
Chap 11 유지보수

목 차

11.1 유지보수의 소개

11.2 유지보수 작업 과정

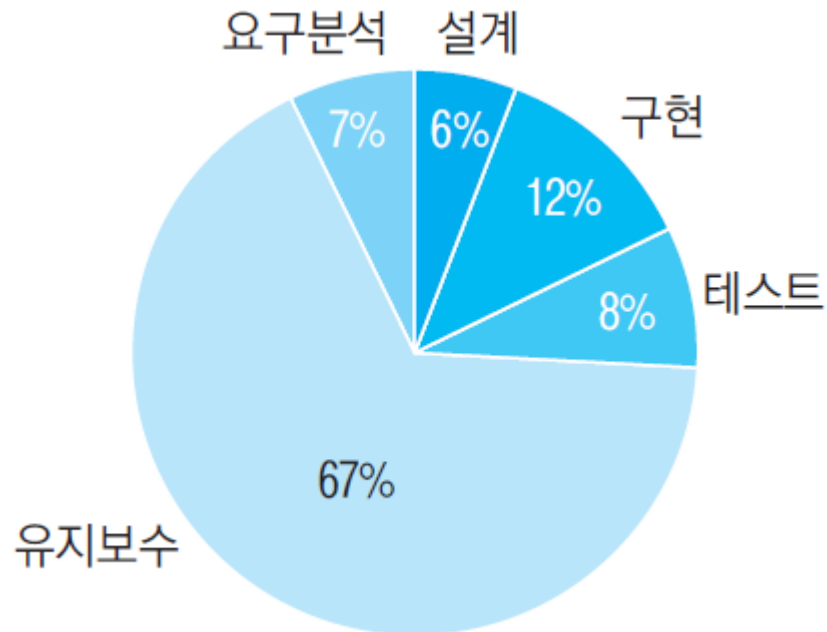
11.3 형상 관리

11.4 역공학

11.5 리엔지니어링

유지 보수

- 개발 후에 이루어지는 소프트웨어의 변경 작업
- 소프트웨어가 유용하게 활용되는 기간
- 소프트웨어는 환경과 비즈니스 요구에 따라 진화함
- 유지보수에 드는 노력



11.1 유지보수 소개

- 레거시(legacy) 시스템
 - 대체하려면 비용이 많이 든다.
 - 지식, 경험, 지능이 녹아 있음
- 여러 가지 이유로 수정 후 배포하는 작업
 - 버그 제거
 - 운영 환경의 변화
 - 정부 정책, 규제의 변화
 - 비즈니스 절차의 변화
 - 미래 문제를 배제하기 위하여 변경

유지보수 유형

- 수정형 유지보수(corrective maintenance)
 - 발견된 결함을 고치기 위하여 소프트웨어 제품을 수정
- 적응형 유지보수(adaptive maintenance)
 - 변경된 환경에서도 계속 사용할 수 있도록 소프트웨어를 이식하거나 변경
- 완전형 유지보수(perfective maintenance)
 - 성능이나 유지보수성을 개선하기 위하여 실시하는 유형
- 예방형 유지보수(preventive maintenance)
 - 오류 발생을 방지하기 위해 수행하는 유지보수

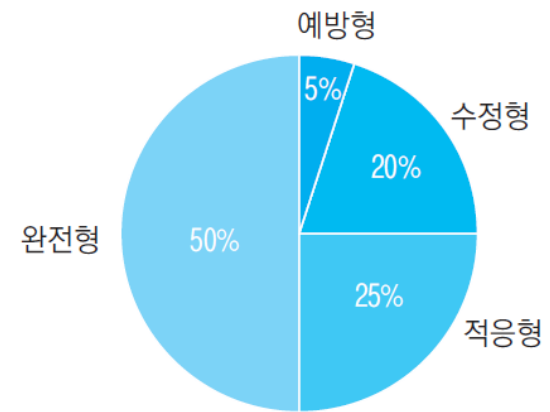


그림 11.2 유지보수 유형별 분포

Lehman의 법칙

- 시스템의 타입
 - E 타입 – 계속 진화하는 타입
 - S 타입 – 완벽히 정의할 수 없는 타입(체스 게임)
- 1. 지속적인 변경의 원칙
- 2. 복잡성 증가의 법칙
- 3. 자기 통제의 법칙
- 4. 안정성 유지의 법칙
- 5. 친근성 유지의 법칙
- 6. 지속적 성장의 법칙
- 7. 품질 저하의 법칙
- 8. 피드백 시스템의 법칙

