1. 소프트웨어와 소프트웨어공학

소프트웨어란?

❖ 프레스만(Pressman)의 정의

- 하고자 하는 기능이나 성능을 실행하기 위한 명령어(컴퓨터 프로그램)
- 정보를 적합하게 가공하여 프로그램을 구동시키는 자료구조
- 프로그램의 사용과 동작을 설명한 문서들

❖ 용도에 따른 소프트웨어의 구분

- **응용 소프트웨어**
  - 개인용 컴퓨터에서 흔히 접하는 소프트웨어
  - 사용자가 원하는 목적에 맞게 개발된 소프트웨어
    - ➢ 예) 워드 프로세서(Word Processor), 스프레드 시트(Spread Sheet), 브라우저(Browser), 회사 업무 지원 프로그램 등
- **시스템 소프트웨어**
  - 하드웨어를 관리하고 응용 소프트웨어를 지원하는 소프트웨어
    - ➢ 예) 운영 체제(Operating System), 네트워크 관리 프로그램 등

소프트웨어의 특징

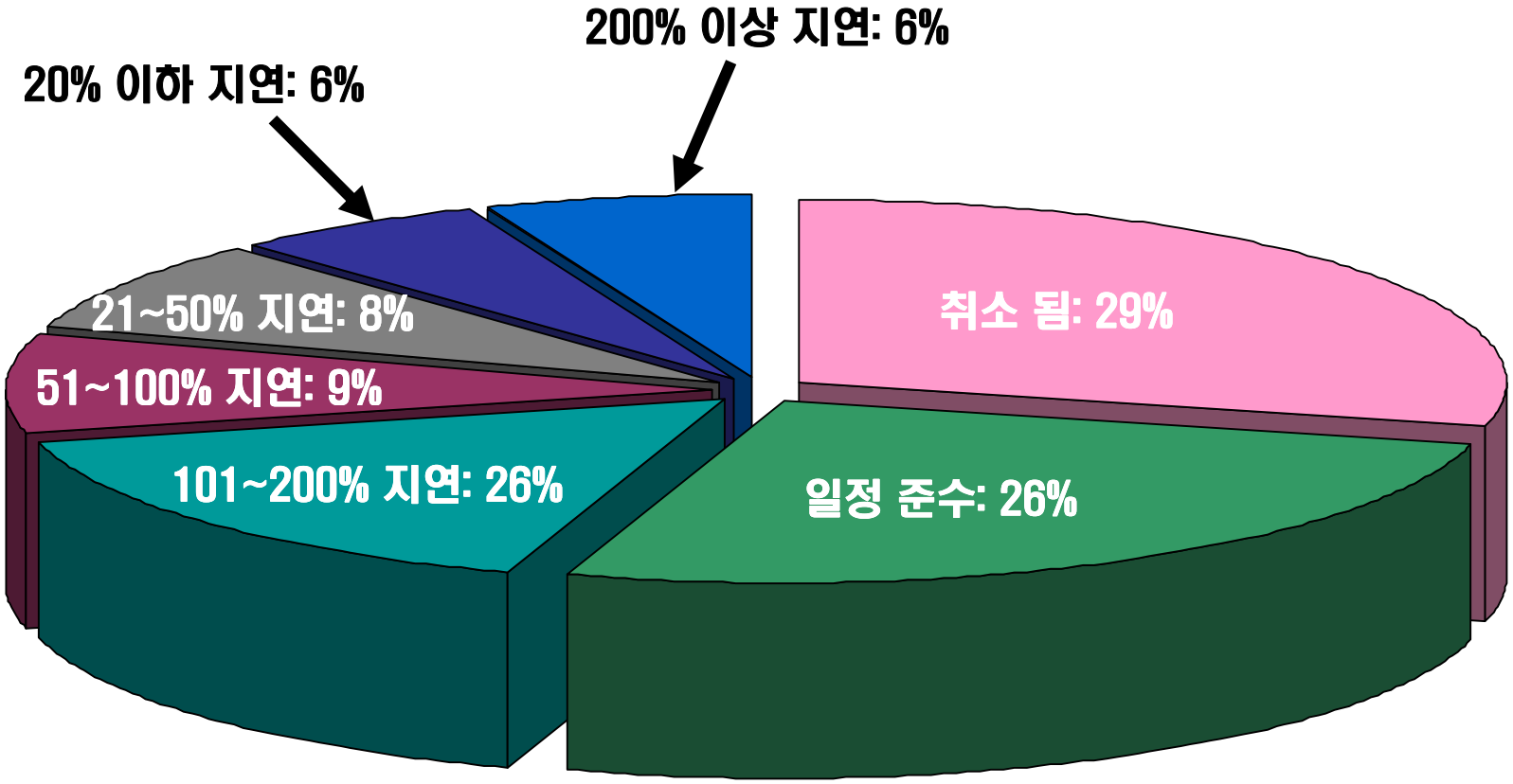
- 소프트웨어의 비가시성(Invisibility)
  - 소프트웨어 완제품의 구조가 개발된 코드 안에 숨어 있어 파악하기 힘든 특징

- 프레스만(Pressman)이 정의한 소프트웨어의 특징
  - 소프트웨어는 고전적인 의미의 '제조(Manufacture)'가 아니라, '개발(Development)'외는 것이다.
  - 소프트웨어는 닳지 않지만, 요구사항의 변경과 주변 환경의 변화에 따라 수정되고 진화한다.

소프트웨어의 특성으로 인한 개발의 어려움

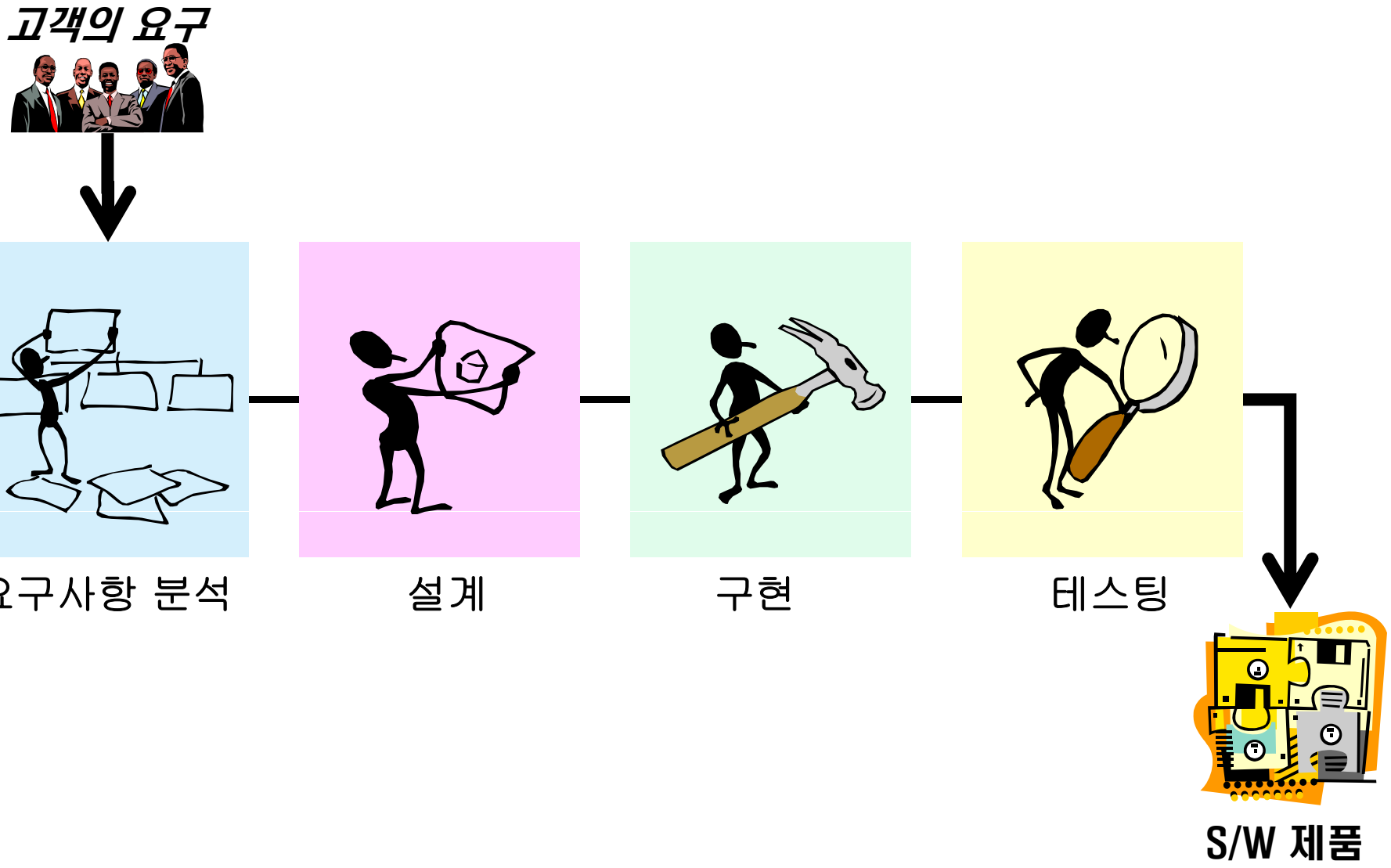
- **소프트웨어는,**
  - **물리적인 형태가 없는 무형의 논리적인 요소**
    - 개발 과정에 대해 정확하게 이해하기 어려움
    - 개발 진행 상황을 파악하기도 어려움
  - **최종 산출물이 개발 과정에서 확인되지 않음**
    - 오류를 발견해야 할 시기를 놓치거나,
    - 오류에 대한 해결책을 못 찾는 경우가 발생
  - **프로젝트의 지연 및 예상 범위 초과로 인한 프로젝트 실패 가능성이 높음**

2001년 미국 소프트웨어 프로젝트 결과



출처: Software Industry Benchmarking Study 2001

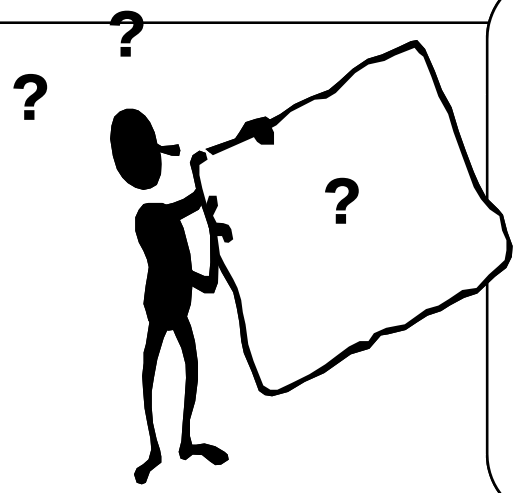
소프트웨어 개발 (1/2)



과거의 소프트웨어 개발

소프트웨어 프로그래밍 = 예술

- 개발자에 따라 다양한 방식이 존재
- 사용자 = 프로그래머 = 유지보수 담당자



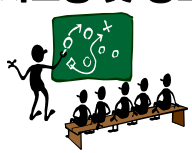
체계적 방법의 부재

- 정형적인 방법론이 거의 없고, 그것을 사용할 수 있는 사람도 거의 없음
- 프로그래머는 시행착오에 의해 기술을 습득 함

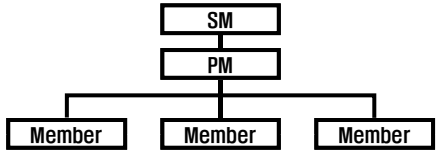
대규모 프로젝트의 어려움

수백 명의 개발자

- 의사소통 및 상호 협력의 어려움

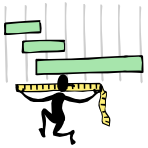


- 조직 및 팀 구조



오랜 개발 시간

- 프로젝트 관리



- 비용 및 효과의 산정

모호하고 복잡한 요구사항

- 수백 페이지의 요구사항

- 빈번한 요구사항의 변화

소프트웨어 공학의 대두 배경

- 소프트웨어 위기(Software crisis)
  - 소프트웨어 수요 증가에 비해 공급 및 개발의 어려움

- 소프트웨어 위기의 해결
  - 다른 분야에서 사용했던 공학(Engineering) 패러다임을 이용하자는 결론
  - 1968년 NATO conference에서 소프트웨어 공학(Software Engineering) 제안됨

소프트웨어 공학이란?

- 정의
  - 소프트웨어의 개발, 운용, 유지보수 및 폐기에 대한 체계적인 접근 방법
- 특징
  - 소프트웨어 개발 전 과정에 걸쳐 필요한 이론, 개념 및 기술을 다룸
  - 소프트웨어 개발 과정에서 생성되는 모든 산출물이 그 대상이 됨
- 목표
  - 소프트웨어 개발이 체계적이고 공학적인 방법으로 이루어져 추정된 비용과 기간에 고객이 원하는 품질 높은 소프트웨어를 개발하는 것

공학이란?

❖ 의미

- 실제적 문제(Practical Problem)를 해결하거나
- 실제적인 산출물을 생산해내기 위해
- 자원과 비용을 효과적으로 활용하면서
- 과학적 지식을 적용하는 것

❖ 공학과 소프트웨어 공학

- **공학**
  - 업무분야에서 문제 발생 시, 실무자가 적절한 해답을 찾을 수 있도록 체계적으로 정리된 기술적 지식을 제공
- **소프트웨어 공학**
  - 소프트웨어 개발 기술, 절차 및 도구의 우수한 사례(Best Practice)들을 정리하여 소프트웨어 개발 시, 누구나 당면한 문제를 해결할 수 있도록 체계적인 기술적 지식을 제공

소프트웨어 공학의 주요 영역들

2. 소프트웨어 프로세스와 생명주기

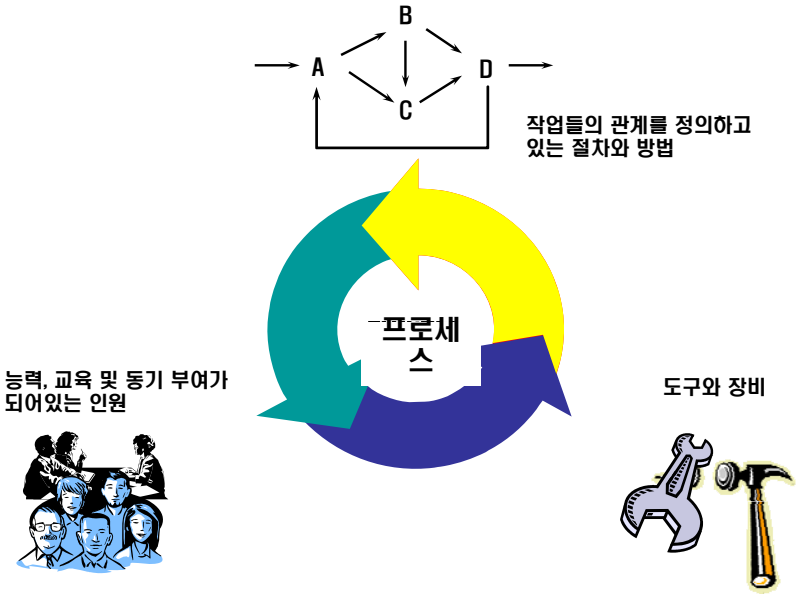
프로세스란?

❖ 의미

- 주어진 목적을 위해 수행되는 일련의 절차

❖ 역할

- 절차, 인력, 기술을 통합
- 각 순서와 활동이 명확하게 정의됨
  - 프로세스를 사용하는 조직원들의 공통된 행동 양식을 지정해주는 역할



소프트웨어 개발 프로세스의 중요성

❖ 소프트웨어 개발의 목표

- 정해진 기한 내에, 주어진 예산을 이용해 사용자가 원하는 좋은 품질로 개발하는 것

❖ 계속되는 프로젝트 실패

- 소프트웨어의 요구사항이 복잡해지고 규모가 점점 커짐
- 정해진 기간 내에 고품질의 소프트웨어를 개발하는 것이 점점 더 어려워짐

❖ 소프트웨어 개발 프로세스의 중요성

- 소프트웨어 제품의 품질은 그 제품을 만들기 위해 사용된 프로세스의 품질에 의해 결정된다 [Watts S. Humphrey]

소프트웨어 프로세스

❖ 정의

- 소프트웨어 개발에 필요한 절차만이 아니라, 그와 관련된 인력, 방법, 도구 들이 통합되는 수단
- 소프트웨어와 이에 관련된 산출물을 개발, 유지하기 위해 사용하는 활동, 방법, 절차의 집합

자료원	소프트웨어 프로세스 정의
IEEE-STD-610	주어진 목적을 달성하기 위한 순서적인 절차 틀
Olson et al.(1989)	특정한 목표나 목적을 달성하기 위한 활동, 작업 및 절차들의 집합.
SEI CMM (Humphrey, 1989: Paulk et al., 1993)	소프트웨어의 생산 및 진화에 사용되는 활동, 방법 및 실무 활동 들의 집합 인력, 절차, 방법, 장치 및 도구들이 원하는 산출물을 생산할 수 있도록 통합하는 수단

소프트웨어 개발 생명주기 (Software Development Life Cycle)

❖ 의미

- 소프트웨어를 어떻게 개발할 것인가에 대한 추상적 표현
- 순차적 또는 병렬적 단계로 구성됨
- 개발 모델 또는 소프트웨어 공학 패러다임이라고도 함

❖ 특징

- 개발 생명주기의 각 단계에 관련된 활동들이 정의되어 있음
- 단계별 활동들을 통해 다음 단계에 활용될 수 있는 산출물이 작성됨
- 전체 프로젝트의 비용 산정과 개발 계획을 수립할 수 있는 기본 골격 제시
- 참여자들 간에 의사소통의 기준과 용어의 표준화를 가능하게 함
- 문서화가 충실한 프로젝트 관리를 가능하게 함

소프트웨어 개발 생명주기 모델의 종류

- 주먹구구식 개발 모델(Build-Fix Model)
- 폭포수 모델(Waterfall Model)
- 원형 모델(Prototyping Model)
- 나선형 모델(Spiral Model)
- UP(Unified Process)
- XP(eXtreme Programming)

주먹구구식 개발 모델(Build-Fix Model) (1/2)

❖ 개요

- 요구사항 분석, 설계 단계 없이 일단 개발에 들어간 후 만족할 때까지 수정작업 수행

❖ 적용 가능한 경우

- 크기가 매우 작은 규모의 소프트웨어 개발

❖ 단점

- 정해진 개발 순서가 없기 때문에
  - 계획이 정확하지 않음
  - 관리자는 프로젝트 진행 상황 파악에 어려움
  - 개발 문서가 없기 때문에 개발 및 유지보수에 어려움

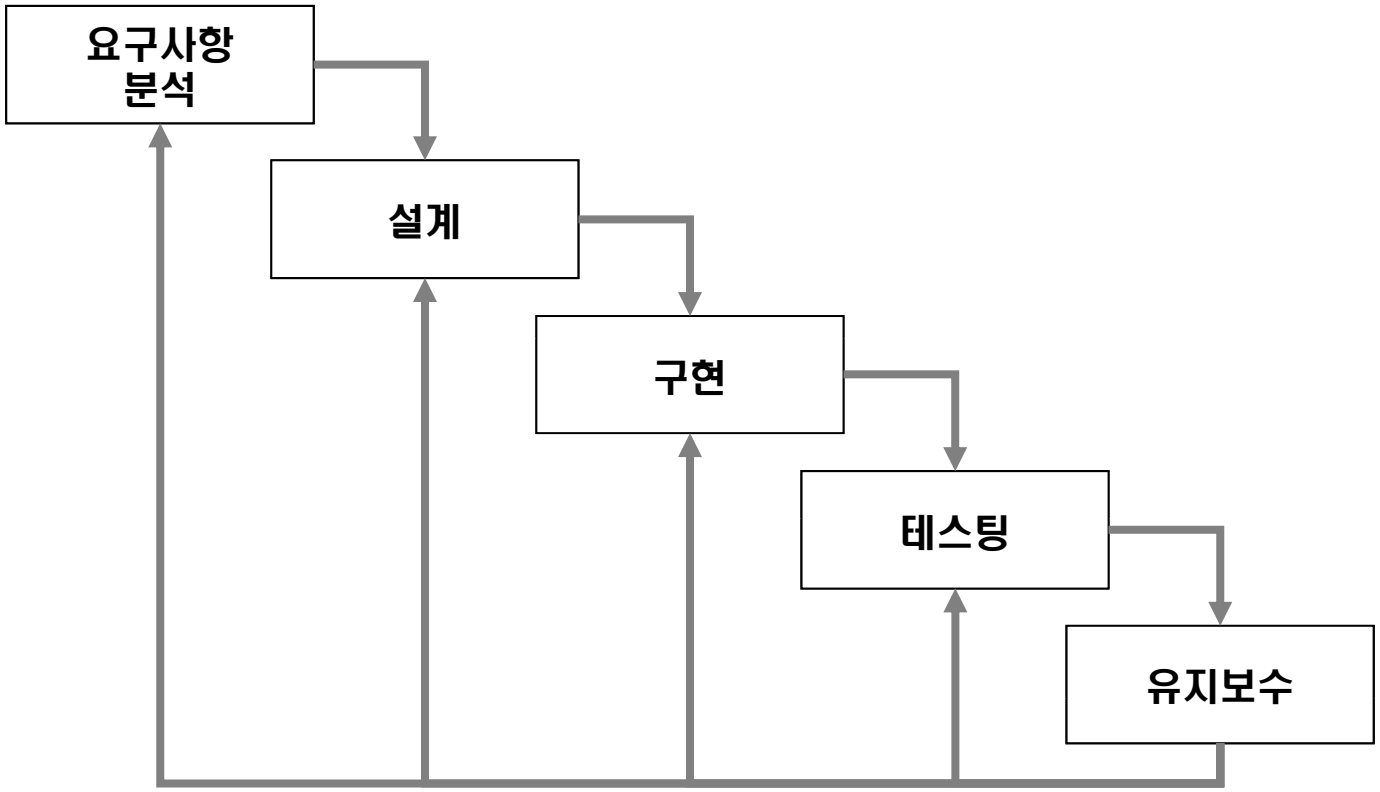
➡ 이후 체계적인 소프트웨어 개발 생명주기 모델의 연구를 가져옴

폭포수 모델(Waterfall Model) (1/4)

