

프로젝트 방법론

프로젝트 및 개발 프로세스 개요

목 차

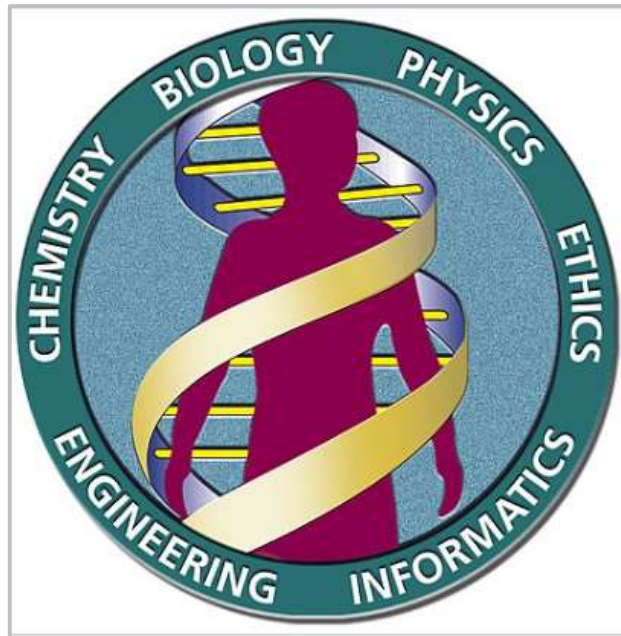
- 프로젝트의 정의 및 특징
- 소프트웨어 공학의 정의
- 소프트웨어 개발 프로세스 모델

프로젝트의 정의와 특성

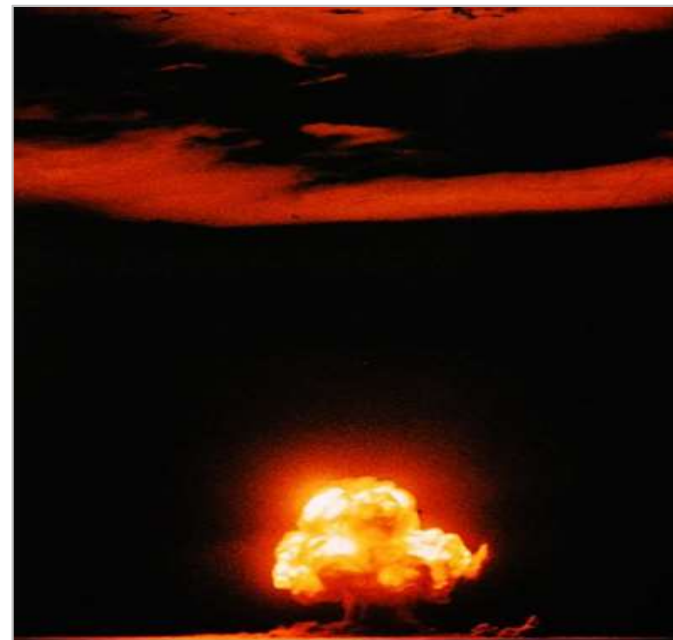
■ 프로젝트 정의

- 제품, 서비스 또는 결과물을 창출하기 위해 한시적 투입하는 노력
- 일정한 기간 안에 일정한 목적을 달성하기 위해 수행하는 업무의 묶음
- 정해진 기간, 배정된 금액, 투입인력 등 일정한 제약조건 하에서 각종 요구사항(requirement)을 수행하는 방식으로 진행

Human Genome Project, HGP



Manhattan Project



프로젝트의 정의와 특성

■ 프로젝트 특성

- **명확한 목적과 목표** 가짐
 - 목표는 성과로 나타나고 평가되며 명확한 결과물을 가짐
- **한시적(Temporary)**
 - 시작과 끝이 정해져 있음
- **유일함(Unique)**
 - 결과가 같은 프로젝트는 가능
 - 투입되는 노력, 환경 등 고유한 특성 지님
- **점진적으로 상세화(Progressive Elaboration)**
 - 초기의 개략적인 정의에서 시작하여 점차 구체화하여 구현
 - 고객의 초기 요구 사항 파악이 어렵고, 변경이 많음

프로젝트 관리의 3대 제약사항

■ 프로젝트 관리란

- 프로젝트 요구사항을 만족시키기 위한 지식, 기술, 도구, 기법 의미
- 프로젝트 관리 지식 체계(PMBOK : Project Management Body Of Knowledge)
- 관리 내용
 - 프로젝트의 요구사항을 정확히 파악
 - 가용한 자원을 효율적으로 사용
 - 예상치 못한 상황에 능동적으로 대응

프로젝트 관리의 3대 제약사항

■ 프로젝트 관리 필요성

- IT 프로젝트의 74%가 실패 (Standish Group 조사 결과)
- 프로젝트 실패 요인
 - 부정확한 요구 사항
 - 사용자 환경에 대한 이해 부족
 - 불충분한 자원
 - 비현실적인 사용자의 기대치
 - 관리 자원의 부족
 - 변경 관리의 부족
 - 불충분한 프로젝트 계획

프로젝트 관리의 3대 제약사항

■ 프로젝트 관리 삼각형

- 프로젝트 품질을 결정하는 세 가지 변수

➤ 범위

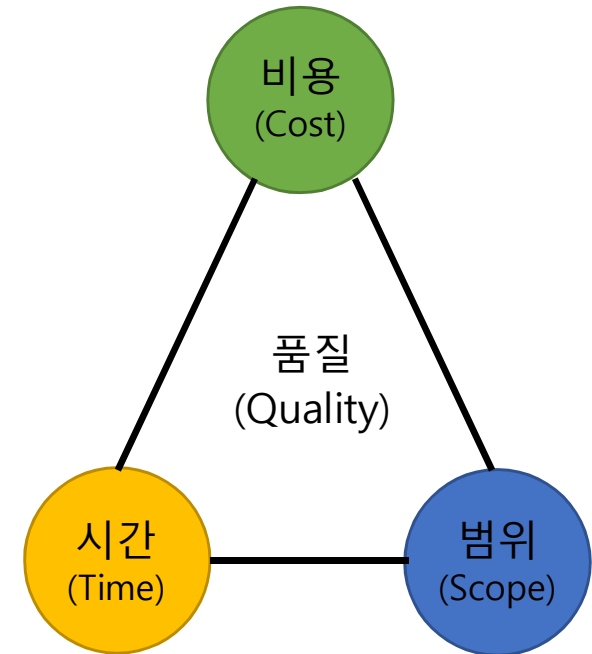
- 프로젝트 크기: 결과물 품질, 세부사항, 규모, 복잡성

➤ 비용

- 단순 금액 이상
- 프로젝트에 할당된 예산
- 팀원 수, 장비 및 시설(리소스)

➤ 시간

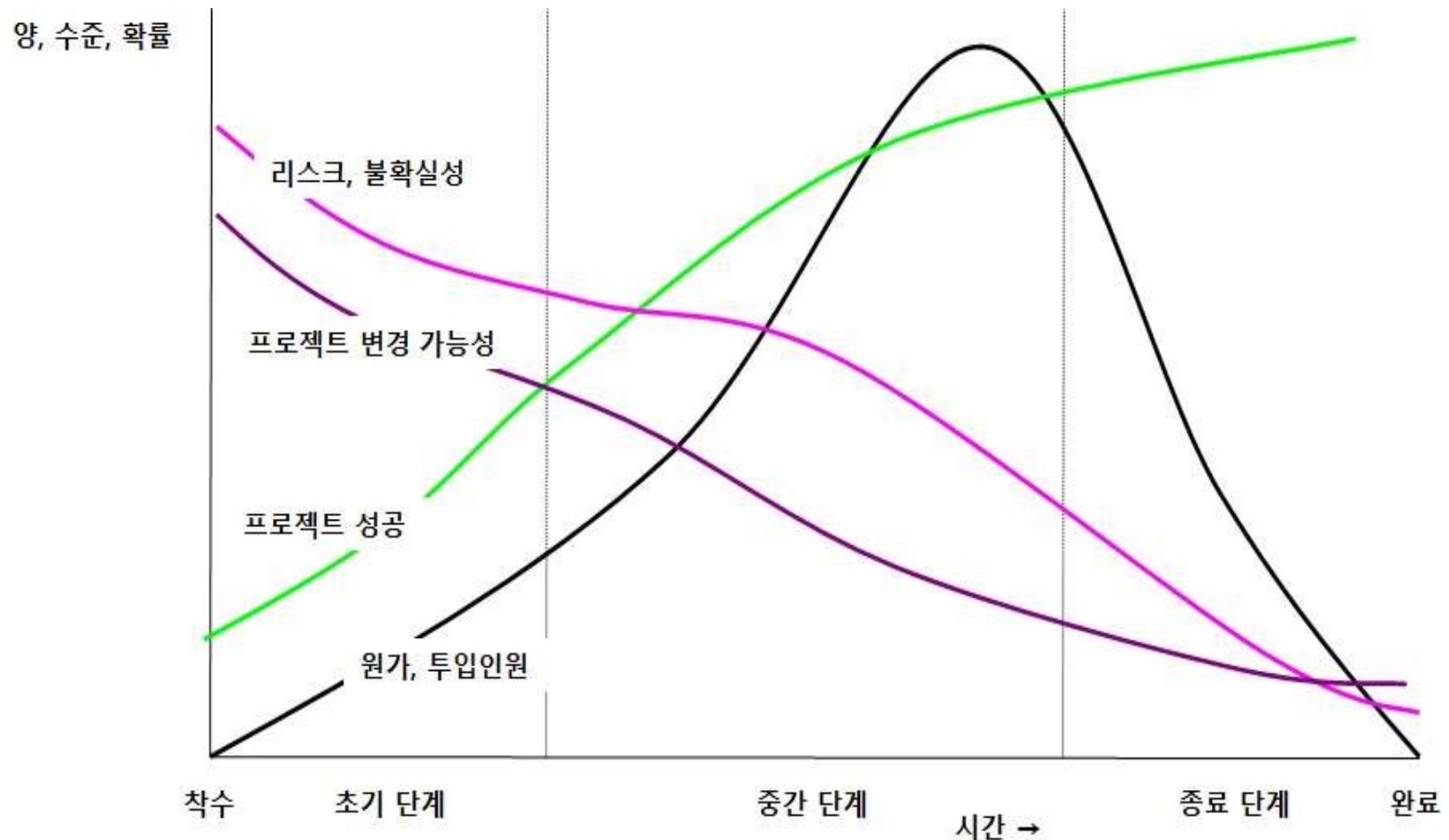
- 프로젝트 시작과 종결의 시점과 기간



프로젝트 관리 프로세스

■ 프로젝트 라이프사이클

- 프로젝트는 착수일과 완료일이 정의된 라이프 사이클



프로젝트 관리 프로세스

■ 프로젝트 라이프사이클

- 프로젝트 유형이나 업종에 따른 차이 있음
- IT 프로젝트 : 분석 → 설계 → 개발 → 구현
- 건설 프로젝트 : 실행 가능성 분석 → 계획 → 설계 → 건설 → 인수

■ 프로젝트 관리 프로세스

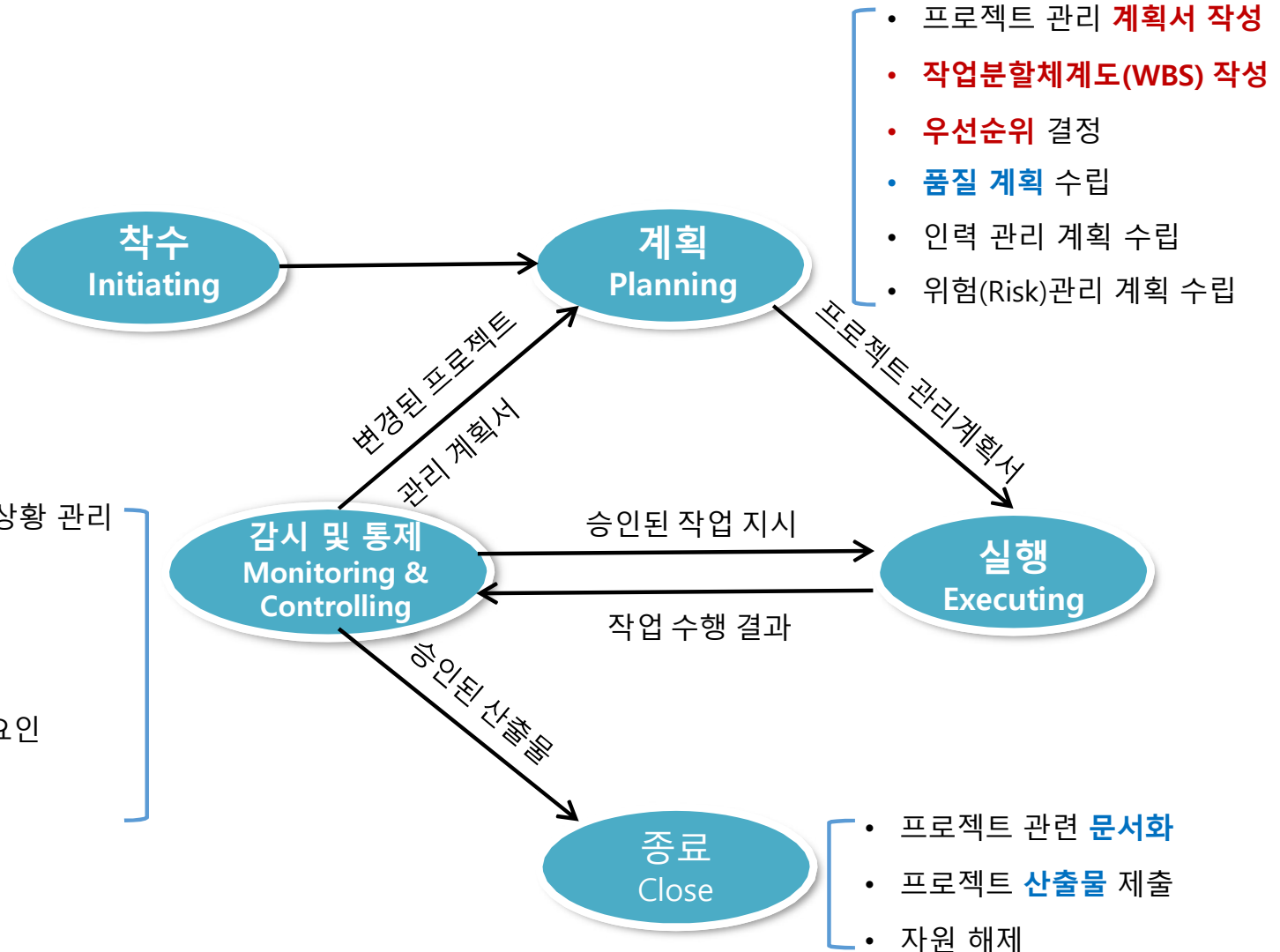
- 업종에 상관 없이 공통적
- 착수 → 계획 → 실행 → 통제 → 종료

프로젝트 관리 프로세스

■ PMBOK 5단계 관리 프로세스

- 프로젝트 **목표 설정**
- **요구사항 분석**
- **일정 및 예산** 산정
- **제약 조건** 문서화
- 필요 자원 결정
- 프로젝트 팀 구성

- 성과 측정 및 진행 상황 관리
- 변경 요청 수행
- 변경 사항 대응
- 비용 및 품질 통제
- 위험(risk) 및 유발 요인
- 프로젝트 활동 감시



프로젝트 관리 프로세스

■ PMBOK 10단계 관리 프로세스

프로젝트 범위 관리	프로젝트 일정 관리	프로젝트 비용 관리
<ul style="list-style-type: none"> 프로젝트를 완료하는데 필요한 작업들 포함 시키기 위한 프로세스 프로젝트 업무 범위 설정 및 승인을 받아 프로젝트 목표에 맞도록 관리하는 기능 프로젝트 범위 변경은 프로젝트 진행 어렵게 함. 처음 범위 잘 정의, 엄격 관리 중요 	<ul style="list-style-type: none"> 프로젝트 적시 완료 위한 프로세스 PMBOK10 영역 중 가장 많은 시간 소비 프로젝트 범위를 세부 작업으로 나눔 각 작업에 대한 일정과 자원, 인력, 예산 선정 변경 요소에 따른 일정 변경 포함되며 지속적으로 관리 계획 업데이트 	<ul style="list-style-type: none"> 승인된 예산 범위안에서 프로젝트 완료할 수 있도록 비용 계획, 추정, 예산 편성, 자금 조달 필수적인 부분이며 가장 민감한 부분
<ul style="list-style-type: none"> 요구 사항 수집 범위 정의 작업 분류 체계(WBS) 작성 범위 검증 범위 통제 	<ul style="list-style-type: none"> 활동 정의 활동 순서 배열 활동별 자원 산정 활동별 기간 산정 일정 개발 	<ul style="list-style-type: none"> 원가 산정 원가 예산 책정 원가 통제

프로젝트 관리 프로세스

■ PMBOK 10단계 관리 프로세스

프로젝트 품질 관리	프로젝트 인적 자원 관리	프로젝트 의사소통 관리
<ul style="list-style-type: none"> 프로젝트 요구사항을 충족할 수 있도록 품질 정책, 품질 목표, 품질 책임사항을 결정하는 프로세스 	<ul style="list-style-type: none"> 프로젝트 팀을 구성하고 관리하고 프로젝트 팀을 이끄는 프로세스 프로젝트 수행 자원들의 활동을 조직하고 조정하는 기능 	<ul style="list-style-type: none"> 프로젝트 계획, 정보 생성, 수집, 배포, 저장, 검색 그리고 최종처리가 적시에 수행되도록 하기 위해 필요한 프로세스 프로젝트의 이해관계자 간에 효율적인 정보 전달체계를 계획, 조직 관리하는 기능
<ul style="list-style-type: none"> 품질 계획 수립 품질 보증 수행 품질 통제 수행 	<ul style="list-style-type: none"> 인적 자원 계획 프로젝트 팀 확보 프로젝트 팀 개발 프로젝트 팀 관리 	<ul style="list-style-type: none"> 이해 관계자 식별 의사소통 계획 수립 및 정보 배포 이해 관계자 기대사항 관리 성과 보고

프로젝트 관리 프로세스

■ PMBOK 10단계 관리 프로세스

프로젝트 위험 관리	프로젝트 조달 관리	프로젝트 통합 관리
<ul style="list-style-type: none"> 프로젝트 수행 과정에서 일어날 수 있는 리스크(위험) 요인을 발견하고 분석 발생 가능한 리스크(위험) 대체 수립 	<ul style="list-style-type: none"> 프로젝트 팀 외부적으로 필요한 제품 및 서비스 조달하기 위해 필요한 활동 구성 및 관리 대부분 프로젝트에는 외부 조달 형태 	<ul style="list-style-type: none"> 프로젝트 정보 생성, 수집, 배포, 저장, 검색 그리고 최종처리가 적시에 수행되도록 하기 위해 필요한 프로세스 프로젝트의 이해관계자 간에 효율적인 정보 전달체계를 계획, 조직 관리하는 기능
<ul style="list-style-type: none"> 리스크 관리 계획 수립 리스크 식별 정성적 리스크 분석 수행 정량적 리스크 분석 수행 리스크 대응 계획 수립 리스크 감시 및 통제 	<ul style="list-style-type: none"> 조달 계획 조달 계약 조달 관리 조달 종결 	<ul style="list-style-type: none"> 프로젝트 현장 개발 프로젝트 관리 계획 개발 프로젝트 실행 지시 및 관리 프로젝트 작업 감시 및 통제 통합 변경 통제

프로젝트 관리 프로세스

■ PMBOK 10단계 관리 프로세스

프로젝트 이해관계자 관리

- 프로젝트 영향을 받는 모든 사람 또는 조직으로 고객, 사용자, 팀원, PM, 후원자, 경영진, 이사회, 지역 주민, 정부기관, 공급자 등..
 - 이해관계자의 긍정적 태도와 지원은 증대하고 반대와 저항은 감소하여 프로젝트 성공 가능성 극대화 목적
-
- [착수] 이해관계자 식별
 - [기획] 이해관계자 참여 계획 수립
 - [실행] 이해관계자 참여 관리
 - [감시및통제] 이해관계자 참여 감시

소프트웨어 공학

소프트웨어란?

