Chap. 11-2
소프트웨어 유지보수

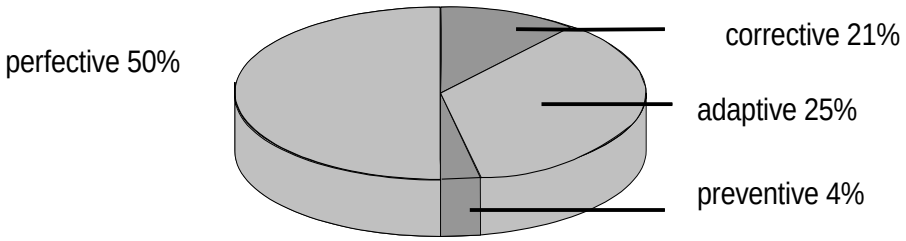
Software Engineering

유지보수의 정의

- 유지 보수(maintenance)
  - 소프트웨어가 제품으로 개발된 후 사용자가 사용하기 시작하면서부터 폐기될 때까지 오류를 수정하거나 새로운 기능을 추가하기 위해 소프트웨어를 변경하는 과정

소프트웨어 유지보수의 형태

1) Corrective maintenance(수리 유지보수)



- 발견된 오류의 문제를 찾아 문제해결
- A/S의 개념

2) Adaptive maintenance(적응 유지보수)

- 새로운 하드웨어나 차기 운영 체제와 같은 환경변화를 소프트웨어에 반영하는 것
- 시스템의 기능에 변화와는 관련이 없음

3) Perfective maintenance(완전화 유지보수)

- 주로 새로운 혹은 변경된 사용자 요구 사항을 수용하는 것을 다룸
- 시스템의 기능적 향상에 관한 것으로 시스템의 성능을 증가 시키거나 사용자 인터페이스를 향상시키기 위한 활동을 포함
- 유지보수 활동 중 가장 큰 업무량 및 비용을 차지

4) Preventive maintenance(예방 유지보수)

- 문서 갱신, 설명 추가, 그리고 시스템의 모듈 구성을 향상시키는 것과 같이 시스템의 유지 보수성을 증가시키는 것을 목표로 한 활동에 관한 것

- **유지 보수 문제 줄이는 방법을 찾을 때 고려할 수 있는 솔루션**
  - 고급코드, 더 좋은 시험 절차, 더 좋은 문서 그리고 표준과 규정에 맞추는 것 등이 사후 유지보수를 줄이는데 도움이 될 수 있음
  - 요구사항 분류 및 설계 시에 변경을 고려하고, 구현에도 고려함으로써 미래의 기능개선 적응 유지보수가 더욱 쉬워질 수 있음
  - 사용자 요구에 대한 세부 조정으로 완전화 보수(perfective maintenance)를 줄일 수 있음
    - 예를 들어 요구분석과 설계단계에서 프로토타이핑 기법과 좀더 강도 높은 사용자의 참여를 통해서 이루어 질 수 있음
  - 코드의 수가 적으면 보수의 필요성도 줄어듦
    - 원시 코드의 길이는 초기 개발 시 뿐만 아니라 유지보수 동안에 전체 비용을 결정하는 주요 요소임
    - 특히, 100 LOC에서의 20%변경보다 200 LOC에서의 10%변경에 더 많은 비용이 들기 때문에 소프트웨어 의 재사용은 유지보수비용에 매우 직접적인 영향을 가하게 됨

유지보수의 문제

- 유지보수의 문제
  - 현재 존재하는 소프트웨어의 에러를 어떻게 할 것인가?
  - 개발이 끝난 소프트웨어를 정해진 목표 소프트웨어로 변환하기 위해서는 어떻게 할 것인가?
  - 지금까지 개발하는 소프트웨어가 목표환경에 맞지 않을 경우에 어떻게 대처할 것인가?
  - 지금부터 개발하는 소프트웨어를 장래의 환경변화에 따라서 대처하기 쉽도록 문제 레벨과 해답 레벨의 양쪽에서 준비하는 것은 어떻게 하면 좋은가?

