SQL**

1. 절차형 SQL의 개요

연속적인 실행이나 분기, 반복 등의 제어가 가능한 SQL을 의미

연속적인 작업들을 처리하는데 적합, 저장 모듈 생성, 입∙출력 패킷小, 기능별 모듈화 가능

프로시저(Procedure) : 특정 기능을 수행하는 트랜잭션 언어, 호출을 통해 실행되어 미리 저장해 놓은 SQL 작업을 실행한다.

트리거(Trigger) : 데이터의 입력, 갱신, 삭제 등의 이벤트가 발생할 때마다 관련 작업이 자동으로 수행

사용자 정의 함수 : 프로시저와 유사하게 SQL을 사용하여 일련의 작업을 연속적으로 처리, 종료시 예약어 Return을 사용하여 처리 결과를 단일값으로 반환

2. 절차형 SQL의 테스트와 디버깅

디버깅을 통해 기능의 적합성 여부를 검증 후, 실행을 통해 결과를 확인하는 테스트 과정을 수행

테스트 전 생성을 통해 구문 오류(Syntax Error)나 참조 오류의 존재 여부 확인 → SHOW 명령어를 통해 내용을 확인하고 문제를 수정 → 디버깅을 통해 로직 검증 → 결과로 최종 확인

* 삽입 및 변경 관련 SQL문을 주석으로 처리해 화면에 출력하여 확인

3. 쿼리 성능 최적화

데이터 입 ∙ 출력 애플리케이션의 성능 향상을 위해 SQL 코드를 최적화하는 것

성능 측정 도구인 APM을 사용하여 최적화할 쿼리 선정 → 옵티마이저가 수립한 실행 계획 검토→ SQL 코드와 인덱스를 재구성

**38. 단위 모듈 구현**

1. 단위 모듈(Unit Module)의 개요

한가지 동작을 수행하는 기능을 모듈로 구현한 것

단위 모듈로 구현되는 하나의 기능 – 단위 기능

사용자나 다른 모듈로부터 값을 전달받아 시작되는 작은 프로그램을 의미하기도 함

두개의 단위 모듈 – 두개의 기능 구현 가능, 구성요소 – 처리문, 명령문, 데이터 구조

독립적인 컴파일 가능, 다른 모듈에 호출되거나 삽입되기도 함

구현 요건 - 단위 기능 명세서 작성 → 입 출력 기능 구현 → 알고리즘 구현

2. 단위 기능 명세서 작성

기능 및 코드 명세서나 설계 지침과 같이 단위 기능을 명세화 한 문서들을 의미

추상화 작업 → 단위 기능별로 구분 → 기능들을 계층적으로 구성(구조화 과정) → 정보 은닉

3. 입 ∙ 출력 기능 구현

명세서에서 정의한 데이터 형식에 따라 입 ∙ 출력 기능을 위한 알고리즘 및 데이터를 구현

단위 모듈 간의 연동, 통신을 위한 입 ∙ 출력 데이터 구현, CLI/GUI 연동 고려

※IPC(Inter-Process Communication) – 인터페이스 집합

대표 메소드 – 메모리공유, 소켓, Semaphores, Pipes&named Pipes, Message Queueing

4. 알고리즘 구현

입 ∙ 출력 데이터를 바탕으로 단위 기능별 요구 사항들을 구현 가능한 언어를 이용, 모듈로 구현

디바이스 드라이버 모듈, 네트워크 모듈, 파일 모듈, 메모리 모듈, 프로세스 모듈로 구분

**39. 단위 모듈 테스트**

모듈이 정해진 기능을 정확히 수행하는지 검증하는 것, 단위 테스트(Unit Test) 라고도 함

화이트 박스 테스트 – 소스 코드를 오픈 시킨 상태에서 모든 논리적인 경로를 테스트하는 방법

블랙 박스 테스트 – 소프트웨어가 수행할 특정 기능이 완전히 작동되는 것을 입증하는 테스트

2. 테스트 케이스(Test Case)

사용자의 요구사항을 정확하게 준수했는지 확인하기 위한 테스트 항목 명세서

구성요소 – 식별자, 테스트 항목, 입력 명세, 출력 명세, 환경 설정, 특수 절차 요구, 의존성 기술

3. 테스트 프로세스

계획 및 제어 단계 – 분석 및 설계 단계 – 구현 및 실현 단계 – 평가 단계 – 완료 단계

**40. 개발 지원 도구**

1. 통합 개발 환경(IDE; Integrated Development Environment)

통합 개발 환경을 제공하는 소프트웨어, 컴파일 가능, 코드 자동생성, 수정 용이

다양한 서비스와 연동하여 개발 편의 제공, 필요 정보 공유 / ex)이클립스, 비쥬얼 스튜디오

2. 빌드 도구

소스 코드 파일을 소프트웨어로 변환하는 과정 또는 결과물, 전처리/컴파일 등의 작업 수행

대표 도구 – Ant(자바, XML기반, 재사용↓), Maven(라이브러리), Gradle(의존성 활용, 그루비 기반)