■ 소프트웨어 (Software)

- 프로그램과 프로그램 개발, 운용, 유지 보수에 필요한 관련 정보 일체
- 소스 코드를 포함한 개발 과정에서 생성되는 모든 산출물
 - 각 단계에서 만들어지는 문서와 사용자 매뉴얼 등

■ 소프트웨어의 특징

- 손에 잡히지 않음 : 개발 작업 및 구조 파악에 대한 이해가 어려움
- 대량 생산이 쉬움 : 비용의 대부분은 개발
- 노동 집약적 : 자동화가 어려움
- 쉽게 변경이 가능 : 완전한 이해 없이 변경 가능
- 많은 훈련이 필요 : 개발자 능력에 많은 영향을 미침
 - 품질 문제, 다양한 지식 및 기술의 습득이 필요

소프트웨어 위기의 등장 (Software Crisis)

■ 소프트웨어 위기

- 1968년 NATO 소프트웨어공학 학회에서 처음 등장
- 컴퓨터에 의한 계산 용량과 문제의 복잡도가 급증
- 새로운 소프트웨어 개발 방법의 필요성을 인식
- 소프트웨어가 하드웨어 개발 속도를 따라가지 못함
- 소프트웨어가 더이상 사용자들의 요구를 충족시킬 수가 없음

소프트웨어 위기의 등장 (Software Crisis)

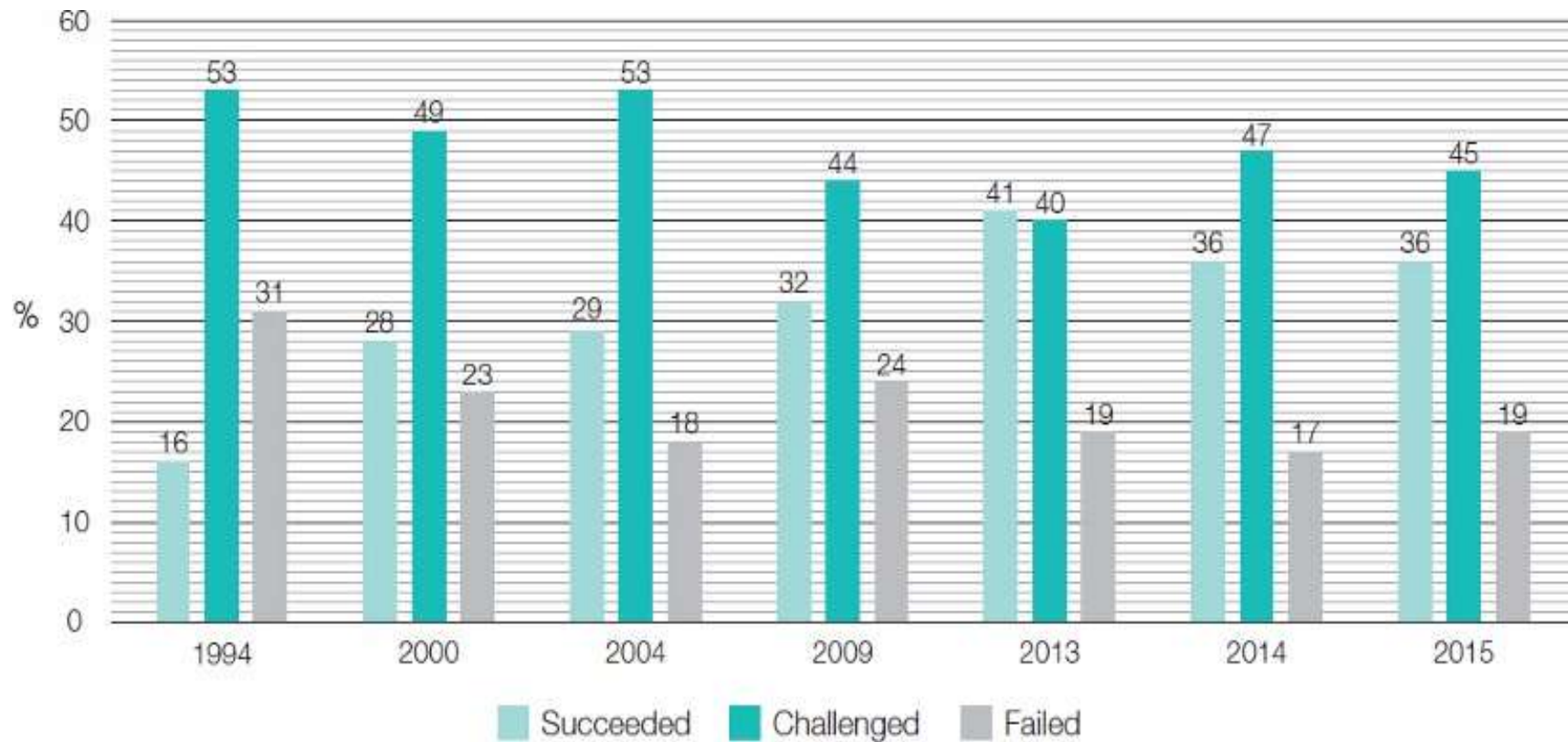
■ 소프트웨어 위기 원인



소프트웨어 개발 프로젝트 통계 분석

■ 소프트웨어 개발 프로젝트의 성공과 실패율

- 소프트웨어 개발 성공률은 조금씩 증가하였지만
- 실패율은 크게 감소하지 않음 → 점점 복잡해지는 소프트웨어, 사용자 요구 사항 다양

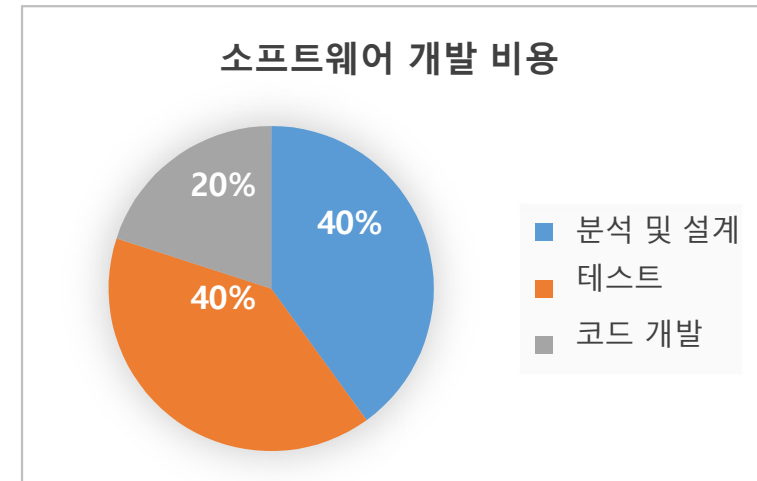


소프트웨어 개발이 어려운 이유

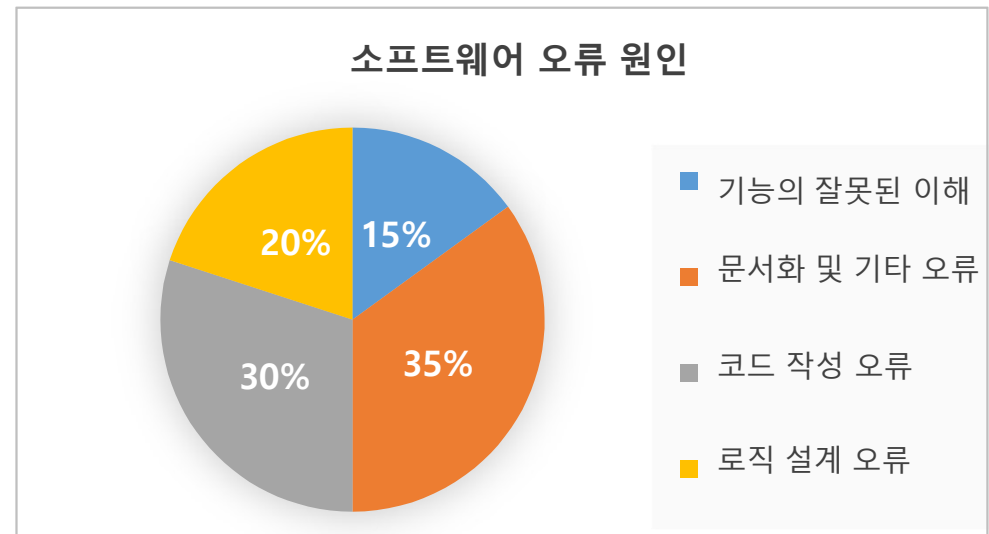
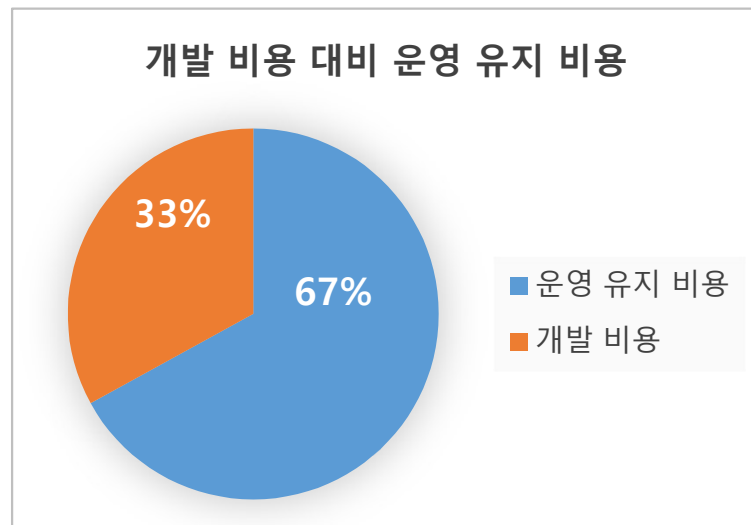
- 의사 소통 문제
 - 소프트웨어 개발 과정에 참여하는 다양한 역할자(프로젝트 관리자, 품질 관리자, 개발자, 사용자 등)간의 의사 소통 오류
- 개발에 의한 결과물
 - 조립이나 공식에 의한 문제 풀이가 아닌 개발자의 지적 활동에 의한 산출물
- 프로젝트 복잡성
 - 프로젝트마다 개발 기간, 개발자 수, 사용자 수준 등 차이로 유연한 관리 필요
- 개발자의 특성
 - 팀 기반 작업 참여도와 특정 언어의 숙련 정도
- 다양한 관리 이슈
 - 요구사항 변경관리, 일정 관리, 코드 버전 관리 등 다수의 활동 간의 조화가 필요

소프트웨어 특성 정보

- 소프트웨어 개발 비용
 - 분석 및 설계 과정이 코드 개발보다 중요



- 소프트웨어 개발 및 유지 보수
 - 소프트웨어 운영 유지 비용이 개발 비용의 2배



소프트웨어 공학

■ 소프트웨어 공학 정의

- 1990년대
 - 주어진 문제를 해결하는 품질 좋은 소프트웨어 개발, 유지 보수를 위해 적용되는 방법 및 기법
- 2010년대 이후
 - 소프트웨어 설계, 구현, 테스트, 문서화를 위한 과학적이고 체계적인 방법 적용
 - 소프트웨어 개발, 운용, 유지 보수와 같은 수명주기 전반을 체계적, 정량적으로 관리 하는 총체적인 활동

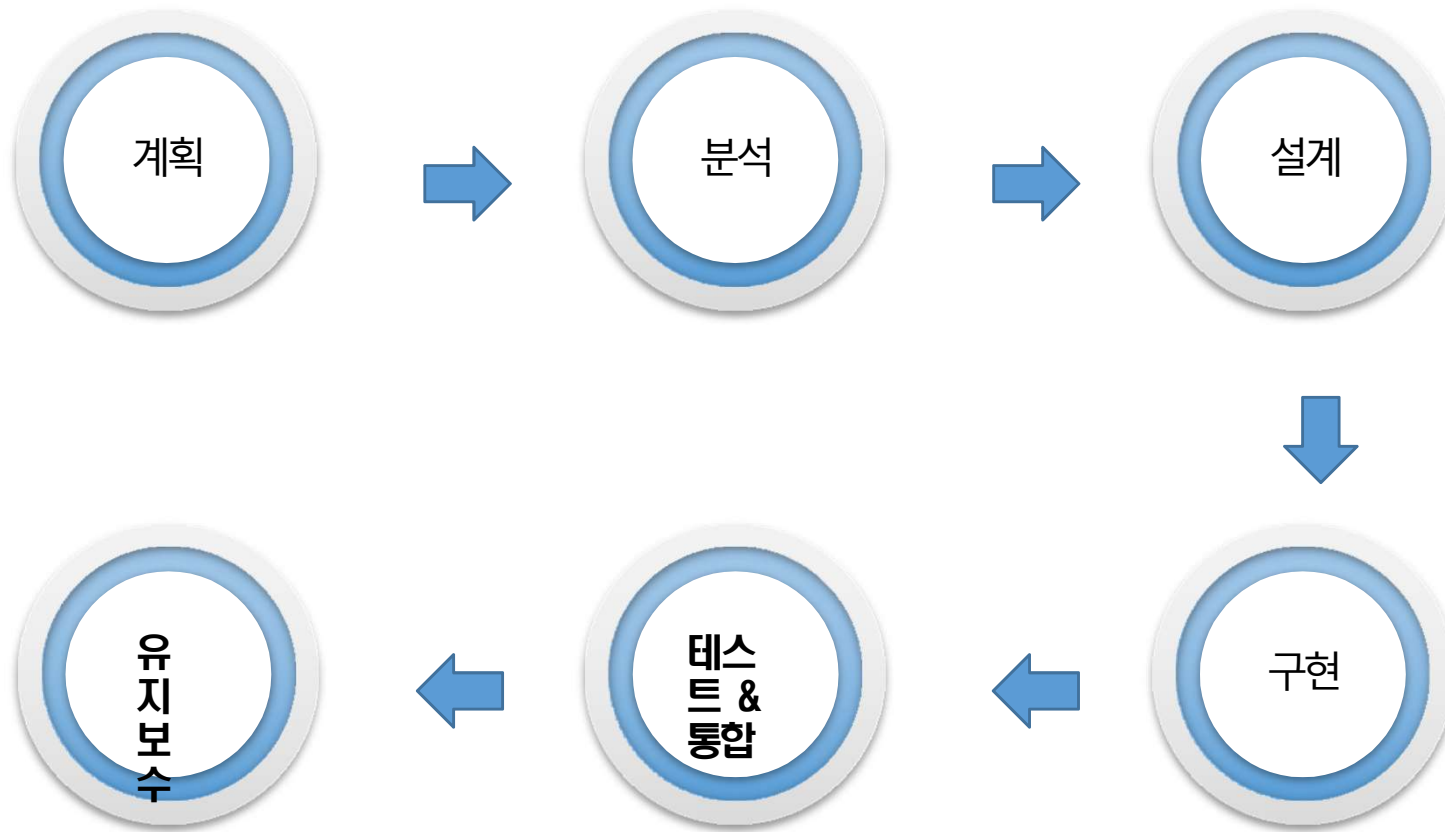
■ 소프트웨어 공학 목표

- 소프트웨어 개발의 어려움을 해결
- 효율적인 개발을 통해 생산성 향상
- 사용자가 만족하는 고품질의 소프트웨어 개발

소프트웨어 개발 생명주기

■ 소프트웨어 개발 생명 주기:

- Software Development Life Cycle: SDLC



소프트웨어 개발 프로세스

실현 가능성 분석

■ 실현 가능성 분석 (타당성 조사)

- 소프트웨어 시스템을 개발하고자 할 때 **비용, 일정, 기술적 수준** 등이 **충분히 가능 한가**를 살펴보는 것
- 개발 비용 **Cost**, 이득 **Benefits**, 대안 **Alternatives** 분석

실현 가능성 분석

■ 실현 가능성 분석 (타당성 조사)

개발 비용 (Cost)

- 개발 인건비, 하드웨어 비용
- 개발 및 운영 소프트웨어 도구 비용
- 유지 보수 비용
- 개발자 교육 비용
- 시스템 교체 비용 등

이득 (Benefits)

- 새로운 시스템이 제공하는 비즈니스 프로세스의 개선된 영역은?
- 운영의 효율성과 정확성은 증진되는가?
- 새로운 시스템을 사용했을 때 얼마만큼의 비용 절감 효과가 있는가?

대안 분석 (Alternatives)

- 비용과 이득 산정 시 다양한 측면을 고려한 Trade-off 분석
- 고급 기술자를 사용하는 경우와 기존 직원을 교육하는 경우 비교 분석

실현 가능성 분석을 위해 고려할 측면 #1

■ 경제적 측면

- 비용 대 효과 분석을 수행하여 프로젝트가 비용 측면에서 정당하게 평가 받을 수 있는가?

■ 기술적 측면

- 소프트웨어 개발 프로젝트에서 필요한 이론이나 기술에 대해 제약 사항이 없는가?
- 현재 가용한 기술 수준으로 목표 소프트웨어를 구현하기에 충분한가?

■ 스케줄 측면

- 소프트웨어 개발 프로젝트를 가용한 인력과 자원으로 주어진 기간 내에 완료할 수 있는가?

실현 가능성 분석을 위해 고려할 측면 #2

■ 운영적 측면

- 개발된 소프트웨어가 사용자에게 전달되었을 때, 데이터 입력이 가능한가?
- 소프트웨어운영에 대한 역할이나 책임 수행하기에 시스템 관리자 주저함이 없는가?

■ 동기적 측면

- 개발된 소프트웨어를 실제로 사용할 사용자가 그 필요성을 인정하고 있는가?
- 소프트웨어가 사용자에게 배포되었을 때, 지체 없이 정확하게 업무에 적용할 수 있는가?

■ 법적·윤리적 측면

- 개발된 소프트웨어 기능을 사용하는 과정에서 개인 정보 유출과 같은 법적 분쟁의 소지가 발생할 수 있는가?
- 사회적인 문제를 유발할 수 있는 기능이 포함되어 있는가?

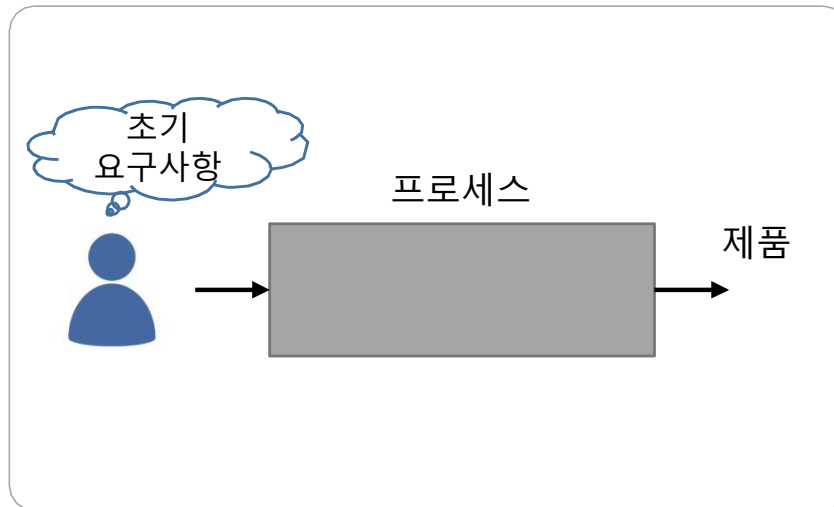
소프트웨어 개발 프로세스의 필요성

■ 소프트웨어 개발 프로세스란?

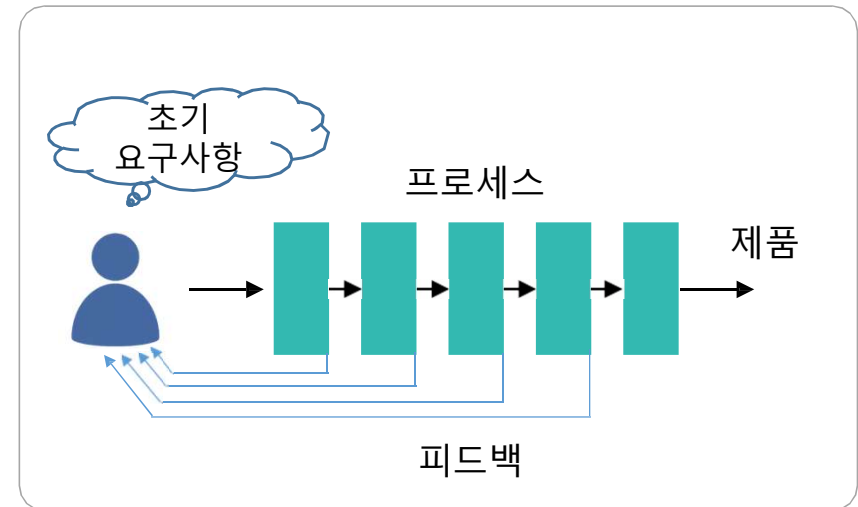
- 소프트웨어 개발과정을 체계적으로 계획, 관리할 목적
- 서로 관련 있는 활동을 단계로 분리하여 정의한 것

■ 소프트웨어 프로세스 필요성

- 요구사항 만족도 향상
- 수정 및 재개발 절감(비용 절감)
- 품질 관리 가능
- 프로젝트 성공



블랙박스 프로세스



투명한 프로세스

소프트웨어 개발 프로세스

■ 소프트웨어 개발 방법론: Methodology

- 소프트웨어를 만들 때 어떻게 만들어야 하는 지를 정의하는 방법

작업절차

- 소프트웨어를 개발할 때 진행하는 작업의 순서

작업방법

- 각 단계마다 수행해야 할 일(Task, 누가 언제, 무엇을)

산출물

- 단계별로 만들어지는 문서(목록, 설계서, 정의서, 명세서)

관리

- 개발 진행을 어떻게 관리하고 제어할 것인가에 대한 방법

기법

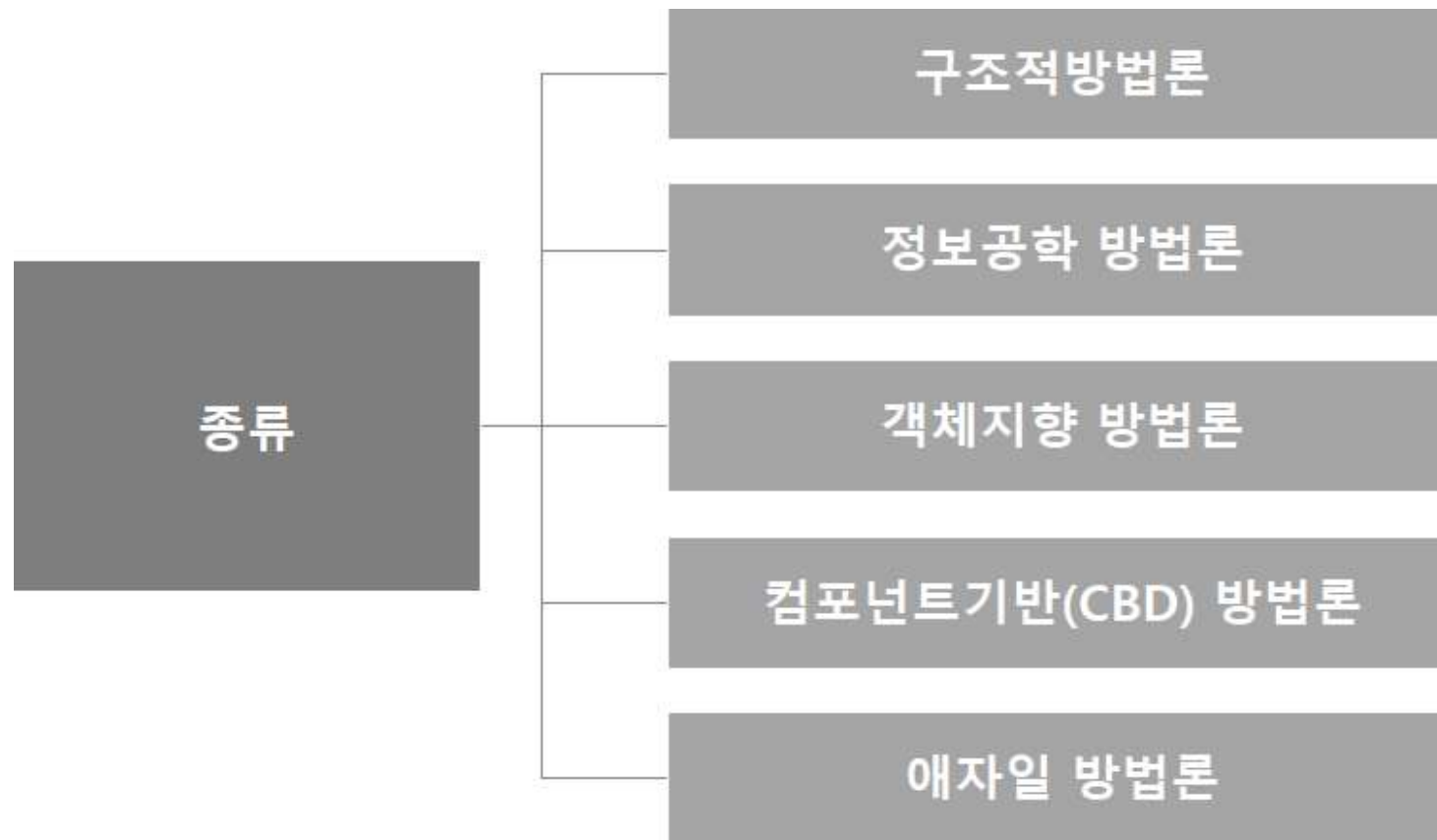
- 단계별 일을 진행할 때 사용하는 기술 이나 기법(DFD, ERD, User Case)

