Object Oriented Software Engineering OOSE

유지보수(Maintenance)

- 유지보수?
- 유지보수 절차
- 유지보수 기법
- 형상관리(Configuration Managements)
- 재공학(Re-engineering)

Computer & Software Engineering
Lee Jong Yeol
soyeum@kumoh.ac.kr

소프트웨어 유지보수(1)

- 유지보수란?
  - 소프트웨어를 설치한 후에 정상적으로 작동하도록 유지 관리하는 작업
  - 오류 수정, 기능 확장, 성능 개선, 변화된 환경에 대한 적응 등 소프트웨어의 발전을 위한 작업
  - 유지보수를 위해서는 소프트웨어 개발과정 전반에 관한 높은 기술력이 요구됨
  - 유지보수에서 가장 중요한 의사결정
    - 가동 중인 소프트웨어를 계속 보수할 것인가?
    - 폐기하고 새롭게 개발할 것인가?

소프트웨어 유지보수(2)

- 유지보수의 종류
  - 수정 보수 (correction)
    - 운용하는 동안 발견된 오류를 수정하는 작업
    - 가능한 한 신속하게 대응해야 함
  - 적응 보수 (adaption)
    - 운용 환경의 변화에 적응시키기 위한 변경 작업
    - 하드웨어 교체, OS 갱신 및 교체, 네트워크 환경 변화, 우편번호 체계 변경, …
  - 개선 보수 (enhancement)
    - 소프트웨어의 기능을 확장하거나 향상시키기 위한 변경
    - 요구 변경, 설계 개선, 개발방법론의 진보, …
  - 예방 보수 (prevention)
    - 소프트웨어를 조사하여 잠재적인 오류를 제거하는 작업

소프트웨어 유지보수(3)

- 유지보수에 필요한 노력
  - 소프트웨어 생명주기 전체에서 차지하는 비중
    - 요구단계: 6%, 설계단계: 5%
    - 구현단계: 7%, 단위 테스트: 8%
    - 통합 테스트: 7%, 유지보수: 67%
  - 유지보수 유형별 분포
    - 수정보수: 17.5%
    - 적응보수: 18%
    - 개선보수: 60.5%
    - 기타: 4%

유지보수의 특성

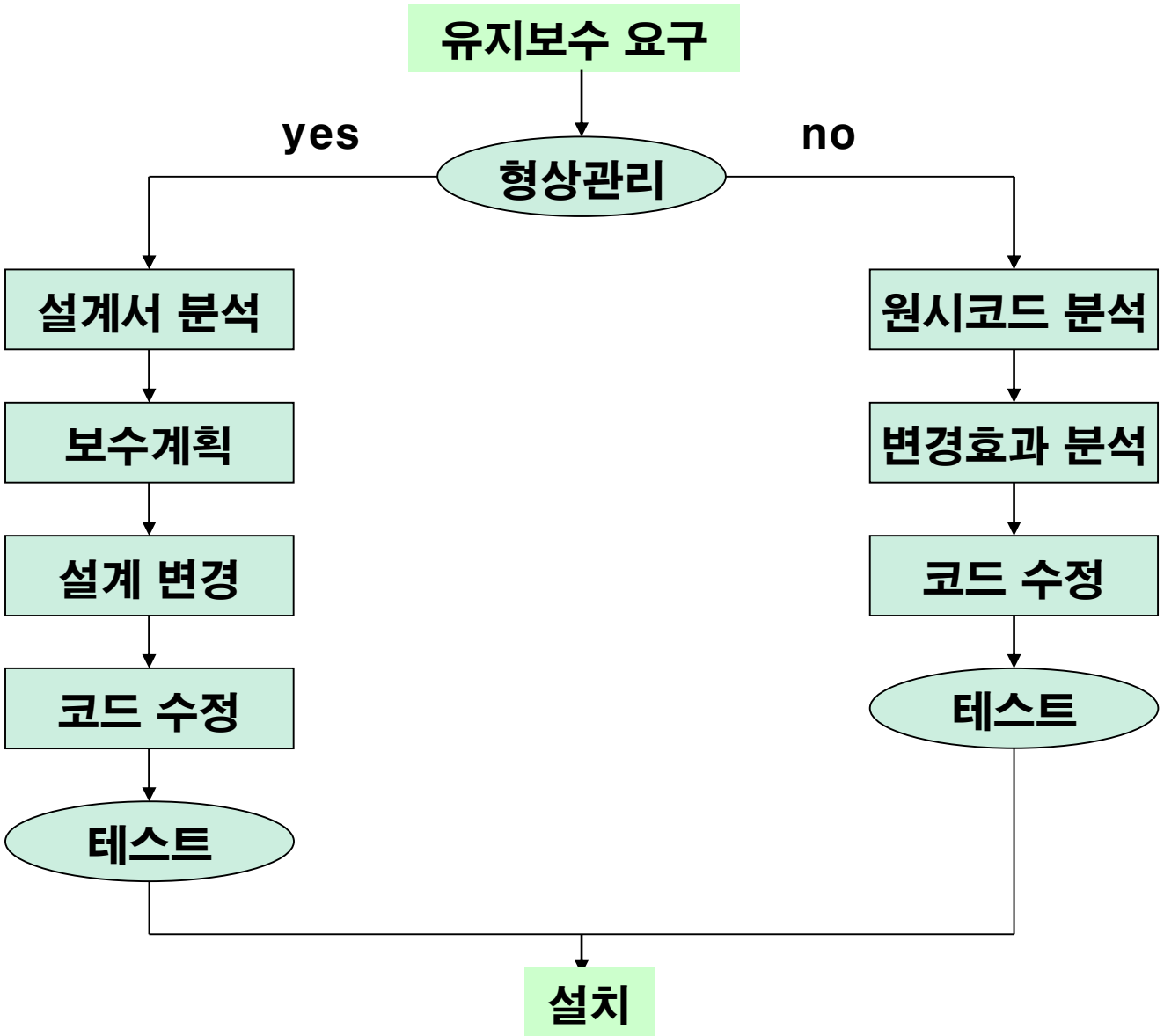
- 소프트웨어의 변화 원리 (Lehman의 관찰)
  - Law of continuing change
    - 소프트웨어는 계속 변화 -> 유지보수는 불가피
  - Law of increasing complexity
    - 시스템이 변경될수록 구조는 복잡해짐
  - Law of program evolution
    - 소프트웨어의 유지보수는 시스템별로 고유한 특성을 가짐
  - Conservation of organizational stability
    - 안정화 상태에 들어가면 개발 생산성이 일정한 수준을 유지
  - Conservation of familiarity
    - 시스템 개발의 모든 단계에서 각 버전의 변화는 거의 일정