Lehman의 법칙 더 알아보기

<https://zrr.kr/JjrNoy>

<https://zrr.kr/eS6IAX>

유지 보수의 종류

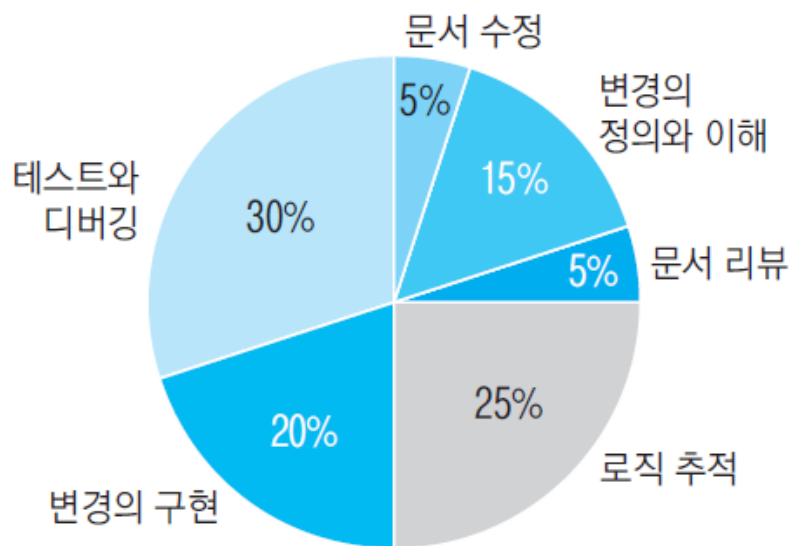
- 교정형 유지보수(corrective maintenance)
 - 발견된 오류의 원인을 찾아 계획적으로 문제해결
- 적응형 유지보수(adaptive maintenance)
 - 새로운 자료나 운영체제, 하드웨어 환경으로 이식
- 완전형 유지보수(perfective maintenance)
 - 성능이나 유지보수성을 개선하기 위한 변경
- 응급형 유지보수(emergency maintenance)
 - 응급처치하기 위한 무계획적 유지보수

10.2 유지 보수 작업 과정

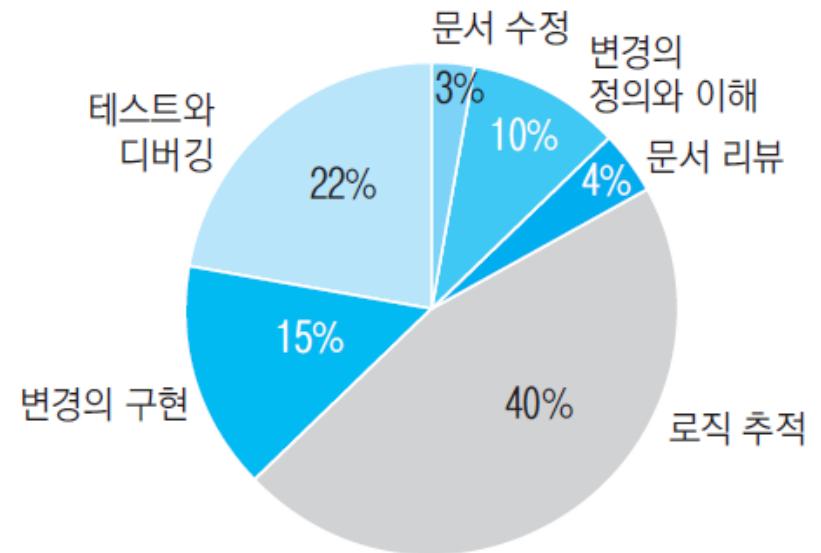
- ① 현재 프로그램의 이해
 - 프로그램 로직을 추적하거나 요구, 설계 등에 대한 이해가 필요
- ② 변경 파악과 분석
 - 필요한 변경을 파악하고 그 영향도와 소요 비용, 변경에 의한 리스크를 분석
- ③ 변경 영향 파악
 - 이해당사자들에게 알리고 피드백을 얻음
- ④ 변경 구현, 테스트, 설치
 - 시스템을 수정하고 확인 후 설치

