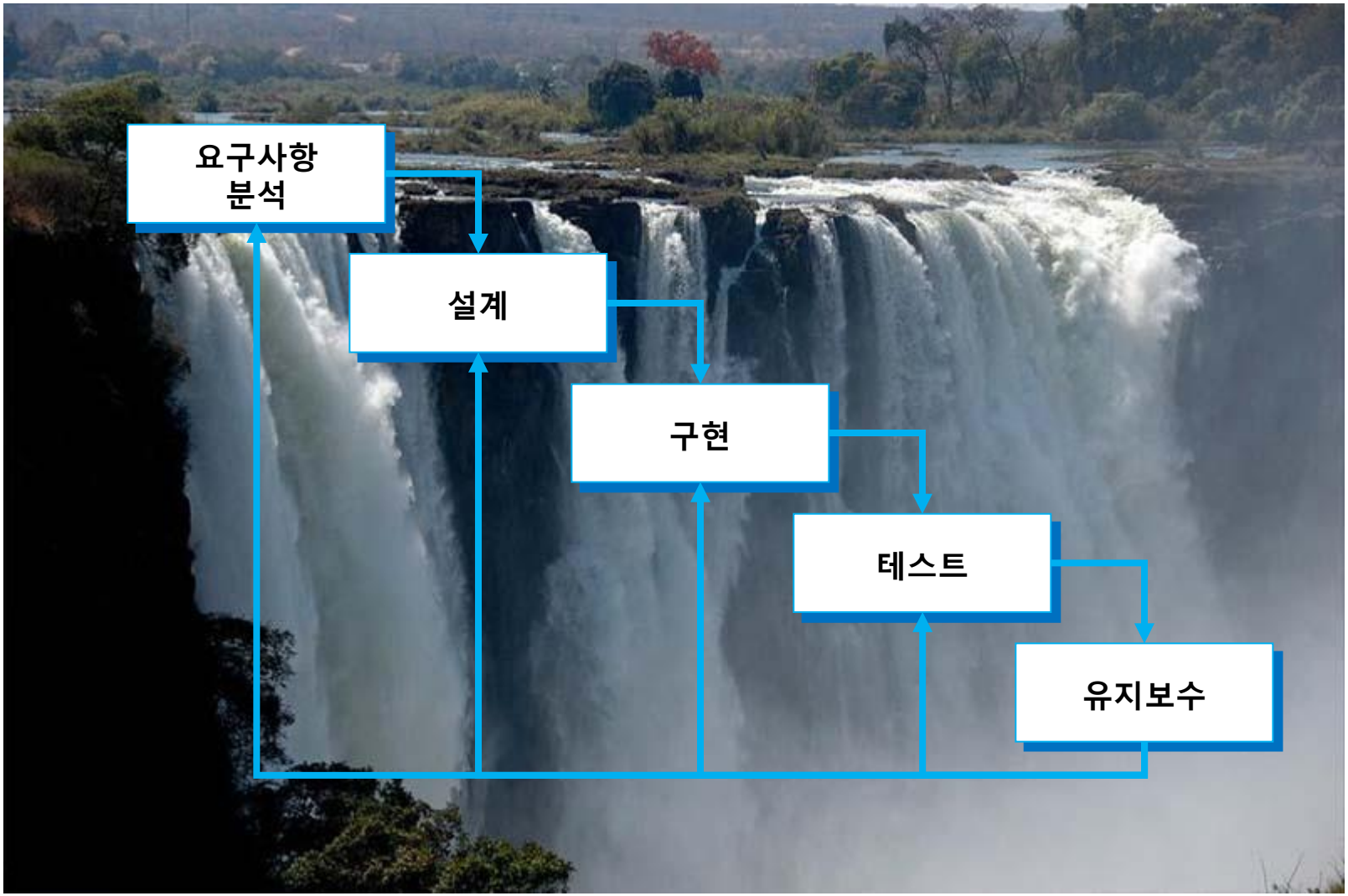
2. 소프트웨어 프로세스

학습목표

- Waterfall 모델
- Evolutionary 모델
- Spiral 모델
- Unified Process
- Agile 프로세스와 XP
- 스크럼

폭포수 모델 (Waterfall Model)



폭포수 모델의 단계

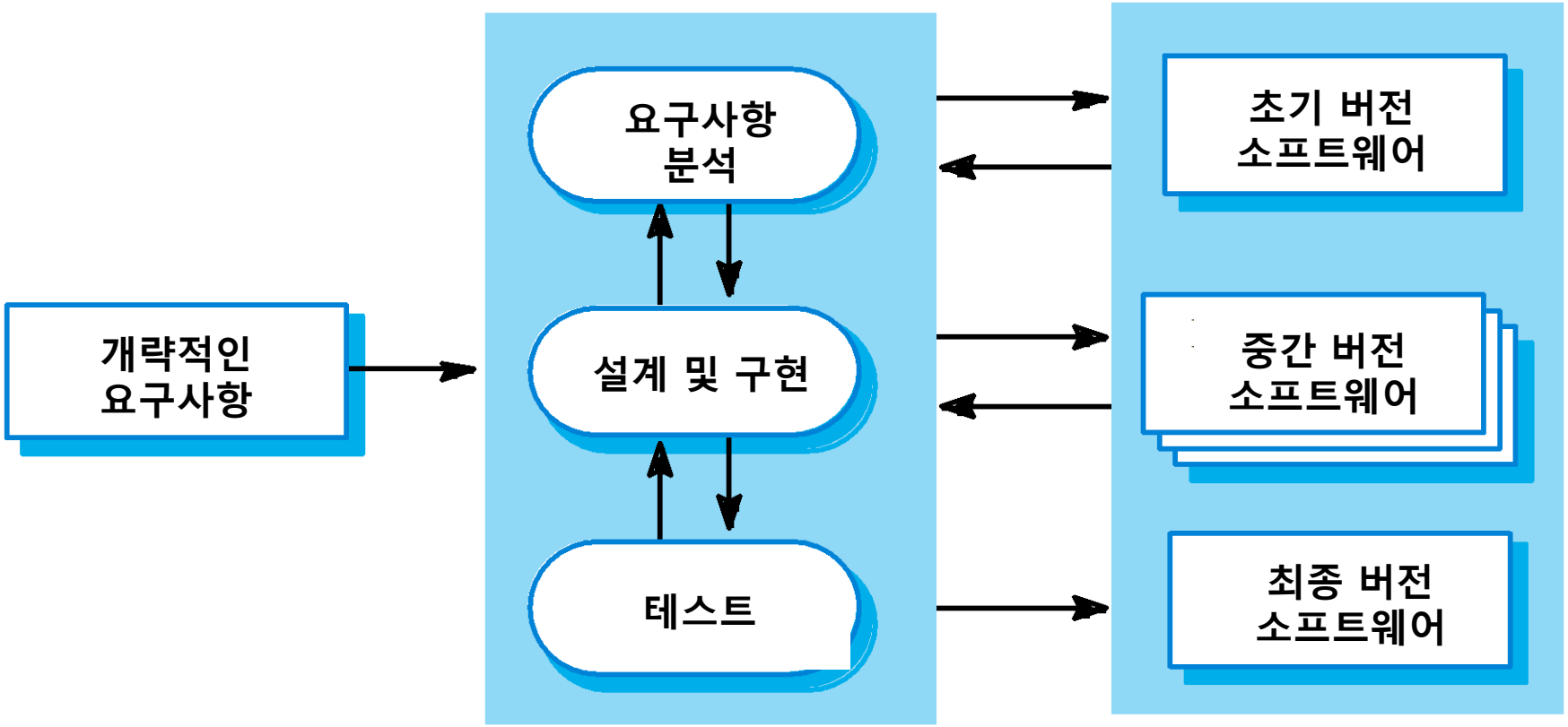
- 단계별 활동
  - 요구사항 분석
    - 시스템의 서비스, 제약 조건, 목표가 소프트웨어 시스템 사용자와의 인터뷰나 사용자 업무의 관찰 등을 통해 정의
    - 소프트웨어 엔지니어 또는 분석가: 고객의 요구사항을 기능, 성능, 인터페이스 등으로 파악하고 문서화
    - 산출물: 요구사항 명세서(Requirement Specification)
  - 설계
    - 한 개 이상의 실행 가능한 프로그램으로 변환할 수 있는 기본적인 소프트웨어 시스템 구성 요소를 식별한 후 이들 간의 관계와 각 기능을 표현
    - 산출물: 설계 명세서
  - 구현
    - 소프트웨어 설계를 프로그램들의 집합 혹은 프로그램의 기본 단위로 변환
    - 산출물: 소스 코드 및 프로그램
  - 테스트
    - 각 프로그램 단위 혹은 프로그램들이 통합되어 소프트웨어 요구사항을 만족하는 완전한 시스템인지 확인
    - 테스트 계획을 세운 후 수행, 문서화
  - 유지보수
    - 전체 생명주기 동안 가장 긴 시간 동안 계속되는 단계
    - 오류를 수정하고 소프트웨어 시스템이 개선되고 새로운 요구사항이 발견됨에 따라 시스템의 서비스를 향상시키기 위한 수정 및 개선 작업을 포함
    - 수정 유지보수, 적응 유지보수, 완전 유지보수 등이 있음

