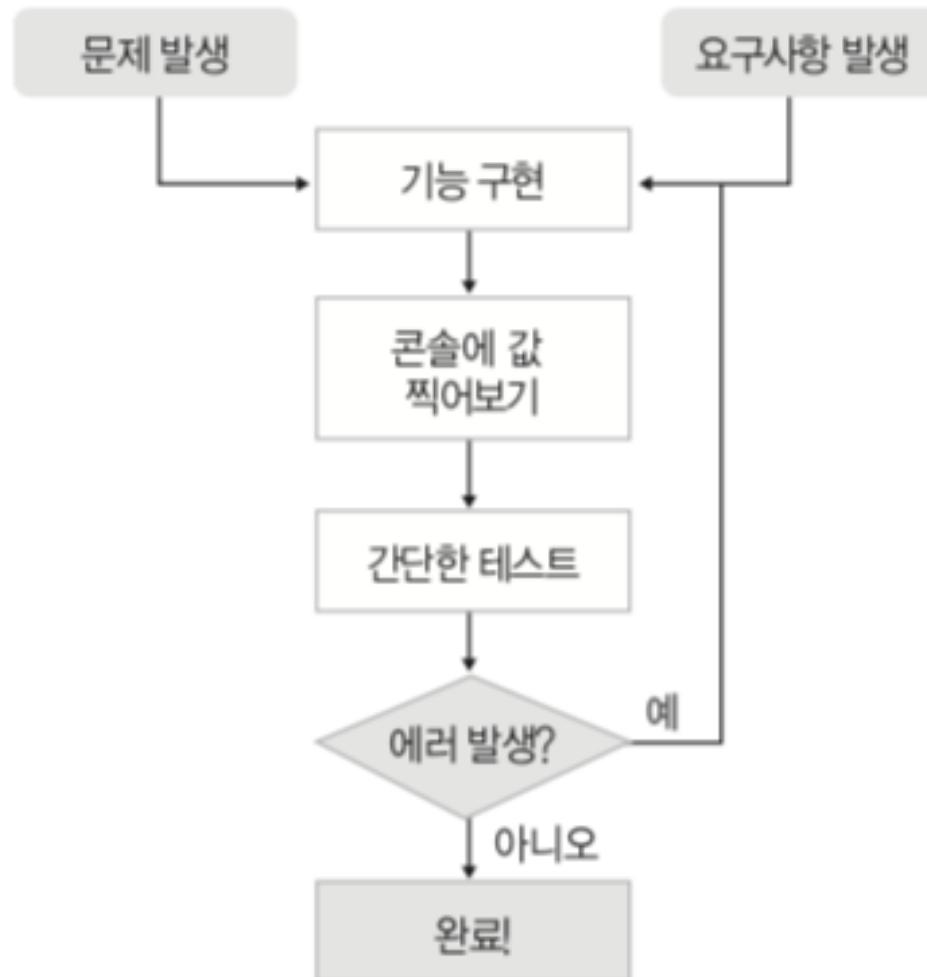


1. DevOps

개발 방법론

기존 소프트웨어 개발 방식

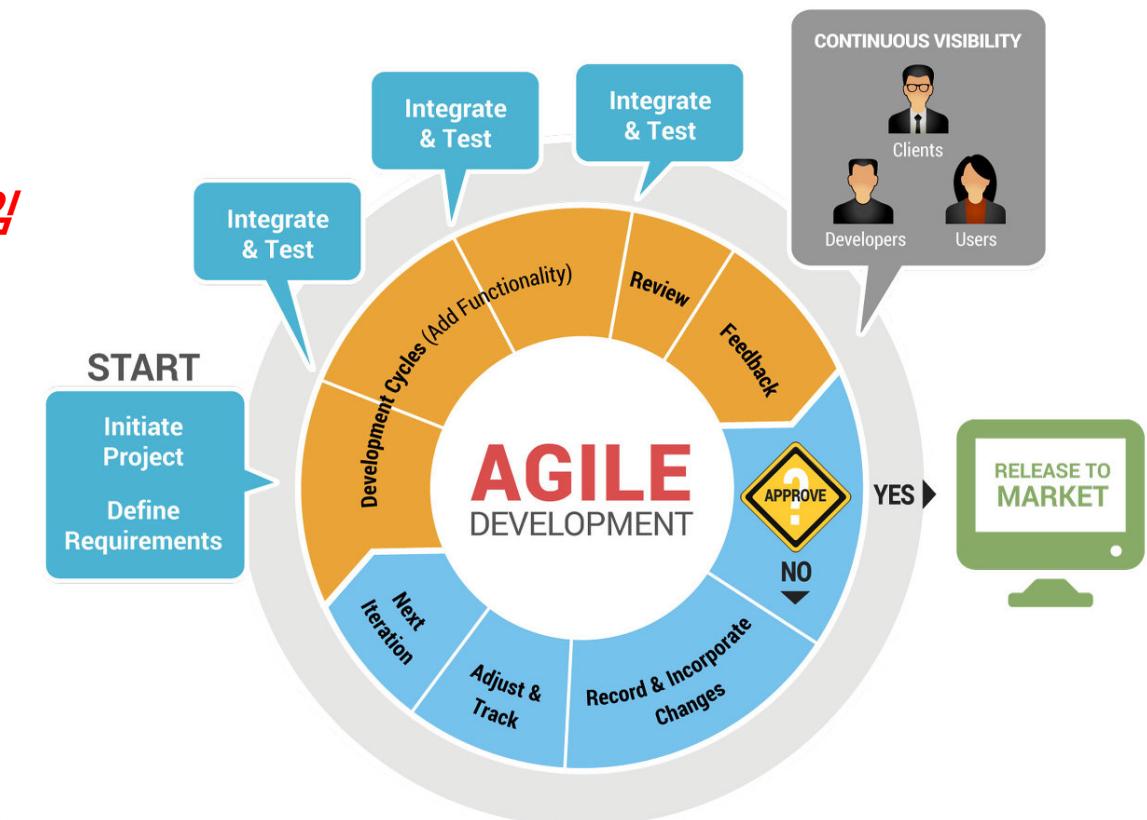


- 특정 모듈의 개발 기간이 길어질수록 개발자의 목표의식이 흐려진다.
- 작업 분량이 늘어날수록 확인이 어려워진다.
- 개발자의 집중력이 필요해진다.
- 논리적인 오류를 찾기가 어렵다.
- 코드의 사용 방법과 변경 이력을 개발자의 기억력에 의존하게 되는 경우가 많다.
- 테스트 케이스가 적혀 있는 엑셀 파일을 보면 매번 테스트를 실행하는 게 점점 귀찮아져서는 점차 간소화하는 항목들이 늘어난다.
- 코드 수정 시에 기존 코드의 정상 동작에 대한 보장이 어렵다.
- 테스트를 해보려면 소스코드에 변경을 가하는 등, 번거로운 선행 작업이 필요할 수 있다.
- 그래서 소스 변경 시 해야 하는 회귀 테스트는 곧잘 희귀 테스트(rare test)가 되기 쉽다.
- 이래저래 테스트는 개발자의 귀중한 노동력(man-month)을 적지 않게 소모한다.

■ Agile

■ Agile software development

- 가벼운 프로세스
- 협업+피드백 ★
- 민첩함, 능동적, 자발적, 형식에 구애 받지 않음
- 반복 점진 개발 + 품질 개선 활동
 - 짧은 기간 단위의 **반복 절차를 통해 리스크를 줄임**
 - 개발 주기(계획, 개발, 출시)가 여러 번 반복
- 고객의 피드백에 민첩하게 반응