도구

- 사용되는 기법별 지원도구(Power Point, Excel, ERWin, Git, Draw IO)

소프트웨어 개발 프로세스

- 소프트웨어 개발 방법론: Methodology



소프트웨어 개발 프로세스

■ 소프트웨어 개발 방법론: Methodology

■ 구조적 방법론

: 절차 중심의 SW개발 방법론, 폭포수 모델 대표적

■ 정보공학 방법론

: 데이터나 모델링을 기반으로 한 프로토타입의 개발 방법론, 특징은 약함
중요성 낮음

■ 객체 지향 방법론

: 객체는 컴포넌트화 방식. 사람이 참여하여 수많은 반복을 거듭하고
개발을 진행하는 반복적 개발 방법론이 주로 사용

■ 컴포넌트 기반 방법론

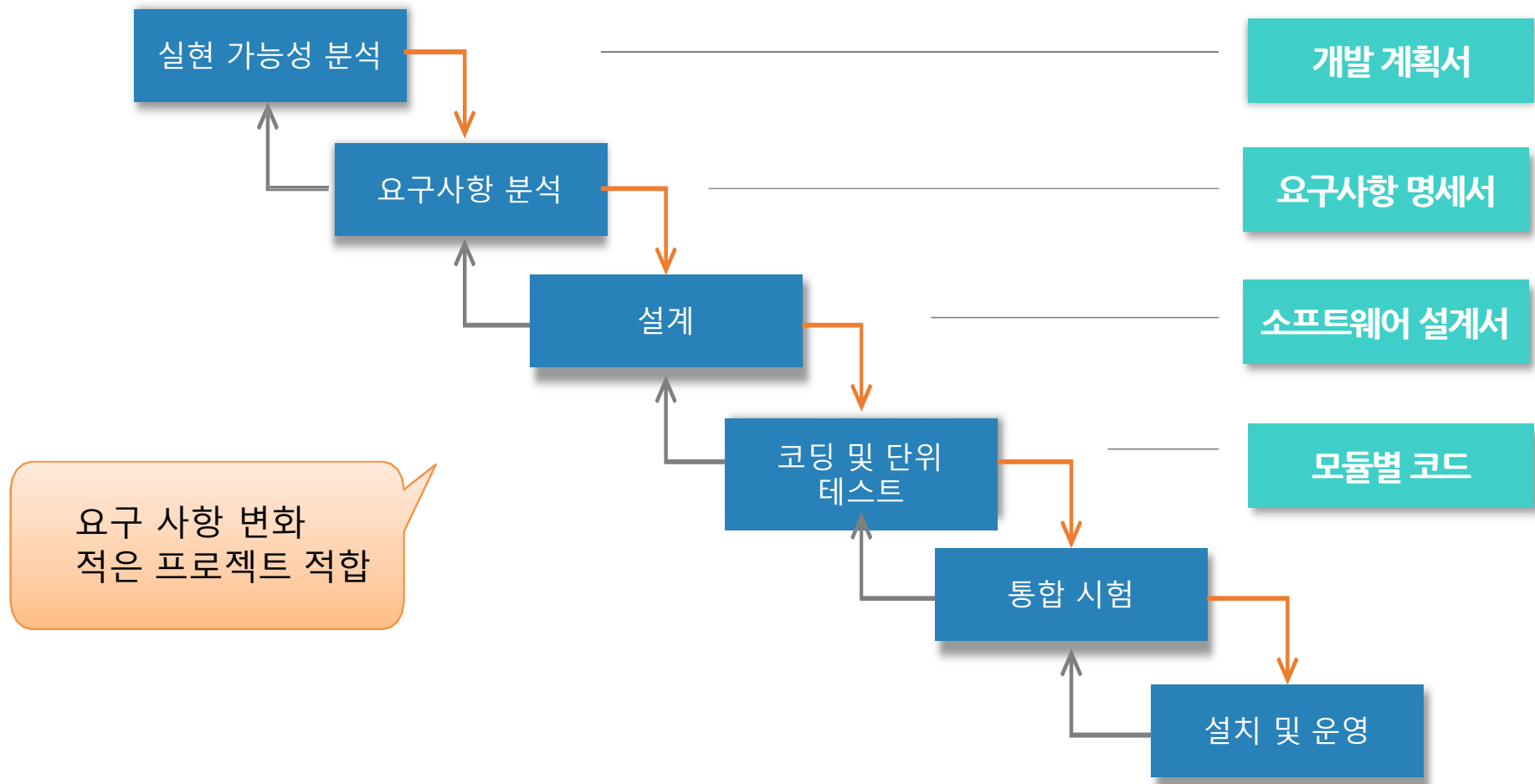
: 사용자와 개발자 간 반복적인 커뮤니케이션 중요하게 여기는 방법
컴포넌트를 **재사용** 함으로써 **개발 시간을 많이 단축**, 애자일 방법론 대표적

소프트웨어 개발 프로세스

- 구조적 방법론 - 폭포수 모델 (Waterfall Model)
 - 순차적인 단계로 소프트웨어를 개발하는 고전적 방법
 - 반드시 앞 단계가 완료되어야 다음 단계로 진행
 - 각 단계마다 정해진 소프트웨어 문서를 작성: 체계적으로 문서화
 - 작성된 문서는 사용자 확인 과정을 거쳐 다음 단계의 입력으로 사용됨
 - 폭포수 모델의 단점
 - 개발 과정에서 요구되는 변경을 수용하기가 어려움
 - 이전 단계의 결과물에 오류가 잔존하는 경우, 다음 단계에 양향을 줌

소프트웨어 개발 프로세스

■ 구조적 방법론 - 폭포수 모델 (Waterfall Model)



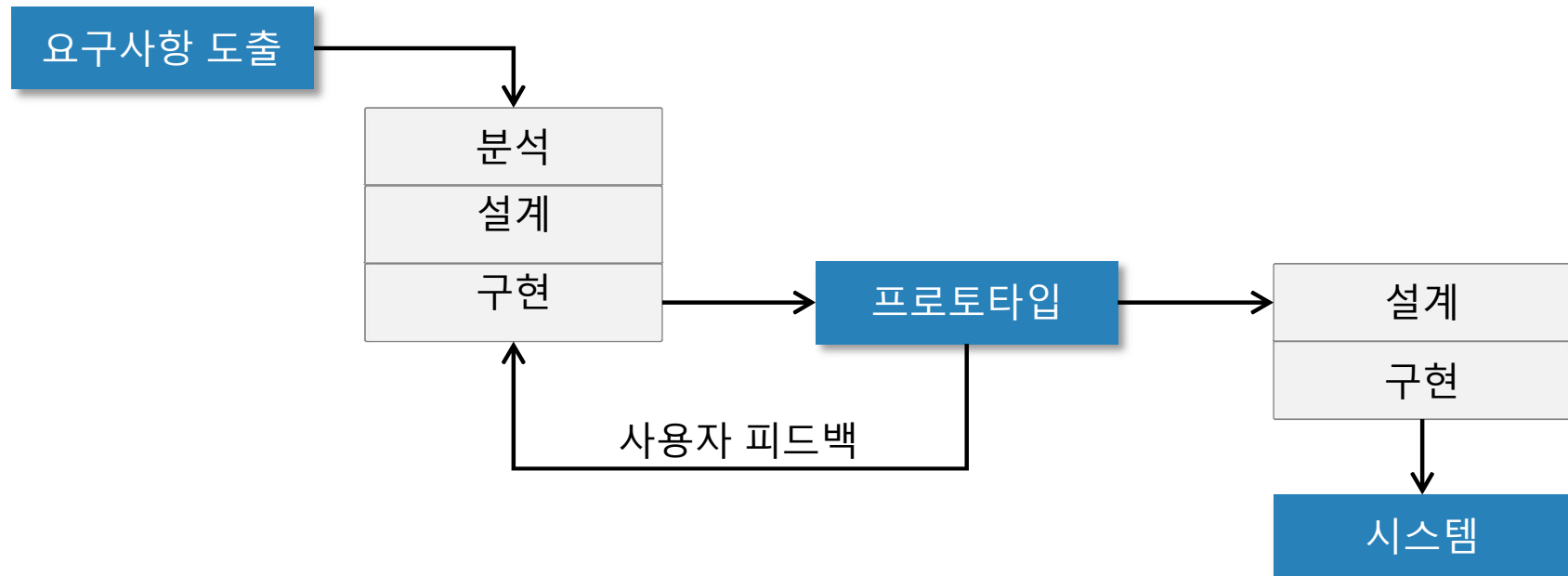
소프트웨어 개발 프로세스

■ 정보공학 방법론 – 프로토타입 모델(Prototype Model)

- 기능 메뉴(혹은 버튼)만 포함하는 사용자 인터페이스 원형만 제공
- 핵심 모듈만 우선적으로 개발하여 사용자에게 제공
 - 이를 통해 사용자의 요구사항을 만족했는지 여부를 판단
 - 이를 바탕으로 최종 시스템을 구현
- 요구사항을 검증하기 위해 프로토타입 개발
 - 폭포수 모델에 따라 전체 시스템을 개발

소프트웨어 개발 프로세스

■ 정보공학 방법론 – 프로토타입 모델(Prototype Model)



소프트웨어 개발 프로세스

■ 정보공학 방법론 – 프로토타입 모델(Prototype Model)

- 장점

- 사용자 요구사항을 검증하여 개발 비용과 시간을 단축할 수 있음
- 프로토타입을 통해 사용자와 개발자, 개발자 간 의사소통
- 상호 동일한 개념을 확보할 수 있음
- 소프트웨어를 개발하는 동안 빠른 시점에서 오류 탐지가 가능

- 단점

- 사용자 검증 과정 이후에 변경이 발생하는 부분을 고려하지 못함

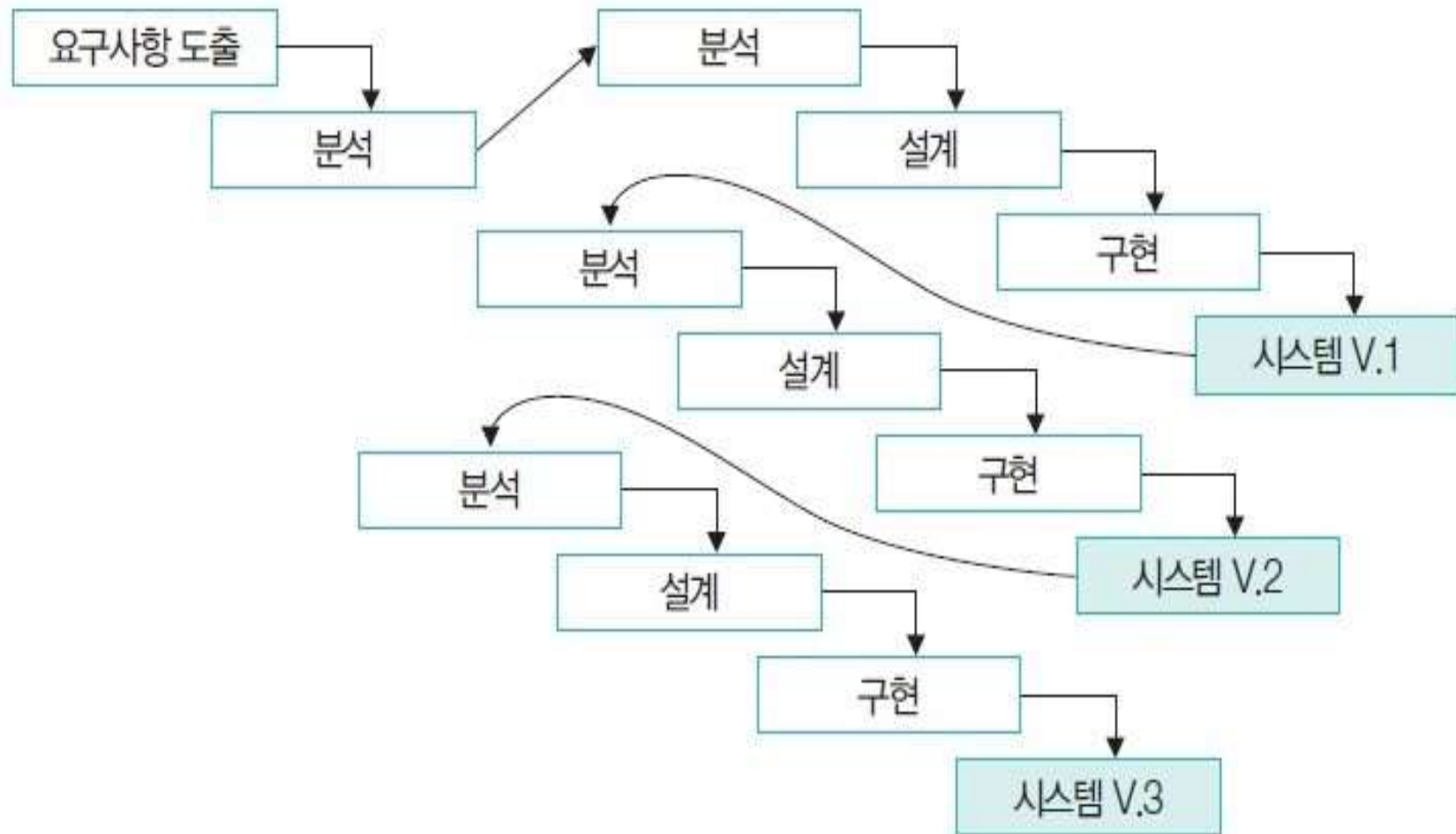
소프트웨어 개발 프로세스

■ 점진적 모델(Incremental Model)

- 제품이 완성 될 때까지 점진적으로 보완되고 조금씩 추가되어 테스트
- 개발과 유지 관리가 모두 포함
- 제품은 모든 요구 사항을 충족 할 때 완제품으로 정의
- **폭포 모델의 요소와 프로토타이핑의 반복 철학을 결합**
 - 개발 및 배포 ➔ 실사용자가 원하는 것 개발하여 제공
 - 측정 및 모니터링 ➔ 배포된 부분에 대한 사용자 유용성을 평가
 - 조정 및 수정 ➔ 모니터링 결과를 설계와 구현에 반영

소프트웨어 개발 프로세스

- 점진적 모델(Incremental Model)



점진적 모델에 의한 소프트웨어 개발 프로세스

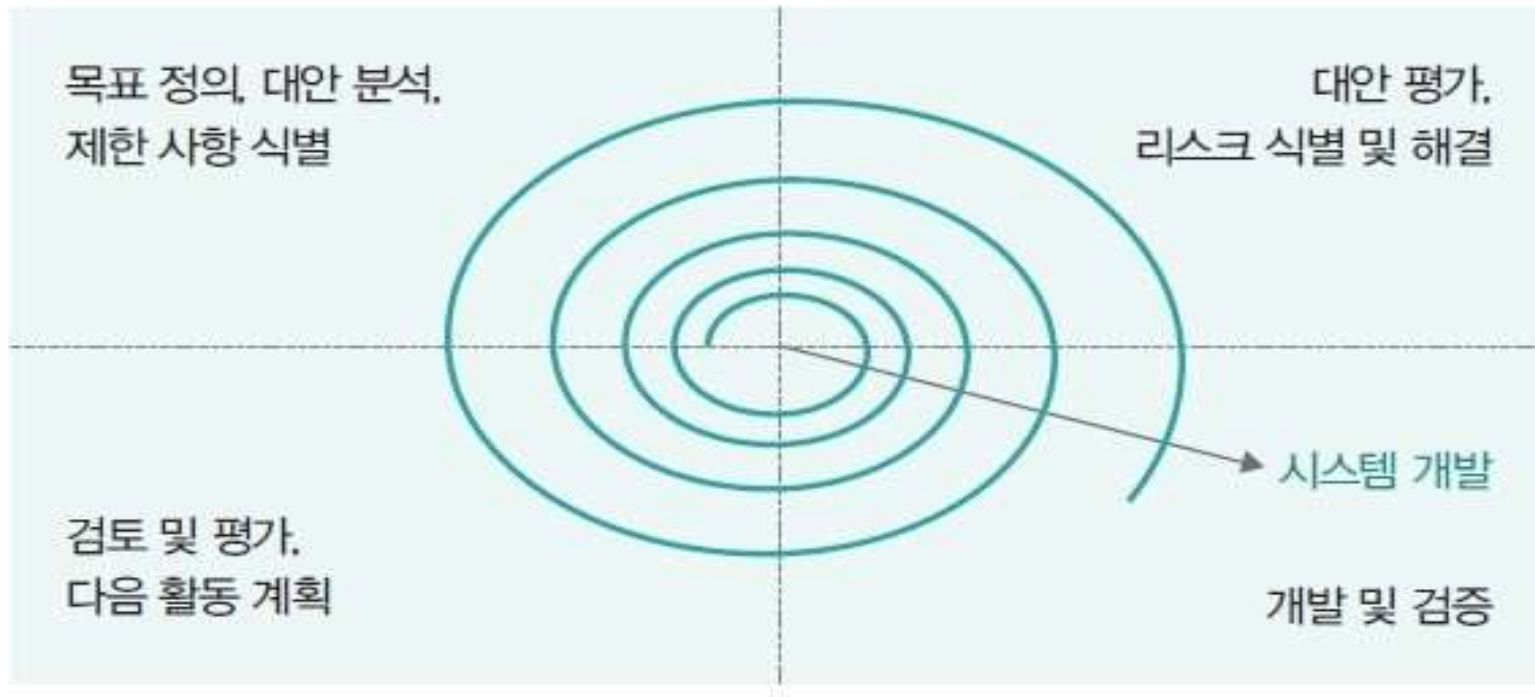
소프트웨어 개발 프로세스

■ 프로토타입 모델 기반 - 나선형 모델 (Spiral Model)

- 소프트웨어 시스템 개발 시 위험을 최소화
- 점진적으로 전체 시스템으로 개발해 나가는 모델
- 소프트웨어 개발 과정이 반복적이고 점진적으로 진행되는 나선 모양
- 목표 설정→위험 분석→구현 및 테스트→평가 및 다음 단계 수립의 활동 반복

소프트웨어 개발 프로세스

- 프로토타입 모델 기반 - 나선형 모델 (Spiral Model)



나선형 모델에 의한 개발 프로세스

소프트웨어 개발 프로세스

■ 프로토타입 모델 기반 - 나선형 모델(Spiral Model)

• 장점

- 체계적인 위험 관리로 위험성이 큰 프로젝트를 수행하기에 적합
- 사용자 요구사항을 더욱 상세히 적용할 수 있음
- 변경되는 요구사항을 반영하기 쉽다.
- 최종 개발된 소프트웨어 시스템에 대한 사용자 만족도와 품질이 높아짐

• 단점

- 프로젝트 기간이 길어짐
- 반복되는 사이클이 많아질수록 프로젝트 관리가 어려움
- 위험 분석에 따른 비용이 발생하고 위험 관리를 위한 전문적 지식이 요구됨

