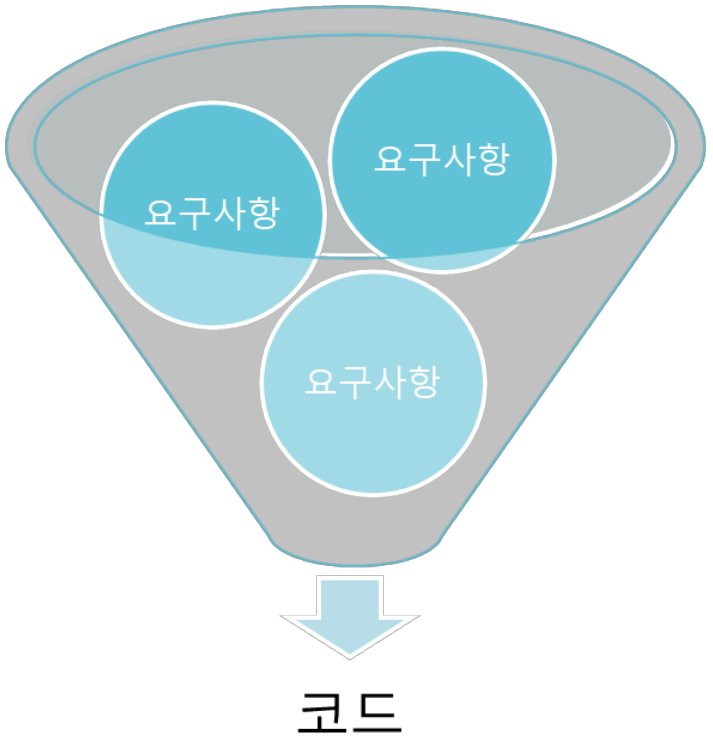
1장. SW개발프로세스

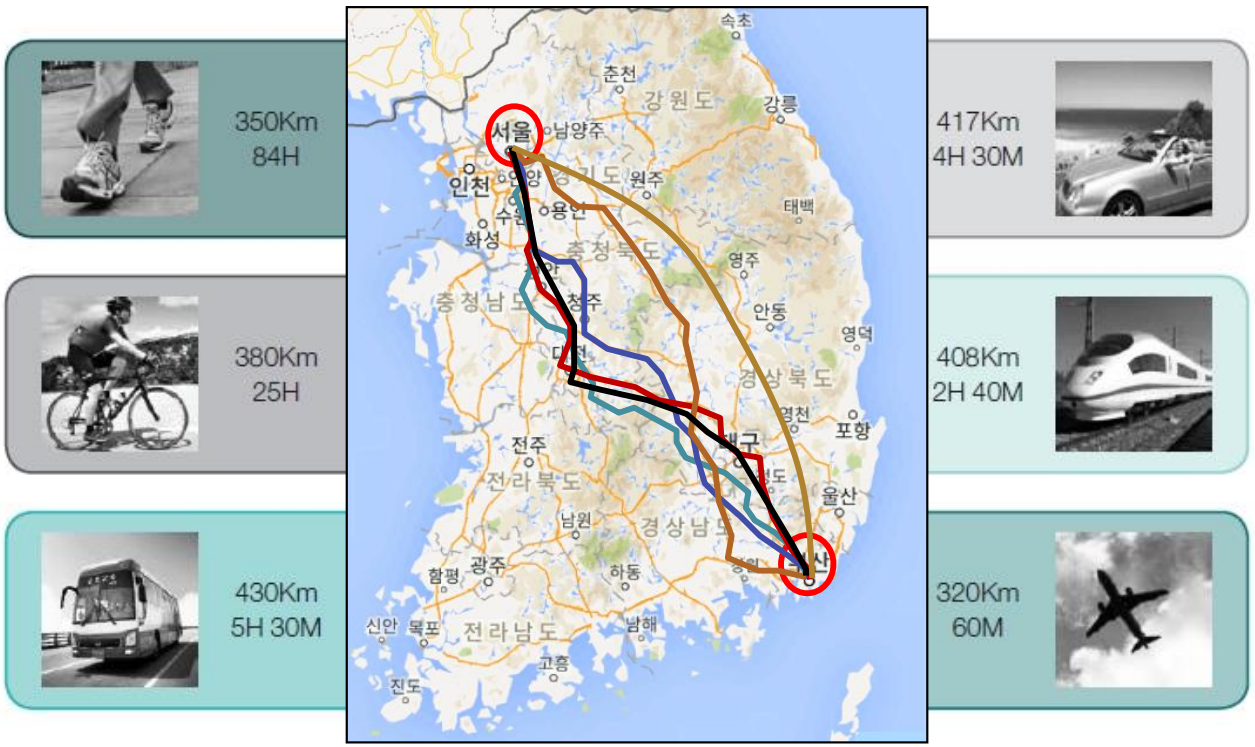
숙명여자대학교
SOOKMYUNG WOMEN'S UNIVERSITY

목차



여행을 떠난다면?

- 서울에서 부산까지 어떻게 갈수 있을까?



비용 / 시간고려

목표는 같지만 개발 조직의 상황에 따라 과정이 다를수 있음

⇓

문제해결을 쉽게하기 위한 프로세스필요

1. SW 개발 프로세스 (1/3)

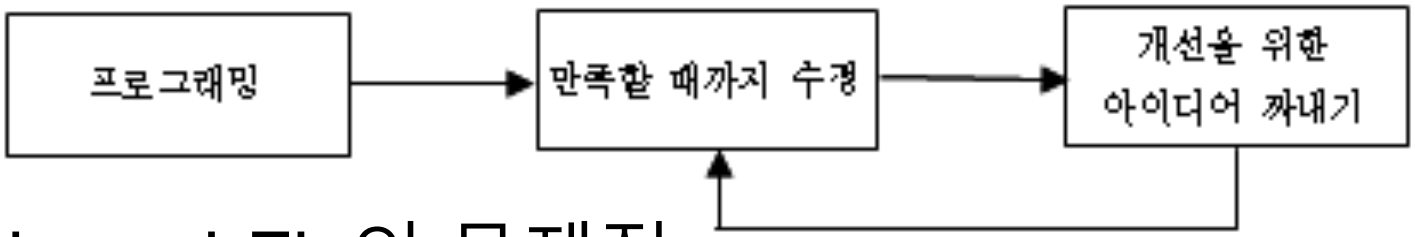
- 소프트웨어 프로젝트
  - 수행할 작업을 조직화한 프로세스를 이용
  - 비용, 일정, 품질에 대한 목표를 성취하는 것
- 프로세스의 정의(사전적)
  - 어떤 일을 하기 위한 특별한 방법으로써 일반적으로 단계나 작업으로 구성
- 소프트웨어 개발 프로세스
  - 소프트웨어를 개발하는 과정, 즉 작업순서
    - 순서제약이 있는 작업의 집합
    - 높은 품질과 생산성이 목표