유지 보수 작업 분포

- 소프트웨어 개발은 코딩 중심의 작업이나 유지보수는 이해 중심의 작업, 통합된 작업



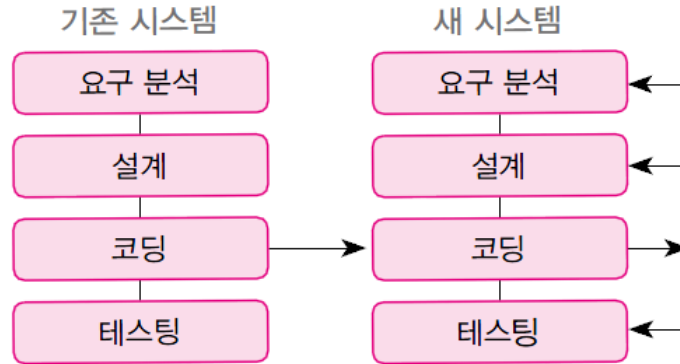
(a) 평균 분포



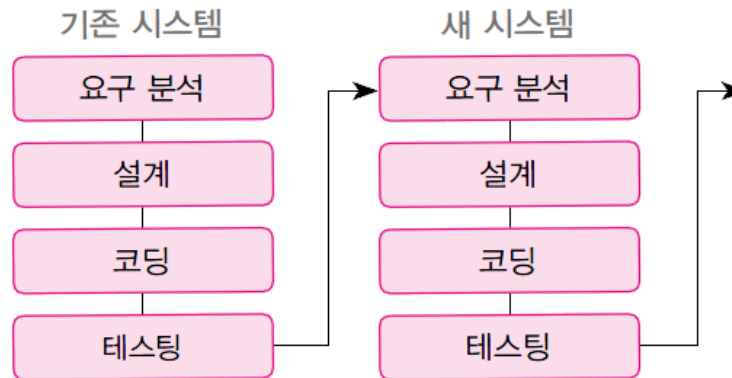
(b) 문서가 빈약한 시스템

유지 보수 프로세스 모델

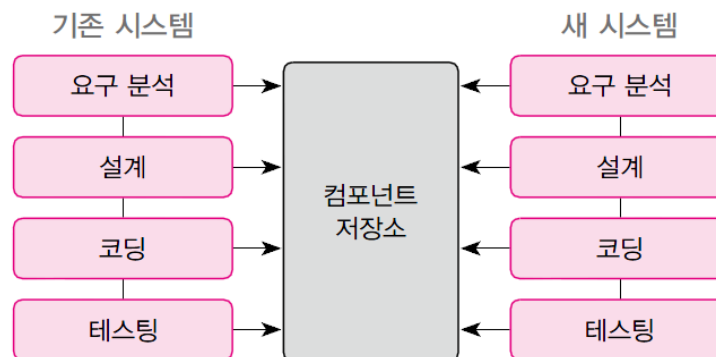
- 즉시 수정 모델



- 반복적 개선 모델



- 재사용 중심 모델

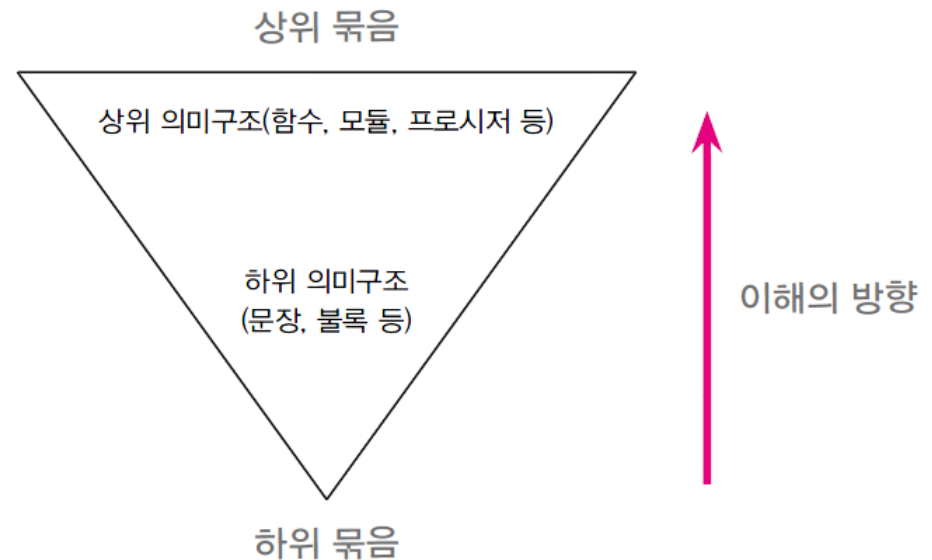


유지보수 프로세스 모델의 비교

반복적 개선	IEEE	ISO
분석	문제의 이해/분류	문제와 변경 분석
요구	분석	변경 구현
설계	설계	
코딩	구현	
테스팅	리그레션/시스템 인수 테스트	유지보수 리뷰/인수
	배포	전환
		소프트웨어 퇴역

프로그램 이해