OOSE 유지보수 작업

- 대상 소프트웨어의 성격에 따라 조금씩 다름

- 유지보수의 기본적인 작업 단계
  - 1단계: 소프트웨어의 이해
    - 요구명세서, 설계문서, 사용자 매뉴얼 -> 시스템의 윤곽 파악
    - Source code reading
  - 2단계: 변경요구에 대한 분석
  - 3단계: change effect 예측
    - 변경으로 인하여 영향을 받는 부분을 파악
  - 4단계: 소프트웨어 변경 및 Regression test
    - 변경으로 인하여 영향을 받는 부분에 대한 테스트

유지보수 절차


유지보수 요구
형상관리
yes
no
설계서 분석
보수계획
설계 변경
코드 수정
테스트
원시코드 분석
변경효과 분석
코드 수정
테스트
설치

OOSE 유지보수의 어려움

- 유지보수의 어려움
  - 수시로 발생하는 변경에 대한 문서화를 제대로 하지 않을 경우 이에 대한 추적이 거의 불가능
  - 다른 사람이 작성한 코드를 이해하는 것은 어렵다
    - 주석이 제대로 되어 있는 않는 경우에는 거의 절망적!!
  - 개발과정에서 변경에 대한 대비를 거의 하지 않는다
  - 유지보수는 가치 있는 일이 아니라는 관리자의 인식
  - 유지보수를 위한 도구 사용이 부족

- 발생원인
  - 개발과정에서 문서화를 하지 않음 !!!
  - 소프트웨어공학의 원리를 준수하지 아니함!!!
    - 특히, 모듈화를 제대로 하지 아니함

유지보수 기법

- 유지보수에 사용되는 공학적 기법
  - 형상관리 (configuration management)
  - 변경영향 분석 (change effect analysis)
  - 회귀 테스트 (regression test)
    - 변경 후에 소프트웨어가 정상적으로 동작하는가?
    - 변경되지 않은 부분도 정상적으로 동작하지 않을 수 있음
    - 변경 전에 적용하였던 테스트 케이스를 그대로 사용
  - 소프트웨어의 시각화 (software visualization)
    - 구성요소 사이의 관계, 변경부분 등을 이해하기 쉬운 형식으로 표현
    - 예) AT&T의 SeeSoft
  - 재공학 (re-engineering)
  - 프로그램의 이해

형상관리(1)

