소프트웨어공학 활용

디자인 및 구성 관리 도구

한국기술교육대학교
온라인평생교육원

학습내용

- 디자인 도구
- 구성 관리 도구

학습목표

- 디자인 도구를 이해하고 설명할 수 있으며, 적합한 도구를 선택할 수 있다.
- 구성 관리 도구를 이해하고 설명할 수 있으며, 적합한 도구를 선택할 수 있다.

디자인 도구

1 소프트웨어 디자인

1 소프트웨어 디자인(설계) 개요

1 정의

소프트웨어 디자인(설계) 개요란?

- 요구사항들을 구현하기 위하여 모듈의 구조, 자료구조, 알고리즘 등 시스템을 구체적으로 정의하기 위한 다양한 기술과 원리를 적용하는 과정
- 요구사항을 소프트웨어적인 표현으로 진화시키는 과정

2 소프트웨어 디자인 단계

자료 디자인 → 구조 디자인 → 절차 디자인 → 기타 내부, 외부 디자인

자료 디자인

- 정보 은폐와 자료 추상화 개념이 기본 접근 방법, DB 설계
- 요구사항 정의와 명세화 단계 시의 식별된 자료객체에 대한 논리적인 표현을 선택하는 과정

구조 디자인

- 주목적은 모듈 단위 프로그램 구조를 개발하는 것
- 모듈 간의 제어 관계, 인터페이스 표현
- 시스템 구조도, ERD로 표현

디자인 도구

1 소프트웨어 디자인

1 소프트웨어 디자인(설계) 개요

2 소프트웨어 디자인 단계

절차 디자인

- 자료와 프로그램 구조가 설정된 후, 모듈 내부를 설계하는 과정
- Flowchart로 표현

기타 내부, 외부 디자인

- 자료, 구조, 절차 디자인 외의 다양한 소프트웨어 구성 설계의 과정

3 소프트웨어 디자인 시 고려해야 할 개념

추상화 (Abstraction)	정제 (Refinement)	모듈화 (Modulation)
구조화 (Structured)	제어계층	자료구조
소프트웨어 Procedure 개념 사용	정보은닉 (Information Hiding)	

디자인 도구

1 소프트웨어 디자인

2 소프트웨어 디자인 방법론

1 소프트웨어 디자인 방법론 종류

자료 흐름 중심 디자인	기능적 관점, 하향식, Yourdon
객체지향 디자인	자료와 기능을 추상화, Jacobson, Rumbough, Booch
자료 중심 디자인	입출력 자료 구조 파악, Jacobson, Warnier-orr

2 소프트웨어 설계 수칙

- 모듈 간의 응집력은 높이고 결합도는 낮추어야 함
- 과다한 깊이(FAN-IN), 넓이(FAN-OUT) 구조가 되지 않게 해야 함
- 모듈의 영향권을 받는 모듈은 하위에 두도록 함

디자인 도구

2 디자인 패턴

1 소프트웨어 디자인(설계) 패턴의 개념

소프트웨어 디자인(설계) 패턴

- 자주 사용하는 디자인 형태를 정형화해서 이를 유형별로 설계 템플릿을 만듦
- 디자인 지식을 검증하고 이를 추상화하여 일반화한 템플릿

→ 많은 개발자가 경험상 체득한 지식

1 장점·단점

장점

- 개발자(설계자) 간의 원활한 의사소통 가능
- 소프트웨어 구조 파악 용이
- 재사용을 통한 개발 시간 단축
- 설계 변경요청에 대한 유연한 대처

단점

- 객체지향 설계
- 구현 위주
- 초기 투자 비용 부담

디자인 도구

2 디자인 패턴

1 소프트웨어 디자인(설계) 패턴의 개념

2 역사

1. 1990년대 초반, Erich Gamma(1992)에 의해 시작됨
2. 일반적으로 GoF(Gang of Four)의 분류가 많이 활용됨
   - 에리히 감마(Erich Gamma), 리처드 헬름(Richard Helm), 랄프 존슨(Ralph Johnson), 존 블리시데스(John Vlissides)
3. 23개의 일반적이고 유용한 패턴을 제공함
4. 패턴을 사용하게 되면 검증된 해결방안을 계속 재사용할 수 있음

