들어가기 : SE 개요

목차

1. SW 개요 (1/3)

- 어디나 존재하는 소프트웨어
  - 소프트웨어 고장 사례
    - NASA 화성탐사 위성 MCO : 궤도변화를 계산하는 변수값 계산 오류 (1998년)
    - 도요타 급발진 사고 : 제어장치 오작동 모니터링 모듈의 프로그래밍 실수 (2009년)
    - 폭스바겐 차량조립과정 로봇팔 오작동 인명피해(2015년), 테슬라 API 장애 발생(2017년), 코레일 중앙관제시스템 장애 (2017년) 등…

새로운 곳에서 발생

1. SW 개요 (2/3)

- 소프트웨어
  - 프로그램 + 프로그램의 개발, 운용, 보수에 필요한 정보 일체

- 소프트웨어의 특징
  - 복잡성(complexity) 표준화X, 표준을 만들기도 힘듦
  - 순응성(conformity) 사용자의 요구사항 / 환경 / 제도의 변화에 적응
  - 변경성(changeability) 기능추가. 개발중 / 개발완료 후 계속
  - 비가시성(invisibility) 내부구조는 시각적으로 볼수 없음. 코드로 숨겨짐

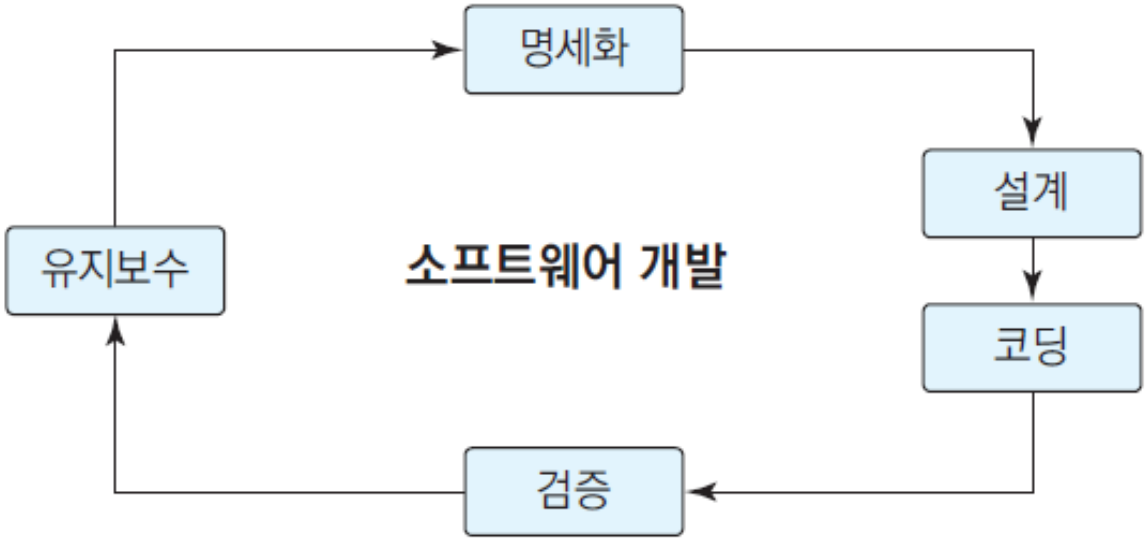
1. SW 개요 (3/3)

- 소프트웨어 유형

소프트웨어 분류	특징	사용되는 카피의 수	요구되는 하드웨어 성능	개발 인력
주문형 소프트웨어 	• 특정 고객 또는 기업의 요구를 만족시키기 위하여 제작한 소프트웨어	적음	낮음	많음
패키지 소프트웨어 	• 패키지화 하여 상업적으로 판매하는 소프트웨어 • 워드프로세서, 스프레드시트, 유통업체의 POS(Point of Sales) 시스템, 재정 분석, 주문 관리, 회계 관리 시스템	중간	높음	중간
임베디드 소프트웨어 	• 다른 시스템에 내장된 소프트웨어	많음	중간	적음

2. 소프트웨어 개발작업 (1/2)

- 소프트웨어 개발의 기본 활동
  - 명세화(specification)
  - 구현(coding)
  - 검증(verification)
  - 유지보수(maintenance)



