

Software Engineering

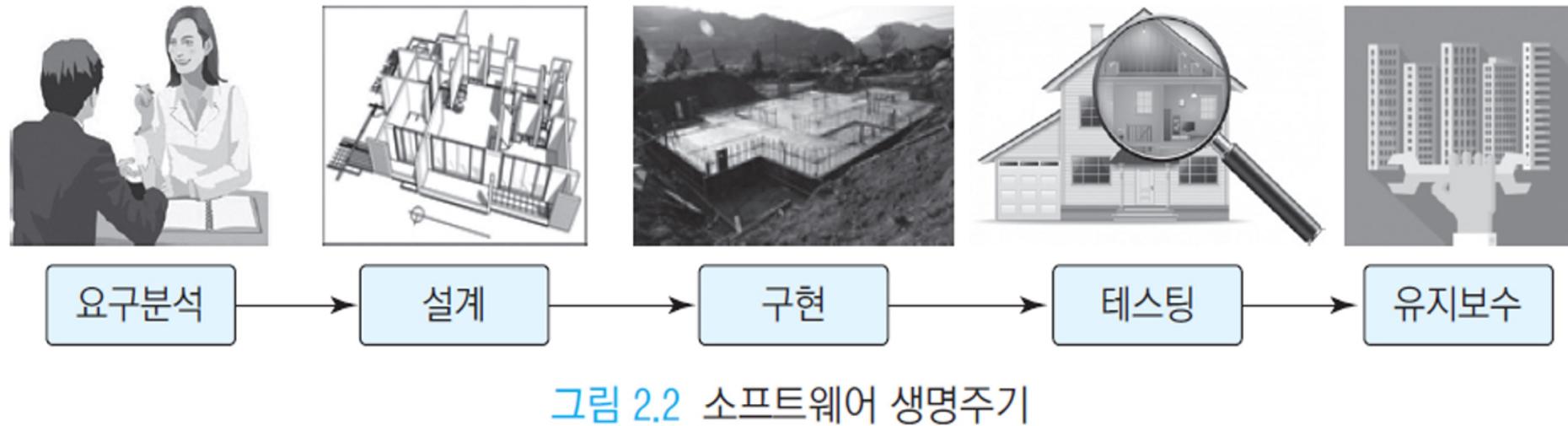
2. process
(Chapter 2)

Ji Eun Kim
Fall 2024

Topics

- Software Development Life-cycle
- Software Development Processes
 - Linear Sequential Process Models
 - Evolutionary Process Models
 - United Process Models
 - Agile Process Models

Software Development Life-cycle (생명주기)



Process

- 프로세스: 일을 처리하는 과정 또는 순서
- 주어진 일을 해결하기 위한 목적으로 그 순서가 정해져 수행되는 일련의 절차
- 어떤 일을 해결하고자 할 때는 해당 프로세스만 잘 따르면 목적을 달성할 수 있음
 - 예) 요리, 빨래, 소프트웨어 설치 등...

Software Development Process

즉흥적 소프트웨어 개발 (Code-and-fix)

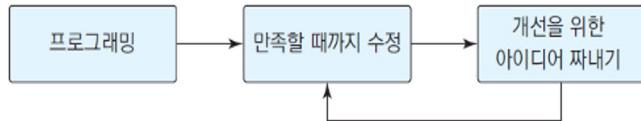
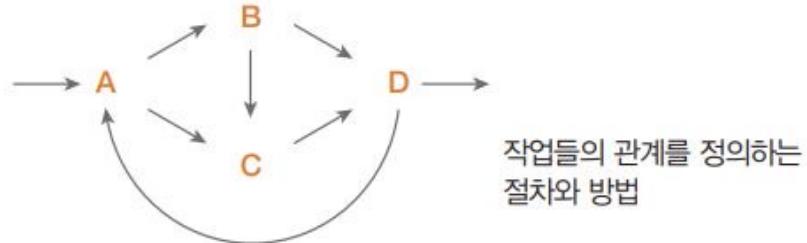


그림 1.6 즉흥적인 소프트웨어 개발

- 공식적인 가이드라인이나 프로세스가 없는 개발 방식
- 코드를 작성해 제품을 만든 후에 요구분석, 설계, 유지보수에 대해 생각
- 첫 번째 버전의 코드를 작성해 제품을 완성한 뒤 작성된 코드에 문제가 있으면 수정해서 해결하고 문제가 없으면 사용
- 단점
 - 정해진 개발 순서나 각 단계별로 문서화된 산출물이 없어 관리 및 유지보수가 매우 어려움
 - 프로젝트 전체 범위를 알 수 없고 좋은 아키텍처를 만들 수도 없음
 - 개발자가 일을 효과적으로 나눠 할 수도 없음
 - 프로젝트 진척 상황을 거의 파악할 수 없음
 - 코딩을 먼저 하므로 계속 수정할 가능성이 높은데, 여러 번 수정하다 보면 가독성이 높았던 프로그램의 구조가 나빠져 수정이 매우 어려워짐

Software Development Process

- 좁은 의미: 소프트웨어 제품을 개발할 때 필요한 절차나 과정
- 넓은 의미: 작업을 수행하는 데 필요한 방법과 도구를 비롯해 개발과 관련된 절차를 따라 작업을 수행하는 참여자 포함



Software Development Process

- 정의
 - 소프트웨어를 어떻게 개발할 것인가에 대한 전체 흐름을 체계화한 개념
 - 개발 계획 수립부터 최종 폐기까지의 전 과정을 순차적으로 다룸
- 목적
 - 주어진 예산과 자원으로 개발하고 관리하는 방법을 구체적으로 정의
 - 고품질의 소프트웨어 제품을 만드는 것이 목적
- 역할
 - 프로젝트에 대한 전체적인 기본 골격을 세워 일정 계획을 수립할 수 있고, 개발 비용 산정 및 여러 자원을 산정하고 분배할 수 있음
 - 참여자 간에 의사소통의 기준을 정할 수 있고 용어의 표준화를 가능하게 함
 - 개발 진행 상황 파악
 - 각 단계별로 생성되는 문서를 포함한 산출물을 활용해 검토할 수 있게 함

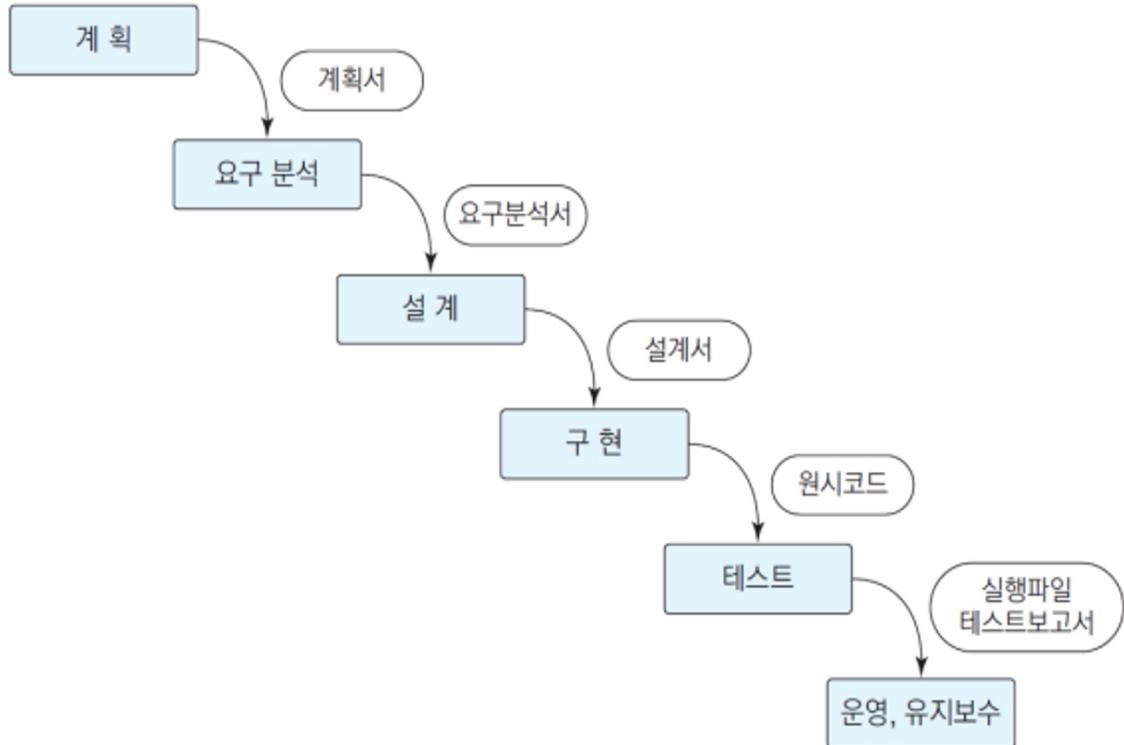
Software Development Process Models

- Waterfall Model (폭포수 모델)
- V Model
- Prototyping Model (프로토타이핑 모델)
- Spirial Model (나선형 모델)
- Phased Development Model (단계적 모델)
- Unified Process Model
- Agile Process Models (XP, Scrum, DevOps)

Software Development Process: Linear Sequential Process Models

Waterfall Model (폭포수 모델)

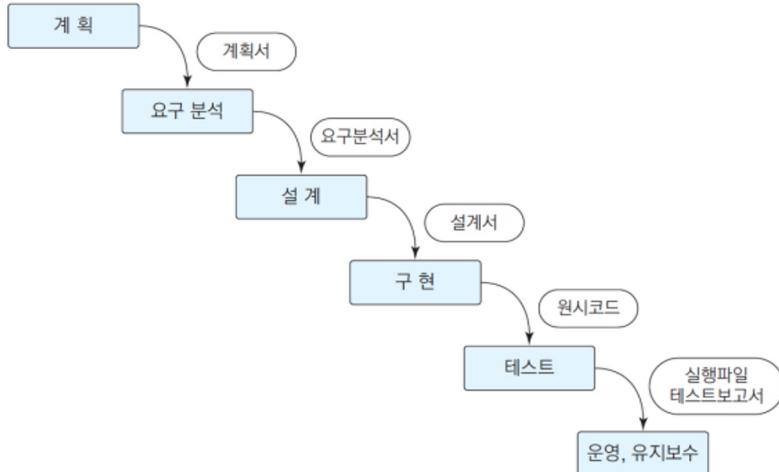
- Established in 1970 by Winston W. Royce
- 선형 순차적 모델
- 폭포에서 물이 떨어지듯이 다음 단계로 넘어가는 모델 (고전적 생명 주기)
- 소프트웨어 공학의 대명사로 여겨질 만큼 초기에 개발된 전통적인 모델
- 표준 프로세스를 정해 소프트웨어를 **순차적**으로 개발



Waterfall Model 장단점

장점

- 프로세스가 단순하여 초보자가 쉽게 적용 가능
- 관리가 용이
- 중간 산출물이 명확하고 체계적인 문서화가 가능
- 코드 생성 전 충분한 연구와 분석 단계



단점

- 각 단계는 앞 단계가 완료되어야 수행할 수 있음
 - 모호한 부분이나 프로세스 진행 과정에 변경될 수 있는데 이를 수용할 수 없다.
- 각 단계마다 작성된 결과물이 완벽한 수준으로 작성되어야 다음 단계에 오류를 넘겨주지 않음
- 테스트 작업이 프로젝트 후반 즉 시스템이 완성된 후에 시작
- 사용자가 중간에 가시적인 결과를 볼 수 없어 답답해할 수 있음

요구사항의 변화가 적은 프로젝트에 적합

V Model

- 검증 (Verification) 을 강화하는 관점에서 Waterfall Model을 확장한 모델
- 폭포수 모델이 산출물 중심이라면 V 모델은 각 개발 단계를 검증하는 데 초점을 두므로 오류를 줄일 수 있음
- **Reliability**가 요구되는 software에 많이 사용됨

