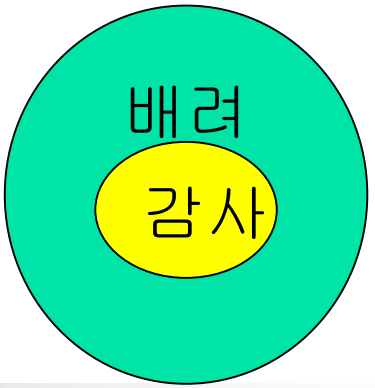
Chapter 31. 정리[프로젝트 관리]

다재 다능한 멀티플레이어

끊임없는 변화,
미래에 대한 열정



제36장.시스템 유지보수

June. 2017
Young-gon, Kim
ykkim@kpu.ac.kr
Department of Computer Engineering
Korea Polytechnic University

Topics covered

- 변화하는 시스템
- 유지보수의 본성
- 유지보수에서 발생하는 문제
- 유지보수 측정
- 유지보수 기법
- 유지보수 도구
- 소프트웨어 재성

1.변화하는 시스템

◆ 유지보수 (maintenance)

- 소프트웨어공학의 목적
  - 정확한 문제정의 및 문제해결을 위한 시스템을 설계 하고
  - 정확하고 효율적인 프로그램 구현
  - 결함을 해결하기 위해 시스템을 테스트하는 기법 을 개발하는 것
- 하드웨어 유지보수
  - 마모된 부품을 교체하거나 시스템의 수명을 연장하기 위한 기술 이용
- 소프트웨어 유지보수
  - 하드웨어와 다르며 하드웨어 유지보수와 같은 방법만으로 소프트웨어 공학을 성공적으로 수행할 수는 없음
- 개발이 완료후 사용자에게 인도되어 사용중에 있는 소프트웨어를 환경적응, 오류수정,성능향상 등을 위하여 계속하여 수정,보완하고 이러한 작업을 통하여 소프트웨어 수명을 연장시켜주는 작업
  - 요구사항이 실세계에 의존적인 시스템일수록 변경될 가능성이 더욱 높음.

1.변화하는 시스템

◆ 시스템 라이프사이클 동안의 변경

- **소프트웨어공학 원칙**
  - ➢ 시스템 개발이 유지보수 되는 동안에도 **변경이 용이**해야 함

    예) 설계와 코드 컴포넌트 그리고 교차 참조하는 컴포넌트가 **모듈화** 되어 있다면 , 요구사항의 변경에 따라 **영향을 받는 컴포넌트**와 다시 수행되어야 하는 **테스트를 쉽게 추적**할 수 있음
  - ➢ 양질의 설계와 올바른 코드 뿐만 아니라

    빠르고 용이하게 변경할 수 있는 기능을 지원하는 것임 .

- 오류가 발생하였을 경우
  - ➢ **원인이 되는 컴포넌트** 를 찾아야 함
  - ➢ 코드 뿐만 아니라 **모든 단계에서도 수정** 할 수 있음 .

1.변화하는 시스템

◆ 소프트웨어 개발하는 동안 변경 사례

초기변경의 결과에 따른 동작	변경되는 가공물
요구사항분석	요구사항명세서
시스템설계	아키텍쳐/기술적인 설계명세서
프로그램설계	프로그램 설계명세서
프로그램구현	프로그램 코드/문서화
통합테스팅	테스트 계획/스크립트
시스템테스팅	테스트 계획/스크립트
시스템인도	사용자/운영자 문서화 시스템/프로그래머 안내서 훈련을위한 지원/강습

1.변화하는 시스템

- 개발 시간 대 유지보수 시간
  - 전통적인 개발 프로젝트
    - 2~3 년 정도 소요
    - 8~12 년의 유지보수 시간
  - 80-20 규칙
    - 노력 : 20%(개발), 80%(유지보수) 사용

- 시스템 **진화** 대 시스템 **쇠퇴**
  - **시스템이 중요** 하고 **지속적으로 변경을 요구**할 경우
    - 레거시 시스템을 제거하고 , 대체할 **새로운 시스템**을 만들어야 하는지 여부를 결정해야 함
  - 라이프사이클 비용 (life-cycle cost)
    - 시스템 생성에서 폐기에 이르기까지 **시스템 개발과 유지보수**와 관련되어 있는 전체 비용을 의미 .

2.유지보수의 본성

◆ 유지보수 활동과 역할

- 유지보수 **활동**
  - 개발단계의 **요구사항 분석**, **시스템 평가** 및 **프로그램 설계**, **코드 작성** 및 **검토**, 변경사항 **테스트**, 문서화의 **업데이트** 등과 유사함 .

- **유지보수**는 시스템 평가에서 4 가지의 주요한 측면에 초점
  - 시스템의 **일상적 기능에 대한 제어 유지보수**
  - **시스템 수정에 대한 제어 유지보수**
  - 기존의 **수용 가능한 기능들을 완전하게 하는 유지보수**
  - 시스템 성능이 **수용 불가능한 수준이 되는 퇴화를 예방하는 유지보수**.

2.유지보수의 본성

◆ 유지보수 종류

1) 수정 유지보수 (Corrective Maintenance)

- 시스템의 일상적인 기능을 제어하기 위해 유지보수팀이 **결함에 따른 문제**에 역점을 두고 유지보수 하는 것을 의미함
- 실패가 발생하면 **발견된 실패**는 유지보수 팀의 관심을 끌게 됨
- 유지보수팀은 사용자와의 상호작용 없이 **적절하게 작동** 하도록 하기위해 설계 , 코드 , 테스트 를 다시 수행함.