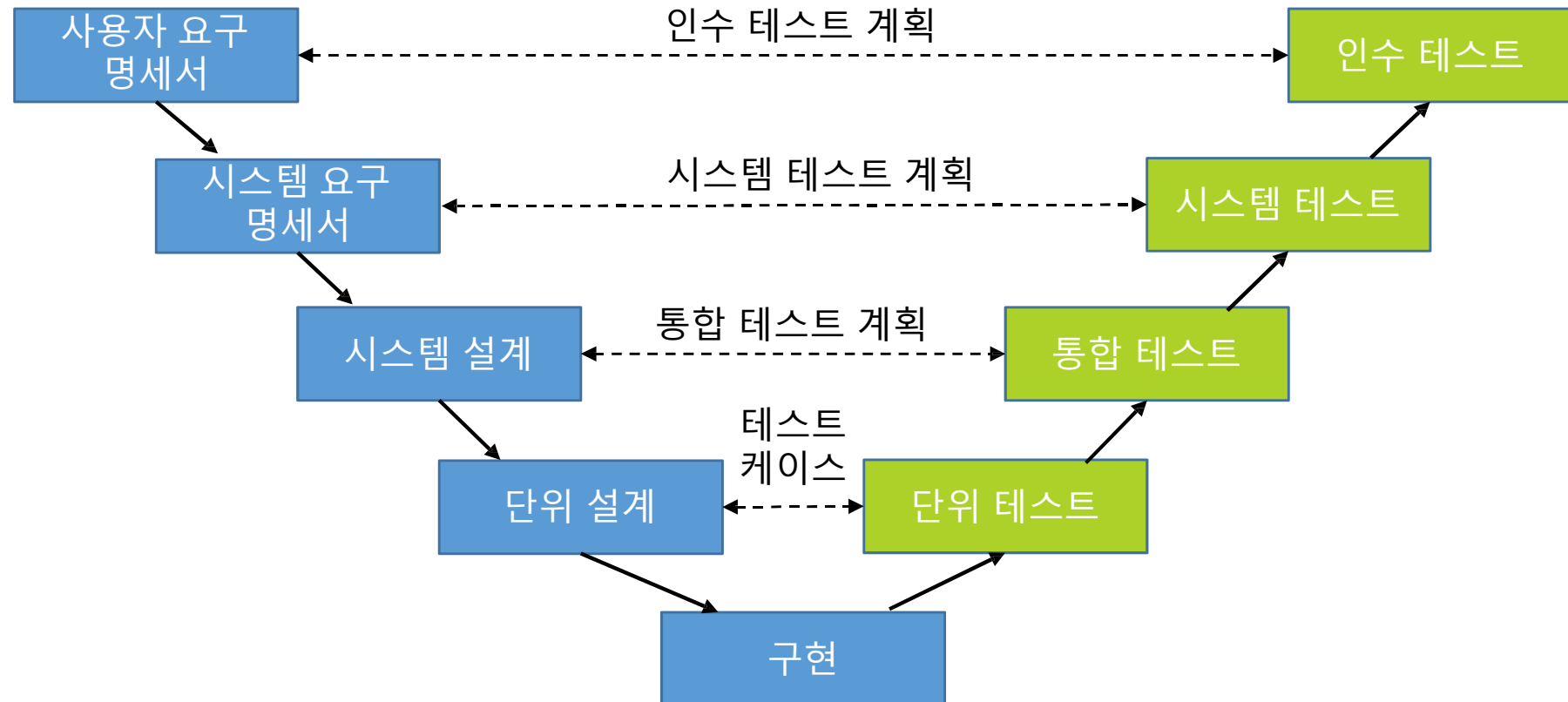
소프트웨어 개발 프로세스

■ 폭포수 모델 기반 – V 모델 (V Model)

- 폭포수 모델의 변형
 - 테스트 단계를 추가 확장
 - 테스트 단계와 분석 및 설계와의 관련성을 나타냄
- 폭포수 모델은 산출물 중심
- V 모델은 **각 개발 단계를 검증하는데 초점을 두어 오류를 줄임**

소프트웨어 개발 프로세스

■ 폭포수 모델 기반 - V 모델 (V Model)



소프트웨어 개발 프로세스

■ 폭포수 모델 기반 - V 모델 (V Model)

- 장점

- 소프트웨어 개발 결과물에 대한 단계적인 검증과 확인 과정 거침
 - 개발 소프트웨어에 대한 오류를 줄일 수 있음
- 요구사항이 정확히 이해되는 작은 시스템 개발 시 매우 유용

- 단점

- 폭포수 모델 적용함으로써 개발 활동에 대한 반복 허용되지 않음
- 요구사항이 변경되는 경우, 처리하기 어려움

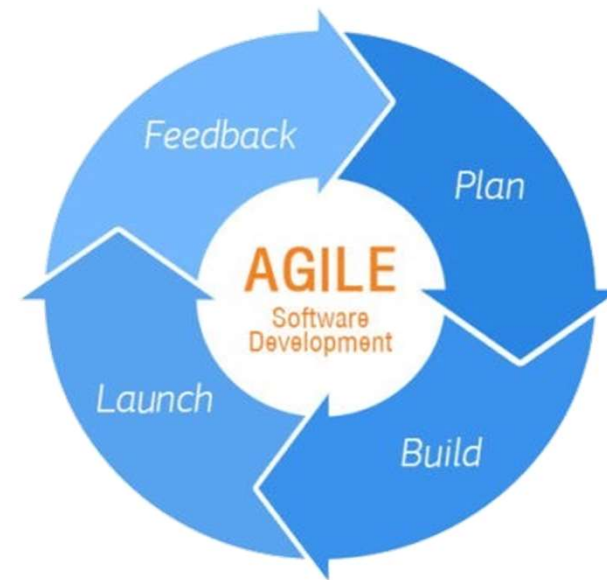
애자일 프로세스 모델

애자일 프로세스 모델

- 애자일(agile)
 - '날렵한', '민첩한'
- 애자일 프로세스 모델
 - 고객 요구 민첩하게 대응하고 그때 그때 주어진 문제 풀어나가는 방법론
 - 문서화 중심의 소프트웨어 개발접근방식 넘어 코드 중심 개발 접근 방법
 - 특정 개발 방법론을 가리키는 말이 아님
 - Agile 개발을 가능하게 해주는 방법론 전체를 일컫는 말

애자일 프로세스 모델

- 애자일의 기본 가치 (애자일 선언문)
 - 프로세스와 도구 중심이 아닌, **개개인과의 상호 소통** 중시
 - 문서 중심이 아닌, **실행 가능한 소프트웨어** 중시
 - 계약과 협상 중심이 아닌, **고객과의 협력** 중시
 - 계획 중심이 아닌, **변화에 대한 민첩한 대응** 중시



애자일 방법과 폭포수 모델 비교

구분	폭포수 모델	애자일 방법론
추가 요구 사항 수용	추가 요구 사항을 반영하기 어려운 구조	추가 요구 사항을 수용
릴리스 시점	최종 완성된 제품을 릴리스	수시
시작 상태	시작 단계에서의 완성도가 높음	시작 단계는 미흡하나 점차 완성도가 높아짐
고객과의 의사 소통	사용자와 산출물의 근거 중심	처음부터 사용자의 참여 유도 대화를 통해 개발 진행
진행 상황 점검	단계별 산출물에 대한 결과로 개발의 진척 상황 점검	개발자와 사용자는 개발 초기부터 진행 상황 공유
분석/설계/구현 진행 과정	분석/설계/구현 과정이 명확	하나의 단계안에 분석/설계/구현 과정이 모두 포함되어 동시에 진행
모듈(컴포넌트) 통합	구현이 완료된 후에 모듈 간의 통합 작업 수행	개발 초기부터 빈번한 통합 문제점을 빨리 발견하고 수정

애자일 프로세스 모델 이해

■ 애자일의 적용 원리

- 최우선적인 목표 → 고객 만족 위해 가치 있는 소프트웨어 빨리, 지속적 제공
- 개발 후반에 새로 추가되는 요구사항도 기꺼이 받아들임
- 실행 가능한 소프트웨어를 1~2주일에 한 번씩 사용자에게 배포
- 사용자와 개발자 간 친밀한 미팅을 거의 매일 수행
- 의사소통을 위해 가능하면 한 곳에 모여 대화
- 진척 상황 척도 표현 방법 : 실행 가능 소프트웨어 보여주는 것
- 일정한 개발 속도를 유지하며, 지속 가능하도록 소프트웨어를 개발
- 작업을 가장 단순화 형태로 분리하여 고민하고 해결
- 팀은 더 효과적인 방법의 적용, 적절한 조정 과정을 정기적으로 수행

애자일 방법론

■ 대표적인 애자일 방법론

- 동적 시스템 개발 방법 (Dynamic System Development Method)
- 적응형 소프트웨어 개발 (Adaptive Software Development)
- 크리스탈 패밀리 (Crystal Family)
- 익스트림 프로그래밍 (XP)
- 스크럼 (Scrum)
- 린 소프트웨어 개발 (Lean Software Development)
- 기능 주도 개발 (Feature-Driven Development)
- 애자일 유폴 (Agile Unified Process)

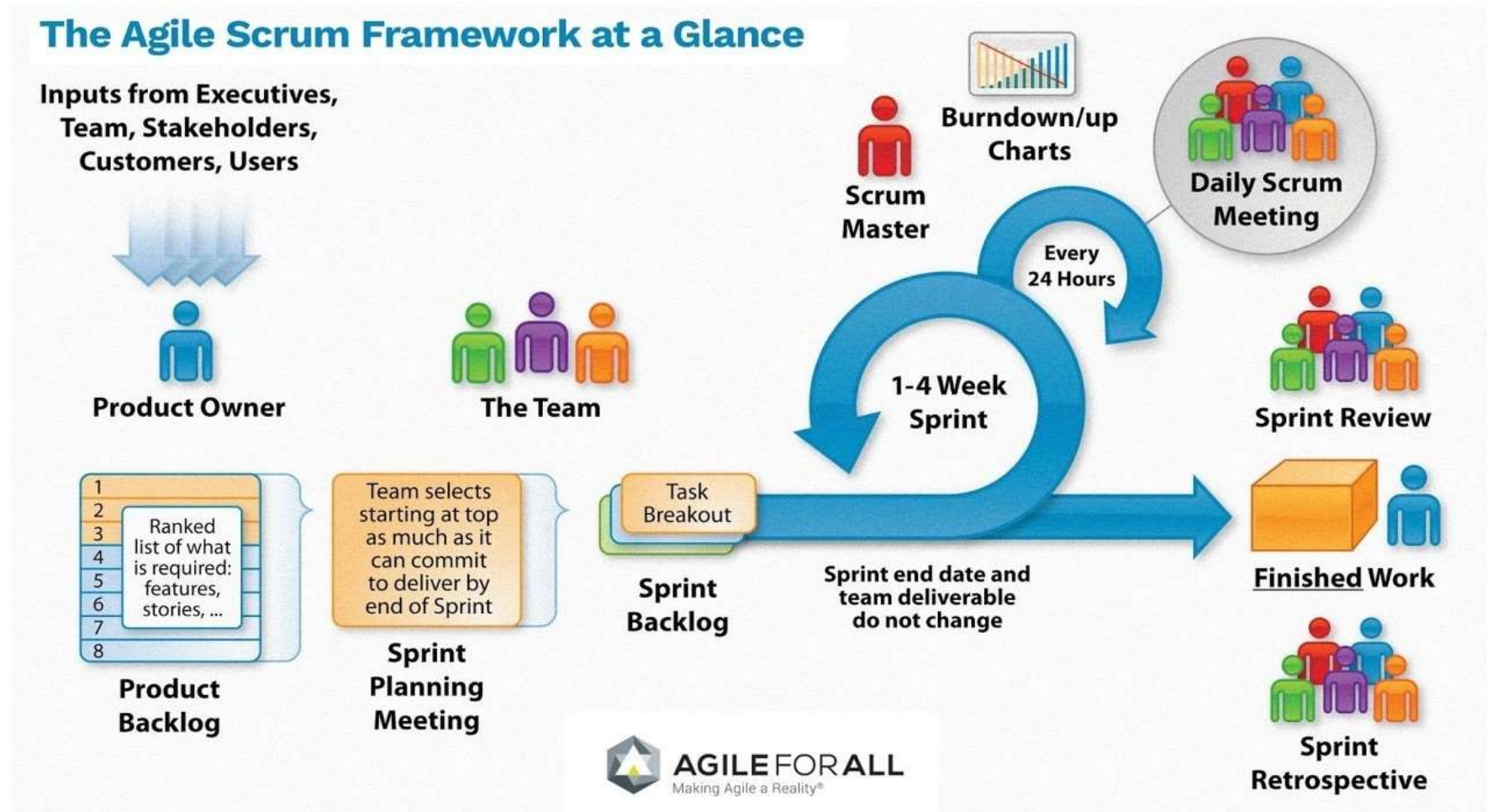
애자일 프로세스 모델: 스크럼

■ 스크럼 방식의 주요 개념

- 스크럼 (scrum)
 - 럭비 경기에서 사용하는 용어로 반칙으로 인해 경기가 중단되었을 때 사용하는 대형
- 스크럼 팀(scrum team)
 - 주도적인 팀 활동을 강조
 - 팀이 경험을 통해 배우고, 문제를 해결하면서 지속적으로 개선하도록 유도
 - **스프린트(sprint)**라고 불리는 업무 주기를 반복

애자일 프로세스 모델: 스크럼

■ 스크럼 방식의 주요 개념



애자일 프로세스 모델: 스크럼 프로세스 용어

■ 스크럼 프로세스 용어

- 스프린트 (sprint)
 - 반복적인 개발 주기(1~4주로 구성)
 - 하나의 스프린트가 끝나면 다음 스프린트가 시작됨
- 제품 백로그 (product backlog)
 - 개발할 제품에 대한 **요구사항을 우선 순위에 따라 나열한 목록**
 - 확정, 고정된 것이 아니라 사업 환경이나 변환에 따라 지속적으로 업데이트 됨
- 스프린트 백로그 (sprint backlog)
 - 하나의 스프린트 동안 완료할 **과제들의 목록**
- 제품 증분 (increment)
 - 매 스프린트에서 팀이 개발한 제품 기능
 - 스프린트 결과로 나오는 실행 가능한 제품

애자일 프로세스 모델: 스크럼 프로세스 용어

■ 스크럼 프로세스 용어

- 번다운 차트: 스프린트 백로그에 남아 있는 작업 목록을 보여주는 차트
- 번업 차트: 배포에 필요한 진행 사항(진척도)을 보여주는 차트
- 제품 책임자: 제품 백로그를 정의하여 우선순위를 정하는 책임자
- 스크럼 마스터 (scrum master): 프로젝트 관리자

프로젝트
PM
팀원..... 팀원
작업1 task1
작업2 task2

스크럼팀
스크럼마스터
팀원..... 팀원
작업1 sprint1 → backlog
작업2 sprint2

애자일 프로세스 모델: 스크럼 진행 과정

■ 스크럼 방식의 진행 과정

단계	수행 목록	내용
1	제품 기능 목록 작성	<ul style="list-style-type: none">요구사항에 우선 순위 선정하여 제품 기능 목록 작성
2	스프린트 계획 회의	<ul style="list-style-type: none">스프린트 구현 목록 작성스프린트 개발 기간 추정
3	스프린트 수행	<ul style="list-style-type: none">스프린트 개발일일 스크럼 회의스프린트 현황판 변경소셜 차트 표시
4	스프린트 개발 완료	<ul style="list-style-type: none">실행 가능한 최종 제품 생산
5	스프린트 완료 후	<ul style="list-style-type: none">스프린트 검토 회의스프린트 회고두 번째 스프린트 계획 회의

스크럼 방식의 이해: 제품 기능 목록

- 사용자 스토리 (요구사항 분석)
 - 제품 기능이 도출되면 각 기능을 간략히 기술
 - 한 장의 포스트 잇에 작성
 - 상세한 기능을 적는 것이 아닌 간략한 기능을 작성
- 사용자 스토리 작성 기준
 - 각 스토리는 서로 의존적이지 않아야 함
 - 사용자 스토리는 변경할 수 있음
 - 사용자가 이해할 수 있는 언어와 용어로 작성
 - 사용자 스토리를 보고, 개발 규모와 범위를 짐작할 수 있어야 함

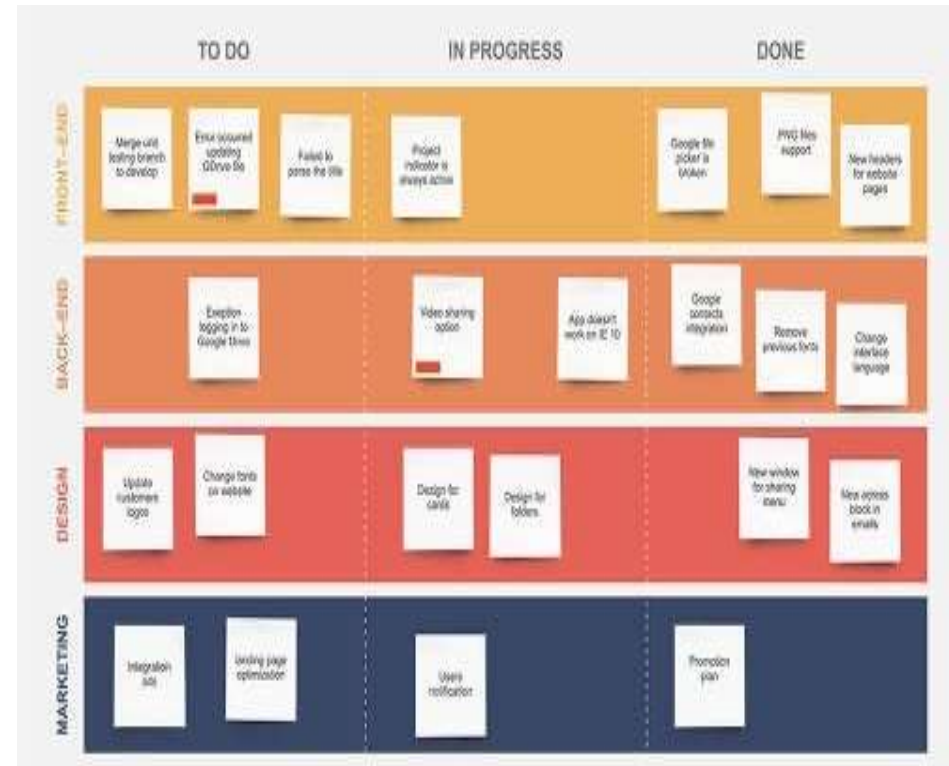
일일 스크럼 회의

■ 일일 스크럼 회의(daily scrum meeting)

- 스프린트 진행하는 동안 매일 정해진 장소와 시간에 모든 팀원이 참여하는 회의
- 매일 현재 상태를 업데이트하고 조율
- 일일 스크럼은 10~15분 정도 프로젝트 진행 상황을 공유
- 매일 각자가 어제 한 일, 오늘 할 일, **문제점/이슈 등을 공유**
- 한 장의 Post-it에 작성
- 매일 완료된 세부 항목을 완료 상태로 옮겨 현황판을 업데이트