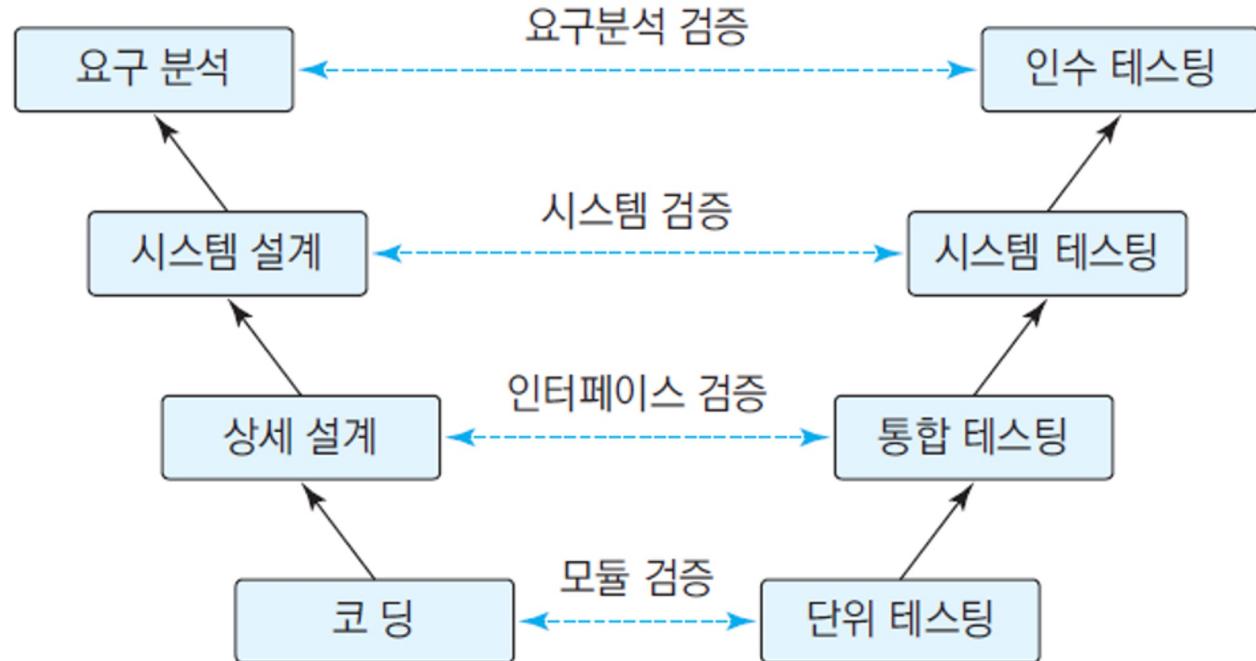


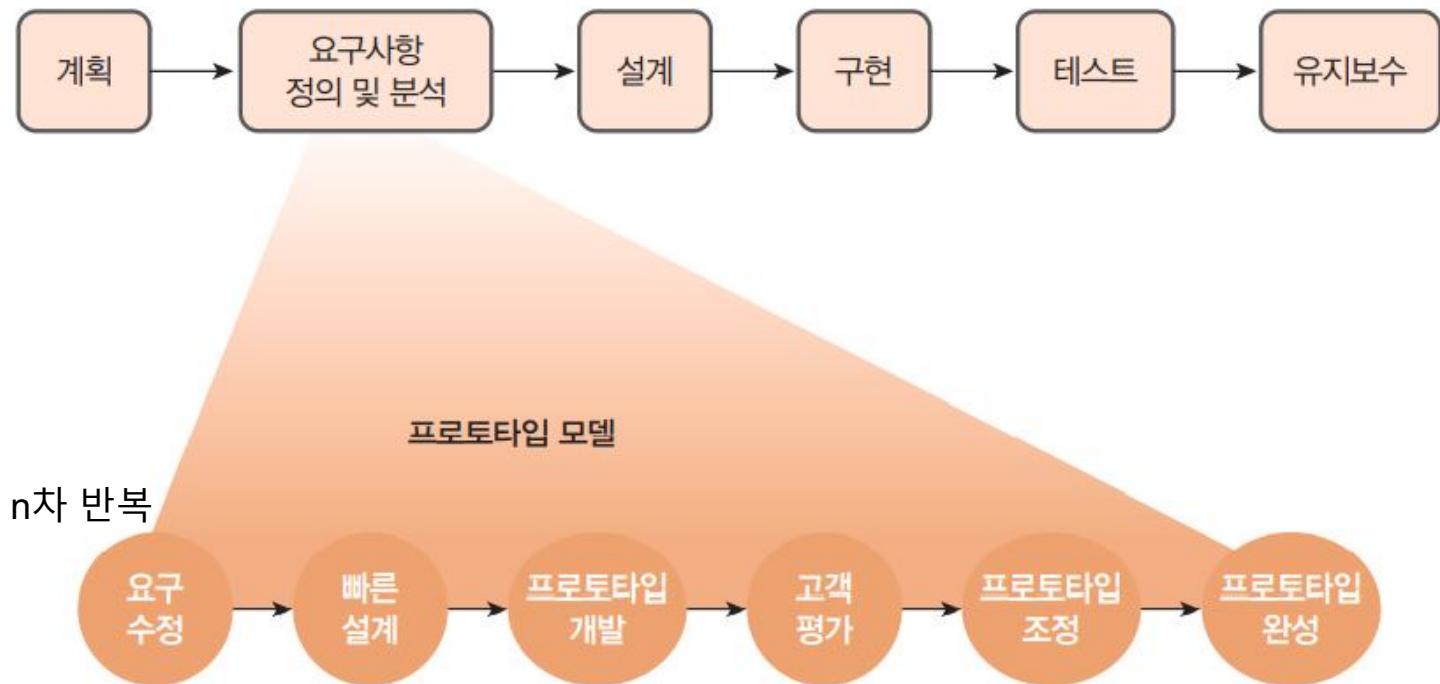
그림 2.11 V 모델

Software Development Process: Evolutionary Process Models

Prototyping Model

- 요구 사항에 대한 피드백을 받기 위해 시스템을 실험적으로 만들어 사용자에게 보여주고 평가하게 하는 방법
- 사용자와 의사 소통하는 도구로 활용
- Prototyping 도구
 - 화면 생성기
 - 시스템의 작동을 시뮬레이션 하여 사용자가 볼 수 있는 반응을 보여줌
- 프로토타입의 목적
 - Throw-away prototyping: 단순한 요구 추출 목적으로 만들고 버림
 - 제작 가능성 탐진: 개발 단계에서 유지보수가 이루어짐

Prototyping Model



Prototype Model 개발 절차

1. 요구사항 정의 및 분석
 - 1차로 개략적인 요구사항을 정의한 후 2차, 3차, ... n차를 반복하면서 완성도를 높여 최종 프로토타입을 개발
2. 프로토타입 설계
 - 동작이 가능한 최종 코딩을 위한 설계가 아니라, 입력 화면과 출력 화면을 통해 사용자가 만들어질 시스템이 어떻게 수행되는지 파악할 수 있도록 사용자 인터페이스를 중심으로 설계
3. 프로토타입 개발
 - 프로토타입 모델의 취지에 맞게 가상 수행을 전제로 한 실행을 보여주는 것이 우선
4. 사용자에 의한 프로토타입 평가
 - 프로토타입 모델에서 매우 중요한 단계
 - 사용자는 1차로 개발된 프로토타입을 보며 요구사항이 충실히 반영되었지를 확인
 - 사용자는 확인이 끝나면 추가 및 수정 요구사항을 전달
 - 개발자는 이를 반영해 2차 설계를 한 뒤 그에 따른 2차 프로토타입을 개발
 - 같은 과정을 n번 반복해 사용자의 추가 요구사항이 더 이상 없을 때 최종 프로토타입이 만들어짐

Prototyping Model 장단점

장점

- 프로토타입이 개발자와 사용자 간의 의사소통 도구로 사용되어 구체적이고 원활하게 대화할 수 있음
- 요구사항을 여러 번 반복하는 과정을 통해 사용자의 요구가 충분히 반영된 요구분석명세서를 만들 수 있음
- 개발되는 소프트웨어가 어떤 모습인지 예측할 수 있으므로 개발 초기에 만족감을 가져 개발에 적극적으로 참여
- 사용자의 요구가 충분히 반영되어 최종 제품이 나오므로 유지보수에 필요한 노력과 시간을 많이 줄일 수 있음

Prototyping Model 장단점 (cont'd)

단점

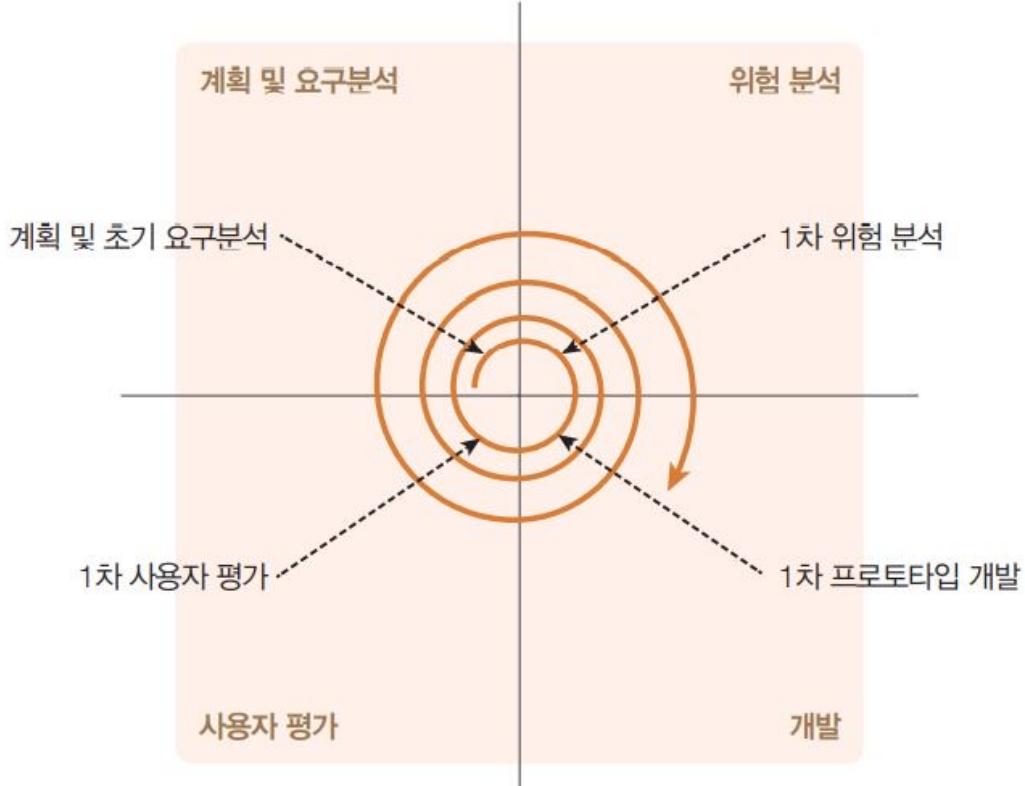
- 반복적인 소프트웨어 개발 단계로 인해 필요한 투입 인력과 비용 산정이 어려움
- 개발된 프로토타입으로는 완전히 동작할 수 없는데도 빠른 시간 안에 최종 결과가 나올 것이라고 사용자가 착각할 수 있음
- 중간 점검을 할 수 있는 이정표나 산출물을 생성할 수 없어 프로토타이핑 과정을 관리·통제하기 어려움
- 개발 범위가 명확하지 않아 소프트웨어 개발의 목표나 종료 시점이 불명확해질 수 있음
- 프로토타입에 따른 추가 비용이 들 수 있음

