1. 개발환경 구축

* 개발환경 구축

1. 개발환경 구축의 개념

구현될 시스템 요구사항의 명확한 이해가 필요하다

개발 도구와 서버의 선정이 이루어져야 하고, 개발에 사용되는 도구들의 사용 편의성과 성능, 라이선스를 확인한다

1. 개발 도구의 분류

* 빌드 도구

작성한 코드의 빌드 및 배포를 수행하는 도구

* 구현 도구

개발자의 코드 작성과 디버깅, 수정 등과 같은 작업을 지원하는 도구

* 테스트 도구

코드의 기능 검증과 전체의 품질을 높이기 위해 사용하는 도구

* 형상 관리 도구

개발자들이 작성한 코드와 리소스 등 산출물에 대한 버전 관리를 위한 도구

1. 개발환경 구성요소
2. 하드웨어 개발환경

* 서버 하드웨어 개발 환경

프로젝트 구성에 따라 웹 서버, 웹 애플리케이션 서버, 데이터베이스 서버, 파일 서버로 구분한다

* 클라이언트 하드웨어 개발 환경

서버 개발환경에서 제공된 서비스를 사용하기 위해 UI를 제공한다

1. 소프트웨어 개발환경

개발을 위한 기본적인 소프트웨어 개발환경을 선택 및 구성한다

프로젝트 요구사항에 부합한 운영체제, 미들웨어, 데이터베이스 시스템을 선정한다

1. 형상 관리

소프트웨어 개발을 위한 전체 과정에서 발생하는 모든 항목의 변경 사항을 관리하기 위한 활동이다

형상 식별 -> 형상 통제 -> 형상 감사 -> 형상 기록

SW 생명 주기 동안 형상 관리를 통해 산출물을 체계적으로 관리하여 SW의 가시성, 추적성, 무결성 등의 품질 보증을 보장할 수 있다

* 형상 관리의 목적

프로젝트 생명주기 동안 제품의 무결성과 변경에 대한 추적성을 확보 할 수 있다

* 형상 식별

형상 관리 대상을 정의 및 식별하는 활동

추적성 부여를 위해 ID 와 관리 변호를 부여

* 형상 통제

변경요구 관리, 변경제어, 형상 관리 등의 통제를 지원하는 활동

형상 항목의 버전 관리를 위한 형상통제위원회 운영

* 형상 감사

소프트웨어 베이스라인의 무결성을 평가하는 활동

* 형상 기록

소프트웨어 형상 및 병경관리에 대한 각종 수행결과를 기록하는 활동

* 소프트웨어 형상 관리 도구

공유 폴더 방식 ( RCS, SCCS )

클라이언트 / 서버 방식 ( CVS, SVN )

분산 저장소 방식 ( GIT )

* 소프트웨어 형상 관리 도구별 특징

RCS 는 CVS 와 달리 소스 파일의 수정을 한 사람만으로 제한 하여 다수의 사람이 파일의 수정을 동시에 할 수 없도록 파일 잠금 방식으로 형상을 관리

CVS 는 가장 오래된 형상 관리 도구

변경 사항 동기화, 파일 단위 관리

SVN 은 중앙 집중형 클라이언트 – 서버 방식으로 CVS 의 단점을 보완한 형상관리 도구

GIT 은 리누스 토발즈가 2005년 리눅스 커널의 개발을 위해 만든 형상 관리 시스템

1. 백업

중요한 데이터나 정보를 안전하게 보관하기 위한 데이터를 복제하는 기법

* 전체 백업 : 백업 받고자 하는 데이터 전체에 대해 백업
* 차등 백업 : 마지막 전체 백업 이후 변경된 모든 데이터를 백업
* 증분 백업 : 정해진 시간을 기준으로 그 이후 변경된 파일만을 백업

1. 모듈 구현

* 모듈 구현

1. 재사용
2. 개념

재사용은 목표 시스템의 개발 기간 및 비용 절감을 위하여 검증된 기능을 파악하고 재구성하여 시스템에 응용하기 위한 최적화 작업

1. 종류

* 재공학 : 기존 소프트웨어를 버리지 않고 기능을 개성 시키거나 기능을 새로운 소프트웨어로 재활용
* 재개발 : 기존 시스템 내용을 참조하여 완전히 새로운 시스템을 개발