디자인 도구

2 디자인 패턴

1 소프트웨어 디자인(설계) 패턴의 개념

3 GoF 디자인 패턴

반복자(Iterator)

- 내부 표현부를 노출하지 않고 어떤 객체 집합에 속한 원소들을 순차적으로 접근할 수 있는 방법

적응자(Adapter)

- 클래스의 인터페이스를 사용자가 기대하는 다른 인터페이스로 변환하는 패턴
- 호환성이 없는 인터페이스 때문에 함께 동작할 수 없는 클래스들이 함께 작동하도록 함

템플릿 메소드(Template Method)

- 객체의 연산에는 알고리즘의 뼈대만을 정의하고 각 단계에서 수행할 구체적 처리는 서브 클래스 쪽으로 미루는 패턴
- 알고리즘 구조 자체는 그대로 두고, 알고리즘 각 단계의 처리를 서브 클래스에서 재정의할 수 있게 함

팩토리 메서드(Factory Method)

- 객체를 생성하는 인터페이스는 미리 정의하고 인스턴스를 만들 클래스의 결정은 서브클래스 쪽에서 내리는 패턴
- 팩토리 메소드 패턴에서는 클래스의 인스턴스를 만드는 시점을 서브 클래스로 미룸

디자인 도구

2 디자인 패턴

1 소프트웨어 디자인(설계) 패턴의 개념

3 GoF 디자인 패턴

단일체(Singleton)

- 어떤 클래스의 인스턴스는 오직 하나임을 보장
- 이 인스턴스에 접근할 수 있는 전역적인 접촉점을 제공하는 패턴

원형(Prototype)

- 생성할 객체의 종류를 명세화하는 데에 원형이 되는 예시물을 이용
- 그 원형을 복사함으로써 새로운 객체를 생성하는 패턴

빌더(Builder)

- 복합 객체의 생성 과정과 표현 방법을 분리하여 동일한 생성 절차에서 서로 다른 표현 결과를 만들 수 있게 하는 패턴

추상 팩토리(Abstract Factory)

- 구체적인 클래스를 지정하지 않고 관련성을 갖는 객체들의 집합을 생성
- 서로 독립적인 객체들의 집합을 생성할 수 있는 인터페이스를 제공하는 패턴

디자인 도구

2 디자인 패턴

1 소프트웨어 디자인(설계) 패턴의 개념

3 GoF 디자인 패턴

가교(Bridge)

- 구현부에서 추상층을 분리하여 각자 독립적으로 변형할 수 있게 하는 패턴

전략(Strategy)

- 동일 계열의 알고리즘 군을 정의하고 각각의 알고리즘을 캡슐화하며, 이들을 상호교환이 가능하도록 만드는 패턴
- 알고리즘을 사용하는 사용자와 상관없이 독립적으로 알고리즘을 다양하게 변경

복합체(Composite)

- 객체들의 관계를 트리 구조로 구성하여 구분
- 부분-전체 계층을 표현하는 패턴으로 사용자가 단일 객체와 복합 객체 모두 동일하게 다룸

장식자(Decorator)

- 주어진 상황 및 용도에 따라 어떤 객체에 책임을 덧붙이는 패턴
- 기능 확장이 필요할 때 서브클래싱 대신 쓸 수 있는 유연한 대안이 될 수 있음

디자인 도구

2 디자인 패턴

1 소프트웨어 디자인(설계) 패턴의 개념

3 GoF 디자인 패턴

방문자(Visitor)

- 객체 구조를 이루는 원소에 대해 수행할 연산을 표현하는 패턴
- 연산을 적용할 원소의 클래스를 변경하지 않고도 새로운 연산을 정의할 수 있게 함