적용

- 개발 착수 시점에 요구가 불투명할 때
- 실험적으로 실현 가능성은 타진해 보고 싶을 때
- 혁신적인 기술을 사용해 보고 싶을 때

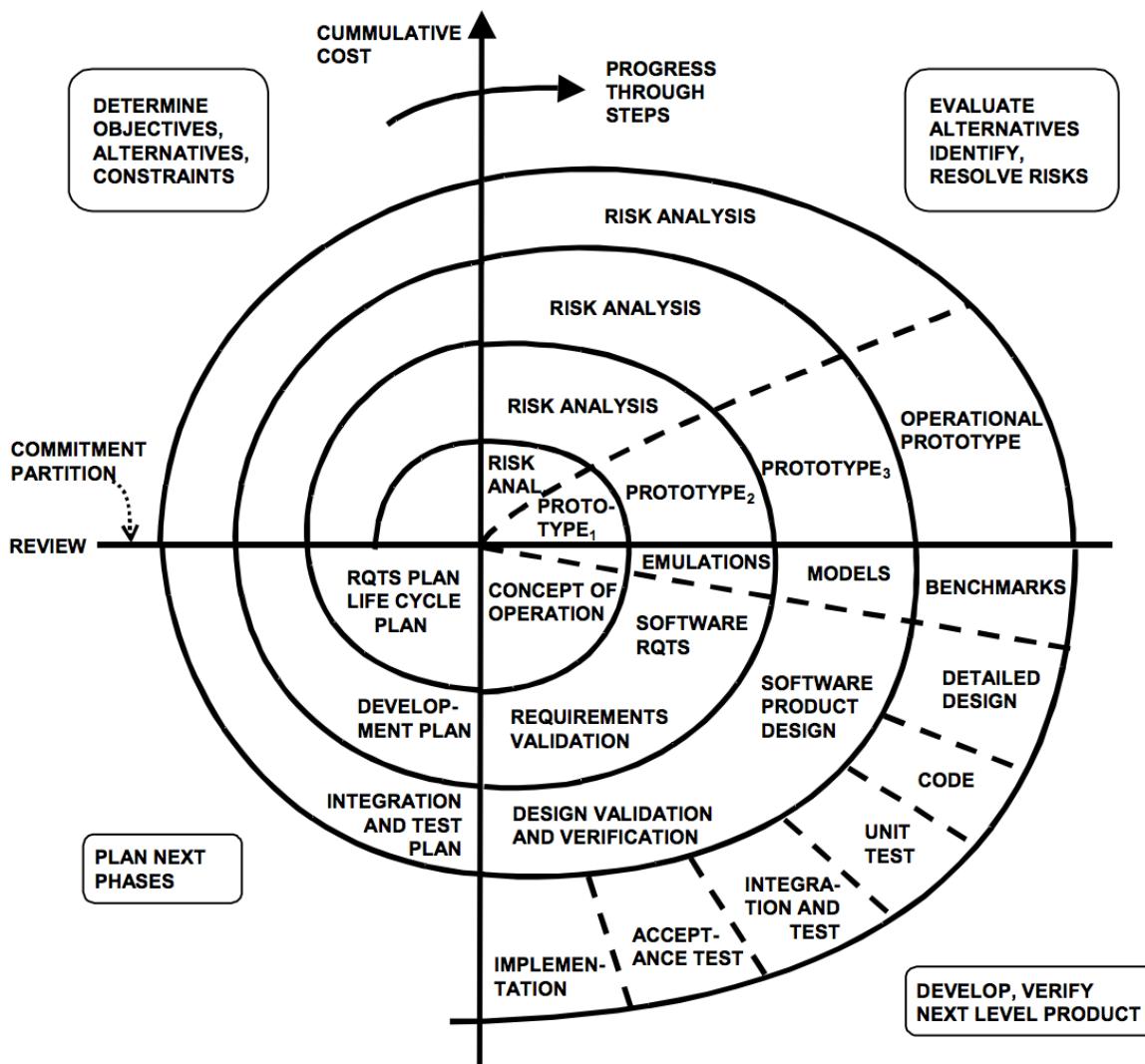
Spiral (나선형) Model

- "A Spiral Model of Software Development and Enhancement"
Berry Boehm, 1986
- **Risk driven software development**
- 위험 분석 단계에서 위험 요소는 소프트웨어 개발 과정이 순조롭게 진행되는 데 방해되는 모든 것
- 초기 요구분석 후 프로토타입 개발 이전에 위험 분석 단계를 거침



Spiral (나선형) Model 개발 절차

1. 계획 및 요구분석 단계
 - 사용자의 개발 의도를 파악해 해당 프로젝트의 목표를 명확히 하고, 여러 제약 조건의 대안을 고려한 계획을 수립
 - 사용자의 요구를 통해 파악 한 기능 요구사항과 성능 같은 비기능 요구사항을 정의하고 분석
2. 위험 분석 단계
 - 프로젝트 수행에 방해되는 위험 요소를 찾아 목록을 작성하고 위험에 대한 예방 대책을 논의
 - 심각한 위험이 존재하는 경우에는 해당 프로젝트를 계속 진행해도 되는지를 결정
3. 개발 단계
 - 프로토타입 개발, 다른 소프트웨어 개발 프로세스의 설계와 구현에 해당
4. 사용자 평가 단계
 - 사용자가 만족할 때까지 n번 반복해 더 이상의 추가 및 수정 요구가 없으면 최종 제품을 개발
 - 사용자 평가 단계는 진화적 프로토타입 모델에서 매우 핵심적이고 중요



Spiral (나선형) Model 장단점

장점

- 사전에 위험을 의식하고 개발하기 때문에 프로젝트가 중단되는 심각한 사태가 일어날 확률이 비교적 적음
- 사용자 평가가 반영된 반복적 개발 방식에 의해 사용자 요구가 충분히 반영되어 사용자의 불만이 적음

단점

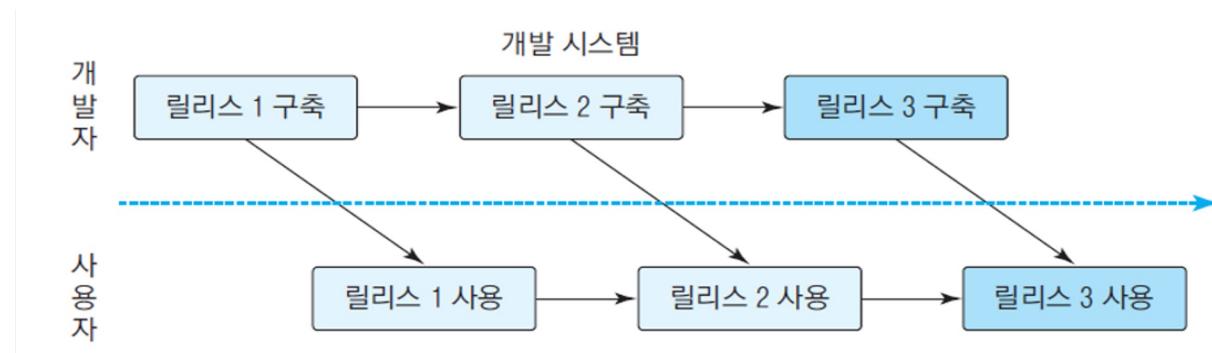
- 각 단계가 반복적으로 계속 진행되기 때문에 프로젝트 기간이 길어질 수 있음
- 반복 횟수가 많아질수록 프로젝트 관리가 어려움
- 위험 관리가 중요한 만큼 위험 관리 전문가가 필요하다는 부담감
- 잘못된 위험 분석으로 인한 피해 가능성
- 성공 사례가 많이 알려지지 않음

적용

- 재정적 또는 기술적으로 위험 부담이 큰 경우
- 요구 사항이나 아키텍처 이해에 어려운 경우

Phased Development Model (단계적 개발 모델)

- 개발자가 먼저 릴리스 1을 개발해 사용자에게 제공하면 사용자가 이를 사용
- 사용자가 릴리스 1을 사용하는 동안 개발자는 다음 버전인 릴리스 2를 개발
- 개발과 사용을 병행하는 과정을 반복해 진행하면서 완료
- 릴리스를 구성하는 방법
 - 점증적 방법: 기능별로 릴리스
 - 반복적 방법: 릴리스 할 때마다 기능의 완성도를 높임



단계적 개발 모델: 점증적 개발 방법 (개발 범위 증가)

- 점증적 개발 방법: **개발 범위의 증가**
 - 중요하다고 생각되는 부분부터 차례로 개발한 후 그 일부를 사용하면서 개발 범위를 점차 늘려 가는 방식
 - Analogy: 코스 요리 (하나가 끝나면 또 하나를 먹음)
 - 예) 대학 종합정보시스템 개발 시
 - 대학에서 사용하는 종합정보시스템을 개발할 경우, 가장 중요한 교무/학사 관련 시스템을 먼저 개발해 사용
 - 회계 업무를 비롯한 다른 부서의 업무 시스템을 차츰 개발 해나가는 방식

단계적 개발 모델: 점증적 개발 방법 (개발 범위 증가)

- 점증적 방법
 - 요구분석명세서에 명시된 시스템 전체를 기능에 따라 독립성 높은 서브 시스템으로 분할
 - 각 서브시스템을 단계적으로 하나씩 릴리스해 완성하는 방법
- 점증적 방법의 장단점
 - 한꺼번에 많은 비용을 들이지 않아도 됨
 - 완전히 새로운 시스템 전체를 한 번에 주었을 때 조직이 받는 충격을 완화할 수 있음
 - 소프트웨어를 단계적으로 도입하면 조직에 자연스럽게 변화를 줄 수 있음
 - 이미 사용하고 있는 서브시스템이 있어 어떤 유형으로 개발해야 하는지 잘 알 수 있음
 - 처음 설계할 때부터 이후에 개발할 다른 서브시스템과의 연관성을 고려해야 함
 - 이미 개발된 서브시스템들과 통합하는 데 어려움을 겪을 수 있음

단계적 개발 모델 : 반복적 개발 방법 (품질의 증가)

- 반복적 개발 방법: 품질의 증가

- 초기에 시스템 **전체**를 일차적으로 개발해 인도한 후, 각 서브시스템의 기능과 성능을 변경 및 보강해 **완성도**를 높임
- 초기의 요구사항이 불분명한 경우에 적합
- Analogy: 한식 (모든 음식을 한 상에 가득 차려놓고 조금씩 모두 맛본 후 맛있는 음식은 여러 번 먹음)

단계적 개발 모델 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

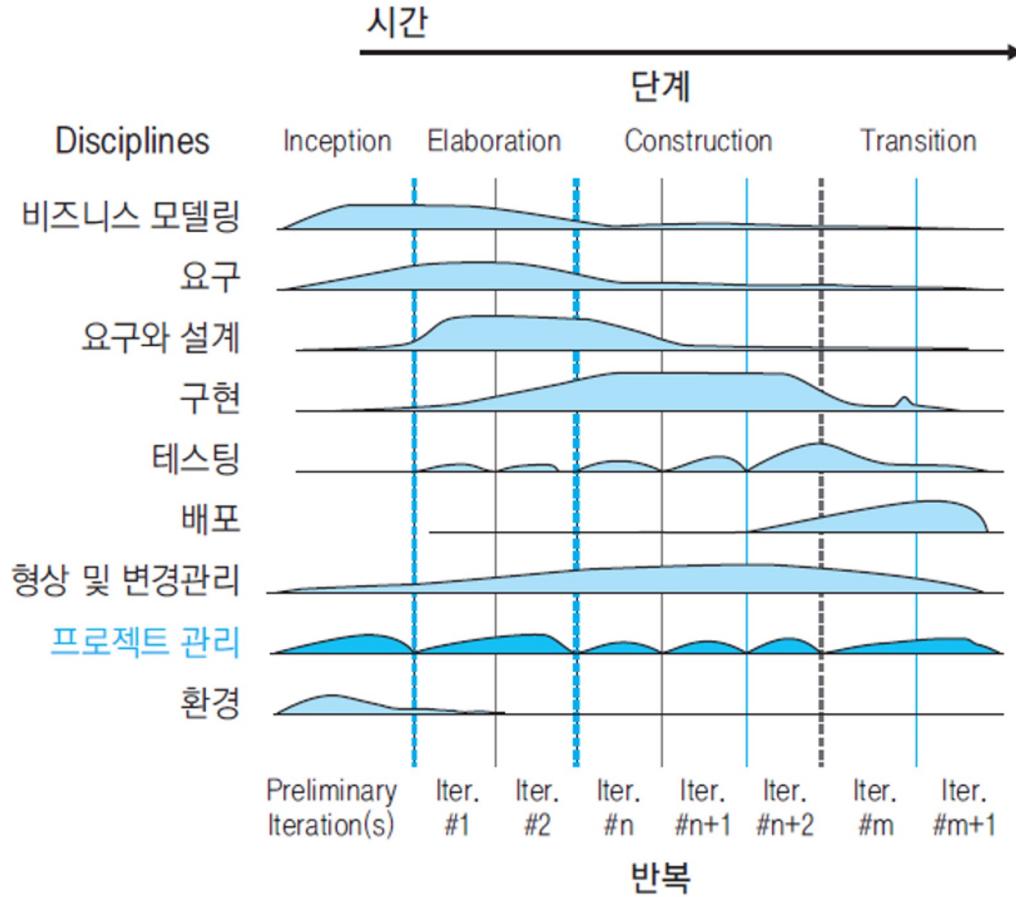
- 프로젝트 관리가 복잡해지기 때문에 큰 프로젝트 부적합
- 끝이 안보일 수 있어 실패의 위험이 커짐
- 프로젝트의 진행이 위험 분석에 크게 의존