2) 적응 유지보수 (Adaptive Maintenance)

- 적응적 변경은 결함을 수정하지 않지만 시스템의 **진화에 따라 적응**하도록 하는 여분의 파라미터를 추가함
- 적응 유지보수는 **하드웨어나 환경에 대해서도 변경**을 수행할 수 있음.

2.유지보수의 본성

◆ 유지보수 종류

3) 기능 향상 유지보수 (Perfective Maintenance)

- 결함으로 인해 제안된 변경 뿐만 아니라 **시스템의 일부 측면을 향상** 시키기 위한 변경을 포함하고 있음
- 완벽한 유지보수의 예
  - 문서화가 **항목을 분류**하기 위해 변경됨
  - 테스트 슈트가 테스트 **적용범위를 향상** 시키기 위해 변경되며 **가독성을 강화**하기 위한 코드 및 설계의 수정 .

4) 예방 유지보수 (Preventive Maintenance)

- **결함을 예방**하기 위한 시스템의 일부 측면들의 변경을 포함하고 있음
- 보통 프로그래머나 코드 분석가가 아직 실패를 유발하지 않으며 손상이 발생하기 전 **결함을 정정하는 활동을 통해 실제적이거나 잠재적인 결함**을 찾았을 때 가능함 .

2.유지보수의 본성

- 유지보수를 하는 **사람**
  - 시스템을 **개발하는 팀**이 항상 동작중인 시스템을 유지보수하는 것은 아님
    - 시스템이 올바르게 동작하는지 확인하기 위해 **별도의 유지보수 팀**을 채용하기도 함
    - 때로는 **별도의 분석가 , 프로그래머 , 설계자 그룹**이 유지보수 팀으로 구성됨.

2.유지보수의 본성

◆ 팀 책임성

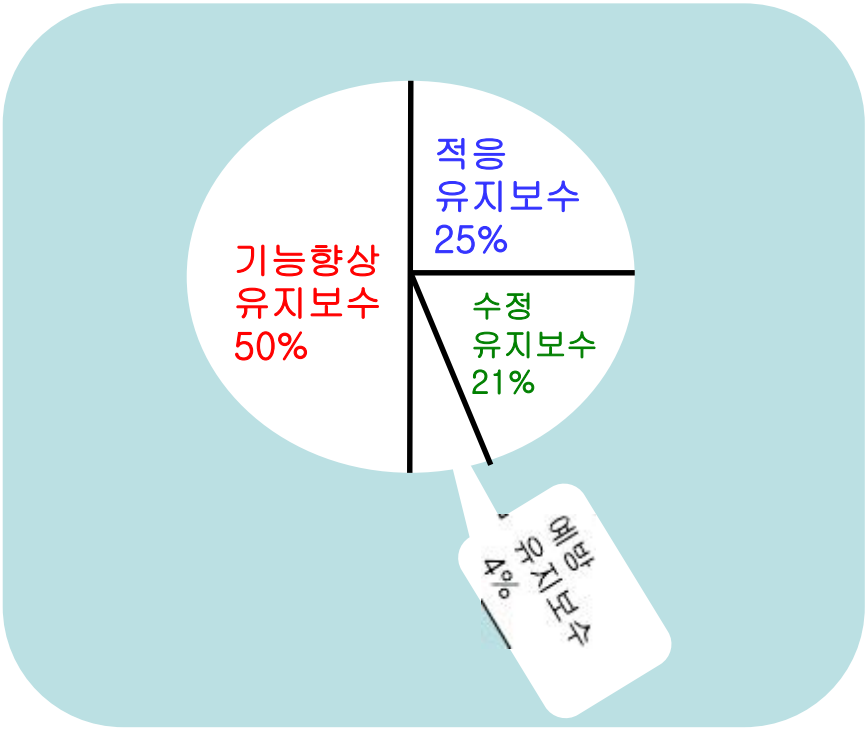
- **사용자 , 운영자 또는 고객 대표자**
  - **비평 또는 문제**를 가지고 **유지보수 팀**과 접촉함.

- 팀에서 수행하는 **활동**
  - **시스템 이해 , 시스템 문서화**에서 정보의 위치파악
  - **시스템 문서화를 최신의 것**으로 유지
  - 새로운 또는 변경된 요구사항에 적응시키기 위한 **기존의 기능 확장**
  - 시스템에 **새로운 기능 추가 , 시스템 오작동/문제를 야기하는 자원** 발견
  - **결함위치 파악 및 수정** , 시스템이 **작동하는 방법에 대한 질문에 응답**
  - **설계와 코드 컴포넌트의 재구성/재작성**
  - **불필요한 설계나 코드 컴포넌트의 삭제**
  - 유지보수 팀이 수행한 시스템 **변경사항 관리**.

2.유지보수의 본성

◆ 유지보수 시간의 이용

- 대부분의 노력은 **기능 향상** 유지보수와 **적응** 유지보수에 집중됨 .



[유지보수 노력의 분포]

3.유지보수에서 발생하는 문제

◆ 1) 제한된 이해

- 소프트웨어 및 하드웨어 요구와 사용자 요구간의 균형 외에도 유지보수 팀은 **인간 이해의 한계**를 다루어야 함
  - 소프트웨어 유지보수 노력의 **47% 가 수정될 소프트웨어를 이해**하는데 소요
- 유지보수 프로그래머가 직면하는 **문제의 절반 이상이 사용자의 기술이나 이해 부족**에 따른 것이라는 것을 발견함
  - **명확하고 완벽한 문서화와 교육**의 중요성을 시사
- 유지보수 팀은 또한 **사람을 잘 다루는 기술을 필요**로 함.