책임 연쇄(Chain of Responsibility)

- 요청을 처리할 기회를 하나 이상의 객체에 부여하여 요청을 보내는 객체와 요청을 받는 객체 사이의 결합을 피하는 패턴
- 요청을 받을 수 있는 객체를 연쇄적으로 묶고, 실제 요청을 처리한 객체를 만날 때까지 객체 고리를 따라서 요청을 전달

퍼사드(Facade)

- 서브 시스템에 있는 인터페이스 집합에 대해서 하나의 통합된 인터페이스를 제공
- 서브 시스템을 좀 더 사용하기 편하게 만드는 상위 수준의 인터페이스를 정의

중재자(Mediator)

- 한 집합에 속해있는 객체들의 상호작용을 캡슐화하는 객체를 정의하는 패턴
- 객체들이 직접 서로를 참조하지 않도록 객체들 사이의 소결합(Loose Coupling)을 촉진시키며 개발자가 객체들의 상호작용을 독립적으로 다양화시킴

디자인 도구

2 디자인 패턴

1 소프트웨어 디자인(설계) 패턴의 개념

3 GoF 디자인 패턴

감시자(Observer)

- 객체 사이에 일대다의 의존 관계를 정의해 둠
- 어떤 객체의 상태가 변할 때 그 객체에 의존성을 가진 다른 객체들이 그 변화를 통지하고, 자동으로 갱신 될 수 있게 만드는 패턴

메멘토(Memento)

- 캡슐화를 위배하지 않는 채 어떤 객체의 내부 상태를 잡아내고 신체화시켜, 이후에 해당 객체가 그 상태로 다시 되돌아올 수 있도록 하는 패턴

상태(State)

- 객체의 내부 상태에 따라 스스로 행동을 변경할 수 있게끔 허가하는 패턴

플라이급(Flyweight)

- 크기가 작은 객체가 여러 개 있을 때, 공유를 통해 이들을 효율적으로 지원하는 패턴

디자인 도구

2 디자인 패턴

1 소프트웨어 디자인(설계) 패턴의 개념

3 GoF 디자인 패턴

프록시(Proxy)

- 어떤 다른 객체로 접근하는 것을 통제하기 위해서 그 객체의 대리자(Surrogate), 자리채움자 (Placeholder)를 제공하는 패턴

명령(Command)

- 요청을 객체의 형태로 캡슐화하여, 서로 요청이 다른 사용자의 매개 변수화, 요청 저장 또는 로깅, 그리고 여난의 취소를 지원하게 만드는 패턴

해석자(Interpreter)

- 주어진 언어에 대해, 그 언어의 문법을 위한 표현 수단을 정의
- 그 표현 수단을 사용하여 해당 언어로 작성된 문장을 해석하는 해석기를 정의하는 패턴

구성 관리 도구

1 소프트웨어 구성 관리

1 소프트웨어 구성 관리(형상 관리) 개요

1 정의

소프트웨어 구성 관리 (형상 관리) 개요란?

- 소프트웨어 구성 관리 또는 형상 관리라고 함
- 소프트웨어 변경 사항을 체계적으로 관리하는 것

소프트웨어 구성 관리(Software Configuration Management) : SDLC 기간 동안 개발되는 소프트웨어의 무결성을 유지하는 것