Software Development Process: United Process Models

Unified Process Model

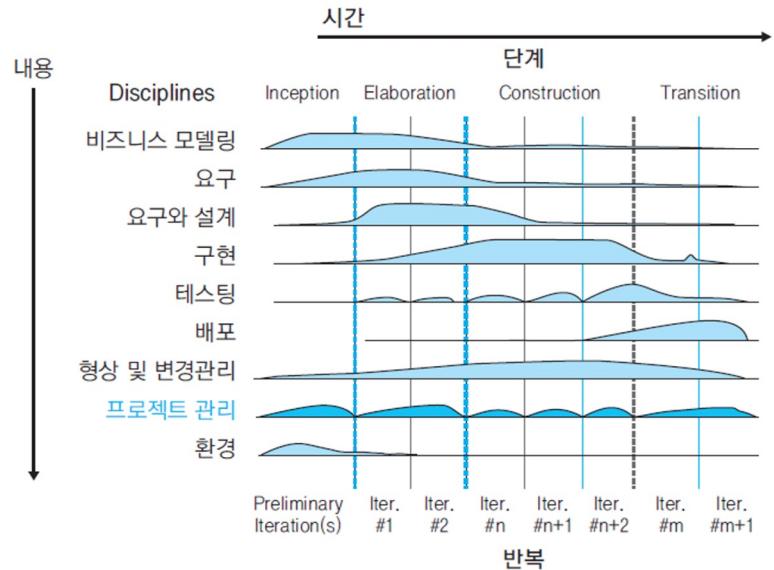
- 도입·구체화·구축·전이 단계의 공통 작업
 - 분석, 설계, 구현, 테스트 작업을 공통으로 수행하되, 각 단계별로 수행하는 정도에는 차이가 있음
 - 형상 및 변화 관리, 프로젝트 관리, 환경 점검 등은 지속적으로 수행

내용



Unified Process Model

- Inception(도입)
 - 1, 2회 정도 반복으로 도입 단계를 진행
 - 간단한 유스케이스 모델과 소프트웨어 구조, 프로젝트 계획을 작성
- Elaboration (구체화)
 - 여러 번의 반복 과정으로 이루어짐
 - 대부분의 유스케이스를 작성
 - 아키텍처 설계
- Construction (구축)
 - 남아 있는 유스케이스에 대하여 구현하고 통합
 - 시스템을 목표 환경에 점증적으로 설치
- Transition (전환)
 - 시스템을 배치, 사용자를 교육
 - 베타 테스팅, 결함 수정, 기능 개선



Unified Process

장점

- 방법론과 프로세스가 잘 문서화되어 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

Software Development Process: Agile Process Models

Agile Process Models 의 이해

- 애자일 프로세스 모델
 - 애자일(agile): ‘날렵한’, ‘민첩한’
 - 애자일 프로세스 모델: 고객의 요구에 민첩하게 대응하고 그때그때 주어지는 문제를 풀어나가는 방법론
 - 가벼운 프로세스 방법론의 공통적인 특성을 정의
 - 2001년 1월에 ‘애자일 연합’ 그룹에서 서로의 공통점을 찾아 ‘애자일 선언문’이라는 공동 선언서를 발표

Agile Process Models 의 이해

- Agile의 기본 가치
 - 프로세스와 도구 중심이 아닌, **개개인과의 상호 소통**을 중시
 - 문서 중심이 아닌, **실행 가능한 소프트웨어**를 중시
 - 계약과 협상 중심이 아닌, 고객과의 **협력을** 중시
 - 계획 중심이 아닌, 변화에 대한 **민첩한 대응**을 중시
 - 환경과 고객의 **변화에 능동적으로 대처**하는 것을 강조

Agile Process Models 의 이해

- 애자일 프로세스 개발 방법
 - 가장 기본이 되는 기능만 1차 요구사항으로 분석하고 이를 반복으로 나누어 개발
 - 사용자가 릴리스 1을 사용하는 동안 개발자는 2차 개발을 진행
 - 이때 좀 더 복잡한 편집 기능과 고급 기능을 추가해 마찬가지로 이를 반복으로 나누어 개발
 - 프로세스 개발 방법은 **반복적인 개발을 통한 잦은 출시를 목표**로 함
 - 실행 가능한 프로토타입을 만들어 사용자에게 확인받음
 - 좀 더 빠른 시간 안에 일부지만 소프트웨어를 사용할 수 있게 하는 것을 중요하게 생각

Agile Process vs. Waterfall Model

구분	애자일 프로세스 모델	폭포수 모델
추가 요구사항의 수용	처음 수집한 요구사항을 전체 중 일부로 인정하고 시작하므로 언제든지 추가 요구사항이 있을 것으로 간주한다. 따라서 추가 요구사항을 수용할 수 있는 방법으로 설계되어 있다.	요구사항 분석이 완전히 완료된 후에 설계 단계로 넘어가므로 새로운 요구사항을 추가하기 쉽지 않다. 추가 요구사항을 반영하기 어려운 구조다.
릴리스 시점	가능하면 자주, 빨리 제품에 대한 프로토타입을 만들어 사용자에게 보여준다. 이러한 방식을 반복적으로 수행해 최종 제품을 만들기 때문에 자주 릴리스된다.	요구사항에 대한 분석, 설계, 구현 과정이 끝나고 최종 완성된 제품을 릴리스한다.
시작 상태	계속적인 추가 요구사항을 전제로 하는 방식이라 시작 단계에서는 부족한 점이 많지만 점차 완성도가 높아진다.	한 번 결정된 단계는 그 이후에 변동이 적어야 한다. 따라서 완성도를 최대한 높여 다음 단계로 넘어가기 위해 시작 단계의 완성도가 매우 높다.

Agile Process vs. Waterfall Model

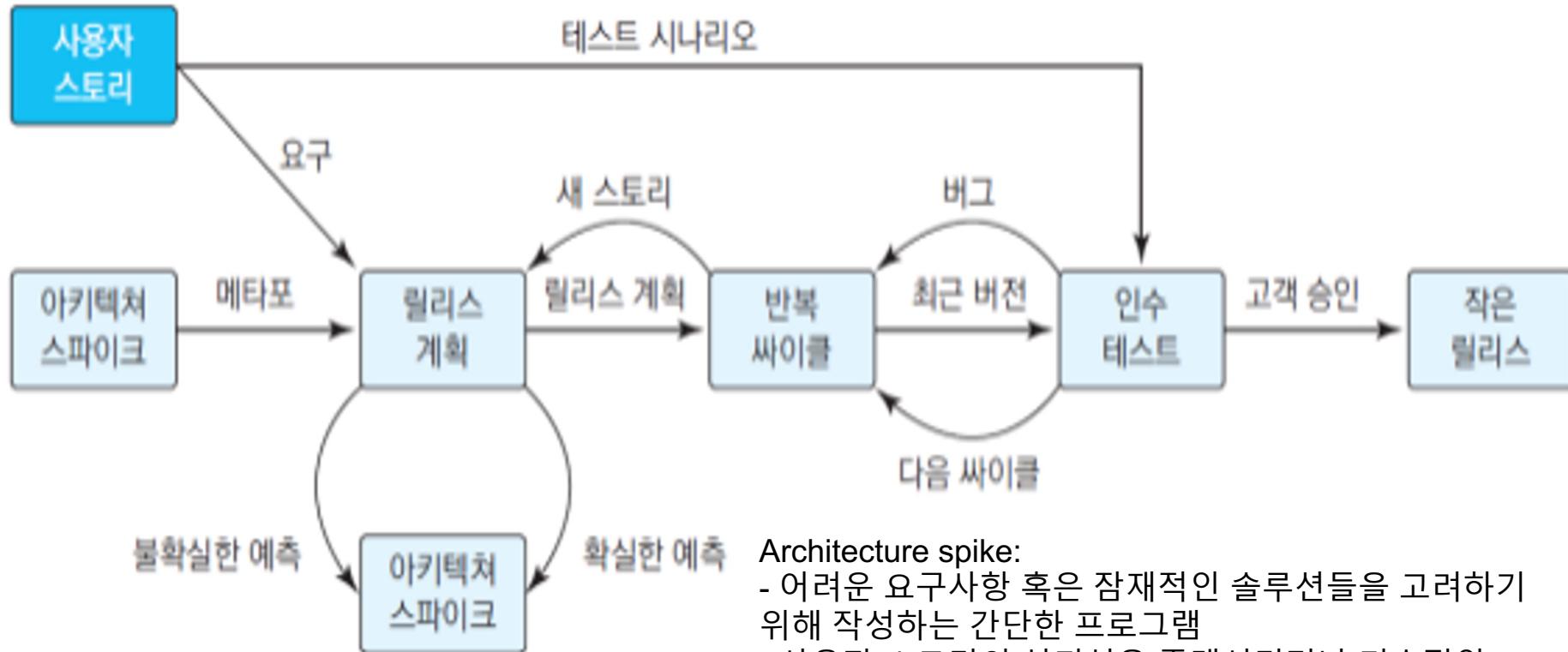
	agile	waterfall
고객과의 의사소통	사용자와 함께 일한다는 개념을 담고 있다. 처음부터 사용자의 참여를 유도하고 많은 대화를 하면서 개발을 진행한다.	사용자 요구사항을 정의한 후 사용자에게 더는 추가 요구가 없다는 확답을 받고 개발에 들어간다. 산출물을 근거로 하기 때문에 사용자와의 대화는 적다.
진행 상황 점검	개발자와 사용자는 개발 초기부터 지속적으로 진행 상황을 공유하며 함께 관심을 갖고 진행해 나간다.	단계별 산출물을 중요시하기 때문에 단계별 산출물에 대한 결과로 개발의 진척 상황을 점검한다.
분석, 설계, 구현 진행 과정	분석, 설계, 구현이 하나의 단계와 그 단계 안의 반복 마다 한꺼번에 진행된다. 다만 어떤 단계에서는 분석이 많고 구현이 적고 어떤 단계에서는 분석이 적고 구현이 많다는 차이만 있을 뿐이다.	분석, 설계, 구현 과정이 명확하다. 각 과정에서 생산되는 산출물 중심의 개발 방식이기 때문에 단계가 명확히 구별되어 있다. 따라서 분석이 끝난 후 설계를, 설계가 끝난 후 구현 작업을 진행한다.
모듈(컴포넌트) 통합	개발 초기부터 빈번한 통합을 통해 문제점을 빨리 발견하고 수정하는 방식을 택한다. 문제점을 빨리 발견 하므로 비용을 절감할 수 있다는 장점이 있다.	V 모델에서 설명한 것처럼 구현이 완료된 후에 모듈 간의 통합 작업을 수행한다.