유지보수의 문제들의 주요 원인

- 소프트웨어는 다른 제품과는 달리 무형이므로 자유로이 변경할 수 있다고 생각하는 경우가 많다.
  - 이 때문에 불필요한 변경의뢰가 사용자로부터 생기게 된다.
  - 그것은 사용자들이 소프트웨어는 완전한 블랙박스이며 실체가 없는 것이라고 생각하기 때문이다.
- 유지보수에 관한 요구정의를 명확히 표현하기 위한 기법이 없다.
  - 신규개발의 소프트웨어와 마찬가지로 요구정의의 불명확함이 여러 가지 트러블을 초래한다.
- 소프트웨어 작성공정이 장시간에 걸쳐 사양동결을 전제로 하기 때문에 아무래도 본가동 후 금방 변경의뢰가 발생하게 된다.
  - 특히 대규모적인 소프트웨어일수록 이 경향이 커지게 된다.
- 개발측의 버그(bug)와 사용자측의 요구변경의 구별이 다음으로 많게 된다.
  - 이 때문에 보수비용의 부담이 어느쪽이 되느냐에 의해 트러블이 일어나기 쉽다.
- 체계화된 유지보수기술 지원도구가 거의 개발되어 있지 않고, 부분적인 지원 도구 정도밖에 없다.
  - 이 때문에 보수공정의 생산성, 신뢰성이 매우 떨어지게 된다.

- **소프트웨어 설계 단계에서 보수를 고려한 방법론, 기법이 제안되어 있지 않다.**
  - 보수를 생각하면, 소프트웨어의 이해용이성에 대한 기준이 설계상에서의 포인트가 되지만 효과적인 대책이 없는 상태이다.
- **유지보수를 수행하기 위하여 필요로 하는 다큐먼트가 준비되지 않는 상태에 있다.**
  - 개발시에 작성된 다큐먼트가 그래도 보수용 자료로서 대신 사용되고 있는 것에 불과하다.
- **소프트웨어의 규모가 커질수록 다큐먼트도 잘 정리되지 못한다.**
  - 소프트웨어 규모의 증대에 의하여 다큐먼트 사이의 인터페이스가 복잡하게 되며, 관리하기가 벅찬 다큐먼트 양이 된다.
- **유지보수 공정 가운데에서 다큐먼트의 품질이 열화되어 가는 경향이 많다.**
  - 그것은 원시 프로그램의 변경을 수행한 후에 다큐먼트의 수정까지 손을 쓰지 못한 상태가 된다.
  - 이 결과로 최신의 원시 프로그램과 매칭성이 취해지지 않은 다큐먼트가 남게 된다.
- **유지보수 작업을 계속해 가면 개발단계에서는 구조화되어 알기가 쉽던 소프트웨어도 서서히 복잡하게 되어 버린다.**
  - 프로그램의 구성 및 다큐먼트도 이와 같은 경향이 되어서 소프트웨어의 품질이 상대적으로 저하하게 된다.

- 소프트웨어의 수정에는 반드시 파급효과가 생겨서 그것에 의하여 잠재적인 에러가 새로이 축적되기 쉽다.
  - 파급효과를 줄이기 위한 유지보수기법이 제안되어 있지 않다.
- 타인이 작성한 소프트웨어를 다시 고치는 것은 자신이 작성한 것을 고치는 것보다 곤란하며, 잘못을 유발할 가능성이 높다.
  - 소프트웨어 개발공정에서 생겨난 작성자의 생각이나 사상까지 이해하지 않으면 소프트웨어의 전체가 파악되지 못하기 때문이다.
- 개발에서부터 유지보수로의 인계시에 전제사항이 불투명하게 되기 쉽다.
  - 특히, 개발담당자와 유지보수담당자가 서로 다른 경우에 문제가 발생하기 쉽다.
- 유지보수의 이력, 경과 정보가 거의 관리되어 있지 않다.
  - 이 때문에 유지보수요원을 교체할 때에 대응이 되지 않는다든지, 소프트웨어 유지보수에 일관성 있는 체제가 되지 못한다.
- 유지보수작업의 견적 예측이 정확히 수행되지 못한다.
  - 유지보수에 드는 경비와 노력이 예정대로 되지 않으므로 유지보수작업을 관리하기가 어렵다.
- 유지보수작업의 관리가 어렵다.
  - 유지보수작업은 그 해당되는 사람 이외에는 이해되지 않는 경우가 많다. 변경의뢰에 의한 유지보수의 납기는 개발보다 짧다. 또한, 유지보수에 대한 스케줄도 설정하기 어렵다.