❖ 개요

- **순차적으로 소프트웨어를 개발하는 전형적인 개발 모델**
- **대부분의 소프트웨어 개발 프로젝트의 기본적 모델이며 가장 많이 사용되는 모델**
- **소프트웨어 개발의 전 과정을 나누어 체계적이고 순차적으로 접근하는 방법**
  - 개발 과정: 요구사항 분석, 설계, 구현, 테스팅, 유지보수

폭포수 모델(Waterfall Model) (2/4)



폭포수 모델(Waterfall Model) (4/4)

❖ 장점

- 각 단계별로 정형화된 접근 방법 가능
- 체계적인 문서화가 가능하여 프로젝트 진행을 명확하게 할 수 있음

❖ 단점

- 앞 단계가 완료될 때까지 다음 단계들은 대기 상태여야 함
- 실제 작동되는 시스템을 개발 후반부에 확인 가능하기 때문에 고객이 요구사항 확인하는데 많은 시간이 걸림

원형 모델(Prototyping Model) (1/3)

❖ 개요

- 폭포수 모델의 단점을 보완한 모델
- 점진적으로 시스템을 개발해 나가는 접근 방법
- 원형(Prototype)을 만들어 고객과 사용자가 함께 평가한 후 개발될 소프트웨어의 요구사항을 정제하여 보다 완전한 요구사항 명세서를 완성함

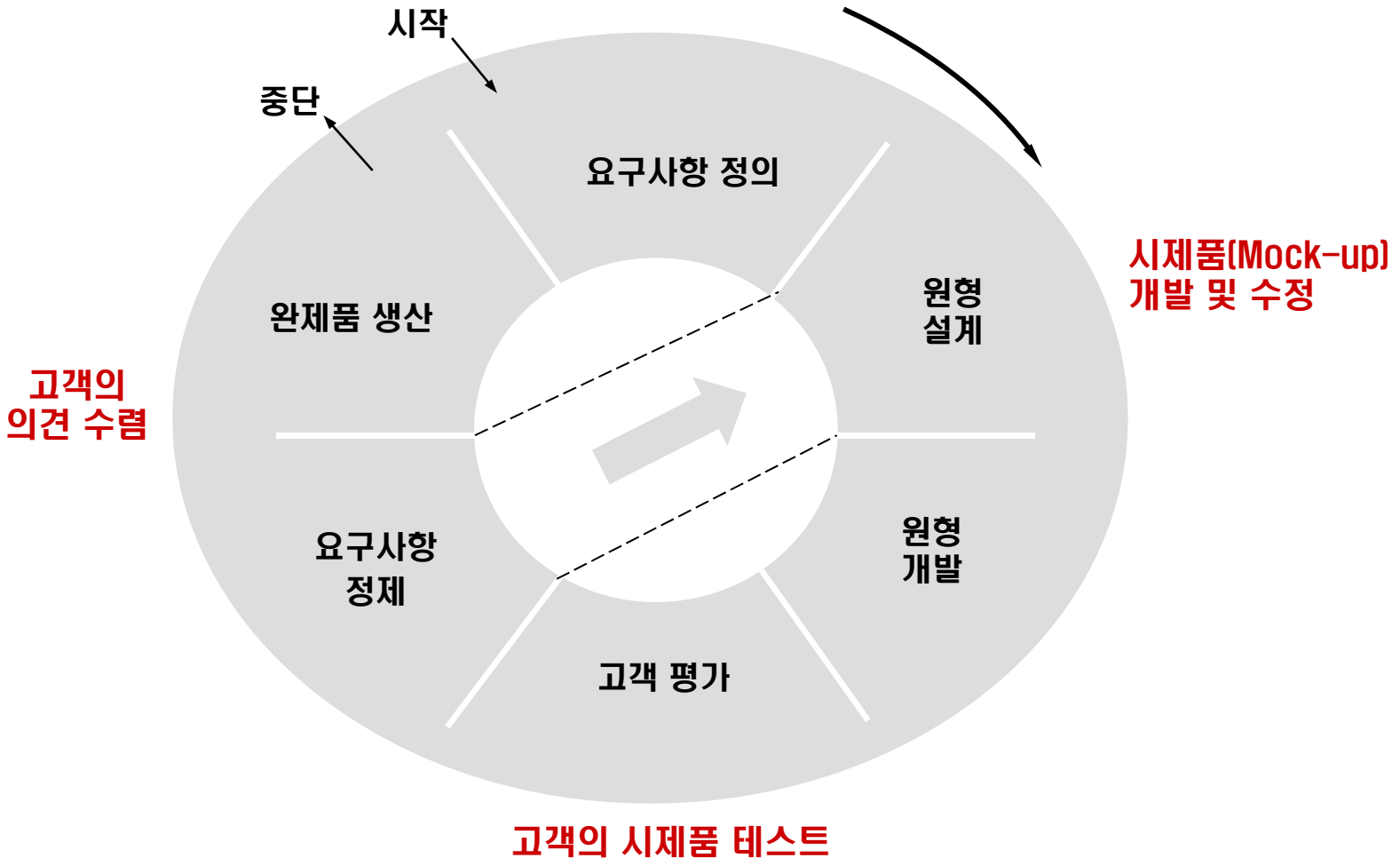
❖ 목적

- 원형을 가능한 빨리 개발하여 고객과 검증하는 것
- 방법
  - 고객으로부터 피드백을 받은 후 원형을 폐기
  - 시스템 기능 중 중요한 부분만 구현하여 피드백을 얻은 후 지속적으로 발전시켜 완제품을 제작

❖ 적용 가능한 경우

- 소프트웨어 개발 초기에 고객 요구사항을 완전히 파악하기 어려울 때

원형 모델 (Prototyping Model) (2/3)



나선형 모델(Spiral Model) (1/4)

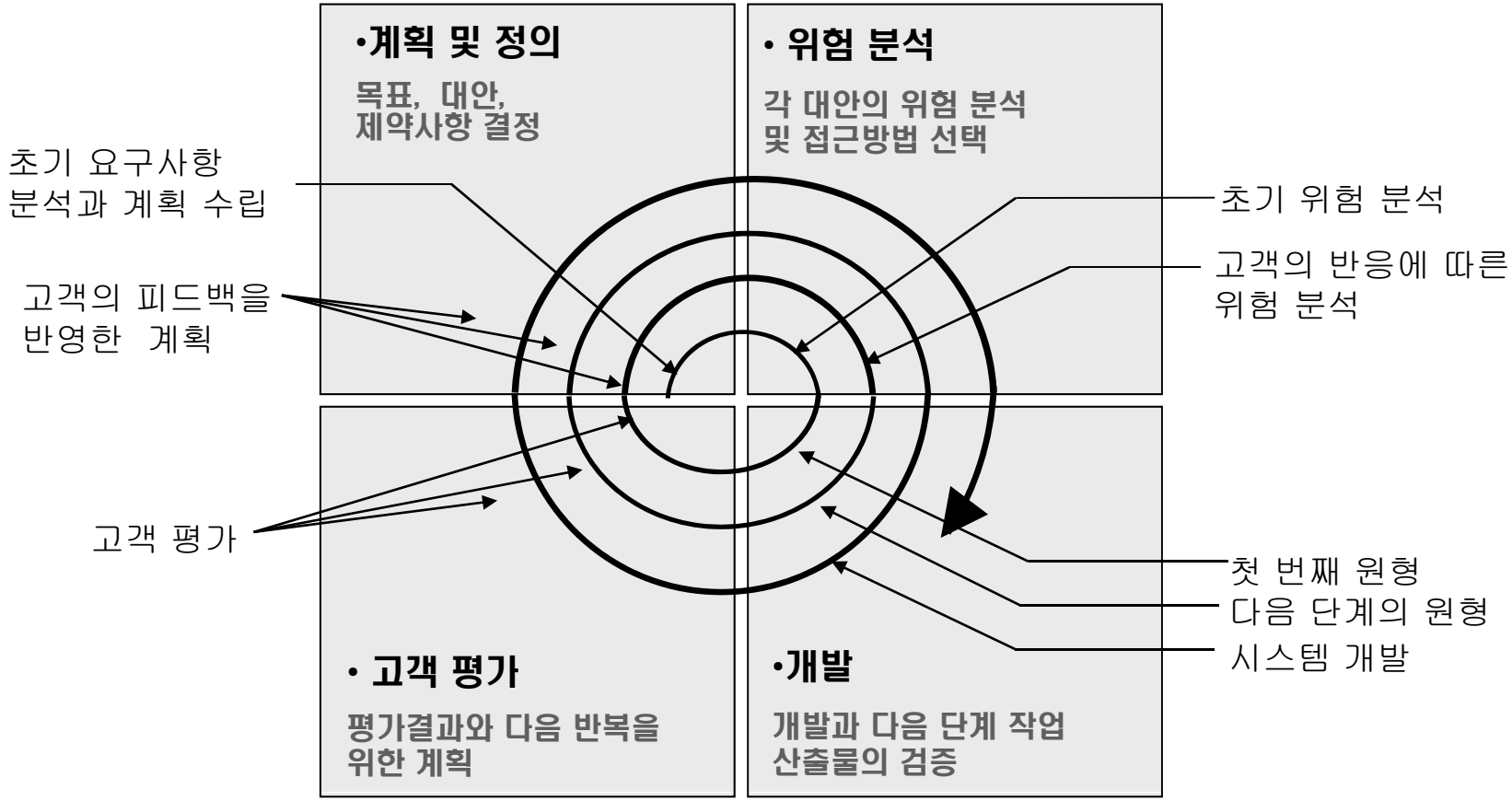
❖ 개요

- 폭포수 모형과 원형 모형의 장점을 수용하고 위험 분석(Risk analysis)을 추가한 점증적 개발 모델
- 프로젝트 수행 시 발생하는 위험을 관리하고 최소화 하려는 것이 목적

❖ 특징

- 여러 개의 작업 영역으로 구분
- 나선상의 각 원은 소프트웨어 개발의 점증적 주기 표현
  - 가장 안쪽 타원부터개념적 개발 프로젝트, 실제 제품 개발 프로젝트, 제품 향상 프로젝트, 유지보수 프로젝트
- 단계가 명확히 구분되지 않고, 엔지니어가 프로젝트 성격이나 진행 상황에 따라 단계 구분

나선형 모델(Spiral Model) (2/4)



나선형 모델(Spiral Model) (4/4)

❖ 적용 가능한 경우

- **개발에 따른 위험을 잘 파악하여 대처할 수 있기 때문에**
  - 고비용의 시스템 개발
  - 시간이 많이 소요되는 큰 시스템 구축 시

❖ 장점

- **프로젝트의 모든 단계에서 기술적인 위험을 직접 고려할 수 있어 사전에 위험 감소 가능**
- **테스트 비용이나 제품 개발 지연 등의 문제 해결 가능**

❖ 단점

- **개발자가 정확하지 않은 위험 분석을 했을 경우 심각한 문제 발생 가능**
- **비교적 새로운 접근 방식**
- **폭포수, 원형 모델에 비해 상대적으로 복잡하여 프로젝트 관리 자체에 어려울 수 있음**

UP(Unified Process) (1/4)

❖ 개요

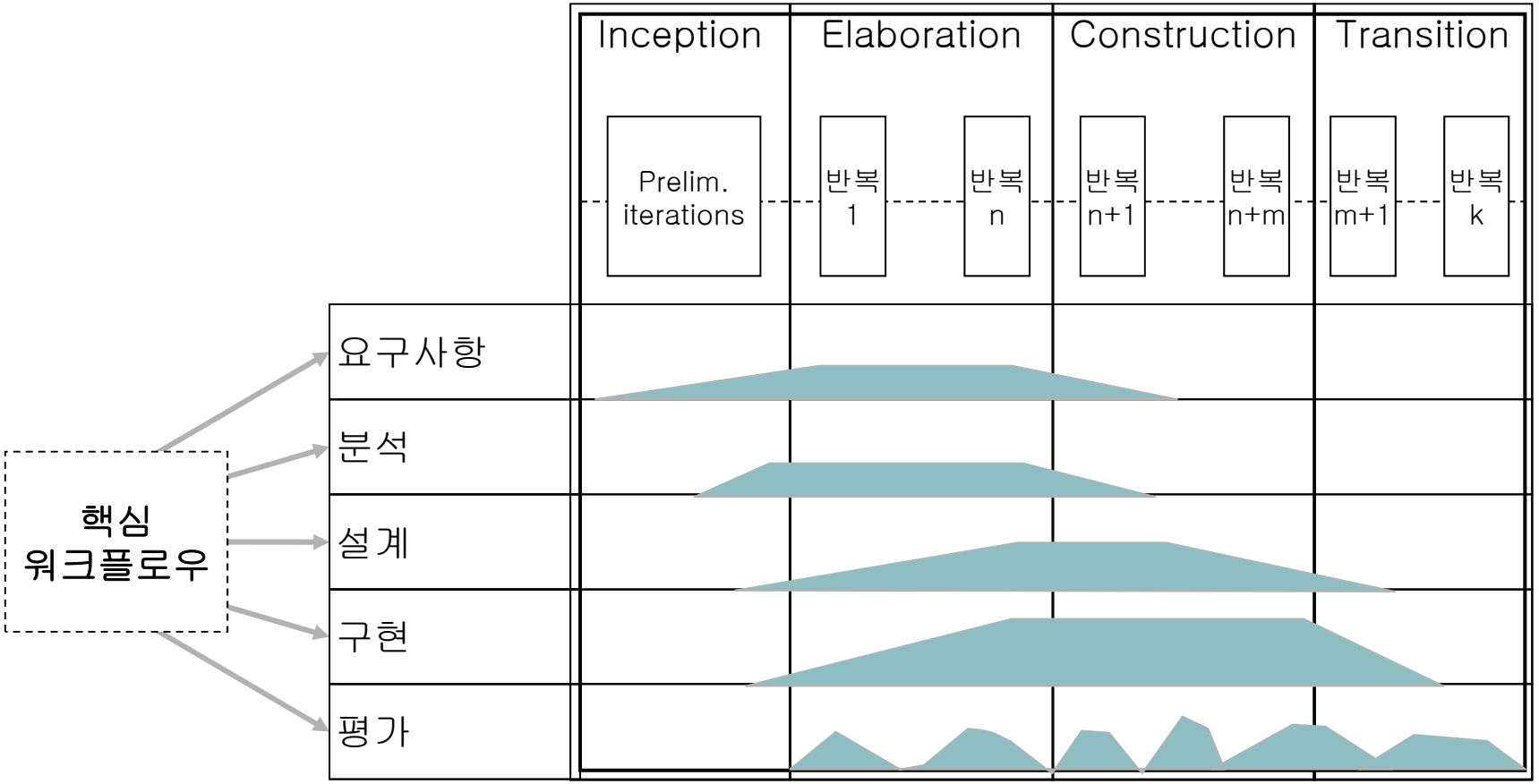
- 1999년 Jacobson, Booch, Rumbaugh에 의해 개발됨
- 소프트웨어 개발이 각각 생명주기를 가지는 여러 번의 반복(iteration)을 거쳐 수행되는 모델

❖ 특징

- 반복마다 실행 가능한 릴리즈가 산출
- 반복이 거듭될수록 향상되어 결국 최종 시스템으로 발전됨

UP(Unified Process) (2/4)

❖ **반복 프로세스의 4가지 범주**



XP(eXtreme Programming) (1/4)

❖ 개요

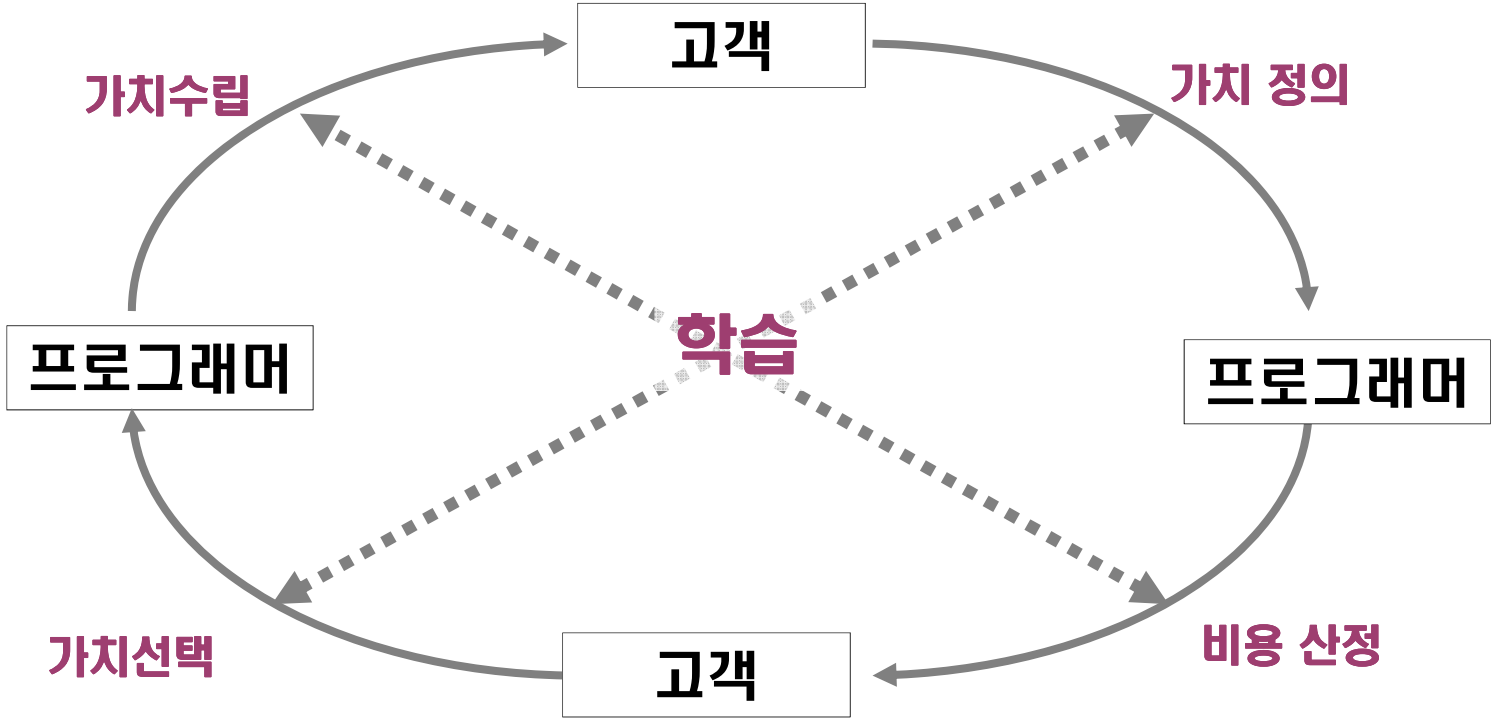
- 1990년대 초, Kent Beck에 의해 고안된 개발 방법론
- 요구사항 변경으로 인한 비용이 개발 기간에 상관없이 일정하게 유지되도록 것을 주목적으로 함
- 고객, 관리자, 프로그래머에 대한 역할 및 권한과 4가지 가치를 중시함
  - 4가지 가치: 의사소통(Communication), 단순성(Simplicity), 피드백(Feedback), 용기(Courage)

❖ 특징

- 프로그래머와 고객, 동료 프로그래머와의 의사소통 중요시 함
- 단순하고 명확한 설계 유지
- 가장 우선순위가 높은 것을 먼저 개발함
- 되도록 초기에 고객에게 시스템을 전달하여 피드백을 받음
- 프로그래머는 요구사항과 기술의 변경에 용감하게 대응할 수 있음

XP(eXtreme Programming) (3/4)

❖ 각 과정에서의 경험을 다음 생명주기에 반영된다.



XP(eXtreme Programming) (4/4)

❖ XP의 12가지 실천 사항

- 계획 세우기(Planning Process)
- 소규모 릴리즈(Small Release)
- 상징(Metaphor)
- 단순한 디자인(Simple Design)
- 지속적인 테스팅(Continuous Testing)
- 리팩토링(Refactoring)
- 짝 프로그래밍(Pair Programming)
- 공동 코드 소유(Collective Code Ownership)
- 지속적인 통합(Continuous Integration)
- 주당 40시간 업무(40 hour Week)
- 현장 고객 지원(On-site Customer)
- 코딩 표준(Coding Standard)

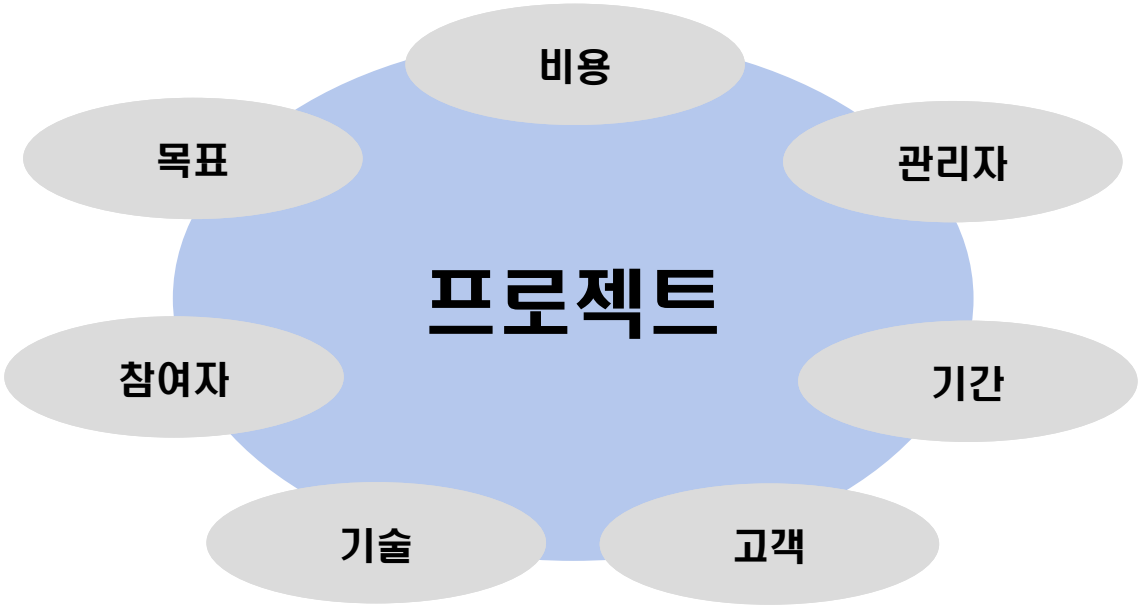
3. 프로젝트 관리

프로젝트란?