2. 소프트웨어 개발작업 (2/2)

- 소프트웨어는 사람의 지적 활동에 의해 구축됨
- 소프트웨어 개발작업의 특징
  - 명세화의 어려움 : 재료비용(원가)이 없음 → 비용계산 어려움
  - 재사용의 어려움 → 객체지향의 등장
  - 예측의 어려움 : 고객이 자신이 무엇을 원하는지 정확히 알지못함
  - 유지보수의 어려움 : 개발자가 짠 코드를 다른 개발자가 보고 이해해야함

➢ 즉흥적인 소프트웨어개발 : 프로그래밍 ⇒ 만족할때까지 수정 ⇒ 개선을 위한 아이디어 짜내기의 반복

➢ 품질 ↓, 생산성 ↓, 비용과 일정 ↑

3. SW 공학이란 (1/4)

- 소프트웨어 공학의 정의
  - 질 좋은 소프트웨어를 경제적으로 생산하기 위하여 공학, 과학 및 수학적 원리와 방법을 적용하는 것 (Watts Humphrey, SEI[1])) 방법론.표준화연구
  - **소프트웨어의 개발, 운용, 유지보수에 체계적이고, 숙달되고 수량화된 접근법을 적용하는 것**, 즉 소프트웨어에 공학을 적용하는 것 (IEEE[2])) 표준화

시험X

1) SEI(Software Engineering Institute) at Carnegie Mellon University
2) IEEE(Institute of Electrical and Electronics Engineers) 국제전기전자기술자협회: ANSI(American National Standards Institute, 미국국립표준협회)에 의하여 미국국가표준을 개발하도록 인증 받은 표준개발 기구

3. SW 공학이란 (2/4)

- 소프트웨어 개발에 공학적 접근이 필요한 이유
  - 보고 만질 수 없는 무형성, 진화성 등 소프트웨어 자체의 특성과, 제조가 아닌 개발 과정이므로 어려움
  - 기타 다른 이유
    - 수시로 바뀌는 요구사항
    - 사람에 의존하여 개발되는 특성 (개인의 특성)
    - 요구되는 신뢰도에 따라 분석의 종류와 깊이를 알 수 없음 (의사소통의 어려움)
    - 프로젝트 규모에 따른 인력, 비용, 의사소통, 복잡도 등이 기하급수적으로 증가

3. SW 공학이란 (3/4)

- 소프트웨어 공학의 목표
  - 복잡도 낮춤
  - 비용 최소화
  - 개발기간 단축
  - 대규모 프로젝트 관리
  - 고품질 소프트웨어
  - 효율성

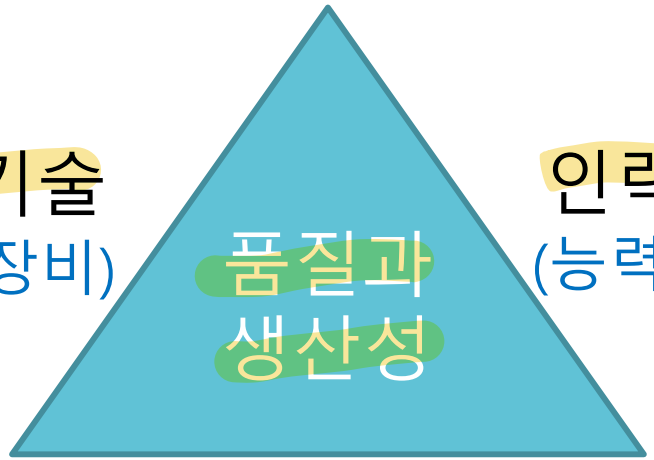
➢ 소프트웨어공학은 여러가지 원리와 방법을 적용하여 품질 좋은 소프트웨어를 최소의 비용으로 계획된 일정에 맞추어 개발하는 것

3. SW 공학이란 (4/4)

- 엔지니어링 식 접근 방법 - **방법, 절차, 도구 사용**
  - 엔지니어링 작업에서는 비용, 일정, 품질과 같은 변수가 중요
  - 일관성
    - 프로젝트의 결과를 어느 정도 정확하게 예측가능
    - 더 높은 품질의 제품을 생산
  - 프로세스의 표준화가 필요 : ISO 9001, CMMI
  - 개발 능력, 결과의 재현성
- 오늘날 비즈니스 환경 변화는 매우 빠름
  - 변경을 조절하고 수용하는 것이 또 하나의 과제