3. 기타 협업 도구

서로 다른 작업 환경에서 프로젝트를 수행할 수 있도록 도와주는 도구

프로젝트 및 일정 관리, 정보 공유 및 커뮤니케이션, 디자인, 기타(에버노트, 깃허브)

**41. 소프트웨어 패키징**

모듈별로 생성한 실행 파일들을 묶어 배포용 설치 파일을 만드는 것

2. 패키징 시 고려사항

시스템 환경 정의, UI 매뉴얼 제공, Managed Service 형태로 제공, 안정적인 배포 중요, 개선 고려

3. 패키징 작업 순서

온라인 배포 : 별도로 운영서버를 마련해 설치, 사용 매뉴얼과 함께 배포 파일 등록 후 직접 다운

오프라인 배포 : CD-ROM, DVD, USB 등에 설치 및 사용 매뉴얼과 함께 배포

기능 식별→모듈화 빌드 진행→사용자 환경 분석→패키징 및 적용 시험→패키징 변경→개선 배포

**42. 릴리즈 노트 작성**

사용자에게 제공하는 소프트웨어에 대한 정보. 버전관리, 릴리즈 정보를 체계적으로 관리 가능

2. 릴리즈 노트 초기 버전 작성 시 고려사항

헤드,개요,목적,문제요약,재현항목,수정/개선내용,사용자영향도,SW지원영향도,노트,면책조항,연락처

3. 릴리즈 노트 추가 버전 작성 시 고려사항

베타 버전 출시, 긴급 버그 수정, 자체 기능 향상, 사용자 요청 등의 상황 발생 시 추가 작성

4. 릴리즈 노트 작성 순서

모듈 식별 → 릴리즈 정보 확인 → 릴리즈 노트 개요 작성 → 영향도 체크 →정정식 릴리즈 노트 작성 → 추가 개선 항목 식별

**43. 디지털 저작권 관리(DRM)**

1. 저작권의 개요

저작물에 대하여 창작자가 가지는 배타적 독점적 권리, 타인의 침해를 받지 않을 고유 권한

저작권 보호 기술 – 복제가 쉬운 저작물에 대해 불법 복제 및 배포 등을 막는 기술적인 방법

2. 디지털 저작권 관리 (DRM; Digital Right Management)의 개요

아날로그 – 디지털로 변환 후 DRM패키징 수행 / 크기↓↑-실시간 패키징수행/미리 패키징 수행

패키징 수행 시 전자 서명 포함 및 라이선스 정보가 클리어링 하우스에 등록

3. 디지털 저작권 관리의 흐름도

클리어링 하우스(Clearing House) : 저작권 사용 권한, 라이선스 발급, 사용량에 따른 결제 관리

콘텐츠 제공자(Contents rovider), 콘텐츠 분배자(Contents Distributor), 소비자(Custumer)

패키저(Packager) : 콘텐츠를 메타 데이터와 함께 배포 가능한 형태로 묶어 암호화하는 프로그램

DRM컨트롤러(DRM Controllor) : 배포된 콘텐츠의 이용 권한을 통제하는 프로그램

보안 컨테이너(Security Container) : 콘텐츠 원본을 안전하게 유통하기 위한 전자적 보안 장치

4. 디지털 저작권 관리의 기술 요소

암호화(Encryption), 키 관리, 암호화 파일 생성(Packager), 식별 기술, 저작권 표현(Right Expression), 정책 관리(Policy Management), 크랙 방지(Tamper Resistance), 인증(Authentication)

**44. 소프트웨어 설치 매뉴얼 작성**

초기 기준이나 설치하는 과정에 필요한 내용을 기록한 설명서와 안내서

2. 서문

문서 이력, 설치 매뉴얼의 주석(주의 사항, 참고 사항), 설치 도구의 구성, 설치 환경 체크 항목

3. 기본 사항

소프트웨어 개요, 설치 관련 파일, 설치 아이콘(Installation), 프로그램 삭제, 관련 추가 정보

4. 설치 매뉴얼 작성 방법

설치 화면을 누락 없이 캡처하고 순서대로 상세히 설명

설치 화면 및 UI(설치 실행, 메인 화면 및 안내창), 설치 이상 메시지 설명, 설치 완료 및 결과, FAQ, 설치 시 점검 사항, Network 환경 및 보안, 고객 지원 방법, 준수 정보 & 제한 보증