Software Development Process: Agile Process Models eXtreme Programming (XP)

eXtreme Programming (XP)

- XP 자체는 기존의 방법론에 비교해 볼 때 매우 가벼운 기법이며 실용성(pragmatism)을 강조한 것
- XP의 목표: ‘고객에게 최고의 가치를 가장 빨리’
- 개발자, 관리자, 고객의 조화로 개발 생산성을 높이고, 고객 요구사항 변경에 적극적이고 긍정적으로 대응
- XP는 의사소통(communication), 단순함(simplicity), 피드백(feedback), 용기(courage), 존중(respect) 등 5가지의 가치에 기초
- XP는 개발 속도를 높이는 가속 기술이며, 그 중심은 **단순한 디자인 정신, 테스트 우선 프로그래밍, 리팩토링.**

eXtreme Programming (XP)



Architecture spike:

- 어려운 요구사항 혹은 잠재적인 솔루션들을 고려하기 위해 작성하는 간단한 프로그램
- 사용자 스토리의 신뢰성을 증대시키거나 기술적인 문제의 위험을 줄이고자 하는 데 목적

표 3-1 XP 프로세스의 10가지 실천 사항

실천 사항	가이드라인
증분 계획 (Incremental Planning)	사용자 요구사항을 스토리 카드에 기록하며, 배포 시 들어갈 스토리는 사용 가능한 시간과 상대적 우선순위에 따라 결정한다. 개발자는 이러한 스토리를 개발의 단위 태스크로 분리한다.
작은 배포 (Small Release)	비즈니스 가치를 제공할 수 있는 최소한의 유용한 기능을 먼저 개발하여 배포한다. 소프트웨어 시스템 배포의 경우, 첫 번째로 배포된 소프트웨어에 기능을 점진적으로 추가하면서 배포한다.
단순 설계 (Simple Design)	추후 나올 가능성이 있는 요구사항이 아닌 현재의 요구사항을 충족시키도록 설계를 수행한다.
테스트 우선 개발 (Test-first Development)	기능 자체가 구현되기 전에 새로운 기능에 대한 테스트를 수행하기 위해 자동화된 단위 테스트 프레임워크를 사용한다.
재구조화 (Refactoring)	모든 개발자는 코드 개선이 가능한 부분을 발견하는 즉시 코드를 재구조화해야 한다. 이를 통해 코드를 간단하고 쉽게 유지보수할 수 있다.

짝 프로그래밍 (Pair Programming)	개발자는 쌍으로 작업하여 서로의 작업을 확인하고, 항상 훌륭한 작업을 수행할 수 있도록 지원한다.
공동 소유권 (Collective Ownership)	개발자들은 시스템의 전 영역에 개발 작업을 수행함으로써, 경험과 전문 지식이 상호 연관되어 적용될 수 있다. 이로 인하여 모든 개발자가 모든 코드를 책임을 지며, 누구나 아무거나 변경할 수 있도록 한다.
지속적 통합 (Continuous Integration)	단위 테스크가 완료되자마자 이는 전체 시스템에 통합한다. 통합한 후, 시스템에 대한 모든 단위 테스트가 통과되어야 한다.
지속 가능한 개발 속도 (Sustainable Pace)	초과 근무가 많아지면 종종 코드 품질이 저하되고 중간 생산성이 감소될 수 있으므로 허용되지 않는다.
현장 고객 (On-site Customer)	고객/사용자 대표는 XP 개발 팀이 항상 접촉할 수 있어야 한다. XP 프로세스에서 고객은 개발 팀의 구성원이며, 고객은 소프트웨어 구현을 위해 시스템 요구사항을 팀에 제공해야 한다.

User Story (사용자 스토리)

- User story는 고객이 원하는 기능을 짧게 표현해 놓은 것
- 해당 기능에 대해 간략하게 설명하거나 기능을 대표하는 키워드를 포함하는 짧은 문장이 포함
- User story를 만들어 고객과 직접 소통.
- 사용자의 요구사항은 언제든지 변할 수 있으며, 사용자 조차도 자신의 요구사항을 정확히 알지 못하는 경우가 대부분이라는 것을 가정

스토리ID	M102	작성일자	2024-09-02
우선순위	상□ 중□ 하□	추정	1주
담당개발자	홍길동		
스토리	쇼핑몰 회원은 카테고리를 선택하여 카테고리에 속한 상품의 목록을 조회한다.		
비고	하위카테고리가 존재하는 카테고리에는 상품이 포함되지 않는다. 최하위 카테고리를 선택한 경우에만 상품 목록이 조회되어야 한다.		

좋은 User Story

- 독립적이다 (Independent)
- 협상 가능하다 (Negotiable)
- 사용자와 고객에게 가치가 있다 (Valuable)
- 추정 가능하다 (Estimable)
- 작다 (Small)
- 테스트 가능해야 한다 (Testable)

XP의 테스트

- XP 프로세스에서 이루어지는 테스트의 특징
 - 테스트를 작성하는 작업은 요구사항을 밝히는 고객과 함께 협동하여 수행한다.
 - 테스터와 개발자는 적대적 관계가 아닌 협력 관계를 유지해야 한다.
 - 프로그램을 작게 나누어 테스트를 자주 수행한다.
- 테스트 케이스 작성 시점
 - 고객과 개발자가 스토리에 대해 토론하는 과정에서 도출된 세부 사항을 기록하기 위해
 - 스토리의 구현을 시작하기 전 개발자가 스토리를 명확하게 이해하고자 할 때
 - 프로그래밍 중 또는 그 이후라도 스토리에 필요한 새로운 테스트를 발견할 때

User Stories to Tasks

- User story를 개발 작업(Task)으로 분해하고 구현에 필요한 노력과 자원을 추정
- Task는 구현의 기본 단위이다.
 - 개발자가 스토리를 구현하는 과정에서 해야 할 임무를 명확히 하기 위해 필요
 - 스토리를 구현하는 데 필요한 일정 계획을 세우기 위해 필요
 - 개발자가 해야 할 일의 목록을 구체적으로 작성하고 나면 개발자는 각각의 작업에 소요되는 시간을 더 정확하게 추정할 수 있게 된다.
- Task에 소요되는 시간의 합계는 Story를 구현하는데 소요되는 시간

Example: User Story, Test Case

- **User Story**
 - 회원은 카테고리를 선택하여 카테고리에 속한 상품의 목록을 조회한다.
- **Test Case**
 1. 사용자가 선택한 카테고리가 하위 카테고리를 갖는 경우 하위 카테고리의 목록을 보여준다.
 2. 최하위 카테고리를 선택한 경우 카테고리에 속한 상품 목록을 보여준다.
 3. 임의의 시점에서 조회중인 카테고리의 상위 카테고리 목록을 조회할 수 있어야 한다.
 4. 하위 카테고리가 존재하지 않는 최하위 카테고리를 선택하였으나 카테고리에 속한 상품이 존재하지 않는 경우 상품이 없음을 사용자에게 알린다.

Example: Tasks

1. 카테고리 목록 조회

- 최초 화면 조회 시에는 최상위 카테고리의 목록을 출력(상위카테고리명=null)
- 사용자가 선택한 카테고리의 카테고리명을 입력 받아 하위 카테고리의 목록 출력
- 상위 카테고리 목록을 조회하는 기능

2. 상품 목록 조회

- 사용자가 선택한 카테고리의 하위 카테고리 목록을 조회한 결과가 null인 경우 최하위 카테고리로 판단
- 최하위 카테고리가 선택되면 카테고리명을 입력 받아 카테고리에 속한 상품의 목록 출력
- 최하위 카테고리에 속한 상품의 목록이 비어 있는 경우 상품 목록 출력 위치에 메시지 출력 “등록된 상품이 없습니다.”

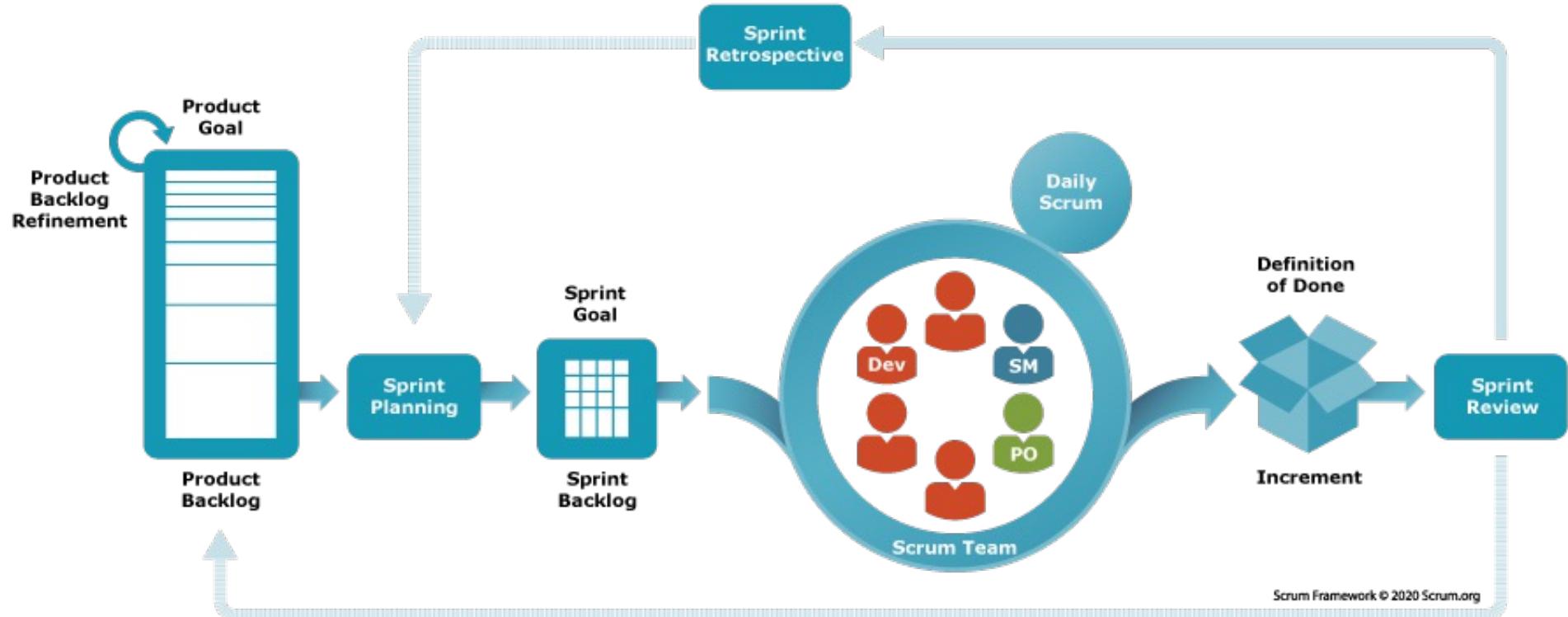
Software Development Process: Agile Process Models Scrum

Scrum

- **Deliver value incrementally in a collaborative manner**
- <https://www.scrum.org/>
- 개발 팀원 모두가 함께 소통하고 협력하여 짧은 주기를 반복하며 소프트웨어를 개발하는 작업, 역할, 결과물
- Backlog를 정하고 여기에 우선순위를 부여 (Prioritization)
- 짧은 주기(Sprint)



SCRUM FRAMEWORK



Scrum Framework © 2020 Scrum.org

Scrum 용어

- **Sprint** : 반복적인 개발 주기(계획 회의부터 제품 리뷰가 진행되는 날짜까지의 기간이 1 스프린트)
- **Product Backlog** (제품 백로그): 개발할 제품에 대한 요구사항 목록
- **Sprint Backlog** (스프린트 백로그): Sprint 목표에 도달하기 위해 필요한 작업 목록
- **Increment** (제품 증분): Sprint 결과로 나오는 실행 가능한 제품
- **Burndown chart** : Sprint backlog 남아 있는 작업 목록을 보여주는 차트
- **Burnup chart** : 배포에 필요한 진행 사항(진척도)을 보여주는 차트

Scrum 개발 관련자의 역할

- **Product Owner (제품 책임자)**
 - 제품 기능 목록을 만듦
 - 비즈니스 관점에서 우선순위와 중요도를 매기고 새로운 항목을 추가함
 - Sprint 계획 수립 시 까지만 역할을 수행하고 Sprint가 시작되면 팀 운영에 관여하지 않음
- **Scrum Master**
 - Product owner를 돋는 조력자
 - 업무를 배분만 하고 일은 강요하지 않음
 - Scrum 팀이 스스로 조직하고 관리하도록 지원함
 - 개발 과정에서 Scrum의 원칙과 가치를 지키도록 지원
 - 개발 과정에 방해될 만한 요소를 찾아 제거함

Scrum 개발 관련자의 역할 (con'td)

- **Scrum Team**
 - 팀원은 보통 5-9명으로 구성되며, 사용자 요구사항을 User story로 도출하고 이를 구현함
 - 기능을 작업 단위로 나누고, 일정이나 속도를 추정해서 Product Owner에게 알려줌
 - 하나의 Sprint에서 생산된 결과물을 Product Owner에게 시연함
 - 매일 스크럼 회의에 참여하여 진척 상황을 점검함

Scrum 진행 과정

단계	수행 목록	내용
1	제품 기능 목록 작성	<ul style="list-style-type: none">• 요구사항 목록에 우선순위를 매겨 제품 기능 목록 작성
2	스프린트 계획 회의	<ul style="list-style-type: none">• 스프린트 구현 목록 작성• 스프린트 개발 기간 추정
3	스프린트 수행	<ul style="list-style-type: none">• 스프린트 개발• 일일 스크럼 회의• 스프린트 현황판 변경• 소멸 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 회의• 스프린트 회고• 두 번째 스프린트 계획 회의

Scrum: Product Backlog (제품 백로그)

Product backlog

- 제품 개발에 필요한 모든 업무를 우선순위한 목록
- 요구사항 정의서 + 작업 분류 체계(Work Breakdown Structure)
- 소프트웨어와 하드웨어 등 각 업무 파트에서 수행해야 할 요구 기능을 중심으로 기술
- 해당 기능을 수행하는 세부 작업(상세 분석과 설계, 구현, 단위 테스트 등)은 포함하지 않는다.
 - 이런 작업들은 주기적으로 수행되는 스프린트 계획에서 도출
- 프로젝트의 전체 요구사항을 구현하는 데 필요하거나 선행해야 하는 작업(프로젝트 관리와 지원, 요구 분석 등)은 기술한다.