3.유지보수에서 발생하는 문제

◆2) 관리 우선순위

- 유지보수 팀은 시스템의 필요에 대한 고객의 관리요구에 중점
- 관리 우선순위는 종종 기술적인 우선순위를 무시함 .

◆3) 사기 (Morale: 의욕)

- 유지보수 중에 발생하는 문제의 11.9% 가 낮은 사기와 생산성 에 기인한다고 지적함
- 사기가 낮은 주요 요인
  - 유지보수 팀을 이류 (second-class)로 분류하는 측면
  - 프로그래머 : 때로 시스템이 계속 동작하도록 유지하는 것보다 시스템 설계 및 개발에 더 많은 기술이 필요하다고 생각함
  - 유지보수 프로그래머 : 개발자들이 결코 예상하지 못하였던 문제를 다루어야 함
- 유지보수 기간동안 발생하는 문제의 80% 는 프로그래머가 동시에 여러 프로젝트에서 작업하기 때문에 문제 해결에 충분한 시간을 투자할 수 없다는데 있음 .

3.유지보수에서 발생하는 문제

◆4) 기술적인 문제

- 구현시에 채택하였던 **특정 패러다임이나 프로세스**에 따라 기술적인 문제가 발생하기도 함 .
- 가공물과 패러다임
  - 객체지향 프로그램을 유지보수하는 것 역시 **복잡한 상속**으로 인해 고도로 상호 연결된 컴포넌트를 포함하기 때문에 많은 문제가 발생 가능
  - **부적절한 설계 명세서**와 **저질 프로그램** 및 **문서화**는 유지보수 노력의 약 **10% 정도**의 비율을 차지함
  - 문제는 **하드웨어 및 소프트웨어 또는 데이터를 신뢰**할 수 없을 때 발생.

◆5) 테스팅의 어려움

- **언제 테스트**를 해야 할지를 결정할 때 문제가 될 수 있음
- 시간 가용성 문제 외에도 테스팅을 위한 **테스트 데이터**가 만족스럽거나 적절하지 못할 수도 있음
- 테스터들이 **설계나 코드 변경의 효과를 예측하고 준비**하는 것이 쉬운 일이 아님.

3.유지보수에서 발생하는 문제

- 시스템을 유지보수하기위해 필요한 노력을 결정하는 요인
  - 애플리케이션 유형
  - 시스템 참신성
  - 교체와 유지보수 스태프의 가용성
  - 시스템 생존 기간
  - 변화하는 환경에 대한 의존도
  - 하드웨어 특성
  - 설계 품질
  - 코드 품질
  - 문서화의 품질
  - 테스팅 품질.

4.유지보수 측정

- 유지보수성을 **측정** 하기 위해 각 문제에 대한 세심한 **기록**이 필요함
  - 유지보수성의 **외적 관점**
    - 문제가 제기된 **시점** , **관리상의 지연**으로 인한 **시간 손실**
    - **문제 분석**에 필요한 **시간** , 이루어져야 할 **변경을 명세**하는데 필요한 시간
    - **변경**에 필요한 시간 , 변경을 **테스트**하는데 필요한 시간
    - 변경을 **문서화** 하는데 필요한 시간 .
  - 유지보수성에 영향을 주는 **내적 요인**
    - **코드가 복잡** 할수록 더 많은 유지보수가 필요함
    - **상호관계**는 측정과 동일하지 않다는 것을 반드시 염두에 두어야 함
      - **구조와 문서화가 빈약**하면 유지보수가 어렵다는 것은 명백함 .

5.유지보수 기법

◆ 변경 제어

- 형상관리 팀은 반드시 시스템의 모든 **컴포넌트와 문서의 상태**를 알아야만 함
- 형상관리는 시스템에 **영향을 미치는 동작**들 사이의 **의사소통을 강조** 해야 함
- 형상관리 팀은 다양한 여러 **개정 내용**이 서로 얼마나 다른지 , **누가 변경** 하였는지, **왜 변경** 하였는지를 알 수 있음.

5.유지보수 기법

◆ 영향분석 (impact analysis)

