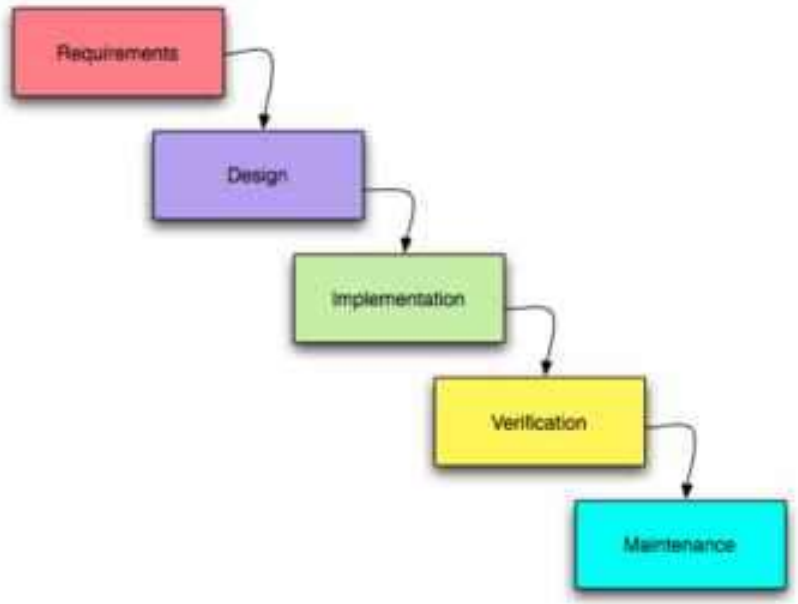
Chap 02 프로세스와 방법론

소프트웨어 공학의 모든 것

최은만

생능출판사

목 차



프로세스 없는 개발

- Code-and-fix
  - 설계하는 작업의 중요성을 깨닫지 못함
  - 계획이 없어 작업 목표가 없음
  - 체계적인 테스트 작업이나 품질 보증 차원의 활동에 대한 필요성의 인식이 없음

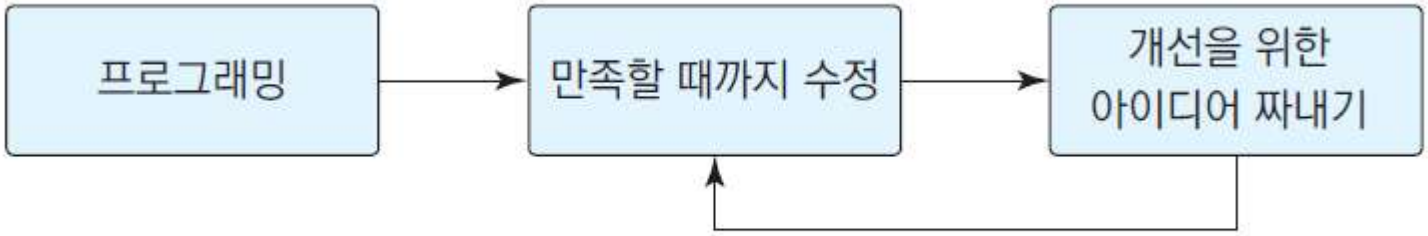


그림 2.1 프로세스 없는 소프트웨어 개발

프로세스와 방법론의 비교

	프로세스	방법론
특징	• 단계적인 작업의 틀을 정의한 것 • 무엇을 하는가에 중점 • 결과물이 표현에 대하여 언급 없음 • 패러다임에 독립적 • 각 단계가 다른 방법론으로도 실현 가능	• 프로세스의 구체적인 구현에 이름 • 어떻게 하는가에 중점 • 결과물을 어떻게 표현하는지 표시 • 패러다임에 종속적 • 각 단계의 절차, 기술, 가이드라인을 제시
사례	• 폭포수 프로세스 • 나선형 프로세스 • 프로토타이핑 프로세스 • Unified 프로세스 • 애자일 프로세스	• 구조적 분석, 설계 방법론 • 객체지향 방법론 • 컴포넌트 • 애자일 방법론

2.1 소프트웨어 생명주기

- 소프트웨어 개발에 대한 기술적, 관리적 이슈를 다루는 작업
  - 개발 모델별 컴포넌트 프로세스, 부프로세스 존재
  - 서로 다른 목적
  - 서로 협력하여 전체 목적을 만족

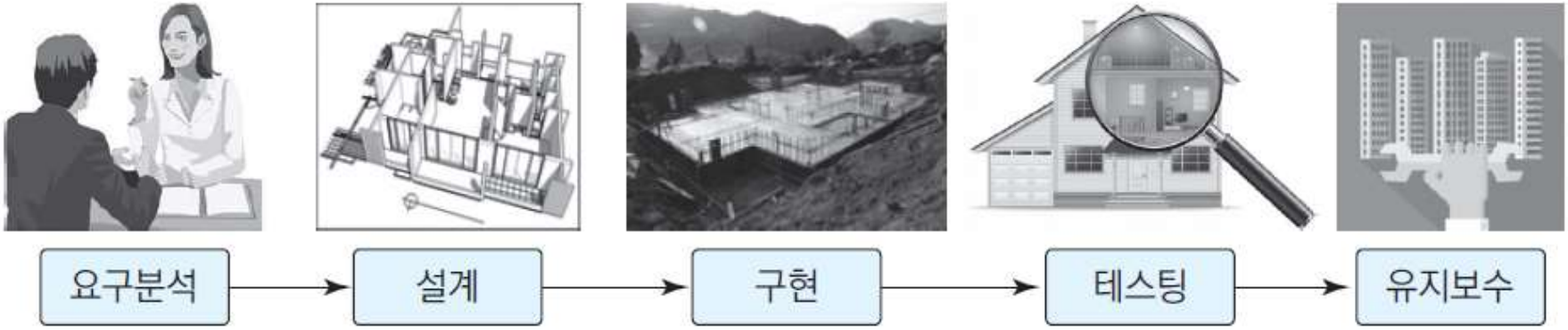
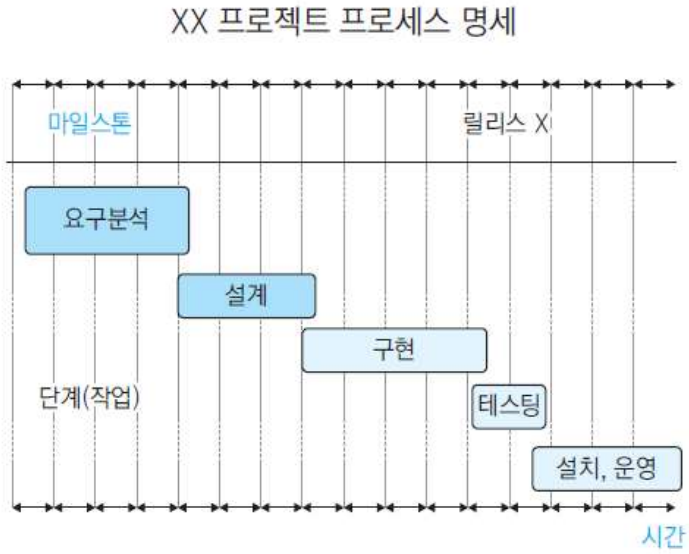
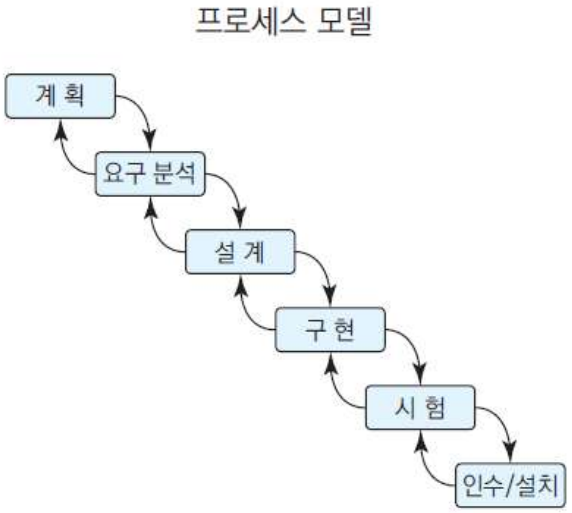