5. 설치 매뉴얼 작성 순서

기능 식별→UI 분류→설치/백업 파일 확인→Uninstall 절차 확인→이상 Case 확인→최종 매뉴얼

**45. 소프트웨어 사용자 매뉴얼 작성**

소프트웨어를 사용하는 과정에서 필요한 내용을 문서로 기록한 설명서와 안내서

2. 서문

문서 이력, 사용자 매뉴얼의 주석(주의 사항, 참고 사항), 기록 보관 내용

3. 기본 사항

소프트웨어 개요/사용 환경/관리/구동환경, 모델/버전별/기능/인터페이스의 특징

4. 사용자 매뉴얼 작성 방법

사용자 화면 및 UI(주의사항, 참고 사항), 주요 기능 분류, 응용 프로그램 및 설정, 장치 연동, Network 환경, Profile 안내, 고객 지원 방법, 준수 정보 & 제한 보증

5. 사용자 매뉴얼 작성 순서

기능 식별→화면 분류→사용자 환경 파일 확인→초기화 절차 확인→이상 Case 확인→최종 매뉴얼

**46. 소프트웨어 버전 등록**

1. 소프트웨어 패키징의 형상 관리(SCM; Sofrtware Configuration Management)

소프트웨어의 개발 과정에서 소프트웨어의 변경 사항을 관리하기 위해 개발된 일련의 활동

2. 형상 관리의 중요성

변경 사항 체계적 추적/통제 가능, 무절제한 변경 반지, 버그나 수정사항 추적, 진행 확인 기준

3. 형상 관리 기능

형상 식별, 버전 제어, 형상 통제(변경 관리), 형상 감사, 형상 기록(상태 보고)

4. 소프트웨어의 버전 등록 관련 주요 용어

저장소(Repository), 가져오기(Import), 체크아웃, 체크인, 커밋(Commit), 동기화(Update)

5. 소프트웨어 버전 등록 과정

가져오기(Import) → 인출(Check-Out) → 예치(Commit) → 동기화(Update) → 차이(Diff)

**47. 소프트웨어 버전 관리 도구**

1. 공유 폴더 방식

버전 관리 자료가 로컬 컴퓨터의 공유 폴더에 저장되어 관리되는 방식

개발자 – 개발 완료 파일을 공유 폴더에 매일 복사

담당자 – 복사 후 컴파일 하여 이상 유무 확인. 오류 발생 시 개발자에게 수정 의뢰

실수 대비 파일 변경 사항을 데이터베이스에 기록하여 관리, 종류 – SCCS, RCS, PVCS, QVCS

2. 클라이언트/서버 방식

버전 관리 자료가 중앙 시스템에 저장되어 관리되는 방식

서버의 자료를 개인 PC에 복사하여 작업한 후 변경된 내용을 서버에 반영

모든 버전 관리는 서버에서 수행, 동시에 동일 파일 작업 시 경고 메시지 출력, 버전 관리 서버

서버 문제 발생 시 복구되기 전까지 협업 및 버전 관리 작업 중단

종류 – CVS, SVN(Subversion), CVSNT, Clear Case, CMVC, Perforce

3. 분산 저장소 방식

버전 관리 자료가 하나의 원격 저장소와 개인 PC의 로컬 저장소에 함께 저장되어 관리되는 방식

원격 저장소의 자료를 개인 PC에서 작업한 후 변경된 내용을 우선적으로 반영하고 업로드

원격 저장소에 문제가 생겨도 로컬 저장소의 자료를 이용하여 작업가능

4. Subversion(서브버전, SVN)

CVS를 개선한 것

명령어 – add, commit, update, checkout, lock/unlock, import, export, info. Diff, merge

5. Git(깃)

리눅스 커널 개발에 사용할 관리 도구로 개발한 이후 주니오 하마노에 의해 유지 보수 중

명령어 – add, commit, branch, checkout, merge, init, remote add, push, fetch, clone, fork

**48. 빌드 자동화 도구**

빌드를 포함하여 테스트 및 배포를 자동화하는 도구

애자일 환경에서는 하나의 작업이 마무리될 때마다 모듈 단위로 나눠서 개발된 코드들이 지속적으로 통합됨 – 지속적인 통합(Continuous Integration)개발 환경

Ex) Ant, Make, Maven, Gradle, Jekins

2. Jenkins

JAVA 기반의 오픈 소스, 가장 많이 사용되는 빌드 자동화 도구

서버 기반 도구, 형상 관리 도구 연동 가능, Web GUI 제공, 분산 빌드나 테스트가 가능

3. Gradle

Groovy 기반 자동화 도구, 안드로이드 앱 개발 환경에서 사용

플러그인을 설정해 다양한 언어로 빌드 가능

DSL(Domain Specific Language)을 스크립트 언어로 사용(영역, 용도에 맞게 기능을 구성한 언어)

실행할 처리 명령들을 모아 태스크로 만든 후 태스크 단위로 실행

빌드 캐시 기능 – 이전에 사용했던 태스크 재사용, 다른 시스템의 태스크 공유, 빌드 속도 향상

**49. 애플리케이션 테스트**

애플리케이션에 잠재되어 있는 결함을 찾아내는 일련의 행위 또는 절차

고객의 요구사항을 만족시키는지 확인(Validation), 기능을 정확히 수행하는지 검증(Verification)