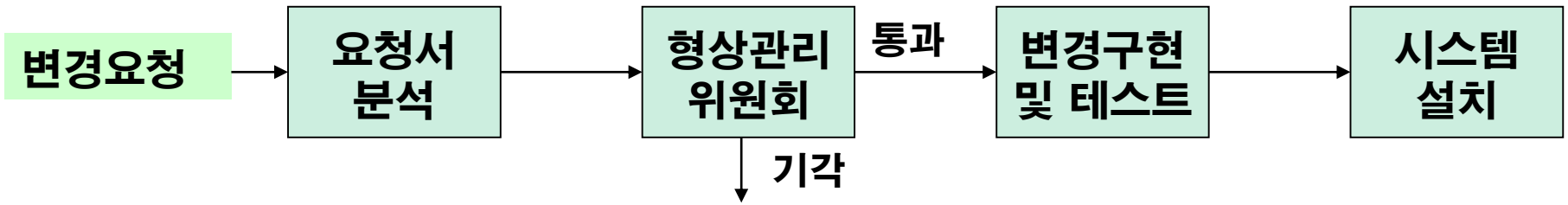
- 형상관리란?
  - 정의: 소프트웨어의 개발 및 유지보수 과정에서 발생하는 변경을 체계적으로 관리하는 것
  - 형상관리 항목 (configuration item)
    - 시스템 명세서, 프로젝트 계획서
    - 요구명세서, 예비 사용자 매뉴얼
    - 설계문서: 구조설계, 자료설계, UI설계, 모듈설계의 문서
    - 소스 코드 목록 및 소스 코드
    - 테스트 문서: 계획 및 절차, 테스트 케이스, 수행 결과
    - 유지보수 문서: 변경요청서, 변경처리 보고서
    - 소프트웨어공학과 관련된 표준 및 절차
  - 형상관리정보
    - 어떤 프로젝트의 어느 단계에서 누구에 의해 작성되었는가?
    - 문서의 관리책임은 누구에게 있는가?

형상관리(2)

- 버전(version)과 릴리즈(release)
  - 버전: 고객의 요구사항을 만족하는 특정한 구성(snapshot)
  - 릴리즈: 이전의 소프트웨어를 수정 또는 개선한 것
  - 관리번호 예시) 0.7, 2.4, …

❑ 형상관리 도구
- UNIX의 SCCS (Source Code Control System)
- CVS (Concurrent Versions System)
- SVN (Sub VersioN)

❑ 변경제어절차



변경영향(change effect) 분석

- 영향분석(파급효과해석)이란?
  - 유지보수는 변경할 부분 외에 다른 많은 부분에 영향
  - 정의: 소프트웨어를 변경하기 전에 영향 범위를 해석하여 유지보수에 필요한 자원이나 공수를 산정하는 작업

- 영향분석 기법
  - 산출물(소스 코드 포함)을 대상
  - 영향범위 파악에 사용 가능한 정보
    - 상호참조(cross reference) 정보로부터 영향범위 파악
    - 프로그램 내부에 존재하는 제어흐름과 자료흐름을 추적하여 영향범위를 파악 -> program slicing 기법
    - 수직추적가능성: 개발단계 내부의 문서 사이의 종속관계
    - 수평추적가능성: 서로 다른 개발단계의 문서 사이의 종속관계

OOSE 재공학 (1)