그림 2.2 소프트웨어 생명주기

2.2 프로세스

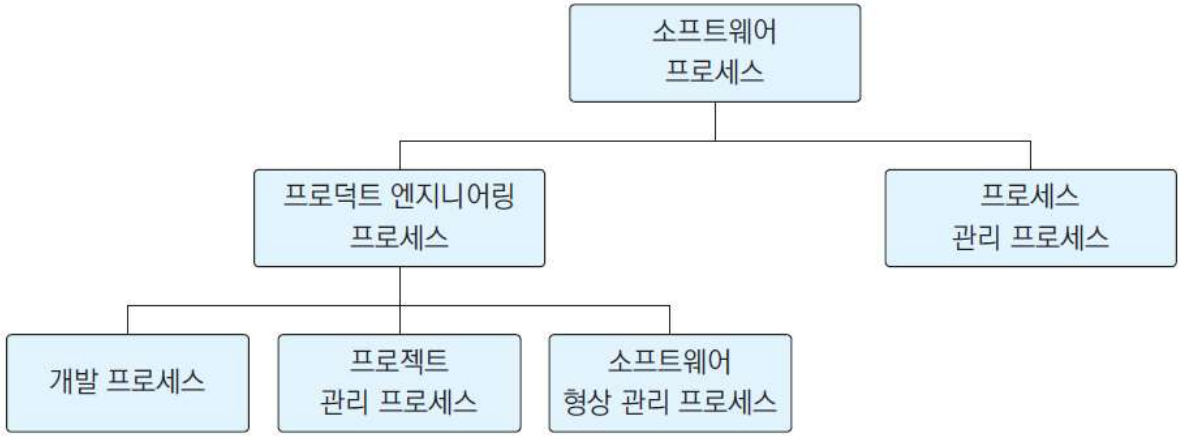
- 프로세스
  - 소프트웨어 시스템을 구축하기 위하여 수행되는 작업의 단계
  - 소프트웨어 개발에 대한 기술적, 관리적 이슈를 다루는 작업





프로세스의 종류

- 프로젝트의 중심 프로세스
  - 개발 프로세스
  - 관리 프로세스
- 기타 프로세스
  - 형상 관리 프로세스
  - 프로세스 관리 프로세스



프로세스의 정의

- 작업 결과와 검증 조건을 명확히 정의하여야 함
- 작업 방법
- 진입 조건, 출구 조건

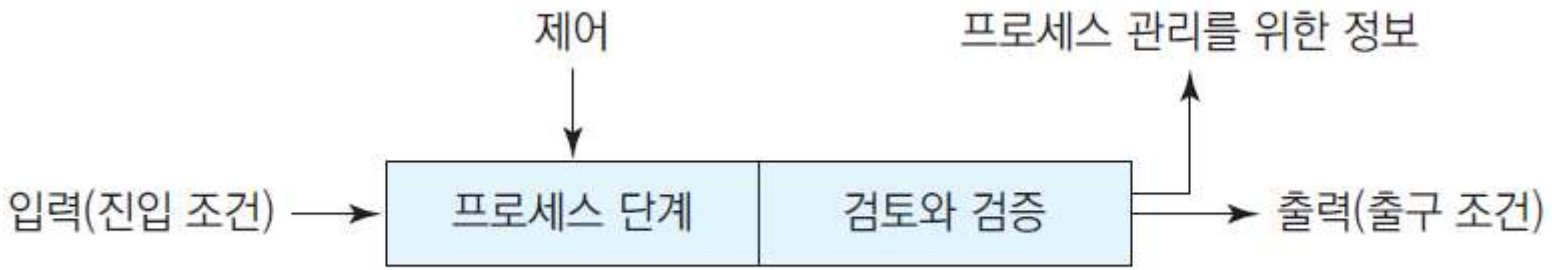


그림 2.5 개발 프로세스의 정의

좋은 프로세스의 특성

- 예측 가능성

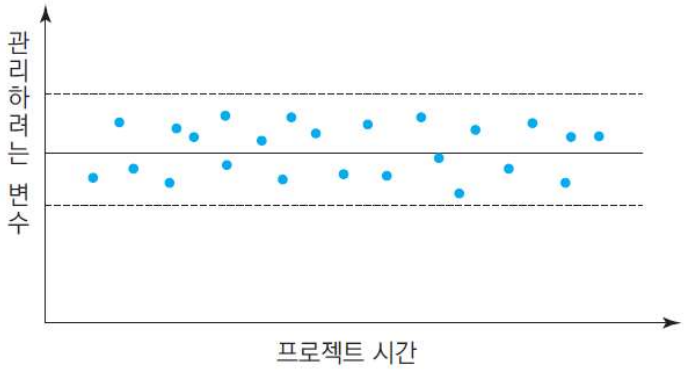


그림 2.6 통계에 근거한 프로세스 제어

- 테스팅과 유지보수 용이성

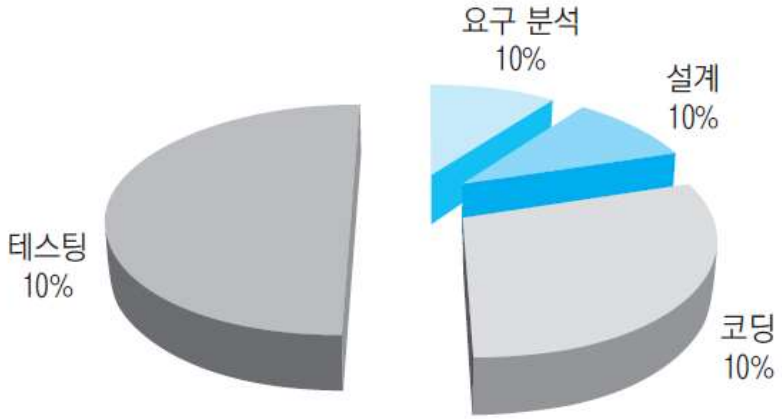
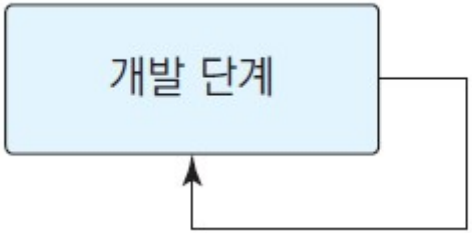