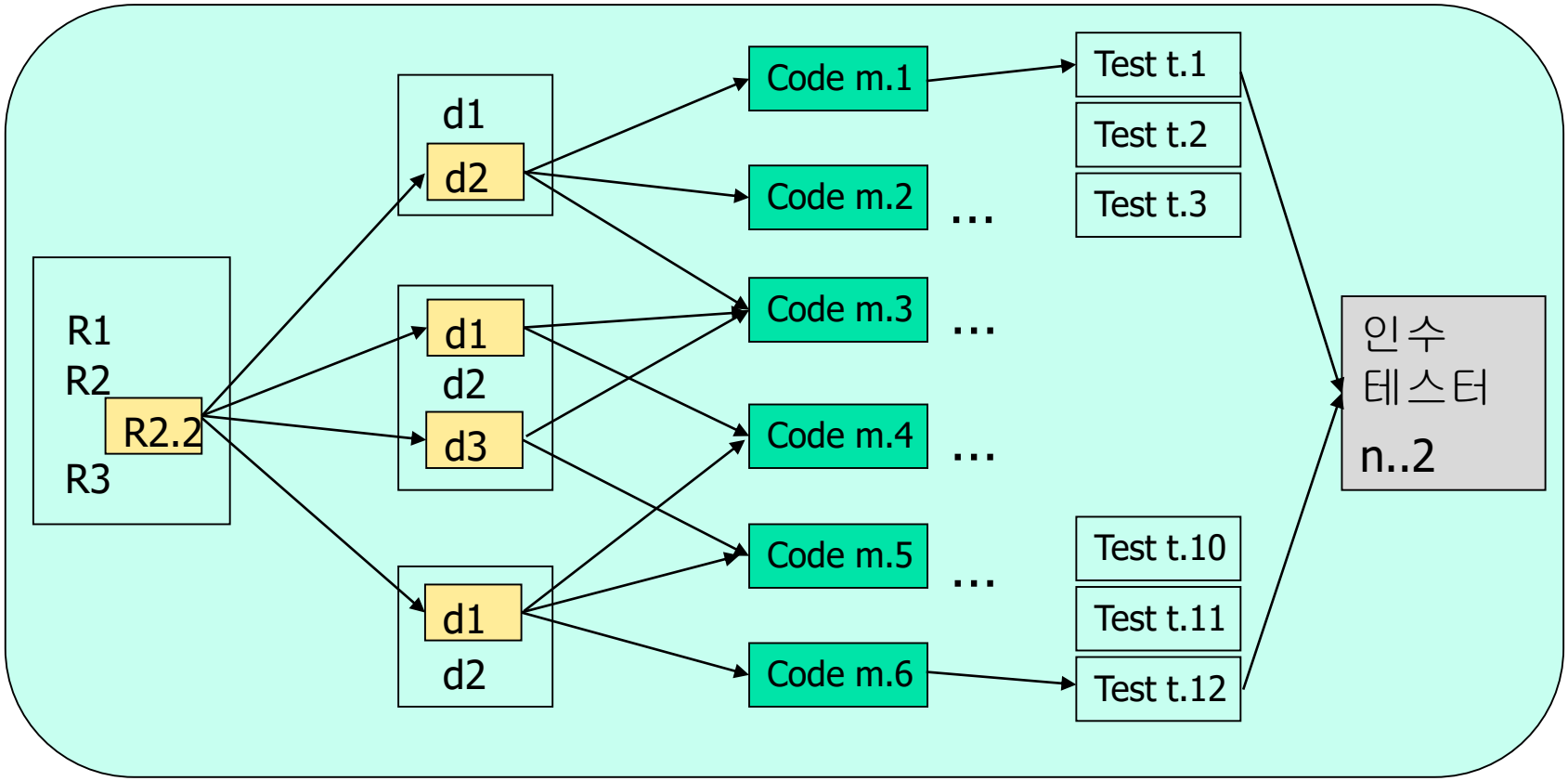
- 양질의 소프트웨어 개발의 원리는 **개발과 유지보수 프로세스** 모두에 적용됨
- 영향분석에서는 **자원에 미치는 영향 , 노력** 그리고 **스케줄**과 같은 변경에 따른 많은 **위험들을 평가**함
- 위험을 결정하고 여러 옵션을 신중히 고려하고자 **제안된 변경이 갖는 영향**을 측정하는 방법
- **작업 산출물 (workproduct)** : 변경이 중요시되는 개발 가공물임
  - **요구사항 및 설계 , 코드 컴포넌트 , 테스트 사례 , 문서화** .

5.유지보수 기법

- 수직적 추적가능성 (vertical traceability)
  - 일부 **워크프로덕트간의 관계**를 표현함
    - 예) 요구사항의 수직적 추적가능성은 **시스템 요구사항간의 상호의존관계**를 기술.
- 수평적 추적가능성 (horizontal traceability)
  - 워크프로덕트 집합과 교차하는 **컴포넌트의 관계**를 표현함
    - 예) 각 설계 컴포넌트는 설계를 구현하는 코드 컴포넌트를 추적함 .
- 영향분석동안 평가된 완벽한 관계의 집합을 이해하기 위해 두 가지 형태의 추적가능성이 모두 필요함
  - **방향성 그래프**를 이용하여 **수평과 수직적 추적가능성을 모두 기술** 할 수 있음
    - **방향성 그래프** : 단순한 **객체 집합인 노드**와 이 노드를 연결하는 **간선**으로 구성.