2 구성 관리 순서

- 형상목록의 변경 요구를 검토, 승인
- 형상통제 위원회의 승인을 통한 변경 통제가 이루어져야 함

소프트웨어 형상 관리 수립 → 형상 식별 → 형상 통제 → 형상 검사 → 형상상태 기록, 보고

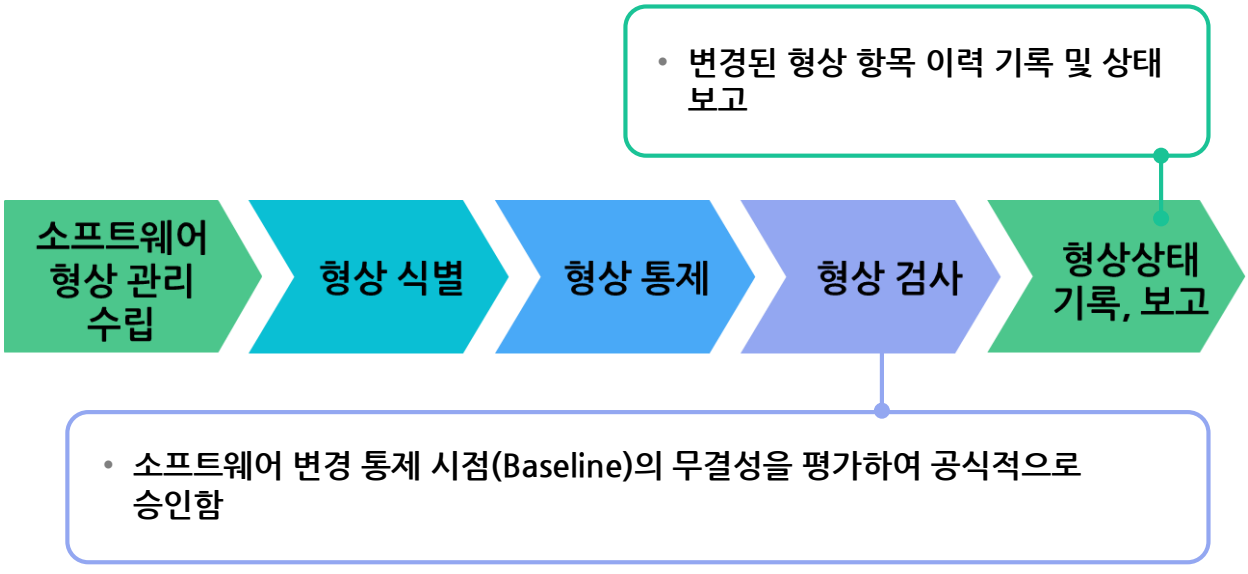
- 형상 관리 대상들을 구분, 관리 목록을 번호로 정리하여 부여
- 형상 항목 : 바이너리 형태의 파일, 소스코드 및 개발 단계에 생성한 문서형식의 모든 산출물의 이력 등을 말함

구성 관리 도구

1 소프트웨어 구성 관리

1 소프트웨어 구성 관리(형상 관리) 개요

2 구성 관리 순서



3 구성 관리를 하는 이유

- 어떤 형태로든 소스코드를 백업하여 보호, 추후 발생할 오류 수정에 대비
- 소스코드의 변경 사항을 추적
- 소스코드를 누가 수정했는지 추적
- 대규모 수정 작업을 더욱 안전하게 진행

구성 관리 도구

1 소프트웨어 구성 관리

1 소프트웨어 구성 관리(형상 관리) 개요

3 구성 관리를 하는 이유

- Branch로 프로젝트에 영향을 최소화하면서 다른 부분을 개발
- Merge로 검증이 끝난 후 새로이 개발된 부분을 합치기 위함

4 구성 관리에 자주 쓰는 용어

Repository	Check Out	Check In, Commit
• 파일의 현재 버전과 변경 이력 정보를 저장하는 저장소	• Repository에서 파일을 가져옴	• Check Out 한 파일의 수정이 끝난 경우 Repository에 새로운 버전으로 갱신 • 갱신할 때 충돌을 보고, Diff 도구를 이용해 수정, Commit 하는 과정을 거침

구성 관리 도구

1 소프트웨어 구성 관리

1 소프트웨어 구성 관리(형상 관리) 개요

4 구성 관리에 자주 쓰는 용어

Revision

- 특정 형상항목에 대한 갱신 작업 또는 그 결과물

Version

- 특정 형상목에 대한 다수의 실체를 말함
- 리비전 결과물이나 형태

Release

- 개발, 변경 완료된 형상항목을 배포하는 행위

Baseline

- 대상 시스템에 대한 기술적인 통제를 하게 되는 시점 또는 그 시점의 기준 문서

구성 관리 도구

2 형상 관리 도구

1 형상 관리 도구의 정의

형상 관리 도구란?