폭포수 모델의 장단점

▌폭포수 모델의 장단점

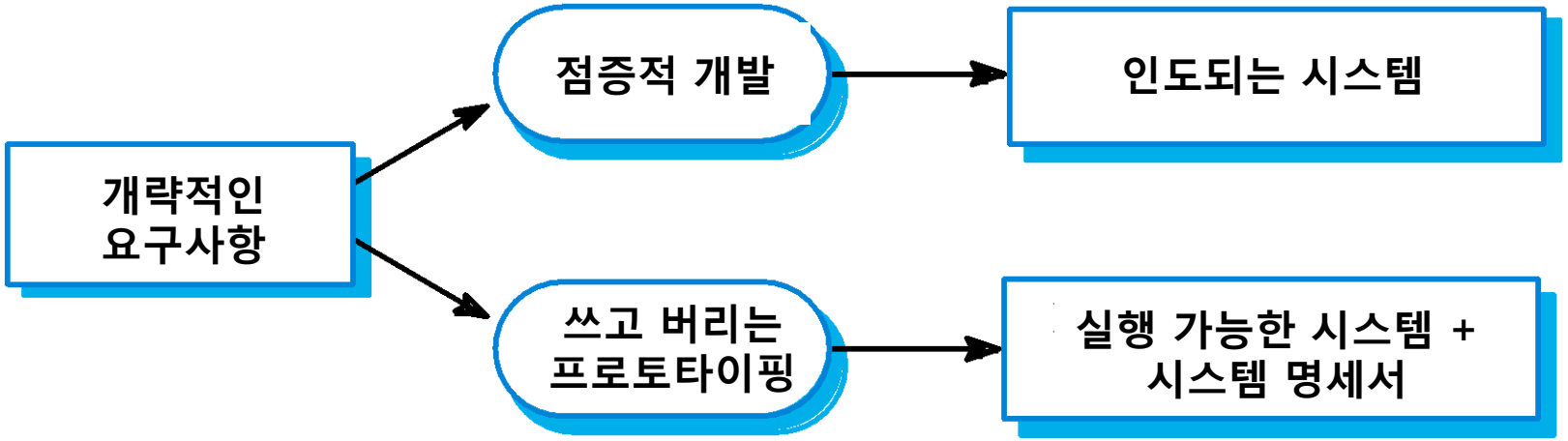
장점	단점
• 가장 오래되고 폭넓게 사용되어 사례가 풍부함 • 순차적인 프로세스이므로 전체 개발 과정이 이해하기 용이함 • 진행 과정이 개별적으로 세분화되고, 순차적이므로 관리하기 용이함 • 기술적 위험이 적고 적용 사례에 따른 경험이 많아 비용, 일정 예측이 용이함 • 산출물에 대한 관리와 적용이 용이함	• 초기에 정확한 요구사항을 유도하고 확정 짓기가 어려움 • 후반부에 개발이 구체화되기 때문에 초기에 중요한 문제점을 발견하기 어려움 • 사용자 피드백에 의한 반복 단계가 불가능 • 초기에 사용자의 의견을 반영하기 어려움 • 초기 단계 강조 시에 구현과 테스트가 지연될 수 있음

진화형 모델



진화형 모델의 두가지 유형

사용자가 참여하여 자신들의 요구사항을 찾아내면서
최종 시스템을 만들어 가는 것



사용자의 요구사항을 잘 이해하고 시스템에
적절한 요구사항을 도출하기 위한 것

진화형 모델의 장단점

❖ 장점
- 사용자의 정확한 요구사항을 반영하는 시스템을 만드는데 있어서 폭포수 모델보다 더 효과적
- 명세서가 점진적으로 개선될 수 있음
- 사용자가 시스템을 직접 확인하므로 자기의 문제를 잘 이해할 수 있게 되고, 소프트웨어 시스템에 반영될 수 있음

❖ 단점
- 프로세스가 보이지 않음
- 시스템의 구조가 훼손

❖ 권장사항
- 대규모 시스템의 경우 폭포수 모델과 진화형 모델을 혼합하여 사용

나선형 모델(Spiral Model) (1/4)