5.유지보수 기법

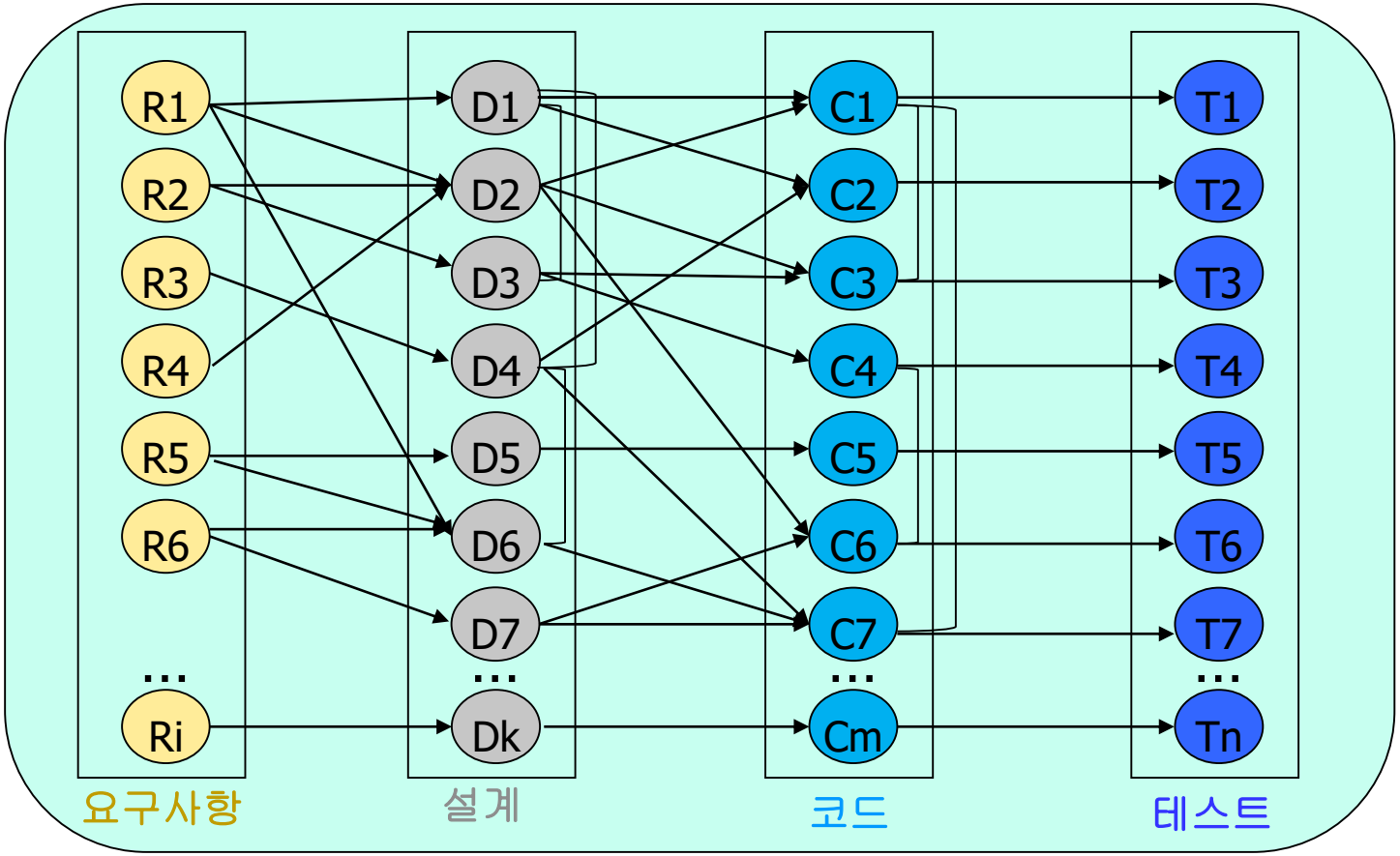
◆ 작업 산출물 사이의 도식적인 관계와 추적가능성 링크



[소프트웨어 작업 산출물에서 수직/평 적 추적가능성]

5.유지보수 기법

◆ 전체 추적가능성 그래프



[유지보수의 기초가 되는 그래프]

6.유지보수 도구

◆ 자동화된 유지보수 도구

- **텍스트 편집기**
- **파일 비교기**
  - 두 파일을 비교 하고 그들간의 차이를 보고
- **컴파일러와 링커**
  - 단순한 유지보수 와 형상관리 특성을 포함
- **디버깅 도구**
  - 프로그램의 논리를 단계별로 추적할 수 있도록 하는데 목적
- **교차 참조 발생기**
  - 제안된 요구사항에 대한 **변경 수행 시** 다른 **요구사항 , 설계 , 코드 컴포넌트**가 **영향**을 받을 것인지 확인
- **정적 코드 분석기**
  - **중첩의 깊이 , 확장되는 경로의 수 , 코드의 라인수 , 실행되지 않는 문장** 과 같은 **코드의 구조적인 속성 정보**를 계산
- **형상 관리 저장소**
  - 각 **문제와 문제를 보고한 조직과 수정한 조직에대한 정보**를 포함하는 **문제 보고서**를 저장하며, 사용자가 현재 사용중인 시스템에서 **보고된 문제에 대한 레벨**을 유지

7.소프트웨어 재성

◆ 소프트웨어 **재성** (rejuvenation)

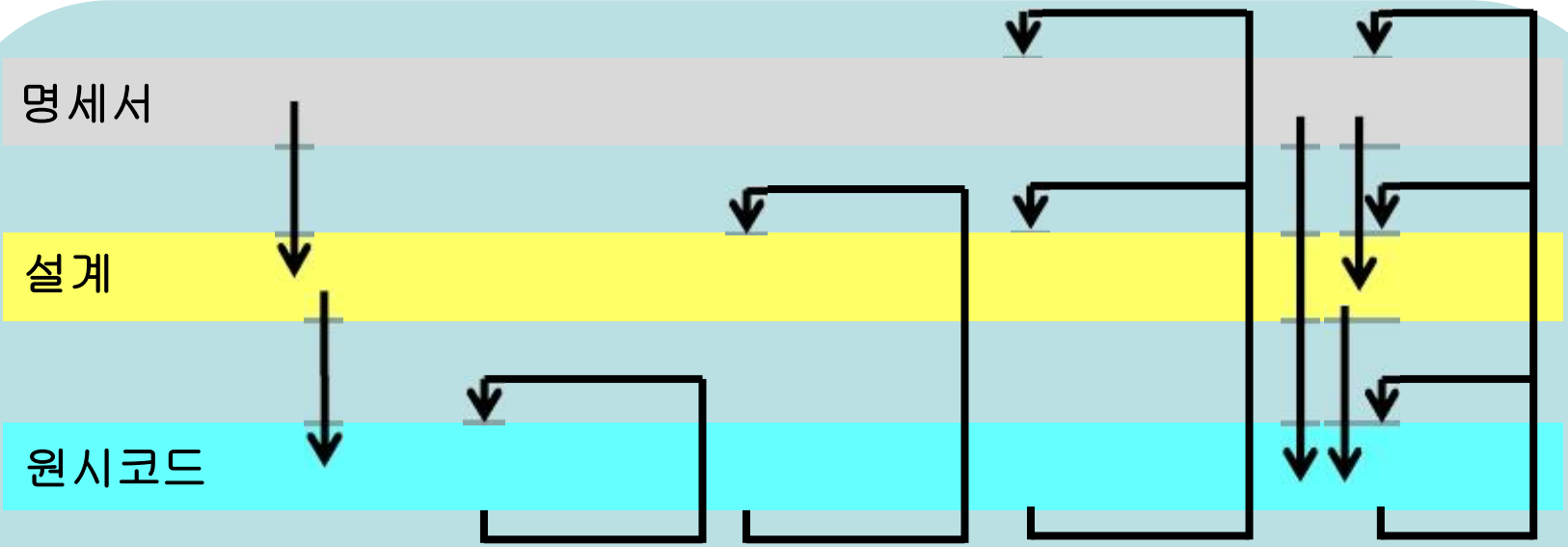
- 기존의 시스템의 전체 **품질을 증가** 시키도록 하는 유지보수 노력
- 추가적인 **정보를 유도** 하거나 보다 **이해하기 쉬운 방법으로 개량** 도록 시스템의 **작업산출물**를 **재조사** 하여야함.