❖ 정의

- 프로젝트는 **유일한** 제품이나 서비스를 만들기 위해 수행되어야 할 **일시적인** 행동 [2000 PMBOK(Project Management Body of Knowledge)]
- 같은 일을 반복하는 "일상생활" 과 구분됨

❖ 프로젝트의 구성 요소



소프트웨어 프로젝트

❖ 소프트웨어 개발의 시작

- **조직이 요구사항에 맞는 프로그램을 획득(Acquire)할 필요가 생겼을 때**
  - 시중에 나와있는 프로그램을 구입하거나, 직접 개발하거나
  - 조직에서 개발하거나, 소프트웨어 개발 전문 업체에 의뢰하거나

❖ 소프트웨어 개발

- **발주자(고객)가 요구사항을 주면 수주자(개발자)가 요구사항에 맞는 프로그램을 개발하는 프로젝트**

소프트웨어 프로젝트 프로세스(1/2)

❖ 소프트웨어 제품 구상

- 발주자는
  - 원하는 소프트웨어 제품을 구상하고, 그 가치를 검증
  - 제품의 투자 대비 효과를 예측하고, 사업에 미치는 영향을 파악
  - 원하는 제품의 기능상의 요구사항, 성능 요구사항들을 정의

❖ 소프트웨어 제안 요청서(RFP: Request for Proposal) 배포

- 제품을 자체 개발하지 않는 경우, 개발 회사들에게 제안 요청서를 발송

❖ 제안서 제출

- 개발 회사들은 발주자에게 제안서 제출

일반적인 제안 요청서 양식

목 차

I. 프로젝트 개요
   1. 프로젝트 명
   2. 프로젝트 목적
   3. 프로젝트 결과물
   4. 프로젝트 내용
   5. 특이사항
   6. 기간
   7. 비용
   8. 추진일정

   …

II. 별지서식

소프트웨어 프로젝트 프로세스(2/2)

- **제안서 심사**
  - 이미 정해진 기준에 따라 심사하여 수주자 선정
- **계약서 작성**
  - 수주자가 선정되면 발주자와 수주자 사이에 계약 체결
- **프로젝트 시작 및 수행**
  - 계약이 완료 후 수주자는 프로젝트 시작
  - 마일스톤 별로 또는 발주자의 참여 필요시 회의를 갖고 요구사항의 변경 등 중요한 사항 협의
- **프로젝트 종료 및 제품 인도**
  - 소프트웨어 개발 완료 후 발주자의 인수 테스트를 거쳐 제품이 인도됨

일반적인 계약서 양식

소프트웨어 개발 계약서

한국 발주사(이하 “갑”이라 함)와 개발 코리아(이하 “을”이라 함)는 제 2조에 명시한 “소프트웨어 개발”의 관련 업무 대하여 다음과 같이 계약을 체결한다

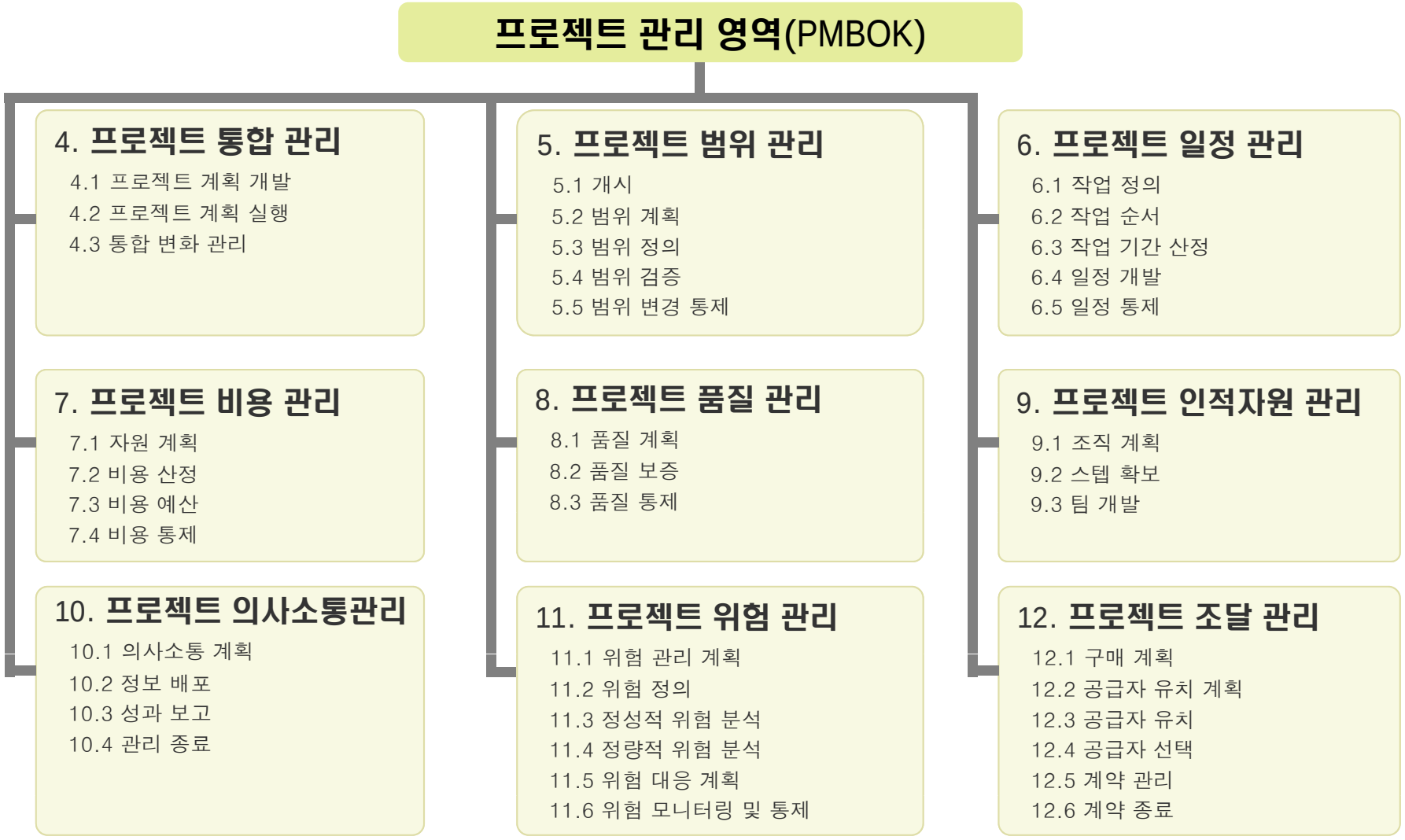
-다 음-

제 1조 계약의 목적
제 2조 계약 내용
제 3조 협조 사항
제 4조 계약 기간
제 5조 계약 금액
제 6조 사용 및 저작권한
제 7조 계약의 해지 및 통보
제 8조 비밀 유지의 의무와 손해배상
제 9조 기타

프로젝트 관리

- 정의
  - 프로젝트의 요구사항을 만족시키기 위해 지식, 기술, 툴 및 기법을 프로젝트 활동에 적용하는 것
    (2000 PMBOK)

PMBOK의 프로젝트 관리 영역



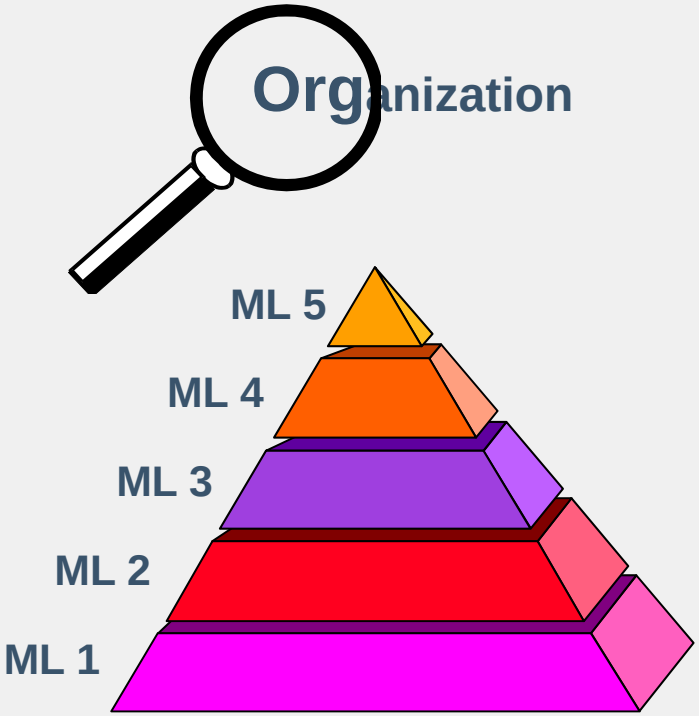
CMMI의 등장

❖ 배경

- 사회적 변화에 적응하고 기존의 문제점 해결하기 위한 새로운 모델의 필요성 제기
- 2000년 8월, 모델들을 통합, 정리하여 ISO15504(SPICE)와 호환 가능한 통합 모델인 CMMI(Capability Maturity Model Integration) 발표

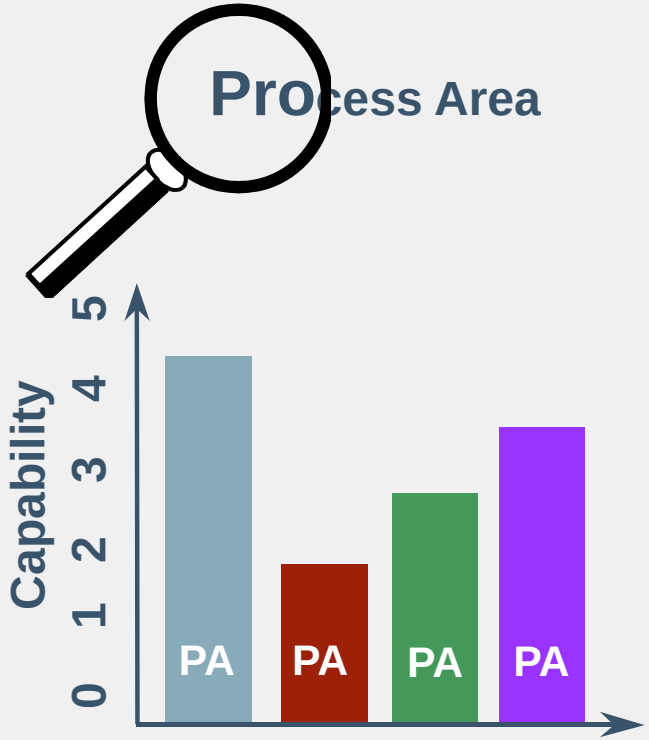
CMMI 모델 표현

단계적 표현
(Staged Representation)



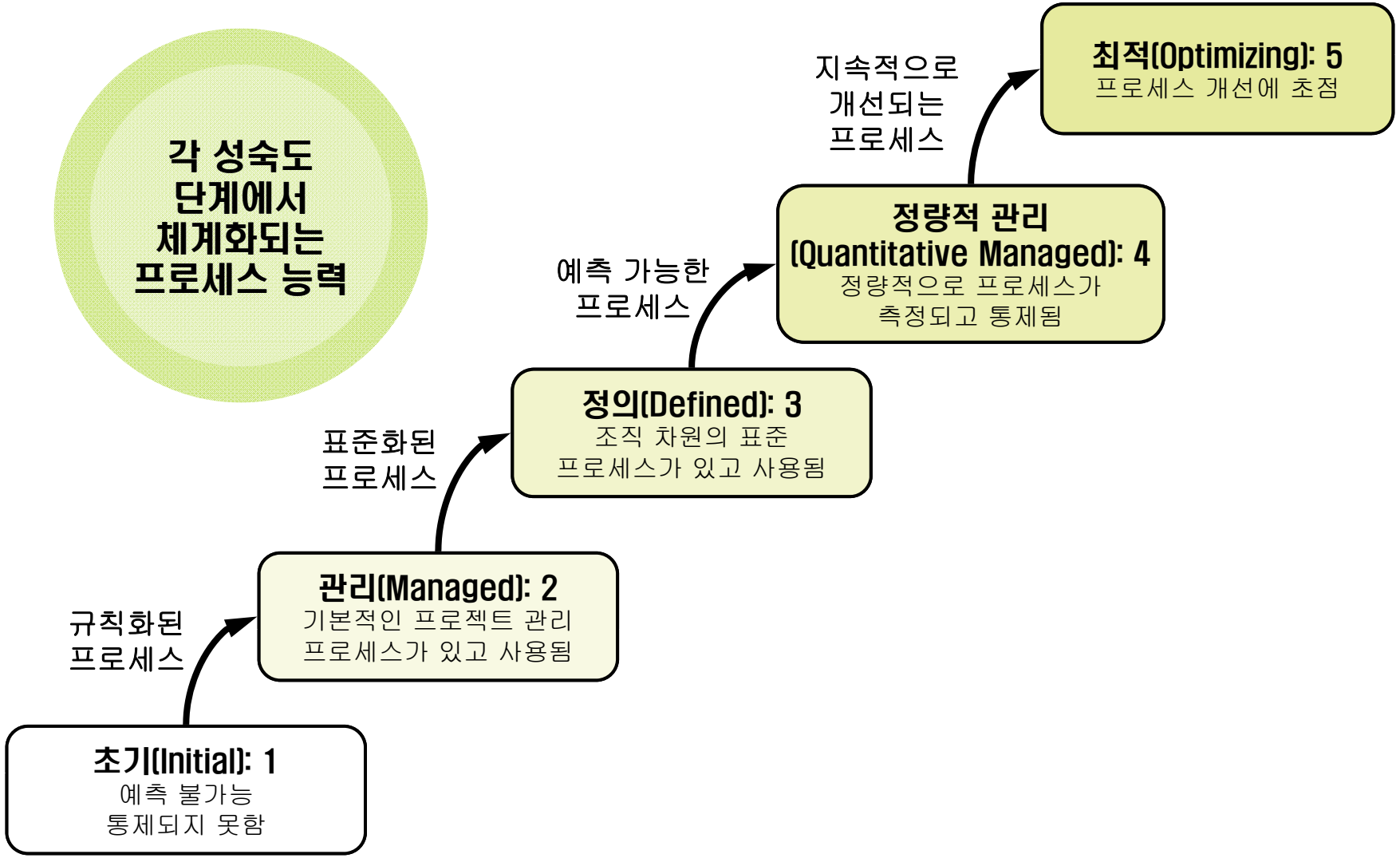
ML = Maturity Level

연속적 표현
(Continuous Representation)



PA = Process Area

CMMI의 5단계 소프트웨어 프로세스 성숙도



CMMI의 단계별 특성(1/3)

- 1단계(Level 1): 초기(Initial)
  - 조직에 정의된 프로세스가 거의 없고 계획 없이 코딩과 시험에 집중
  - 경험 많은 관리자나 뛰어난 개발자에 의해 프로젝트의 성공이 좌우됨
  - 능력은 조직이 아닌 개인의 특성

- 2단계(Level 2): 관리(Managed)
  - 기본적인 프로젝트 관리 프로세스가 설정됨
  - 소프트웨어의 크기, 공수 및 비용, 일정, 컴퓨터 자원, 위험, 기능을 추적할 수 있는 단계
  - 프로젝트의 중간 산출물에 대한 통제가 가능
  - 새로운 프로젝트에 대한 계획과 관리가 이전의 성공한 프로젝트에 근거하여 이루어짐
  - 성공한 프로젝트의 실무 활동을 반복하기 때문에, 유사한 응용 분야에서의 프로젝트의 성공을 반복 가능

CMMI의 단계별 특성(2/3)

- 3단계(Level 3): 정의(Defined)
  - 표준과 일관성 있는 프로세스
  - 조직 전체에 걸쳐 소프트웨어의 개발 및 유지에 관한 표준 프로세스가 문서화되고 통합되는 단계
  - 조직의 소프트웨어 프로세스 활동에 대한 책임이 있는 팀(EPG: Engineering Process Group)이 구성
  - 각 프로젝트는 “조직의 표준 프로세스”를 기반으로 하여, “프로젝트에서 정의된 소프트웨어 프로세스”에 따라

CMMI의 단계별 특성(3/3)

- 4단계(Level 4): 정량적 관리(Quantitatively Managed)
  - 소프트웨어 제품과 프로세스에 대한 계량적인 품질 목표가 설정됨
  - 모든 프로젝트에 대한 중요한 소프트웨어 프로세스 활동에 대한 생산성과 품질이 측정됨
  - 프로세스의 성과의 변동을 수용 가능한 계량적인 범위 내로 최소화하여 제품과 프로세스에 대한 통제 수행

- 5단계(Level 5): 최적화(Optimizing)
  - 조직은 지속적인 프로세스 개선에 몰두함
  - 프로세스에 대해 비용/효과 분석 수행
  - 분석을 통한 가장 좋은 소프트웨어 공학 실무 활동을 활용하는 혁신이 식별되고 전 조직에 보급됨
  - 발견된 결함의 형태가 다시 발생하지 않도록 소프트웨어 프로세스가 평가되고 얻어진 교훈이 전 조직에 확산됨

ISO 12207

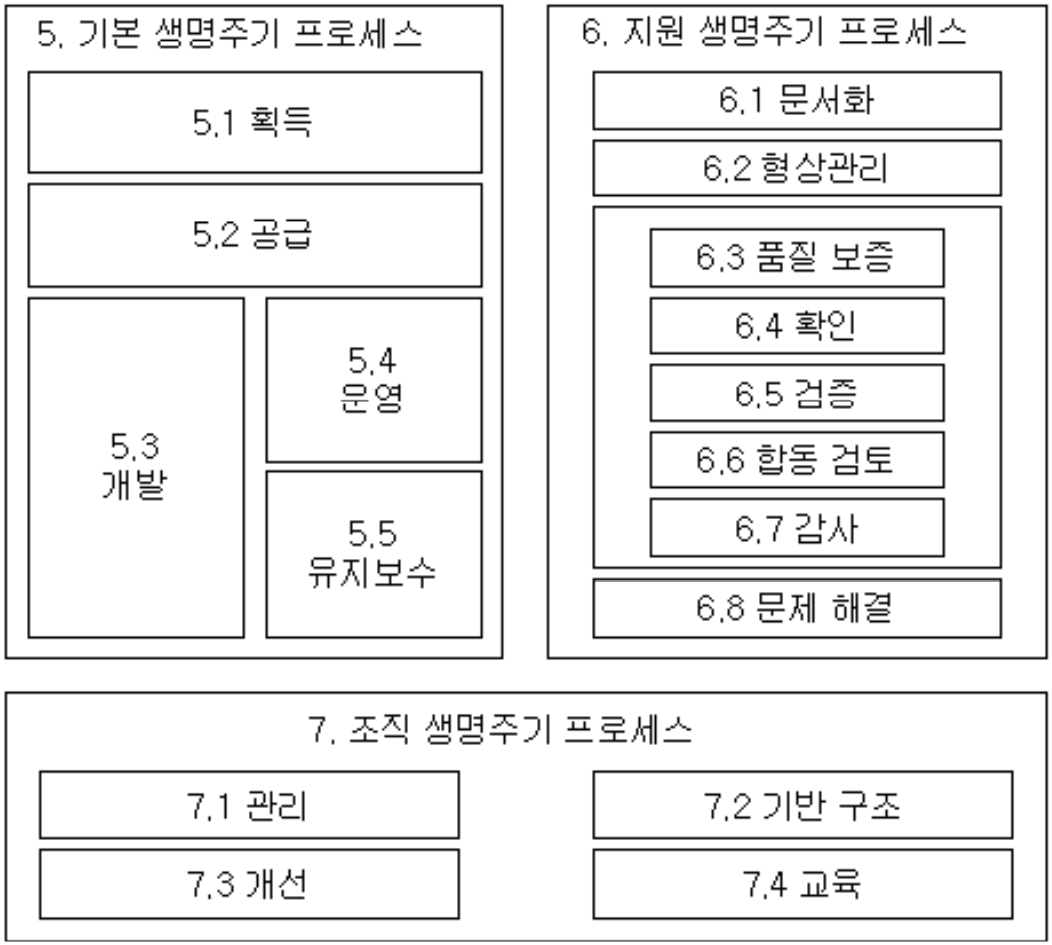
❖ 소개

- 소프트웨어 생명주기 공정 표준
- 1995년 소프트웨어 개발을 위한 일관적이고 체계적인 구조(framework)를 제공하기 위하여 제정

❖ 구성

- 소프트웨어 개발 시 고려해야 할
  - 5개의 기본(primary) 프로세스
  - 8개의 지원(supporting) 프로세스
  - 4개의 조직(organizational) 프로세스

ISO 12207의 구성 (1/2)



ISO 12207의 구성 (2/2)

❖ 구성

- **기본 프로세스(Primary Process)**
  - 소프트웨어 개발 프로세스의 주요 프로세스들
  - 소프트웨어의 획득, 공급, 개발, 운영, 유지보수에 대한 활동을 정의
- **지원 프로세스(Supporting Process)**
  - 기본 프로세스들을 보조해주는 역할을 하는 프로세스
  - 각 기본 프로세스로부터 산출되는 문서, 품질 보증, 감사, 문제해결 등에 대한 활동을 정의
- **조직 프로세스(Organizational Process)**
  - 개발 전 생명주기에 거쳐 전체 프로젝트를 관리하는 역할을 하는 프로세스
  - 프로젝트의 기반구조, 개선, 인력 훈련 등에 대한 활동을 정의

❖ 소프트웨어 개발 기반 표준 지향

- **실제 프로세스 간의 상호 연관이나 프로세스 내 액티비티 및 태스크 간의 상호 연관에 대해서는 자세히 명시하지 않음**

4. 요구사항 개발 및 관리

요구사항이란?