형상 관리 도구는 소스코드나 문서의 버전 관리, 이력 관리, 추적 등 변경 사항을 체계적으로 관리할 수 있는 기능을 제공하는 도구

가장 널리 사용되는 도구는
CVS(Concurrent Versions System), SVN(Subversion), Git

2 CVS(Concurrent Versions System)

1 주요 기능

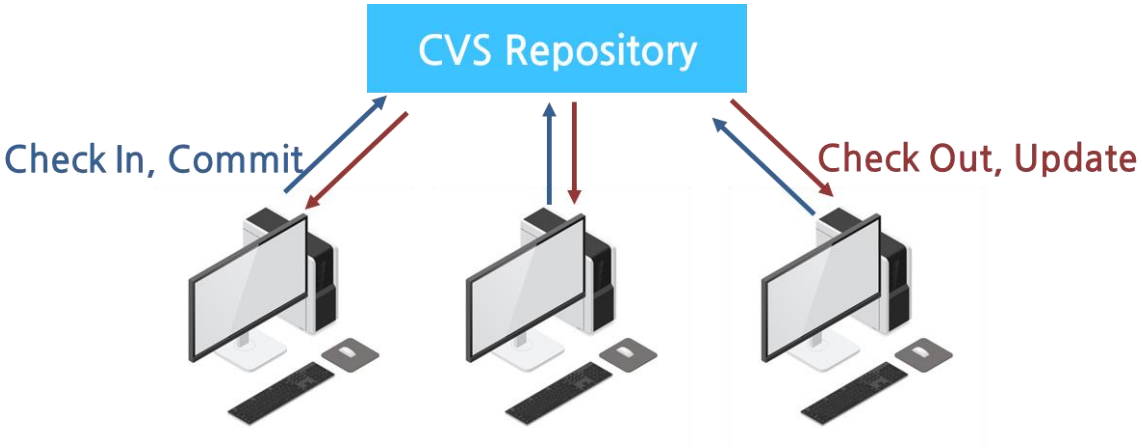
- 서버 저장소와 클라이언트의 변경 사항 전송
  - EVM(Earned Value Management)
- Diff를 통한 파일 내용 비교
- 파일 단위의 변경 사항 관리

구성 관리 도구

2 형상 관리 도구

2 CVS(Concurrent Versions System)

1 주요 기능



2 장점·단점

장점

- 직관적이고 비교적 단순한 명령 세트

단점

- 유니코드 파일명 지원 부족
- 텍스트 기반 소스 코드만 지원
- 등록된 파일이나 디렉터리의 이름 변경이나 이동이 불편
- Commit 실패 시 롤백 미지원