상용 소프트웨어 – 보통의 사용자들이 공통적으로 필요로 하는 기능 제공

Ex) 산업 범용 소프트웨어(시스템 소프트웨어, 미들웨어, 응용 소프트웨어), 산업 특화 소프트웨어

서비스 제공 소프트웨어 – 특정 사용자가 필요로 하는 기능만을 구현해서 제공

Ex) 신규개발 소프트웨어, 기능개선 소프트웨어, 추가 개발 소프트웨어, 시스템 통합 소프트웨어

2. 애플리케이션 테스트의 필요성

프로그램 실행 전에 오류 발견하여 예방 가능, 반복 테스트로 신뢰도 향상, 오류 유입 예방

3. 애플리케이션 테스트의 기본 원리

완벽한 테스팅은 불가능, 잠재적인 결함↓, 결함은 대부분 특정 모듈에 집중됨(파레토 법칙)

테스트↑위험↓, 작은 부분에서부터 진행, 별도의 팀에서 수행, 지속적인 테스트 케이스 개선 필요

**50. 애플리케이션 테스트의 분류**

1. 프로그램 실행 여부에 따른 테스트

정적 테스트 – 명세서나 소스 코드를 대상으로 분석, 개발 초기에 결함 발견 가능, 개발 비용↓

동적 테스트 – 프로그램을 실행하여 오류를 찾음, 개발의 모든 단계에서 테스트 진행 가능

2. 테스트 기반(Test Bases)에 따른 테스트

|  |  |
| --- | --- |
| 명세 기반 | 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 구현중인지 확인하는 것 |
| 구조 기반 | 소프트웨어 내부의 논리 흐름에 따라 테스트 케이스를 작성하고 확인 |
| 경험 기반 | 테스터의 경험을 기반으로 수행하는 테스트, 명세 불충분, 시간 제약시 수행 |

3. 시각에 따른 테스트

검증(Verification) – 개발자의 시각에서 제품의 생산 과정 테스트, 명세서대로 작성됐는지를 테스트

확인(Validation) – 사용자의 시각, 요구한대로 완성됐는지, 정상적으로 동작하는지를 테스트

4. 목적에 따른 테스트

회복(Recovery), 안전(Safe), 성능(Perfomance), 구조(Structure), 회귀(Regression), 병행(Parallel)

**51. 테스트 기법에 따른 애플리케이션 테스트**

1. 화이트박스 테스트(White Box Test)

원시코드를 오픈 시킨 상태에서 논리적인 모든 경로를 테스트하여 테스트 케이스 설계하는 방법

2. 화이트박스 테스트의 종류

기초 경로 검사 – 절차적 설계의 논리적 복잡성 측정 가능, 실행 경로의 기초를 정의하는데 사용

제어 구조 검사 – 조건 검사(Condition Testing), 루프 검사, 데이터 흐름 검사(Data Flow Testing)

3. 화이트박스 테스트의 검증 기준

테스트 케이스들이 테스트에 얼마나 적정한지를 판단하는 기준

문장 검증 기준(Statement Coverage) – 소스 코드의 모든 구문이 한번 이상 수행되도록 설계

분기 검증 기준(Branch Coverage) – 소스 코드의 모든 조건문이 한번 이상 수행되도록 설계

조건 검증 기준(Condition) – 모든 조건문에 대해 True인 경우와 False인 경우 한번 이상 수행

분기/조건 기준(Branch/Condition) – 모든 조건문, 조건문에 포함된 개별 조건식의 결과가 True인 경우와 False인 경우가 한번 이상 수행되도록 테스트 케이스 설계

4. 블랙박스 테스트

소프트웨어가 수행할 특정 기능을 알기 위해 각 기능이 완전히 작동되는 것을 입증(기능 테스트)

5. 블랙박스 테스트의 종류

동치 분할 검사, 경계값 분석, 원인-효과 그래프 검사, 오류 예측 검사, 비교 검사

※검증 기준(Coverage)의 종류 – 기능 기반 커버리지, 라인 커버리지, 코드 커버리지

**52. 개발 단계에 따른 애플리케이션 테스트**

소프트웨어의 개발 단계에 따라 분류되며 테스트 레벨이라 지칭함. 다양한 오류 발견 가능,V-모델

2. 단위 테스트(Unit Test)

코딩 직후 소프트웨어 설계의 최소 단위인 모듈이나 컴포넌트에 초점을 맞춰 테스트

인터페이스, 외부적 I/O, 자료 구조, 독립적 기초 경로, 오류 처리 경로, 경계 조건

구조기반 테스트(화이트박스, 제어흐름, 조건 설정), 명세 기반 테스트(블랙박스, 동등 분할, 경계값)

