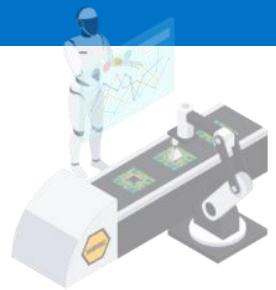
8강. 코딩과 소프트웨어 유지보수

컴퓨터과학과 김희천 교수

목차



Chapter. 1

코딩

1. 코딩 스타일

표준 코딩 스타일

- 프로그램 작성 시 준수해야 하는 코딩 스타일에 관한 지침이나 관행
- 스타일의 일관성과 구성원 간 합의가 중요함

+ 코딩 스타일의 예

- × 변수의 사용, 변수나 함수 이름의 작명
- × 주석이나 인덴테이션 사용
- × 제어 구조의 사용
- × 함수 크기와 인터페이스
- × 프로그램 문서화

2. 코드 스멜과 리팩토링

✚ **코드 스멜은 프로그램 안에 존재하는 나쁜 부분으로 문제를 야기시킬 소지가 있는 부분**

– 중복 등장하는 동일 코드, 규모가 큰 클래스, 사용되지 않는 데이터 등

✕ **리팩토링**

- 기능적 행위를 바꾸지 않고 구조를 개선하는 것
- 가독성, 유지보수성, 확장성을 높이고 복잡성을 줄이기 위함
- 버그를 잡거나 기능을 추가하는 것이 아님

✕ **코드 리팩토링은 코드 스멜을 개선하는 것**

- 코딩 스타일, 구조 뿐 아니라 성능의 개선도 고려함
- 리팩토링 과정에서 테스트가 중요함(익스트림 프로그래밍)

Chapter. 2

소프트웨어 유지보수 개요

1. 소프트웨어 유지보수

SW 유지보수

- 고객에게 인도된 후 발생되는 소프트웨어의 변경 작업
- 고객에게 인도된 후에도 변경 발생은 불가피

✕고려 사항

- 사용 조직은 시스템 획득을 위해 큰 투자를 했으며 시스템에 의존적이므로 유지보수가 중요
- 변경으로 발생하는 시스템과 요소들의 여러 버전을 관리해야 함
- 소프트웨어의 아키텍처를 변경할 수도 있음

2. 유지보수 유형

수정 유지보수
(corrective maintenance)

- 오류를 수정하기 위한 것

적응 유지보수
(adaptive maintenance)

- 외부 환경의 변화에 적응하기 위한 변경

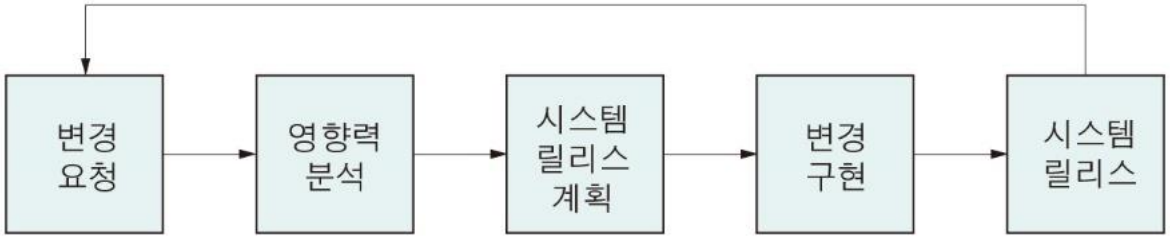
완전 유지보수
(perfective maintenance)

- 기능의 추가나 개선 및 성능 향상을 위한 변경

예방 유지보수
(preventive maintenance)

- 이해성과 유지보수성의 개선을 위한 변경

3. 유지보수 프로세스



4. 유지보수 비용과 문제점

- 응용 분야에 따라 다르나 개발 비용보다 유지보수 비용이 더 큼
- 유지보수성을 고려하여 개발하면 개발비는 증가하나, 유지보수 비용을 크게 절감할 수 있음