일일 스크럼 회의

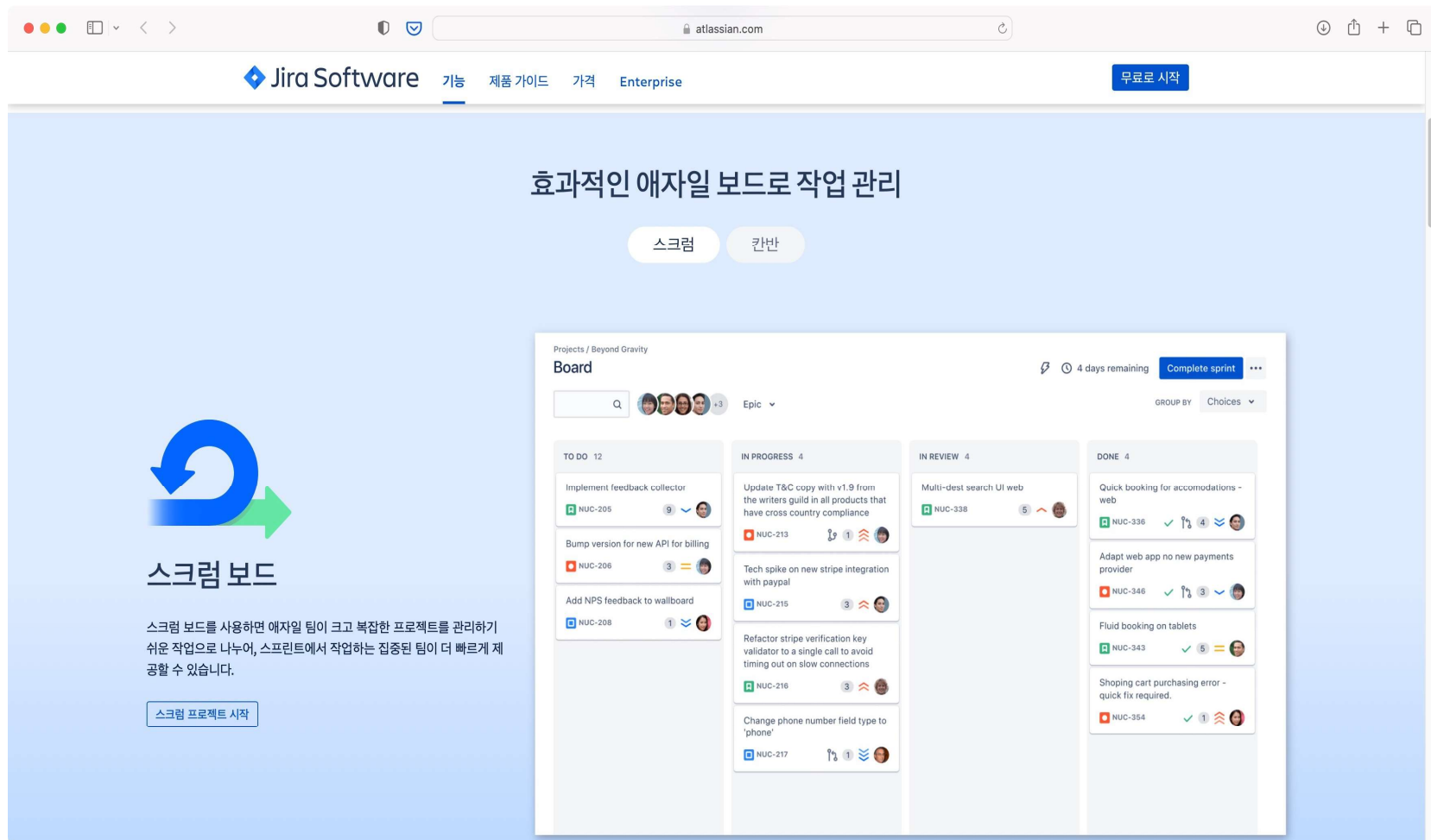
■ 스크럼 보드



스크럼 보드 앱 #1

- Jira software are

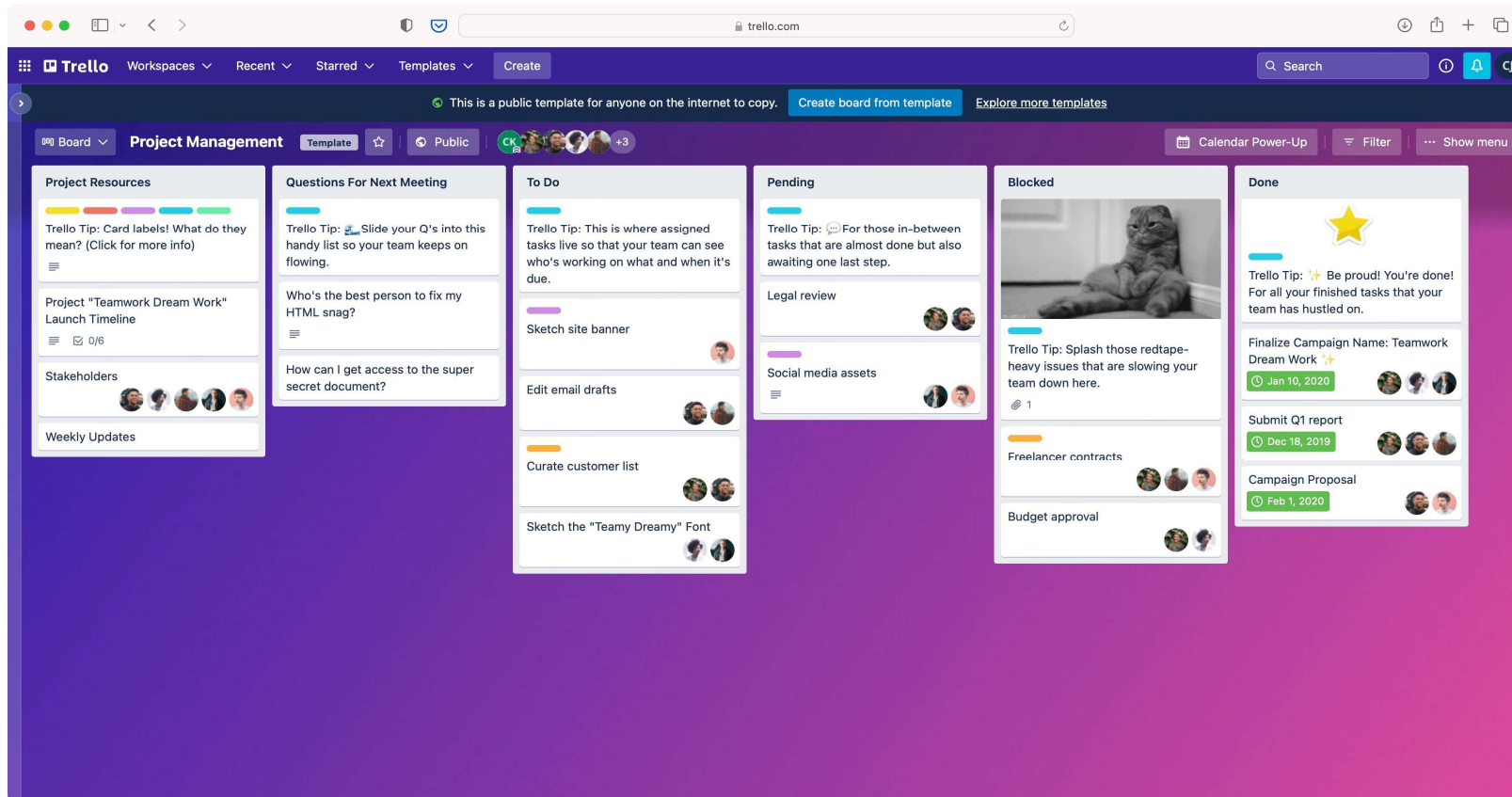
- <https://www.atlassian.com/ko/software/jira/features>



스크럼 보드 앱 #2

■ Trello

- <https://trello.com>
- Web 서비스 , 아이OS, 안드로이드 앱 제공

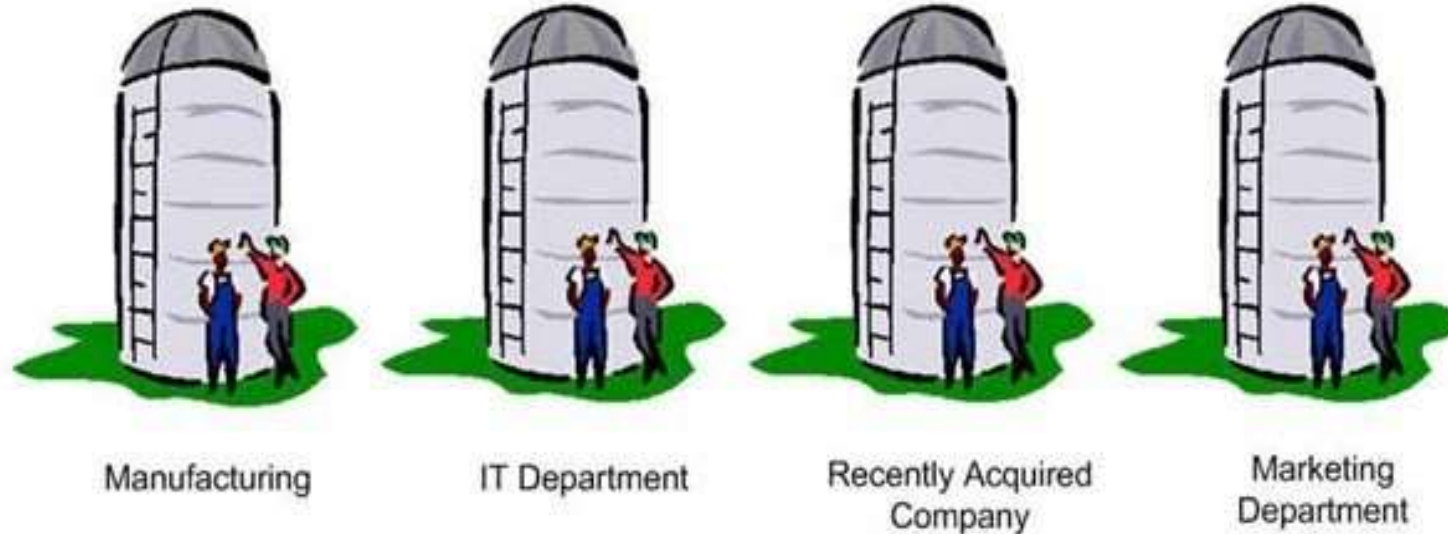


DevOps 개요

DevOps 개요

■ 사일로(Silo) 기반 개발 방식

- 소프트웨어 개발의 각 단계가 종료된 후 다음 단계로 진행
- 각 단계(부서)간 상호작용 부재
- 각각의 부서들이 서로서로 차단되어 통하지 못 하는 상황



DevOps 개요

■ DevOps란

- 개발(development)과 운영(operation)을 결합해 탄생한 개발 방법론
- 소프트웨어 개발자와 운영자 간의 소통과 협업, 통합 강조
- 개발, 운영, 품질 보증 부서, 서비스 제공자 상호 협력하여 소프트웨어 시스템
개발 및 배포
- 고객 피드백을 신속히 수행하여 제품 및 서비스의 신속한 개발 목표

DevOps 개요

■ 핵심 원리

- 지속적 개선
- 협업
- 지속적 테스트
- 지속적 통합과 지속적 배포
- 모니터링 및 피드백을 통한 개선



DevOps 개요

■ 적용 동향

- 고객 서비스 365일 24시간 내내 제공해야 되는 IT 조직에서 도입
- 아마존, 구글, 넷플릭스 등

표 4-1 주요 IT 서비스 기업의 소프트웨어 배포 현황

회사	배포 주기	배포 소요 시간	신뢰성	사용자 응답
아마존	23,000/day	minutes	high	high
구글	5,500/day	minutes	high	high
넷플릭스	500/day	minutes	high	high
페이스북	1/day	hours	high	high
트위터	3/week	hours	high	high
일반 회사	1/9 months	months/quarters	low/medium	low/medium

DevOps와 애자일 방법 비교

■ 유사점

- 주기적인 릴리스 일정

■ 차이점

- DevOps

- production, pre-production, 출시, 출시 후 지원 모두 포함 포괄적 프로세스
- 다양한 부서를 포함

- 애자일

- 개발자 work flow에 집중
- production에 중점을 둠

DevOps 프로세스

■ 계획 단계

- 애플리케이션에 대한 **비즈니스 가치**와 **사용자 요구사항** 정의
 - 요구사항 명세
 - 비즈니스 사례 정의, 배포 계획, 배포 시기 등

■ 코딩 단계

- 소프트웨어 설계
- 단위 테스트 수행

■ 빌드 단계

- 작성된 코드의 통합, 통합 테스트
- 애플리케이션 버전 관리

DevOps 프로세스

- 테스트 단계
 - 요구사항 만족도 테스트
 - 변경, 개선 사항 발생시 회귀 테스트
 - 애플리케이션 보안 및 취약성 분석
 - 성능 테스트

- 릴리즈 단계
 - 배포 승인
 - 애플리케이션 배포

DevOps 프로세스

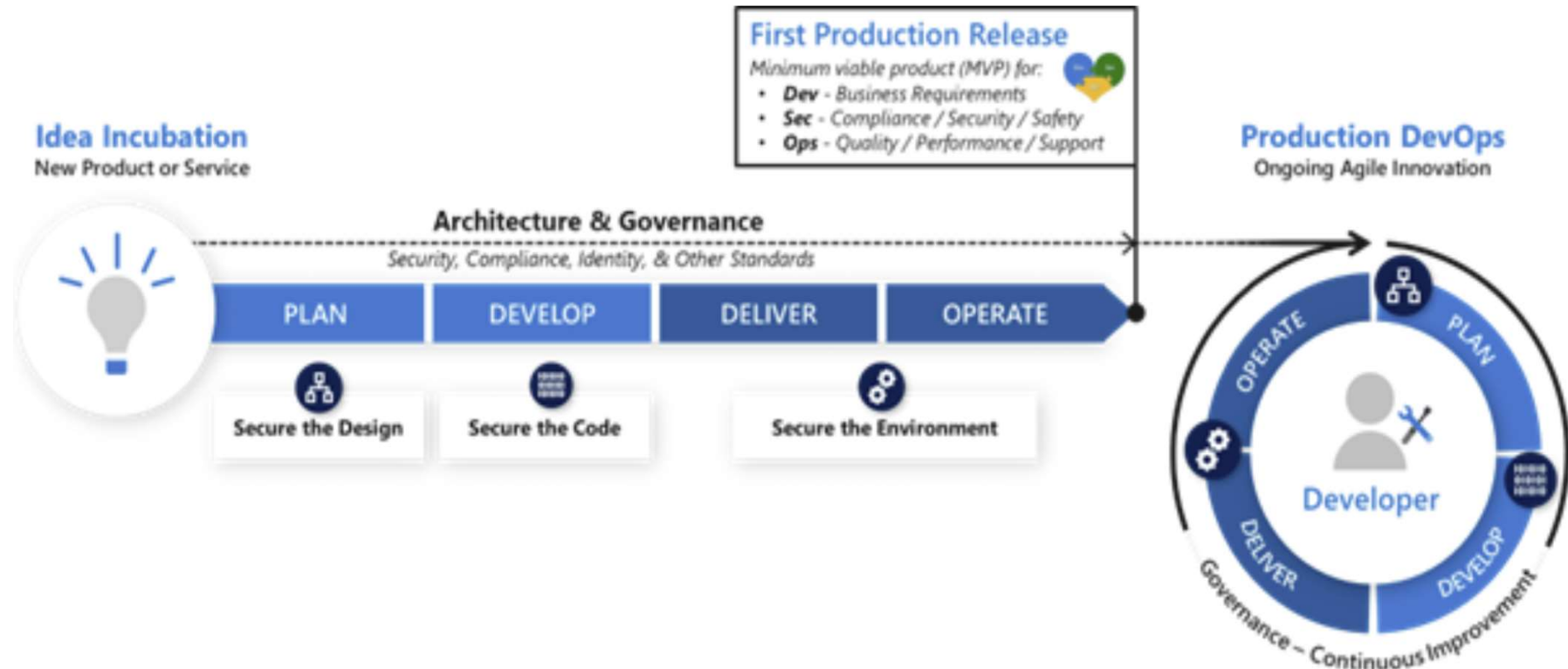
■ 설치 단계

- 사용자 환경에 배포된 애플리케이션 설치
- 인프라 구축
- 설치 과정에서 발생한 문제 복구

■ 운영 및 모니터링 단계

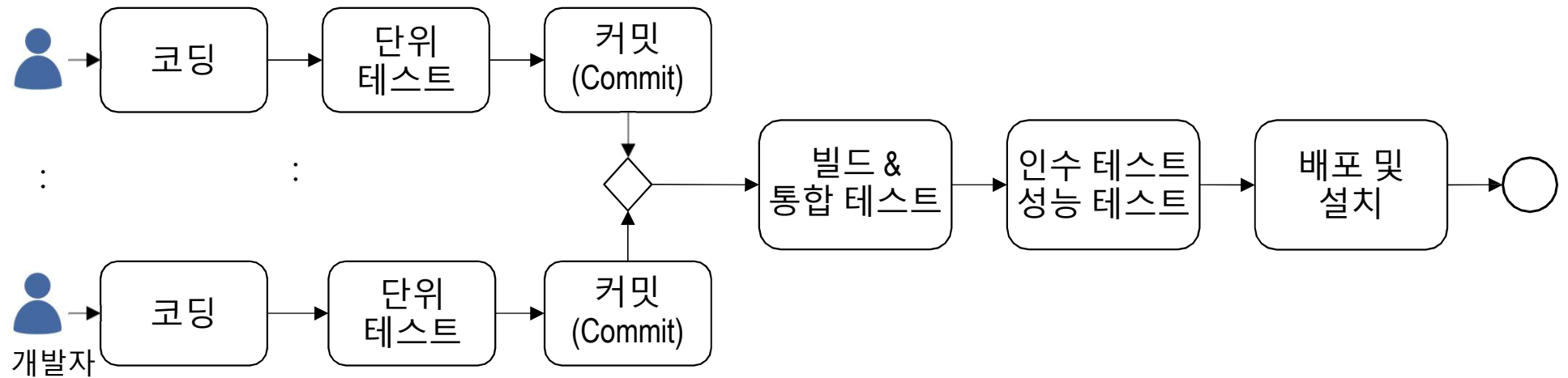
- 지속적인 모니터링
- 사용자 반응 및 경험 추적
- 개발 척도의 달성 여부 확인
- 모니터링 결과를 개발팀에 제공

DevOps 프로세스



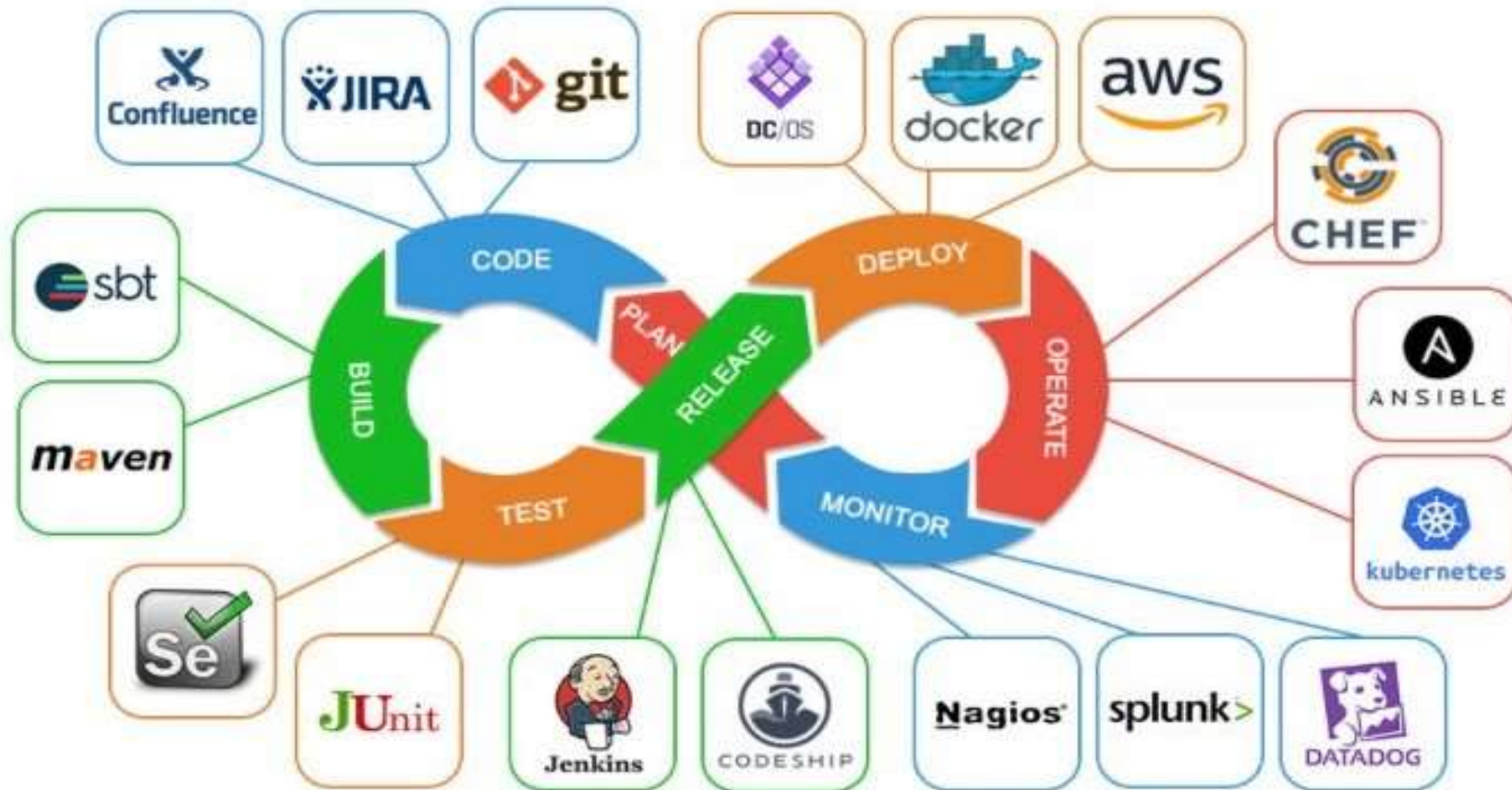
DevOps 프로세스

- DevOps의 In-House(사내) 서비스 개발 단계



DevOps 프로세스

- DevOps의 In-House(사내) 서비스 개발 단계



DevOps toolchains

■ 툴체인(toolchain)

- A set of tools for software development
- 소프트웨어 개발에 필요한 도구 모음

단 계	단계별 활용 가능한 자동화 도구
계 획	Jira, Trello, Asana, Confluence, Azure etc.
코 디ング	Git, Github, Bitbucket, eclipse IDE, Artifactory etc.
빌 드	Maven, Gradle, Jenkins, TravisCI, Apache ANT etc.
테 스트	Junit, nunit, Jmeter, Selenium etc.
릴 리즈	Red Hat, Puppet, openstack, Chef, Release, Spinnaker, Bamboo, apigee, MS Azure, AWS, JFrog etc.
설 치	docker, Virtualbox, Automtic, uDeploy, Jenkins, TFS, Kubertens, Elasticboc, bmchelix, LXD etc.
운 영	Puppet, chef, Fortify, Veracode, Flyway, MongoDB, PowerShell, Ravello etc.
모니터링 & 피드백	Dynatrace, APPDYNAMICS, elastic, splunk, New Relic, sensu etc.