1. SW 개발 프로세스 (2/3)

↗ HW개발이주력 : SW와개발자 일두한정적

↗ 1970년대

- 프로세스가 없는 개발 = Code-and-Fix(즉흥적 개발)



- Code-and-Fix의 문제점
  - 요구나 설계 작업의 중요성을 깨닫지 못함
    - 즉흥적인 방법으로는 사용자의 높은 요구 수준에 도달하기 어려움
    - 계속 고치는 작업이 필요
  - Code-and-fix로는 좋은 소프트웨어 구조를 만들 수 없음

1. SW 개발 프로세스 (3/3)

- 프로세스가 없는 개발은 계획이 없기 때문에 작업의 목표가 없음
  - 일을 한 후에도 잘한 것인지 못한 것인지 판단할 수가 없음 → 개선의 밑바탕
  - 비용과 일정의 조절을 할 수가 없음 → 작업량.비용의 예측을 위함
  - 체계적인 테스트 작업이나 품질 보증 차원의 활동에 대한 인식이 없음 ⇒ 발견되지 않은 결함이 남아 계속 고치게 되므로, 시스템이 더욱 악화

2. 전통적인 프로세스 모델

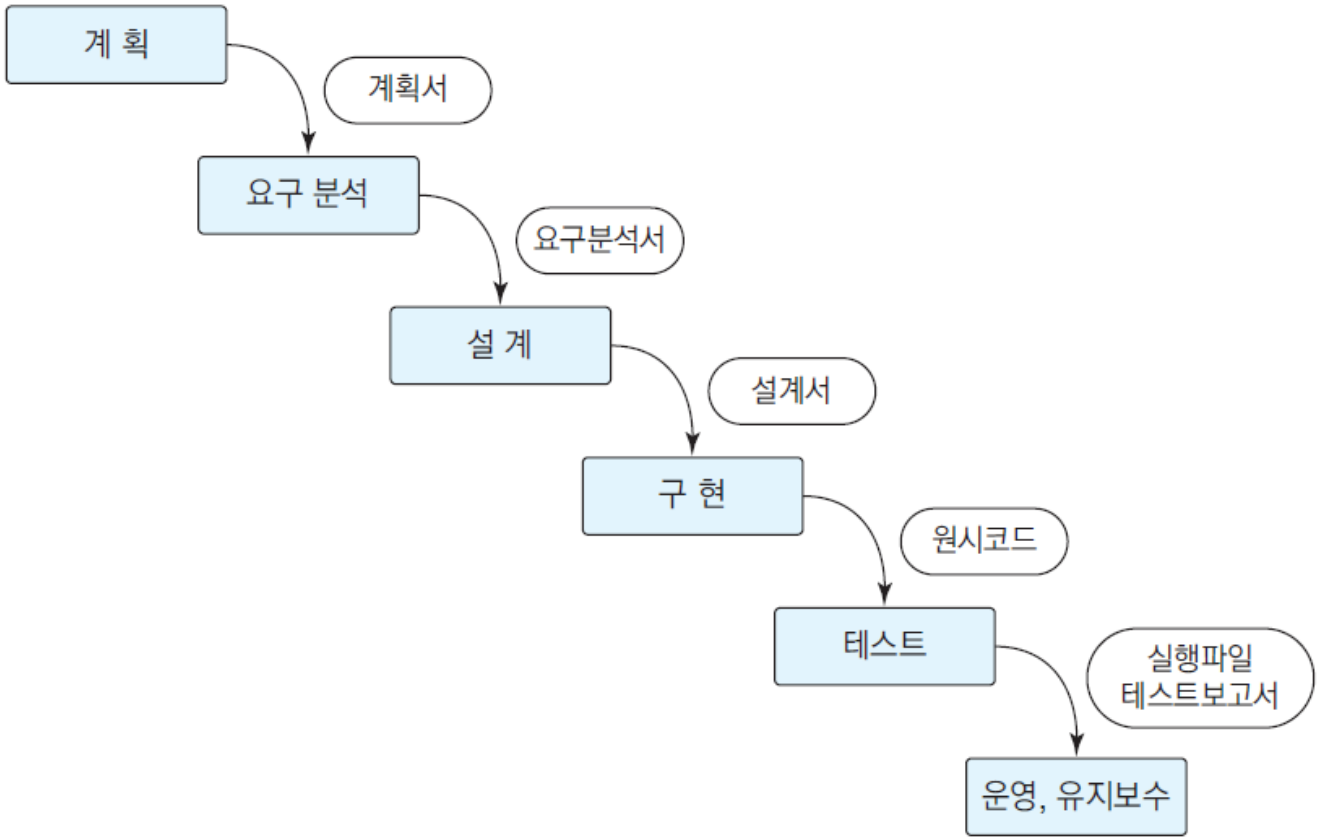
- SDLC(SW Development Life Cycle)
  - 소프트웨어(정보 시스템)를 개발하는 절차, 혹은 개발 단계의 반복하는 과정

같은의미로 생각

- 전통적인 프로세스 모델 (가이드라인) 애자일모델?
  - 폭포수 모델 +) 애자일모델
  - 프로토타이핑 모델
  - 나선형 모델
  - 진화적 모델
  - 점증적 모델
  - V 모델

2.1 폭포수(waterfall) 모델 (1/3)

- Waterfall Model



2.1 폭포수(waterfall) 모델 (2/3)

- Waterfall Model의 특징
  - 1970년대 소규모 프로그램에 적용하는 Code-and-Fix model의 문제를 해결하기 위해 정의 → 처음으로 체계를 잡은것!
  - 각 단계가 완료된 후 다음 단계로 진행, 이전단계로 돌아갈 수 없음
    - 순차적 : 각 단계 사이에 중복이나 상호작용이 없음
    - 각 단계의 결과는 다음 단계가 시작 되기 전에 점검 → 회의 多
  - 결과물 정의가 중요 : 다음단계로 넘어갔을때 이전단계의 결과물 보고 진행 결과물 관리를 잘 해야함!