✕ 유지보수의 문제점

- 자주 개발 팀과 유지보수 팀이 다르며, 이때 유지보수를 위해 먼저 시스템을 이해해야 함
- 상대적으로 능력이 뛰어나지 못한 인원이 유지보수 팀에 배정되는 경향
- 자주 유지보수 계약과 개발 계약이 별개임
- 시간이 지나면서 계속되는 변경으로 인해 소프트웨어 구조와 가독성이 떨어짐

Chapter. 3

재공학과 역공학

1. 레가시 시스템과 재공학

✚ 레가시(legacy) 시스템

- 과거에 개발되어 사용되고 있으며 사업적으로 중요한 시스템
- 레가시 시스템의 진화를 준비하기 위해 시스템을 재공학함

✚ 재공학(reengineering)

- 기존 시스템의 이해도 개선, 유지보수성의 개선, 재사용성의 향상을 위해 소프트웨어를 변경하는 작업

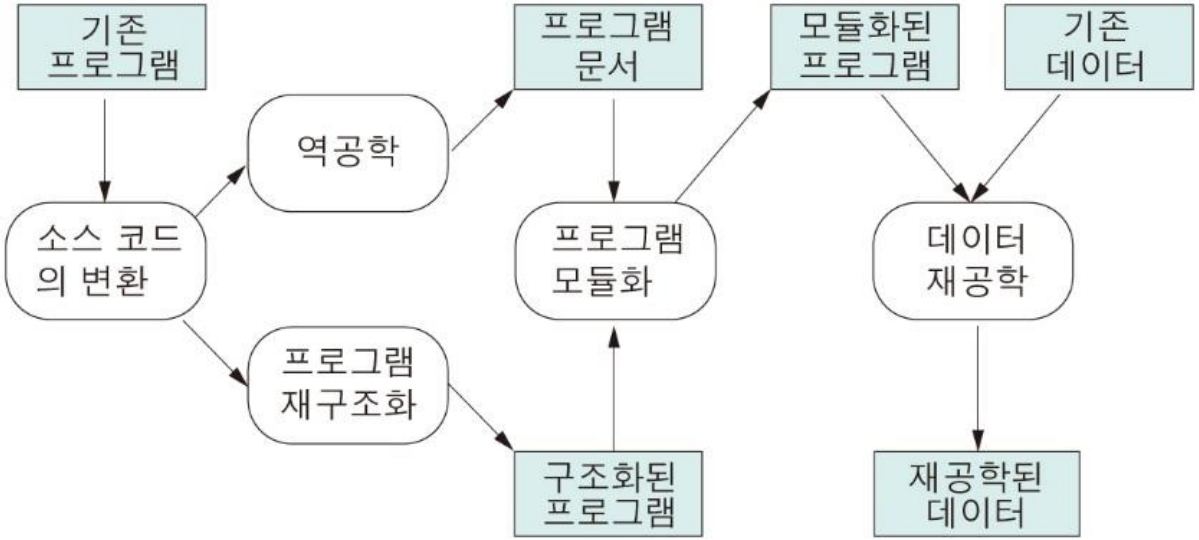
✕ 재공학과 새로 개발하는 것과의 비교

- 재공학의 경우 위험과 비용이 축소됨
- 재공학은 기존 시스템을 요구 명세서로 생각하는 작업
- 재공학에 의한 개선 정도는 새로운 개발에 비해 제한적

2. 재공학 프로세스(1/2)

- 입력은 레가시 시스템
- 출력은 같은 기능을 가지나 구조화되고 모듈화된 버전
- 재공학 프로세스의 활동
  - 소스 코드의 변환: 프로그래밍 언어를 최신의 것으로 변경
  - 역공학: 소스 코드를 분석하여 설계 문서를 추출
  - 프로그램 재구조화: 프로그램의 제어 구조를 개선하는 일
  - 프로그램 모듈화: 전체 모듈 구조를 개선하는 일
  - 데이터 재공학: 데이터 구조를 변경

2. 재공학 프로세스(2/2)


기존
프로그램
역공학
소스 코드
의 변환
프로그램
재구조화
프로그램
문서
프로그램
모듈화
구조화된
프로그램
모듈화된
프로그램
데이터
재공학
재공학된
데이터
기존
데이터