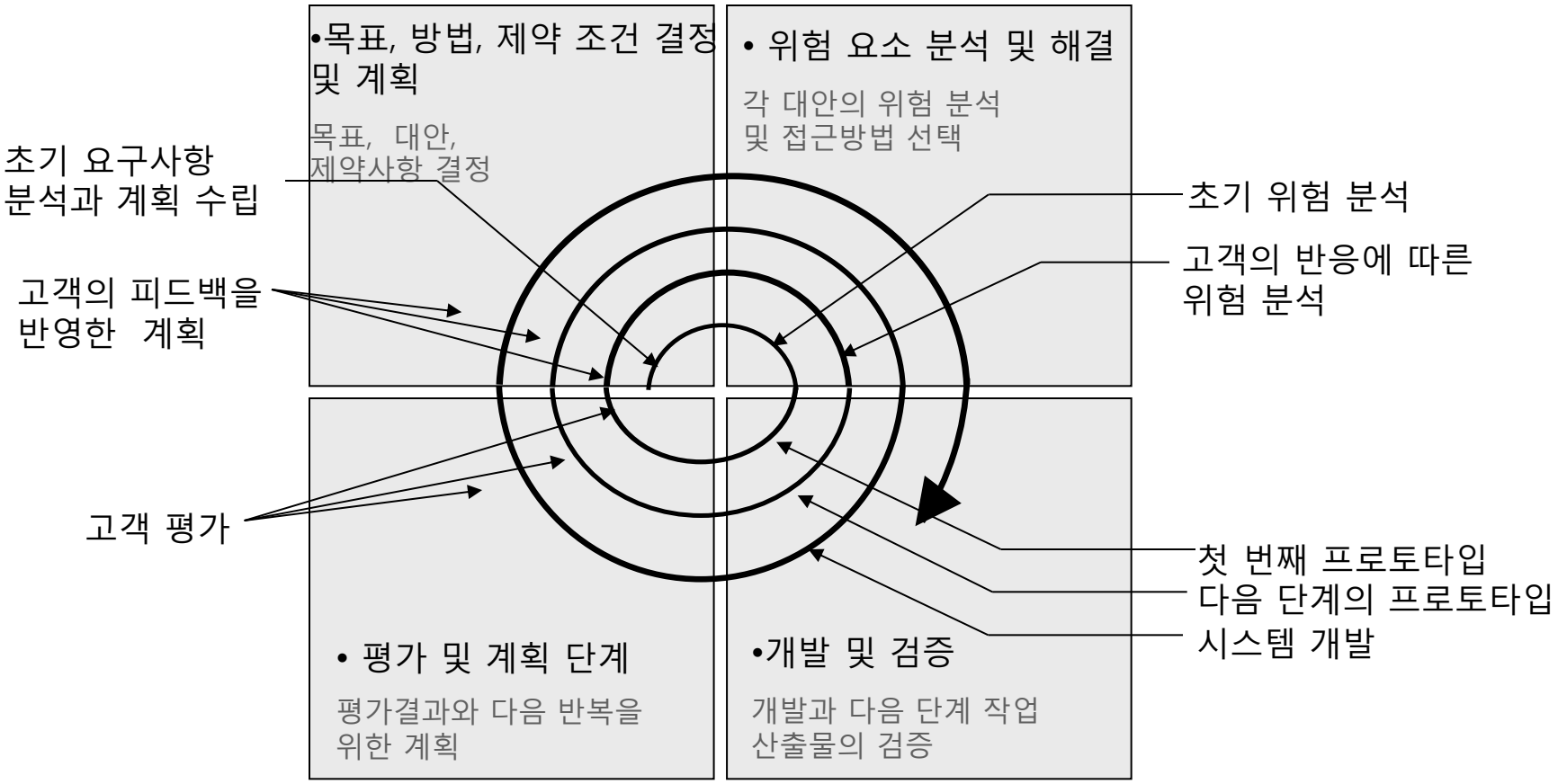
❖ 개요

- 폭포수 모델과 프로토타이핑 모델의 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 타원부터 개념적 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) (3/4)

1. 목표, 방법, 제약 조건 결정 및 계획: 프로젝트의 단계에 대한 목표가 설정된다. 프로세스와 소프트웨어 제품에 대한 제약 조건이 식별되고, 상세한 관리 계획이 수립된다. 프로젝트 위험이 식별되고, 위험의 종류에 따라 대안 전략이 수립된다.

2. 위험 요소 분석 및 해결: 식별된 각 위험의 종류에 따라, 상세한 분석이 수행된다. 위험을 줄이기 위한 단계가 취해진다. 예를 들어 만약 요구사항 분석이 부적절하다는 위험 요소가 있으면, 프로토타입 시스템을 개발한다.

3. 개발 및 검증: 위험 평가 후에 그 시스템에 대한 개발 모델이 선택된다. 예를 들어 만약 사용자 인터페이스 위험이 크면, 적절한 개발 모델은 진화형 프로토타이핑일 것이다. 만약 안전이 중요한 위험 요소라면, 정형 기법에 기반한 개발이 가장 적절할 것이다. 주요 위험 요소가 서브시스템 통합이면 폭포수 모델이 적절할 것이다.

4. 평가 및 계획 단계: 프로젝트가 검토되고, 다음 나선에 대한 추가 반복을 수행할 지에 대한 결정이 이루어져야 한다. 만약 계속하기로 하였다면, 프로젝트에 대한 다음 단계의 계획이 수립되어야 한다.

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 유용

❖ 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
- 폭포수 모델, 프로토타이핑 모델에 비해 상대적으로 복잡하여 프로젝트 관리 자체가 어려울 수 있음

UP (Unified Process) (1/9)

❖ 개요

- Jacobson, Booch, Rumbaugh에 의해 개발된 객체지향 소프트웨어 개발 방법론
- 소프트웨어 개발 단계를 시간의 순서에 따라 네 개의 단계(Inception, Elaboration, Construction, Transition)로 나누고, 각 단계에는 요구사항 도출부터 평가까지 개발 생명주기가 포함되어 있음

❖ 기본적인 원칙

- 반복적(Iterative)이고, 점증적(Incremental)으로 개발
  - 요구사항 분석, 설계, 구현 그리고 평가의 한 사이클이 여러 번 반복되어 개발
  - 반복되는 과정을 통해서 실행 가능한 Release가 산출되어, 결국 최종 시스템으로 발전
- 유스케이스(Use case)를 기반으로 요구사항 관리
  - 요구사항을 식별하고, 정의하는데 있어서 유스케이스 사용
- 컴포넌트 기반 아키텍처(Architecture) 중심의 개발을 지향
  - 시스템 전체를 표현한 아키텍처는 프로젝트 참여자들에게 최종 산출물의 모습을 인지하게 하고, 구성원들을 공통된 시각을 갖도록 함
- 소프트웨어의 시각적인 모델링
- 소프트웨어의 품질 검증
- 소프트웨어 변경 관리

UP (2/9)

UP (3/9)

RUP는 프로세스를 다음 세 가지 관점으로 표현한다.

① 시간에 따라 변하는 모델을 나타내는 동적인 관점(dynamic perspective)
② 정해진 대로 행동하는 프로세스 활동을 나타내는 정적인 관점(static perspective)
③ 프로세스 동안에 사용되는 실무 관행을 제시하는 실무 관점(practice perspective)

UP (4/9)

❖ 도입(Inception)

<도입 단계의 반복 워크플로우>

- 도입 단계의 목표는 시스템에 대한 비즈니스 사례를 만드는 것이다. 시스템과 반응하는 모든 외부 개체와 그들의 상호작용을 식별한다. 그 다음에 이 정보를 이용하여 시스템이 비즈니스에 대한 역할을 평가한다. 만약 이 역할이 중요한 것이 아니면 해당 프로젝트는 취소한다.

UP (5/9)

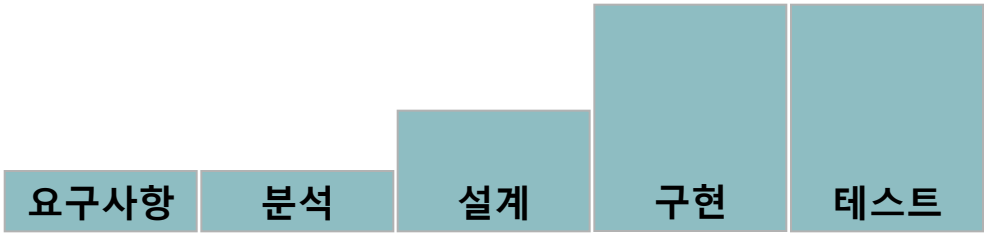
❖ 정련(Elaboration)

<정련 단계의 반복 워크플로우>

- 정련 단계의 목표는 문제 영역을 이해하고 시스템에 대한 아키텍처 프레임워크를 설정하고, 프로젝트를 계획하고 주요 프로젝트 위험을 찾아내는 것이다. 이 단계가 끝나면, 시스템에 대한 요구분석 모델(UML 유스케이스 이용)과 아키텍처 정의, 개발 계획이 산출된다.