Scrum: User Story

- 사용자 스토리(user story)는 제품 백로그에서 기능 요구사항을 기술할 때 사용하는 방식으로, 고객과 사용자에게 가치를 줄 수 있는 기능을 서술
- 제품 기능이 도출되면 각 기능 + Acceptance Criteria 을 간략하게 서술
- 주로 한 장의 인덱스 카드 (포스트잇)로 작성
- 기능을 상세하게 적는 것이 아니라 개발자와 대화를 지속할 수 있는 단서 정도로만 활용할 수 있게 작성
- “**As a [personona], I [want to] _____, [so that] _____.**”
 - As a student I want to add, delete and view courses so that class registration is done.
- “[누가] [비즈니스 가치]를 위해 [어떤 기능]을 원한다 ”
 - 학생은 수강 신청을 위해 강의 신청, 취소, 리스트 보기 기능을 할 수 있다.

Scrum: User Story

- Ron Jeffries 가 제시한 사용자 스토리의 구성 요소
 - Card (카드): 스토리를 추정하거나 계획하는 데 활용한다. 기능을 서술하는 것으로 포스트잇을 많이 사용
 - Conversation (대화): 사용자와 개발자가 충분한 대화를 통해 요구사항을 도출
 - Confirmation(확인): 스토리가 완료되는 조건을 서술

Scrum: User Story (cont'd)

- **Story points estimation**

- 스토리 포인트: 요구사항의 규모를 측정하는 단위로 업무량을 이용해 산정
- 스토리 간의 **상대적인 업무량을 비교해** 가장 적을 때를 1로 두고 이를 기준으로 얼마나 큰지에 따라 스토리 포인트를 산정
- 일반적인 소프트웨어 개발 방법론에서는 개발에 소요되는 시간을 일/주/월의 시간 단위로 예측
- 애자일 방법론에서는 스토리 포인트라는 **추상 개념으로 예측**

Scrum: User Story

- **Acceptance Criteria**

- 완료 조건은 사용자 스토리의 완성 여부를 객관적으로 확인할 수 있는 조건을 기술한 것.
- 구현 결과가 제품 책임자의 기대사항과 달라지는 것을 예방한다.
- 초기 제품 백로그에서는 개략적으로 기술하고 해당 스토리를 본격적으로 개발하는 스프린트 계획 미팅에서 상세히 구체화한다.
- 구체화한 완료 조건을 기반으로 상세 테스트 케이스를 작성할 수 있다.
- 실제 프로젝트에서는 완료 조건 외에 제약 조건이나 참조 사항 등을 추가로 기술하는 것이 사용자 스토리 이해에 도움이 된다.

User Story Template Example

Title:	Priority:	Estimate:
<p>User story</p> <p>As a [type of user],</p> <p>I want to [perform some task]</p> <p>so that I can [achieve some goal].</p>		
<p>Acceptance criteria</p> <p>Given that [some context],</p> <p>when [some action is carried out]</p> <p>then [a set of observable outcomes should occur].</p>		

Scrum: Product backlog Example

표 1-3 제품 기능 목록의 예: 학사관리시스템

순위	기능	사용자 스토리	포인트	중요도
4	교과목등록	직원은 교과목등록 기간을 설정한다.	4	중
		교수는 교과목을 등록, 수정, 삭제한다.	7	
		사용자(교수/학생/직원/조교)는 등록된 교과목을 조회한다.	5	
3	개설과목등록	직원은 개설과목 기간을 설정한다.	4	중
		교수는 개설과목을 등록, 수정, 삭제한다.	7	
		사용자는 등록된 개설과목을 조회한다.	5	
1	수강신청	직원은 수강신청 기간을 설정한다.	4	상
		직원은 수강신청 변경 기간을 설정한다.	4	
		학생은 수강신청 과목을 등록, 수정, 삭제한다.	7	
		사용자는 학생의 수강신청 내역을 조회한다.	5	
2	성적등록	직원은 성적등록 기간을 설정한다.	4	중
		교수는 성적을 입력, 수정, 삭제한다.	7	
		사용자는 성적을 조회한다.	5	
...

Sprint 계획 회의 (planning)

- **Sprint 계획 수립**
 - Sprint: 작업량이 그렇게 많지 않고 개발 기간도 짧은 경우를 의미
 - 작은 단위의 개발 업무를 단기간 내에 전력 질주해 수행
 - 하나의 스프린트에 어떤 사용자 스토리를 몇 개 포함할 것인지는 스프린트 계획 회의에서 결정
 - 보통 1~4주 정도를 하나의 스프린트로 봄
 - 외부의 개발 방해 요소를 차단하는 것은 스크럼 마스터의 역할
 - 하나의 스프린트 개발이 끝나면 사용자에게 시연하고 사용자는 피드백을 제공
 - 계획된 일정 안에 개발을 마치지 못해도 정해진 일정이 끝나면 하나의 스프린트가 끝남

Sprint 계획 회의 (planning)

- 스프린트 계획 회의
 - 전체적인 스프린트 계획 회의
 - 사용자의 대변인격인 product owner를 통해 사용자가 원하는 것이 무엇인지를 파악하는 데 중점을 둠
 - 이를 위해 scrum master는 제품 기능 목록을 검토하면서 어떤 항목을 가장 높은 순위로 놓았는지 확인
 - 그 배경과 목표에 대해 팀원들과 토의하며 product owner의 의도를 파악
 - 세부적인 스프린트 계획 회의
 - 우선순위가 높은 항목을 어떻게 구현할 것인지 구체적인 작업 계획을 설립
 - 먼저 우선순위가 매겨진 사용자 요구사항 목록인 제품 기능 목록에서 개발 항목을 결정
 - 스프린트 구현 목록을 작성한다. 여기에 팀원들은 정해진 작업을 수행하는 데 소요되는 시간을 추정

Sprint 목록 예

- 스프린트 구현 목록 작성
 - 제품 기능 목록에 있는 스토리 중에서 선택해 작성
 - 스프린트 구현 목록은 스프린트 계획 회의에서 결정
 - 기준은 기간 내에 완료할 수 있는 만큼의 사용자 스토리
 - 실행 가능한 결과가 나올 수 있는 수준에서 결정

표 1-4 스프린트 구현 목록의 예

스프린트	사용자 스토리	SP	작업	MD	개발자
수강신청	학생은 수강신청 과목을 등록/수정/삭제/조회할 수 있다.	20	UI 설계	1	김희석
			웹페이지 설계	2	
			장바구니 구현	2	
			DB 테이블 설계	3	
			단위 테스트	1	
			코드 검토	1	
성적등록	교수는 성적을 등록/수정/삭제/조회할 수 있다.	16	UI 설계	1	서응식
			웹페이지 설계	2	
			DB 테이블 설계	2	
			단위 테스트	1	
			코드 검토	1	
교과목, 개설 과목 등록	교수는 교과목을 등록/수정/삭제/조회할 수 있다.	9	UI 설계	1	박양구
			웹페이지 설계	2	
			DB 테이블 설계	2	
			단위 테스트	3	
			코드 검토	1	
	교수는 개설과목을 등록/수정/삭제/조회할 수 있다.	9	UI 설계	1	이칠현
			웹페이지 설계	2	
			DB 테이블 설계	2	
			단위 테스트	3	
			코드 검토	1	
...

※ SP: 스토리 포인트 Story Point, MD: 연 작업 시간 Man Day

Sprint 수행

- Daily Scrum Meeting (일일 스크럼 회의)
 - 보통 매일, 서서 약속된 시간에 짧게(15분 정도) 진행
 - 진행 상황만 점검
 - 스프린트 작업 목록을 잘 개발하고 있는지 확인
 - 모든 팀원이 참석
 - 개별 팀원에 대한 진척 상태를 확인
 - 한 사람씩 어제 한 일, 오늘 할 일, 문제점 및 어려운 점 정도만 얘기
 - (그날의 남은 작업량을 소멸 차트에 표시)
- Scrum Master의 역할
 - 팀원들의 개발 작업에 방해되는 요소를 찾아 문제를 해결
 - 개발자 개개인이 수행하고 있는 일의 진행 상황을 확인
 - 소멸 차트에 그날의 남은 작업량을 표시

Sprint 수행: Sprint Board

- Sprint Board (스프린트 현황판)
 - 개발 팀의 개발 현황(진척도, 남은 작업, 진행 속도)을 나타냄

Projects / Beyond Gravity

Board

Epics

TO DO 12

Implement feedback collector
NUC-205 9 ↘ ↗

Bump version for new API for billing
NUC-206 3 =

Add NPS feedback to wallboard
NUC-208 1 ↘ ↗

IN PROGRESS 4

Update T&C copy with v1.9 from the writers guild in all products that have cross country compliance
NUC-213 1 ↗ ↖ ↗ ↗

Tech spike on new stripe integration with paypal
NUC-215 3 ↗ ↖ ↗ ↗

IN REVIEW 4

Multi-dest search UI web
NUC-338 5 ↗ ↖ ↗ ↗

Refactor stripe verification key validator to a single call to avoid

DONE 4

Quick booking for accomodations - web
NUC-336 ✓ ↗ ↖ ↗ ↗

Adapt web app no new payments provider
NUC-346 ✓ ↗ ↖ ↗ ↗

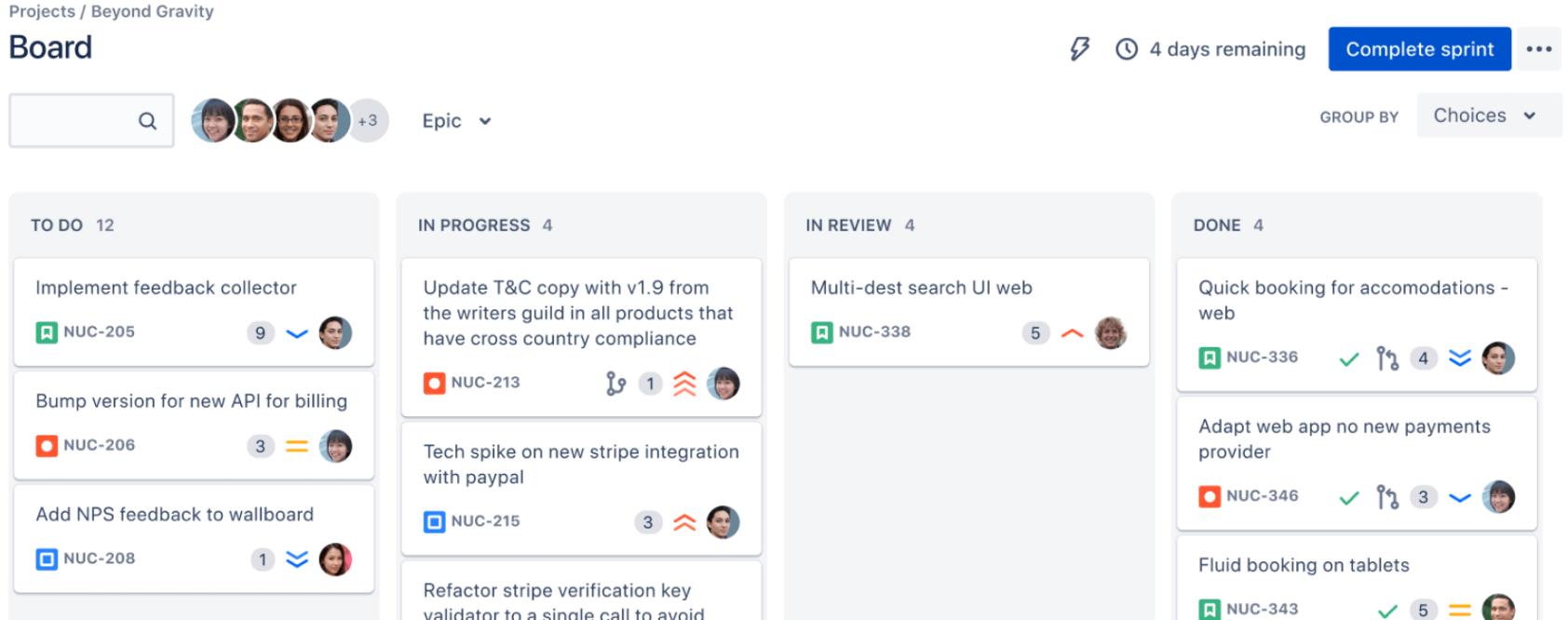
Fluid booking on tablets
NUC-343 ✓ ↗ ↖ ↗ ↗