프로젝트 개발 단계별 산출물

■ 계획 단계

- 프로젝트 계획서

■ 요구사항 분석 단계

- WBS(Work Breakdown Structure): MS-Project, Excel, Asana 등
- 요구 사항 정의서

■ 설계 단계

- 기능 설계 문서
 - 화면정의서, 메뉴 구성
 - 테이블 정의서, 클래스 다이어그램 등

프로젝트 개발 단계별 산출물

- 테스트 단계
 - 테스트 계획서
 - 테스트 시나리오
- 완료
 - 프로젝트 최종 보고서

프로젝트 방법론

요구사항 분석

목 차

- 요구사항 개요
- 요구사항 수집 기법
- 요구사항 정의서 작성

요구사항의 특징 및 요구 사항 분석

■ 요구사항 정의

• 사전적 정의

- ‘이용자가 어떤 문제를 풀거나 목표를 달성하는 데 필요한 조건이나 능력’

• 소프트웨어 개발에서의 정의

- ‘사용자와 개발자가 합의한 범위 내에서 사용자가 필요로 하는 기능’
- 시스템이 제공하는 기능 요구와 품질과 같은 비기능 요구로 나뉨
- 요구사항이 정확히 무엇인지 파악하는 작업은 요구분석 단계에서 이루어짐

요구사항의 특징 및 요구 사항 분석

■ 요구사항 관련 특성

- 소프트웨어 개발 수명 주기에서 **가장 중요한 요소**
- **소프트웨어 개발 기준**
- **상세한 요구사항 목록으로 구체화**해야 됨
- **기능적 요구 사항**과 **비기능적 요구 사항**으로 구성
- 요구사항 문제점 늦은 단계 발견: 수정하기 위해 많은 비용 발생
- **명확하지 않은 요구사항 정의: 프로젝트 실패** 가장 중요한 원인

요구 분석의 정의 및 목적

■ 요구 분석 정의

- 컴퓨터 용어사전:
 - '시스템이나 소프트웨어의 요구사항 정의하기 위해 사용자 요구사항 조사하고 확인 하는 과정

■ 요구 분석의 목적

- 사용자에게 필요한 요구 사항 추출 -> **요구 사항 명세서 작성**
- **구현 가능 여부**에 대한 논의
- 프로젝트 전체 규모 파악
- 프로젝트 일정 수립

요구 분석의 정의 및 목적

- **요구 사항 명세서 포함 내용**

- ➔ 요구 분석 명세서, 요구 사항 정의서

- 요구분석 단계에서 생성되는 산출물
 - 기술적 요구와 제약 조건
 - 시스템 설계시 참조할 사항
 - 개발자와 사용자가 합의한 성능에 관한 사항 등

요구 분석 절차와 요구사항 종류

■ 요구 분석 절차



자료수집

현행 시스템 문제점 도출, 실무 담당자 인터뷰, 설문 조사, 현재 사용 문서 검토

요구사항도출

수집한 자료를 정리해 적절히 분류하고 개발에 반영할 요구사항을 도출

문서화

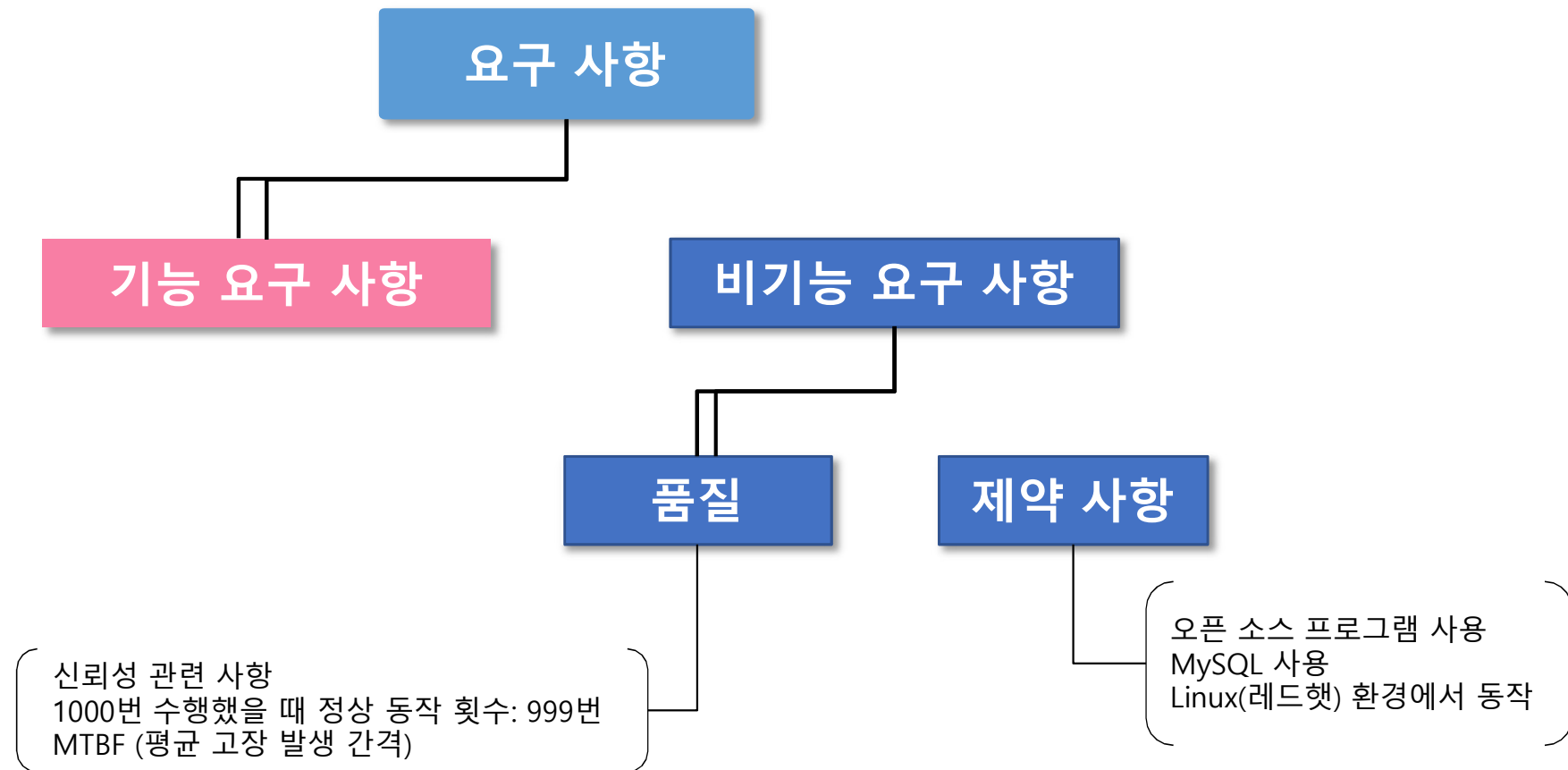
도출한 요구사항을 **요구사항 명세서**로 작성

검증

사용자 요구가 정확히 기록되어 있는지, 모순되는 사항은 없는지 점검

요구 분석 절차와 요구 사항 종류

■ 요구 사항 분류



요구사항 정의서

■ 요구사항 정의서

- 사용자의 요구 사항과 개발 관련 요구 사항을 정의한 문서
 - 개발자가 해당 문서를 기반으로 설계 및 코딩하며 검증 기준으로 사용

■ 작성시 주의할 점

- 모든 요구 사항에 고유한 식별자를 부여
- 각 요구 사항은 그 의미를 정확히 표현
- 요구 사항을 정의하는 문장은 **단문으로 작성**
- 요구 사항에 **우선 순위를 부여**
- 요구 사항을 **그룹화하여 분류 (체계화)**

요구사항 명세서

■ 요구사항 명세서

요구사항 명세서					
구분	서비스(메뉴)	기능명	기능설명	우선순위	비고
Common	1. Regist	1.1 이메일 회원가입	이메일을 사용한 회원가입 (이메일 인증 필요) [이메일], [비밀번호], [이름], [국적], [직업], [닉네임], [핸드폰] 등 입력	1	
		1.2 SNS 회원가입	SNS를 이용한 회원가입 연동	2	
	2. Login	2.1 이메일 로그인	[이메일], [비밀번호]를 입력하여 로그인	1	
		2.2 SNS 로그인	SNS를 이용한 로그인 연동	1	
		2.3 자동 로그인	자동 로그인 체크/해제	3	
	3. User Info	3.1 회원 정보 관리	닉네임 수정, 비밀번호 변경 등		
		3.2 계정/비밀번호 찾기	이메일 계정/비밀번호 찾기		
	Common	CS 상담	Customer Service (사용자 채팅 상담)		
		북마크 즐겨찾기 추가	북마크 즐겨찾기 추가		
		북마크 즐겨찾기 수정	북마크 즐겨찾기 수정		
		북마크 즐겨찾기 삭제	북마크 즐겨찾기 삭제		
		북마크 즐겨찾기 공유	북마크 즐겨찾기 외부로 공유		

프로젝트 방법론

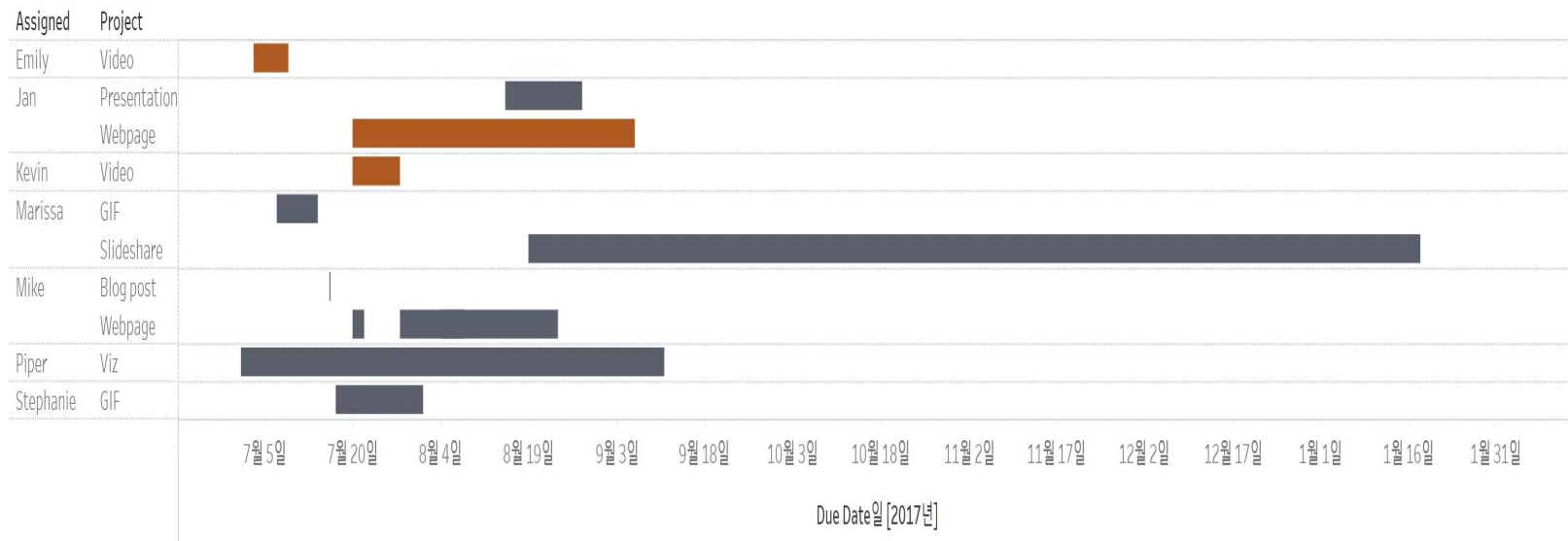
일정 관리 계획

프로젝트 일정 계획

■ 간트 차트(Gantt chart)

- 프로젝트 일정관리를 위한 바(bar)형태의 도구
- 각 업무별로 일정의 시작과 끝을 그래픽으로 표시
 - 전체 일정 한눈에 파악
 - 각 업무(activities) 사이의 관계 파악

Status



프로젝트 일정 계획

■ 간트 차트(Gantt chart)



프로젝트 일정 계획

■ 간트 차트(Gantt chart)

- 장점

- 프로젝트 타임라인의 전체상을 파악 가능

- 프로젝트의 로드맵

- 고위 경영진이나 고객에게 간략한 개요를 설명할 때 특히 유용

- 단점

- 만드는 데 시간이 많이 소요

프로젝트 일정 계획

■ 간트 차트(Gantt chart)

- 무료 소프트웨어 도구
 - <https://www.onlinegantt.com/#/gantt>
 - 엑셀
 - 테블로

프로젝트 방법론

업무분류체계서

업무분류체계서

■ WBS(Work Breakdown Structure)

- 프로젝트 작업 시 업무를 카테고리로 구분하고, 각각 카테고리는 세부적인 작업으로 나누어서, 일정 및 진행사항을 체크하는 기법
- 초기 프로젝트 제안서 작성할 때 함께 작성
 - 프로젝트 범위 체크
 - 스케줄에 따른 친척도, 업무 선후관계 파악
 - 업무별 담당 할당, 전체 진행 사항 시각화 및 도식화 빠른 이해

업무분류체계서

■ WBS(Work Breakdown Structure)

- 구성 요소

항목	설명
계정 코드(WBS ID)	WBS의 각 요소를 식별하기 위한 코드
주요업무	세부 업무를 묶는 그룹
세부업무	업무의 구체적인 내용
작업자	세부업무를 수행할 작업자 또는 파트를 작성
상태	업무 진행 사항을 작업중(In Progress), 완료(Completed), 확인(Milestone) 구분
시작일	업무의 시작일자
종료일	업무의 종료 일자
기간	업무의 시작과 종료 일지를 기준으로 작성
진척도	업무의 진척도를 %로 표기
일정차트	기간과 진척도를 반영하여 컬러로 표기
산출물	업무별 결과물 표기
WBS사전	WBS 각 요소 상세 정보 제공 문서

프로젝트 방법론

프로젝트 설계

목 차

- 소프트웨어 모델링
- UML
- ERD

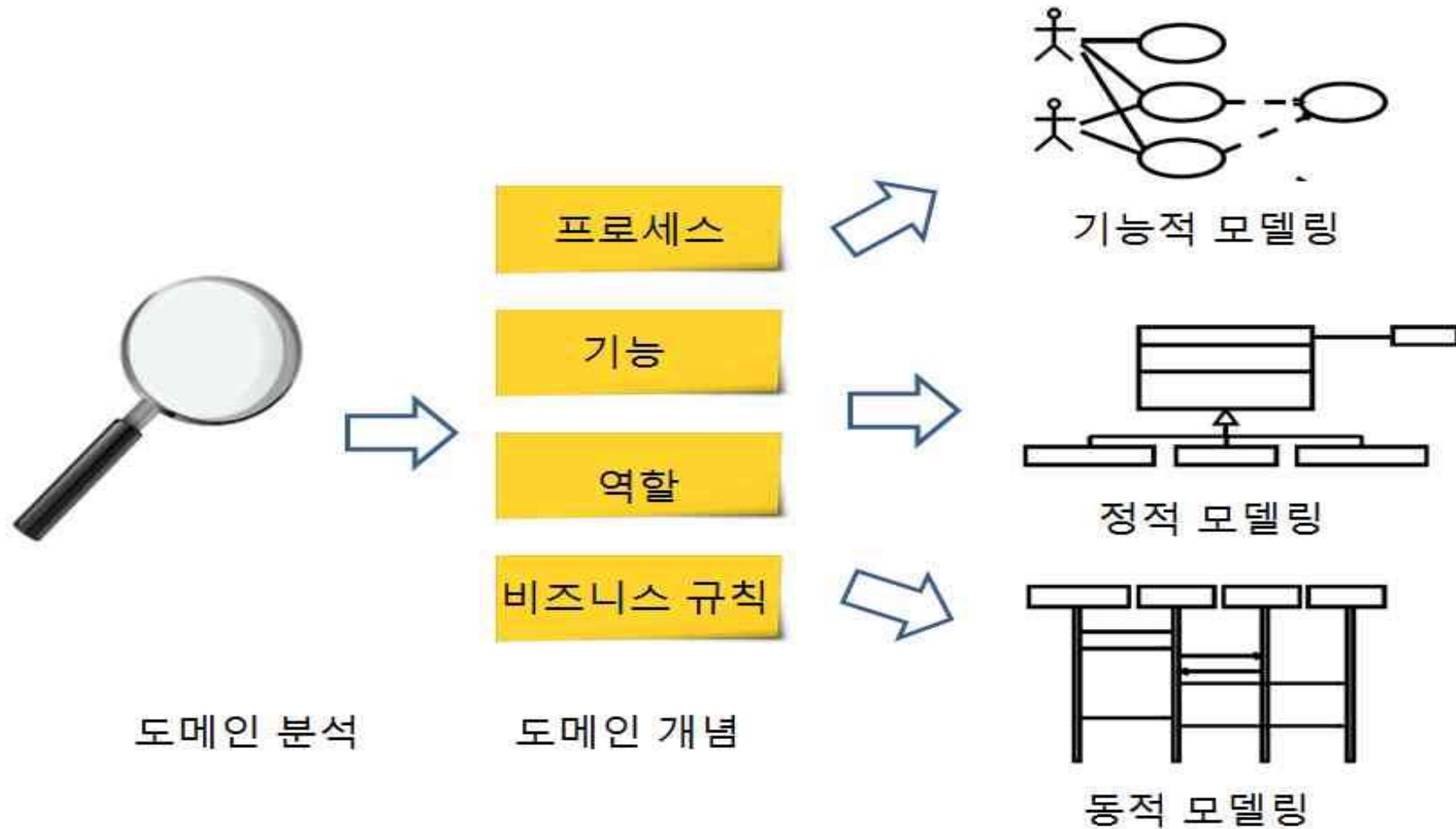
소프트웨어 모델링

■ 모델링(Modeling)

- 도메인 지식을 체계화하는 과정
- 중요한 도메인 개념과 특성, 관계를 파악하여 다이어그램으로 정형화
- 과거 - 자료 위주의 모델링
- 현재 - 객체지향 패러다임에서의 모델링
 - ➔ UML(Unified Modeling Language) 사용 표현

소프트웨어 모델링

■ 모델링(Modeling) 도식화



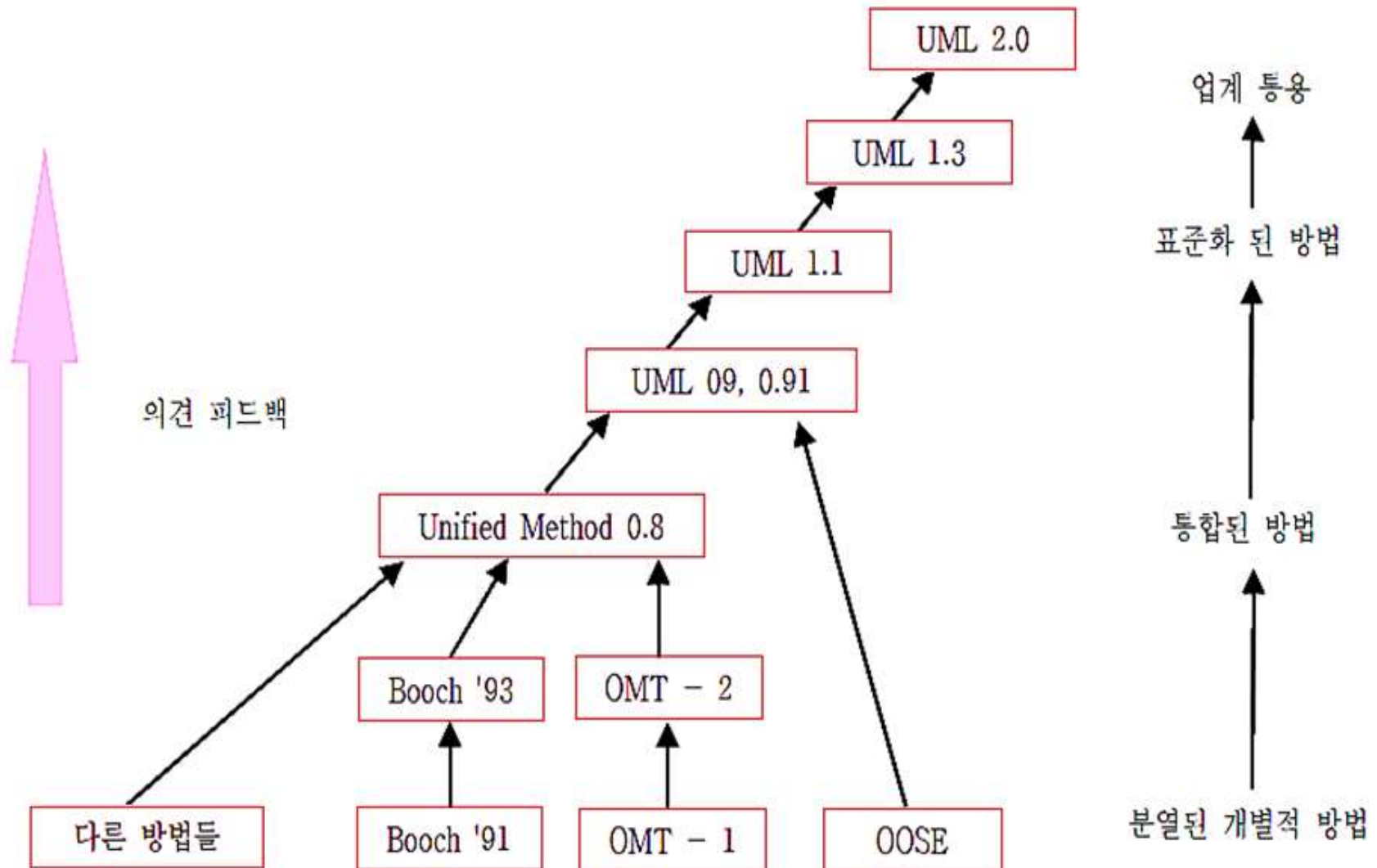
UML

■ UML이란?

- 객체지향 소프트웨어를 모델링 하는 (실질적) 표준 그래픽 언어
- 시스템의 여러 측면을 그림으로 모델링
- 하드웨어의 회로도 같은 의미
- OMT(Object Modeling Technique)[Rumbaugh, 1991]와 Booch[Booch,1994], OOSE(Object-Oriented Software Engineering)[Jackson, 1992] 방법의 통합으로 만들어진 표현

UML

■ UML이란?