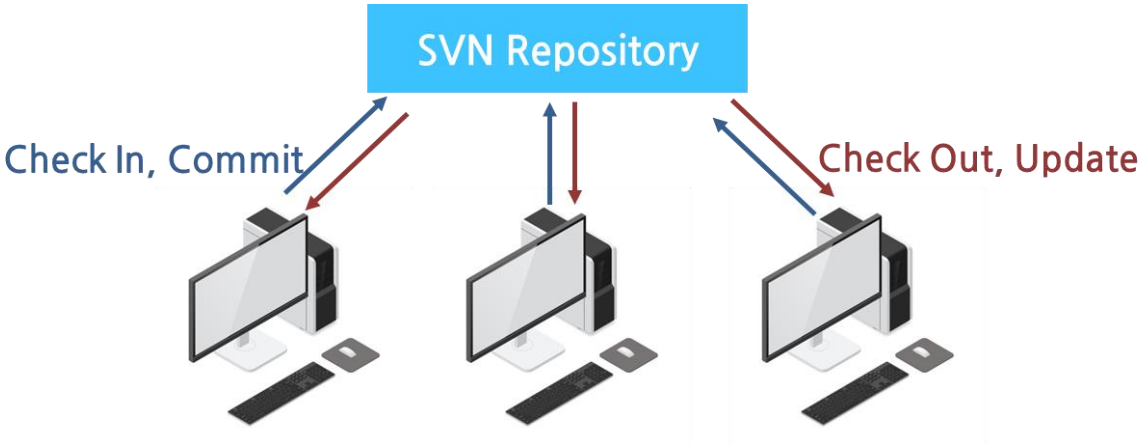
구성 관리 도구

2 형상 관리 도구

3 SVN(Subversion)

1 주요 기능

- 서버 저장소와 클라이언트의 변경 사항 전송
  - Check Out/Check In, Update/Commit 등
- Diff를 통한 파일 내용 비교, 바이너리 문서 형상 관리
- 작업 단위의 변경 사항 관리, Atomic Commit
- Svn:Ignore를 통한 형상 관리 배제 자원 지정 가능



구성 관리 도구

2 형상 관리 도구

3 SVN(Subversion)

2 장점·단점

장점

- CVS 사용자가 쉽게 적용 가능
- 디렉터리나 파일을 자유롭게 이동해도 이력 유지
- gzip을 통한 압축으로 저장공간 절약
- CVS에 비해 빠른 속도
- Atomic Commit으로 Commit 실패 시 롤백 지원
- 다양한 써드 파티 GUI 도구 존재

단점

- Trunk, Branch, Tag가 모두 물리적인 저장 위치를 점유하므로 비효율적
- Git에 비해 Branch, Tag 작업이 무거움

구성 관리 도구

2 형상 관리 도구

4 Git

1 주요 기능

- Branch, Checkout, Commit, Tag 등 로컬 환경에서 형상 관리 기능
- Push, Fetch, Pull 등 원격 환경에서의 변경 사항 전송 가능
- 변경은 했지만 커밋에는 포함하지 않을 수 있는 Staging 기능
- SVN으로 관리되던 저장소를 Git로 전환해 주는 마이그레이션 기능
- Diff를 통한 파일 내용 비교
- 바이너리 문서 형상 관리
- 작업 단위의 변경 사항 관리
- GitIgnore를 통한 형상 관리 배제 자원 지정 기능

구성 관리 도구

2 형상 관리 도구

4 Git

2 장점

코드 꼬임에 따른 위험 감소	Branch 생성, 이동 병합이 매우 가벼우므로 Branch를 자주 사용하여 상황에 맞게 자주 분기 및 병합할 수 있음
쉽게 복구 가능	각 로컬에 완전한 로컬 저장소가 있으므로 원격 저장소에 장애가 나더라도 쉽게 복구 가능
네트워크 빈도 감소, 속도 향상	여러 번의 커밋을 로컬 저장소에 실행하고, 모아진 커밋을 원격 저장소에 반영할 수 있음
공간 절약	Pack 방식의 압축으로 SVN에 비해 저장 공간 절약
GUI 도구 존재	다양한 서브 파티 GUI 도구 존재