UP (6/9)

❖ 구축(Construction)



<구축 단계의 반복 워크플로우>

- 구축 단계는 본격적으로 설계, 구현, 테스트를 하는 단계이다. 동시에 개발된 시스템의 각 부분들은 이 단계에서 통합된다. 이 단계가 끝나면, 실행 가능한 소프트웨어 시스템이 만들어지며 관련 문서가 사용자에게 인도되기 위해 준비된다.

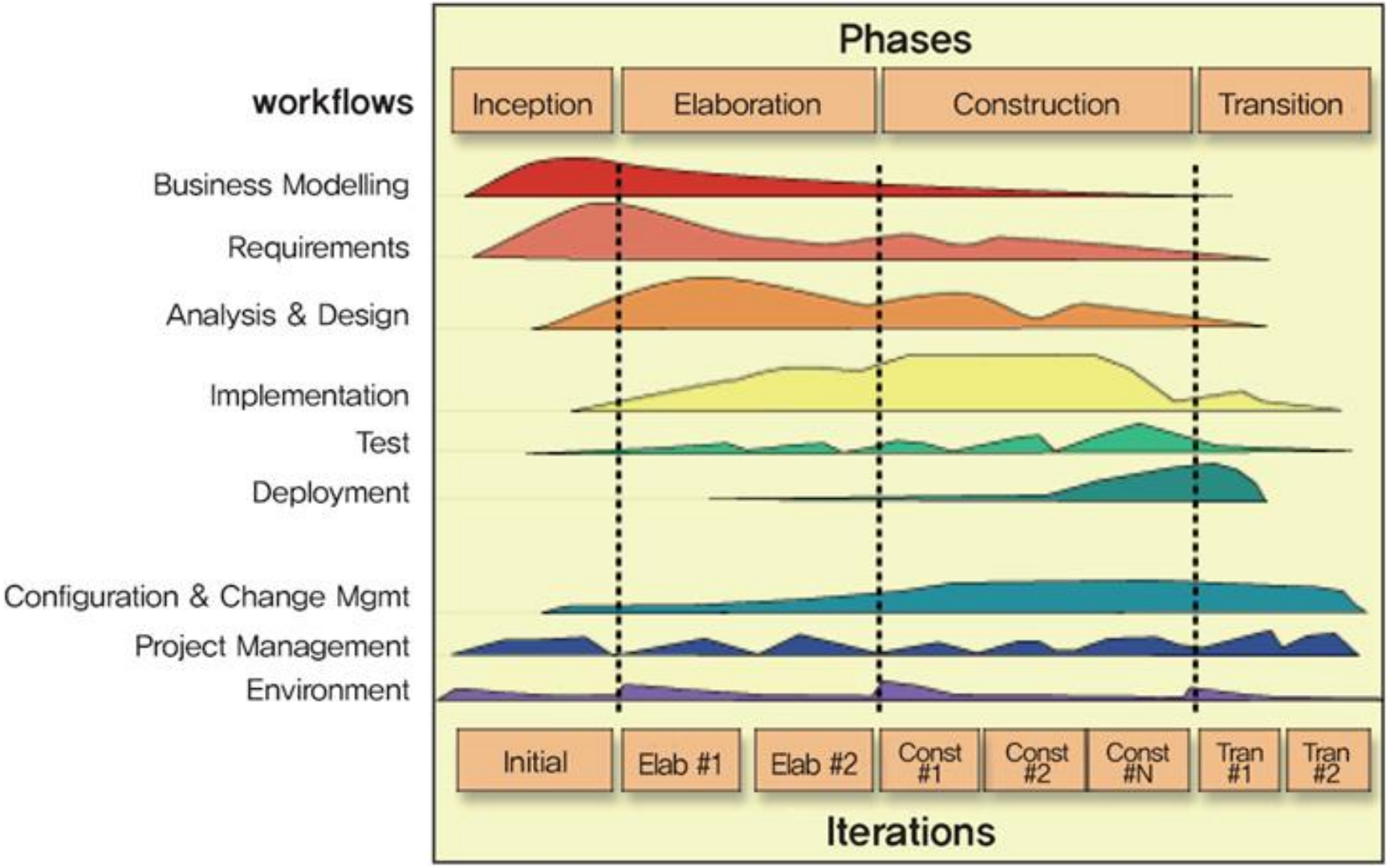
UP (7/9)

❖ 전이(Transition)

<전이 단계의 반복 워크플로우>

- RUP의 마지막 단계는 시스템이 개발 팀으로부터 사용자에게 전달되며 실제 환경에서 동작하는 단계이다. 이 단계는 대부분의 소프트웨어 프로세스에서 중요하게 고려되지 않지만, 실제로는 중요한 단계이다. 이 단계가 끝나면, 실제의 운영 환경에서 올바르게 동작하는 문서화된 소프트웨어 시스템을 가질 수 있다.

UP (8/9)


Phases
workflows
Inception
Elaboration
Construction
Transition
Business Modelling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration & Change Mgmt
Project Management
Environment
Initial
Elab #1
Elab #2
Const #1
Const #2
Const #N
Tran #1
Tran #2
Iterations

UP (9/9)

- 장점
  - 기술적 또는 요구사항 변경 등에 관한 위험요소를 초기에 완화 시킬 수 있음
  - 진척 사항을 가시화할 수 있음
  - 발주자의 실제 요구사항에 근접한 시스템을 만들 수 있음
  - 이전 반복을 통해 얻은 교훈은 다음 반복의 피드백으로 작용하여 반복이 거듭될 수록 개선된 소프트웨어 개발이 가능

UP의 워크플로우

[표 2-1] UP의 워크플로우

워크플로우	설명
비즈니스 모델링	비즈니스 유스케이스를 이용하여 비즈니스 프로세스를 모델링한다.
요구분석	시스템과 반응하는 액터가 식별되며 유스케이스가 시스템의 기능을 모델링하기 위해 개발된다.
설계	설계 모델, 아키텍처 모델, 컴포넌트 모델, 객체 모델, 순차 모델이 생성된다.
구현	시스템 컴포넌트가 구현되고 서브시스템으로 구축된다. 설계 모델로부터 자동적인 코드 생성을 하는 경우 프로세스가 빠르게 진행될 수 있다.
테스트	테스트는 구현과 관련되어 반복되는 프로세스이다. 구현이 끝나면 시스템이 테스트된다.
배치	제품의 릴리스가 생성되고, 사용자에게 전달되며, 업무 공간에 설치된다.
형상관리	시스템에 대한 변경을 관리한다.
프로젝트 관리	시스템 개발을 관리한다.
개발환경	소프트웨어 개발 팀에게 적절한 소프트웨어 도구를 사용할 수 있도록 한다.

애자일 프로세스

[표 2-2] 애자일 기법의 원리

원리	설명
사용자 참여	사용자는 개발 프로세스 전체에 긴밀하게 참여해야 한다. 사용자의 역할은 새로운 시스템 요구사항을 개발하고 우선순위를 결정하고, 시스템의 반복을 평가하는 것이다.
점증적인 인도	소프트웨어는 각 증분에 포함될 요구사항을 검증하기 위해 사용자에게 점증적으로 인도된다.
프로세스가 아닌 사람 중심	개발팀의 기술이 인식되고 활용되어야 한다. 팀 구성원들은 규정된 프로세스 없이 자신들의 작업 방식대로 개발해야 한다.
변경을 수용	시스템 요구사항은 당연히 변경될 것으로 예상하여, 변경되는 요구사항이 쉽게 수용될 수 있도록 시스템을 설계한다.
단순성의 유지	개발 중인 소프트웨어와 개발 프로세스 모두 단순성에 초점을 맞춘다. 가능하면 시스템에서 복잡성을 제거하기 위해 적극적으로 작업한다.

