
## 제품소프트웨어 패키징
## (Git과 GitHub를 활용한 소스코드 관리)

# 소스코드 관리의 중요성

한국기술교육대학교
온라인평생교육원


## 학습내용

- 소스코드 관리란?
- 소스코드 관리 사례
- 형상관리를 통한 생산성 향상

## 학습목표

- 소스코드 관리가 무엇인지 설명할 수 있다.
- 소스코드 관리 사례를 살펴보고 소스코드 관리를 적용할 수 있다.
- 형상관리를 통해 생산성을 향상시킬 수 있다.


# 소스코드 관리란?

## 1. 소스코드 관리 정의

### 1) 소스코드란

- 소프트웨어를 개발 하거나 유지보수 하는 과정에서 발생하는 소스코드, 문서, 인터페이스 등 각종 결과물에 대해 소스코드를 만들고, 이들 소스코드에 대한 변경을 체계적으로 관리하기 위한 활동

### 2) 소스코드 관리의 장점

- 소스코드의 변화 내용을 저장하고 이것을 되돌리는데 용이
- 다양한 버전으로 분기하는 등의 다양한 관리 가능
- 개발 과정 중에 발생하는 다양한 문서들과 문서들의 버전 관리 : 프로젝트 전반에 대한 관리 가능
- 각 시점에 맞추어 내부적으로나 공식적으로 배포하는 것을 버전으로 관리 가능 : 버전 관리 시스템

### 3) 소스코드 관리의 필요성

- 대규모 프로젝트
  - 성공적이고 효율적인 소프트웨어 개발 프로젝트 수행
  - 소스코드를 관리하지 않으면 다양한 이유로 소스코드가 유실되고 전체 소프트웨어의 개발 일정에 차질이 생김
  - 소스코드 관리를 지나치게 할 경우 실제 개발은 이루어지지 않고, 집중력이 저하되고 좋은 질의 소스코드가 생산될 수 없게 됨
- 소규모 프로젝트
  - 원하는 최종 결과물이 발생할 때까지 효과적으로 소스들을 관리
  - 기능별로 구현을 완료한 시점에 버전을 관리 가능
  - 새로운 기능을 구현하다가 문제가 생겼을 때 잘 되던 버전으로 돌아가는 등의 작업을 손쉽게 가능

### 4) 소스코드 관리시스템의 종류

- CVS, SVN, Git
- 소스코드에 변경된 사항을 확인하고, 수정하며, 충돌을 방지하는 작업 수행




## 2. 소스코드 관리의 필요성

### 1) 작업자가 교대로 작업하는 방법

- 장점 : 소스코드의 충돌이 나타날 위험이 없음
- 단점 : 서로 다른 기능 개발 시 매우 비 효율적

### 2) 각자 따로 할 일을 구현하는 방법

- 장점 : 작업자가 교대로 작업해야 하는 방법에 비해 인적 자원을 효율적으로 활용할 수 있음
- 단점 : 차이점을 찾기 위해 엄청난 시간이 소요되고, 재작업 중 소스코드 손실 가능성이 높음

### 3) 버전관리시스템을 관리하는 방법

- 활용 순서
  - 중앙 저장소(소스코드 관리 서버)에 소스코드 저장 → 개발자 본인 PC에서 중앙 저장소에 저장되어 있는 소스코드 복사 → 각자 수정 저장소로 소스코드를 Commit 하여 반영 →Update를 통해 중앙 저장소의 최신 정보를 본인 PC로 복사
- 장점 : 주기적인 수정에도 바로 확인이 가능하므로 충돌 방지 및 관리가능
- 활용효과
  - 특정 시기에 모든 개발자가 모여 합치는 작업 감소
  - 서버 중심시스템을 통해 실시간으로 코드를 합칠 수 있음
  - 실시간 소스코드 확인을 통해 협업 및 토론이 활발해질 수 있음
  - 소프트웨어 개발의 효율성 및 소스코드의 질이 향상될 수 있음
- 수정사항관리방법
  - Commit : 작업 디렉토리에서 변경, 추가 및 삭제된 파일을 원본 저장소인 서버에 적용하는 단계
  - Revision : 소스코드를 수정할 때 마다 수정시간, 수정한 사람, 수정한 내용을 관리하는 단계
  - Rollback : 작업 디렉토리의 사본을 특정 Revision 시점으로 되돌리는 단계


## 소스코드 관리란?

### 3. 소스코드 관리의 기본 용어

- Repository : 원본 소스를 저장하고 있는 저장소
- Working Copy : 소스코드 서버에서 내려 받은 코드가 있는 내 컴퓨터의 저장 장소
- Commit : 수정된 사항을 반영하는 단계
- Update : 현재 소스코드 서버에 있는 최신 정보를 내 컴퓨터로 가져오는 단계
- Revision : 각 소스코드가 반영될 때마다 버전을 관리하는 것
- Rollback :  작업 디렉토리의 사본을 특정 Revision 시점으로 되돌리는 단계