- 원시코드로부터 설계나 명세를 추출하여 멘탈 모델로 표현하는 작업
- 개발 프로세스와 반대로 추상성을 추구하는 방향
- 상향식 이해 모델
 - 상향식(bottom-up)
 - 묶음화(chunking)



변경 파악과 분석

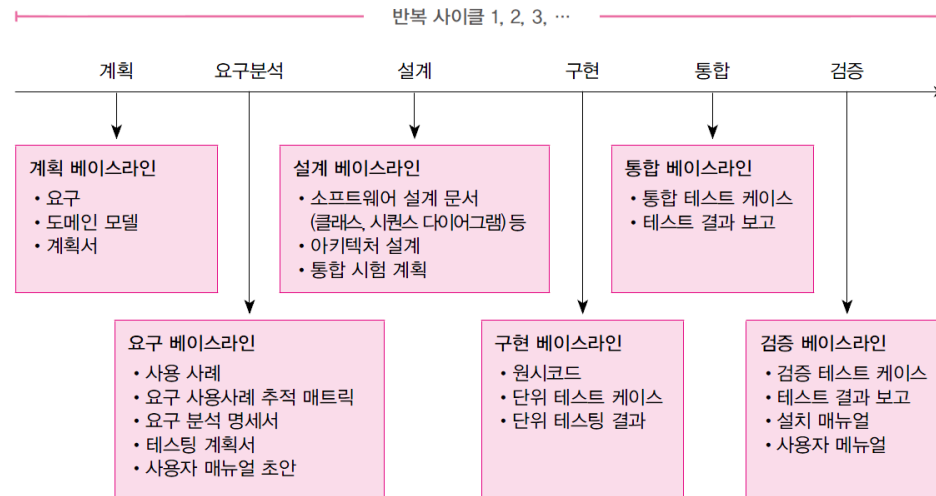
- 변경 요구를 기초로 어떤 부분을 변경할지 찾아냄
- 다른 변경 방법(COTS)도 찾아냄
- 변경 분석
 - 변경 효과 분석
 - 변경을 구현하고 테스트하는 데 드는 비용, 시간의 예측
 - 리스크 파악
- 객체지향 소프트웨어
 - 변경 효과 – 클래스 사이의 의존관계로 파악
 - 클래스 B가 A의 서브클래스이면 B는 A에 의존관계
 - 클래스 B가 A의 집합이면 B는 A에 의존관계
 - 클래스 B가 A를 사용하면 B는 A에 의존관계
 - 클래스 B가 A와 다른 클래스 사이의 연관을 위한 클래스이면 B는 A에 의존관계