구성 관리 도구

2 형상 관리 도구

4 Git

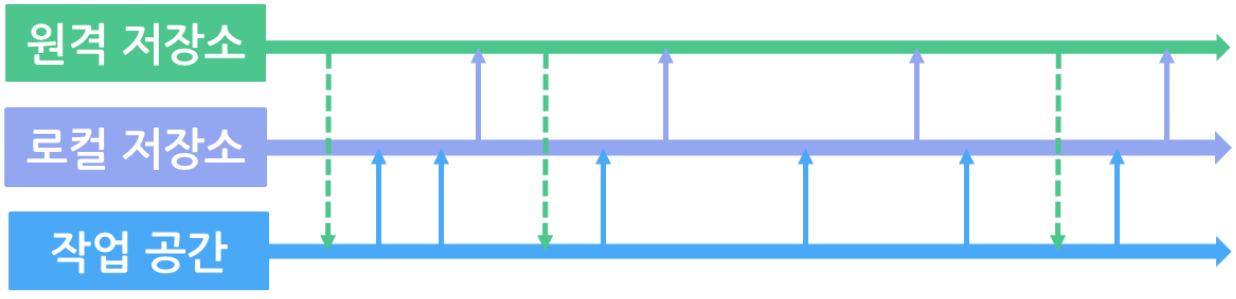
3 단점

CVS, SVN과 기본 개념이 매우 다르므로 적응에 시간이 필요

의미나 동작이 SVN이나 CVS와 다른 명령어가 있어 혼란

➡ Check Out, Commit 등 텍스트는 같음

빈 디렉토리가 저장되지 않음



구성 관리 도구

2 형상 관리 도구

5 형상 관리 스펙

	CVS	SVN	Git
라이선스	GNU GPL	Apache License	GUN GPL
적용언어	무관	무관	무관
OS	Window/Linux Mac은 써드 파티 도구	Window /Linux/Mac	Window/ Linux/Mac
실행환경	Command Line Interface	Command Line Interface	Command Line Interface
GUI	Tortoise CVS 등 써드 파티 도구	TortoiseSVN, WinSVN 등 써드 파티 도구	번들 제공 Source Tree, GitEye, git-cola 등 다양한 써드 파티 도구

학습정리

1. 디자인 도구

- 요구사항들을 구현하기 위하여 모듈의 구조, 자료구조, 알고리즘 등 시스템을 구체적으로 정의하기 위한 다양한 기술과 원리를 적용하는 과정으로 요구사항을 소프트웨어적인 표현으로 진화시키는 과정
- 소프트웨어 디자인 단계는 자료 디자인, 구조 디자인, 절차 디자인, 기타 내부, 외부 디자인 단계로 이루어짐
- 소프트웨어 디자인 시 고려할 개념으로는 추상화, 정제, 모듈화, 구조화, 제어계층, 자료구조, 정보 은닉 등이 있음
- 소프트웨어 디자인 방법론으로 자료 흐름 중심 디자인, 객체지향 디자인, 자료 중심 디자인 방법론이 존재
- 설계 수칙으로는 모듈 간의 응집력은 높이고 결합도는 낮추며 과다한 깊이, 넓이 구조가 되지 않게 해야 하며 모듈의 영향권을 받는 모듈의 하위에 두도록 해야 함
- 디자인(설계) 패턴을 사용하면 개발자 간의 의사소통이 가능하며 소프트웨어 구조 파악이 쉽고 재사용을 통한 개발 시간 단축 및 설계 변경요청에 대한 유연한 대처가 가능함

학습정리

2. 구성 관리 도구

- 소프트웨어 구성 관리는 형상 관리라고도 불림
- 구성 관리 순서 : 소프트웨어 형상 관리 수립 → 형상 식별 → 형상 통제 → 형상 검사 → 형상 상태 기록 및 보고
- 구성 관리를 통하여 소스코드 혹은 문서의 변경 사항 및 이력을 추적 관리 가능하며 대규모 수정 작업을 좀 더 안전하게 진행할 수 있는 장점이 있음
- 가장 널리 쓰이는 형상 관리 도구 : CVS, SVN, Git
- CVS : 가장 간단한 구조, Check Out/ Check In, Update/Commit 등 서버 저장소와 클라이언트의 변경 사항을 전송
- SVN : 서버 저장소와 클라이언트의 변경 사항을 전송하며, 작업 단위의 변경 사항 관리가 가능하고 형상 관리 배제 자원을 지정할 수 있음
- Git : 소스코드가 변경된 이력을 쉽게 확인할 수 있고, 특정 시점에 저장된 버전과 비교하거나 특정 시점으로 되돌아갈 수 있으며 분산형 버전 관리 시스템