가장 오래되고 폭넓게 사용 → 사례↑

2.1 폭포수(waterfall) 모델 (3/3)

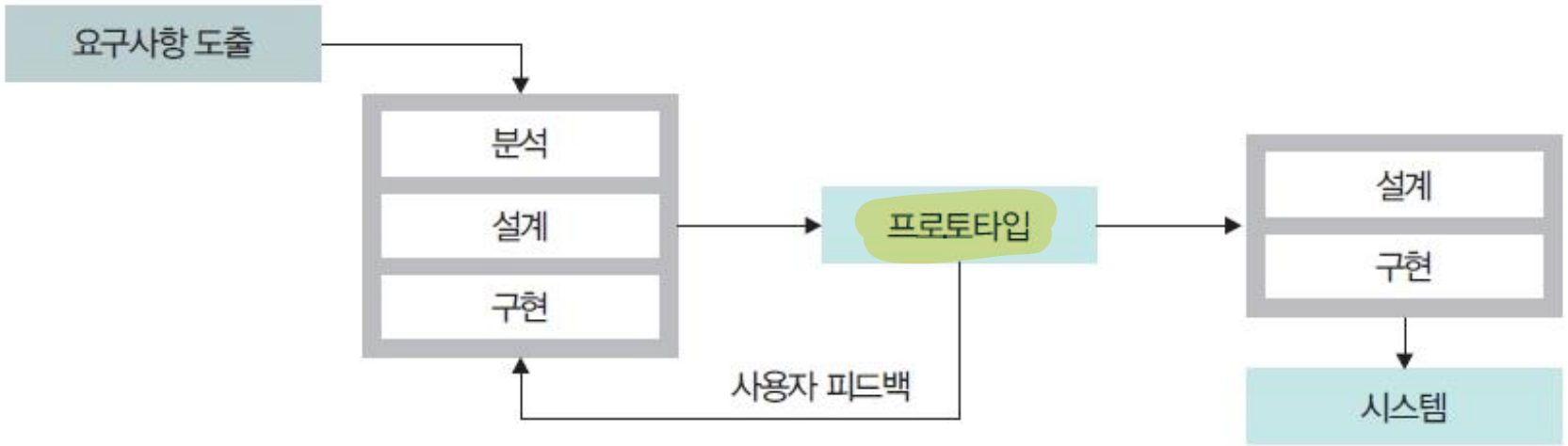
- 프로젝트 진행과정을 세분화하여 관리하기에 용이
- 목표시스템이 과정의 후반부에 가서야 구체화되므로 중요한 문제점이 나중에 발견되기도 함 ⇒ 중간피드백이 없음 → 후반의 문제점이 누적됨
- 적용시 고려사항
  - 관리가 상대적으로 쉬우나 요구 사항의 변경에 대한 대응력이 떨어짐 유연성↓ ⇒ 애자일프로세스 도입
  - 기술 위험이 낮고 유사한 프로젝트 경험이 있는 경우 적용
  - 요구사항이 비교적 명확히 정의되어 있는 경우 적용

신뢰성이 높이 요구되는 분야 X

2.2 프로토타이핑 모델 (1/3)

prototype : 시제품

- Prototyping Model의 정의
  - 사용자의 요구사항을 충분히 분석할 목적으로 시스템의 일부분을 일시적으로 간략히 구현한 다음 다시 요구사항을 반영하는 과정을 반복하는 개발모델



프로토타입 모델에 의한 소프트웨어 개발 프로세스

2.2 프로토타이핑 모델 (2/3)

인터페이스위주. 상호작용을 보여주기위함

- 프로토타이핑 모델의 목적
  - 요구 분석의 어려움 해결 ⇒ 사용자의 참여 유도
  - 요구 사항 도출과 이해에 있어 사용자와의 커뮤니케이션 수단으로 활용 가능 (의사 소통 도구)
  - 사용자 자신이 원하는 것이 무엇인지 구체적으로 잘 모르는 경우, 간단한 시제품으로 개발
  - 개발 타당성으로 검토 시장성확인등
  - 요구사항을 검증하기 위해 프로토타입 개발 → 폭포수 모델에 따라 전체 시스템을 개발 : 폭포수 모델의 단점을 보완

2.2 프로토타이핑 모델 (3/3)

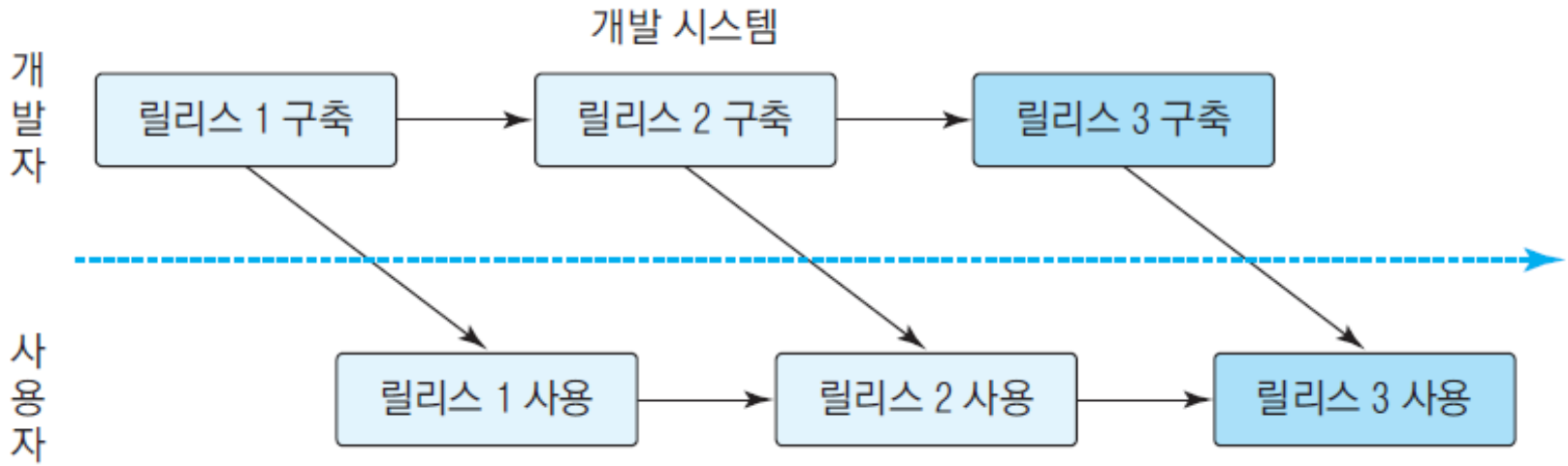
- 장단점

장점	단점
• 프로토타입을 통해 요구 사항 도출 용이 • 실제 모습을 확인할 수 있기 때문에 시스템의 이해와 품질 향상 • 개발자와 사용자 간 의사소통 용이	• 프로토타입 결과를 최종 결과물로 오해할 수 있음 • 잘못된 프로토타입 개발 시 폐기 처분에 따른 경제적인 손실 발생 가능 +프로토타입을 반복적으로 수정 • 프로토타입에 대한 산출물 관리가 어려움 버전이 여러개

→ 어디까지가 실제 제품에 대한 설계와 구현인지?