11.3 형상관리

- 형상 관리(Configuration Management)
 - 개발 주기 동안 생성된 문서를 관리하고 소프트웨어 시스템과 컴포넌트의 상태를 추적하는 작업
- 문서와 결과물에 대한 변경이 잘 조정되지 않는다면 불일치 발생
- 클래스 변경 후 의존 클래스도 함께 업데이트
- 하드웨어에 적용되었던 전통적 원리를 소프트웨어 개발에 적용

베이스 라인

- 베이스 라인
 - 소프트웨어 형상 항목(configuration item)의 집합
- 목적
 - 프로젝트의 중요한 상태 정의
 - 프로덕트가 특정 상태에 이르렀는지를 나타냄
 - 계속되는 개발, 유지보수 작업의 기준
 - 형상 항목에 대한 변경을 제어하는 메커니즘

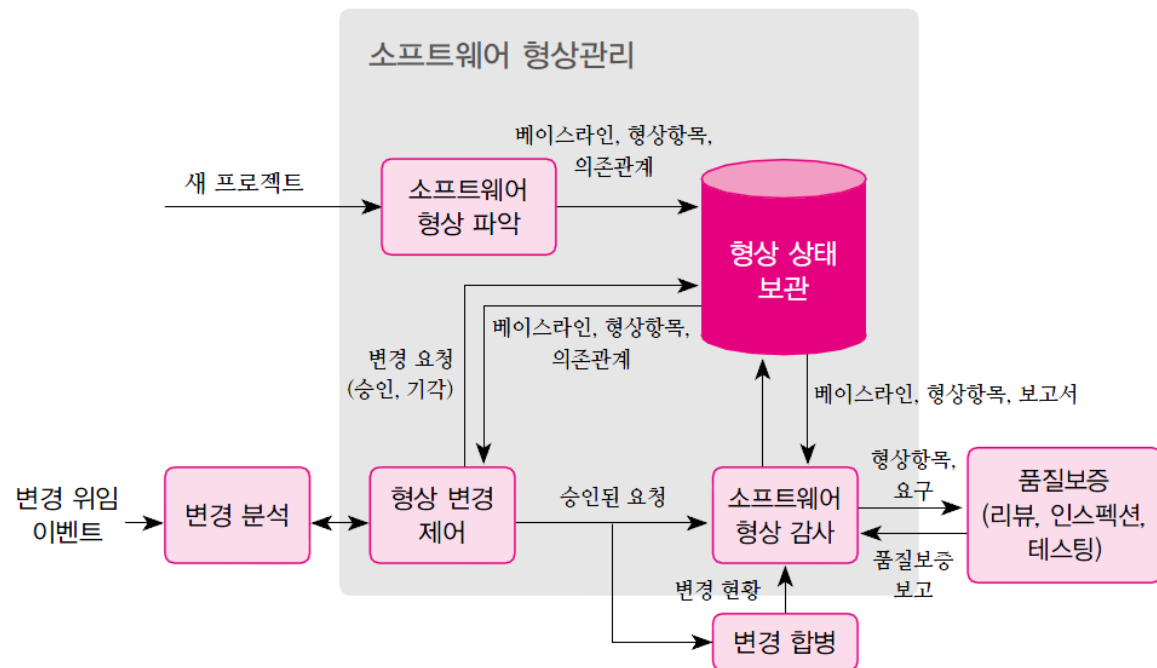


형상관리의 필요성

- 소수의 개발자가 한 장소에서 일한다면 형상관리는 불필요
- 시스템을 개발하는 많은 팀과 개발자들이 협력하고 동기화 할 필요
- 여러 버전을 유지하여야 할 경우
 - 다양한 고객을 만족시키기 위한 제품을 유지하기 위해