애자일 기법의 특징

- 소규모나 중간규모의 비즈니스 시스템, 개인용 데스크 탑 컴퓨터나 모바일 기기에서 주로 사용되는 소프트웨어 제품의 개발에 가장 적합함
- 개발 팀들이 각자 일하는 장소가 서로 원격지로 떨어져 있고 다른 하드웨어와 소프트웨어 시스템과의 복잡한 상호작용이 존재할 수 있는 대규모 시스템 개발에는 잘 들어맞지 않음
- 안전성이나 보안성과의 관계를 이해하기 위해 모든 시스템 요구사항의 상세한 분석이 필요한 중대한 시스템(Critical System) 개발에는 적절하지 않음

XP(eXtreme Programming)

❖ 개요

- 1990년대 초, Kent Beck에 의해 고안된 개발 방법론
- 반복적 점증적인 개발과 같은 실무 관행과 고객의 참여를 '극한' 수준까지 밀고 나감

❖ 특징

- 요구사항이 변경된다는 것을 가정하고, 고객의 피드백을 수용하기 위해 고객과 개발 팀이 함께 상주
- 동료 프로그래머와의 의사소통을 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

XP에서 사용하는 용어

용 어	설 명
스토리(Story)	고객이 직접 작성하는 요구사항
스토리 추정	개발자가 고객이 제시한 스토리가 어느 정도의 난이도 또는 기간인지를 결정하는 것
릴리스(Release)	고객에게 구현된 제품을 배포하는 것
반복(Iteration)	하나의 릴리스 안에 반복되는 작업
드라이버(Driver)	짝 프로그래밍에서 타이핑하면서 코드를 작성하는 사람
파트너(Partner)/ 네비게이터(Navigator)	짝 프로그래밍에서 드라이버를 도와 코드의 구조 및 결함에 대한 조언을 하는 사람

XP의 개발 활동

① 점증적인 개발은 시스템의 작고 빈번한 릴리스를 통해 그리고 프로세스 계획 수립의 기초가 될 수 있는 사용자 스토리나 시나리오에 기반을 둔 요구사항 기술 접근법에 의해 지원된다.

② 사용자의 참여는 사용자가 개발팀에 전임으로 고용되는 것을 의미한다. 사용자 대표는 개발 팀에 참여하고 시스템의 인수 테스트에 책임이 있다.

③ 프로세스가 아닌 사람 중심, 짝 프로그래밍, 시스템 코드의 공동 소유, 그리고 지나치게 긴 작업 시간을 포함하지 않고 계속할 수 있는 개발 프로세스를 통해 지원된다.

④ 변경은 정기적인 시스템 릴리스, 테스트 우선 개발 그리고 계속적인 통합을 통해 지원된다.

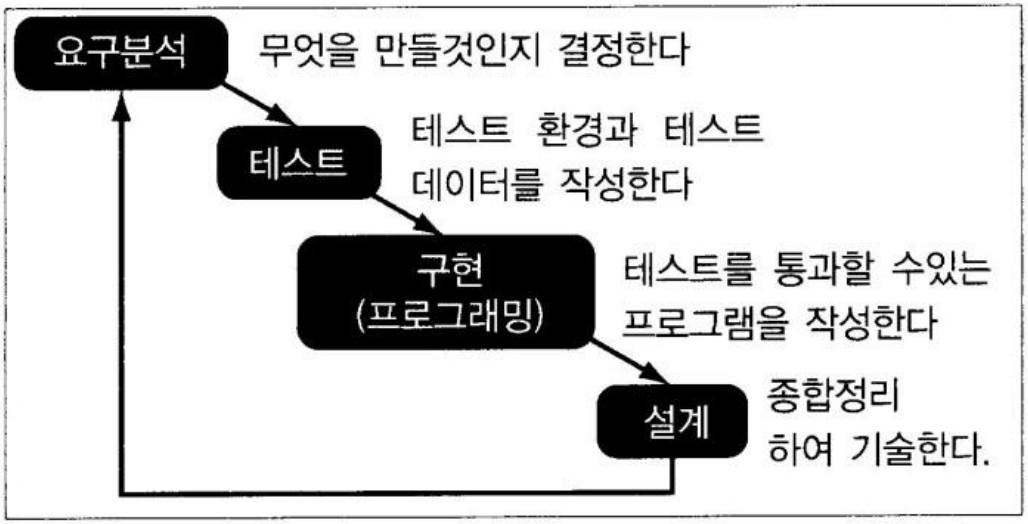
⑤ 단순성을 유지하는 것은 코드 품질을 개선하기 위한 규칙적인 리팩토링을 통해 가능하고, 또한 시스템에 대한 장래의 변경을 쉽게 수용할 수 있도록 단순하게 설계를 하는 것으로 가능하다.

XP의 실무 관행

[표 2-3] 익스트림 프로그램의 실무 관행

실무 관행	설명
점증적인 계획수립	요구사항은 릴리스에 포함될 스토리 카드와 스토리에 기록되고, 가용 시간에 상대적인 우선순위를 결정한다. 개발자들은 이러한 스토리를 개발 '작업'으로 분해한다.
소규모 릴리스	비즈니스 가치를 제공하는 최소한의 유용한 기능이 먼저 개발된다. 시스템의 릴리스들은 빈번하게 최초의 릴리스에 기능을 점증적으로 추가한다.
단순한 설계	현재의 요구사항만을 만족하기 위한 설계가 수행된다.
테스트 우선 개발	기능 그 자체가 구현되기 이전에 기능을 테스트하기 위해 자동화된 단위 테스팅 프레임워크가 이용된다.
리팩토링	모든 개발자들은 코드가 개선될 수 있는 가능성이 발견되자마자 계속적으로 코드를 리팩토링한다.
짝 프로그래밍	개발자들은 짝을 이루어 작업하고, 각자의 작업을 점검하고 항상 좋은 작업을 하도록 지원한다.
공동 소유권	개발자들은 시스템의 모든 영역에서 짝을 이루어 작업하여 단독으로 수행되는 개발 작업이 없도록 하고, 개발자들이 모든 코드를 공동으로 소유하도록 한다.
계속적인 통합	작업이 완료되자마자 전체 시스템으로 통합된다. 이러한 통합 이후에 시스템의 모든 단위 테스트를 통과해야만 한다.
유지 가능한 속도	종종 코드의 품질을 낮추고 중간 수준의 생산성만을 달성하는 효과를 내는 과다한 초과 근무는 고려되지 않는다.
현장의 사용자	시스템의 최종 사용자 대표(고객)는 XP 팀의 이용을 위해 전적으로 시간을 활용할 수 있어야 한다. 익스트림 프로그래밍 프로세스에서 고객은 개발팀의 구성원이 되고 시스템 요구사항을 구현 팀에게 전달하는 책임이 있다.