2.3 진화적 모델 (1/2)

- 진화적(evolutionary) 개발
  - 빠른 시간 안에 시장에 출시해야 이윤에 직결 ⇒ 시스템을 나누어 릴리스, 릴리스 할 때마다 기능의 완성도를 높임
  - 개발된 모듈을 업그레이드 하는 방식으로 SW가 진화해 가는 개념

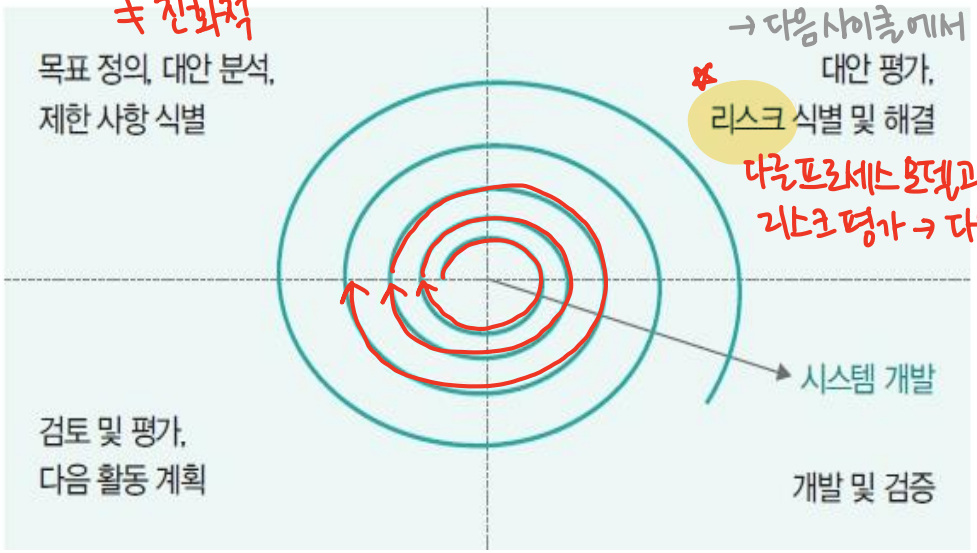


2.3 진화적 모델 (2/2)

- 장점
  - 사용자의 요구를 빠르게 반영
  - 새로운 기능을 가진 SW에 대한 시장을 빨리 형성
  - 가동 중인 시스템에서 일어나는 예상하지 못했던 문제를 신속하고 꾸준하게 고쳐 나갈 수 있음
- 단점
  - 프로젝트 관리가 복잡
  - 반복적인 프로세스로 끝이 안보일 수 있어 실패의 위험이 커짐

2.4 나선형(spiral) 모델 (1/2)

- 폭포수(Waterfall) 모델과 프로토타입(Prototype) 모델의 장점을 활용
  - 복잡해지고 있는 소프트웨어 개발 환경에 위험 요소를 분석하고 해결할 수 있도록 지원하는 모델
  - 여러 번의 점증적인 릴리즈 : 한 사이클에서 개발한 부분이 적어짐
    여러 개라면 한 사이클에서 중요한 게
    → 다음 사이클에서 그 다음 중요한 거 그게 → …



2.4 나선형(spiral) 모델 (2/2)

- 장점
  - 대규모 시스템 개발에 적합
  - 반복적인 개발 및 테스트로 강인성(robustness) 향상
  - 한 사이클에 추가 못한 기능은 다음 단계에 추가
- 단점
  - 관리 및 위험 분석이 중요하므로 매우 어렵고 개발기간이 장기화될 가능성
- 적용
  - 재정적 또는 기술적으로 위험 부담이 큰 경우
  - 요구 사항이나 아키텍처 이해에 어려운 경우

↳ 상대적으로 중요한 부분!
↳ 앞선 사이클에서 만든 부분의 테스트가 반복됨 (누적형)

2.5 점증적 모델 (1/2)

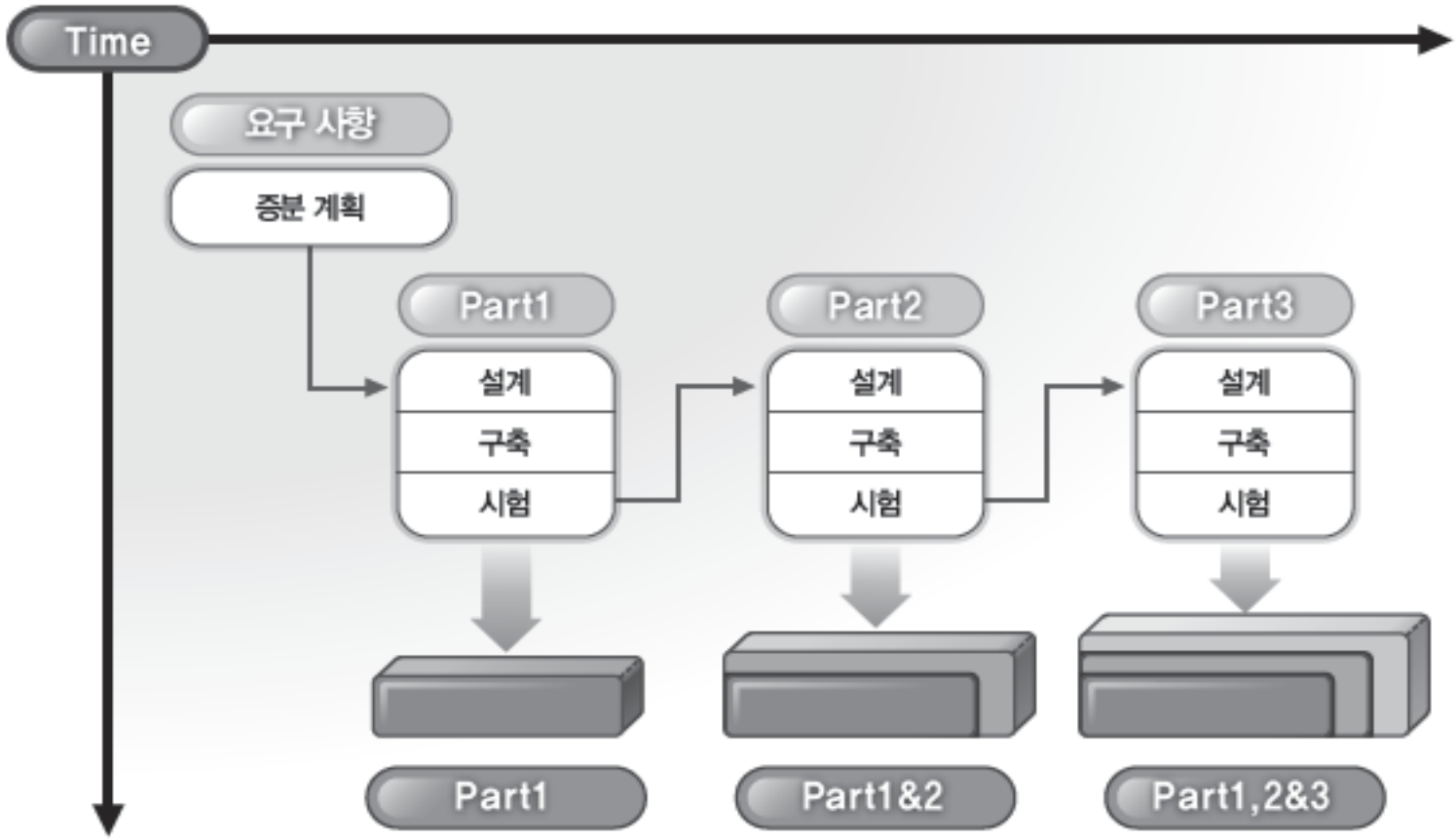
(나선형 모델을 포함함)