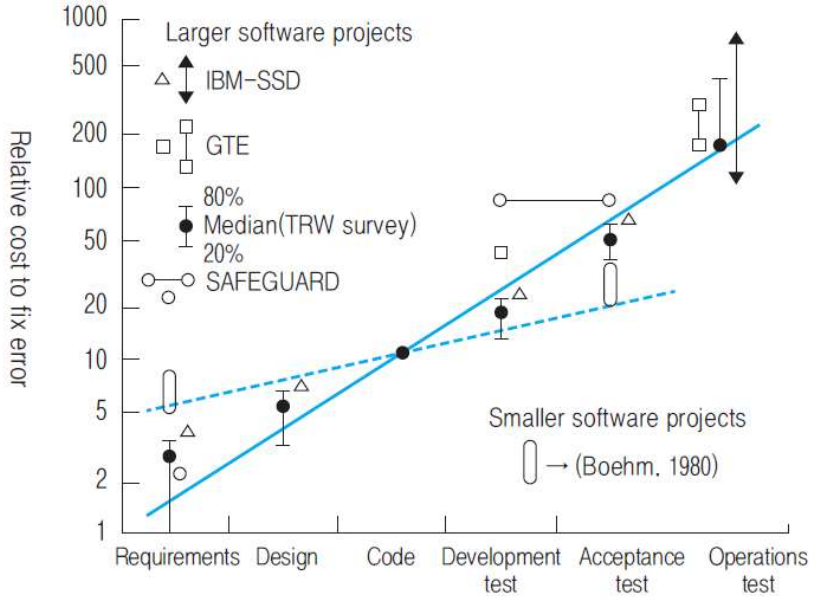
그림 2.7 프로젝트 각 단계의 비용 분포

바람직한 프로세스의 특징(2)

- 변경 지원 – 변경을 쉽게 다룰 수 있는 프로세스



- 결함 제거



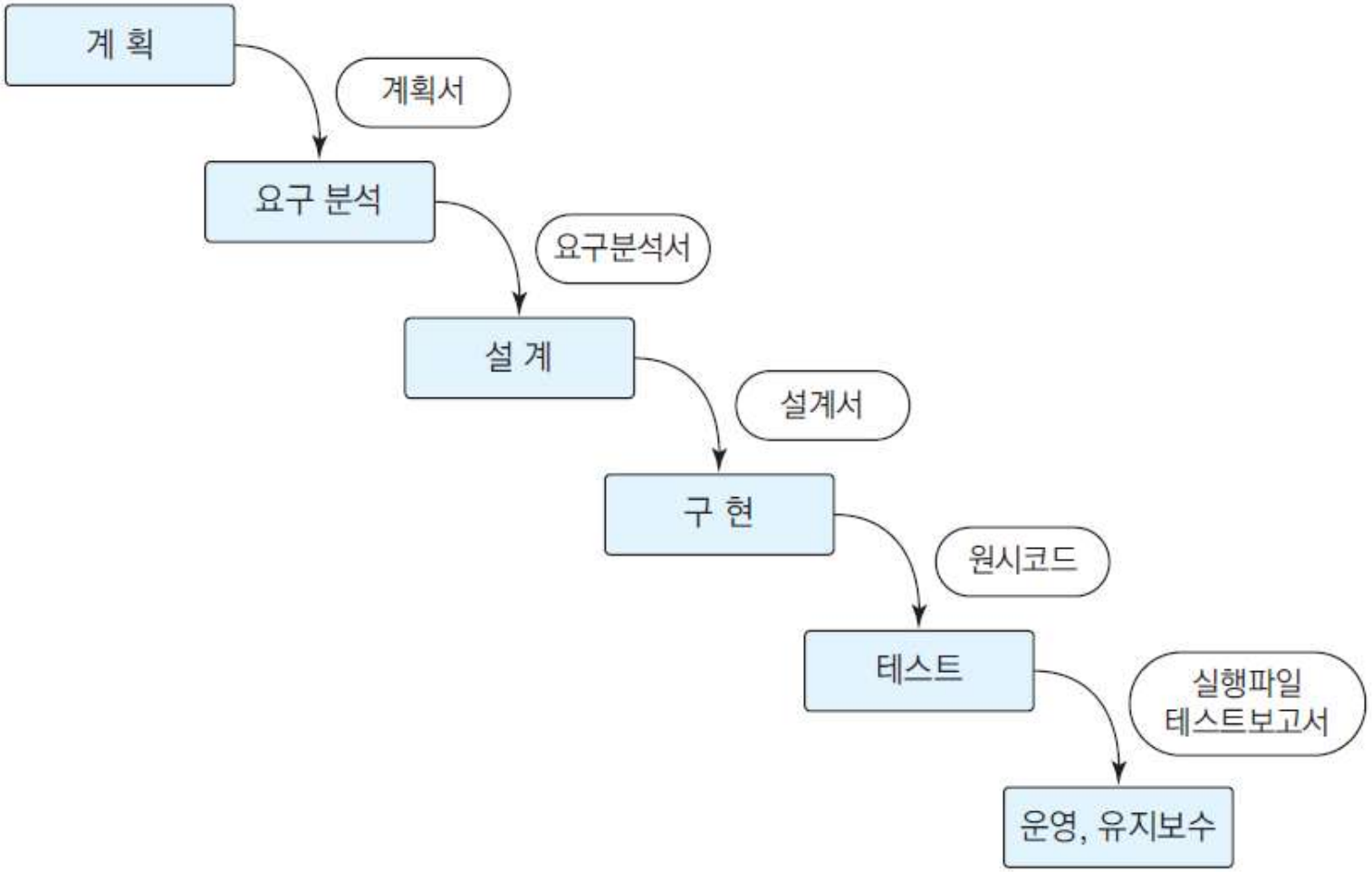
2.3 프로세스 모델

- 프로세스 모델
  - 일반적인 모델이 될만한 프로세스를 기술한 것
- 대표적인 프로세스 모델
  - 폭포수 모델
  - 프로토타이핑 모델
  - 나선형 모델
  - 진화적 모델
  - Unified Process
  - 애자일 프로세스

폭포수(waterfall) 모델

- 가장 오래되고 널리 사용된 프로세스 모델
- 각 단계가 다음 단계 시작 전에 끝나야 함
  - 순서적 - 각 단계 사이에 중복이나 상호작용이 없음
  - 각 단계의 결과는 다음 단계가 시작 되기 전에 점검
- 직능 중심의 프로젝트 조직이
- 결과물 정의가 중요
- 크고 복잡한 오래 지속되는 프로젝트에 적합