4. SW공학의 접근방법 (1/2)

- 프로젝트를 수행하는 동안 얻은 품질과 생산성은 여러 가지 요인에 좌우됨
- 품질을 좌우하는 세가지 : **인력, 프로세스, 기술**
  - 프로젝트 삼각 균형



기술
(도구와 장비)

인력
(능력, 교육 및 동기가 부여된 인원)

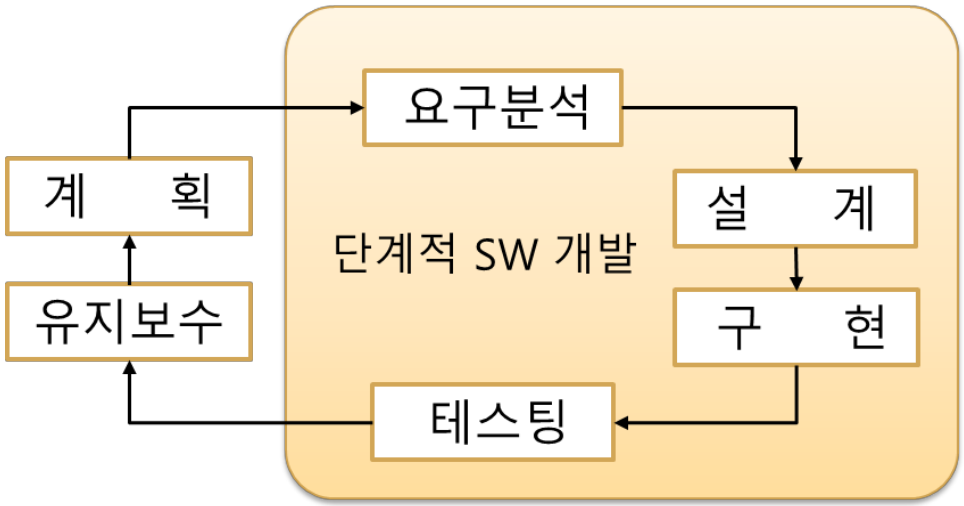
프로세스
(작업들의 관계를 정의하는 절차와 방법)

4. SW공학의 접근방법 (2/2)

- SW를 개발하는 프로세스를 SW와 분리
  - SW공학은 소프트웨어 제작과정에 집중
  - 알고리즘, 운영 체제, 데이터베이스 등은 소프트웨어 제품 자체에 초점
- SW공학 작업의 종류
  - 단계적 SW 개발 프로세스
  - 품질 보증(QA)
  - 프로젝트 관리

4.1 단계적 개발 프로세스 (1/3)

- 단계적 개발 프로세스를 따르는 이유
  - 소프트웨어의 문제를 나눠 여러 개발 단계에서 다른 관점을 다루기 때문
  - 소프트웨어 개발을 코딩에 치중하지 않고 요구분석, 설계, 코딩, 테스팅 등 정해진 절차에 따라 작업



4.1 단계적 개발 프로세스 (2/3)

- 요구분석 : 소프트웨어 시스템이 풀어야 할 문제를 이해하기 위한 작업
  - 시스템을 위해 무엇을 만들어야 하나 ?
  - 결과물 : 요구분석명세서(requirement specification)

- 설계 : 요구명세서에 기술된 문제의 솔루션을 계획
  - 시스템을 어떻게 구축할 것인가?
  - 아키텍처 설계, 인터페이스 설계, 모듈 알고리즘 설계, 데이터베이스 설계
  - 결과물 : 설계 명세서

4.1 단계적 개발 프로세스 (3/3)

- 구현 : 시스템 설계를 프로그래밍 언어로 변환
  - 읽기 쉽고 이해하기 쉬운 코드가 되어야 함
  - 결과물 : 새로운 시스템(실행되는 코드), 유지보수 계획

- 테스팅 : 시스템이 요구에 맞게 실행되나?
  - 소프트웨어 개발 단계에서 사용되는 중요한 품질 제어 수단
  - 단위 테스팅, 통합 테스팅, 인수 테스팅
  - 결과물 : 테스팅 결과 보고서