테스트 우선 개발과 짝 프로그래밍

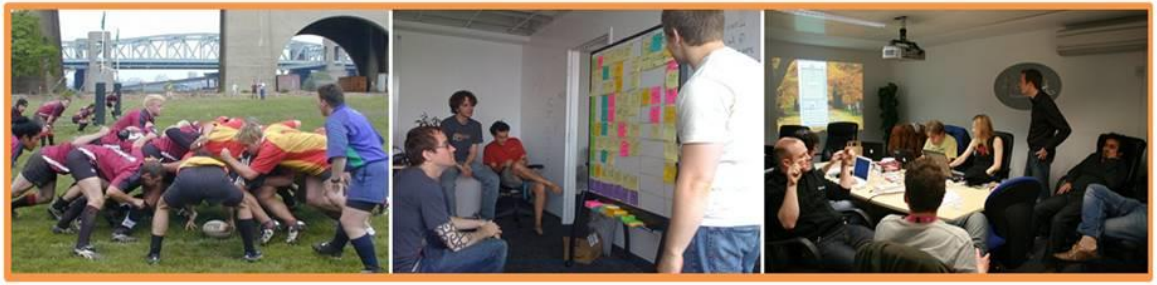


테스트 우선 개발(Test First Development)

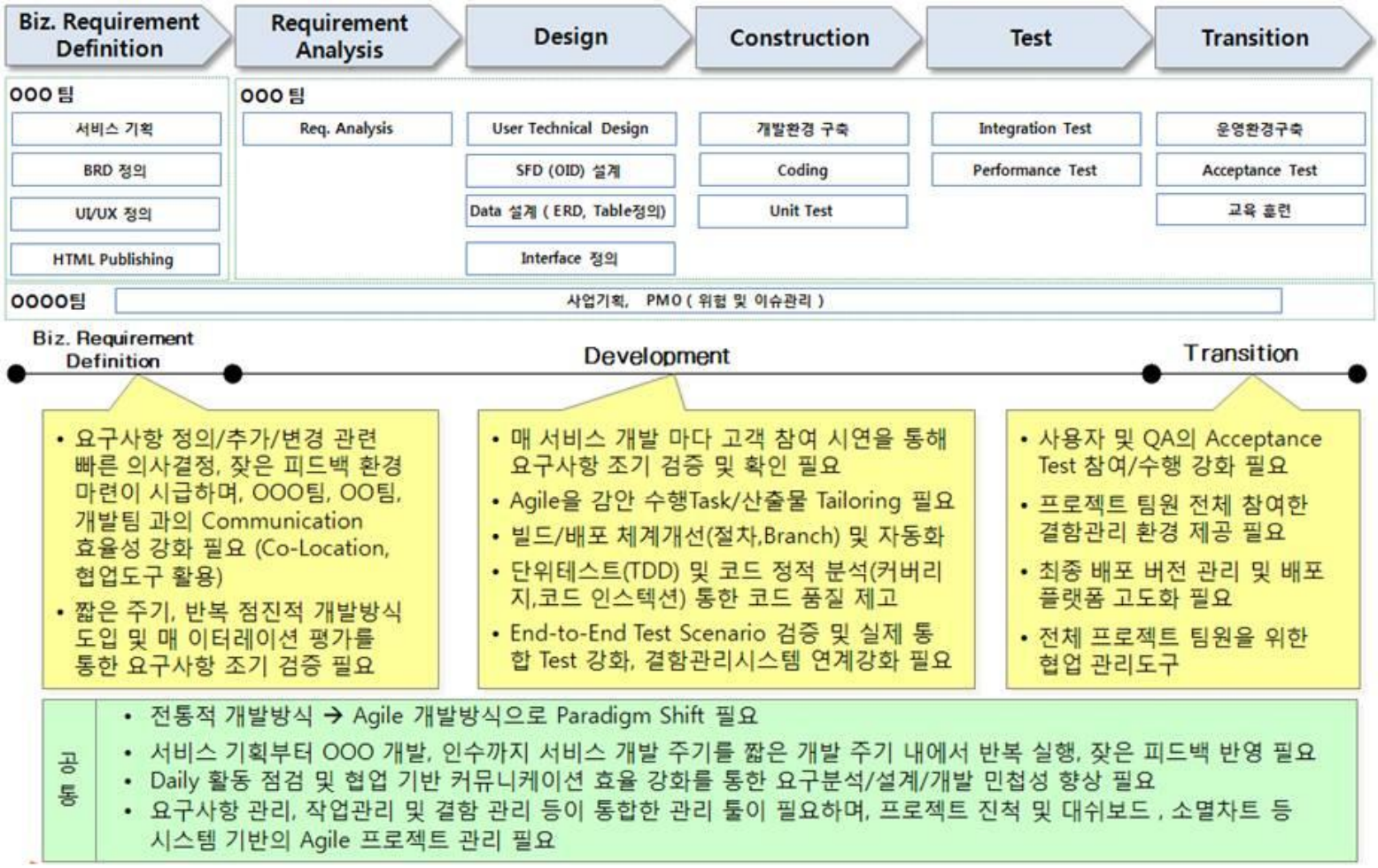
짝 프로그래밍(Pair Programming)

스크럼(Scrum)