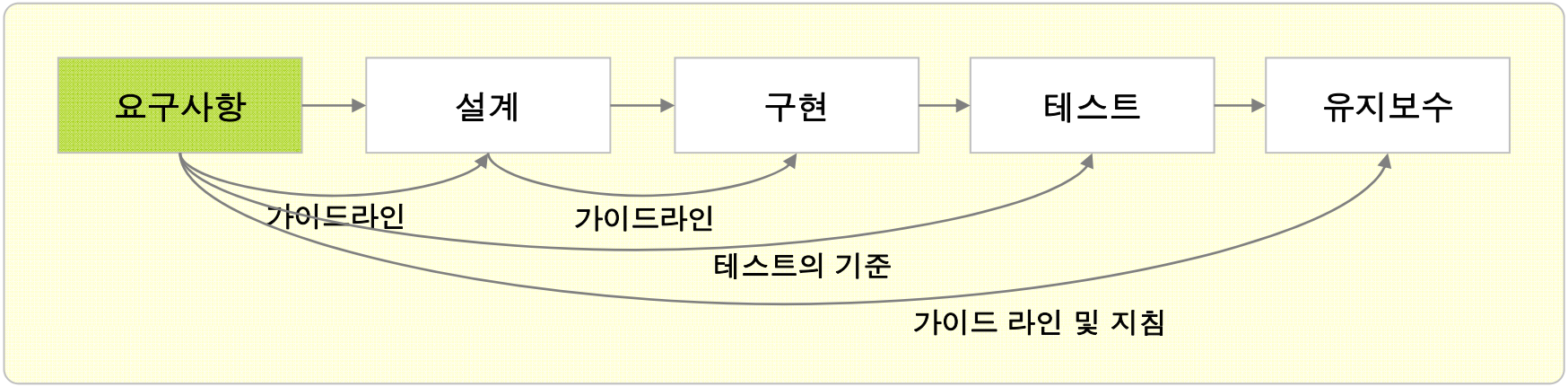
❖ 정의

- 문제의 해결 또는 목적 달성을 위하여 고객에 의해 요구되거나, 표준이나 명세 등을 만족하기 위하여 시스템이 가져야 하는 서비스 또는 제약사항
- 고객이 요구한 사항과 요구하지 않았더라도 당연히 제공되어야 한다고 가정되는 사항들

요구사항의 중요성

❖ 요구사항의 중요성

- 참여자들로 하여금 개발되는 소프트웨어 제품을 전체적으로 파악하도록 하여 의사 소통 시간을 절약하게 해 주는 것
- 상세한 요구사항이 있어야만 산정이 가능하고, 이를 기반으로 계획을 세울 수 있기 때문



요구사항의 분류

❖ 기능적 요구사항(Functional Requirements)

- **수행될 기능과 관련되어 입력과 출력 및 그들 사이의 처리과정**
- **목표로 하는 제품의 구현을 위해 소프트웨어가 가져야 하는 기능적 속성**
  - 예) 워드 프로세서에서 파일 저장 기능, 편집 기능, 보기 기능 등

❖ 비기능적 요구사항(Non-Functional Requirements)

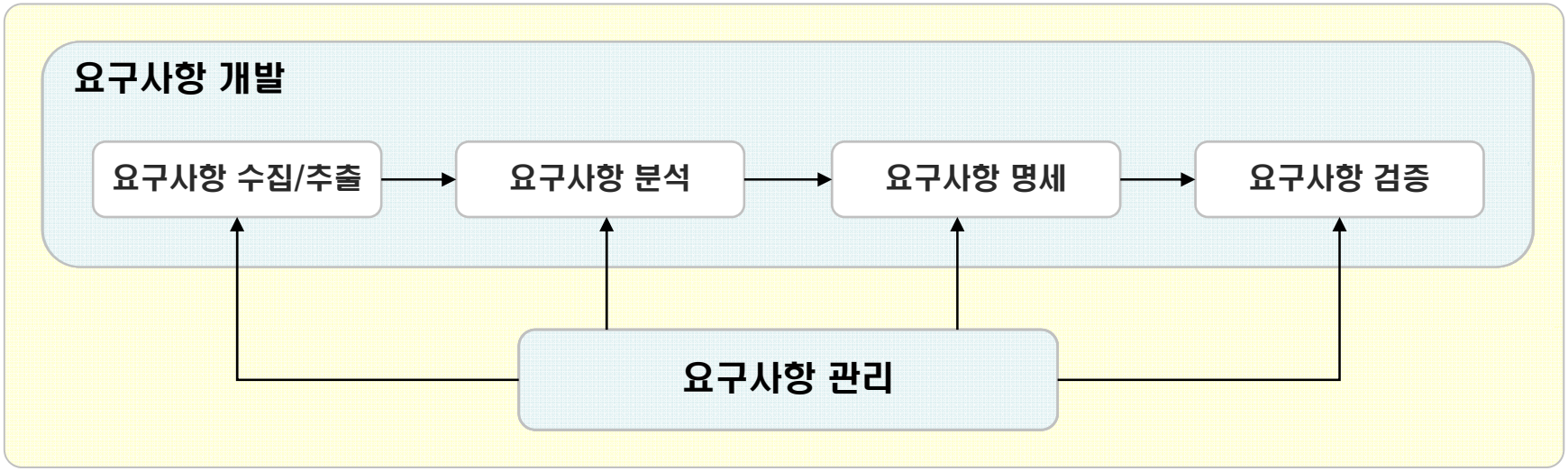
- **제품의 품질 기준 등을 만족시키기 위해 소프트웨어가 가져야 하는 성능, 사용의 용이성, 안전성과 같은 행위적 특성**
- **시스템의 기능에 관련되지 않는 사항을 나타냄**
  - 예) 성능(응답 시간, 처리량), 사용의 용이성, 신뢰도, 보안성, 운용상의 제약, 안전성 등

요구사항 개발

❖ 의미

- 발주자나 고객으로부터 구현될 소프트웨어 제품의 사양을 정확히 도출하여 요구사항을 명세하고, 이를 분석한 결과를 개발자들이 이해할 수 있는 형식으로 기술하는 작업

❖ 요구사항 개발 단계



요구사항 추출 (1/2)

❖ 개요

- 고객이 원하는 요구사항을 수집
- 수집된 요구사항을 통해 개발되어야 하는 시스템에 대한 사용자 요구와 시스템 기능 및 제약사항을 식별하고 이해하는 단계

❖ 중요성

- 고객의 최초 요구사항은 추상적이기 때문에 수주자는 정확한 요구사항을 파악
- 요구사항은 계약 및 최초 산정의 기본이 됨

요구사항 분석 (1/2)

❖ 개요

- 추출된 고객의 요구사항을 분석 기법을 이용하여 식별 가능한 문제들을 도출하고 요구사항을 이해하는 과정
- 참여자들로부터 추상적 요구사항을 명세서 작성 전에 완전하고 일관성 있는 요구사항으로 정리하는 활동

❖ 요구사항 분석의 기준

- 시스템을 계층적이고 구조적으로 표현하여야 한다.
- 외부 사용자와의 인터페이스 및 내부 시스템 구성요소 간의 인터페이스를 정확히 분석하여야 한다.
- 분석단계 이후의 설계와 구현단계에 필요한 정보를 제공하여야 한다.

요구사항 분석 (2/2)

❖ 요구사항 분석 기법의 종류

- **구조적 분석(Structured Analysis)**
  - 시스템의 기능을 중심으로 구조적 분석을 실행
  - 시스템의 기능을 정의하기 위해서 프로세스들을 도출하고, 도출된 프로세스 간의 데이터 흐름을 정의

- **객체지향 분석**
  - 요구사항을 사용자 중심의 시나리오 분석을 통해 유스케이스 모델(Usecase Model)로 구축하는 것
  - 요구사항을 추출하고, 유스케이스의 실체화(Realization)과정을 통해 추출된 요구사항을 분석

요구사항 명세 (1/4)

❖ 의미

- 분석된 요구사항을 명확하고 완전하게 기록하는 것
- 소프트웨어 시스템이 수행하여야 할 모든 기능과 시스템에 관련된 구현상의 제약 조건 및 개발자와 사용자가 합의한 성능에 관한 사항 등을 명세

❖ 최종 결과물

- 요구사항 명세서(SRS: Software Requirement Specification)

요구사항 명세 (3/4)

❖ IEEE-Std-830 명세 표준

1. **소개**(Introduction)
   - 1.1 SRS**의 목적**(Purpose of SRS)
   - 1.2 **산출물의 범위**(Scope of product)
   - 1.3 **정의**,**두문자어**,**약어**(Definitions, acronyms and Abbreviations)
   - 1.4 **참조문서**(References)
   - 1.5 SRS **개요**(Overview of rest of SRS)
2. **일반적인 기술사항**(General Description)
   - 2.1 **제품의 관점**(Product Perspective)
   - 2.2 **제품의 기능**(Product Functions)
   - 2.3 **사용자 특성**(User Characteristics)
   - 2.4 **제약사항**(Constraints)
   - 2.5 **가정 및 의존성**(Assumptions and Dependencies)

3. **상세한 요구사항** (Specific requirements)
   - 3.1 **기능적 요구사항**(Functional requirements)
     - 3.1.1 **기능적 요구사항**1 (Functional requirements 1)
       - 3.1.1.1 **개요**
       - 3.1.1.2 **입력물**
       - 3.1.1.3 **프로세싱**(Processing)
       - 3.1.1.4 **산출물**(Outputs)
       - 3.1.1.5 **수행 요구사항**(Performance requirements)
       - 3.1.1.6 **디자인 제약사항**(Design constraints)
       - 3.1.1.7 **속성**(Attributes)
       - 3.1.1.8 **기타 요구사항**(Other requirements)
       - . . .
   - 3.2 **외부적인 인터페이스 요구사항**(External interface requirements)
     - 3.2.1 **사용자 인터페이스**(User Interface)
     - 3.2.2 **하드웨어 인터페이스**(Hardware interface)
     - 3.2.3 **소프트웨어 인터페이스**(Software interface)
     - 3.2.4 **커뮤니케이션 인터페이스**(Communications interface)

부록(Appendices)

인덱스(Index)

요구사항 검증 (1/5)

❖ 개요

- 사용자 요구가 요구사항 명세서에 올바르게 기술되었는가에 대해 검토하는 활동

❖ 검증 내용

- 요구사항이 사용자나 고객의 목적을 완전하게 기술하는가?
- 요구사항 명세가 문서 표준을 따르고, 설계 단계의 기초로 적합한가?
- 요구사항 명세의 내부적 일치성과 완정성이 있는가?
- 기술된 요구사항이 참여자의 기대에 일치하는가?

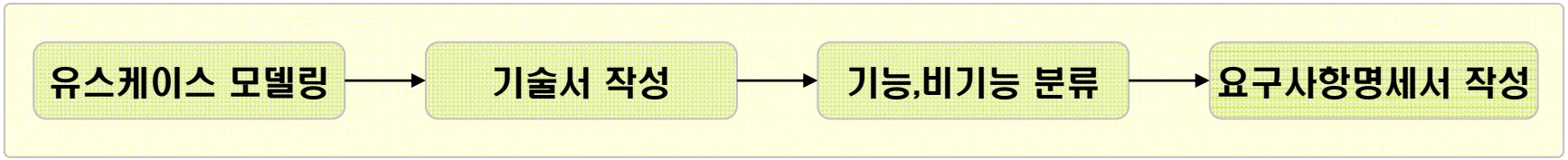
요구사항 분석

❖ 의미

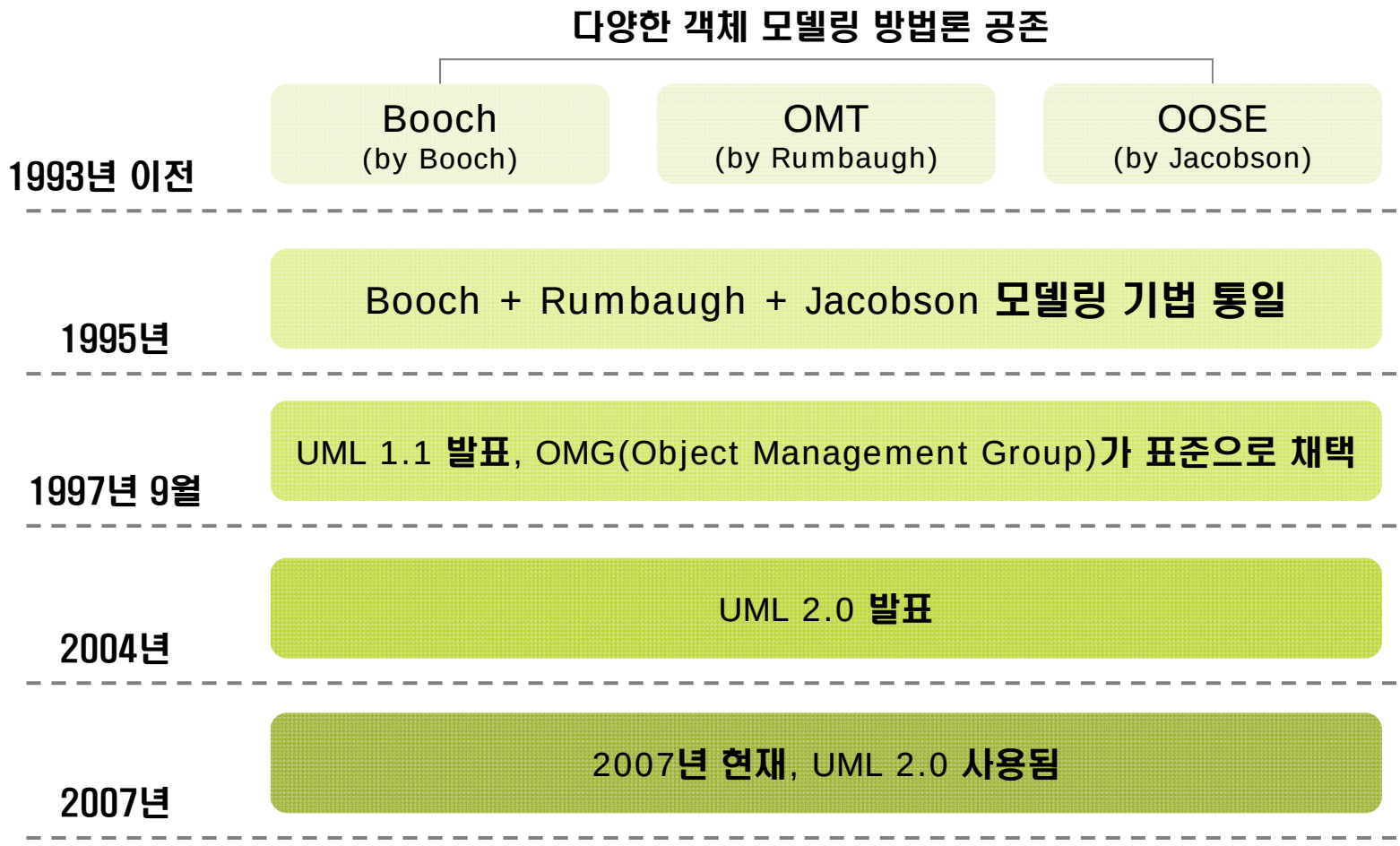
- 요구사항 명세서 작성의 기반을 다지는 작업

❖ 요구사항 분석 방법

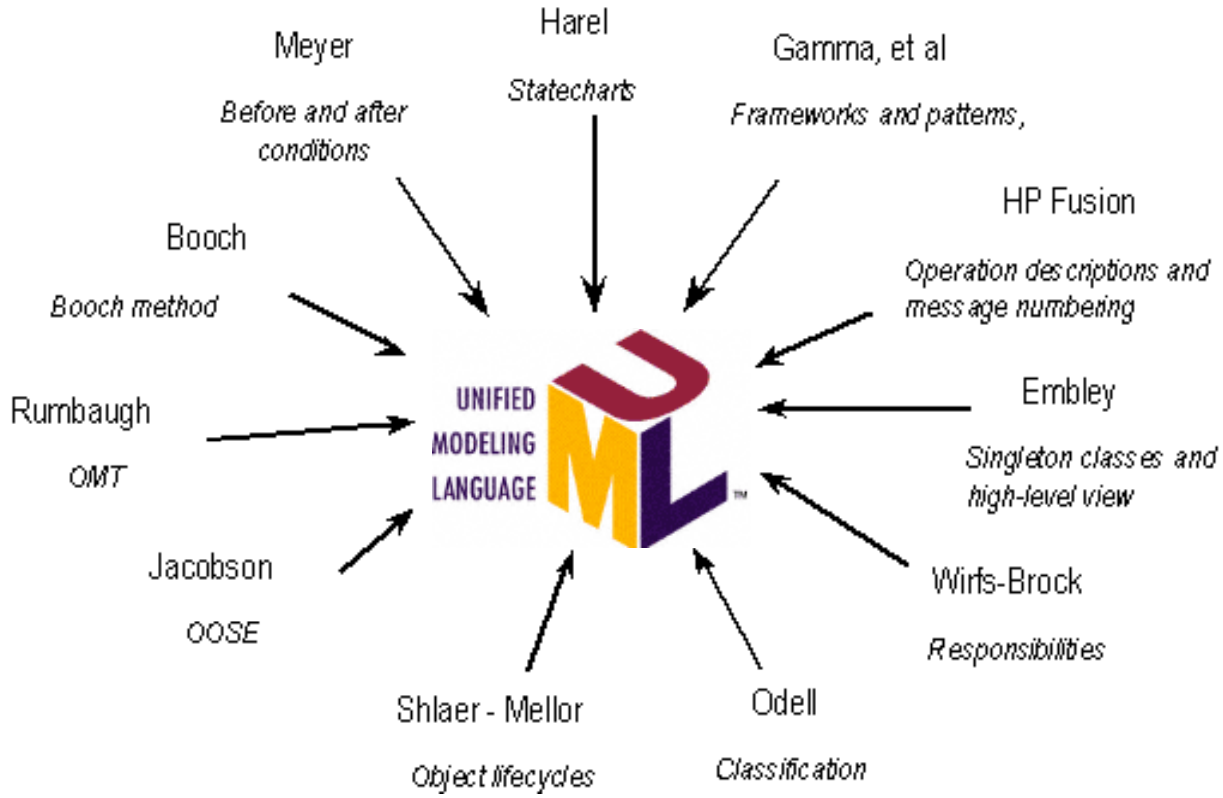
- 객체지향 방법인 유스케이스 기반 분석



UML의 역사

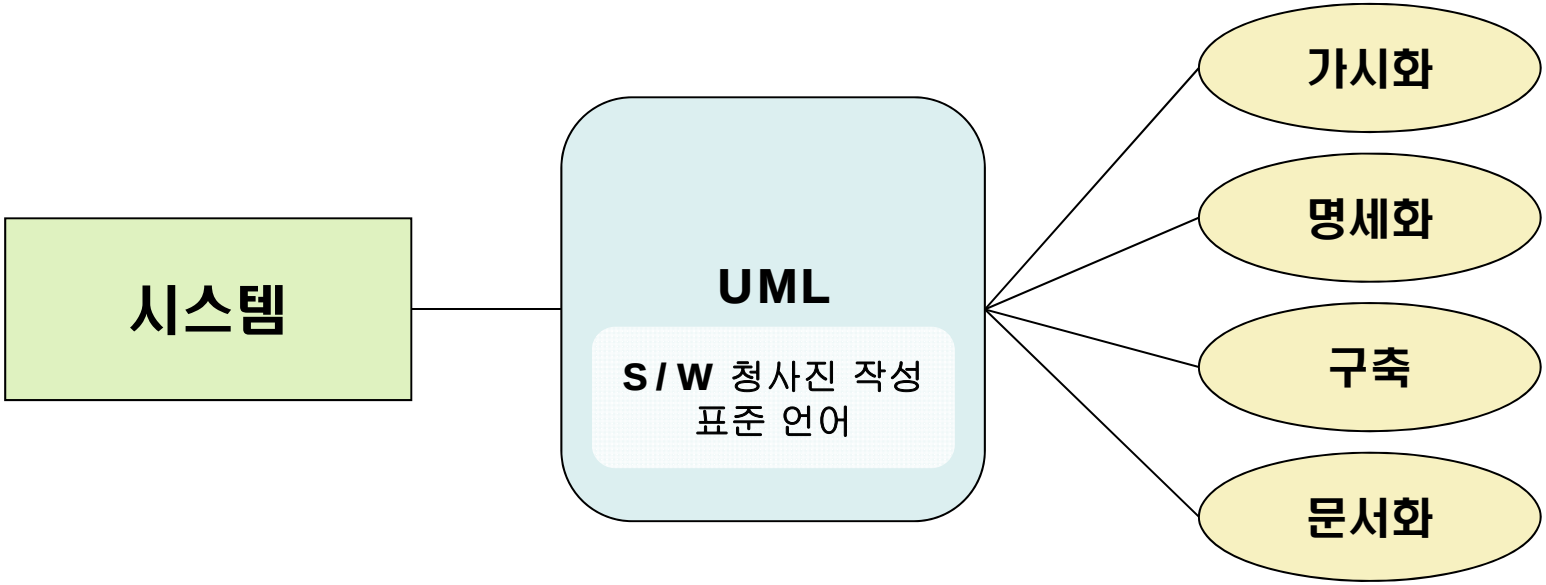

다양한 객체 모델링 방법론 공존
1993년 이전
Booch
(by Booch)
OMT
(by Rumbaugh)
OOSE
(by Jacobson)
1995년
Booch + Rumbaugh + Jacobson 모델링 기법 통일
1997년 9월
UML 1.1 발표, OMG(Object Management Group)가 표준으로 채택
2004년
UML 2.0 발표
2007년
2007년 현재, UML 2.0 사용됨