형상관리 절차

- 소프트웨어 형상 파악
- 형상 변경 제어
- 소프트웨어 형상 감사
- 소프트웨어 형상 상태 보관



형상 파악

- 고유 식별자 – 고유 번호 예, LIS-Incl-DM
- 이름
- 문서 종류 – 요구 분석서, 설계 문서, 원시코드, 테스트 케이스
- 문서 파일 – 파일 이름과 경로
- 저자
- 생성 날짜, 목표 완성일
- 버전 번호
- 업데이트 이력
- 설명
- SQA 담당자 – 품질 보증 책임자
- SCM 담당자 – 항목 체크한 책임자

형상 변경 제어

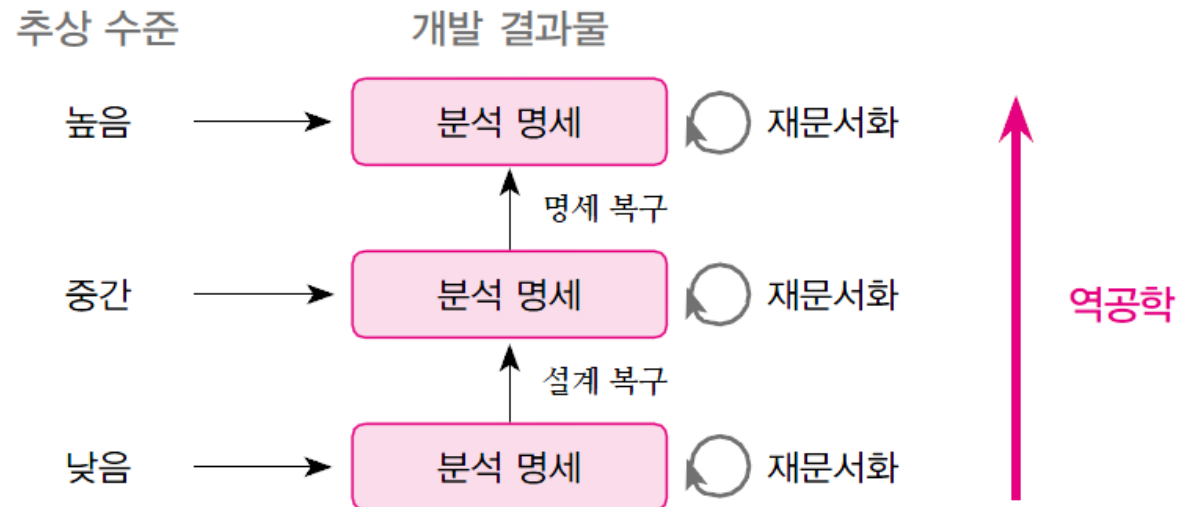
- 변경의 이유를 파악
 - 소프트웨어의 결함
 - 하드웨어 변경
 - 운영 요구의 변경
 - 고객이나 사용자로부터 개선 요구
 - 예산, 프로젝트 일정, 기간의 변경
- 변경 분석
- 변경 제안 준비
 - 변경의 설명, 조직 및 개발자 파악, 변경의 이유, 영향 받는 항목, 소요 노력, 프로젝트 일정에 대한 영향
- 변경 제안의 평가
- 변경을 추가

형상 감사

- 베이스라인을 구축하기 위한 메커니즘 정의
 - 향후 구축될 베이스라인과 승인된 베이스라인
- 형상 항목 검토
 - 업데이트되어도 차이가 없음을 보장하여야
- 형상 항목 확인
 - 올바른 문제를 해결하였는지 입증하기 위하여 정확성을 체크