4.2 품질보증 (1/2)

- 품질 보증 : 개발되고 있는 소프트웨어가 요구와 품질 수준을 만족시킬 것이라는 것을 보장하는 작업

4.2 품질보증 (2/2)

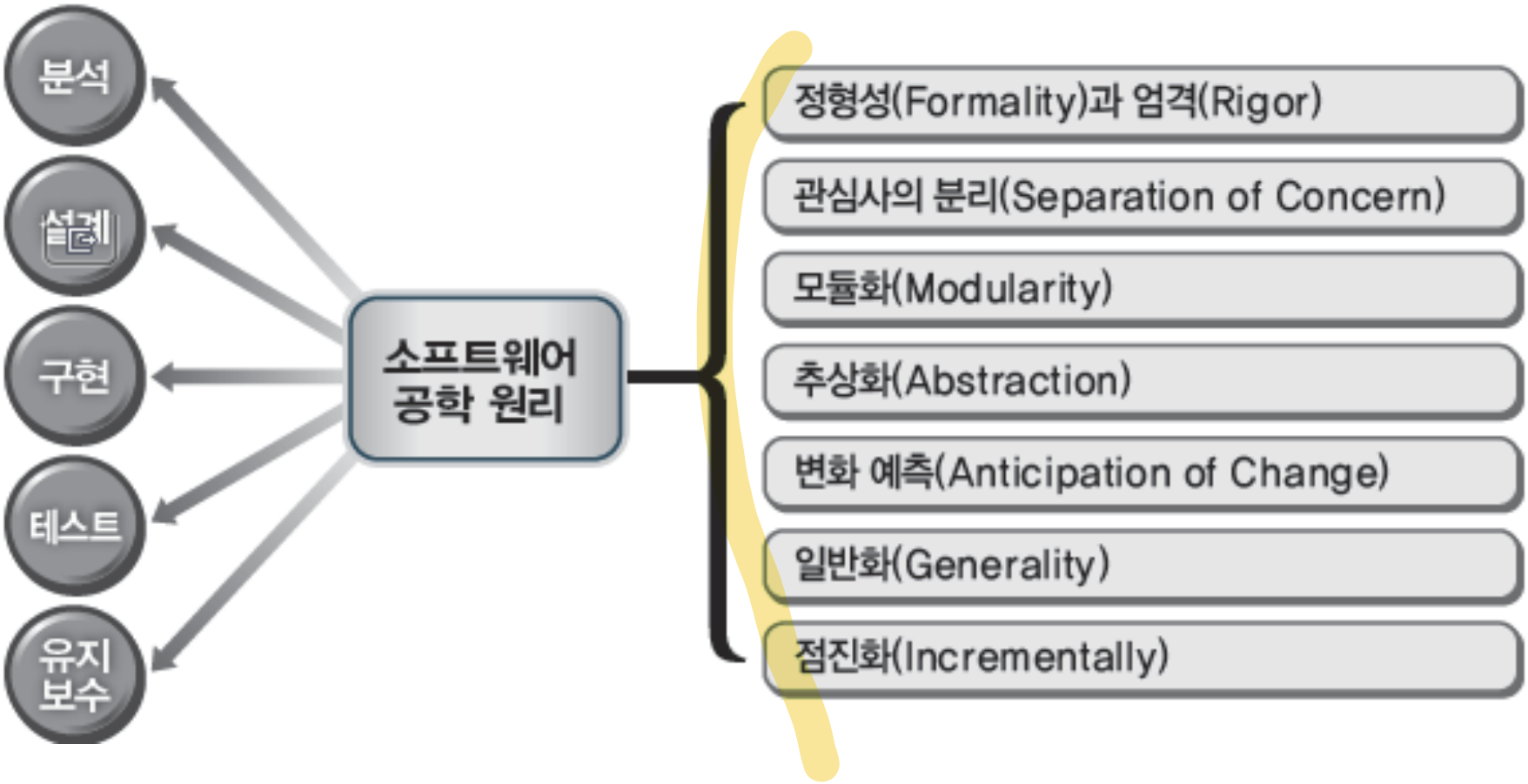
- 확인(verification)
  - 개발작업이 프로젝트를 위해 선택된 프로세스와 방법에 맞게 수행되었는지 체크
  - 요구된 소프트웨어 결과물이 품질 수준에 맞게 생산되었는지 검사
- 검증(validation) : 개발 프로세스에 의하여 생성된 결과물의 정확성을 체크
  - 정적검증(static validation)
  - 동적검증(dynamic validation)