# 소스코드 관리 사례

## 1. 소스코드 관리방법 적용하기

### 1) 개인프로젝트에서 소스코드 관리

- 장점 : 소스코드를 구현할 때는 대부분 생각나는 대로 바로 개발하고 저장하므로 빠른 구현을 할 수 있음
- 단점 : 다음 과정을 진행하면서 구조를 자주 바꾸기 때문에 잦은 설계 수정 등으로 인해 개발 중이나 후에 근본적인 원인을 찾고 수정하기가 매우 어려움
- 주의사항
  - 각 단계별 구현 내용의 저장
  - 문제가 발생하였을 때 잘 되는 버전으로 Rollback
  - 개발 내용을 소스코드에 잘 담아 타인에게 명확하게 전달
- 고려하지 않아도 되는 것
  - 불편한 소스코드 관리 정책 관리 작업
  - 별도의 소스코드 관리 서버 구성
  - 중앙 서버에서 소스코드 관리
  - 같은 시점에 여러 코드의 중복


# 소스코드 관리 사례

## 1. 소스코드 관리방법 적용하기

### 2) 사내프로젝트에서 소스코드 관리 고려사항

- 다수의 개발자들의 소스코드가 충돌되지 않도록 고려
  - 개인 프로젝트와 가장 다른 부분이며 가장 많이 신경 써야 할 부분
  - 다양한 스타일의 개발자들이 함께하여 많은 시간이 소요됨
  - 개발자들간 소스코드 변동 및 해석의 어려움이 발생하지 않는 방법 고민
- 보안상 문제 고려
  - 소스코드 관리 서버의 보안(내부 망에서만 접근)
  - 사내 프로젝트는 외부에 최종 결과물 외에는 공개하지 않고 진행
  - 내부 망으로 구성하여 외부인들이 소스코드를 확인할 수 없도록 금지
  - 개발자들이 다양한 곳에서 접속하여 개발한다면 권한을 관리하여 소스코드의 열람 제한
  - 각 개발자에 따라 소스코드를 검토하는 권한을 부여하여 잘못된 코드가 반영되는 것을 미연에 방지
  - 각자의 스타일에 따라 코드를 보기 힘들 수 있으니 가독성 높은 코드 구현
  - 코딩 스타일을 가이드 하고 지속적으로 검토하는 역할 필요

## 1. 소스코드 관리방법 적용하기

### 3) 오픈소스 프로젝트에서 소스코드 관리

- 오픈소스 : 누구든지 언제나 소스코드를 볼 수 있고, 수정하여 반영할 수 있는 소스
- 주의사항
  - 항상 접속이 원활하도록 서버의 환경이 갖추어져야 함
  - 다수의 개발자가 참여하면 할수록 더욱 활성화되고 완성도가 높아짐
  - 스타일과 코드의 안정적 동작을 위한 의사결정 체계를 민주적으로 구성
  - 집단지성이 발휘될 수 있는 정책
  - 백업기능 강화
  - 가독성을 향상시킬 수 있는 방안
  - 수많은 개발자가 관심을 가지고 참여할 수 있도록 독려

## 2. 소스코드 관리 사례 알아보기

### 1) 잘못된 소스코드 관리

- 개인관리
  - 해당 프로젝트를 진행한 개발자가 없으면 무조건 중단
  - 소스코드의 유실 또는 오류가 발생하였을 때 복구 불가
- 단순 파일 관리
  - 소스코드를 압축하여 날짜별로 저장해 놓는 방법
  - 단순히 백업의 의미 외엔 용량만 차지하는 경우 발생
- 코딩 스타일 과일 소홀
  - 코딩 스타일을 전혀 고려하지 않고 무조건 백업하면 개발자마다 모두 다르게 구현
  - 가독성이 매우 떨어져 다른 사람과의 협업이 매우 어려움
- 테스트 없는 관리
  - 테스트를 하지 않는다면 동작하지 않는 소스코드가 반영되는 문제
  - 향후에 문제 발생 시, 문제점을 확인불가
- 소스 버전 툴 투자 없음
  - 개발 중 문제가 발생 시 해결기간이 개발기간보다 더 많은 시간 소요
- 소스코드 품질관리 소홀
  - 각기 다른 방법으로 구현이 되는 것을 검토하지 않고 방치한다면, 전체 프로젝트에서의 안정성에 큰 문제 발생
  - 소스코드 반영 단계에서 선임 개발자들의 검증이 이루어지는 절차 꼭 필요

## 2. 소스코드 관리 사례 알아보기

### 2) 구글의 소스코드 관리 사례

- 모든 제품의 소스코드를 하나의 저장소로 관리 : 파편화 방지
- 코드의 성능 보다는 읽기 쉬움을 가장 중요시함
- 새롭게 생성된 소스코드 전담 팀 : 코딩 스타일을 관리하고 쉬운 코드 읽기가 모든 개발자에게 적용
- 개발 문서를 따로 만들지 않고 코드에 모두 적용하여 별도의 문서 없이 코드로만 개발이 진행될 수 있도록 관리


# 형상관리를 통한 생산성 향상

## 1. 소프트웨어 특성

### 1) 소프트웨어란

- 사람에 의해 무형의 코드를 생산하는 것으로 이루어진 지식 기반의 제품
- 제조를 위한 시간이나 비용 등이 발생하지 않기 때문에 언제든지 변화 가능