통합된 표준 모델링 언어, UML

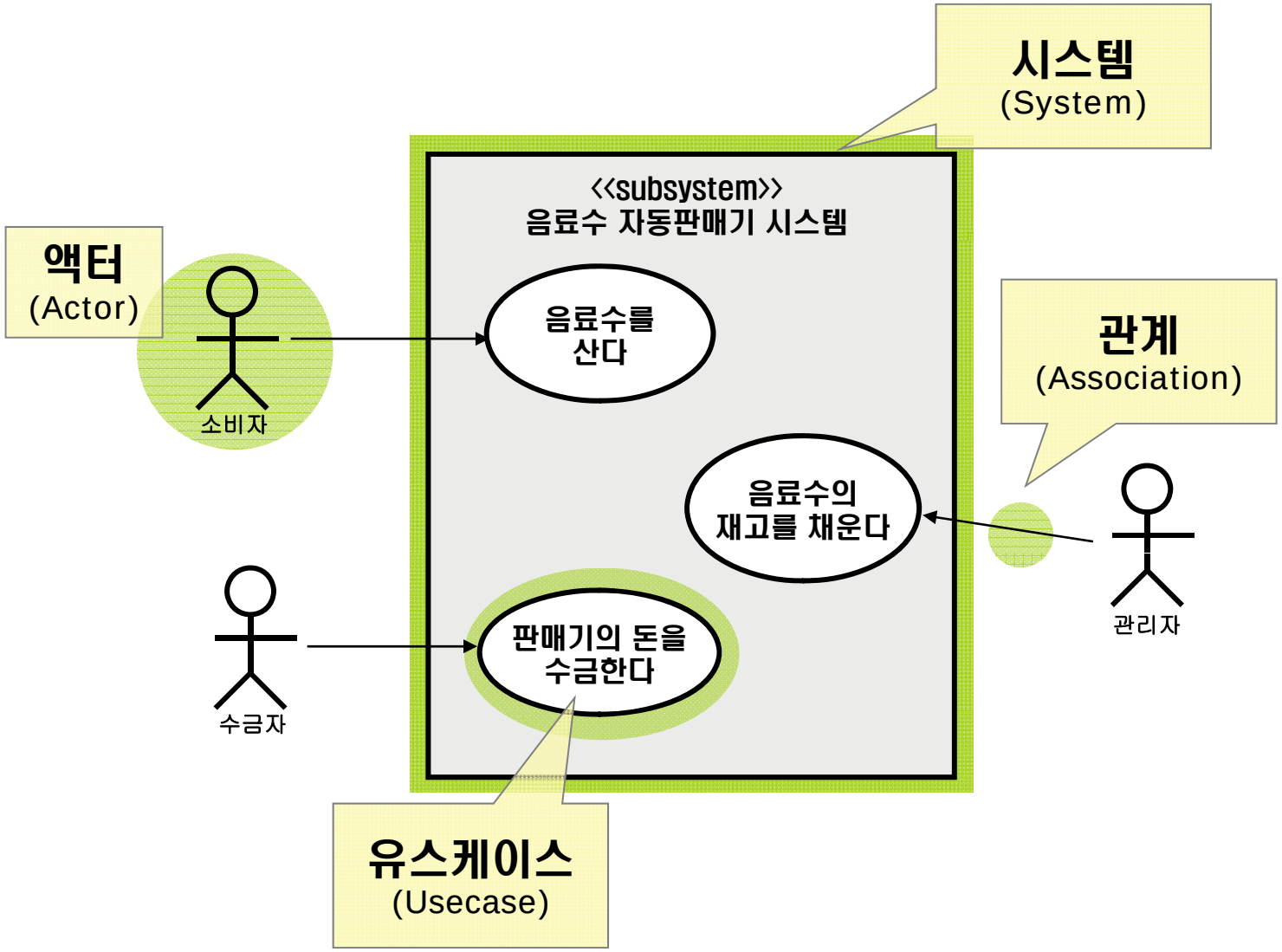


UML의 표기법(notation)만 알고 있다면
프로젝트 이해 관계자간의 의사소통의 불일치를 지적할 수 있다!

시스템 구축 시 UML의 역할



유스케이스 다이어그램의 구성요소



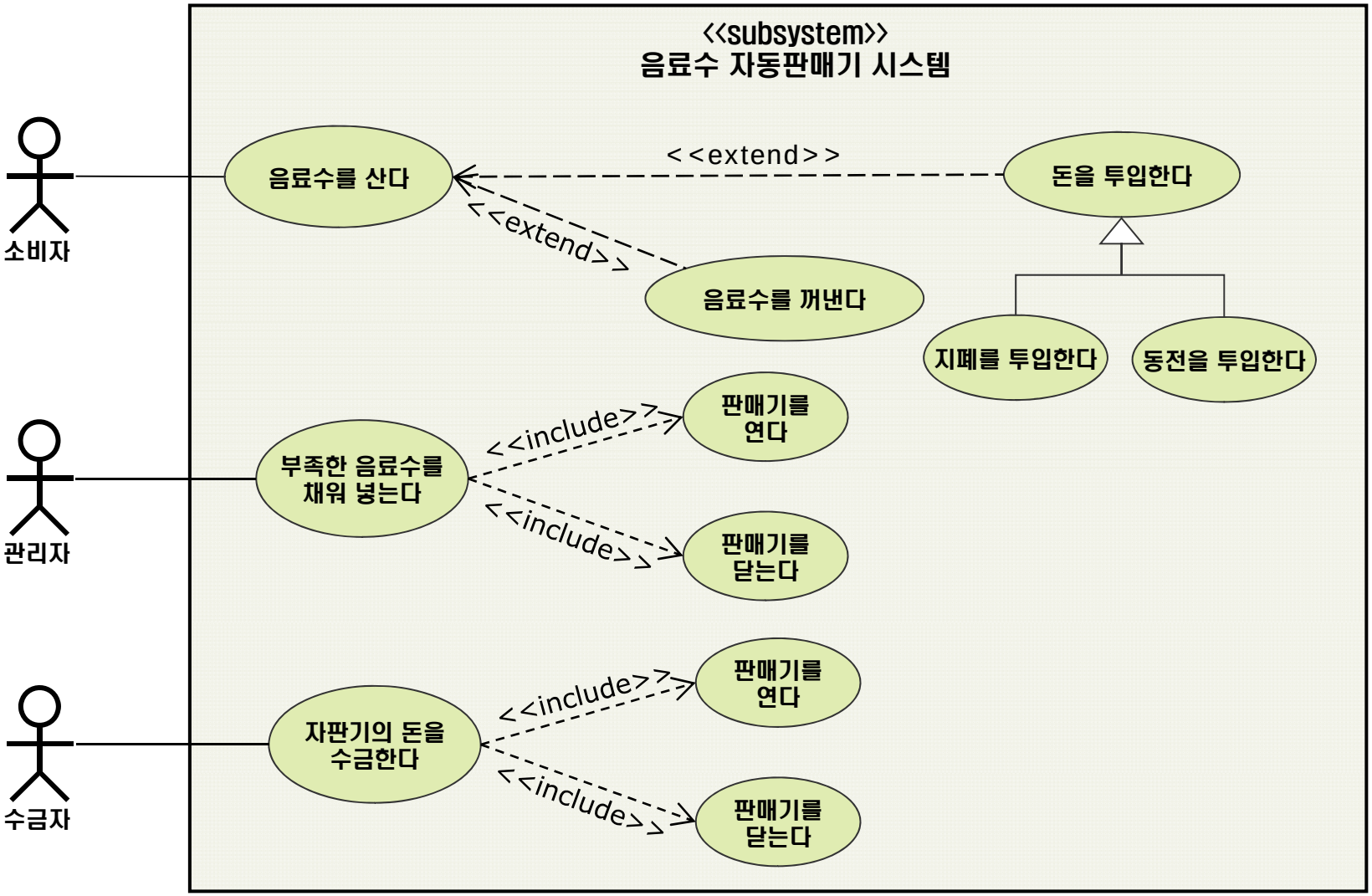
유스케이스 다이어그램의 예 (1/6)

- 예제 요구사항
  - SE사는 K고객으로부터 다음의 요구사항을 전달받았다.
  - 음료수 자동판매기 시스템을 만드시오.

- SE사는 K고객의 요구사항을 Usecase Diagram으로 모델링하기로 한다.

유스케이스 다이어그램의 예 (6/6)

완성된 유스케이스 다이어그램의 예제



유스케이스 기술서 작성 (1/3)

❖ 개요

- **유스케이스 다이어그램을 보완하기 위한 산출물**
- **유스케이스 다이어그램과의 차이**
  - 유스케이스 다이어그램: 유스케이스는 시스템의 기능을 표현하는 것
  - 유스케이스 기술서: 각각의 유스케이스에 대해서 해당 유스케이스가 어떻게 수행되는지를 표현하는 수단

유스케이스 기술서 작성 (2/3)

❖ 유스케이스 기술서 항목

- ❖ 유스케이스 명
- ❖ 액터 명
- ❖ 유스케이스 개요 및 설명
- ❖ 사전 및 사후 조건
- ❖ 작업 흐름
  - 정상흐름(Normal Flow): 해당 유스케이스가 정상적으로 수행되는 흐름을 표현하는 절차
  - 대치 흐름(Alternative Flow): 유스케이스 내의 작업 흐름이 수행되는 중에 특정 시점에서 여러 가지 선택적인 흐름으로 나뉘어질 경우에 발행하는 흐름
  - 예외 흐름(Exceptional Flow): 유스케이스 내의 작업 흐름이 수행되는 중에 발생할 수 있는 예외 상황이나 오류를 표현하는 흐름
- ❖ 시나리오: 각 시나리오는 유스케이스의 특정한 예를 나타냄

유스케이스 기술서 작성 (3/3) 유스케이스 기술서 예제

❖ 유스케이스 기술서 예제

❖ **유스케이스명: 음료수 보충**

❖ **액터명: 관리자**

❖ **유스케이스 개요 및 설명**

- **음료수 관리자는 2주마다 음료수 자동판매기의 부족한 음료수를 보충한다.**

❖ **사전 조건: 마지막으로 음료수를 보충한지 2주가 지났다.**

❖ **작업 흐름**

- **정상흐름**
  - 1. 유스케이스는 2주마다 시작한다.
  - 2. 관리자는 음료수 자동판매기를 연다.
  - 3. 부족한 음료수를 보충한다.
  - 5. 관리자는 음료수 자동판매기를 닫고 유스케이스는 종료된다.

❖ **사후 조건: 음료수 자동판매기에 부족한 음료수가 보충되었다.**

5. 프로젝트 계획 및 통제

프로젝트 계획서

❖ 의미

- **프로젝트 관리자 뿐만 아니라 프로젝트 참여자 모두가 프로젝트를 진행해 가면서 참조하는 프로젝트의 중심이 되는 문서**

❖ 작성 순서

- **프로젝트 관리자는,**
  - 프로젝트 태스크 파악
  - 각 태스크를 수행하기 위해 필요한 노력 예측
  - 인적 자원 및 기타 자원을 각 태스크에 할당
  - 일정 계획 작성
- **프로젝트 참여자의 검토를 거쳐 합의 하에 프로젝트 채택함**

IEEE 1058.1-1987 프로젝트 계획서 양식

1개요
- 1.1 프로젝트 개요
- 1.2 프로젝트 산출물
- 1.3 계획서의 변경기록
- 1.4 참고문헌
- 1.5 정의와 약어

2프로젝트 조직
- 2.1 프로세스 모델
- 2.2 조직 구조
- 2.3 조직의 범위와 인터페이스
- 2.4 프로젝트 책임

3관리적 프로세스
- 3.1 관리적 목적과 우선순위
- 3.2 가정과 제한
- 3.3 위험관리
- 3.4 통제 메커니즘
- 3.5 인력

4기술적 프로세스
- 4.1 방법론 도구
- 4.2 소프트웨어 문서화
- 4.3 지원기능

5작업, 스케줄, 예산
- 5.1 작업
- 5.2 작업간 의존관계
- 5.3 자원요구
- 5.4 예산 및 자원할당
- 5.5 스케줄

프로젝트 계획서의 역할 및 중요성

- 프로젝트 진행 과정의 주기적 통제의 기본
  - 주간, 월간 회의를 통해 점검

- 프로젝트가 크고 참여자가 많을수록 잘 짜여진 프로젝트 계획서가 중요함
  - 프로젝트 계획서가 현실적으로 작성되어 전체 프로젝트 진행상황 파악에 크게 문제가 되지 않아야 함

스케줄링이란?

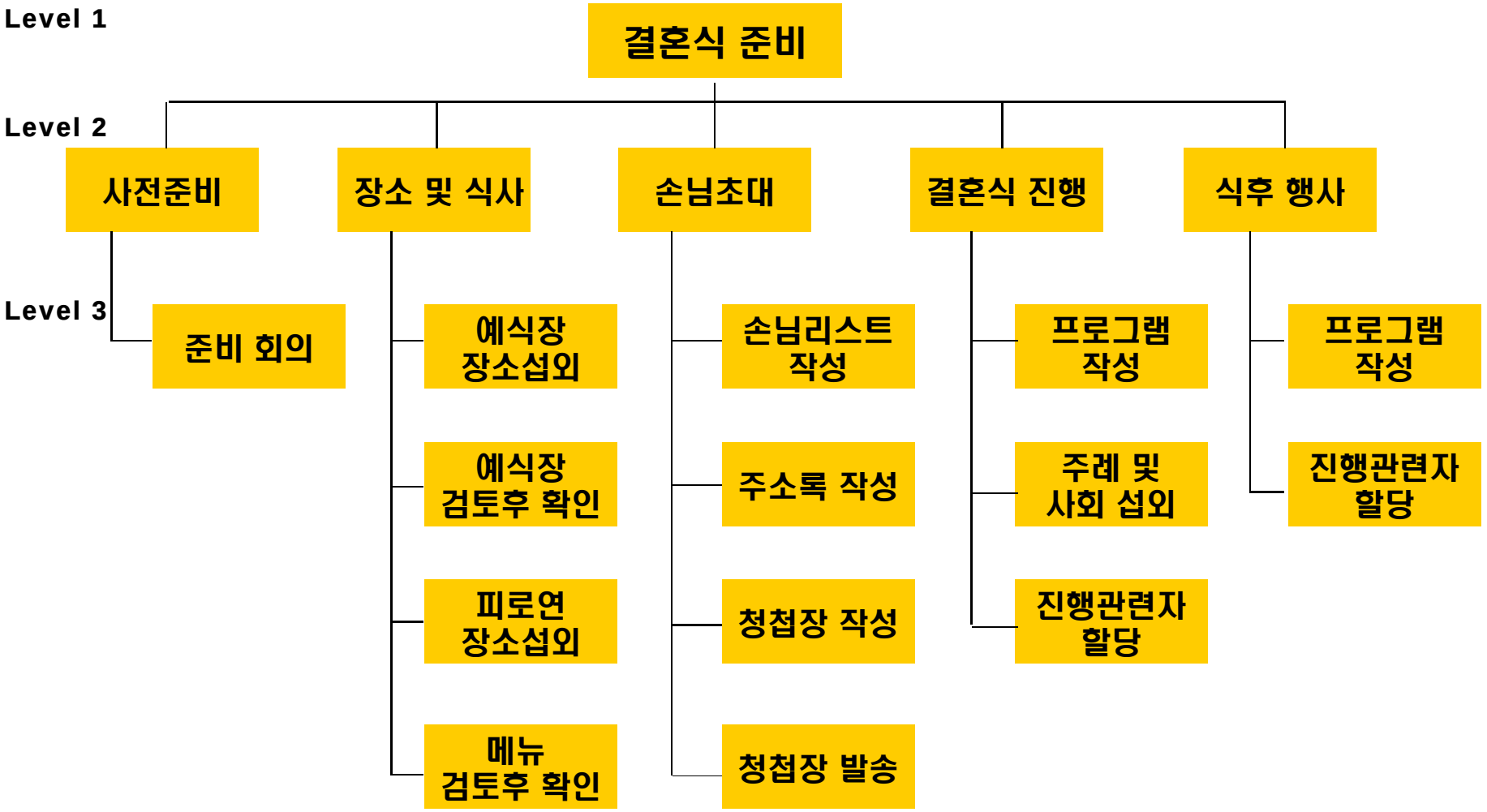
- **의미**
  - **프로젝트의 완성을 위해 수행되어야 할 작업을 나열한 후 연관 관계와 순서에 따라 기간 별로 나타내는 것**

- **스케줄링 방식**
  - **WBS(Work Breakdown Structure)**
    - 프로젝트 중 수행되어야 하는 작업들을 파악

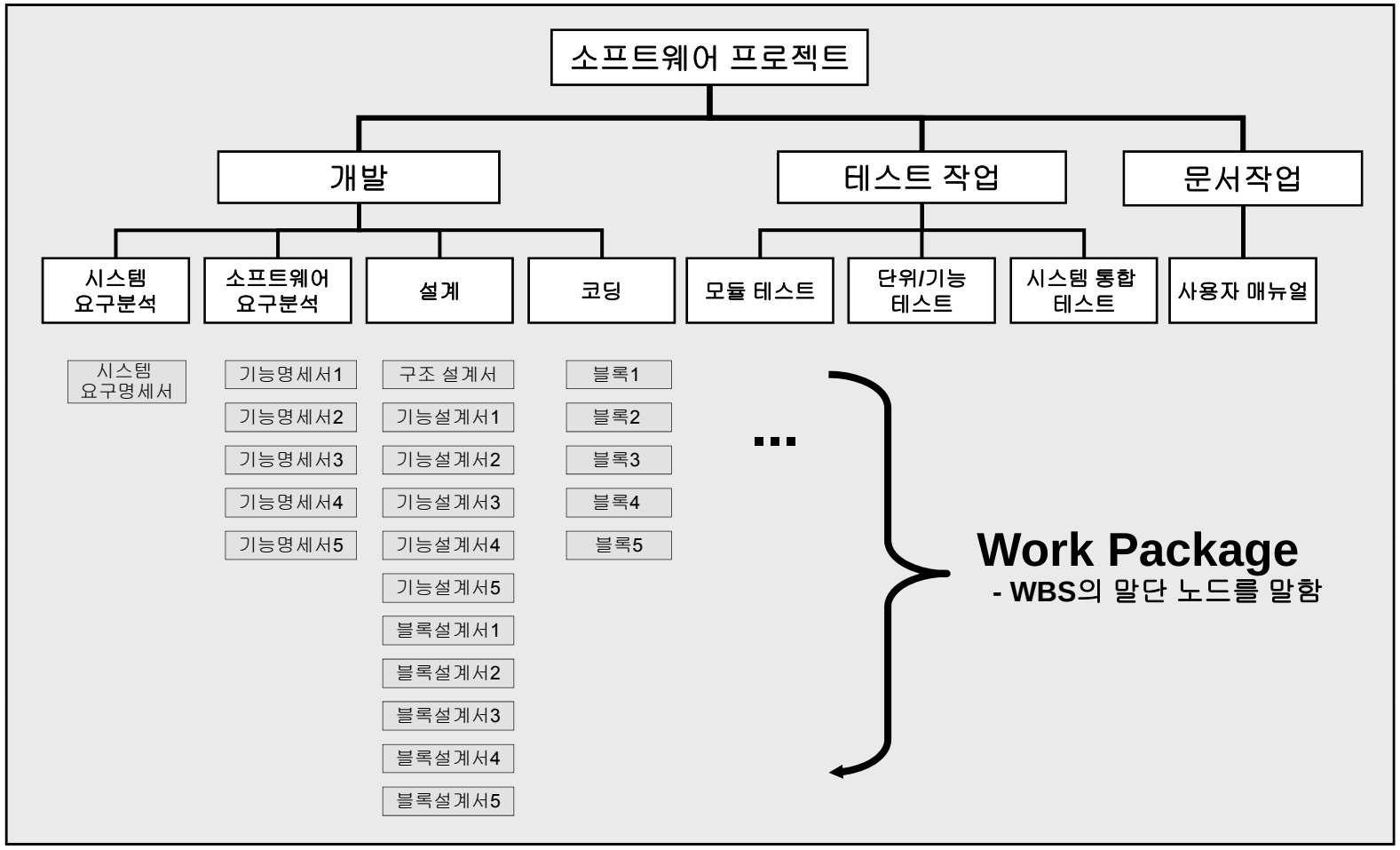
WBS

- WBS (Work Breakdown Structure)
  - 프로젝트를 톱 다운(Top Down) 방식으로 세분화하여 프로젝트의 단위 작업에 대해 파악하는 기법