3. 재공학 비용

- 재구조화 작업이 소스 코드 변환 작업보다 비용이 큼
- 아키텍처를 바꾸는 경우 가장 큰 비용이 발생함

✕ 비용 발생 요인

- 소프트웨어와 관련 문서의 품질이 낮은 경우
- 재공학을 위한 도구를 사용할 수 없는 경우
- 재공학해야 하는 데이터의 양이 많은 경우
- 유지보수 전담 요원이 참여하지 못하는 경우

4. 역공학

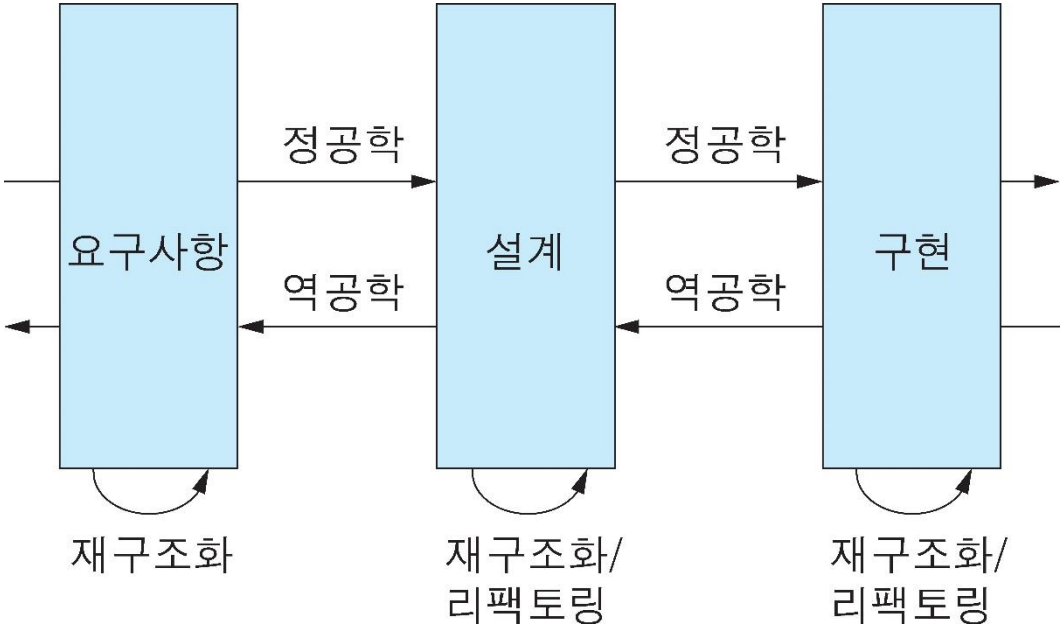
역공학

- 후기의 생성물에서 이전의 생성물을 추출하기 위해 과학적 지식을 이용하는 것
- 유지보수를 위해서는 시스템에 대한 이해가 선행되어야 함

✚ 소프트웨어 공학 관점에서의 역공학

- ✕프로그램이나 사용자 매뉴얼 등으로부터 기능 명세나 설계 문서 등을 만드는 과정
- ✕시스템을 분석하여 구성 요소들과 그들의 관계를 추출하는 것
- ✕목적 코드에서 소스 코스를 유도하는 것
- ✕재문서화를 간단한 역공학의 예로 보기도 함

5. 역공학 프로세스



6. 재구조화와 재문서화

✚ 재구조화

- 같은 추상 수준에서 문서의 구조를 개선하는 일
- 자주 코드의 재구조화를 의미(코드 리팩토링)

✕ 문서의 복구 또는 재문서화

- 문서를 새로 만드는 일
- 코드, 기존 문서 등으로부터 설계 문서를 유도하는 일(설계의 복구)
- 재문서화는 문서의 의미를 바꾸지 않고 개조하는 일

Chapter. 4

소프트웨어 형상 관리

1. 소프트웨어 형상 관리

✚ 소프트웨어 시스템의 변경을 제어하고 관리하기 위한 프로세스

- 변화하는 시스템을 관리하기 위해 표준과 절차를 사용하는 것
- 특정 시점에서 시스템을 구성하는 요소들이 무엇인지 파악하고 기록하는 것
- 설계 문서와 코드가 작성 완료(기준 문서)된 후에는 공식 검토를 거쳐야만 변경 가능