3. 통합 테스트(Integration Test) - 상호 작용 오류 검사

단위 테스트가 완료된 모듈들을 결합하여 하나의 시스템으로 완성시키는 과정에서의 테스트

4. 시스템 테스트(System Test)

개발된 소프트웨어가 해당 컴퓨터 시스템에서 완벽하게 수행되는가를 점검하는 테스트

기능적 요구사항 – 명세서 기반의 블랙박스 테스트 시행

비기능적 요구사항 – 구조적 요소에 대한 화이트박스 테스트 시행(성능, 회복, 메뉴 구조)

5. 인수 테스트(Acceptance Test)

개발한 소프트웨어가 사용자 요구사항을 충족하는지에 중점을 두고 테스트

사용자 인수 테스트, 운영상의 인수 테스트, 계약 인수/규정 인수 테스트, 알파/베타 테스트

**53. 통합 테스트(Integration Test)**

단위 테스트가 끝난 모듈을 통합하는 과정에서 발생하는 오류 및 결함을 찾는 테스트 기법

비점진적 통합 방식(프로그램 전체 테스트), 점진적 통합 방식(단계적으로 통합하면서 테스트)

2. 하향식 통합 테스트(Top Down Integration Test)

상위 모듈 → 하위 모듈 통합, 깊이/넓이 우선 통합법

제어모듈-작성 프로그램 사용, 주요 제어 모듈의 종속 모듈들은 스텁(Stub)으로 대체 → 통합 방식에 따라 스텁들이 실제 모듈로 교체 → 모듈 통합 때마다 테스트 실시 → 회귀 테스트 실시

3. 상향식 통합 테스트(Bottom Up Integration Test)

하위 모듈 → 상위 모듈 통합, 하나의 주요 제어 모듈과 관련된 종속 모듈 그룹 클러스터 필요

하위 모듈들을 클러스터로 결합 → 입∙출력 확인 위해 더미 모듈인 드라이버 작성 → 통합된 클러스터 단위로 테스트 → 클러스터는 상위로 이동해 결합, 드라이버는 실제 모듈로 대체

4. 혼합식 통합 테스트

하위 수준 → 상향식 통합, 상위 수준 → 하향식 통합을 사용. 최적의 테스트 지원

5. 회귀 테스팅(Regression Testing)

이미 테스트된 프로그램의 테스팅을 반복, 통합 테스트로 변경된 모듈이나 컴포넌트 오류 확인

선정방법 – 모든 기능이 수행가능한 대표 테스트 케이스 선정, 파급효과 분석 후 선정, 실제 수정이 발생한 모듈 또는 컴포넌트에서 시행하는 테스트케이스 선정

**54. 애플리케이션 테스트 프로세스**

개발된 소프트웨어가 사용자의 요구대로 만들어졌는지, 결함은 없는지 등을 테스트하는 절차

|  |  |
| --- | --- |
| 테스트 계획 | 목표 정의 및 대상/범위 결정, 시스템 구조 파악, 조직 및 비용 산정 |
| 테스트 분석 및 디자인 | 목적과 원칙 검토, 요구사항 분석, 우선순위 결정, 도구 등 준비 |
| 케이스, 시나리오 작성 | 설계 기법을 따름, 검토 및 확인 후 시나리오 작성, 스크립트 작성 |
| 테스트 수행 | 테스트 환경 구축 후 테스트 수행, 테스트 결과 측정 및 기록 |
| 결과 평가 및 리포팅 | 테스트 결과를 비교 분석하여 테스트 결과서 작성, 결함 위주 |
| 결합 추적 및 관리 | 처리 시간 단축, 결함 재발 방지 / 에러 발견, 에러 등록, 에러 분석, 결함 확정, 결함 할당, 결함 조치, 결함 조치 검토 및 승인 |

테스트 계획서, 테스트 케이스, 테스트 시나리오, 테스트 결과서

**55. 테스트 케이스 / 테스트 시나리오 / 테스트 오라클**

1. 테스트 케이스(Test Case)

입력 값, 실행 조건, 기대 결과 등으로 구성된 테스트 항목에 대한 명세서, 시스템 설계 시 작성

2. 테스트 케이스 작성 순서

계획 검토 및 자료 확보 → 위험 평가 및 우선순위 결정 → 테스트 요구사항 정의 →구조 설계 및 테스트 방법 결정 → 테스트 케이스 정의 → 테스트 케이스 타당성 확인 및 유지 보수

3. 테스트 시나리오(Test Scenario) - 테스트 케이스들을 적용시에 구체적인 절차를 명세한 문서

4. 테스트 시나리오 작성시 유의사항

시스템, 모듈, 향목 동으로 분리 후 작성, Use Case 간 업무흐름 테스트, 동작 테스트(모듈 등)

5. 테스트 오라클(Test Oracle)