◆ 고려할 측면

- **재문서화** (redocumentation)
  - 유지보수자들의 코드에 대한 **이해와 참조를 보조하기 위해 부가적인 정보를 생산** 하는 **소스 코드의 정적 분석**을 수행함
- **재구성** (restructuring)
  - 구조화되지 않은 코드를 **구조적으로 변형** 함
- **역공학** (reverse engineering)
  - 앞서 만들어진 **소스 코드로부터 산출물**에 이르기까지 재조사하며 **코드로부터 설계 및 명세서 정보를 재생성**
- **재공학** (reengineering)
  - 역공학의 **더 넓은** 개념 .

7.소프트웨어 재성

◆ 재성의 4 가지 유형간의 관련성



순공학
- 프로세스를 통한 순서대로 진행

재구성
- 코드
- 내부적 표현
- 반복적인 구조 단순화 및 쓸모없는 코드 제거
- 코드 재생성

재문서화
- 코드
- 구조에 관한 정적분석 보고서
- 복잡도 , 볼륨 , 데이터 등
- 소프트웨어 방법론에 기초하지 않음

역공학
- 코드
- 선택된 소프트웨어 방법론을 기초로 한 설계와 명세서 생산
- 표현관리

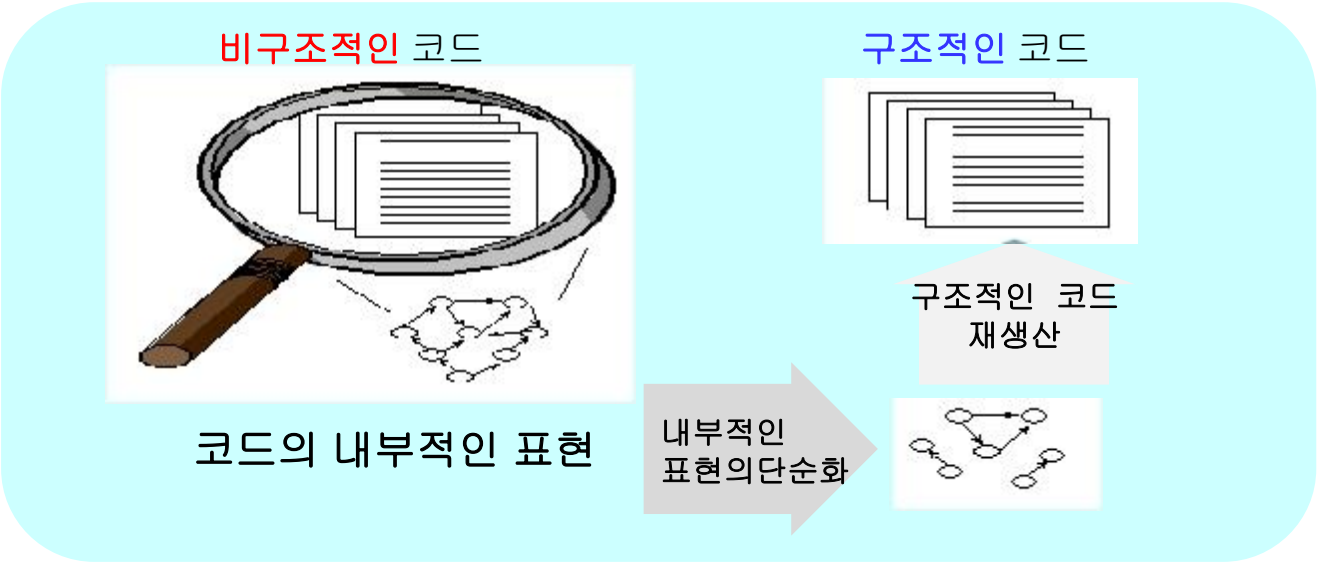
재공학
- 코드
- 역공학자 : 코드
- 순공학자 : 표현변경 및 완료
- 코드 재생성

[소프트웨어 재성의 분류법]

7.소프트웨어 재성

◆ 1. 재구성

- **이해와 변경이 용이**하도록 소프트웨어를 재구성함
- 재구성과 관련된 세가지 주요한 활동
  - 정적 분석은 **코드를 시맨틱 네트워크나 방향성 그래프**로서 표현하는데 이용할 수 있는 정보를 제공함
  - **변형 기술**을 기초로 하는 연속된 **단순화**를 통해 정제됨
  - 정제된 표현은 **구조화되고 동등한 코드**를 생성하기 위해 번역, 이용됨.