- 점증적 개발 (Incremental) 모델
  - 사용자 요구사항의 일부분, 제품의 일부분을 반복적으로 개발하면서 대상 범위를 확대해 나아가서 최종제품을 완성
- 폭포수 모델의 변형으로 증분을 따로 개발 후 통합
  - 프로토타이핑과 같이 반복적이나, 각 증분마다 실행이 가능한 완성 모듈을 인도
  - 규모가 큰 개발 조직일 경우 자원을 증분 개발에 충분히 할당할 수 있어 병행 개발 가능

2.5 점증적 모델 (2/2)

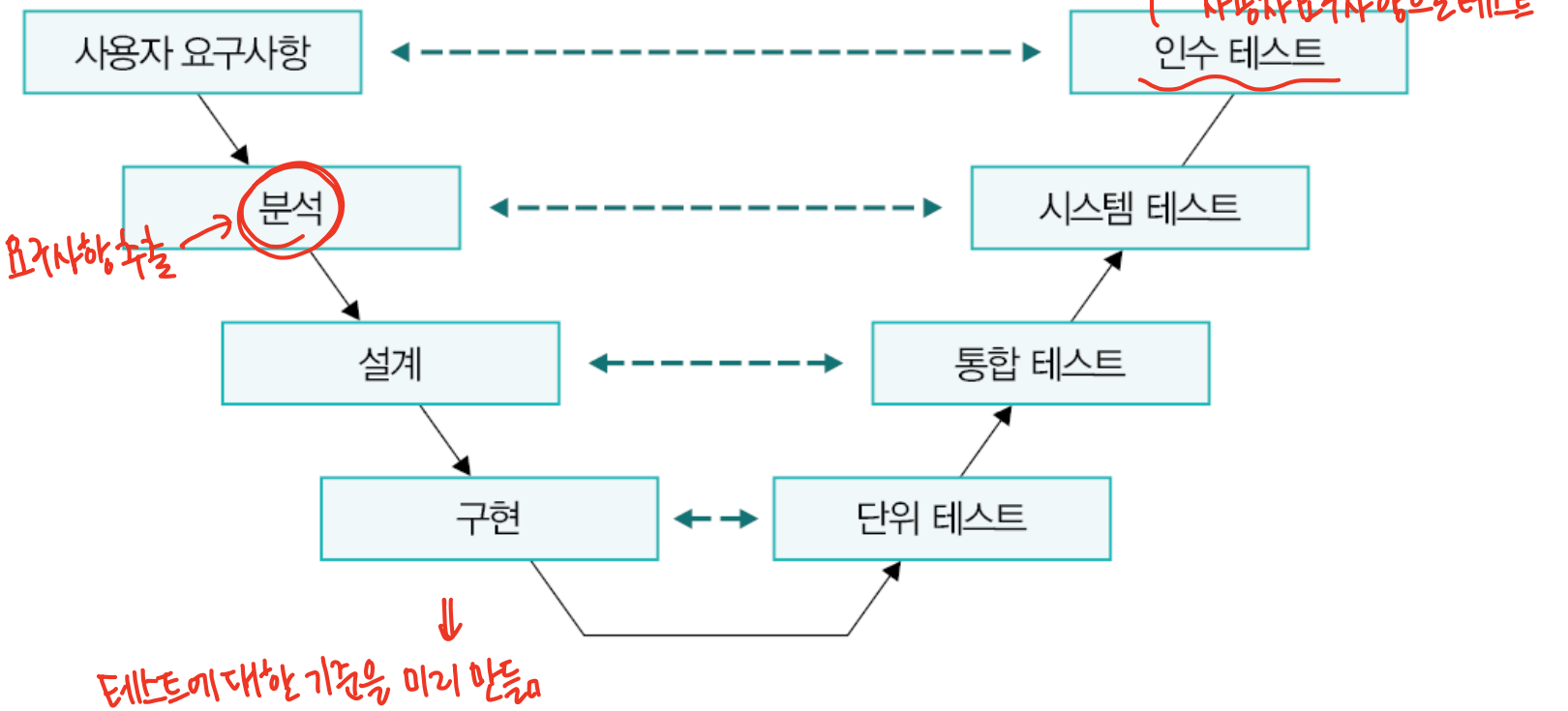
- 릴리즈 과정

*증분에 대한 계획 필수
→ 언제 끝날지 알수 없음
*반복적인 테스트 → 안정성↑



2.6 V 모델 (1/2)

- 폭포수 모델을 확장한 Verification & Validation 모델
  - 감추어진 반복과 재 작업을 드러냄
  - 작업과 결과의 검증에 초점



2.6 V 모델 (2/2)

테스트를 매우 많이 하는 모델
→ 잠재된 오류를 많이 찾음
→ 오류가 치명적인 SW에 사용
(ex. 국방, 의료, …)

- 장점
  - 소프트웨어 개발 결과물에 대한 단계적인 검증과 확인 과정을 거침으로써, 오류를 줄일 수 있음
- 단점
  - 폭포수 모델을 적용함으로써 개발 활동에 반복이 없어 변경을 다루기가 쉽지 않음
- 적용
  - 요구사항이 정확히 이해되는 작은 규모 개발 시 유용
  - 신뢰성이 높이 요구되는 분야

9/11

전통적
→ Rapid Application Development : 코드위주, 문서지향X
↓
(RAD, 애자일프로세스)
→ 애자일

3. 애자일 방법론

VS 전통적 개발프로세스모델

- 애자일(Agile) 방법론 ↗ 사용자의 요구를 빠르고 정확하게 캐치
  - 좋은 것을 빠르고 낭비 없게 만드는 개발을 가능하게 해주는 방법론 전체 (able to move quickly and easily)
  - 대표적인 애자일 방법론들
    - 적응형 소프트웨어 개발
    - 크리스털 패밀리(Crystal Family) : 프로젝트유형으로 level 정의, level마다 정해둠 (ex. red, orange, …)
    - 익스트림 프로그래밍(XP)
    - 스크럼(Scrum)
    - 린 소프트웨어 개발(Lean Software Development) 도요타 (생산라인)
    - 기능 주도 개발(Feature-Driven Development)

(사용자가 말하는 기능)
* function (개발자가 말하는 기능)

3.1 애자일 적용 원리 (1/2)

- 신속한 소프트웨어 개발을 위한 원칙
  - 고객 만족을 가능한 한 빨리 이끌어내고, 유연한 소프트웨어를 지속적으로 사용자에게 제공
  - 개발 과정에서 늦은 시점일지라도 요구사항 변경을 적극 수용
  - 실행 가능한 소프트웨어를 1~2주일에 한 번씩 사용자에게 배포 → 개발주기가 짧음!
  - 사용자와 개발자 간 친밀한 미팅을 거의 매일 수행
  - 프로젝트는 신뢰할 수 있고 동기가 부여된 개인을 중심으로 진행
  - 의사소통을 위해 가능하면 한 곳에 모여 대화