폭포수(waterfall) 모델



V 모델

- 검증을 강화하는 관점에서 폭포수 모델을 확장한 모델

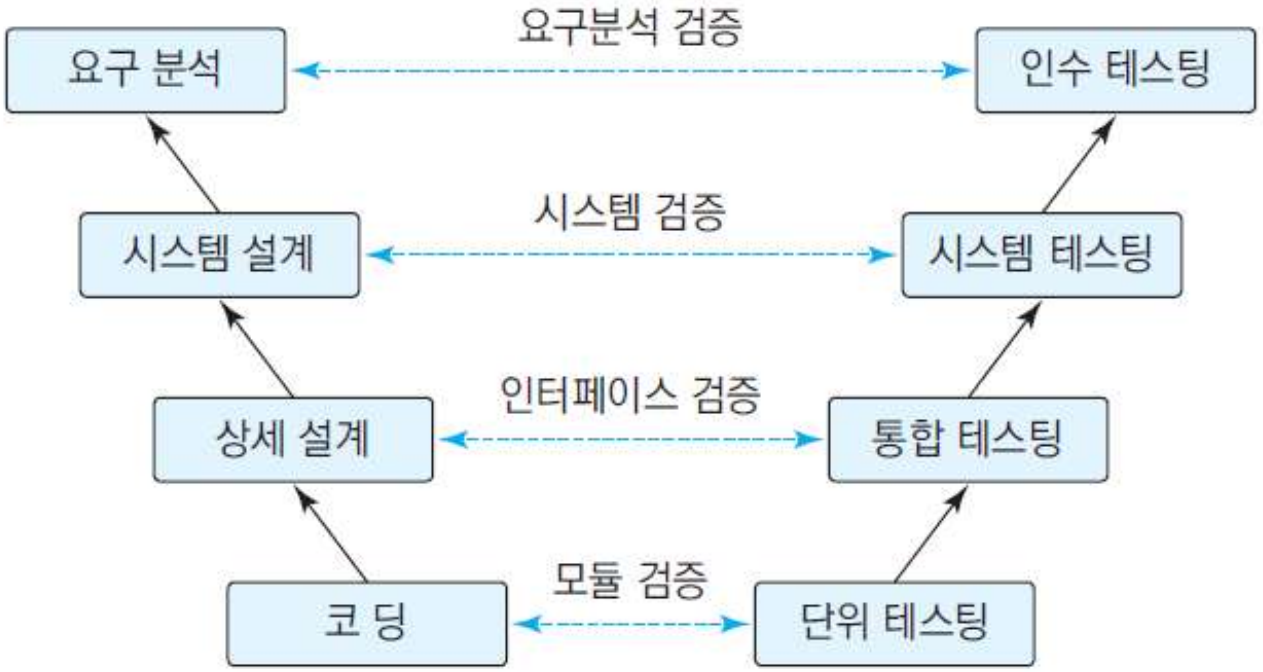


그림 2.11 V 모델

폭포수 모형의 장단점

- 장점
  - 프로세스가 단순하여 초보자가 쉽게 적용 가능
  - 중간 산출물이 명확, 관리하기 좋음
  - 코드 생성 전 충분한 연구와 분석 단계
- 단점
  - 소용 없는 다종의 문서를 생산할 가능성 있음
  - 애매한 부분이 남아 있거나 프로세스 진행 과정에 변경될 수 있는데 이를 수용할 수 없다.
  - 테스트 작업이 프로젝트 후반, 즉 시스템이 완성된 후에 시작됨

프로토타이핑 모델

- 프로토타이핑
  - 요구 사항에 대한 피드백을 받기 위해 시스템을 실험적으로 만들어 사용자에게 보여주고 평가하게 하는 방법
- 프로토타이핑 도구
  - 화면 생성기
  - 시스템의 작동을 시뮬레이션 하여 사용자가 볼 수 있는 반응을 보여줌
- 공동의 참조 모델
  - 사용자와 개발자의 의사소통을 도와주는 좋은 매개체
- 프로토타입의 목적
  - 단순한 요구 추출 – 만들고 버림
  - 제작 가능성 타진 - 개발 단계에서 유지보수가 이루어짐

프로토타이핑 모델

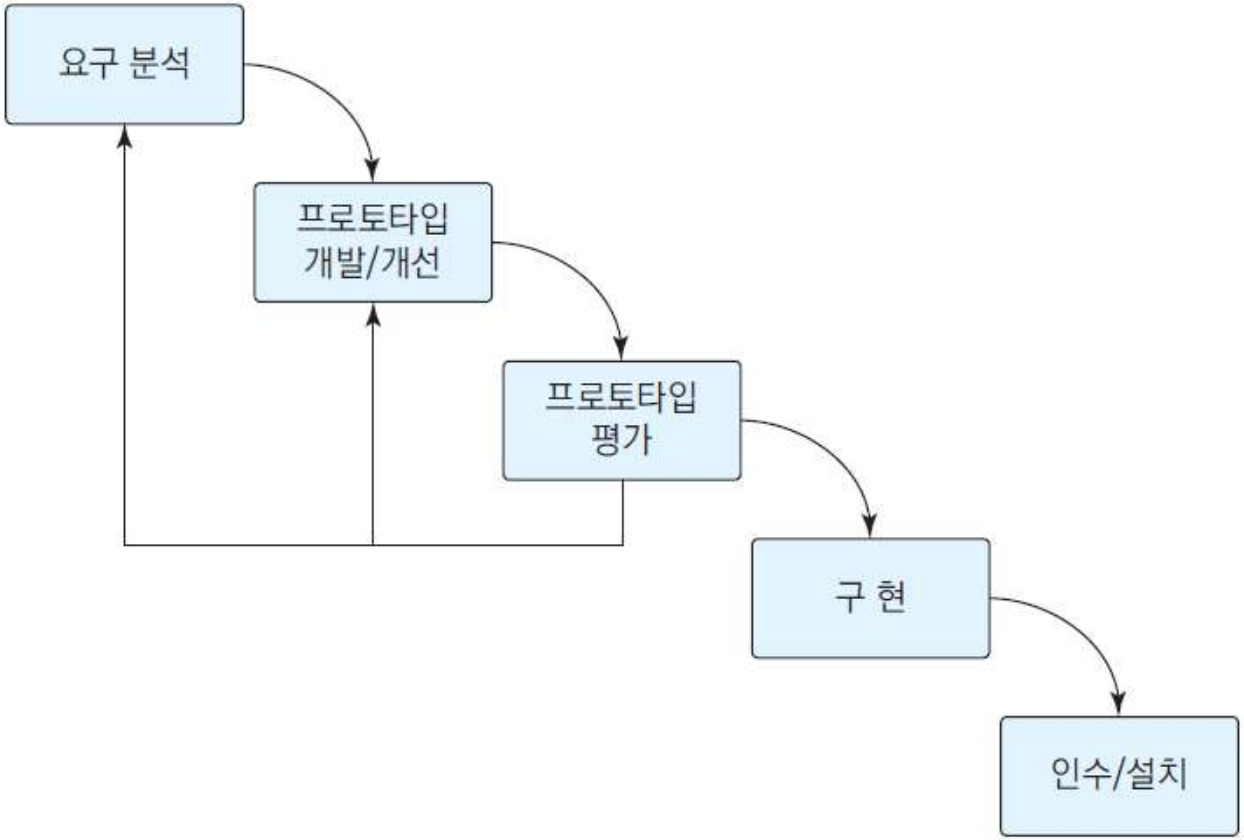


그림 2.13 프로토타이핑 모델

프로토타이핑 모델의 장단점

- 장점
  - 사용자의 의견 반영이 잘 됨
  - 사용자가 더 관심을 가지고 참여할 수 있고 개발자는 요구를 더 정확히 도출할 수 있음
- 단점
  - 오해, 기대심리 유발
  - 관리가 어려움(중간 산출물 정의가 난해)
- 적용
  - 개발 착수 시점에 요구가 불투명할 때
  - 실험적으로 실현 가능성을 타진해 보고 싶을 때
  - 혁신적인 기술을 사용해 보고 싶을 때

나선형(spiral) 모델

- 소프트웨어의 기능을 나누어 점증적으로 개발
  - 실패의 위험을 줄임
  - 테스트 용이
  - 피드백
- 여러 번의 점증적인 릴리스(incremental releases)
- Boehm이 제안
- 반복 순환 단계