WBS 작성 예제 – 결혼식 준비



폭포수 생명 주기 기반의 WBS 예제



산정

- 개념
  - 프로젝트 수행에 필요한 규모(Size), 공수(Effort), 비용(Cost) 등을 정량적으로 예측하는 것

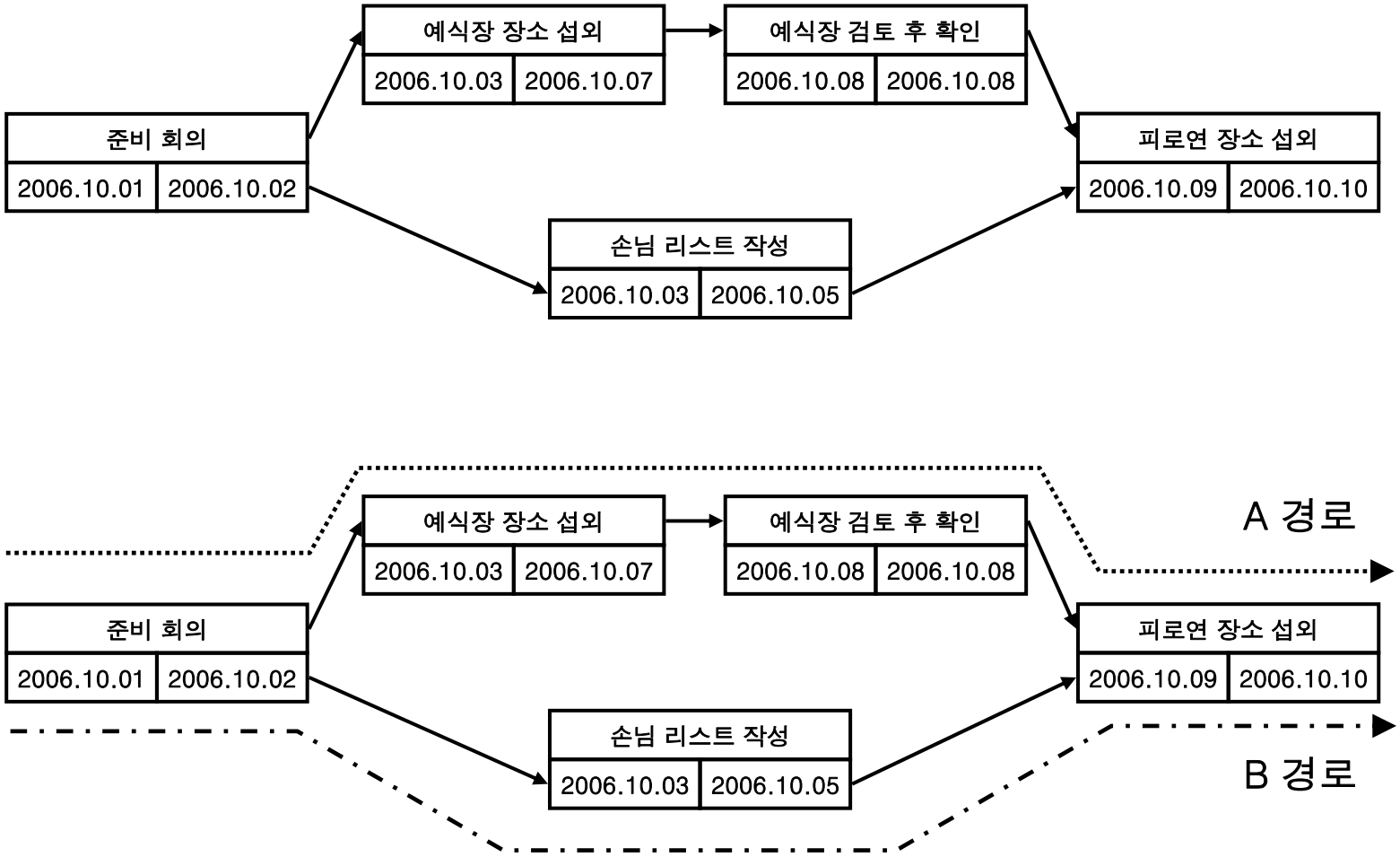
산정의 방법

- 경험적 방법
  - 프로젝트의 수량을 예측하기 위해서 노력과 시간에 대한 수식을 경험적으로 유도한 것
  - 예: 델파이 기법
- 크기 중심 방법
  - LOC(Line of Code : 프로그램 코드 라인 수)로 측정
  - 예: LOC, COCOMO
- 기능 중심 방법
  - 사용자 중심의 기능의 크기로 측정
  - 예: 기능점수(Function Point)로 측정

일정 계획

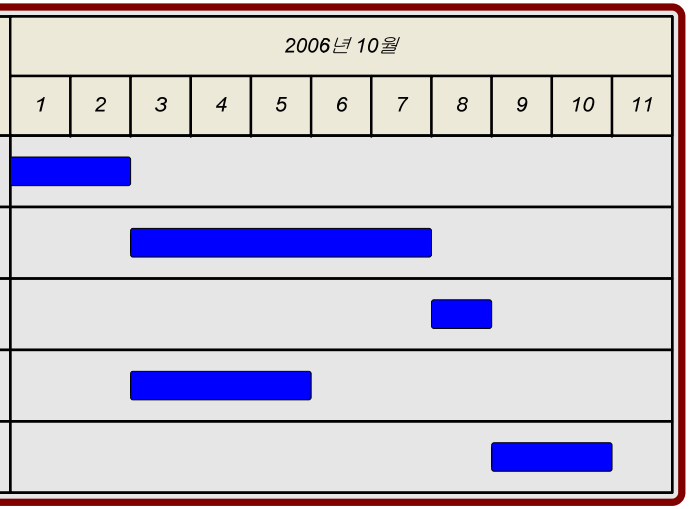
- **의미**
  - **WBS를 이용하여 구성한 하위 작업들을 산정을 통해 나온 기간이나 비용에 맞도록 계획하는 활동**
- **표현 방법**
  - **차트를 사용**
  - **장점**
    - 차트 자체의 비주얼적 특징 및 역할 할당
    - 병렬 작업 구성 등 일정 구성에 도움을 받음
  - **종류**
    - 퍼트(PERT) 차트
    - 간트(Gantt) 차트

퍼트 차트의 예제 – 결혼식 준비



간트 차트의 예 – 결혼식 준비

ID	작업 이름	시작	완료	기간
1	준비회의	2006-10-01	2006-10-02	2d
2	예식장 장소섭외	2006-10-03	2006-10-07	5d
3	예식장 겁토후 확인	2006-10-08	2006-10-08	1d
4	손님 리스트 작성	2006-10-03	2006-10-05	3d
5	피로연 장소 섭외	2006-10-09	2006-10-10	2d



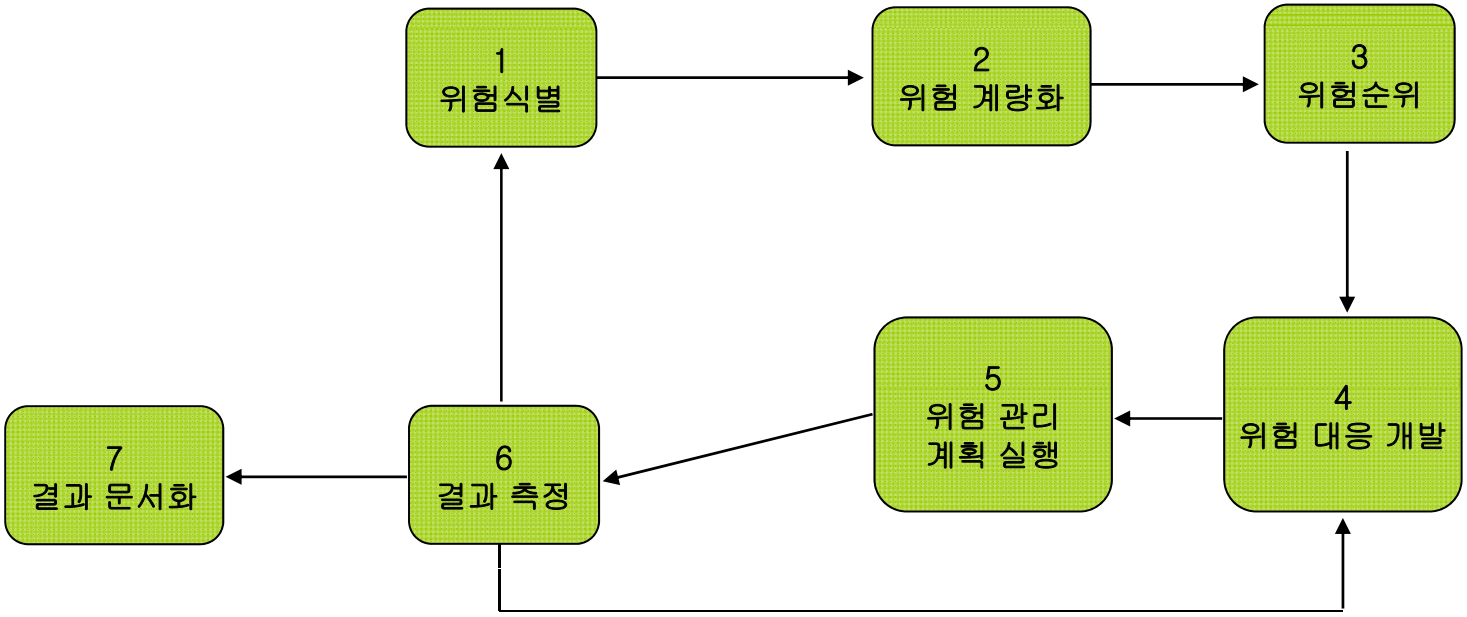
위험 관리란?

- 의미
  - 프로젝트의 위험 요소들을 인식하고 그 영향을 분석, 관리하는 활동

- 위험의 의미
  - 프로젝트의 결과가 잘못되는 데에 심각한 영향을 주는 요소
  - 예) 프로젝트 기간 중 개발 인력의 부족, 갑작스런 요구사항의 변경, 예정되었던 기술의 실패 등

위험 관리 과정

❖ 위험 관리 과정



6. 설계 및 구현

설계란?

- 정의
  - 설계는 개발될 제품에 대한 의미 있는 공학적 표현
  - 설계는 고객의 요구사항으로 추적 가능해야 하며, 동시에 좋은 설계라는 범주에 들도록 품질에 대해서도 검증되어야 한다
    [IEEE-Std-610]

- 소프트웨어의 설계(design)
  - 본격적인 프로그램의 구현에 들어가기 전에 소프트웨어를 구성하는 뼈대를 정의해 구현의 기반을 만드는 것
  - 종류
    - 상위 설계(High-Level Design)
    - 하위 설계(Low-Level Design)

상위 설계와 하위 설계

❖ 상위 설계(High-Level Design)

– 의미

- 아키텍처 설계(Architecture Design), 예비 설계(Preliminary Design)라고 함
- 시스템 수준에서의 소프트웨어 구성 컴포넌트들 간의 관계로 구성된 시스템의 전체적인 구조
- 시스템 구조도(Structure Chart), 외부 파일 및 DB 설계도(레코드 레이아웃, ERD), 화면 및 출력물 레이아웃 등이 포함됨

❖ 하위 설계(Low-Level Design)

– 의미

- 모듈 설계(Module Design), 상세 설계 (Detail Design)이라고 함
- 시스템의 각 구성 요소들의 내부 구조, 동적 행위 등을 결정
- 각 구성 요소의 제어와 데이터들간의 연결에 대한 구체적인 정의를 하는 것

– 하위 설계 방법

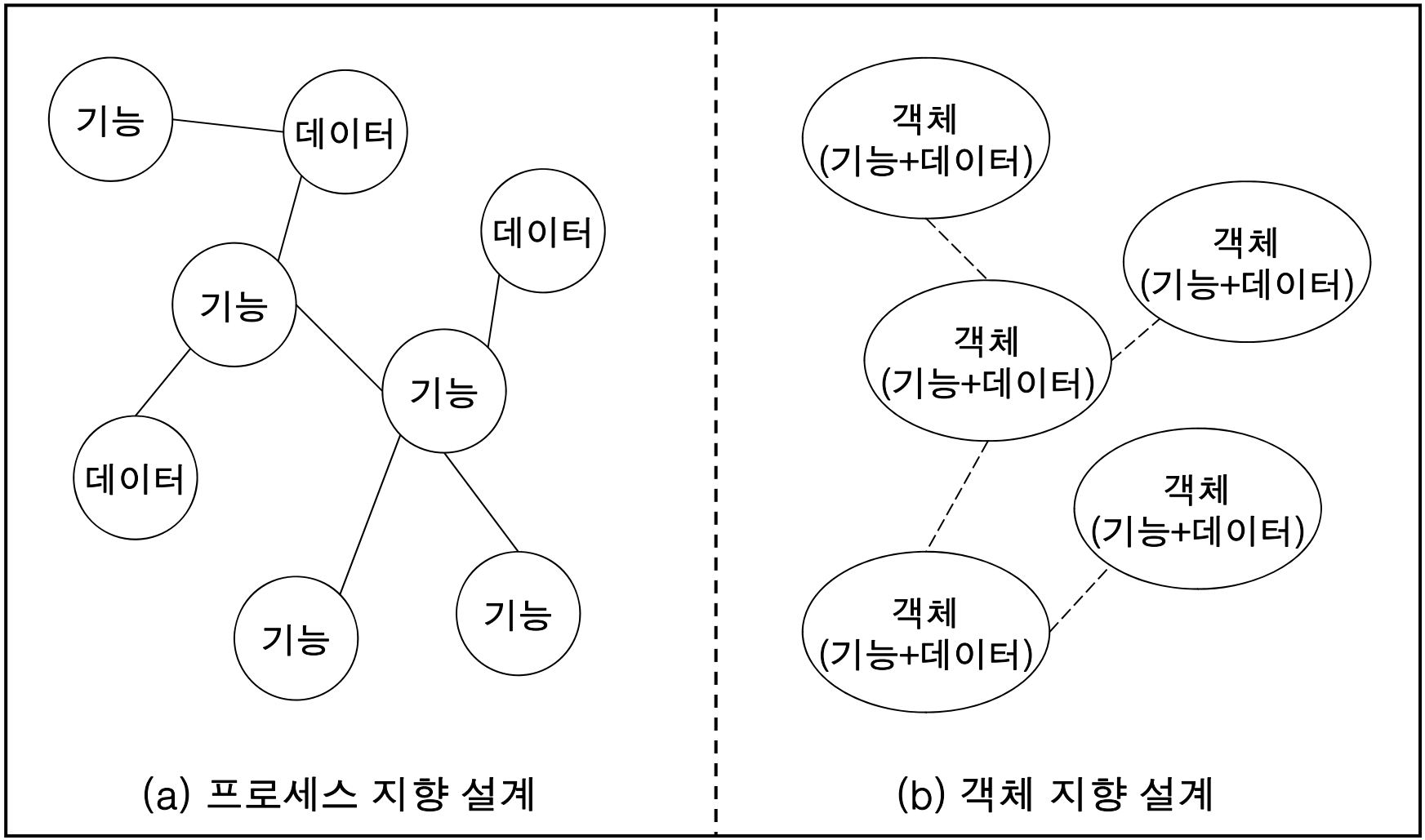
- 절차기반(Procedure-Oriented), 자료위주(Data-Oriented), 객체지향(Object-Oriented) 설계 방법

설계 방식

- 프로세스 지향 설계(Process Oriented Design)
  - 업무의 처리절차를 중심으로 설계의 구성 요소들을 구분
  - 어떠한 절차를 거쳐서 작업을 수행하는가, 어떠한 입출력 자료를 생성하는가에 초점
  - 시스템은 “기능과 데이터”들이 노드를 이루고 이들의 관계가 링크를 형성하는 그래프

- 객체지향 설계(Object Oriented Design)
  - 시스템의 실제 객체 요소를 중심으로 설계
  - 자료구조와 그에 대한 연산을 묶어서 구성되는 객체들을 정의하고 이들이 상호 작용의 기본이 되도록 설계
  - 객체들이 노드를 이루고 이들간의 관계가 링크를 형성하는 그래프

시스템을 해석하는 관점의 차이



설계 원리

- 추상화(Abstraction)
- 단계적 분해(Stepwise refinement)
- 모듈화(Modularization)

추상화(Abstraction)

- 의미
  - 자세한 구현에 전에, 상위 레벨에서의 제품의 구현을 먼저 생각해보는 것
- 단계
  - 상위 레벨에서 설계를 생각해본 후 점차 구체적인 단계로 옮겨가는 것
- 종류
  - 과정 추상화(Procedure Abstraction)
  - 데이터 추상화(Data Abstraction)
  - 제어 추상화(Control Abstraction)

추상화의 종류 (1/2)

- **과정 추상화**
  - 수행 과정의 자세한 단계를 고려하지 않고, 상위 수준에서 수행 흐름만 먼저 설계
- **데이터 추상화**
  - 데이터 구조를 대표할 수 있는 표현으로 대체하는 것
  - 예) 날짜 구조를 단순히 "날짜" 로 추상화 하는 것
- **제어 추상화**
  - 3-A와 3-B를 "3. 윤년 여부에 따라 요일 계산을 수행한다."로 추상화 하는 것

단계적 분해(Stepwise Refinement)

- 의미
  - Niklaus Wirth에 의해 제안됨
  - 문제를 상위 개념부터 더 구체적인 단계로 분할하는 하향식 기법의 원리
  - 모듈에 대한 구체 설계를 할 때 사용

- 과정
  - 문제를 하위 수준의 독립된 단위로 나눈다.
  - 구분된 문제의 자세한 내용은 가능한 한 뒤로 미룬다.
  - 점증적으로 구체화 작업을 계속한다.

모듈화

- 모듈의 의미
  - 수행 가능 명령어, 자료구조 또는 다른 모듈을 포함하고 있는 독립 단위
- 특성
  - 이름을 가지며
  - 독립적으로 컴파일 되고
  - 다른 모듈을 사용할 수 있고
  - 다른 프로그램에서 사용될 수 있다
- 모듈의 예
  - 완전한 독립 프로그램, 라이브러리 함수, 그래픽 함수 등
- 모듈의 크기
  - 되도록 쉽게 이해될 수 있도록 가능한 한 작아야 함
  - 너무 작은 모듈로 나눠지지 않도록 함

정보 은닉(Information Hiding)

- 의미
  - 각 모듈 내부 내용에 대해서는 비밀로 묶어두고, 인터페이스를 통해서만 메시지를 전달 할 수 있도록 하는 개념
  - 설계상의 결정 사항들이 각 모듈 안에 감추어져 다른 모듈이 접근하거나 변경하지 못하도록 함

- 장점
  - 모듈의 구현을 독립적으로 맡길 수 있음
  - 설계 과정에서 하나의 모듈이 변경되더라도 설계에 영향을 주지 않음

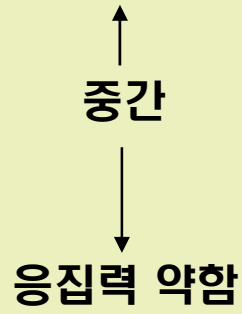
모듈의 응집력 (1/2)

❖ 모듈의 응집력이란?

- 모듈을 이루는 각 요소들의 서로 관련되어 있는 정도
- 강력한 응집력을 갖는 모듈을 만드는 것이 모듈 설계의 목표

❖ Myers의 응집력 정도 구분

1. 기능적 응집(Functional cohesion)
2. 교환적 응집(Communication cohesion)
3. 절차적 응집(Procedural cohesion)
4. 시간적 응집(Temporal cohesion)
5. 논리적 응집(Logical cohesion)
6. 우연적 응집(Coincidental cohesion)



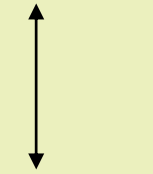
모듈의 결합도 (1/2)

❖ 의미

- 모듈간에 연결되어 상호 의존하는 정도
- 낮은 결합도를 갖는 모듈(Loosely coupled)을 만드는 것이 모듈 설계의 목표

❖ 모듈간의 의존도

자료 결합(Data coupling)
구조 결합(Stamp coupling)
제어 결합(Control coupling)
공통 결합(Common coupling)
내용 결합(Content coupling)



객체지향

❖ 객체지향의 등장 배경

- 기존의 구조적 기법으로 유지보수가 어렵다는 단점을 극복하기 위해 등장

❖ 객체란?