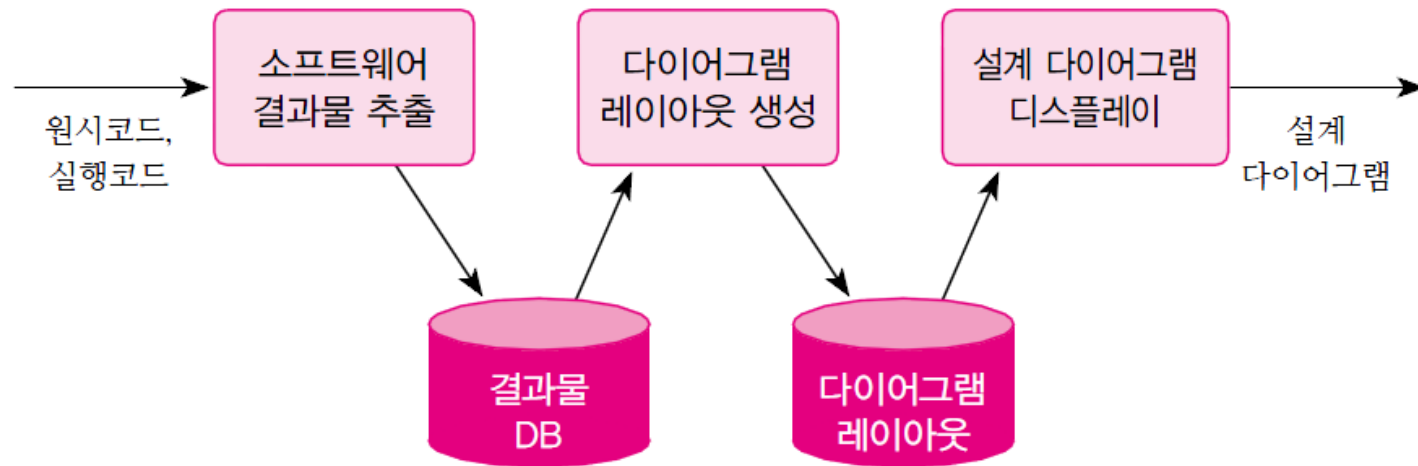
10.4 역공학

- 역공학의 정의
 - 대상 시스템을 분석하여 시스템의 컴포넌트와 관계를 찾아내어 같은 수준의 다른 표현이나 더 높은 수준의 표현으로 만드는 작업
- 프로그램의 추상 수준을 점증적으로 복구해 나가는 과정



역공학 작업순서

- 원시코드에서 소프트웨어 결과물들을 추출하는 것
- 역공학 도구의 구성



역공학의 용도

- 복원된 다이어그램은 다음 여러 방면에 사용
- 프로그램 이해
 - 소프트웨어의 구조, 기능, 동작 이해 용이
- 정형적 분석
 - 소프트웨어에 존재할 수 있는 문제를 감지
- 테스트 케이스 생성
 - 흐름도의 경로 - 경로 테스트 케이스에 도움
- 리엔지니어링

재문서화

- 의미적으로 같은 추상 수준을 가진 표현을 생성하는 작업
- 목적
 - 소프트웨어의 이해를 증진시키기 위하여 시스템의 다른 관점
 - 현재 보유한 문서를 개선
 - 새로 수정된 프로그램의 문서화

설계 복구

- 원시코드를 자세히 검토하여 의미 있는 추상성 높은 표현을 찾아내고 추출하는 작업
- 복구된 설계
 - 원시코드 이해에 도움이 될 수 있음
 - 향후 유지보수 또는 리엔지니어링을 위한 베이스라인으로 사용
 - 유사한 다른 애플리케이션을 위하여 사용될 수도 있음
- 프로그래밍 언어 구조에 크게 좌우
 - 객체지향 프로그램 – UML 도구에 의하여 자동화
 - 원시코드에 내재된 설계의 의미, 의사결정을 파악 설계로 표현하는 작업
- 도메인 지식이 필요할 수도 있음

11. 5 리엔지니어링

- 시스템 또는 컴포넌트를 재구조화 하는 과정
- 목적
 - 소프트웨어 아키텍처 개선
 - 소프트웨어의 복잡도 경감
 - 변경에 대한 적응성 개선
 - 성능, 효율성, 자원 유용성 개선
 - 소프트웨어 시스템의 유지보수 개선

리엔지니어링 과정

- 개선이 필요한 위치 파악
- 개선 전략을 선택
- 제안된 개선의 구현
- 목표를 기준으로 시스템 평가