테스트 결과가 올바른지 판단하기 위해 정의된 참 값을 대입하여 비교하는 기법 및 활동

제한된 검증, 수학적 기법, 자동화 가능

6. 테스트 오라클의 종류

참(True), 샘플링(Sampling), 추정(Heuristic), 일관성 검사(Consistent)

**56. 테스트 자동화 도구**

테스트 절차를 스크립트 형태로 구현하는 자동화 도구를 적용해 쉽고 효율적으로 테스트 수행

장점 – 인력, 시간↓, 품질 보장, 평가 기준 제공, 정밀 테스트 가능, 일관성 있게 검증

단점 – 사용방법 학습 필요, 시간/비용/노력 필요, 비공개 상용 도구 고가 비용 필요

3. 테스트 자동화 수행 시 고려 사항

재사용 및 측정이 불가능한 프로그램 제외, 용도에 맞는 적절한 도구 선정, 일정 계획

4. 테스트 자동화 도구의 유형

정적 분석 도구(프로그램x), 테스트 실행 도구(스크립트언어), 성능 테스트 도구, 테스트 통제 도구, 테스트 하네스 도구

5. 테스트 수행 단계별 테스트 자동화 도구

테스트 계획 – 요구사항 관리, 테스트 분석/설계 – 테스트 케이스 작성

테스트 수행 – 테스트 자동화, 정적 분석, 동적 분석, 성능 테스트, 모니터링

테스트 관리 – 커버리지 분석, 형상 관리, 결함 추적/관리

**57. 결함 관리**

2. 결함 관리 프로세스

결함 관리 계획→기록→검토→수정→재확인→상태 추적 및 모니터링 활동→분석 및 보고서 작성

3. 결함 상태 추적

결함 분포(속성 수 측정), 결함 추세(진행 시간 결함 수 추이 분석), 결함 에이징(지속 시간 측정)

4. 결함 추적 순서

등록 → 검토 → 할당 → 수정(Resolved) → 보류(Differred) → 종료(Closed) → 해제(Clarified)

5. 결함 분류 – 시스템 결함, 기능 결함(기획, 설계 유입), GUI 결함(화면설계에서 발생), 문서 결함

6. 결함 심각도 – High(프로세스 진행 불가 결함), Midium(시스템 흐름 영향), Low(시스템 영향x)