### 2) 소프트웨어 변화의 특성, 관리방안

- 가시성의 부족 : 형태가 없는 무형의 자산
- 통제의 어려움 : 무형자산으로 그 형태의 변화를 쉽게 관리하거나 예측하는 것이 불가능
- 과정추적 방법 제한 : 무형의 상태로 계속 변화하면서 제품이 탄생
- 지속적인 관리의 어려움 : 모든 결과물이 무형의 자산으로 존재하므로 개발 현황파악이 어렵고 향후 개발 전망에 대한 예측이 매우 어려움
- 관리의 어려움 : 사람의 지식기반으로 변경되므로

## 2. 형상관리의 기준

### 1) 형상관리 단계

- 단계 : 계획 → 요구분석 → 설계 → 개발 → 배포 → 운영
  - 운영 이후, 지속적인 업데이트는 앞의 순서를 그대로 따라서 순환하는 형태로 끊임없이 진행
  - 각 단계에서 관리할 기준을 정리하고 그에 맞게 관리가 이루어진다면 효율적인 소프트웨어의 형상 관리가능
- 형상관리의 과거와 현재
  - 과거의 형상관리 : 문서나 소스 코드의 버전을 관리하는 수준
  - 최근의 형상관리 : 다양한 프로젝트 관리 도구와 자동화 도구가 많이 활용
- 형상관리의 예: TDD를 활용한 테스트
  - TDD는 기준선이 되는 테스트 수행 방법을 정의하여 형상화하면, 이후의 테스트는 형상화된 테스트 수행 방법을 반복하면 된다는 개념
  - 똑같은 방법으로 테스트 되기 때문에 테스트 자동화 가능

## 2. 형상관리의 기준

### 2) 형상관리 단계의 기준 및 결과물

- 계획
  - 개발을 위한 요구사항을 알아보는 단계
  - 요구사항 명세 및 기능 정의
  - 결과물 : 시스템 명세서나 프로젝트 계획서 등
- 요구분석
  - 요구 기능에 대해서 기본적인 설계 수행
  - 실제 개발을 위한 기준으로 설계
  - 결과물 : 분석 결과서 등
- 설계
  - 개발을 위한 기능을 확정한 후 실질적인 구조 설계
  - 제대로 진행되지 않으면 개발이 다시 진행 되야 하는 문제 발생
  - 결과물 : 프로그램 설계서 등
- 개발
  - 기능과 성능으로 구분하여, 요구하는 기능이 잘 동작하는 지와 성능이 기준을 충족하는지 관리
  - 결과물 : 소스코드, 테스트보고서 등
- 배포
  - 소프트웨어를 통합하고 품질을 검토하는 단계 수행
  - 결과물 : 인증시험 보고서 등
- 운영
  - 사용자 환경에서 지속적인 사용성을 검사하여 버그 수정이나 개선되어야 할 부분을 업데이트 할 수 있도록 관리
  - 운영 단계에서의 수정이 더 좋은 품질의 소프트웨어가 개발되는데 큰 역할
  - 지속적으로 개발되는 한 단계로 인식


# 핵심정리

## 소스코드 관리란?

### 1. 소스코드 관리 정의

- 소프트웨어를 개발 하거나 유지보수 하는 과정에서 발생하는 소스코드, 문서, 인터페이스 등 각종 결과물에 대해 소스코드를 만들고, 이들 소스코드에 대한 변경을 체계적으로 관리하기 위한 활동

### 2. 소스코드 관리의 필요성

- 서버 중심 시스템을 통해 실시간으로 소스코드를 합칠 수 있음
- 실시간 소스코드 확인을 통해 협업 및 토론이 활발해질 수 있음
- 소프트웨어 개발의 효율성 및 소스코드의 질이 향상될 수 있음

### 3. 소스코드 관리의 기본용어

- Repository
- Working Copy
- Commit
- Update
- Revision
- Rollback


# 핵심정리

## 소스코드 관리 사례

### 1. 소스코드 관리 방법 적용하기

- 소스코드 변경 사항 반영 시, 항상 체크할 수 있는 정책을 수립함
- 소스코드는 소프트웨어의 성능도 고려해야 하지만 가독성이 좋아야 여러 사람의 협업이 쉬움
- 소스코드 관리를 효과적으로 하면 큰 프로젝트도 문제없이 수행할 수 있음

### 2. 소스코드 관리 사례 알아보기

- 구글의 소스코드 관리 원칙
  - 구글 모든 제품의 소스코드를 저장소 하나로 관리함
  - 성능보다 코드의 읽기 쉬움이 중요함
  - 소스코드를 청소하는 팀이 있음
  - 개발문서가 거의 없으며, 소스코드에 개발 문서를 거의 모두 담음


# 핵심정리

## 형상관리를 통한 생산성 향상

### 1. 소프트웨어의 특성

- 가시성 부족
- 통제 어려움
- 과정의 추적 어려움
- 지속적인 관리의 어려움
- 잦은 변경

### 2. 형상관리의 기준

- 계획
- 요구분석
- 설계
- 개발
- 배포
- 운영