7.소프트웨어 재성

◆ 2. 재문서화

- 시스템 문서화를 생산하는 소스코드에 대한 정적인 분석을 포함
- 출력에 포함되는 내용
  - 컴포넌트 호출 관계
  - 데이터 인터페이스 테이블
  - 데이터 사전 정보
  - 데이터 흐름 테이블 또는 다이어그램
  - 의사코드
  - 테스트 경로
  - 컴포넌트와 다양한 전후 참조.

7.소프트웨어 재성

◆2. 재문서화 프로세스

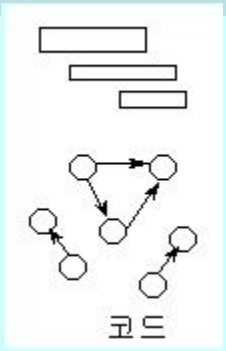
정적분석을 이용한 평가

컴포넌트와 변수 테이블

상호 참조

데이터 인터페이스 테이블

의사코드



데이터및 제어흐름

컴포넌트 호출 계층구조

프로그램 트리

복잡도와 데이터 크기

데이터 사용

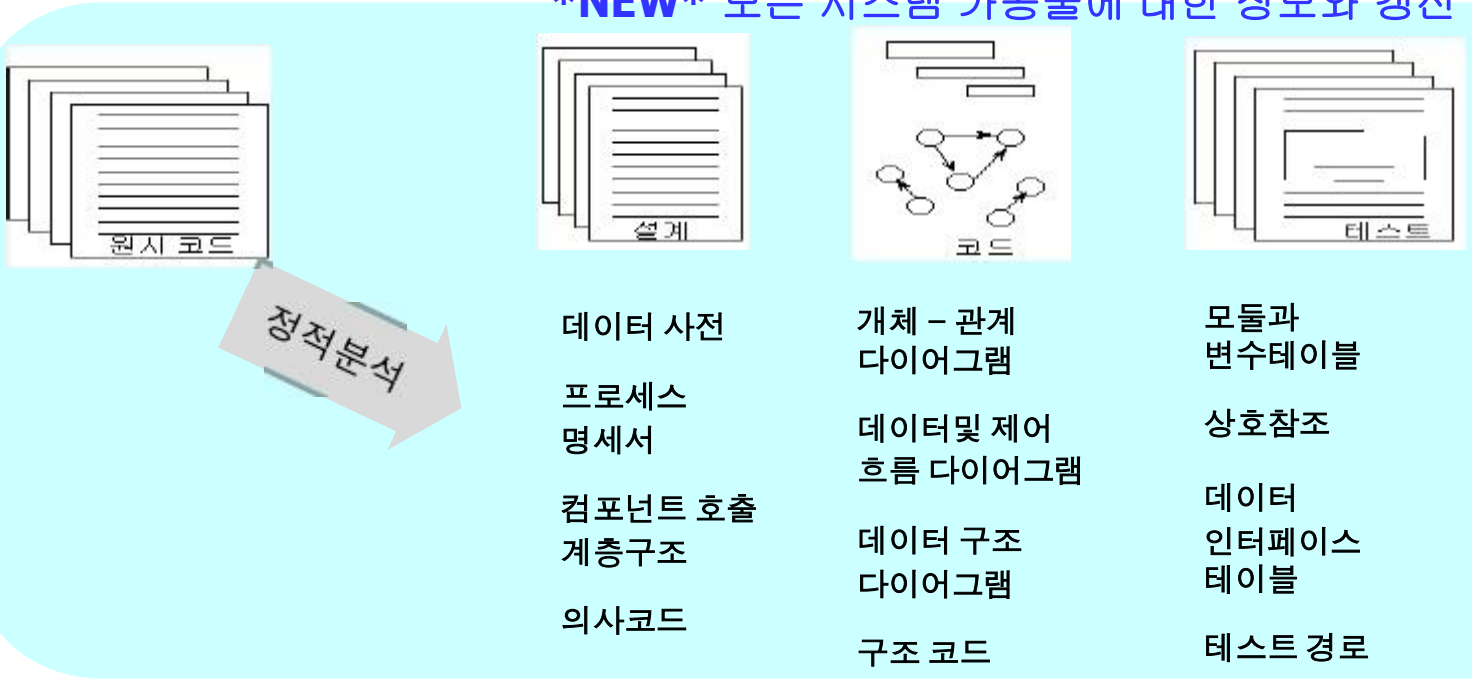
파급효과

테스트 경로

7.소프트웨어 재성

◆3. 역공학

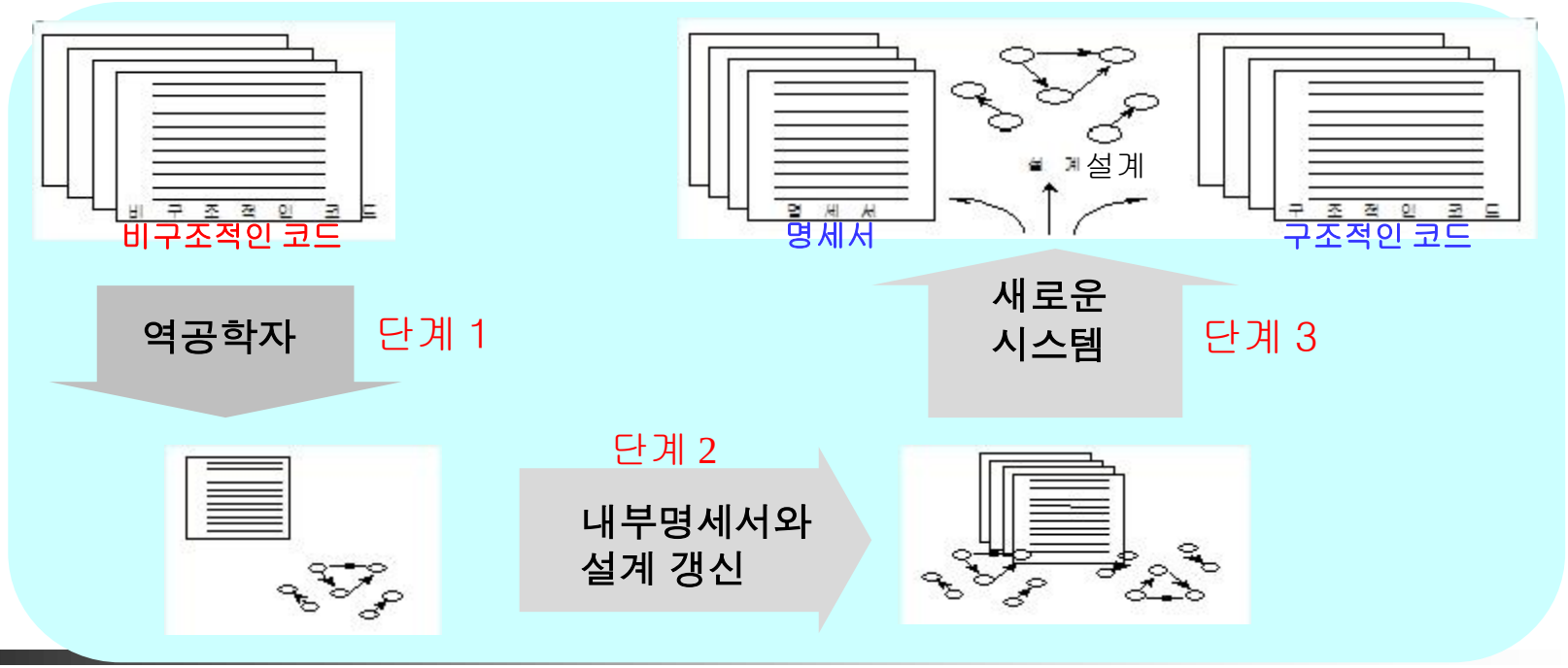
- 소스코드로부터 소프트웨어 시스템에 관하여 **명세서와 설계정보**를 제공함 (정보를 추출함)
- 역공학의 핵심은 상세한 **소스코드** 구현으로부터 **명세서를 추상화**
- **역공학 프로세스**



7.소프트웨어 재성

◆4. 재공학

- 재공학은 **역공학의 확장**
- 전체 시스템의 기능을 변경하지 않고 **새로운 소프트웨어 소스코드**를 생산함
- 완전히 자동화된 재공학은 가까운 장래에는 가능할 것 같지 않으므로 **프로세스는 변환과 인간과의 상호작용을 결합**해야만 함
- 재공학 프로세스



정리 및 Homework

1) 소프트웨어 개발하는 동안 변경 사례
2) 유지보수 종류와 특징
3) 유지보수에서 발생하는 문제
4) 시스템을 유지보수하기 위해 필요한 노력을 결정 요인
5) 소프트웨어 재성의 종류와 특징 그리고 프로세스