3.1 애자일 적용 원리 (2/2)

≒ code oriented

- 실행 가능한 소프트웨어를 기준으로 프로젝트 진척률을 결정
- 일정한 개발 속도를 유지하며, 지속 가능하도록 소프트웨어를 개발
- 우수한 기술의 도입과 좋은 설계에 지속적인 관심 갖기
- 단순화는 필수
- 최상의 아키텍처, 정제된 요구사항, 좋은 설계는 잘 구성된 팀을 통해 개발이 가능
- 팀은 더 효과적인 방법의 적용, 적절한 조정 과정을 정기적으로 수행

3.2 XP프로세스 (1/2)

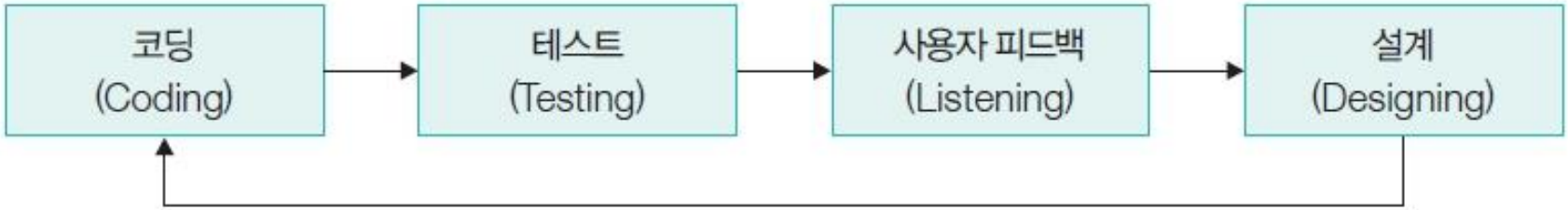
- XP(eXtream Programming) 반복 개발 방법의 실천사항(Practices)
  - 증분계획(incremental planning) → 중요한것먼저 개발하도록
  - 작은 배포(small release)
  - 단순 설계(simple design)
  - 테스트 우선 개발(test-first development)
  - 재구조화(refactoring) 꾸준히! 실행단위를 확장/변경이 쉽도록 구조수정(결과, 기능은 같음)
  - 페어프로그래밍(pair programming) 개발자들이 짝을지어서 → 코드품질↑
  - 공동소유권(collective ownership)
  - 지속적 통합(continuous integration) 오류없이 돌아가도록
  - 현장 고객(on-site customer)

설계패턴

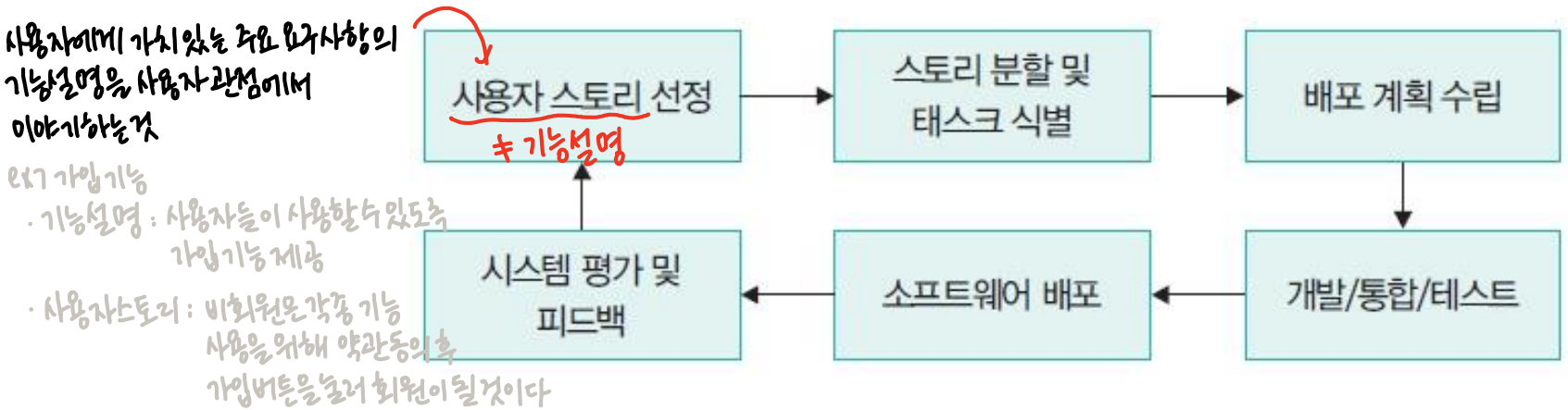
→ 개발주기가 짧음

3.2 XP프로세스 (2/2)

- 개발 사이클
  - 엔지니어링 사이클



  - 배포 사이클



사용자에게 가시성있는 주요 요구사항의 기능설명을 사용자 관점에서 이야기하는것

RE) 가입기능
· 기능설명 : 사용자들이 사용할수있도록 가입기능 제공
· 사용자스토리 : 비회원은 각종 기능 사용을 위해 약관동의후 가입버튼을 눌러 회원이 될것이다

'~는 ~하게위해 ~할것이다'
각스토리마다의 구현난도로 스토리포인트 구현

3.3 스크럼 프로세스 (1/3)

- 스크럼(Scrum) : 반복적 개발의 관리 관점을 강조
- 스크럼의 용어
  - 스프린트(sprint): 반복적인 개발 주기 1~4주
  - 제품 백로그(backlog): 개발할 제품에 대한 요구사항 목록 (사용자스토리목록)
  - 스프린트 백로그: 스프린트 목표에 도달하기 위해 필요한 작업 목록
  - 제품 증분
  - 번다운 차트, 번업 차트 다음페이지
  - 스크럼 마스터: 프로젝트 관리자