1. 모듈 구현
2. 모듈의 개념

그 자체로 하나의 완전한 기능을 수행할 수 있는 독립된 실체이다

1. 모듈의 특징

독립성이 높은 모듈일 수록 모듈 수정 시에도 다른 모듈들에는 영향을 거의 미치지 않고 오류 발생 시에도 쉽게 해결할 수 있다

독립성은 결합도와 응집도에 측정되며 독립성을 높이려면 결합도는 약하게, 응집도는 강하게, 크기는 작게

1. 모듈화의 개념

소프트웨어의 성능을 향상시키거나 시스템 수정, 재사용들이 용이하도록 기능 단위의 모듈로 분해 하는 설계 및 구현 기법

1. 모듈화 기법

* 루틴 : 특정 동작을 수행하는 일련의 코드로 기능을 가진 명령들의 모임
* 메인 루틴 : 전체의 개략적인 동작 절차를 표시하도록 만들어진 루틴
* 서브 루틴 : 필요할 때마다 호출되는 루틴

1. 소프트웨어 모듈 응집도
2. 개념

모듈의 독립성을 나타내는 정도, 모듈 내부 구성요소 간 연관 정도이다

하나의 모듈은 하나의 기능을 수행할수록 응집도가 높다

1. 유형

응집도의 순서대로 응집도가 높아진다

* 우연적 : 모듈 내부의 각 구성요소가 연관이 없을 경우의 응집도
* 논리적 : 유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들이 한 모듈에서 처리되는 경우의 응집도
* 시간적 : 연관된 기능이라기보다는 특정 시간에 처리되어야 하는 활동들을 한 모듈에서 처리할 경우의 응집도
* 절차적 : 모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성요소들이 그 기능을 순차적으로 수행할 경우의 응집도
* 통신적 : 동일한 입력과 출력을 사용하여 다른 기능을 수행하는 활동들이 모여 있을 경우의 응집도
* 순차적 : 모듈 내에서 한 활동으로부터 나온 출력값을 다른 활동이 사용할 경우의 응집도
* 기능적 : 모듈 내부의 모든 기능이 단일한 목적을 위해 수행되는 경우의 응집도

1. 소프트웨어 모듈 결합도
2. 개념

모듈 내부가 아닌 외부의 모듈과의 연관도 또는 모듈 간의 상호 의존성

소프트웨어 구조에서 모듈 간의 관련성을 측정하는 척도

1. 유형 : 갈 수록 결합도가 낮아지고 낮아지면 더 좋은 것

* 내용 : 다른 모듈 내부에 있는 변수나 기능을 다른 모듈에서 사용하는 경우의 결합도
* 공통 : 파라미터가 아닌 모듈 밖에 선언되어 있는 전역 변수를 참조하고 전역 변수를 갱신하는 식으로 상호 작용하는 경우의 결합도
* 외부 : 두 개의 모듈이 외부에서 도입된 데이터 포맷, 통신 프로토콜, 또는 디바이스 인터페이스를 공유할 경우의 결합도
* 제어 : 어떤 모듈이 다른 모듈의 내부 논리 조직을 제어하기 위한 목적으로 제어 신호를 이용하여 통신하는 경우의 결합도
* 스탬프 : 모듈간의 인터페이스로 배열이나 객체, 구조 등이 전달되는 경우의 결합도
* 자료 : 모듈 간의 인터페이스로 전달되는 파라미터를 통해서만 모듈 간의 상호 작용이 일어나는 경우의 결합도

1. 팬인 및 팬아웃
2. 개념

소프트웨어의 구성 요소인 모듈을 계층적으로 분석하기 위해서 팬인, 팬아웃을 활용

팬인과 팬아웃 분석을 통하여 시스템의 복잡도를 측정

팬인은 어떤 모듈을 제어 하는 모듈의 수고 모듈 자신을 기준으로 모듈에 들어오면 팬인

팬아웃은 어떤 모듈에 의해 제어되는 모듈의 수 자신을 기준으로 모듈에 나가면 팬아웃

1. 계산 방법

위에서 부터 밑으로는 A -> B 이면 A 팬인은 0개 B 팬인은 1개가 있는거다 A 팬 아웃은 1개 B 팬 아웃은 0개다

* 모듈 테스트

모듈 테스트는 화이트 박스 기법을 활용하고 대표적인 단위 테스트 도구인 xUnit 을 활용

테스트 종류

xUnit 은 자바 ( jUnit ), C++ ( cppUnit ), .Net ( nUnit ) 등 다양한 언어를 지원하는 단위 테스트 프레임 워크

소프트웨어의 함수나 클래스 같은 서로 다른 구성 원소 ( 단위 ) 를 테스트

* jUnit : 자바 프로그래밍 언어용 단위 테스트 도구
* CppUnit : 자바의 jUnit 을 C++ 로 구현한 단위 테스트 도구
* HttpUnit : 웹 브라우저 없이 웹사이트 테스트를 수행 하는 단위 테스트 도구