① 목표, 방법, 제약 조건 결정
② 위험 요소 분석 및 해결
③ 개발과 평가
④ 다음 단계의 계획

나선형(spiral) 모델

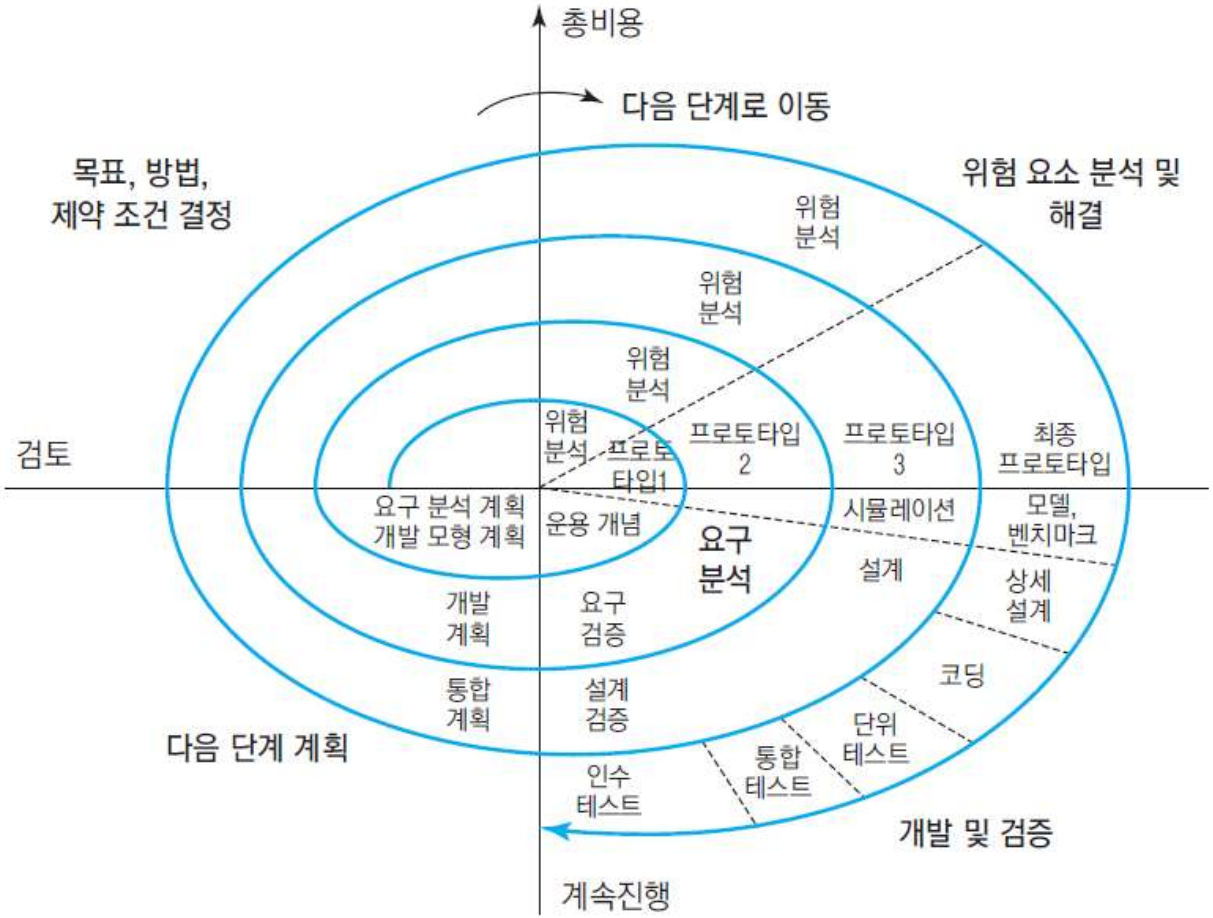


그림 2.14 나선형 모델

나선형(spiral) 모델의 장단점

- 장점
  - 대규모 시스템 개발에 적합 - risk reduction mechanism
  - 반복적인 개발 및 테스트 - 강인성 향상
  - 한 사이클에 추가 못한 기능은 다음 단계에 추가 가능
- 단점
  - 관리가 복잡
  - 위험 분석을 잘못하여 지나친 경우 피해가 큼
  - 성공 사례가 많이 알려지지 않음
- 적용
  - 재정적 또는 기술적으로 위험 부담이 큰 경우
  - 요구 사항이나 아키텍처 이해에 어려운 경우

진화적 모델

- 개발 사이클이 짧은 환경
  - 빠른 시간 안에 시장에 출시하여야 이윤에 직결
  - 개발 시간을 줄이는 법 – 시스템을 나누어 릴리스
- 릴리스 구성 방법
  - 점증적 방법 – 기능별로 릴리스
  - 반복적 방법 – 릴리스 할 때마다 기능의 완성도를 높임

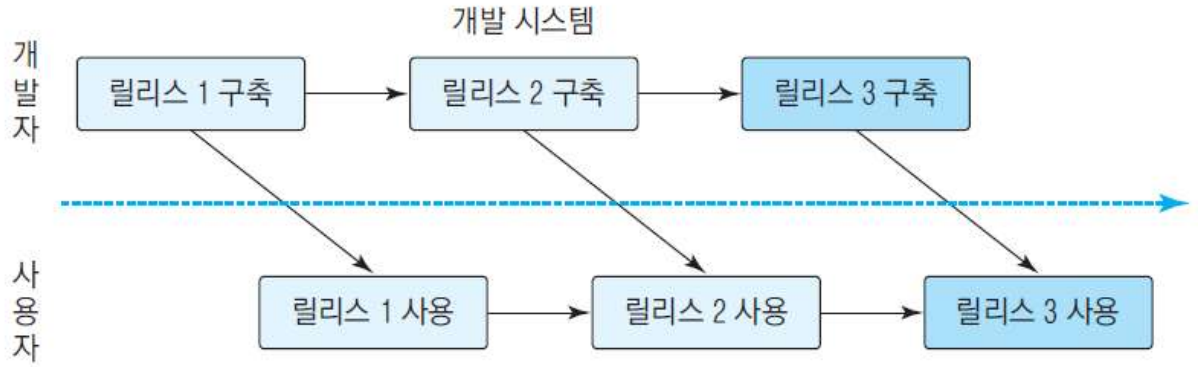
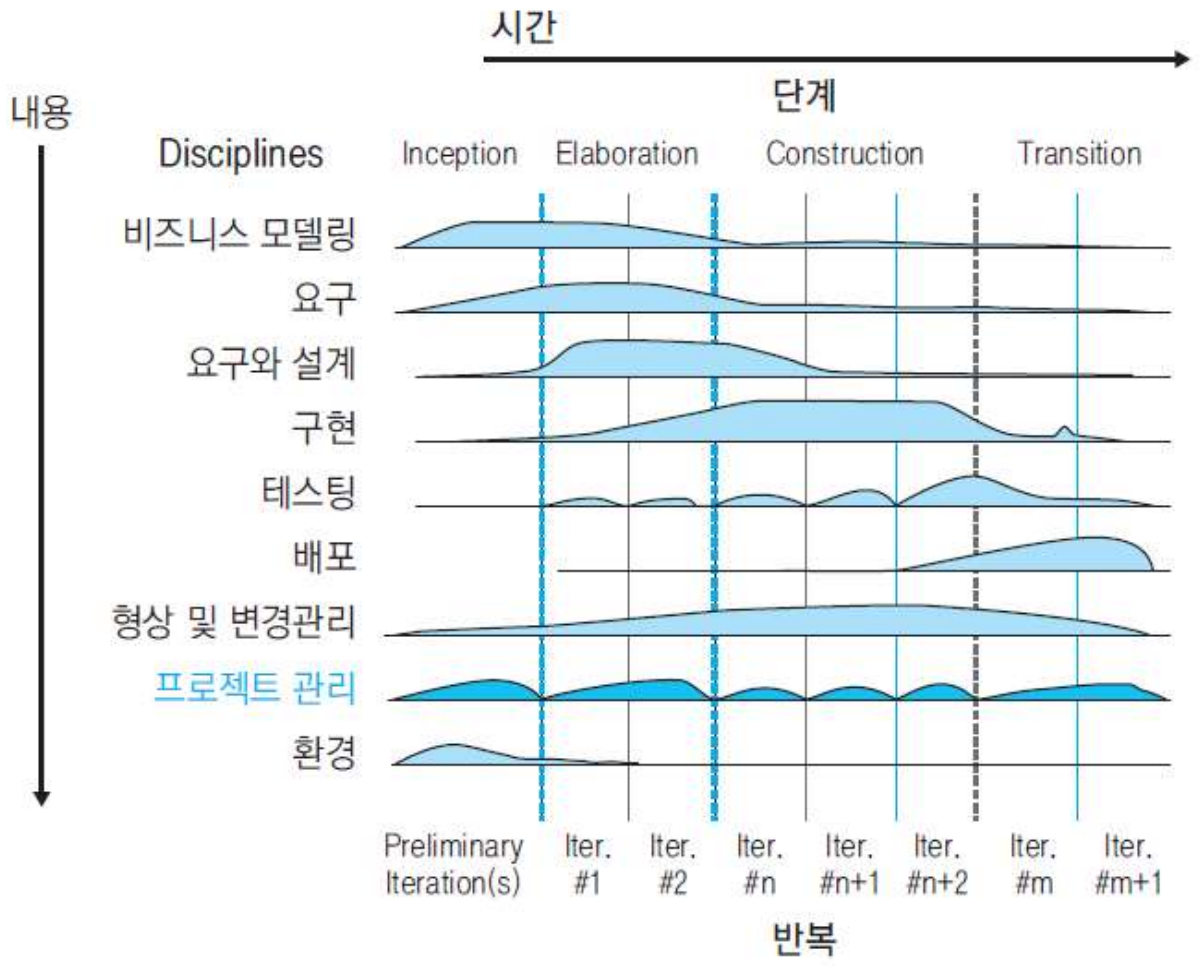