스크럼팀 참여자는 모두동등한 위치

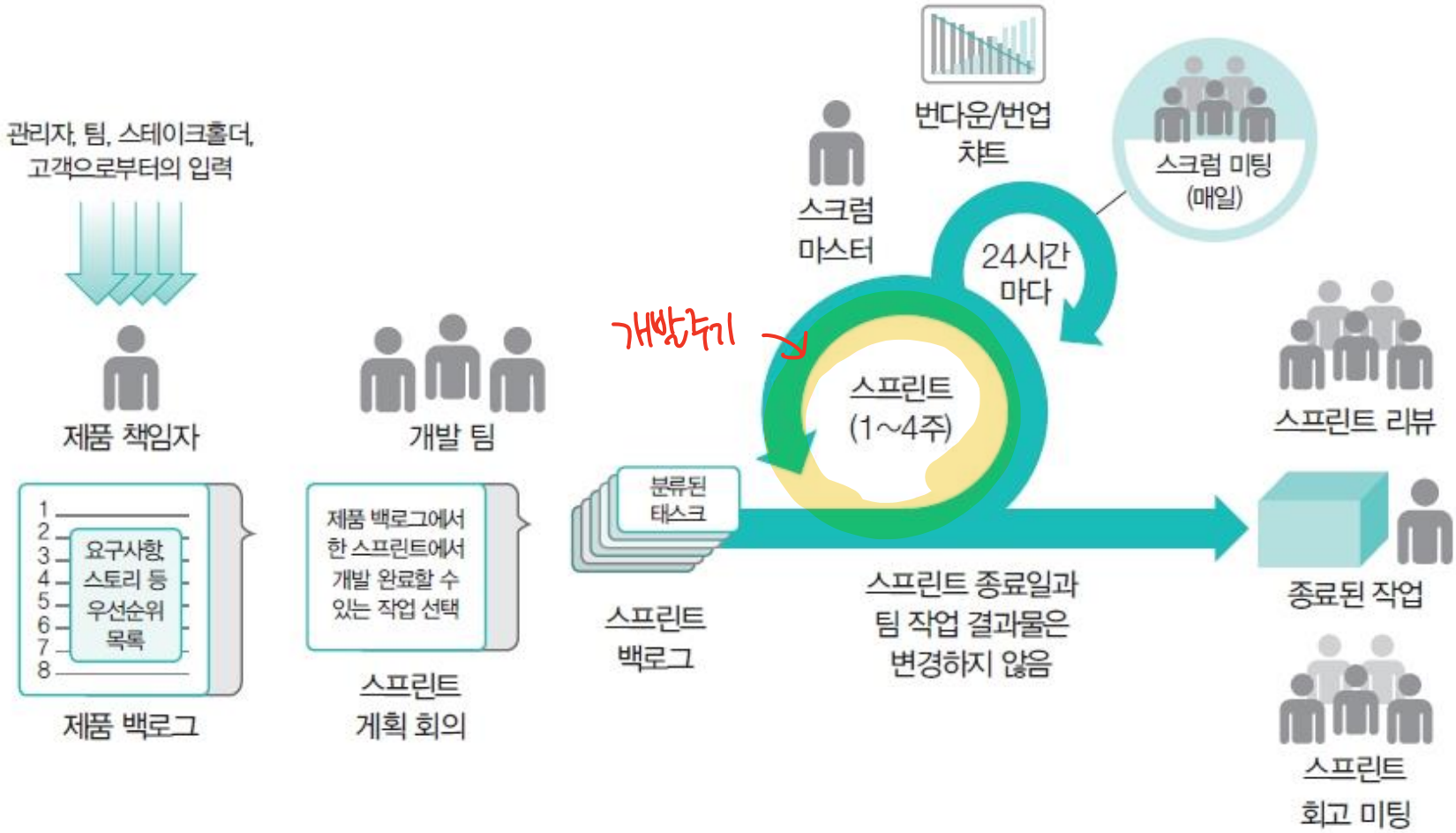

남아있는것에 대한것
Release Burndown
3/7/2017 - 12/22/2017
Story Points Remaining 164
Completed 97%
Average burndown 32
Items not estimated 6
Total Scope Increase 26
Projected completion: 1/10/2018 (1 more iteration)
500
400
300
200
100
0
3/7/2017
Sprint 115
Sprint 116
Sprint 117
Sprint 118
Sprint 119
Sprint 120
Sprint 121
Sprint 122
Sprint 123
Sprint 124
Remaining
Burndown
Total Scope
진척에관한것
Burnup
8/1/2022 - 8/31/2022
Stories Remaining 50
Completed 47%
Average burnup 2
Total Scope Increase 28
40
30
20
10
0
8/1/2022
8/3/2022
8/5/2022
8/7/2022
8/9/2022
8/11/2022
8/13/2022
8/15/2022
8/17/2022
8/19/2022
8/21/2022
8/23/2022
8/25/2022
8/27/2022
8/29/2022
8/31/2022
Bug
User Story

3.3 스크럼 프로세스 (2/3)

- 스크럼은 특정 언어나 방법론에 의존적이지 않은, 넓은 응용 범위의 개발 기법
- 스크럼 특성
  - 개발 할 기능 및 개선점에 대한 우선순위 부여
  - 개발 주기는 30일 정도
    - 개발 주기마다 실제 동작할 수 있는 결과를 제공
    - 개발 주기마다 적용할 기능이나 개선 목록을 제공
  - 날마다 15분 정도 회의
  - 원활한 의사소통을 할 수 있도록, 구분 없는 열린 공간을 유지

3.3 스크럼 프로세스 (3/3)

- 스크럼 프로세스



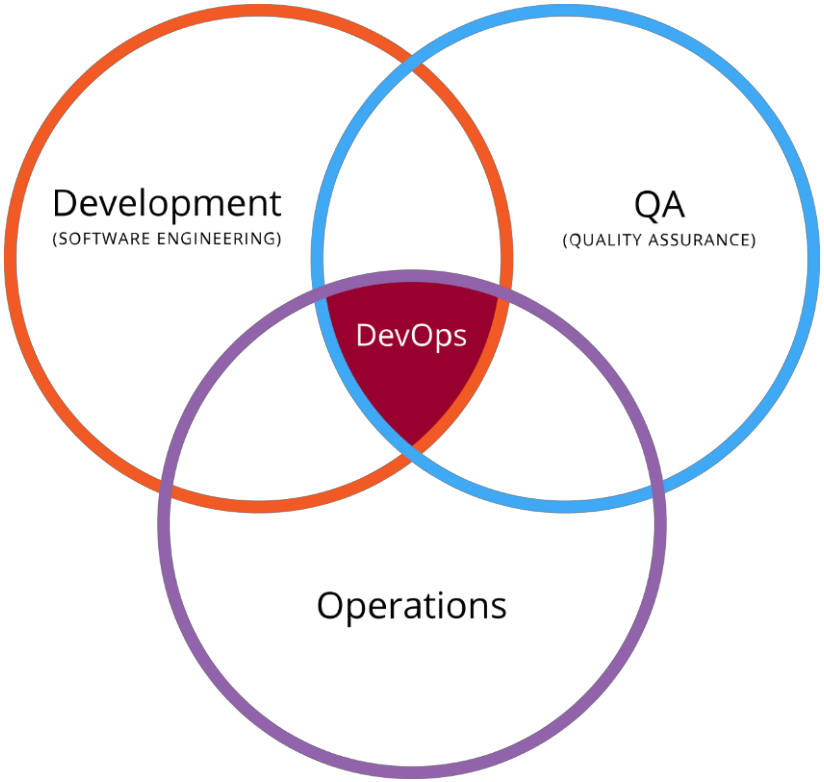
4. Chaos와 DevOps (1/10)

- Chaos Engineering netflix 논문
  - 다양한 서비스 운영 환경에 대응하고 새로운 기능을 중단 없이 제공하기 위해서, 기존 서비스에 임의의 결함을 주입하고 이들이 어떠한 행동을 보이는지 동적으로 검사하면서, 시스템 운영에 대응하는 방법을 찾는 것
  - 예측 불가능한 상황에서 나타나는 시스템 동작의 규칙성을 찾아내고 이에 대응하는 활동