- 특성(Attribute)와 행위(Behavior)를 가지고 있는 인지할 수 있는 개체(Entity)
  - 특성
    - ➢ 해당 객체에 저장되어 있는 데이터
  - 행위
    - ➢ 객체가 할 수 있는 일, 객체의 상태가 변하게 하는 원인을 제공
- 다른 객체와 구별할 수 있는 정체성(identity)을 가짐
  - 정체성
    - ➢ 해당 객체를 다른 개체와 구별 할 수 있는 식별 값

❖ 객체의 예: 차

- 특성: 검정색 차체, 6기통 엔진, 자동 변속기, 4개의 바퀴 등
- 행위: 출발하다, 정지하다, 가속하다, 감속하다 등
- 정체정: 차량 번호

클래스와 객체

❖ 클래스(Class)

- 여러 객체들을 위한 대표적 구조
- 객체들이 내부적으로 어떻게 구성되어 있는지 설명
- 객체의 특성은 클래스의 변수로, 행위는 메소드로 표현됨

❖ 객체(Object)

- 클래스의 실례 또는 실체(Instance)라고도 부름
  - 객체가 클래스에서 정의하는 변수, 메소드를 그대로 가지면서 메모리에 할당되기 때문
- 각 객체 내부의 변수 이름은 같지만 서로 독립적임

```
public class Car
{
    ...
    String carName; //차 이름
    Int carNumber; //차 번호
    ...
}
Car myCar = new Car();
```

클래스

객체

객체지향 방법의 특징

- 절차를 강조하는 구조적 방법
  - 데이터를 소홀히 하게 됨

- 객체지향 방법
  - 시스템을 구성하는 요소들은 객체로,
  - 시스템 개발의 복잡한 문제들을 캡슐화(Encapsulation), 상속(Inheritance), 다형성(Polymorphism) 개념으로 해결하려 함

캡슐화(Encapsulation) (1/2)

- 의미
  - 소프트웨어 모듈인 객체의 내부에 가진 상세한 정보와 처리 방식을 외부로부터 감추는 것
  - 객체의 추상화를 통해 독립성을 보장해 주는 개념

캡슐화(Encapsulation) (2/2)

❖ 캡슐화의 예

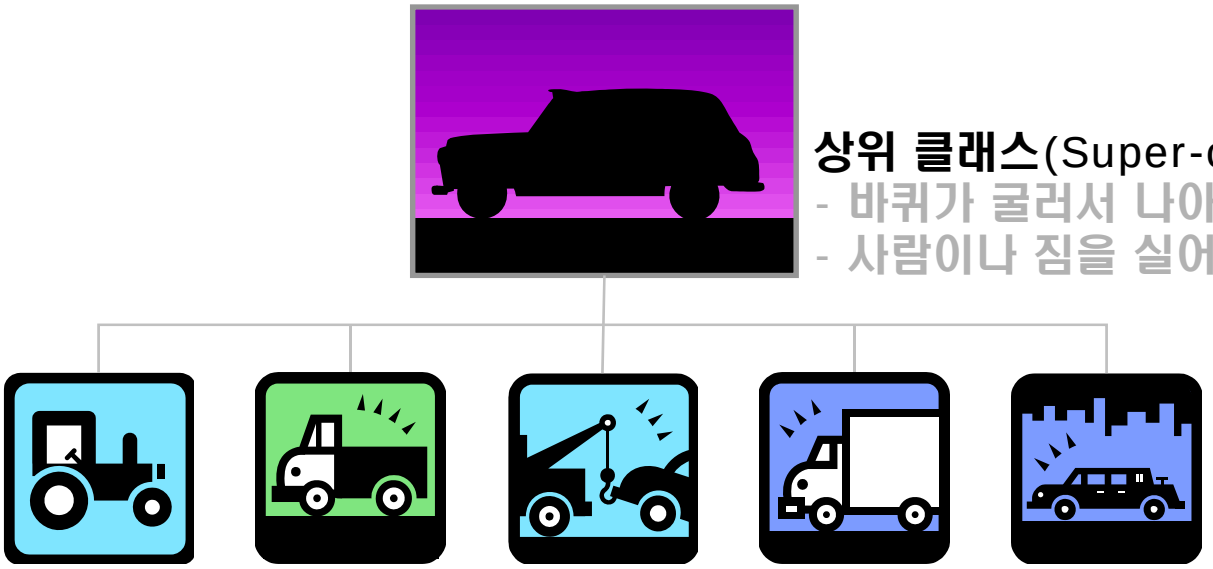
```
class Car
{
    ...
      private String carName; //차 이름
      private Int carNumber; //차 번호

      public String getCarName() // 차 이름 반환
      {
            return carName;
      }

      public Int getCarNumber() // 차 번호 반환
      {
            return carNumber;
      }
    ...
}
```

private 이용
외부에 감춤

public 이용
외부에 공개

상속(Inheritance) (1/2)

- 의미
  - 다른 클래스의 속성을 물려받아 내 것처럼 쓰는 것

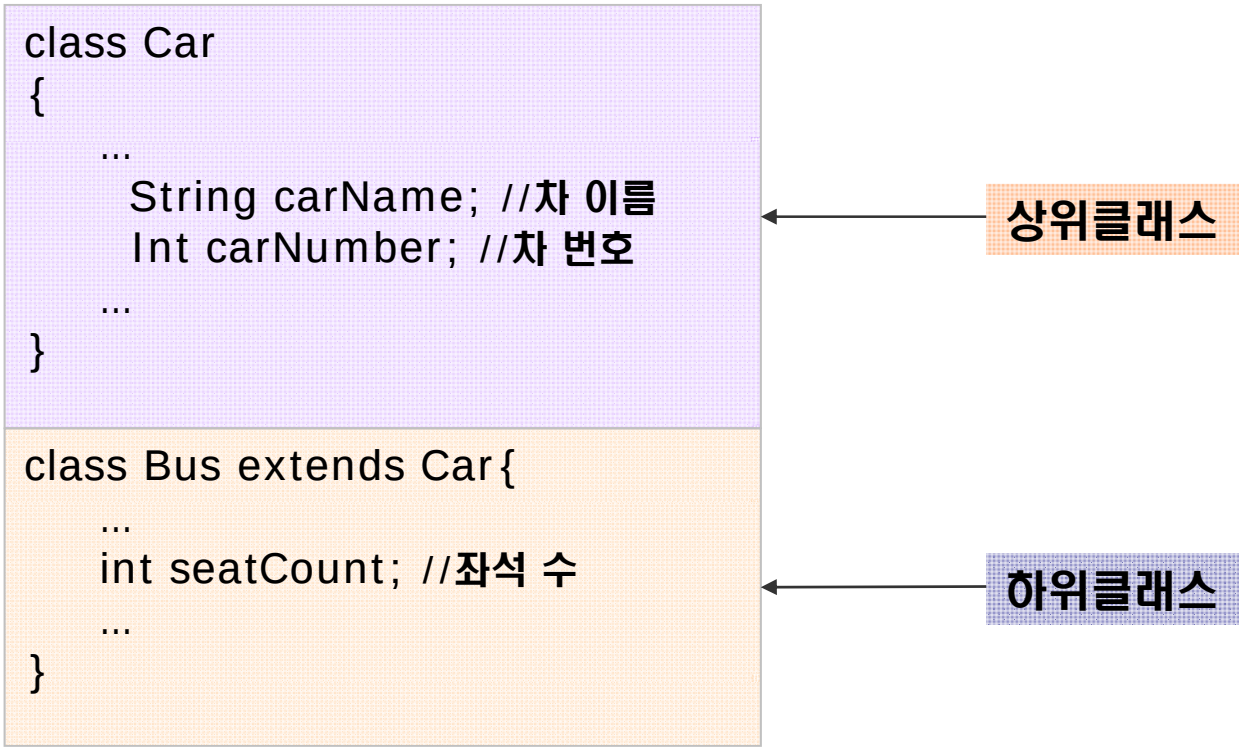


상위 클래스(Super-class)
- 바퀴가 굴러서 나아감
- 사람이나 짐을 실어 옮기는 기관

하위클래스(Sub-class)
- 각각의 구체적인 기능

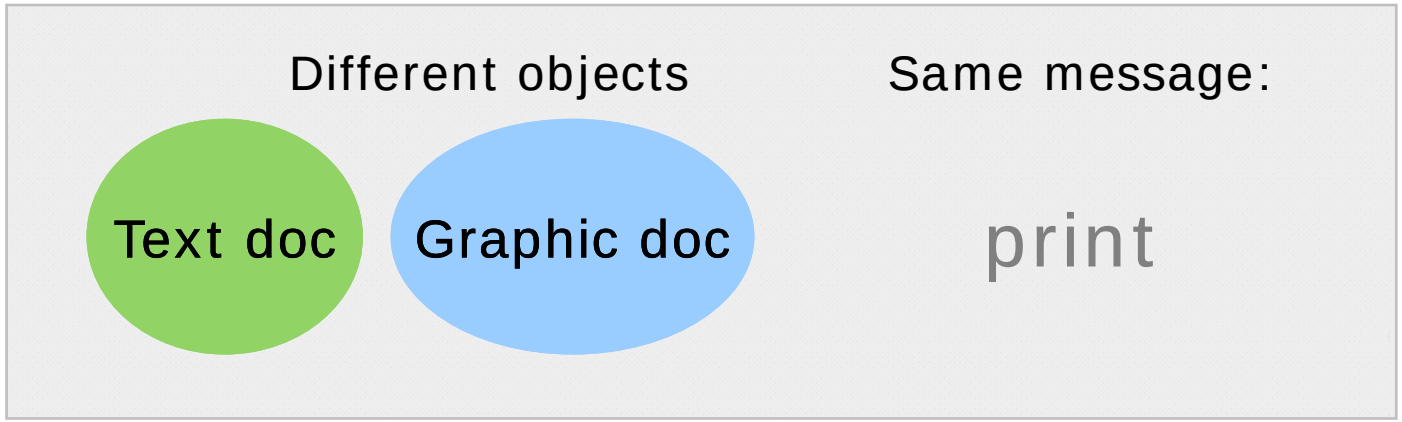
상속(Inheritance) (1/2)

❖ 상속의 예



다형성(Polymorphism)

- 의미
  - 하나의 인터페이스를 통해 서로 다른 구현을 제공하는 것



구현 (1/2)

- 의미
  - 코드 작성 또는 프로그래밍이라고 함
  - 설계의 최하위 상세화 과정
  - 코드 작성, 디버깅, 통합, 개발자 테스트(단위 테스트, 통합 테스트) 작업을 포함

- 개발자의 코딩 스타일
  - 일의 효율에 영향을 끼칠 수 있음
  - 각 개발사는 코딩 스타일 지침서를 구비하여 팀원들이 지침대로 코드를 작성하도록 조율하기도 함

구현 (2/2)

- 코딩 스타일
  - 한 줄에 한 문장만 써라
  - 선언문과 실행문을 구분하라
  - 단락을 구분하라
  - 내부 블록과 피제어부는 들여써라
  - 쓸데없는 들여쓰기를 하지 마라
  - 한 줄 주석과 주석 상자를 구분하라
  - 프로그램의 앞부분에 머리 주석을 반드시 달아라
  - 함수의 역할을 접두사로 활용하라
  - 이름을 의미 있게 지어라
  - 이름은 의미를 잃지 않는 범위에서 짧게 지어라
    [좋은 코딩 나쁜 코딩 중]

7. 확인과 검증

소프트웨어 개발과 품질

- 품질의 다양한 의미
  - 프로그램이 정상적으로 작동하는 것
  - 프로그램에 기대하는 막연한 완성도
  - 명시된 요구사항을 만족시키는 것
  - 고객이 의도한대로 요구사항을 올바르게 정의하는 것

확인과 검증 (1/2)

- 확인(Verification)
  - 올바른 제품을 생성하고 있는가?(Are we building the right product?) [Boehm]
  - 소프트웨어가 정확한 요구사항에 부합하여 구현되었음을 보장하는 활동
  - ‘요구사항 명세서에 맞게 올바른 방법으로 제품을 만들고 있음’ 을 보장
- 검증(Validation)
  - 제품이 올바르게 생성되고 있는가?(Are we building the product right?) [Boehm]
  - 소프트웨어가 고객이 의도한 요구사항에 따라 구현되었음을 보장하는 활동
  - ‘고객이 의도한 환경이나 사용 목적에 맞게 올바른 제품을 만들고 있음’ 을 보장
- 확인과 검증 작업은 실제로 구분하기 어려운 경우가 존재함
  - 결국, 소프트웨어의 품질을 보장하는 것

확인과 검증 (2/2)

❖ 확인과 검증 방법의 종류

– 정적(Static)인 방법

- 소프트웨어를 실행하지 않고 결함을 찾아내는 것
- 여러 참여자들이 모여 소프트웨어를 검토하여 결함을 찾아냄
- 소프트웨어 개발 중에 생성되는 모든 산출물들에 대해서 적용 가능
- 대표적인 방법
  - ➢ 검토(Review)
  - ➢ 인스펙션(Inspection)
  - ➢ 워크스루(Walk-through)

– 동적(Dynamic)인 방법

- 소프트웨어를 실행하여 결함을 찾아냄
- 발견된 결함은 디버깅 활동으로 확인하여 수정함
- 대표적인 방법
  - ➢ 테스팅

동료 검토란?

- 정의
  - 개발 동료들이 검출된 결함의 개선을 위해 정의된 순서를 따르는 소프트웨어 작업 산출물을 검토하는 작업
    [SEI/CMU, "The Capability Maturity Model", Addison-Wesley, 1994]
  - 개발자가 자신의 동료들이 완료한 작업을 검토하는 것

- 목적
  - 사용자 인터페이스 프로토타입, 요구 명세서, 아키텍처, 설계 및 기타 기술적 산출물의 품질 보증

테스팅(Testing)

- 의미
  - 기존 조건 및 필요 조건(즉, 결함/에러/버그) 사이의 차이점을 발견하기 위하여 소프트웨어 항목을 분석하고, 분석된 항목의 특성을 평가하는 프로세스
    [IEEE-Std-829]
  - 에러를 발견하려는 의도를 가지고 프로그램을 실행하는 프로세스
    [Myers]

테스팅과 디버깅의 차이점

	테스팅(Testing)	디버깅(Debugging)
목적	알려지지 않은 에러의 발견	이미 알고 있는 에러의 수정
수행	시스템 내부 관련자, 테스팅 팀 등 외부의 제 3자	시스템 내부 관련자
주요 작업	에러 발견 (Fault Detection)	에러의 정확한 위치 파악(Fault Location) 에러의 타입 식별(Fault Identification) 에러 수정(Fault Correction)

테스트 케이스(Test Case)

- 의미
  - 테스트의 목적에 맞게 테스트 조건, 입력값, 예상 출력값, 실제 테스트 결과를 기록하는 것
- 목적
  - 테스터가 테스팅을 체계적으로 할 수 있도록 함
  - 개발자가 테스팅 결과를 통해 디버깅을 하는 기준이 됨

테스트 케이스의 예

테스트 케이스 ID: ST-0001					
목적	로그인 시 아이디와 패스워드의 대소문자를 구분하여 처리한다.				
테스트 조건	아이디/비번 : abcd / abcd 가 DB에 이미 입력되어 있음.				
테스터	한동석	테스트 일자	2006.10.01~2006.10.01		
단계	입력값	예상 출력값	실행 결과	조치사항	조치 후 시험결과
1	아이디: ABCD 패스워드: abcd	아이디 없음 경고	정상 로그인	디버깅 필요	아이디 없음 경고
2	아이디: abcd 패스워드: ABCD	패스워드 틀림 경고	패스워드 틀림 경고	-	-
3	아이디: abcd 패스워드: abcd	정상 로그인	정상 로그인	-	-

테스팅 종류

- **테스트 정보를 얻는 대상에 따른 분류**
  - **블랙박스 테스팅(Black-Box Testing)**
    - 요구사항 명세서(SRS)나 설계서로부터 테스트 케이스 추출
  - **화이트박스 테스팅(White-Box Testing)**
    - 내부구조(소스 코드)를 기반으로 테스트 케이스 추출

블랙박스 테스팅(Black-Box Testing)

❖ 개요

- **요구사항 명세서나 설계서를 참조하면서 수행하는 테스팅**
  - 소스 코드 자체의 로직에는 관심이 없고 입, 출력값에만 관심이 있다

- **방법**
  - 동등분할
  - 경계값 분석
  - 의사결정 테이블

동등분할

- 개요
  - 입력값이 범위가 정해져 있을 경우, 각 범위의 대표값을 이용하여 테스팅
- 장점
  - 간단하고 이해하기 쉬움
  - 이용자가 작성 가능
  - 무작위 방법보다 체계적인 방법

동등분할의 예 (1/2)

❖ 사용자 요구사항

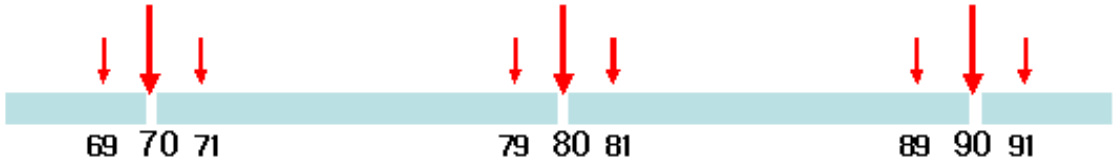
- 100점이 만점이고 0 ~ 100점을 받을 수 있는 시험이 있다. 시험 점수를 입력하면, 점수에 따라 다음과 같이 A부터 F까지의 성적을 출력하라.

	성적
90점 이상 ~ 100점 이하	A
80점 이상 ~ 90점 미만	B
70점 이상 ~ 80점 미만	C
0점 이상 ~ 70점 미만	F

경계값 분석

❖ 개요

- 입력 값의 주요 오류 대상인 경계값을 입력값으로 테스트 케이스를 작성하여 테스팅



❖ [예제]

- 동등분할의 예제를 경계값 분석 방법을 이용하여 테스트 케이스를 추출한 경우

테스트케이스	1	2	3	4	5
입력 값(점수)	-1점	0점	99점	100점	101점
점수 범위	점수 범위 초과	정상	정상	정상	점수 범위 초과
예상 결과값	경고창	F	A	A	경고창
실제 결과값	경고창	F	A	A	경고창

의사결정 테이블

- 개요
  - 입/출력값이 True, False로 결정될 수 있는 경우 모든 경우의 수를 확인해볼 수 있는 방법
- 활용
  - 입력, 출력 값이 Yes, No 으로 결정 될 수 있는 경우
  - 적은 수의 조건을 가진 입력값에 유용함

의사결정 테이블의 예

❖ 사용자 요구사항

- 아이디와 비밀번호를 입력하여 둘 모두 유효하면 정상 로그인이 된다. 그러나 아이디가 유효하지 않을 경우 잘못된 아이디라는 경고창을, 아이디는 유효하나비밀번호가 유효하지 않으면 잘못된 비밀번호라는 경고창을 보여준다.

❖ 의사결정 테이블 테스트 케이스

테스트 조건		1	2	3	4
입력값	유효한 아이디	T	T	F	F
	비밀번호	T	F	T	F
예상 출력값	로그인 성공	T	F	F	F
	잘못된 아이디 경고창	F	F	T	T
	잘못된 비밀번호 경고창	F	T	F	F
실제 출력값	로그인 성공	T	F	F	F
	잘못된 아이디 경고창	F	F	T	T
	잘못된 비밀번호 경고창	F	T	F	F

화이트박스 테스팅 (White-Box Testing)

❖ 개요

- **소스코드를 직접 참조하면서 수행하는 테스팅 기술**
- **방법**
  - 문장 커버리지(Statement Coverage)
  - 분기 커버리지(Branch Coverage)
  - 조건 커버리지(Condition Coverage)
  - 다중 조건 커버리지(Multiple Condition Coverage)

문장 커버리지 (Statement Coverage)

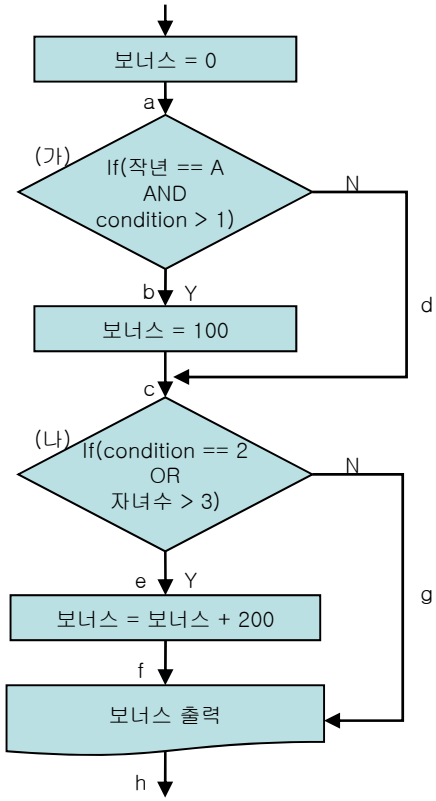
❖ 개요

- 프로그램을 구성하는 문장들이 최소한 한번은 실행될 수 있는 입력 값을 테스트 케이스로 선정함

❖ [예제]

- 테스팅 예제 순서도를 문장 커버리지를 적용하여 추출한 테스트 케이스

ID	테스트 케이스		
	입력값	경로	출력값
1	(A, 2, 2)	(a-b-c-e-f-h)	300
…	…	…	…



< 테스팅 예제 순서도 >

분기 커버리지 (Branch Coverage)

❖ 개요

- 프로그램에 있는 분기를 최소한 한번은 실행하게 하는 테스팅하는 방법

❖ [예제]

- 테스팅 예제 순서도를 분기 커버리지를 적용하여 추출한 테스트 케이스

ID	테스트 케이스		
	입력값	경로	출력값
1	(A, 2, 2)	(a-b-c-e-f-h)	300
2	(B, 1, 2)	(a-d-c-g-h)	0
…	…	…	…

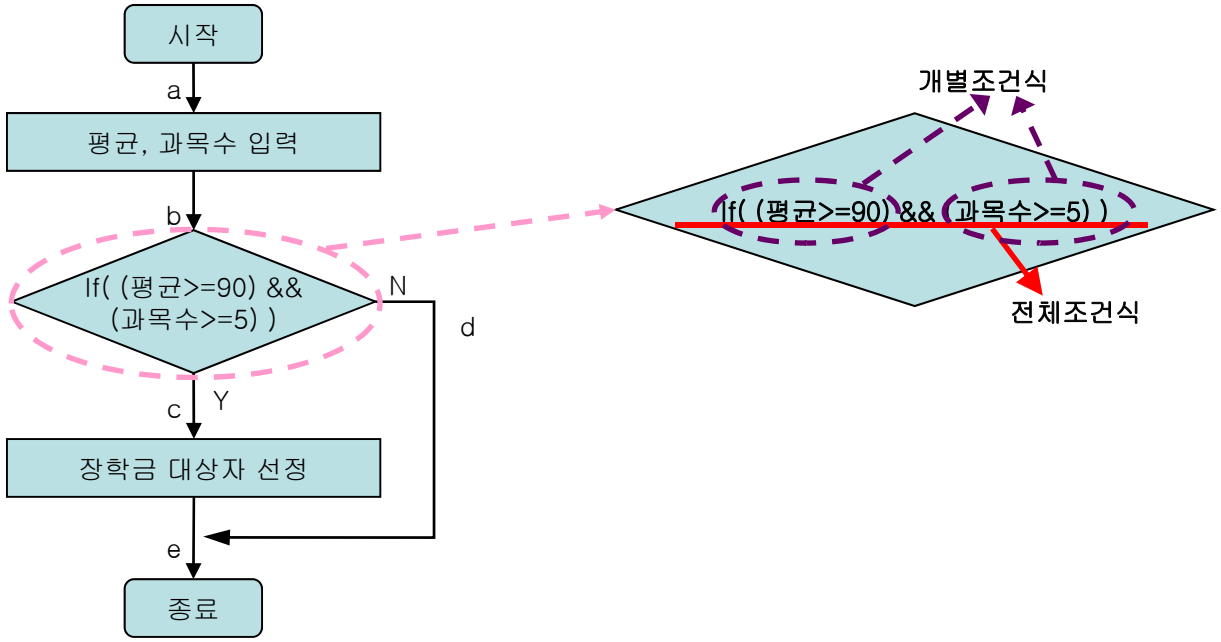
조건 커버리지 (Condition Coverage)

- 개요
  - &&, || 등의 조건을 가진 분기문이 전체 조건식의 결과와 관계없이 &&나 || 전후의 각 개별 조건식이 참 한 번, 거짓 한 번을 갖도록 테스트 케이스를 만드는 방법

조건 커버리지 (Condition Coverage)의 예

❖ 사용자 요구사항

- 학생의 평균과 과목수를 받아서, 장학금 대상자를 선정하라. 장학금 대상자는 평균이 90점 이상이고, 과목수가 5과목 이상인 학생으로 한다.