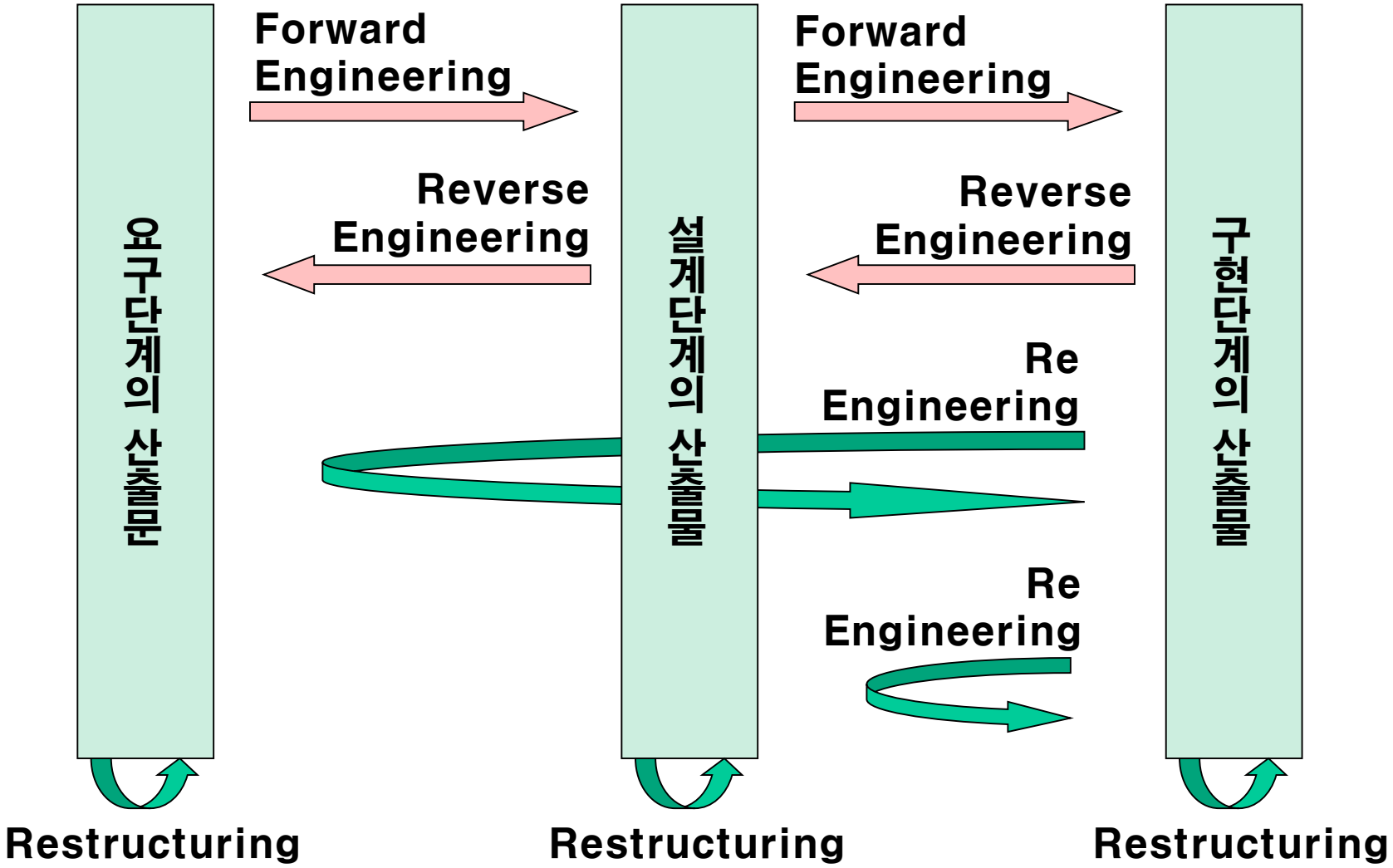
- 재공학(Re-Engineering)이란?
  - 기존의 소프트웨어를 고쳐서 새롭게 구성하는 것
  - 역공학 + 순공학
    - 역공학(Reverse Engineering)
      : 소스 코드로부터 설계문서 또는 요구명세를 찾아내는 작업
      - Design Recovery: 소스 코드로부터 설계정보를 생성
    - 순공학(Forward Engineering)
      : 요구명세로부터 설계문서, 설계문서로부터 소스 코드를 만드는 작업

- 소프트웨어 재구성 (Restructuring)
  - 개발단계 내부에서 산출물을 새롭게 변환하는 작업
  - 예) 내부 알고리즘 교체를 위한 소스 코드 변환 작업
  - 리팩토링(Refactoring)은 재구성 작업의 일종

OOSE 재공학 (2)

- 리팩토링이란?
  - 소프트웨어의 외부 동작은 그대로 유지하면서 내부구조를 재구성하는 작업
  - 대규모의 설계변경 작업을 일련의 작은 변환으로 분할하여 실현하는 것을 전제로 함
  - 작은 규모의 변환을 수행할 때마다 테스트를 실시하여, 소프트웨어의 동작을 확실하게 보전
  - 일련의 변환작업은 기록을 유지
  - 유지보수의 반복에 따라 악화된 소프트웨어 구조를 개선하는데 중요한 역할을 담당

재공학 (3)


Forward Engineering
Reverse Engineering
Forward Engineering
Reverse Engineering
Re Engineering
Re Engineering
요구단계의 산출문
설계단계의 산출물
구현단계의 산출물
Restructuring
Restructuring
Restructuring

프로그램의 이해

- 프로그램 이해의 중요성
  - 프로그램=실제로 가동되고 있는 소프트웨어의 실체
  - 다른 문서들이 프로그램과 정확하게 대응하지 않는 경우에 유지보수에서 유일하게 믿을 수 있는 것은 프로그램 밖에 없다

- 프로그램 이해 기법
  - 제어흐름 분석: 프로그램 내부의 제어흐름을 도식화
  - 자료흐름 분석: 자료구조 정의 및 참조 관계를 해석
  - 프로그램 슬라이싱 (program slicing)
    - 프로그램 슬라이스: 특정 변수와 관련된 문장의 집합
    - 원래 디버깅을 위해 제안된 개념
    - 변경영향분석, 프로그램 이해 등의 다양한 용도로 이용

OOSE 유지보수 지원도구(tools)

- 유지보수 지원도구
  - 기술지원도구
    - 소스코드 이해를 위한 도구
      : cross reference table, call graph, data flow chart, system structure chart
    - 변경영향 분석도구
    - 변경 후의 테스트 도구: comparator
    - 프로그램 특성을 측정하는 도구
  - 관리지원도구
    - 버전관리도구: UNIX의 SCCS, RCS
    - 형상관리도구: UNIX의 Make