사용자가 많이 사용하지 않는 시간대에
chaos 주입 후 상황 기록, 해결 (test)
→ 나중에 실제로 일어났을때 해결에 참고

4. Chaos와 DevOps (2/10)

- DevOps = Development + Operations 합성어
- 소프트웨어 개발자와 정보기술 전문가 간의 소통, 협업 및 통합을 강조하는 개발 환경이나 문화



4. Chaos와 DevOps (3/10)

- DevOps의 정의
  - IBM : 소프트웨어 개발 팀과 운영 팀 간의 조정을 위하여 제시된 프로세스
  - IEEE 표준 : 소프트웨어 개발과 관련된 스테이크홀더들이 소프트웨어 시스템 또는 서비스를 명세·개발·운영하고 시스템 전체 수명주기에 걸쳐 지속적인 개선을 도모하기 위하여 의사소통과 협업을 증진하기 위한 원리와 실천 사항들의 모임

4. Chaos와 DevOps (4/10)

- DevOps 등장 배경
  - 사일로기반 개발방식의 문제 : 소프트웨어 개발의 각 단계가 종료된 후 다음 단계로 진행되므로, 각 단계 간 상호작용 부재

부서이기주의

사일로 개발 접근 방법의 개념도(출처: syfte blog, 2016)

- DevOps
  - 서비스를 적시에 제공할 수 있는 린(Lean) 및 애자일(Agile) 원칙에 기반한 소프트웨어 개발 접근 방식

4. Chaos와 DevOps (5/10)

- DevOps의 핵심 원리

개발과 운영의

자동화 : 운영 환경의 오류를 신속하고 일관성 있게 해결하기 위하여 자동화된 환경을 전체 수명주기에 구축

- 반복 : 유지보수라는 기존 개념보다는 운영 과정에서 모니터링된 요구사항을 시스템에 반영해 개발하는 반복 개발 형식으로 진행 사용자가 요청하기 전에 모니터링해서 미리!
- 자기 서비스 : 협업 및 자동화 환경 구축을 통해 개발자와 운영자가 서로 방해하지 않고 독립적으로 일할 수 있도록 지원 자기가 필요한걸 스스로 구축

4. Chaos와 DevOps (6/10)

- DevOps의 핵심 원리 (계속)
  - 지속적 개선 : 사용자의 피드백을 소프트웨어 및 운영 환경을 개선하기 위해 활용하는 것
  - 협업 : 개발 팀과 운영 팀 간의 성공적이고 지속적인 협업 능력이 요구
  - 지속적 테스트 : 개발 혹은 수정된 코드에 대해 단위테스트 후 서버에 전달된 코드는 통합테스트를 수행 => 품질보증팀, 운영팀 등이 성능테스트
  - 지속적 통합과 지속적 배포

개발 : 만들면서
유지보수

애자일리티! 기본
아키텍처자체도작게 (마이크로서비스아키텍처)
↳ 충돌, 영향↓

4. Chaos와 DevOps (7/10)

- DevOps 적용 동향
  - 고객 서비스를 365일 24시간 내내 제공해야 하는IT 조직에서 도입하고 있는 소프트웨어 시스템의 개발 및 운영 전략
  - 아마존, 구글, 넷플릭스 등

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

4. Chaos와 DevOps (8/10)

- DevOps 프로세스
  - 소프트웨어 요구사항을 작은 단위의 개발 태스크로 분할
  - 각 태스크에 대한 개발 및 배포 계획을 수립(Plan)한 후, 이 계획에 따라 단위 소프트웨어 컴포넌트를 개발(Code)
  - 개발된 코드들은 상호 통합되어 배포 가능한 기능으로 구성(Build)
  - 이는 테스트(Test) 과정을 거쳐 Ops로 릴리즈(Release)

4. Chaos와 DevOps (9/10)

- DevOps 프로세스 실현을 위해서는 자동화는 필수
  - DevOps의 주요목표 중 하나는 애플리케이션 배포 및 운영환경과 개발 및 테스트 환경 간의 격차를 줄이는 것 ⇒ 애플리케이션 배포시점에서 발생할 수 있는 환경 불일치 문제를 완화
  - 이를 위해 여러가지 도구를 사용해서 개발을 진행
  - 사용되는 도구 모임을 'DevOps Toolchain'이라고 함

4. Chaos와 DevOps (10/10)

- DevOps Toolchain

단계	단계별 활용 가능한 자동화 도구
계획	Jira, Trello, Asana, Confluence, Azure etc.
코딩	Git, Github, Bitbucket, eclipse IDE, Artfactory etc.
빌드	Maven, Gradle, Jenkins, TravisCI, Apache ANT etc.
테스트	Junit, nunit, Jmeter, Selenium etc.
릴리즈	Red Hat, Puppet, openstack, Chef, Release, Spinnaker, Bamboo, apigee, MS Azure, AWS, JFrog etc.
설치	docker, Virtualbox, Automtic, uDeploy, Jenkins, TFS, Kubertenes, Elasticboc, bmchelix, LXD etc.
운영	Puppet, Chef, Fortify, Veracode, Flyway, MongoDB, PowerShell, Ravello etc.
모니터링 & 피드백	dynatrace, APPDYNAMICS, elastic, splunk, New Relic, sensu etc.

5. 소프트웨어 프로세스 개선 (1/3)

- 소프트웨어 제품의 품질은 소프트웨어 제품을 개발하고 유지보수 하는데 사용하는 프로세스의 품질에 의해 결정된다 [Humphrey, 1995]

- 소프트웨어 개발의 문제점 : 낮은 생산성/품질 저하
  - 소프트웨어 프로젝트의 납기지연 및 비용초과
  - DoD 소프트웨어 프로젝트 중 품질/납기/비용 충족 16%, 납기지연 및 비용초과 53%, 실패 31% (F-22의 80%, B-2의 65%가 소프트웨어) [2015년 보고서]

5. 소프트웨어 프로세스 개선 (2/3)

- 프로세스 개선의 필요성
  - 좋은 프로세스가 없음에도 좋은 품질의 소프트웨어 제품을 생산할 수 있을까?
    - 품질을 계획하거나 관리할 수 없음
    - 좋은 품질을 얻었다 하더라도 그 이유가 무엇인지, 다시 또 반복적으로 좋은 품질을 얻을 수 있을 것인가에 대한 확신이 없음
  - 문제 해결 방향
    - 하드웨어 개발 접근 방법의 도입 : 제도 및 생산 공정, 철저한 품질관리 및 지속적인 공정개선

5. 소프트웨어 프로세스 개선 (3/3)

- 국제 표준 : CMM과 CMMI, SPICE
- Six Sigma (6σ)
- 기타 프로세스 표준
  - ISO 12207, ISO 29110, ISO 15288 등
- 국내 프로세스 개선
  - 정보통신산업진흥원(www.nipa.kr) : 소프트웨어 프로세스 품질인증제도(SP인증)