Project

1장. 프로젝트 개요
- 1.1 프로젝트 제목
- 1.2 선정 이유
- 1.3 팀 운영 방법

2장 시스템 정의
- 2.1 시스템 간략한 설명
- 2.2 유사 사례 간략한 설명

3장 프로세스 모델
- 3.1 규범적인 프로세스 모델 선정 및 이유
- 3.2 특수한 프로세스 모델 선정 및 이유

4장. 실무 가이드 원칙
- 4.1 각 프레임워크 원칙에서 중요한 3 개 정의
- 4.2 프로젝트 계획 보고서

5장. 요구사항 획득
- 5.1 기능 요구사항과 비기능 요구사항 정의
- 5.2 표준 양식을 사용한 시스템 요구사항 명세 3개 작성
- 5.3 정형적인 형식에 따른 유스케이스 작성

6장. 시스템 설계
- 6.1 설계 개념의 중요한 개념을 적용
- 6.2 설계 모델에 따른 요소별 설계

7장. 아키텍처 개념
- 7.1 아키텍처 스타일 선정및 이유
- 7.2 아키텍처 설계 프로세스 정의 및 설계

8장. 품질
- 8.1 시스템 품질 속성 정의
- 8.2 비즈니스 품질 속성 정의
- 8.3 아키텍처 품질 속성 정의
- 8.4 소프트웨어 품질 목표, 속성과 척도
- 8.4 소프트웨어 통계적 방법 이용 사례 선정

9장 . 테스팅 전략
- 9.1 테스팅 전략적 이슈 순위정의 (전략 성공)
- 9.2 통합테스팅 방법 및 순서
- 9.3 시스템 테스팅 중요이슈 정의
- 9.4 소프트웨어 결함 유형 정의

10장. 프로젝트 관리
- 10.1 우리 팀의 융합적인 요소[현재시점]
- 10.2 우리 팀의 독소 요소[시작 시점]
- 10.3 프로젝트 스케줄링의 활동 네트워크 작성
  - 1) 타스크 일정 및 의존성
  - 2) 활동 네트워크

11장 시스템 유지보수
- 11.1 자동화된 유지보수 도구

(개인별 3개, 팀별 5개 선정)

12장 Review

(교훈:개인별 A4 1Page 정도)