4.3 프로젝트 관리

어디까지 SW를 만들것인가

- 소프트웨어 프로젝트에 대한 세가지 제약 조건
  - 범위, 시간, 비용
- 프로젝트 관리 활동
  - 프로젝트 계획
  - 자원관리
  - 리스크 관리
  - 프로젝트 수행과 모니터링

5. 소프트웨어공학의 원리



5.1 정형성

- 정형성(Formality)
  - 수학적 표현이 가능하고 이를 증명하는 것이 가능해야 한다
  - 프로그래밍 언어들은 정형성을 근거로 만들어짐
  - 분석 및 설계에서 사용되는 정형화 기법
  - 사용자 요구사항을 표현하는 UML 모델도 정형화의 원리

5.2 엄격

- 엄격
  - 정형화 기법 대신 사용할 수 있는 원리
  - 소프트웨어의 명세 또는 문서화를 엄격하게 함으로써 프로세스의 재사용을 향상

➢ 정형성과 엄격함은 명확한 표현을 통해서 **모호함을 줄이려는 원칙**

5.3 관심사의 분리

- 개발하는 소프트웨어의 규모가 크고 복잡해짐에 따라 해결해야 하는 문제들을 효과적으로 다루기 위한 방법
  - 소프트웨어 개발 과정에서 각 단계별, 품질별, 크기별, 역할별로 **분리하여 문제를 해결**
  - 예: 분석과정에서는 사용자의 요구사항에 집중하고, 요구사항을 명확하게 정의하는데 관심을 둠

5.4 모듈화

- SE에서 가장 많이 활용되고 있는 중요한 원리
- 소프트웨어는 수많은 모듈로 구성
  - 모듈들이 정보를 서로 교환
  - 가능한 모듈 간의 정보를 공유하거나 교환하는 횟수를 줄일 수 있는 방향으로 설계 독립적인수록 영향↓
  - 모듈화의 기준 : 응집력과 결합력
  - 객체기반개발(OOD), 컴포넌트기반 개발(CBD), 서비스기반 아키텍처 (SOA) 등이 모듈화를 기반으로 하는 기법들

5.5 추상화

- 현실세계의 주어진 문제나 요구사항을 중요하고 관련 있는 부분만을 왜곡하지 않게 추출하여 간결하게 기술하거나 만드는 작업 ex) 자바칠판도
  - 복잡한 소프트웨어 개발을 단계적으로 구현하기 위해 각 단계별 목적에 필요한 정보 이외의 내용은 생략함으로써 사용자나 개발자가 쉽게 문제를 이해할 수 있음
  - 설계 시 사용되는 디자인패턴이 대표적인 추상화 기법의 예

5.6 변화 예측

- 소프트웨어를 개발할 때 미래에 예상되는 변화를 예측하여 변화에 쉽게 대응할 수 있도록 하자는 것
- 변화예측의 대표적인 예 :
  - 3-tier 형식의 클라이언트/서버 구조 : UI와 business logic을 분리
  - SOA(Service-Oriented Architecture)와 웹서비스 : 이기종간의 통합과 연계이슈 해결을 위한 변화예측의 원리를 적용

잘변하지 않음

잘변함

플에 넣고 필요한 것만 꺼내서
→ 새로운 서비스를 플에 넣으면 됨!

5.7 일반화와 점진화

- 일반화
  - 가능하면 예외적이거나 특이한 사항이 아닌 공통적으로 활용가능한 솔루션을 제공하여 재사용성을 높이고자 함
  - 객체와 컴포넌트가 대표적인 일반화의 예
- 점진화
  - 개략적인 것부터 해결하고 순차적으로 조금씩 덧붙여서 완성해 나가는 방법
  - 분석, 설계, 구현 등으로 분리한 소프트웨어 개발과정은 점진화의 원리를 이용한 것