4 days remaining

Complete sprint

GROUP BY Choices

68



Sprint 수행: Burndown chart (소멸 차트)

- Burndown chart (소멸 차트)
 - 계획 대비 작업이 어떻게 진행되고 있는지를 날짜 별 남은 작업으로 나타냄
 - 개발 후 남은 작업량을 표현하므로 시간이 지남에 따라 차트가 감소
 - 계획 그래프: 처음 계획을 세웠을 때 날짜별로 남은 작업량(점선으로 표시)
 - 실제 그래프: 작업을 수행하면서 날짜별로 실제 남은 작업량(실선으로 표시)

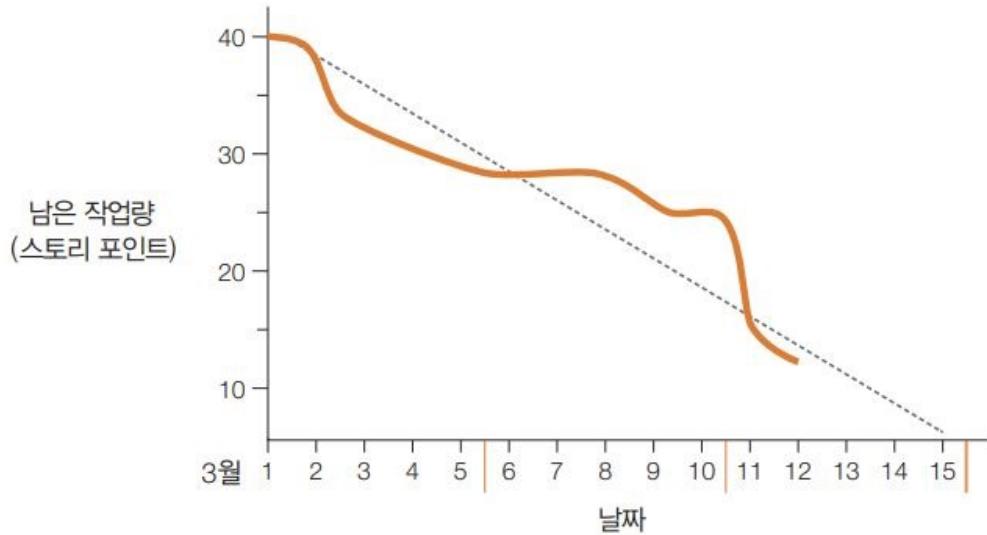


그림 1-33 소멸 차트

Sprint 개발 완료

- 스프린트 개발 완료
 - 처음에 Product owner (제품 책임자) 중심으로 제품 기능 목록을 작성
 - 스프린트 계획 회의에서 스프린트 구현 목록을 작성하고 일일 스크럼 회의를 통해 스프린트를 개발
 - 그 결과 스프린트 개발이 완료되고 최종 제품이 생산
- 최종 제품 (Increment)
 - 모든 스프린트 주기가 끝나면 제품 기능 목록에서 개발하려고 했던 최종 제품이 완성

Scrum 완료 후: Sprint Review Meeting

- Sprint Review Meeting (스프린트 검토 회의)
 - 하나의 스프린트 반복 주기(2~4주)가 끝났을 때 생성되는 실행 가능한 제품을 검토
 - 스프린트 목표를 달성했는지 작업 진행과 결과물을 확인하고, 전체 흐름을 확인해 비즈니스 가치를 점검하는 데 중점
 - 스크럼 팀은 스프린트 동안 작업한 결과를 참석자(고객 포함)들에게 시연하고 요구사항에 얼마나 부합하는지 검토
 - 스크럼 마스터는 스프린트 동안 잘된 점, 아쉬운 점, 개선할 점 등을 찾기 위한 회고를 진행할 수 있음
 - 검토는 가능한 한 4시간 안에 마침

Scrum 완료 후: Sprint Retrospective Meeting

- Sprint Retrospective Meeting (스프린트 회고 미팅)
 - 그동안 스프린트에서 수행한 활동과 개발한 것을 되돌아보고, 개선할 점은 없는지, 팀이 정한 규칙이나 표준을 잘 준수했는지 등을 검토
 - 이때 팀의 단점을 찾기보다는 강점을 찾아 더 극대화하는 데 주안점을 둠
 - 또한 문제점에 대한 해결 방안을 찾는 회의가 아니므로 문제 점을 확인하고 기록하는 정도로만 진행
 - 추정 속도와 실제 속도를 비교해보고, 차이가 크면 그 이유를 분석(프로세스 품질은 측정하지 않음)

Scrum 완료 후: 배포 목록 작성

● 배포 목록 작성

- 배포 사용자에게 시스템 일부를 제공하는 것
- 배포 목록은 제품 기능 목록의 항목 중에서 이번 배포 본에 포함하기로 결정한 것
- 배포 목록을 작성하면 이번 배포 본의 개발 범위와 일정을 수립할 수 있음
- 사용자에게 전달되는 배포 본의 기능 내역과 시기, 스프린트 주기, 배포 일정을 결정하게 됨

표 1-6 배포 목록

기능	내용	작업 일수(MD)	스프린트 주기
수강신청	학생이 수강신청을 할 수 있게 한다.	10MD	스프린트 1
성적등록	교수가 성적을 입력할 수 있게 한다.	7MD	스프린트 2
교과목, 개설과목등록	교수가 교과목과 개설과목을 등록할 수 있게 한다.	18MD	스프린트 3
총 MD		35MD	
총 배포 스프린트			3개
배포 날짜			2021. 10. 31

Scrum 장점

- 반복 주기마다 생산되는 실행 가능한 제품을 통해 사용자와 충분히 의견을 나눌 수 있음
- 일일 회의를 함으로써 팀원들 간에 신속한 협조와 조율이 가능
- 일일 회의 시 직접 자신의 일정을 발표함으로써 업무에 집중할 수 있는 환경이 조성
- 다른 개발 방법론에 비해 단순하고 실천 지향적
- 스크럼 마스터는 개발 팀원들이 목표 달성을 집중할 수 있도록 팀의 문제를 해결
- 프로젝트의 진행 현황을 볼 수 있어 신속하게 목표와 결과 추정이 가능
- 프로젝트의 진행 현황을 볼 수 있어 목표에 맞게 변화를 시도할 수 있음

Scrum 단점

- 반복 주기가 끝날 때마다 실행 가능하거나 테스트할 수 있는 제품을 만들어야 하는데 이 작업이 많아지면 그만큼의 작업 시간이 더 필요
- 일일 스크럼 회의 시간(15분)이 넘어가면 작업시간이 늦어지고 작업하는데 방해 받을 수 있음
- 작업이 얼마나 효율적으로 수행되었는지 알기 어려움
- 프로세스 품질을 평가하지 않기 때문에 품질 관련 활동이 미약하고 따라서 품질의 정도를 알 수 없음

Software Development Process: Agile Process Models DevOps

DevOps

- Development + Operations together
 - A software development methodology which improves the collaboration between developers and operations team using various automation tools.
- Driving factors:
 - Shared responsibility, agile processes, automated tasks, smarter workflows, fearless innovation and continuous feedback
- Benefits
 - shorter development cycles
 - Reduced risks
 - quicker issue resolution
 - better productivity

DevOps Lifecycle

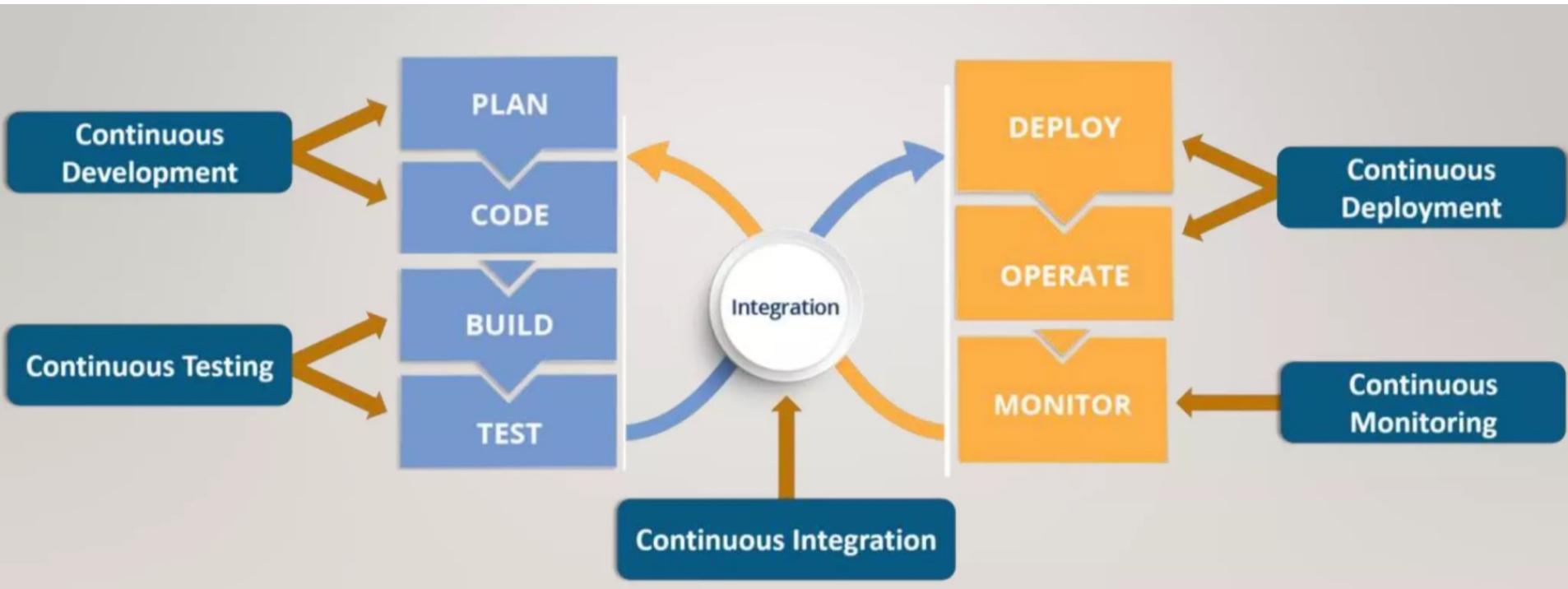
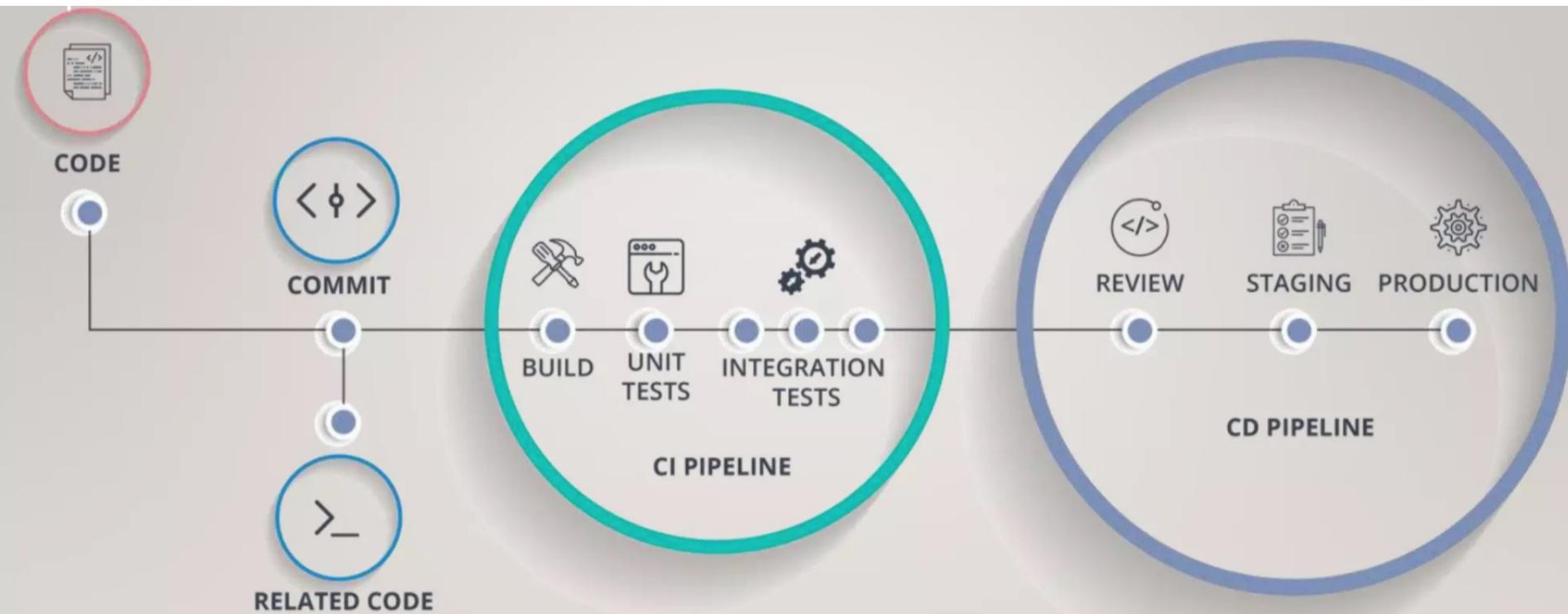