✖ 형상 관리의 효과

- 구성 요소들을 조직화하여 부주의로 인한 수정 가능성 감소
- 변경으로 인한 혼란의 최소화
- 버전 관리를 통해 요소의 재사용성 증가

2. 형상 관리 계획

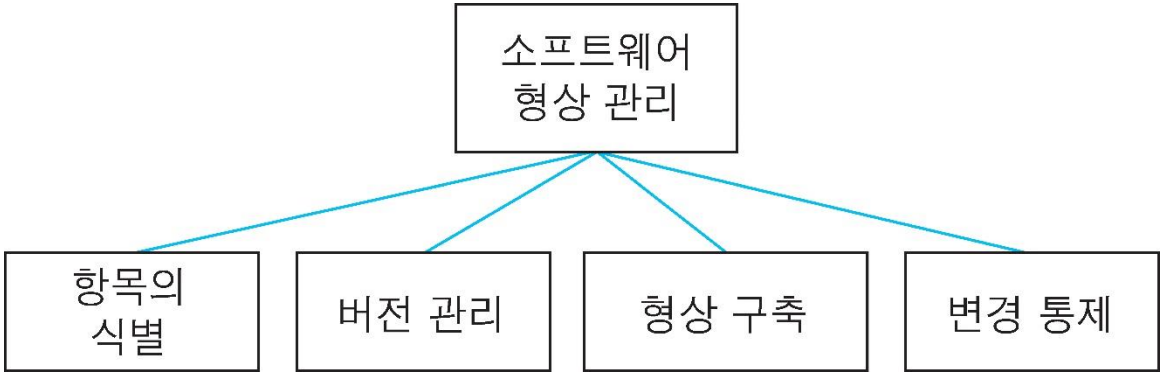
✚ 형상 관리에 사용되는 표준과 절차를 기술하는 것

– 조직 내부의 표준과 외부 형상 관리 표준에 기초함

✕ 형상 관리 항목(SCI)

- 계획서, 요구사항 명세서, 설계 문서
- 각 모듈별 소스 코드, 실행 코드
- 테스트 계획, 테스트 케이스, 테스트 스크립트
- 시스템 구축에 사용되는 도구들, 컴파일러, 링커, 구문 분석기, 파서 등
- 개발 보고서

