01 소프트웨어 공학 개념

Section 1. 소프트웨어 공학

1. 소프트웨어 공학(Software Engineering)

(1) 소프트웨어 공학의 정의

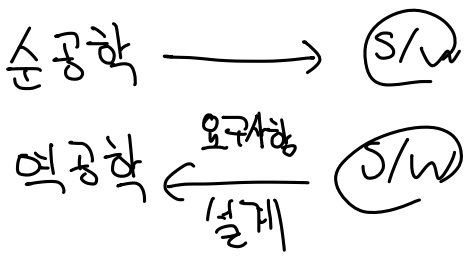
- 소프트웨어 위기를 극복하고 효율적으로 품질 높은 소프트웨어를 개발하기 위한 학문
- 소프트웨어를 개발하는데 있어서, 어떻게 개발할지, 무엇을 개발할지와 같은 방법 도구, 이론을 모두 포함한 포괄적인 개념

(2) 소프트웨어의 위기의 원인

- 소프트웨어 특성에 대한 이해 부족
- 소프트웨어 관리 방법론 부재
- 올바른 설계 없이 프로그래밍에만 치중
- 소프트웨어 개발에 대한 전문적 교육 부족
- 작업일정과 비용의 추정치가 부정확

(3) 소프트웨어의 위기의 결과

- 개발 인력의 부족과 인건비 상승
- 소프트웨어 성능 및 신뢰성 부족
- 개발 기간 및 비용의 증가
- 소프트웨어 품질저하 및 유지보수 비용 증가
- 소프트웨어의 생산성 저하



2. 소프트웨어 공학의 3R

(1) 소프트웨어 공학의 3R의 정의

- 완성된 소프트웨어를 기반으로 역공학, 재공학, 재사용을 통해 소프트웨어의 생산성을 극대화 하는 기법

(2) 소프트웨어 3R의 필요성

- 소프트웨어 유지보수 효율성 향상 및 비용 절감
- 소프트웨어 개발 생산성 향상
- 소프트웨어 이해, 변경, 테스트 용이
- 소프트웨어 변경 요구사항에 대한 신속한 대응
- 소프트웨어 위기 극복

(3) 역공학(Reverse Engineering)

- 기존 개발된 시스템을 CASE도구를 이용하여 사양서, 설계서 등의 문서로 추출하는 작업
- 개발 단계를 역으로 올라가 기존 개발된 시스템의 코드나 데이터로부터 설계 명세서나 요구 분석서 등을 도출하는 작업

- 역공학의 특징
  - 상용화되거나 기 개발된 소프트웨어의 분석을 도와줌
  - 기존 시스템의 자료와 정보를 설계 수준에서 분석 가능해 유지보수성 향상
  - 기존 시스템 정보를 Repository에 보관하여 CASE의 사용을 용이하게 함

차세대 : 재개발 + 재사용
고도화 : 재공학

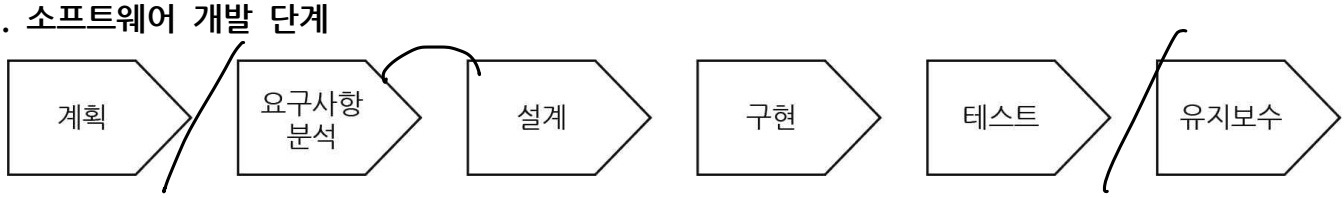
(4) 재공학(Re-engineering)

- 기존 시스템을 널리 사용되는 프로그래밍 표준에 맞추거나 고수준의 언어로 재구성하고, 이기종에서 사용할 수 있도록 변환하는 작업
- 재공학의 특징
  - 현 시스템의 유지보수성 향상
  - 시스템의 이해와 변형을 용이하게 하며, 유지보수 비용 및 시간 절감
  - 표준 준수 및 CASE의 사용 용이

(5) 재사용(Reuse)

- 이미 개발되어 그 기능, 성능 및 품질을 인정받았던 소프트웨어의 전체 또는 일부분을 다시 사용
- 재사용의 특징
  - 소프트웨어 생산의 TCO (Total Cost Overhead) 절감
  - 높은 품질의 소프트웨어 생산을 위한 공유 및 활용 효과
- 재사용의 범위
  - 함수와 객체 재사용 : 클래스나 함수 단위로 구현한 소스코드를 재사용
  - 컴포넌트 재사용
  - 애플리케이션 재사용
- 재사용 방법
  - 합성 중심(Composition Based) : 전자 칩과 같은 소프트웨어 부품, 즉 블록(모듈)을 만들어서 끼워 맞추어 소프트웨어를 완성시키는 방법
  - 생성 중심(Generation Based) : 추상화 형태로 쓰여진 명세를 구체화하여 프로그램을 만드는 방법

3. 소프트웨어 개발 단계



(1) 계획

- 무엇을 개발할 것인지 명확하게 정의
- 개발 범위를 결정
- 시스템의 성격을 파악하여 비용과 기간을 예측

(2) 요구사항 분석(Requirements Analysis)

- 개발할 소프트웨어의 기능과 제약조건, 목표 등을 고객과 함께 정의
- 요구사항의 정확한 이해 및 요구사항 유도
- 과다하거나 불필요한 요구사항에 대한 협상 및 조율
- 요구사항의 적합성 검토 및 향후 예측
- 현재 실행 환경에 대한 분석

(3) 소프트웨어 설계(Design)

- 시스템이 어떻게 동작하는지를 정의
- 요구사항 분석 단계에서 산출된 요구사항을 기준으로, 입력자료, 처리내용, 출력자료 등을 정의
- 설계 구분
  - 시스템 구조 설계 : 모듈간의 관계와 구조 설계
  - 프로그램 설계 : 각 모듈의 처리 절차나 알고리즘 설계
  - 사용자 인터페이스 설계 : 사용자가 시스템을 사용하기 위해 보여지는 부분을 설계

(4) 구현(Development)

- 프로그래밍 언어를 이용하여 실제로 프로그램을 작성
- 코딩과 디버깅이 이루어지며, 단위(모듈) 테스트를 진행

(5) 테스트(Testing)

- 구현된 소프트웨어가 요구사항을 만족하는지 검사 확인과정
- 실행 결과가 예상 결과와 맞는지 검사하고 평가
- 테스트 계획, 통합 테스트 결과서 등을 작성

(6) 유지보수(Maintenance)

- 소프트웨어를 사용하며 문제점을 수정하고, 새로운 기능을 추가
- 소프트웨어를 좀 더 발전시키는 단계