image reference: <https://www.slideshare.net/slideshow/what-is-devops-250717908/250717908>

Automated CI/CD Pipeline

image reference:
<https://www.slideshare.net/slideshow/what-is-devops-250717908/250717908>

- CI/CD (Continuous Integration/Continuous Deployment or Delivery)



Important Phases in a DevOps cycle

- **Continuous Development**
 - Involves committing code to version control tools (e.g., git) for maintaining the different versions of codes and tools (e.g., Ant, Maven, Gradle) for building/packaging the code into an executable file that can be forwarded to the QAs for testing
- **Continuous Integration**
 - The key in automating the whole DevOps process
 - Deals with integrating the different stages of the DevOps lifecycle.

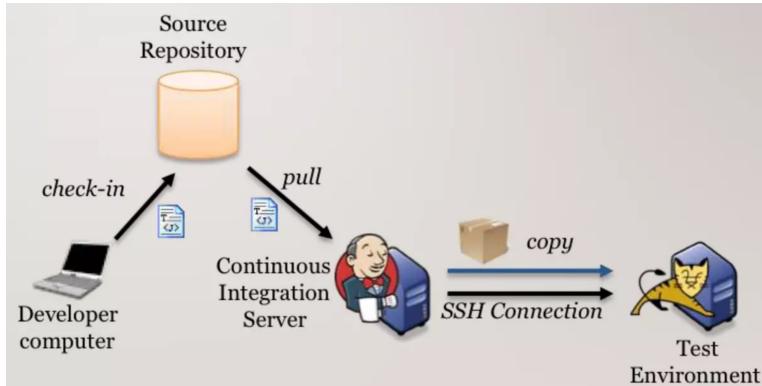


image reference:
<https://www.slideshare.net/slideshow/what-is-devops-250717908/250717908>

Important Phases in a DevOps cycle

- **Continuous Deployment**
 - the code is built the environment or the application is containerized and is pushed on the desired server.
 - Configuration management, virtualization and containerization

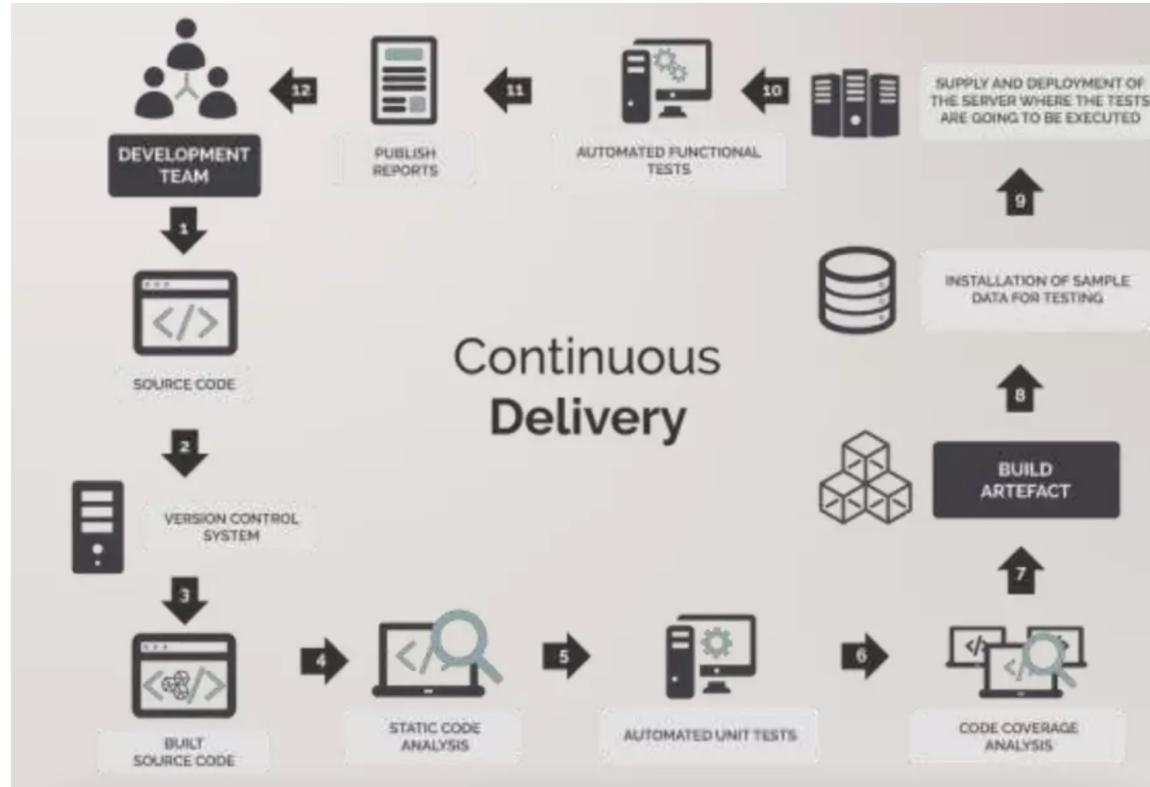
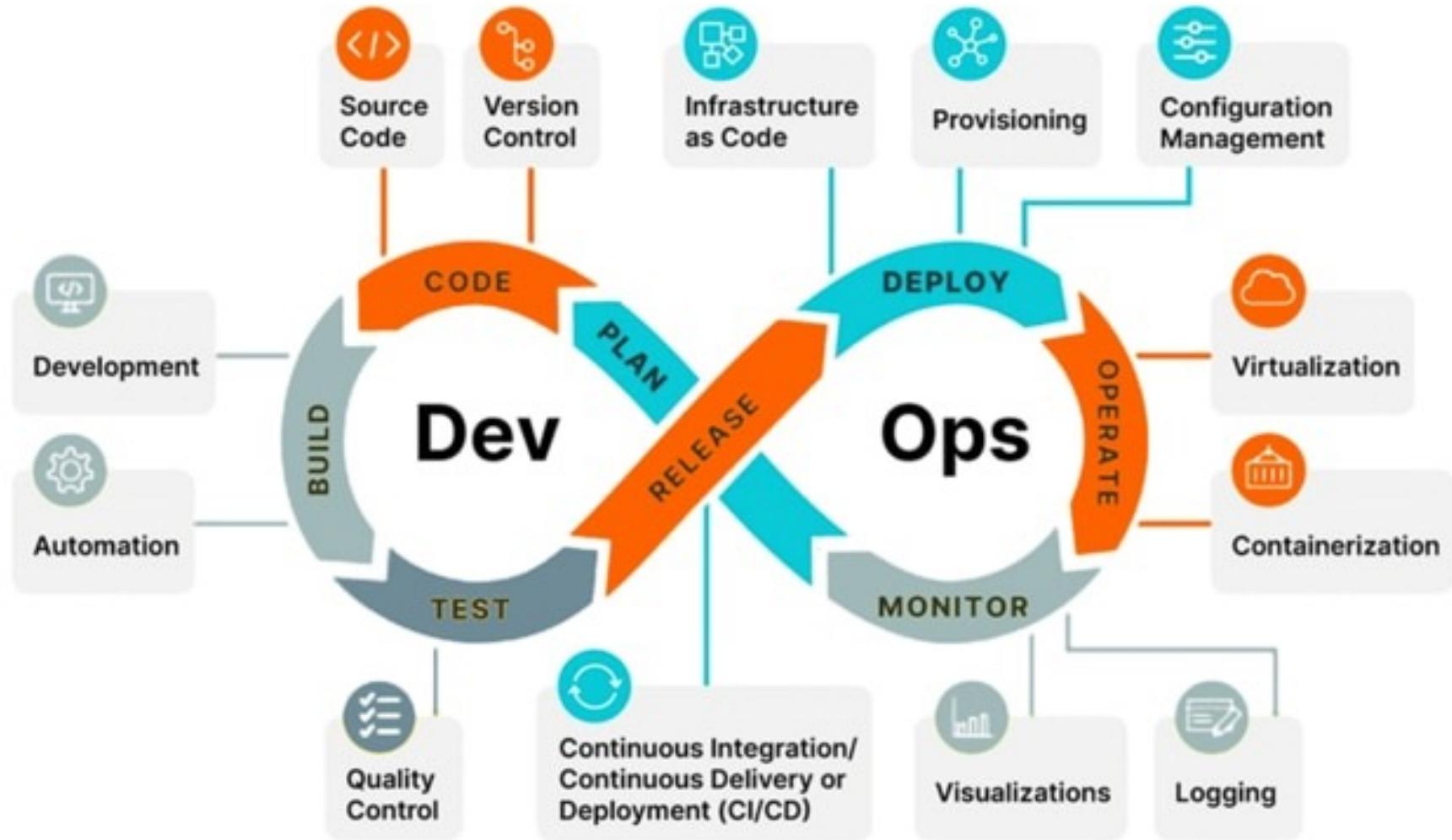


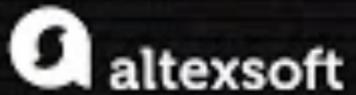
image reference: <https://www.slideshare.net/slideshow/what-is-devops-250717908/250717908>

Important Phases in a DevOps cycle

- **Continuous Testing**
 - Automated testing of the application pushed by developer
 - if an error, the message is sent back to the integration tool, which in turn notifies the developer of the error.
 - If successful, the integration tool pushes the build on the production server
- **Continuous Monitoring**
 - Continuously monitors the deployed application for bugs or crashes
 - Collect user feedback
 - Collected data is sent to the developers to improve the application







SOFTWARE DEVELOPMENT LIFE CYCLE

EXPLAINED



References

- [1] 소프트웨어 공학의 모든 것, 최은만, 생능출판사
- [2] 쉽게 배우는 소프트웨어 공학, 김치수, 한빛아카데미
- [3] 소프트웨어 공학 이론과 실제, 홍장의, 한빛아카데미
- [4] scrum.org
- [5] <https://www.slideshare.net/slideshow/what-is-devops-250717908/250717908>
- [6] https://youtu.be/SaCYkPD4_K0?si=WHJAIN9qRZfDUWkO