UML

■ 시스템 모델링 구성

- 기능적 관점 ➔ **사용자 측면**에서 본 시스템 기능
요구 분석 단계의 사용 사례 다이어그램
- 구조적 관점 ➔ 시간 개념 포함되지 않은 정적 모델, 구조 표현
- 동적 관점 ➔ 시간 개념 포함되는 동적 모델, 상태, 순서 표현

UML

■ 모델 분류

- 정적 모델(Static Model)
 - 객체, 클래스, 속성, 연관, 패키지, 컴포넌트 등으로 시스템 구조 나타냄
 - 클래스 다이어그램, 패키지 다이어그램, 배치 다이어그램(하드웨어 배치)
- 동적 모델(Dynamic Model)
 - 변화, 흐름 등의 표현 위해 시스템 내부 동작 나타냄
 - 시퀀스 다이어그램, 상태 다이어그램, 액티비티 다이어그램

UML

■ UML 다이어그램 종류

● 구조적 다이어그램

- 소프트웨어 또는 시스템의 정적 구조, 추상화, 구현의 다양한 계층 표현
- 데이터베이스, 응용 프로그램처럼 시스템 구성하는 다양한 구조 시각화
- 구성 요소, 모듈이 서로 어떻게 연결되어 상호작용하는지 표현

- 클래스 다이어그램(Class Diagram): 클래스 명세와 클래스 간의 관계를 표현
- 객체 다이어그램(Object Diagram): 인스턴스 간의 연관 관계 표현
- 패키지 다이어그램(Package Diagram): 패키지 간의 연관 관계 표현
- 컴포넌트 다이어그램(Component Diagram): 파일과 데이터베이스, 프로세스와 프로세스 등의 소프트웨어 구조 표현

UML

■ UML 다이어그램 종류

• 행위 다이어그램

– 처음부터 끝까지 단계별 프로세스 표시

– 목표에 도달하기 위해서 반드시 이루어져야 하는 일단의 활동

– 한 가지 활동이 다음 활동으로 이어지는 방법, 활동이 어떻게 모두 연결되는지

– 소프트웨어 개발, 모든 비즈니스 환경 사용 ➔ 비즈니스 프로세스 매핑/모델링

■ **유스케이스 다이어그램(Use Case Diagram):** 시스템 제공 기능과 이용자 관계 표현

■ **액티비티 다이어그램(Activity Diagram):** 일련의 처리에 있어 제어의 흐름을 표현

■ **시퀀스 다이어그램(Sequence Diagram):** 인스턴스의 상호작용을 시계열로 표현

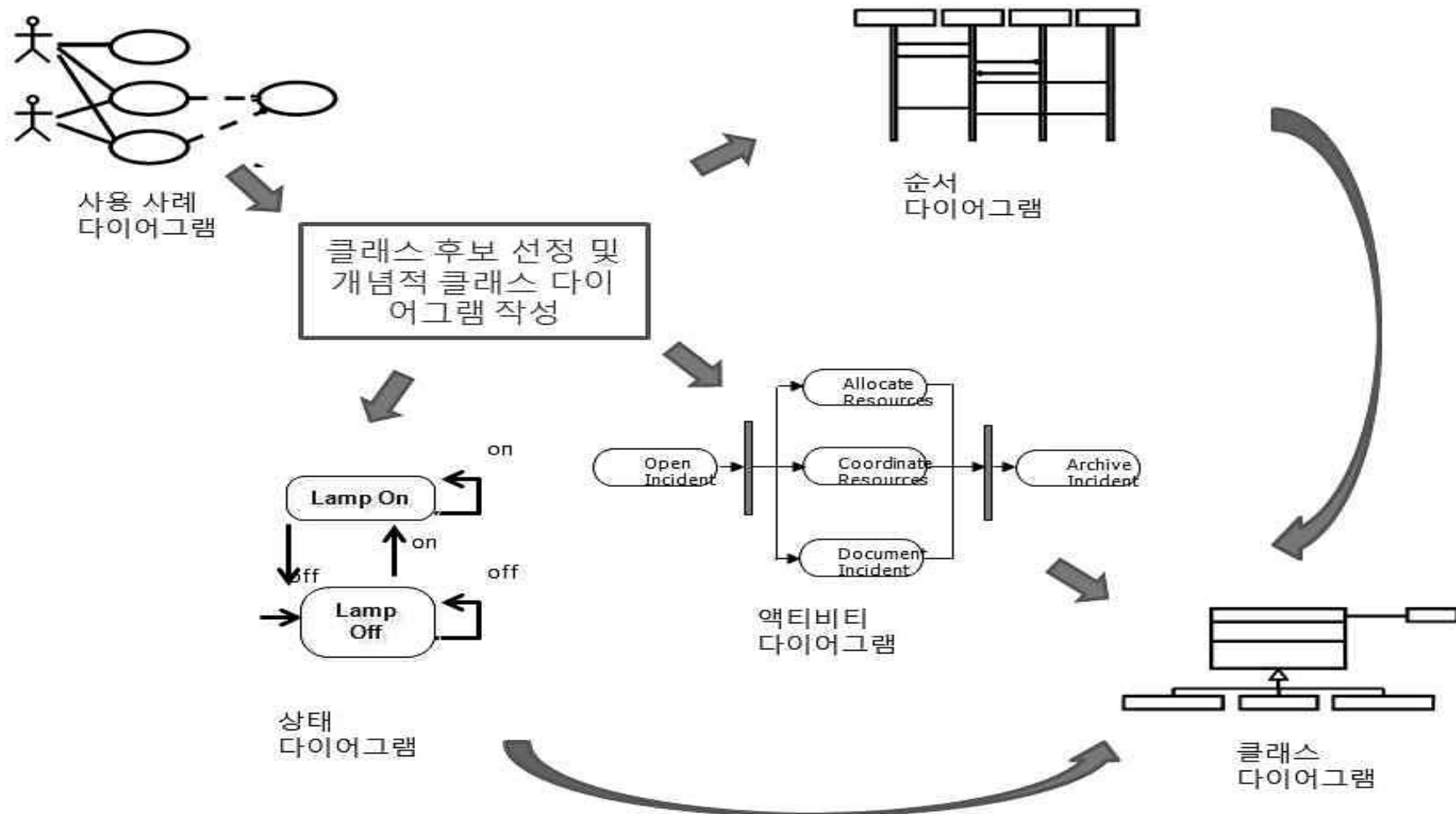
UML

■ UML 모델링 도시화 과정

- ① 요구를 사용 사례로 정리하고 **사용 사례 다이어그램 작성**
- ② 클래스 후보 찾아내고 개념적인 객체 모형 즉 **클래스 다이어그램 작성**
- ③ 사용 사례 기초하여 **시퀀스 다이어그램 작성**
- ④ 클래스 속성, 오퍼레이션 및 클래스 사이의 관계 찾아 객체 모형 완성
- ⑤ **상태 다이어그램이나 액티비티 다이어그램** 등 다른 다이어그램을 추가하여 **UML 모델을 완성**
- ⑥ 서브시스템을 파악하고 전체 시스템 구조를 설계
- ⑦ 적당한 객체를 찾아내거나 커스텀화 또는 객체를 새로 설계

UML

■ UML 모델링 도시화 과정



UML – Usecase Diagram

■ 유스케이스(Usecase diagram) 구성 요소

● 시스템과 사용자의 관계 표현

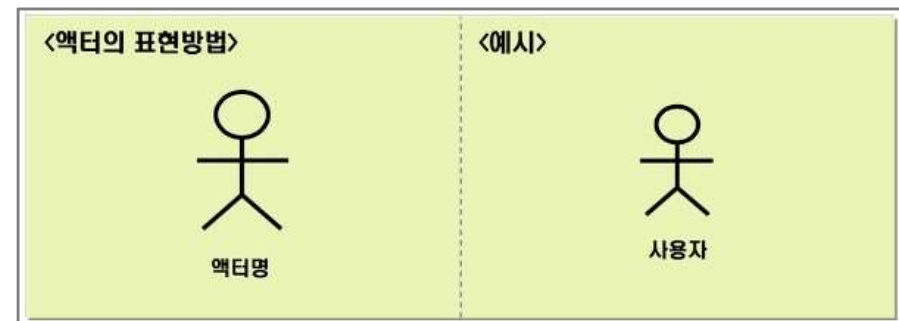
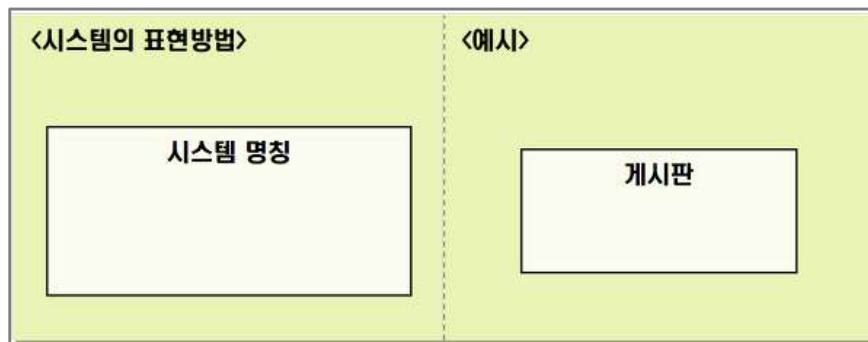
➤ **시스템**: 만들고자 하는 프로그램

– 사각형 틀로 시스템 명칭을 안쪽 상단에 작성

➤ **액터(Actor)**

– 시스템의 외부에 있고 시스템과 상호작용을 하는 사람

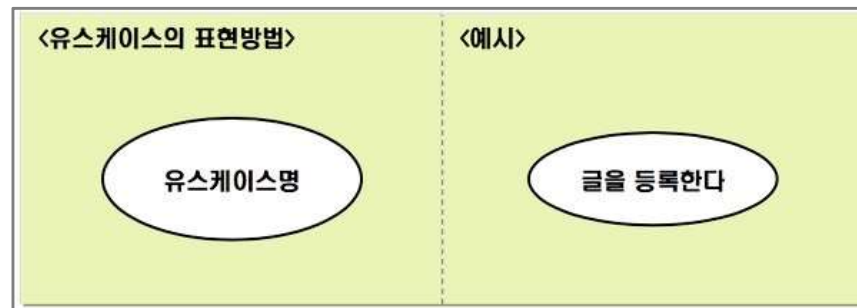
– 시스템에 정보를 제공하는 또 다른 시스템



UML – Usecase Diagram

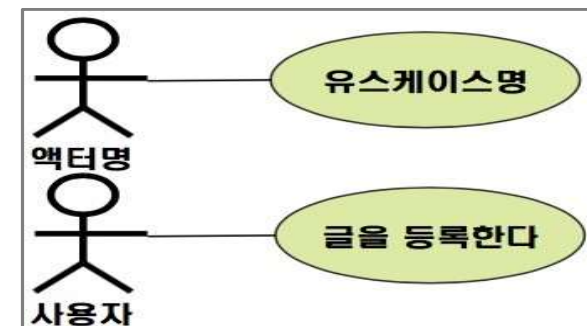
➤ Usecase

- 사용자 입장에서 바라본 시스템 기능
- 시스템 → actor에게 **제공해야 되는 기능**으로 시스템의 요구사항을 나타냄
- 타원으로 표시하고, 안쪽에 유스케이스명을 작성



➤ 관계 (relation)

- actor와 usecase 사이의 의미있는 관계 나타냄
- **연관관계(association)**
 - : actor와 usecase간 **상호작용** 있음 표시 (**실선**)

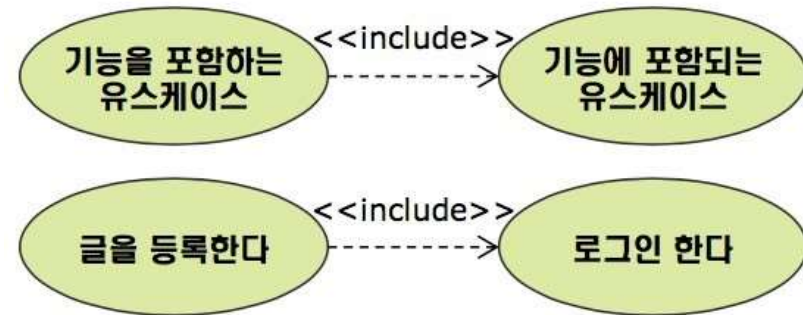


UML – Usecase Diagram

➤ 관계 (relation)

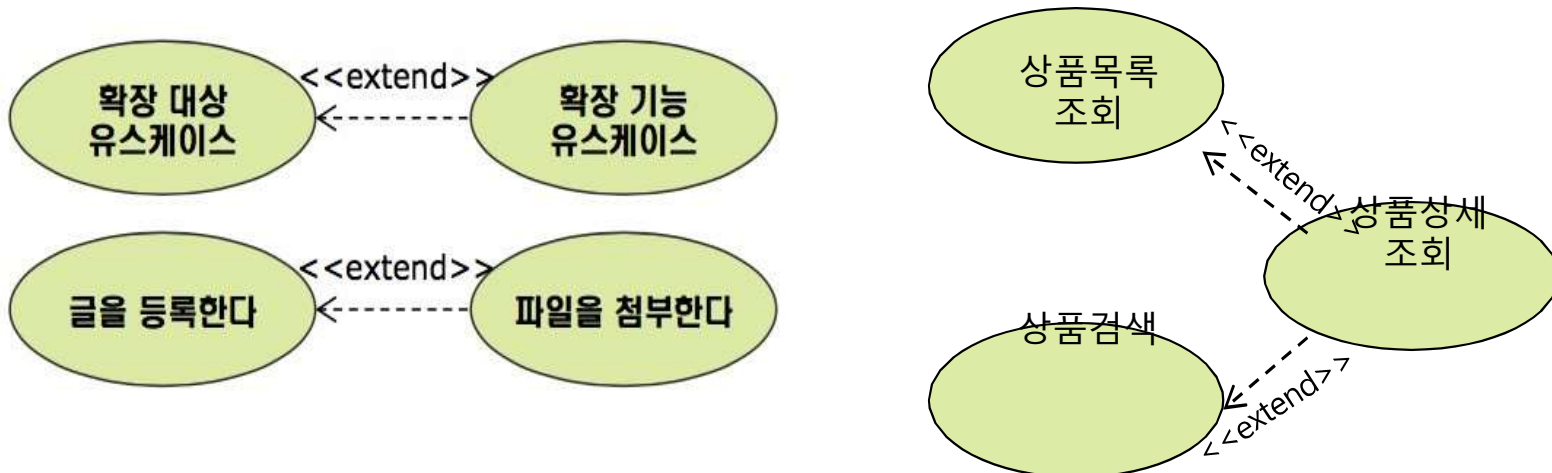
– 포함 관계(Include)

: 하나의 유스케이스가 다른 유스케이스의 실행을 전제로 할 때 형성



– 확장 관계(extend)

: 기존 유스케이스에 부가적으로 추가된 기능을 표현

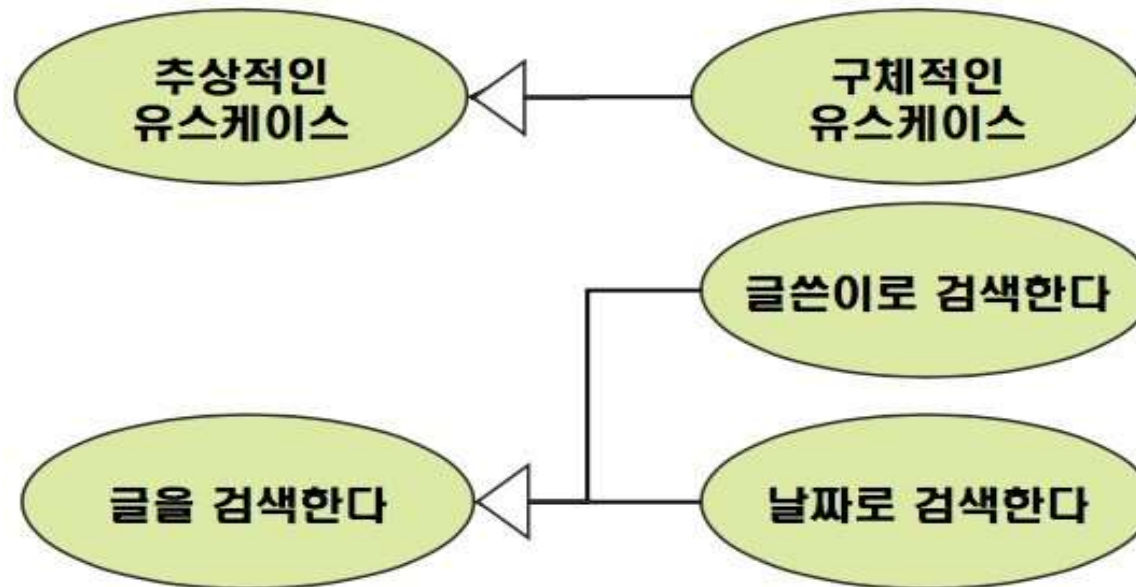


UML - Usecase Diagram

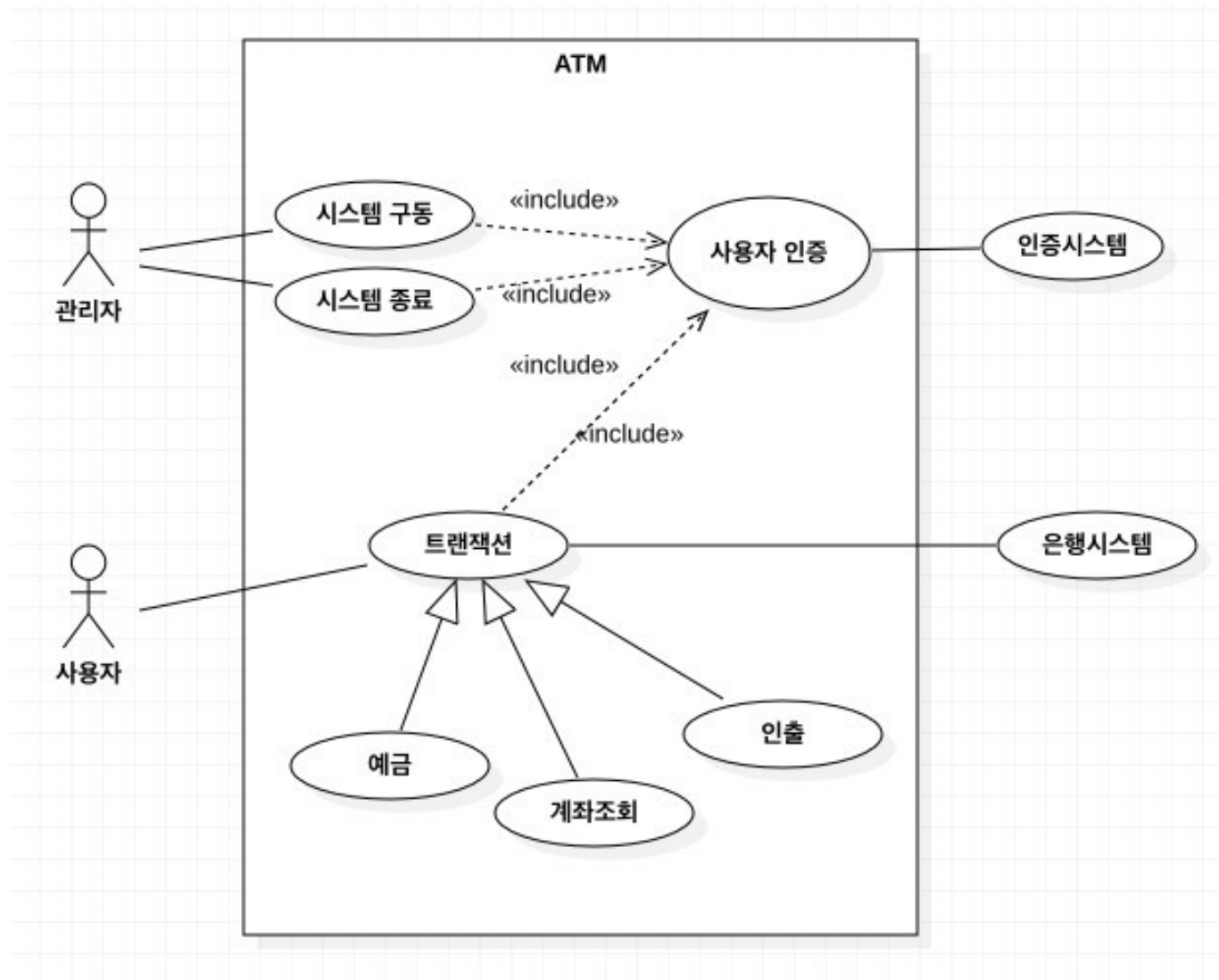
➤ 관계 (relation)

– 일반화 관계 (generalization)

- 개별적인 것에서 공통 특징을 뽑아 이름을 붙인 것



UML - Usecase Diagram



UML - Sequence Diagram

■ 시퀀스 다이어그램 (Sequence Diagram)

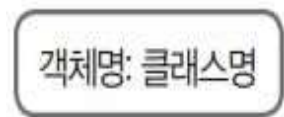
- 특정 행동이 어떠한 순서로 다른 객체와 상호 작용하는지 표현
 - 메시지 순서에 초점을 맞춰 나타낸 것
 - 어떤 작업이 객체 간에 발생하는지 시간 순서에 따라 쉽게 파악 가능
-
- 현재 존재하는 시스템이 어떠한 시나리오로 움직이고 있는지 표현
 - API 등 유즈케이스를 디테일하게 알 수 있음
 - 메서드 콜, DB 조회, 타 시스템의 API 호출등 로직을 모델링 가능
 - 시나리오 파악하기 좋음

UML - Sequence Diagram

■ 구성 요소

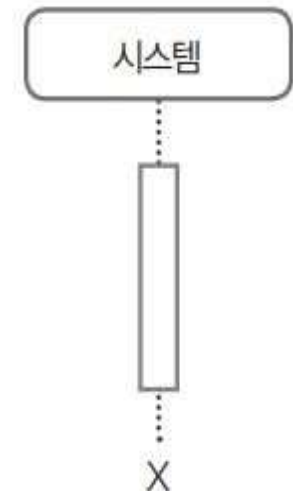
● 객체

- 객체는 메시지를 보내고 받는 주체
- '객체명:클래스명'으로 나타냄



● 객체 생명선

- 객체의 생존 기간을 나타내며 X 표시는 객체가 소멸하는 시점
- 위에서 아래로 내려가는 점선으로 나타냄
- 생명선을 따라 나타나는 직사각형은 활성 구간
: 객체의 메서드가 실행되고 있음을 나타냄
- 구간의 길이는 메서드의 실행 시간을 나타냄
- 활성 구간은 반드시 존재하는 것은 아니며 생략 가능




UML - Sequence Diagram


- 메시지

- 객체와 객체의 상호작용을 표현하는 것으로 화살표를 이용
- 화살표 위에는 수신 객체의 함수명을 명기

- 동기 메시지

- 송신 객체가 수신 객체에 서비스를 요청(메서드 호출)하면 메서드 실행이 완료될 때까지 기다림
- 실선과 속이 채워진 삼각형으로 나타냄 

- 비동기 메시지

- 송신 객체가 수신 객체에 서비스를 요청(메서드 호출)할 때 메서드의 실행과 상관없이 다음 작업을 수행
- 수신 객체로부터의 반환도 기다리지 않는다.
- 일반 화살표 모양으로 나타냄 