- **유지보수 요원의 작업에 대한 유도가 저하하는 경우가 있다.**
  - 창조성이 없는 작업 내용이나 타인의 생각을 이해하여 그것에 맞추는 수밖에 없으면, 자기표현 의장이 설정될 수 없다.
- **유지보수 요원으로서 경험이 적은 신인 등이 배치되는 경우가 많다.**
  - 유지보수를 실시 하기 위해서는 담당의 경험과 지식이 요구되지만 유지보수작업 자체가 쉽게 받아 들여져 버리는 결과로 이러한 현상이 생기게 된다.
- **유지보수요원의 배치전환은 매우 어렵다.**
  - 유지보수공정 자체는 상당한 수년 사이에 계속적으로 지속되기 때문에 배정된 사람이 그대로 유지보수작업 하기가 어렵다.
  - 유지보수하고 있는 노하우가 그 사람밖에 축적되어 있지 않는 것도 원인이라고 할 수 있다.
- **유지보수작업의 노동형태가 과로하기 쉽다.**
  - 납기가 짧고, 낮에는 본래의 시스템이 가동되고 있기 때문에 야간에 소프트웨어의 변경을 하여야 하며, 잔업의 증가를 초래한다.
- **업계 전체가 소프트웨어의 유지보수에 대한 중요성의 관심이 낮다.**
  - 소프트웨어의 설계, 개발, 테스트 등의 테마에만 관심이 집중되어 있어서 유지보수에 관한 연구나 실용화에 어프로치가 미흡하다.

유지 보수성의 척도

- 유지보수성
  - *얼마나 유지보수하기 쉽게 고려되고 있는가?*
  - 시스템을 유지보수하기 위해 어느 정도의 노력이 필요한가를 예측하기 위해 시스템의 유지보수성을 평가하는 방법이 요구

- 유지보수성을 나타내는 척도(metrics)
  - 프로그램의 복잡도와 관련
  - 프로그램의 복잡도를 측정하여 높은 복잡도의 수치가 나오면 시스템 성분을 유지하는데 많은 어려움이 있다는 것을 나타냄
    - 프로그램의 복잡도 측정법
      - Halstead, McCabe 등에 의한 방법론 (p.363)
  - 절대적 의미가 아닌 상대적 의미를 지님

시스템의 재구성

- 시스템의 재구성
  - 계속된 변경으로 인해 원래 시스템의 구조를 판별할 수 없거나 점차적인 시스템의 변경에 따른 비용이 높을 때 ⇒ 시스템을 재작성하거나 완전히/부분적으로 재구성
  - 재구성(Restructuring)
    - 현 시스템을 검사하여 전체 구조를 개선하기 위해 시스템의 일부를 재작성하는 것
      - 변경이 시스템의 일부로 국한될 때 특히 유용
      - 필요한 부분만 재구성하고 나머지 부분은 변경하거나 재검증 될 필요가 없음
  - 고려사항
    - 변경부분과 빈도
      - 대부분의 시스템이 안정적이고 자주 변경하지 않는다면, 변경될 프로그램의 일부를 재구성하는 것이 바람직
    - 지원 소프트웨어에 대한 의존성
      - 지원 소프트웨어가 구식이고 의존도가 높다면, 재작성되는 것이 바람직
    - 재구성의 존재여부
      - 재구성 툴이 유용하지 않다면 수동적인 재구성을 선택

유지보수의 부작용

- 유지보수의 부작용
  - 코딩 부작용
    - 코딩 내용의 변경으로 인하여 발생하는 부작용
  - 자료 부작용
    - 자료(자료구조) 변경으로 인하여 발생하는 부작용
  - 문서화 부작용
    - 자료코드에 대한 변경이 설계 문서나 사용자가 사용하는 매뉴얼에 적용되지 않을 때 발생하는 부작용