그림 2.15 진화적 모델

진화적 모델의 장단점

- 장점
  - 몇 가지 기능이 부족하더라도 초기에 사용 교육 가능
  - 사용자의 요구를 빠르게 반영
  - 새로운 기능을 가진 소프트웨어에 대한 시장을 빨리 형성
  - 가동 중인 시스템에서 일어나는 예상하지 못했던 문제를 신속하고 꾸준하게 고쳐 나갈 수 있음
- 단점
  - 프로젝트 관리가 복잡해지기 때문에 큰 프로젝트 부적합
  - 끝이 안보일 수 있어 실패의 위험이 커짐
  - 프로젝트의 진행이 위험 분석에 크게 의존

Unified 프로세스



Unified 프로세스

① 도입(inception)
- 1, 2회 정도 반복으로 도입 단계를 진행
- 간단한 유스케이스 모델과 소프트웨어 구조, 프로젝트 계획을 작성

② 정련(elaboration)
- 여러 번의 반복 과정으로 이루어짐
- 대부분의 유스케이스를 작성
- 아키텍처 설계

③ 구축(construction)
- 남아 있는 유스케이스에 대하여 구현하고 통합
- 시스템을 목표 환경에 점증적으로 설치

④ 전환(transition)
- 시스템을 배치, 사용자를 교육
- 베타 테스팅, 결함 수정, 기능 개선

Unified Process의 장단점

- 장점
  - 방법론과 프로세스가 잘 문서화 되어 있어 교육받기 좋음
  - 고객의 요구 변경과 관련된 리스크를 적극적으로 해결
  - 통합을 위한 노력과 시간을 줄일 수 있음
  - 쉽고 빠르게 코드를 재사용
- 단점
  - 프로세스가 너무 복잡, 이해하기 어렵고 정확히 적용하기가 어려움
  - 소프트웨어 프로젝트 참여자들의 협동, 의사소통에 대한 가이드가 없음
  - 조직화되지 않은 개발로 완전히 알려지지 않은 형태의 소프트웨어 개발로 이어질 수 있음

애자일 프로세스

- 2~6주간의 짧은 주기로 개발을 반복
- 실행되는 소프트웨어를 개발하여 단계적으로 시스템 전체를 완성
- 애자일 선언
  1) 형식적인 문서보다는 커뮤니케이션을 통하여 프로젝트가 목표를 향하여 나아가게 함
  2) 사용자는 문서가 아니라 실행되는 소프트웨어를 통하여 요구를 확인
  3) 사용자의 요구는 비즈니스 환경에 따라 프로젝트 중간에 바뀔 수 있음을 고려
  4) 짧은 주기 동안 요구정의에서 구현, 테스트까지 이루어지며 각 반복 주기의 반성 의견을 다음 계획에 포함

익스트림 프로그래밍

- 사용자 스토리
- 매일 빌드와 통합
- 테스트 주도 개발(Test-Driven Development)
- 페어 프로그래밍

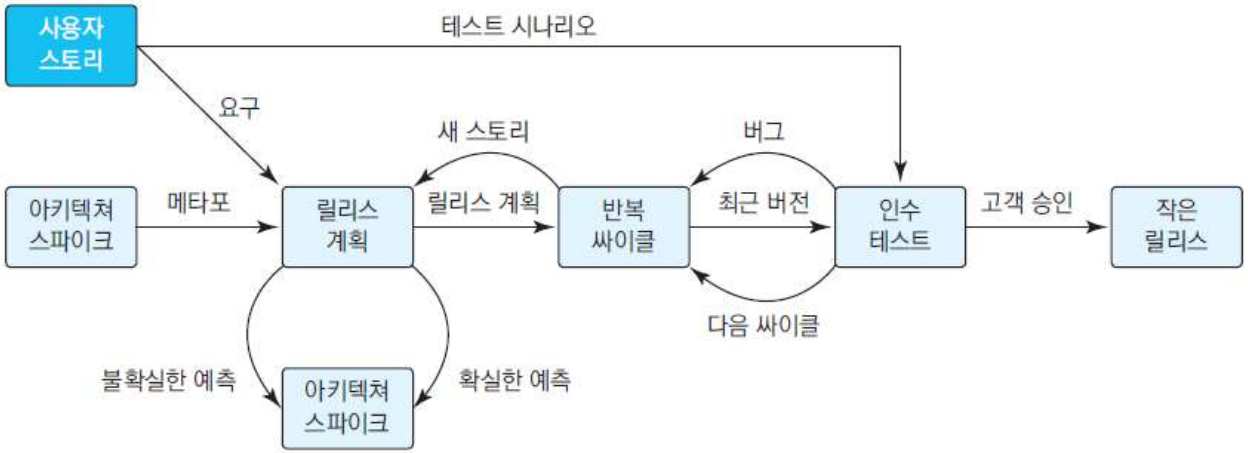


그림 2.17 익스트림 프로그래밍

스크럼

- 개발 팀원 모두가 함께 소통하고 협력하여 짧은 주기를 반복하며 소프트웨어를 개발하는 작업, 역할, 결과물
- 백로그를 정하고 여기에 우선순위를 부여
- 짧은 주기(스프린트)