- 가장 인기 있고 널리 사용되는 애자일 기법



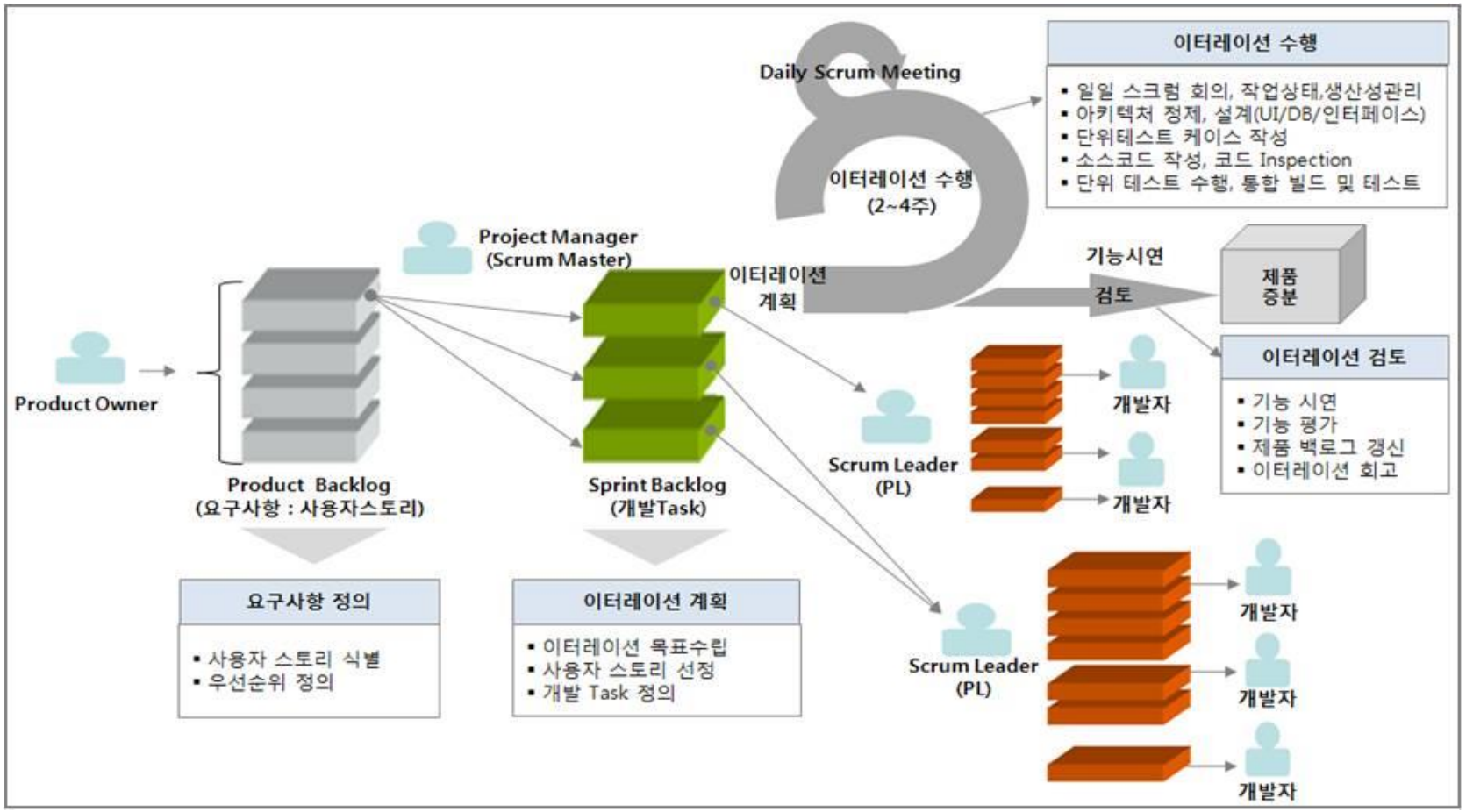
프로젝트 개선점 도출 사례



짧은 개발 주기 반복을 통한 서비스 조기 출시

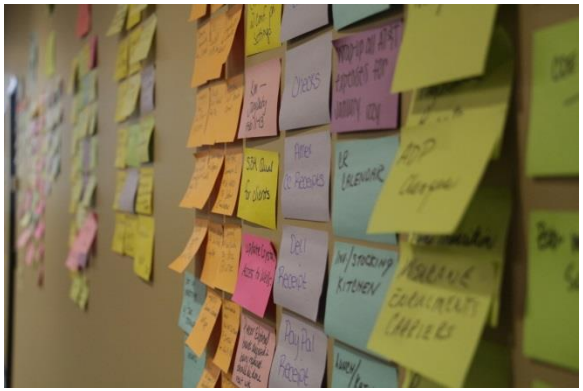
구분	이터레이션 주요 원칙
기본 프레임	이터레이션 계획 -> 수행 -> 검토로 구성
반복주기 및 기간	반복주기는 2회 ~ N회, 주기당 기간은 2~4주이며, 프로젝트 특성에 따라 조정 가능함
Daily Scrum 회의	Scrum 조직별 15분 남짓 진행하며, 절대 30분을 넘기지 않는다. Scrum of Scrums 회의 (주당 1~2회)