소프트웨어 형상관리(1)

- 제품을 개발하거나 유지보수하는 과정에서 변경(change)을 통제하는 절차는 소프트웨어 개발 과정의 산출물들을 관리하고 고품질의 소프트웨어를 얻기 위해 매우 중요

- 소프트웨어 형상관리(Configuration Management)
  - 소프트웨어에 대한 변경을 관리하기 위해 개발된 일련의 활동
    - 소프트웨어 개발 과정에서 나오는 산출물을 체계적으로 관리하고 산출물에 대한 변경이 요구되면 체계적으로 변경하여 효율적인 변경관리가 될 수 있도록 하는 기법
  - 형상관리를 제대로 하지 않으면, 소프트웨어 개발 시 여러 종류의 낭비와 프로젝트 지연요소 발생
  - 소프트웨어 형상의 변경에 대한 철저한 관리가 필요

    ⇒ *소프트웨어 형상관리 (software configuration management)*

소프트웨어 형상관리(2)

- 형상관리의 기본
  - Change Control : 소프트웨어의 개발 중에 행해지는 수정/변경의 관리 및 추적
  - Version Control : Product version의 관리
    - 특정 version – 최신의 버전이 아닐 수도 있음 – 의 생성 및 수정
  - Release Control : Product release의 관리

- 소프트웨어 형상관리 items
  - 형상관리 대상은 소스코드 만이 아님
  - 소프트웨어 개발 단계의 각 과정에서 만들어지는 모든 산출물
    - 프로그램(source code)
    - 프로그램 내부/외부에 포함된 자료 구조
    - 유지보수 단계의 변경사항
    - 시스템 명세서와 설계 명세서
    - 소프트웨어 공학을 위한 표준과 절차
    - 데이터베이스 기술서 등
    - .....

형상관리 도구(1)

- 형상관리 도구
  - 프로그램 모듈의 유지보수와 개발과정의 기록, 모듈의 상호 종속성의 결정과 한 시스템의 다른 버전에서 공통 코드가 일관성이 있는지를 보증하기 위해 도구를 지원
  - 종류
    - 형상관리 데이터베이스(Configuration Management DataBase)
      - 제품 구조에 대한 정보
      - 현재의 개정 번호
      - 현재 상황
      - 각 제품의 버전에 대한 변경 요청 이력 등 제공
    - 버전 제어 라이브러리 시스템(Version Control Library Systems)
      - 원시코드
      - 재배열이 가능한 목적 코드
      - 작업 제어 명령문(JCL)
      - 데이터 파일
      - 보조 문서 등

형상관리 도구(2)

- 형상 관리 예
  - SCCS (Source Code Control System)
    - UNIX 환경 하에서 운용
    - 시스템 모듈에 대한 변경을 저장/기록하기 위한 시스템
    - 모듈이 변경될 때 마다 델타(delta)라는 곳에 기록되고 저장
  - MAKE
    - 한 시스템의 소스 코드와 목적 코드 버전 사이의 일치를 유지하는 보조 시스템
    - 파일들 사이의 종속성을 명시하기 위한 메커니즘을 제공
    - 시스템 소스 코드의 일부에 변경이 이루어질 때 시스템의 목적 코드가 재생성되도록 할 수 있음