2.4 지원 프로세스

- ISO/IEC 12207에서의 프로세스 그룹

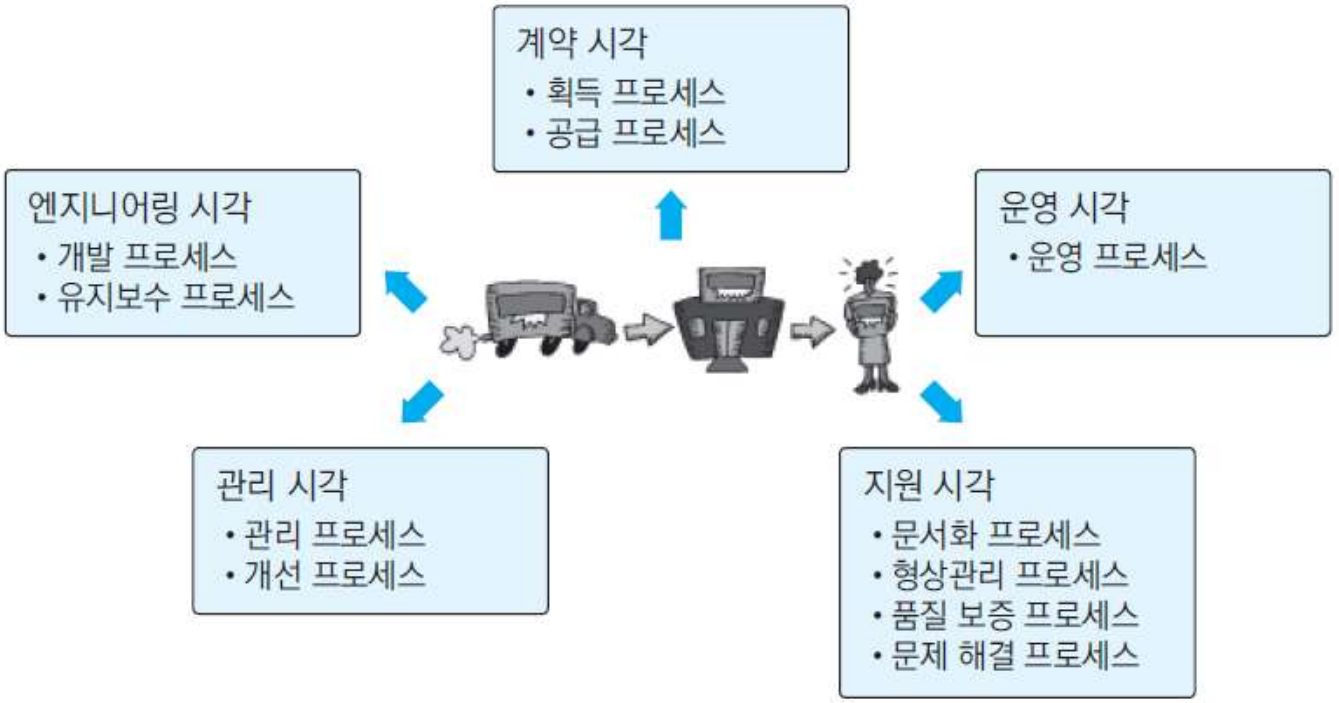


그림 2.19 ISO/IEC 12207에서의 프로세스 그룹

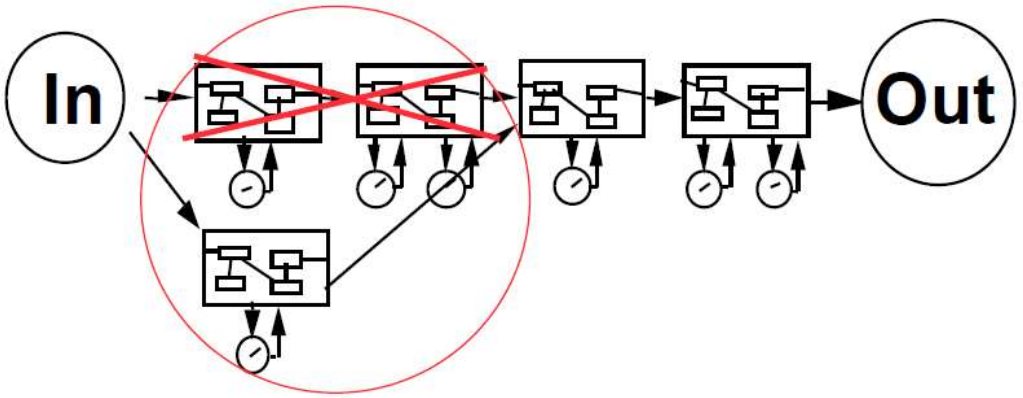
관리 프로세스

- 비용과 품질 목표를 달성하기 위하여 프로젝트 관리하는 데 필요한 모든 작업
  - 계획, 모니터링과 제어, 분석

- 프로젝트 모니터링과 제어는 개발 프로세스의 모든 단계를 포함하므로 가장 긴 기간 동안 이루어짐

품질 보증 프로세스

- 프로세스와 프로덕트에 대한 품질을 관리하고 향상시키는 것
- 인스펙션 프로세스
  - 개발 결과에서 결함을 찾거나 방지하기 위한 노력
  - 정의된 프로세스에 따라 동료 그룹이 작업 결과를 검사하는 것
- 프로세스 관리 프로세스



형상 관리 프로세스

- 개발 중에 발생하는 변경을 체계적으로 컨트롤 하는 것
- 개발작업과 독립적인 작업

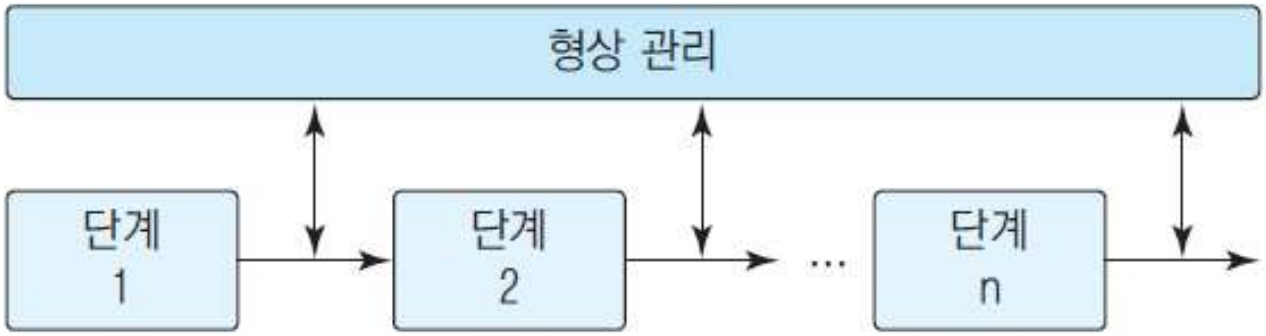


그림 2.22 형상 관리와 개발 프로세스

2.5 방법론

- 방법론
  - 소프트웨어 프로세스의 각 작업을 어떻게 수행하느냐를 정의

- 프로세스
  - 일반적으로 개발할 때 하여하는 할 작업만을 명시
  - 어떤 관계가 있는지 나타내지 않음

구조적 방법론

- 분리와 정복(divide and conquer)원리 적용
- 자료 흐름도를 구조도로 변경하는 과정
  - 구조도 : 모듈 사이의 관계를 나타내는 그래프