3. 소프트웨어 형상 관리 활동



4. 항목의 식별

✚ 형상의 구성 요소로 포함되어 관리될 항목들을 정하고 항목의 특성을 기록하는 것

✚ 각 요소를 구분하기 위해 항목의 유형과 이름 및 버전 번호가 필요함

– 이것 외에 설명, 날짜, 책임자에 대한 정보, 다른 항목과의 관계도 기록

– 형상 자체도 버전 번호를 가짐

✖ 형상 관리 항목이 개발되어 공식적으로 인정된 상태를 기준선이라 함

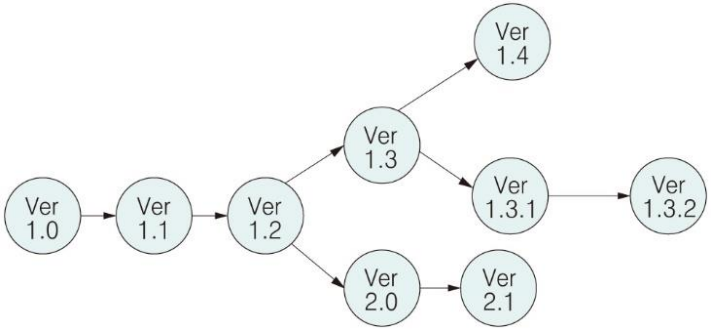
- 기준선이 설정된 이후의 변경은 공식적 승인을 얻어야 함

- 여러 항목들을 묶어서 기준선을 설정할 수도 있음

5. 버전 관리

- 개별 형상 항목과 시스템 형상은 진화되므로 누가 무엇을 언제 변경했는지 기록하여 이력을 제공하는 버전 관리가 필요함

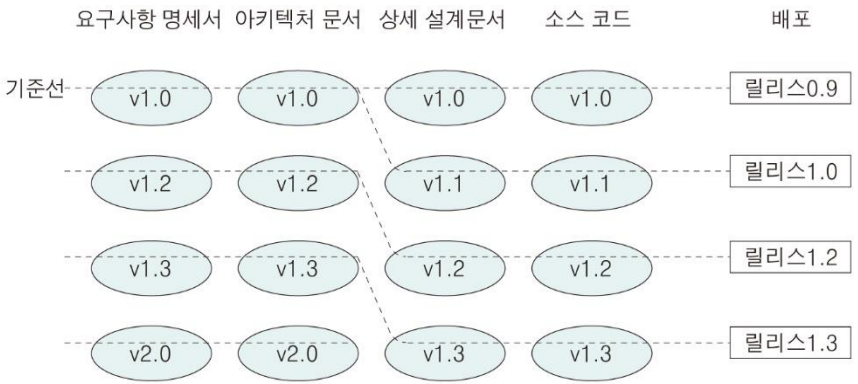
- 소스 코드의 버전을 관리하는 경우
  - 소스 코드의 이름과 버전 및 변경 사항
  - 사용된 컴파일러와 링커의 버전
  - 만든 사람의 이름
  - 만든 날짜와 시간



6. 형상 구축과 변경 통제

✚ 형상 구축

– 시스템 형상을 만들기 위해 그것을 구성하는 요소들의 정확한 버전을 파악하고 연결해야 함



✕ 변경 통제(또는 형상 통제 프로세스)

- 형상 관리 항목들에 대한 변경을 통제하기 위한 의사결정 프로세스
- 변경 제어 프로세스, 형상 상태의 보고, 형상 감사를 포함함

7. 변경 제어 프로세스(1/2)

- 변경을 제안하고 승인을 받아 구현하는 절차
- 변경의 요청, 변경의 파급 효과 분석, 변경의 승인(또는 기각), 변경 작업, 새로운 버전의 생성, 형상의 구축이 필요함
- 변경된 항목의 새로운 기준선과 함께 구 버전도 라이브러리에 유지되어야 함

7. 변경 제어 프로세스(2/2)

변경 제어 프로세스 절차

- 수정의 필요성 인식, 변경 요구서 제출
- 형상관리위원회가 승인을 결정
- 개발자를 배정하고 형상 항목을 체크 아웃
- 변경 작업, 변경에 대한 검토와 감사
- 품질 평가와 체크인
- 새로운 버전의 생성
- 시스템 릴리스에 변경 내용을 반영

형상관리위원회(CCB)는 형상 관리를 책임지며 변경 요청을 평가함

Chapter. 5

소프트웨어 재사용

1. 재사용 단위

- **애플리케이션 시스템 재사용**
  - COTS 재사용
  - 공통 도메인 아키텍처를 사용하여 애플리케이션 패밀리를 개발
- **컴포넌트 재사용**
  - 하나의 클래스에서 서브시스템 정도의 규모
- **함수 재사용**
  - 전통적 표준 라이브러리

2. 재사용의 장단점

장점

- 개발 및 유지보수 비용의 절감, 빠른 인도와 품질의 향상
- 검증된 컴포넌트이므로 신뢰도 증가
- 프로젝트 위험의 감소
- 컴포넌트 개발에 특화된 전문가를 활용
- 표준을 준수하는 컴포넌트
- 개발과 검토 시간의 단축

단점

- 소스 코드가 없어서 유지보수가 어려움
- CASE 도구와 연동되지 못하는 컴포넌트
- 재사용에 대한 거부감
- 컴포넌트 라이브러리의 유지 문제