7. 결함 우선순위 – 결함 처리에 대한 신속성을 나타내는 척도

8. 결함 관리 도구 – Mantis, Trac, Redmine, Bugzilla

**58. 애플리케이션 성능 분석**

1. 애플리케이션 성능

사용자 요구 기능을 최소한의 자원을 사용해 최대한 많은 기능을 신속하게 처리하는 정도

처리량(Throughput), 응답 시간(Response Time), 경과 시간(Tum Around Time), 자원 사용률

2. 성능 테스트 도구

앱에 부하나 스트레스를 가하면서 성능 측정 지표를 점검하는 도구, JMeter, LoadUI, OpenSTA

3. 시스템 모니터링(Monitoring) 도구 - 시스템 자원의 사용량을 확인하고 분석하는 도구

성능 저하의 원인 분석, 시스템 부하량 분석, 사용자 분석 / 종류 – Scouter, Zabbix

4. 애플리케이션 성능 저하 원인 분석

Connection 객체를 생성하거나 쿼리를 실행하는 앱 로직에서 많이 발생

데이터 요청 과다, 타임 아웃, 연결 누수, Commit 과다 발생, 읽기 오류, 데이터 손실

**59. 애플리케이션 성능 개선**

1. 소스 코드 최적화

클린 코드(Clean Code) : 잘 작성된 코드 / 나쁜 코드(Bad Code) : 스파게티 코드, 로직 중복 코드

작성 원칙 : 가독성, 단순성, 의존성 배제, 중복성 최소화, 추상화

2. 소스 코드 최적화 유형

클래스 분할 배치(응집도↑, 크기↓), 느슨한 결합(인터페이스 클래스 이용), 코딩 형식 준수, 좋은 이름 사용, 적절한 주석문 사용

3. 소스 코드 품질 분석 도구

정적 분석 도구 : 실행x, 코딩 표준, 스타일, 결함 등을 확인

동적 분석 도구 : 소스 코드를 실행하여 코드에 존재하는 메모리 누수, 스레드 결함 등을 분석

4. 소스 코드 품질 분석 도구의 종류

Pmd, cppcheck, SonarQube, checkstyle, ccm, cobertura, Avalanche, Valgrind

**60. 모듈 간 공통 기능 및 데이터 인터페이스 확인**

확인 순서 - 모듈 별 기능 확인 → 공통 기능과 데이터의 인터페이스 확인

2. 인터페이스 설계서

시스템들의 데이터 교환, 처리를 위해 교환 데이터, 관련 업무, 송∙수신 시스템에 관한 내용 정의

일반적인 인터페이스 설계서 – 각 기능의 세부 인터페이스 정보를 정의한 문서

시스템 인터페이스 설계서 – 상세 데이터 명세 정의, 상세 기능별 인터페이스 설계서

정적 ∙ 동적 모형을 통한 인터페이스 설계서 – 다이어그램 이용

3. 인터페이스 설계서 별 모듈 기능 확인

외부 모듈 : 송신, 전달, 오퍼레이션, 사전 조건, 인터페이스 / 내부 모듈 : 수신, 사후 조건, 기타

4. 모듈 간 공통 기능 및 데이터 인터페이스 확인

Ex) 외부 모듈 : 급여 계산, 급여 결과 확인 / 내부 모듈 : 전표 결과 / 공통 : 전표 발생

**61. 모듈 연계를 위한 인터페이스 기능 식별**

(내, 외부)모듈 간 데이터의 교환을 위해 관계를 설정하는 것

EAI(Enterprise App Integration) – 앱 및 플랫폼 간의 상소 연동 제공, 효율성↑, 확장성 증대

유형 – Point-to-Point, Hub & Spoke, Message Bus, Hybrid

ESB(Enterprise Service Bus) – 표준 기반의 인터페이스 제공, 서비스 중심의 통합 지향, 앱결합도↓

2.모듈 간 연계 기능 식별 - 모듈과 연계된 기능을 시나리오 형태로 구체화하여 식별

3. 모듈 간 인터페이스 기능 식별

식별된 모듈 간 관련 기능을 검토하여 인터페이스 동작에 필요한 기능을 식별

**62. 모듈 간 인터페이스 데이터 표준 확인**

모듈 간 인터페이스에 사용되는 데이터의 형식을 표준화하는 것

데이터 인터페이스, 데이터 표준 확인→기능→인터페이스 표준을 검토하여 최종적인 표준 확인

2. 데이터 인터페이스 확인

식별된 데이터 인터페이스에서 입∙출력값의 의미와 데이터의 특성 등을 구체적으로 확인

3. 인터페이스 기능 확인

인터페이스 기능 구현을 위해 필요한 데이터 항목 확인, 수정∙추가∙삭제될 항목이 있는지 확인

4. 인터페이스 데이터 표준 확인

데이터 표준과 항목들을 검토하여 최종적으로 데이터 표준을 확인, 출처를 구분해 기록

**63. 인터페이스 기능 구현 정의**

구현을 위해 인터페이스 기능에 대한 구현 방법을 기능별로 기술한 것

순서 – 컴포넌트 명세서 확인 → 인터페이스 명세서 확인 → 기능 구현 정의 → 정형화

2. 모듈 세부 설계서

모듈의 구성 요소와 세부적인 동작 등을 정의한 설계서

컴포넌트 명세서 – 개요, 내부 클래스의 동작, 외부와 통신하는 명세 등을 정의

인터페이스 명세서 – 컴포넌트 명세서의 항목 중 인터페이스 클래스의 세부 조건 및 기능 정의

3. 모듈 세부 설계서 확인 – 컴포넌트, 인터페이스 명세서를 기반으로 필요 기능을 확인

4. 인터페이스 기능 구현 정의 – 기능, 데이터표준, 모듈 세부 설계서를 기반해 필요 기능 확인

일관성 있는 인터페이스 기능 구현 정의, 정의된 인터페이스 기능 구현 정형화

**64. 인터페이스 구현**

송 ∙ 수신 시스템 간의 데이터 교환 및 처리를 실현해 주는 작업

2. 데이터 통신을 이용한 인터페이스 구현

app영역에서 데이터 포맷을 인터페이스 대상으로 전송, 수신측에서 Parsing하여 해석하는 방식

3. 인터페이스 엔티티를 이용한 인터페이스 구현

인터페이스가 필요한 시스템 사이에 별도의 인터페이스 엔티티를 두어 상호 연계하는 방식

**65. 인터페이스 예외 처리**

구현된 인터페이스가 동작하는 과정에서 기능상 예외 상황이 발생했을 때 이를 처리하는 절차

2. 데이터 통신을 이용한 인터페이스 예외 처리

동작이 실패했을 때, 송∙수신 시 발생할 수 있는 예외 케이스를 정의하고 예외 처리 방법을 기술

인터페이스 객체 송∙수신 실패 시 예외 처리 방안

|  |  |  |  |
| --- | --- | --- | --- |
|  | 시스템 환경 | 데이터 | 프로그램 자체 원인 |
| 송신 | 네트워크 불안정(404) | 크기, 정합성 체크 오류 발생 | 데이터 생성/처리 시 프로세스의 논리적 결함.  결함 수정, 테스트로 사전 예방, 사용자에게 알람. |
| 수신 | 네트워크, 서버 불안정.  입력 대기 큐 요청 적재 및 순차적 처리 | 특수문자 등으로 파싱 시 오류 발생, 문자 임의 대치, 이후 다시 처리 |