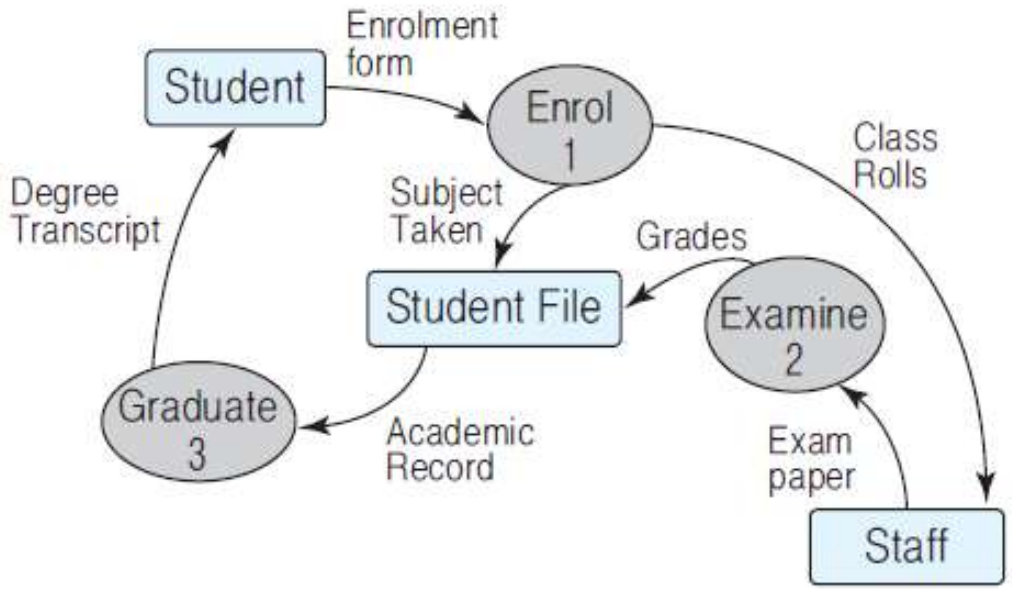


그림 2.23 자료 흐름도

정보공학 방법론

- 특징
  - 기업 중심
  - 전략적 시스템 계획 중심
  - 데이터 중심
  - 분할과 정복
  - 공학적 접근
  - 사용자의 적극적 참여

절차와 방법

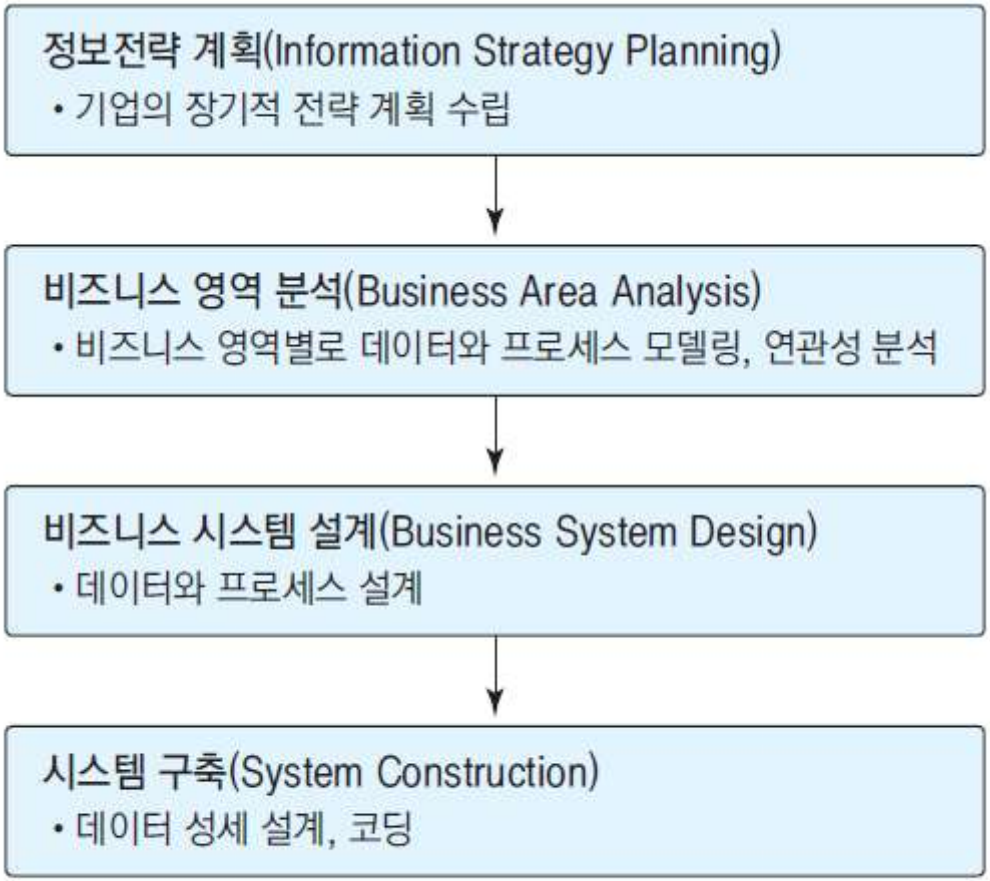
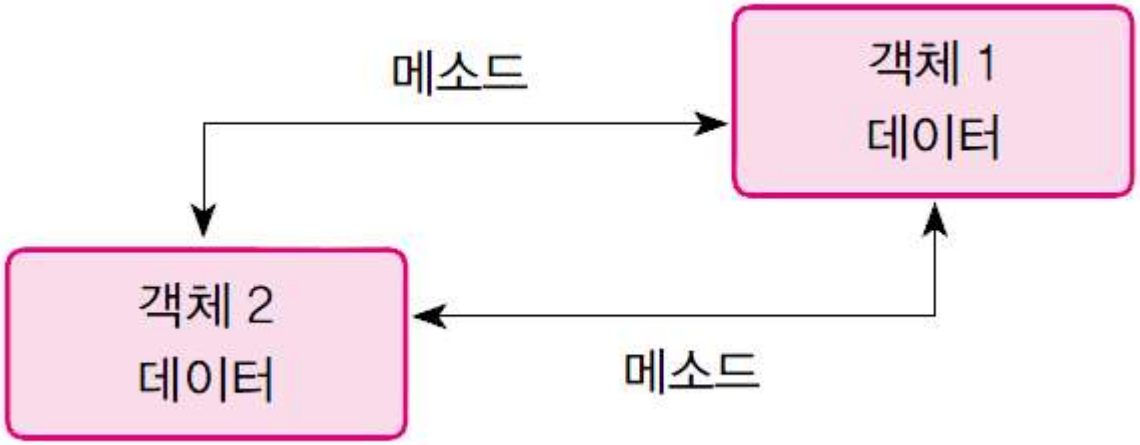


그림 2.24 정보공학 방법론의 절차

객체지향 방법론

- 자료와 함수를 가까운 곳에 정의하여 객체로 묶어두고 객체 사이에 메시지를 호출하여 원하는 기능을 담당하게 하는 것

- 객체지향 패러다임



세 가지 방법론의 비교

표 2.2 방법론의 비교

	구조적 방법론	정보공학 방법론	객체지향 방법론
특징	프로그램 로직 중심. 그림(DFD, 구조적 다이어그램)을 그려 요구사항을 분석, 문서화 하는 체계적 방법	기업정보 중심이며 전략 계획을 수립한 후 데이터 중심으로 CASE 도구를 사용하여 공학적으로 접근한다.	비즈니스를 유스케이스로 분석하고 자료와 함수를 묶은 클래스를 파악하여 상호작용하는 객체들로 시스템을 구성하는 방법
설계 관심사	함수위주	자료 위주	클래스 위주
설계의 핵심	모듈	엔티티	객체
중심 방법	프로그래밍 기법	기업의 전략 및 산출물 중심	설계의 표현