이터레이션 기본 프레임 사례



이터레이션

- 이터레이션 계획
  - P. 35
- 이터레이션 수행
  - 일일 스크럼 회의



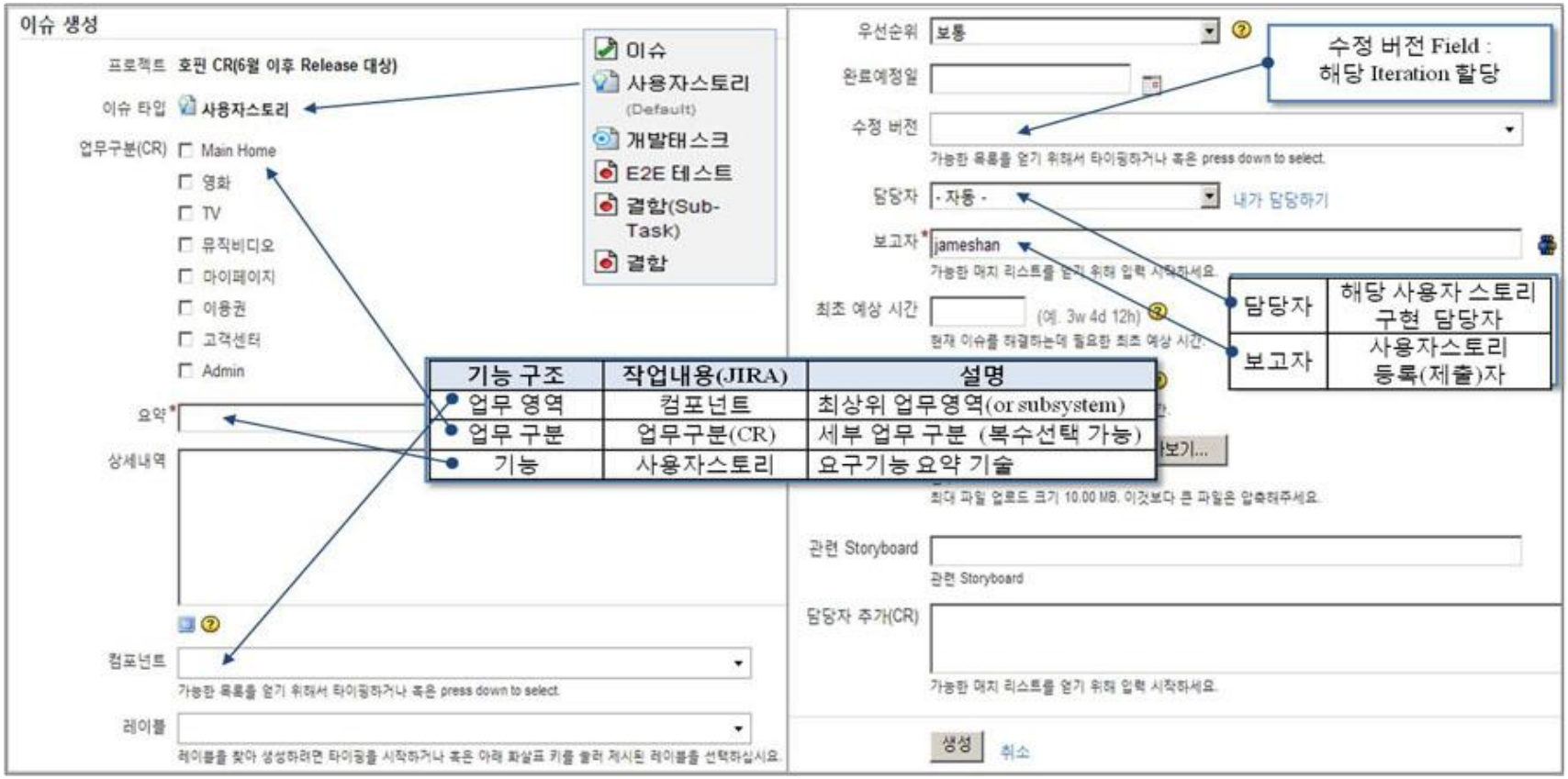
  - P. 35
- 이터레이션 검토
  - P. 36

사용자 스토리

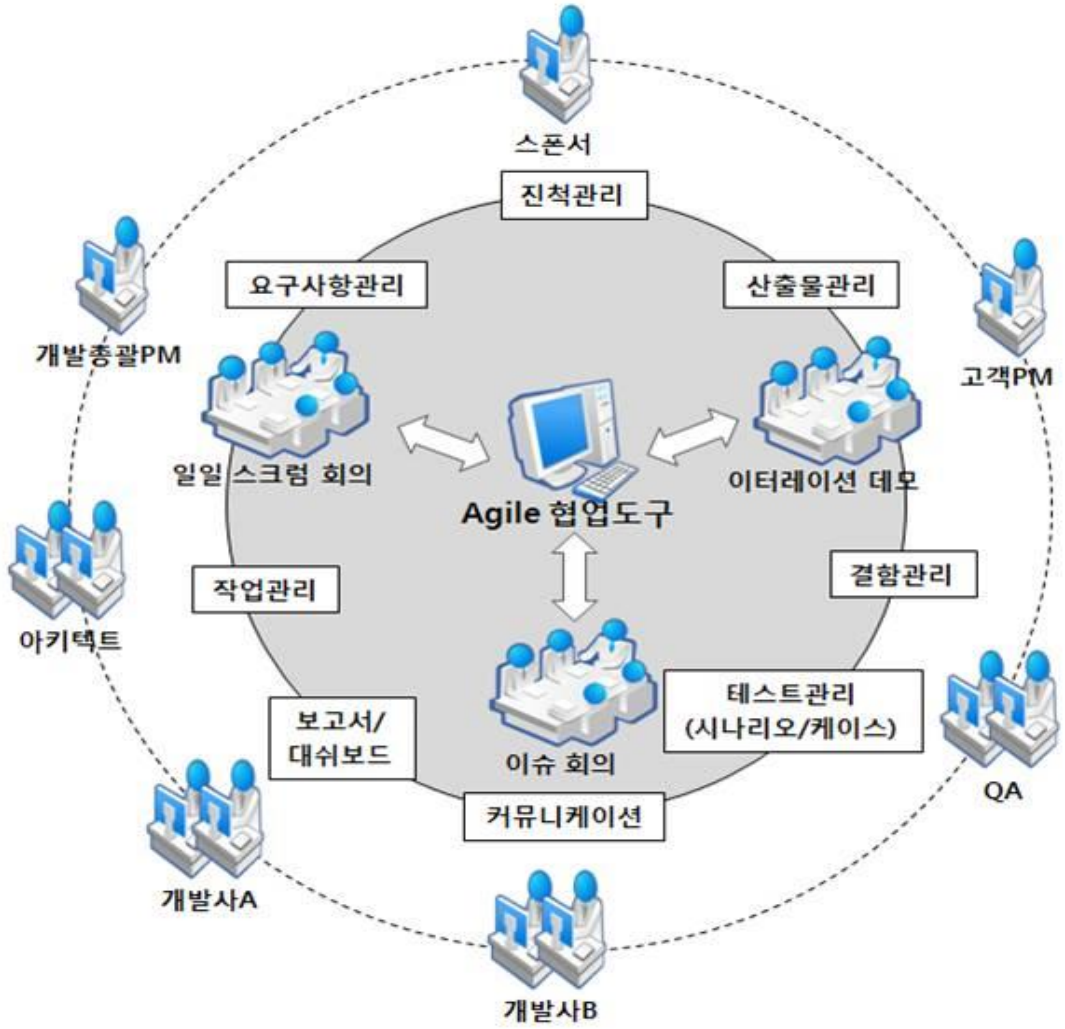
- 사용되는 소프트웨어 기능 혹은 요구사항
  - 일반적으로 한 두 문장으로 짧게 표현
  - 상세한 명세는 아님
  - 한두 명의 개발자가 2주일 안에 구현하고 테스트할 정도의 크기가 적당
  - INVEST 원칙
    - Independent
    - Negotiable
    - Valuable
    - Estimable
    - Small
    - Testable

사용자 스토리 생성 사례 (1/2)

사용자 스토리 생성 사례 (2/2)



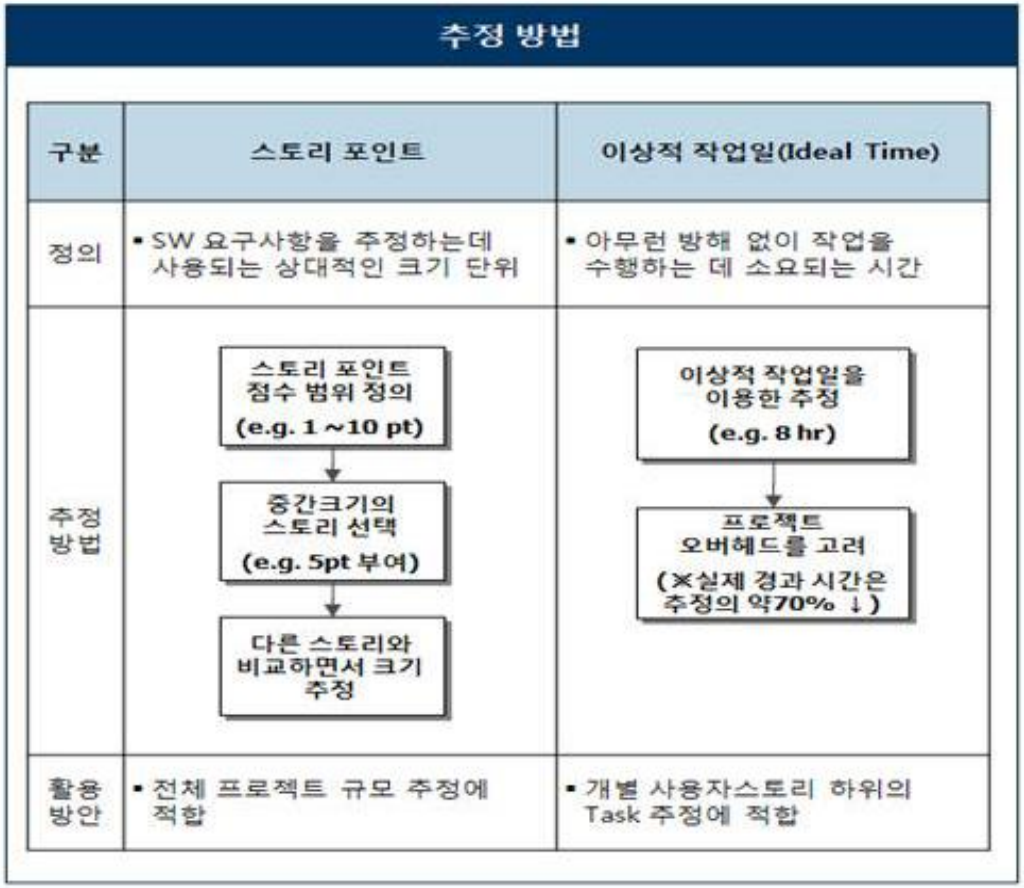
애자일 협업체계 구성


스폰서
진척관리
요구사항관리
산출물관리
개발총괄PM
고객PM
일일 스크럼 회의
이터레이션 데모
Agile 협업도구
작업관리
결함관리
아키텍트
보고서/
대쉬보드
테스트관리
(시나리오/케이스)
이슈 회의
QA
커뮤니케이션
개발사A
개발사B

협업도구의 애자일 태스크보드

사용자 스토리 규모추정

스토리 포인트 추정 방법



스토리 포인트 추정방법은 사용자스토리 중 중간으로 생각되는 스토리 하나를 골라, 스토리에 할당될 점수 범위의 중간쯤에 오는 값을 할당합니다. 예를들어, 1에서 10까지의 값을 갖도록 선정하였다면 중간 사이즈의 사용자 스토리에 5점을 부여하면 됩니다.

Burn down chart (소멸차트) 활용