3. 인터페이스 엔티티를 이용한 인터페이스 예외 처리

인터페이스의 실패 상황과 원인 등을 기록, 조취를 취할 수 있도록 사용자 및 관리자에서 알림

인터페이스 데이터 생선, 테이블 입력, 전송, 읽기, 트랜잭션, 처리 결과 응답

**66. 인터페이스 보안**

정보 교환으로 인해 보안 취약점이 될 수 있음, 보안 취약점 분석 후 적절한 보안 기능 적용

2. 인터페이스 보안 취약점 분석

각 구간의 구현 현황 확인 후 어떤 보안 취약점이 있는지를 분석 → 구현 기술 및 특징 구체적 확인 → 영역별로 발생할 수 있는 보안 취약점을 시나리오 형태로 작성

3. 인터페이스 보안 기능 적용

네트워크 영역(암호화), 애플리케이션 영역(코드 보안), 데이터베이스 영역(동작 개체 보안)

**67. 연계 테스트**

구축된 연계 시스템과 구성 요소가 정상적으로 동작 하는지 확인하는 화동

2. 연계 테스트 케이스 작성

연계 시스템 간의 데이터 및 프로세스의 흐름을 분석하여 필요한 테스트 항목을 도출하는 과정

송 ∙ 수신용 연계 응용 프로그램의 단위 테스트 케이스, 연계 테스트 케이스

3. 연계 테스트 환경 구축 – 테스트의 일정, 방법, 절차, 소요 시간 들을 협의를 통해 결정하는 것

4. 연계 테스트 수행 - 연계 테스트 케이스의 시험 항목 및 처리 절차 등을 실제로 진행하는 것

단위 테스트 선 수행 → 데이터 추출, 데이터 송∙수신, 데이터 반영 과정 등의 연계 테스트 수행

5. 연계 테스트 수행 결과 검증

연계 테스트 케이스의 시험 항목 및 처리 절차를 수행한 결과가 예상 결과와 동일한지를 확인하는 것

검증 방법 – DB 테이블 건수 확인, 파일 오픈, 파일 생성 여부 및 크기 확인, 모니터링, 로그

**68. 인터페이스 구현 검증**

인터페이스가 정상적으로 문제없이 작동하는지 확인하는 것, 구현 검증 도구와 감시 도구 이용

2. 인터페이스 구현 검증 도구 - xUnit, STAF, FitNesse, NTAF, Selenium, watir

3. 인터페이스 구현 감시 도구

인터페이스 동작 상태는 APM을 사용해 모니터링, 성능 관리 도구로 종합적인 정보 조회, 분석

APM – 접속자, 자원 현황, 트랜잭션 수행 내역, 장애 진당 등의 모니터링 기능을 제공하는 도구

4. 인터페이스 구현 검증 도구 및 감시 도구 선택

인터페이스 명세서의 세부 기능을 참조하여 정상적인 동작 여부를 확인하기 위한 도구 요건 분석

5. 인터페이스 구현 검증 확인

검증 도구를 이용해 외부 시스템과 연계 모듈의 동작 상태 확인, 단계별 오류 처리 구현 확인

전반적인 인터페이스 동작 프로세스 상에서 예상되는 결과값과 실제 검증값이 동일한지 비교

6. 인터페이스 구현 감시 확인

감시 도구를 이용해 외부 시스템과 연결 모듈이 서비스를 제공하는 동안 정상적으로 동작하는지 확인. 인터페이스 동작여부, 에러 발생 여부 등의 제공 리포트를 활용

**69. 인터페이스 오류 확인 및 처리 보고서 작성**

오류 발생 시 확인 후 오류 처리 보고서 작성, 보고 체계에 따라 관리 조직에 보고

2. 인터페이스 오류 발생 즉시 확인

오류 발생 시 오류 메시지 표시 후 SMS, 이메일 발송(즉시 확인 가능), 가장 직관적인 방법

3. 주기적인 인터페이스 오류 발생 확인

시스템의 현재 상태를 보여주는 시스템 로그나 오류 관련 테이블을 통해 주기적으로 확인

오류 로그 확인, 테이블 확인, 감시 도구 사용(APM)

4. 인터페이스 오류 처리 보고서 작성

오류의 발생 및 종료 시점, 원인 및 증상, 처리사항 등을 정리한 문서

(최초 발생)오류 발생 상황 인지 후 조직에 보고, 대응 조직 설립 → 오류 발생 구간, 시점, 영향도 등을 간이 보고서, 이메일, SMS 등으로 보고 → (처리 경과)진행 상황과 공지사항 등록 등 보고 → (완료)최종 조치 후 내부 조직, 고객사에 보고 → 종합적인 내용 보고