소프트웨어 유지보수와 형상관리

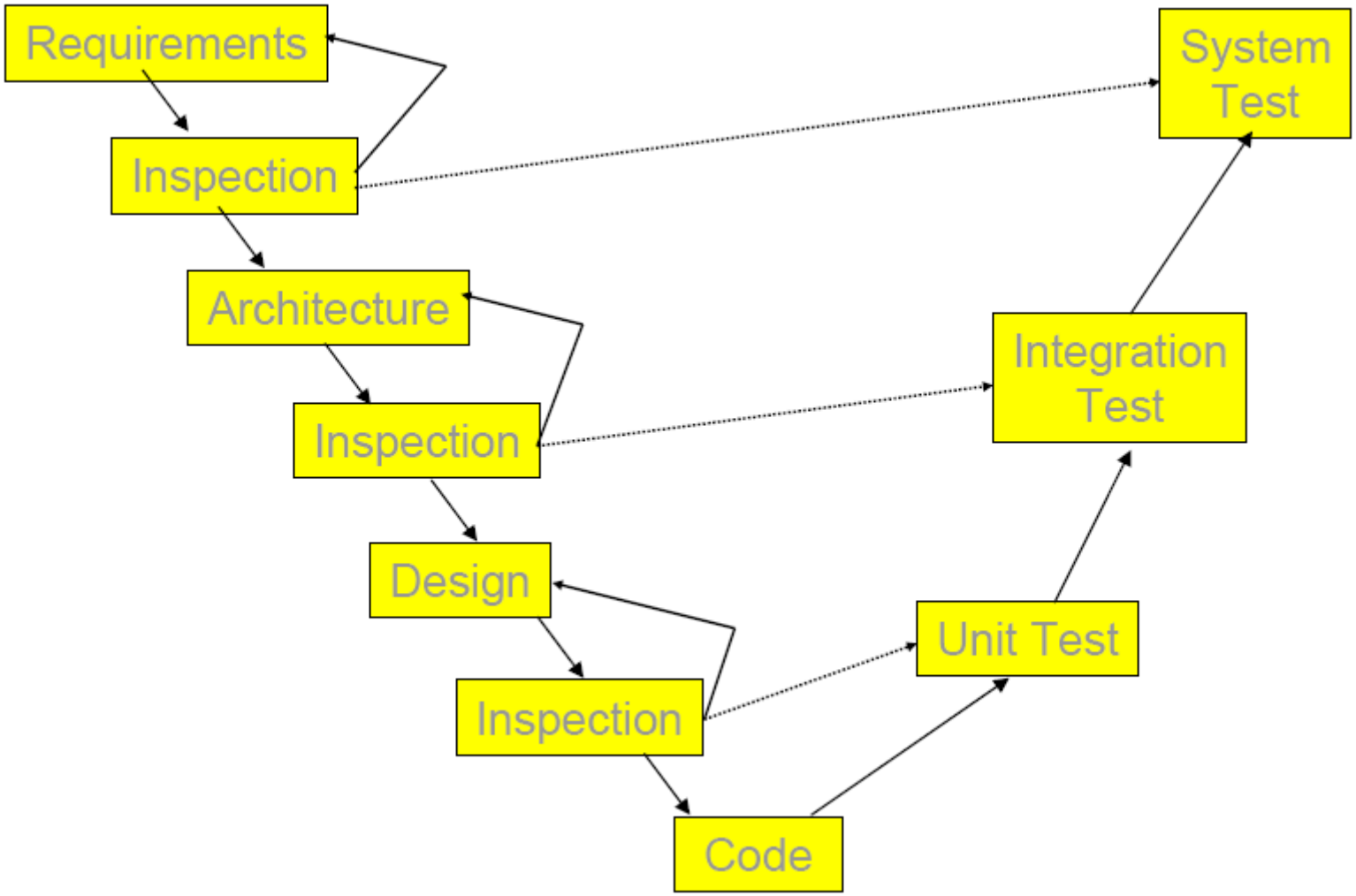
- 소프트웨어 유지보수와 형상관리
  - 소프트웨어의 유지보수를 위해서는, 특정 소프트웨어의 특정 버전을 “재생”해야 함
  - 체계적인 형상관리 도구의 사용 없이는 많은 노력이 필요한 “재미없는” 단순작업
  - Build/release control 또한 형상관리 도구를 이용하여 수행되어야 함

Inspection (1)

- 인스펙션(Inspection)
  - 오류(error), 개발 표준(development standard) 또는 다른 문제점들을 발견하기 위해 개발 산출물(products)을 시각적으로 검사하는 정적 분석 기법(IEEE, 1991)
  - 팀 단위로 수행되는 체계적인 코드 읽기
  - 산업체에서 효과가 입증되고 널리 사용되는 기법
  - 소프트웨어 개발의 전 단계에서 적용 가능한 기법

Inspection (2)

- V-모델



Inspection (3)

- 인스펙션과 개발 단계