조건 커버리지 (Condition Coverage)의 예

❖ 조건 커버리지 테스트 케이스

ID	테스트 케이스		
	입력값	경로	출력값
1	(95, 4)	(a-b-d-e)	대상자 아님
2	(72, 7)	(a-b-d-e)	대상자 아님
…	…	…	…

※ 입력값은 (평균, 과목수)이며, 출력값은 대상자 선정 여부이다.

❖ 조건 커버리지 테스트 케이스 진리표

평균	과목수	전체조건식
95 이면 참	4 이면 거짓	거짓
72 이면 거짓	7 이면 참	거짓
…	…	…

다중조건 커버리지(Multiple Condition Coverage)

❖ 개요

- 조건 커버리지가 각 개별 조건식의 조건을 검사하는 것이라면, 다중조건 커버리지는 전체 조건식의 조건을 검사하는 테스트 케이스를 만드는 방법

❖ [예제]

- 조건 커버리지의 예제를 다중조건 커버리지를 적용하여 추출한 테스트 케이스

ID	테스트 케이스		
	입력값	경로	출력값
1	(95, 4)	(a-b-d-e)	대상자 아님
2	(72, 7)	(a-b-d-e)	대상자 아님
3	(80, 4)	(a-b-d-e)	대상자 아님
4	(92, 5)	(a-b-c-e)	대상자
…	…	…	…

※ 입력값은 (평균, 과목수)이며, 출력값은 대상자 선정 여부이다.

다중조건 커버리지(Multiple Condition Coverage)

❖ [예제]

- 조건 커버리지의 예제를 다중조건 커버리지를 적용하여 추출한 테스트 케이스 진리표

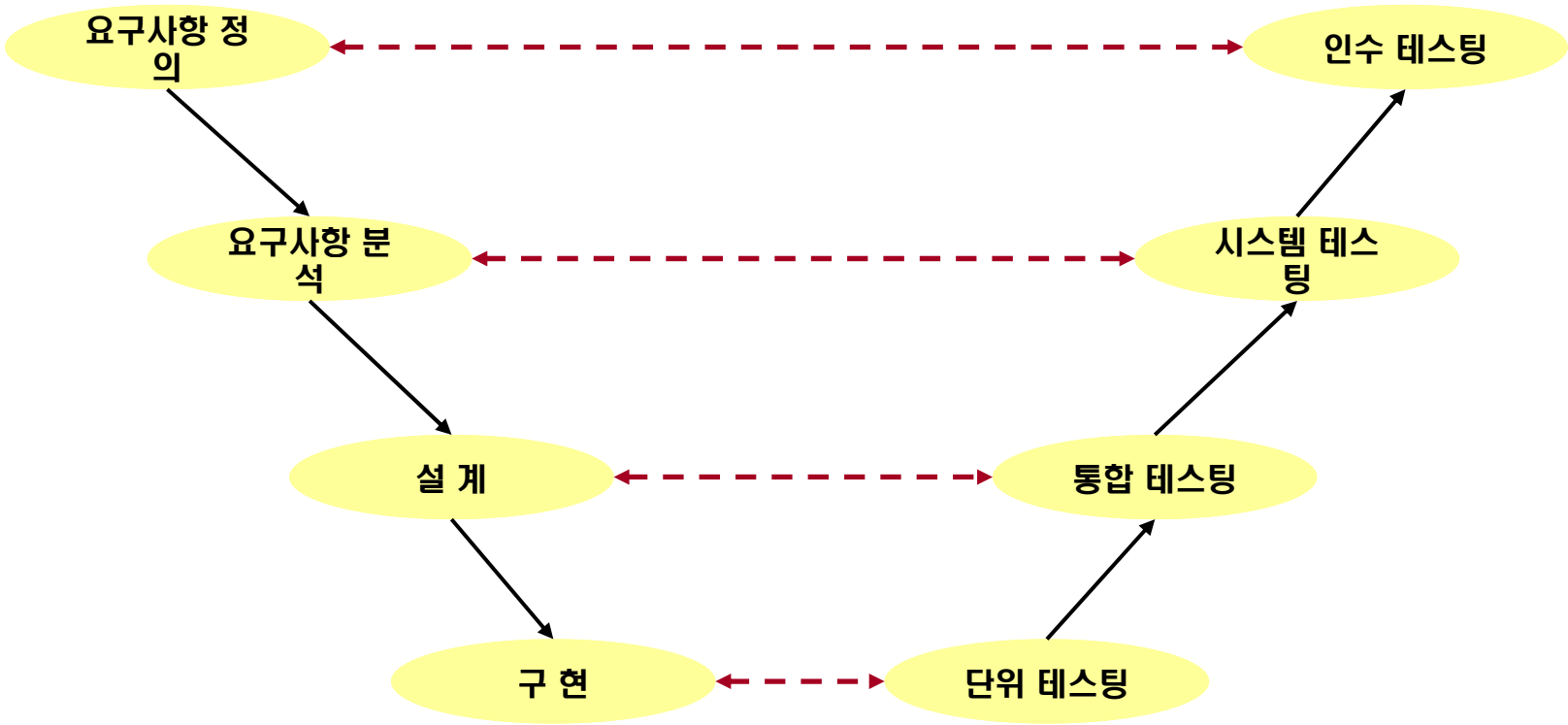
평균	과목수	전체조건식
95 이면 참	4 이면 거짓	거짓
72 이면 거짓	7 이면 참	거짓
80 이면 거짓	4 이면 거짓	거짓
92 이면 참	5 이면 참	참
…	…	…

화이트박스 테스팅의 특징

- 테스트의 목적과 조건에 맞게 적절한 방법 선택
- 각 테스팅 방법에 따라 복잡도, 소요되는 시간(비용)이 다름

테스팅 단계

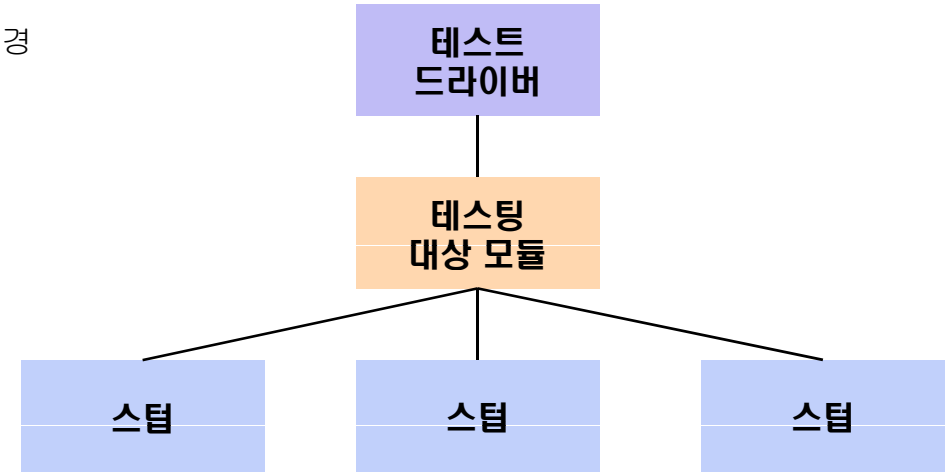
- 소프트웨어 개발 단계마다 생산되는 산출물을 이용하여 테스팅 수행



테스팅 단계

❖ 단위 테스팅(Unit Testing)

- **개요**
  - 구현 단계에서 각 모듈이 완성되었을 경우 개별적인 모듈을 테스팅
  - 테스팅의 주체는 해당 모듈의 개발자
  - 화이트박스/블랙박스 테스팅 모두 가능
- **테스트 할 모듈을 단독적으로 실행할 수 있는 환경 필요**
  - 스텁(Stub)
    - ➢ 테스팅 대상 모듈에서 호출하는 모듈
  - 테스트 드라이버(Test Driver)
    - ➢ 테스팅 대상 모듈을 호출하는 환경



테스팅 단계

❖ 통합 테스팅(Integrating Testing)

- **개요**
  - 모듈을 통합한 단계에서 수행되는 테스팅
  - 모듈간의 상호작용을 검사하는 테스팅
- **모듈 통합 방법에 따른 테스팅 기법 종류**
  - 빅뱅(Big Bang) 기법
    - ➢ 모듈을 한꺼번에 통합하여 테스팅을 하는 방법
    - ➢ 오류가 발생하였을 경우 어느 부분에서 오류가 났는지 찾기가 힘듬
  - 하향식(Top-Down) 기법
    - ➢ 가장 상위 모듈부터 하위 모듈로 점진적으로 통합하는 방법
    - ➢ 상위 모듈 테스팅 시, 하위 모듈에 대한 스텁이 필요
  - 상향식(Bottom-Up) 기법
    - ➢ 하위 모듈부터 테스팅 하고 상위 모듈로 점진적으로 통합하는 방법
    - ➢ 하위 모듈 테스팅 시, 상위 모듈에 대한 테스트 드라이버가 필요

테스팅 단계

❖ 시스템 테스팅(System Testing)

- **개요**
  - 모듈이 모두 통합된 후, 사용자의 요구사항이 만족되었는지 검사하는 테스팅
  - 고객에게 시스템을 전달하기 전, 시스템을 개발한 조직이 주체가 되는 마지막 테스팅
- **테스팅 대상**
  - 요구사항 명세서를 기초로 하여 사용자의 기능 요구사항
  - 보안, 성능, 신뢰성 등의 비기능 요구사항

테스팅 단계

❖ 인수 테스팅(Acceptance Testing)

- **개요**
  - 시스템이 사용자에게 인수되기 전, 사용자에 의해 실시되는 테스팅
  - 실제 사용자가 운영하는 환경에서 실시
  - 인수 테스팅을 통과해야만 시스템이 정상적으로 사용자에게 인수되고 프로젝트는 종료됨

8. 형상 관리

형상이란?

❖ 의미

- 소프트웨어 개발 산출물(문서나 소스 코드 등)이 배치되어 있는 배열

Configuration = 형상 = 形狀

영어사전 (총 1 건의 검색결과를 찾았습니다.)

configuration [kənfìgjuréiʃən] 발음듣기 단어장에 추가

명

1. (각 요소·부분의) 상대적 배치[배열]; 그것에 의해 결정되는 외형, 형태.
2. 〈천문〉
a) 성위(星位).
b) 성군, 별자리.
3. 〈물·화〉 원자 배열.
4. 〈컴퓨터〉 (시스템의) 구성.
5. 〈심리〉 형태(Gestalt).

~·al 형 ~·al·ly 부 ~·ra·tive 형

출처: 네이버 사전

형상 관리란?

- 정의
  - 형상 항목을 식별하여 그 기능적 물리적 특성을 문서화하고,
  - 그러한 특성에 대한 변경을 제어하고,
  - 변경 처리 상태를 기록 및 보고하고,
  - 명시된 요구사항에 부합하는지 확인하는 기술적이고 관리적인 감독, 감시 활동 [IEEE-Std-1042]

- 목적
  - 프로젝트의 생명 주기 동안 제품의 무결성(integrity)과 변경에 대한 추적성 (traceability)을 확보하기 위한 활동

형상 관리 활동의 필요성

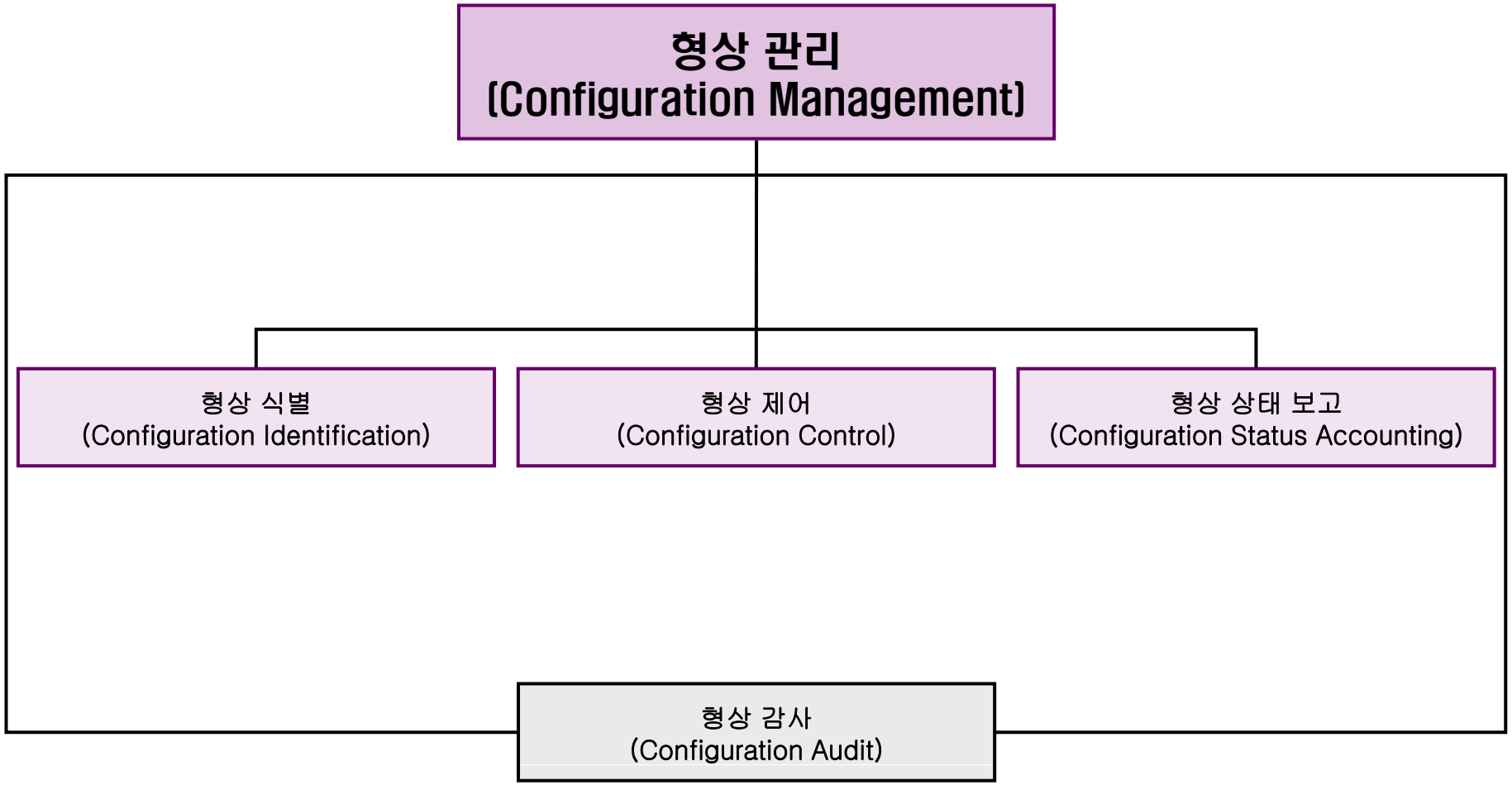
❖ 프로젝트에 내재된 문제점

- 요구사항의 변화가 많다.
- 산출물에 대한 수정 결과가 관련자들에게 제대로 통보되지 않는다.
- 많은 개발자들이 동일한 산출물에 대해 개별적으로 이중 작업을 실시한다.
- 하나의 산출물이 여러 개의 사본으로 존재하여 작업에 혼란을 초래한다.

❖ 형상 관리 활동의 필요성

- 소프트웨어의 특징으로 인해 발생할 수 있는 위험을 최소화하기 위해
  - 소프트웨어의 특징?
    - ➢ 비가시성, 변경 추적의 어려움, 관리와 통제의 어려움, 요구사항 변경으로 인한 잦은 변경 발생

형상 관리 활동(1/2)



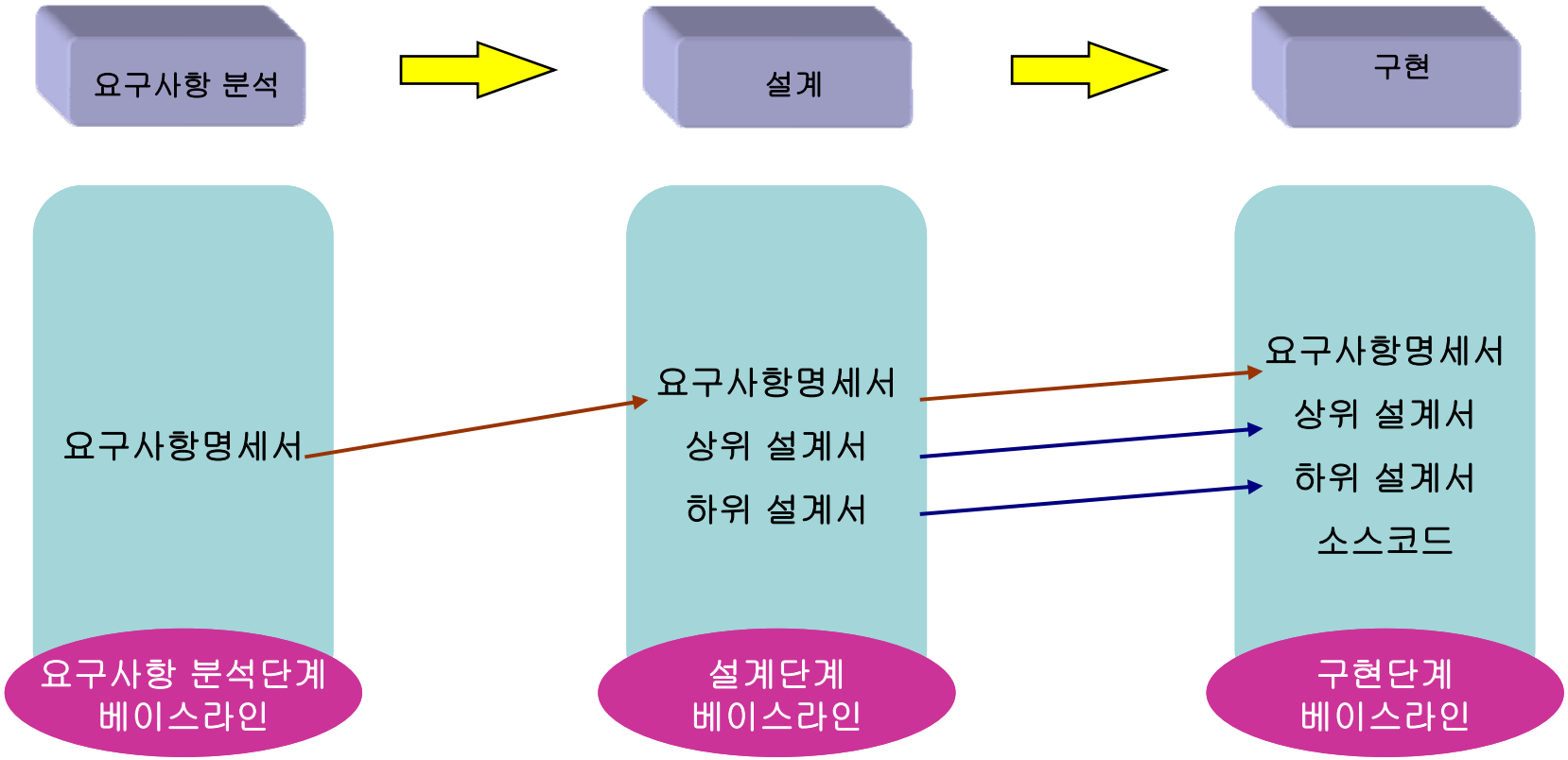
베이스라인(Baseline) 기준 선정 (1/3)

- 베이스라인(Baseline)이란?
  - 소프트웨어 개발의 특정 시점에서 형상 항목이 소프트웨어 개발에 하나의 완전한 산출물로써 쓰여질 수 있는 상태의 집합(버전 1.0)
  - 책임이 있는 관리를 통해 공식적으로 검토 및 동의되었고, 추후 개발의 기초가 되며, 오직 공식적인 변경 통제 절차에 의해서만 변경될 수 있는 상태
    [IEEE 1024]

- 베이스라인 기준 수립
  - 형상 관리 계획서 작성시 수립
  - 베이스라인 변경을 위해서는 형상 통제 위원회의 평가와 승인이 필요함

베이스라인(Baseline) 기준 선정 (2/3)

❖ 개발 주기 단위의 베이스라인 기준 수립



베이스라인(Baseline) 기준 선정 (3/3)

❖ 구현 단계 후 베이스라인 예제