UML - Sequence Diagram

- 재귀 메시지

- 수신 객체가 자신의 메서드를 호출할 때 사용



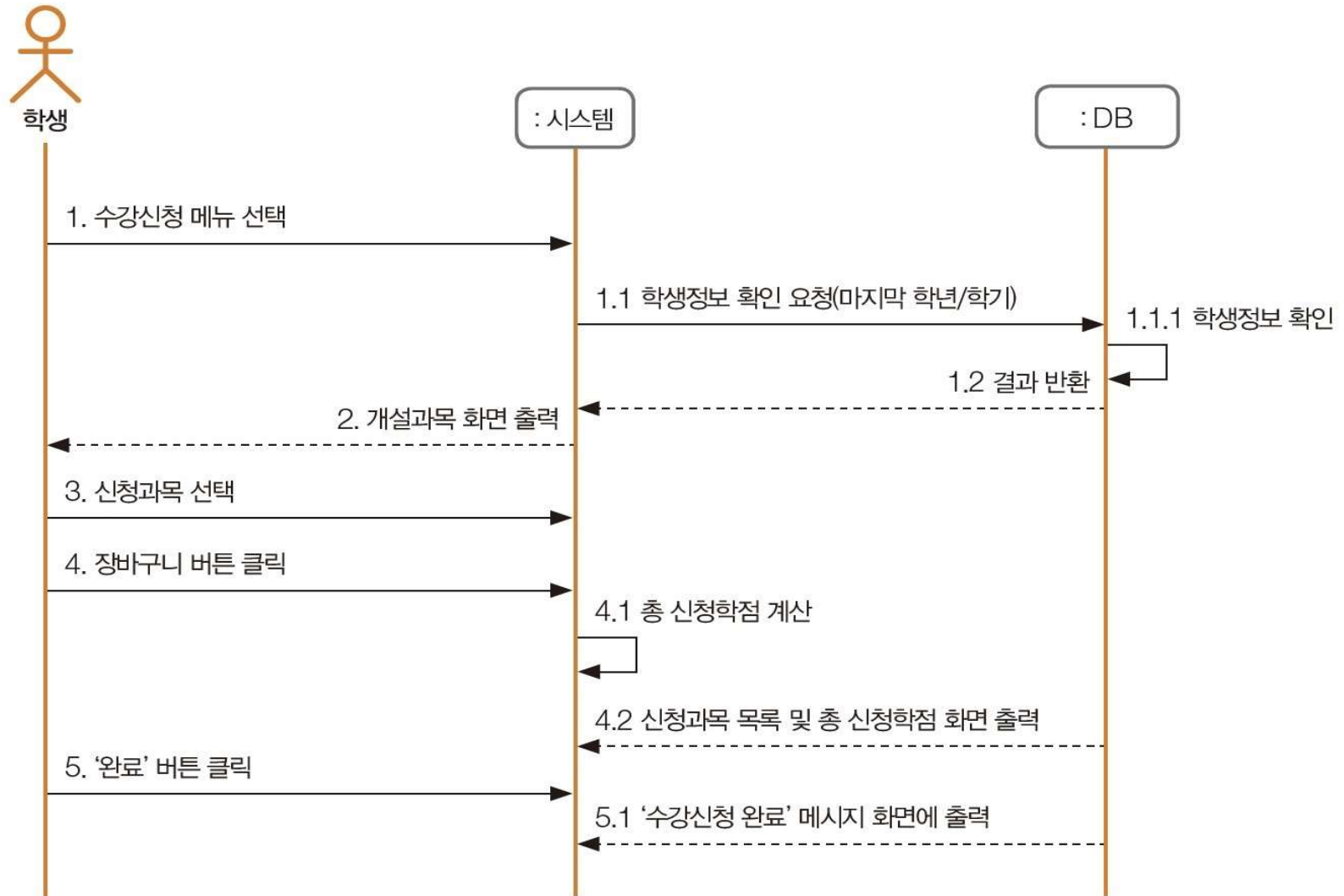
- 답신 메시지

- 호출 메서드의 결과를 반환할 때 사용

- 점선과 속① 채워진 삼각형으로 나타냄



UML - Sequence Diagram



수강 신청 sequence diagram

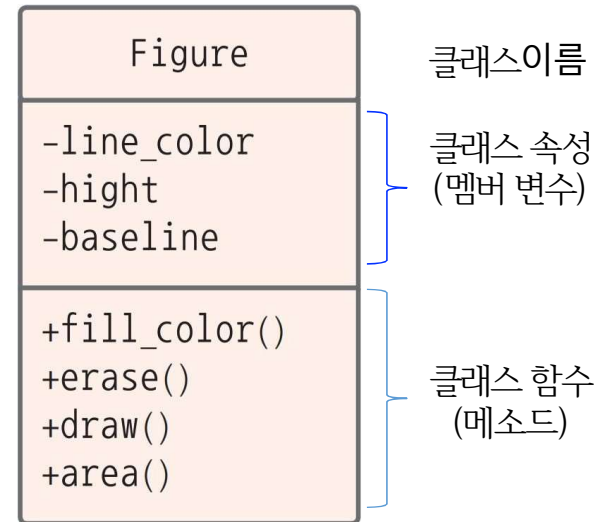
UML - Class Diagram

■ 클래스 다이어그램

- 소프트웨어의 기본 구성 단위인 **시스템에서 사용하는 클래스를 정의**
- **클래스들이 서로 어떻게 연결되어 있고 어떤 역할을 하는지 표현**

■ 클래스

- 데이터(속성)와 메서드를 묶어 놓은 것
- 첫 번째 칸 : 클래스 이름
- 두 번째 칸 : 클래스 속성
- 마지막 칸 : 클래스 제공 기능 메서드



UML - Class Diagram

- 클래스간의 관계

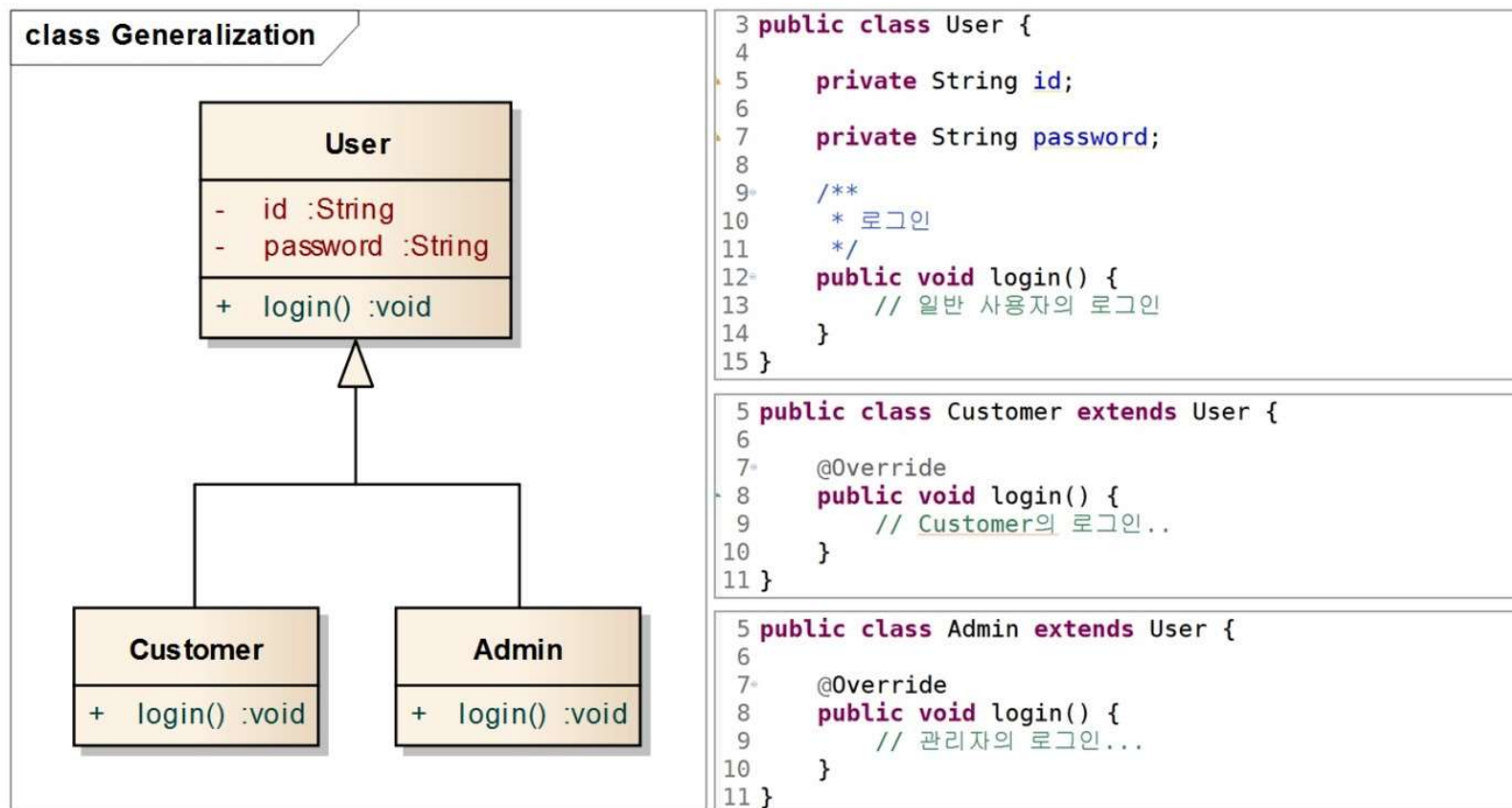
관계	UML 표기
Generalization (일반화)	
Realization (실체화)	
Dependency (의존)	
Association (연관)	
Directed Association (직접연관)	
Aggregation (집합, 집합연관)	
	
Composition (합성, 복합연관)	
	

<https://www.nextree.co.kr/p6753/>

UML - Class Diagram

■ 일반화 (generalization)

- 부모 클래스와 자식 클래스의 **상속 관계**
- **부모 클래스의 내용을 자식 클래스에서 구체화**

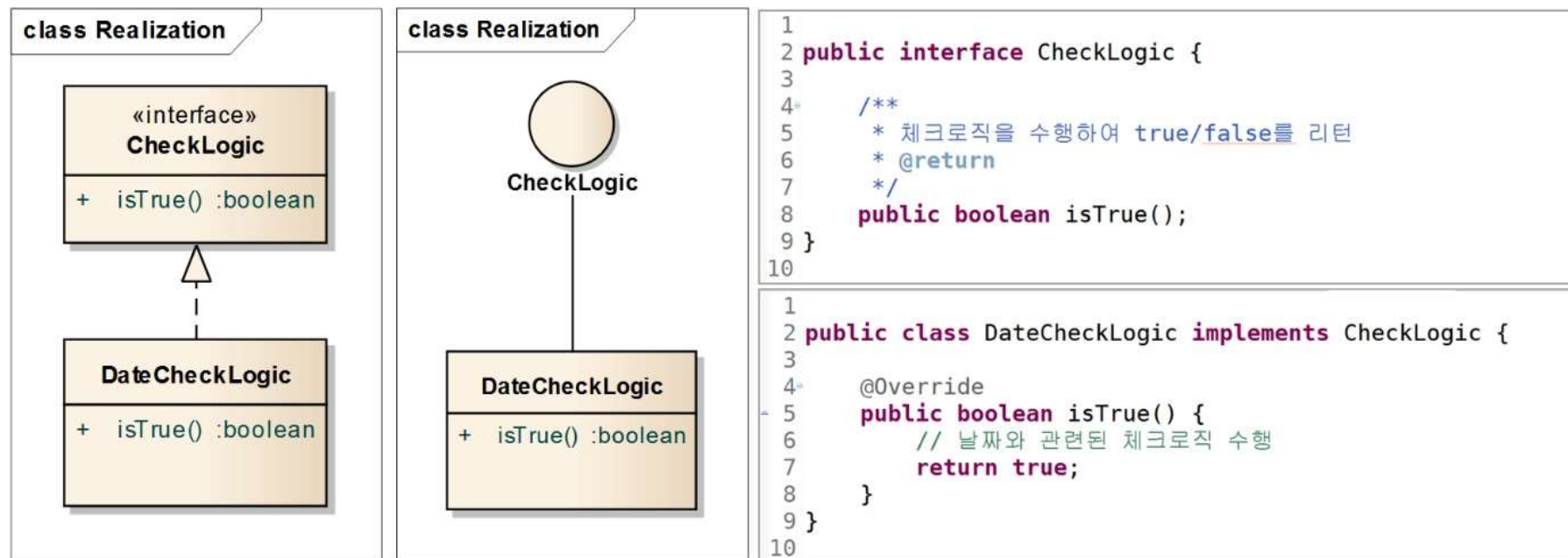


– Customer클래스와 Admin클래스는 User 클래스(부모 클래스)를 상속

UML - Class Diagram

■ 실체화 (Realization)

- 인터페이스(interface)의 메소드를 오버라이딩하여 실제 기능 구현
- 빈 화살표와 점선으로 표현



– DateCheckLogic 클래스에서 인터페이스 CheckLogic의 메소드의 기능을 구현

UML - Class Diagram

■ 의존 (Dependency)

- 어떤 클래스가 다른 클래스를 참조
- 클래스 다이어그램에서 가장 많이 사용되는 관계

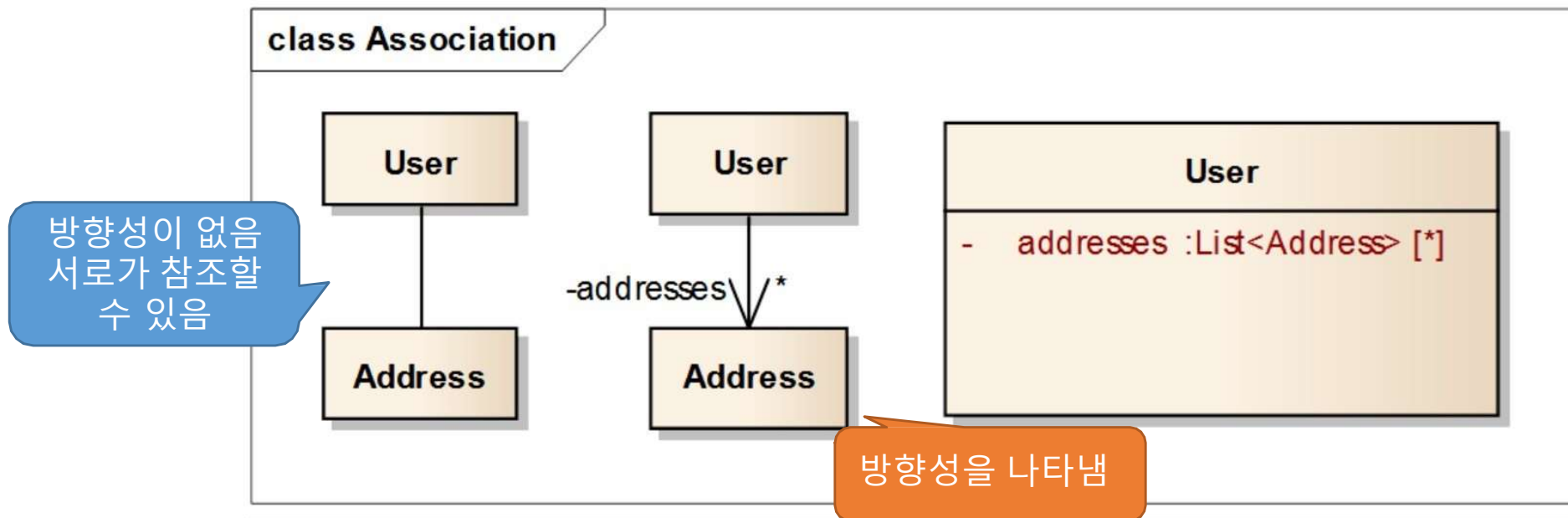


– User 클래스 내부에서 Schedule 클래스의 객체를 참조

UML - Class Diagram

■ 연관 (Association)

- 하나의 클래스에서 다른 객체의 참조를 가짐
 - User 클래스의 멤버 변수로 Address 객체를 참조함
 - *: 인스턴스 개수 범위 (0..1)

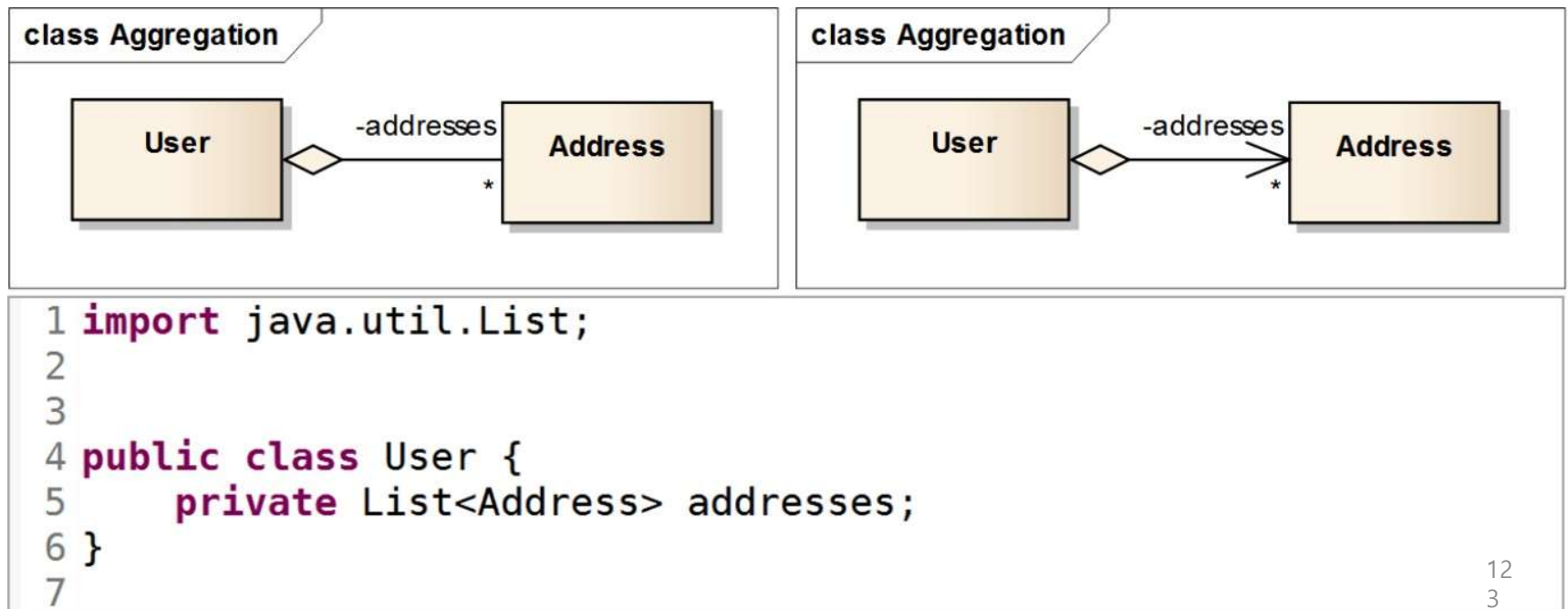


```
1 import java.util.List;
2
3
4 public class User {
5     private List<Address> addresses;
6 }
7
```

UML - Class Diagram

■ 집합 (Aggregation)

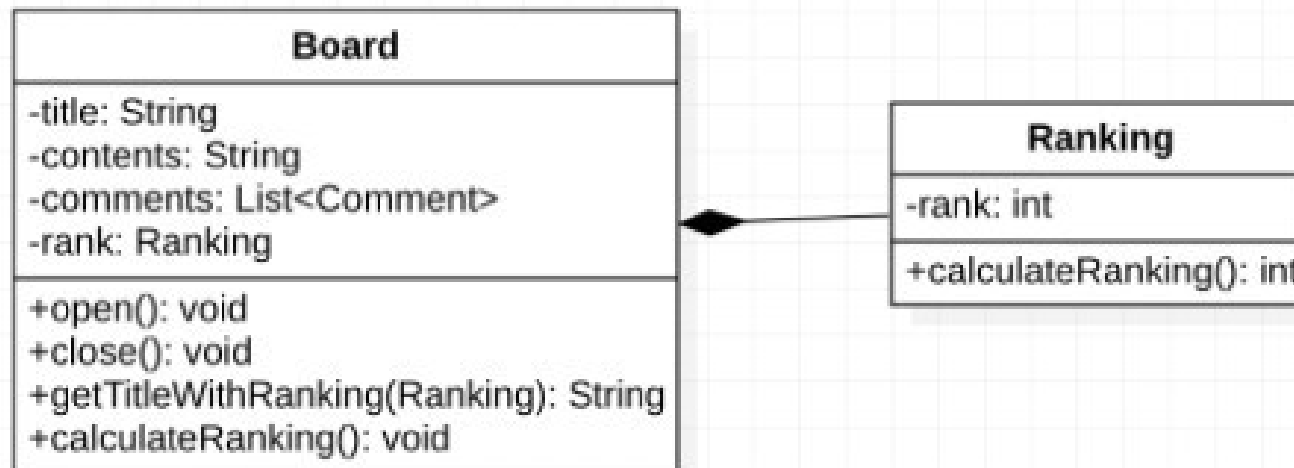
- whole(전체)와 part(부분)의 관계를 나타냄: Association과 차이점이 모호
- Association의 집합 관계를 나타냄 (Collection, Array를 이용하는 관계)
- part가 whole에 대해 독립적 (whole이part를 빌려씀)
- 실선으로 연결, whole쪽에 비어 있는 다이아몬드를 배치
 - 화살표 사용은 옵션



UML - Class Diagram

■ 합성 (Composition)

- Aggregation과 비슷하게 전체와 부분의 집합 관계를 나타냄
- Aggregation 보다 더 강한 집합을 의미
 - part가 whole에 종속적 (whole이 part를 소유)
- 다이어몬드 내부가 채워져 있음



- Ranking 클래스는 Board 클래스 내부에서만 사용
- Board 클래스가 사용되지 않으면, Ranking 클래스도 참조가 불가능한 경우