Chapter. 6

소프트웨어 척도

1. 소프트웨어 척도

✚ 소프트웨어 척도는 제품이나 개발 프로세스의 특성을 평가하기 위한 정량적 측정법

- 프로젝트 관리에 필요한 정보를 얻기 위함
- 프로세스의 개선에 필요한 정보를 얻기 위함
- 제품의 품질을 개선하기 위한 정보를 얻기 위함

✕ 유지보수에 영향을 주는 요인

- 모듈화 정도, 모듈의 복잡도, 모듈의 가독성, 요구사항의 추적성, 문서화 수준

2. 소프트웨어 척도의 분류

제품 척도

- 제품의 특성을 표현하는 척도
- 크기, 복잡도, 설계 특성, 성능, 신뢰도 등과 같은 제품의 특성

프로세스 척도

- 프로세스의 효율성과 프로세스 품질을 나타내는 것
- 프로세스의 수행 비용, 제품 생산에 걸린 시간, 프로세스 성숙도, 결함 발견과 제거의 효율성

프로젝트 척도

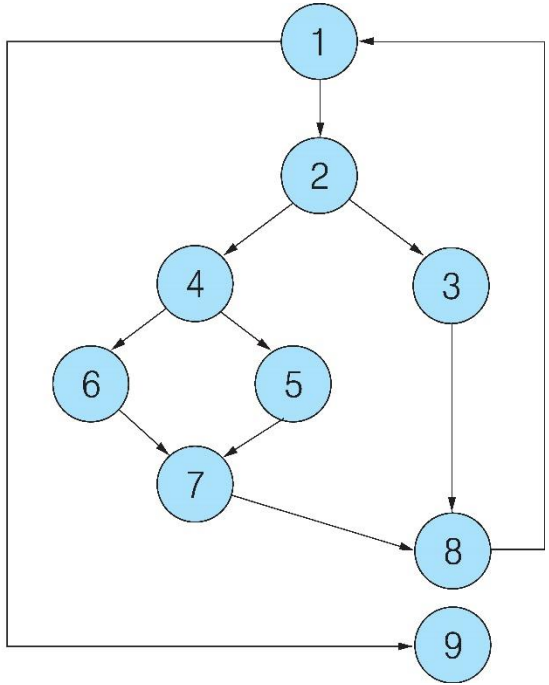
- 프로젝트의 특성을 나타내는 것
- 개발자의 수, 비용과 일정, 생산성, 인사 패턴 등

3. 사이클로매틱 복잡도(1/2)

- 유지보수를 위한 수정 작업은 모듈화 정도와 모듈의 복잡성 및 가독성에 큰 영향을 받음
- 모듈의 복잡도를 평가하기 위해 제어 구조와 논리적 복잡성 등을 고려하는 **복잡도 척도**가 존재함
- 사이클로매틱 복잡도
  - 매케이브의 그래프 이론으로 프로그램에 존재하는 독립적인 경로의 수로 복잡도를 평가
  - 사이클로매틱 수 = 제어 흐름 그래프에서 영역(region)의 수
    = 에지 수 - 노드 수 + (2 × 연결 요소의 수)
    = 비교 횟수 +1

3. 사이클로매틱 복잡도(2/2)

```
1: while(not EOF)
2: { read record;
2:    if(field1 equals 0)
3:       { add field3 to total;
3:         increment counter; }
4:    else
4:       { if(field2 equals 0)
5:            { subtract field3 from total;
5:              increment counter; }
6:         else
6:              reset counter;
7:         print counter; }
8:     print  “End Record”; }
9: print total;
```



4. 소프트웨어 사이언스

할스테드의 이론으로 코드에서 측정한 값으로 복잡성을 계산

- 프로그래밍 또는 디버깅에 드는 노력을 예측하는데도 사용

척도

- 프로그램 길이 N = N1 + N2 또는
  $N = n1 \times \log_2 n1 + n2 \times \log_2 n2$
- 프로그램 부피 V = N × $\log_2$ n (단, n=n1+n2)
- 프로그램 난해성 D = n1/2 × N2/n2
- 프로그램 노력 E = V × D

(n1=프로그램에서 사용된 유일 연산자의 수, n2=프로그램에서 사용된 유일 피연산자의 수, N1=프로그램에서 사용된 연산자의 총 수, N2=프로그램에서 사용된 피연산자의 총 수)

다음강의

9강. 객체지향 분석과 설계