- **Less document-oriented → code-oriented**
 - 프로그래밍에 집중하는 유연한 개발 방식
 - eXtreme Programming, Scrum



■ Agile

■ ***Manifesto for Agile Software Development***

“

We are uncovering better ways of *developing software* by doing it and *helping others* do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

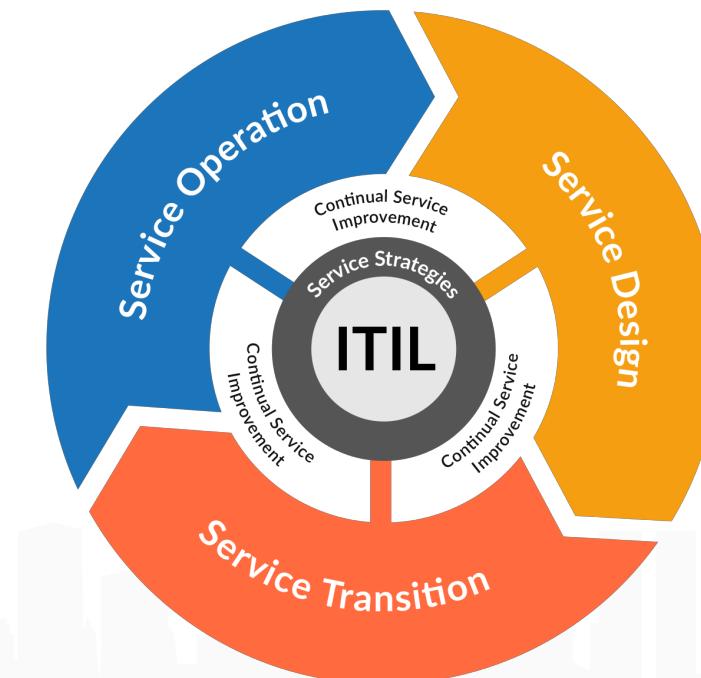
Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

<http://agilemanifesto.org/iso/en/manifesto.html>

■ ITIL

■ *IT Infrastructure Library*

- **IT 서비스 관리에 대한 프레임워크** 구현을 돋기 위한 문서들의 집합
- 과거의 기술 중심으로 업무를 집중하던 것에 대해, 비즈니스 조직의 요구사항에 따라 IT 서비스 품질 향상에 역량을 집중하고 고객 지향적인 접근방식 채택
- 영국 정부의 지적 소유물으로, 영국뿐 아니라 전 세계 **IT 서비스 관리에 대한 사실상의 표준** (de-facto standards)
 - 서비스 전략
 - 서비스 설계
 - 서비스 전환
 - 서비스 운영
 - 지속적인 서비스 개선

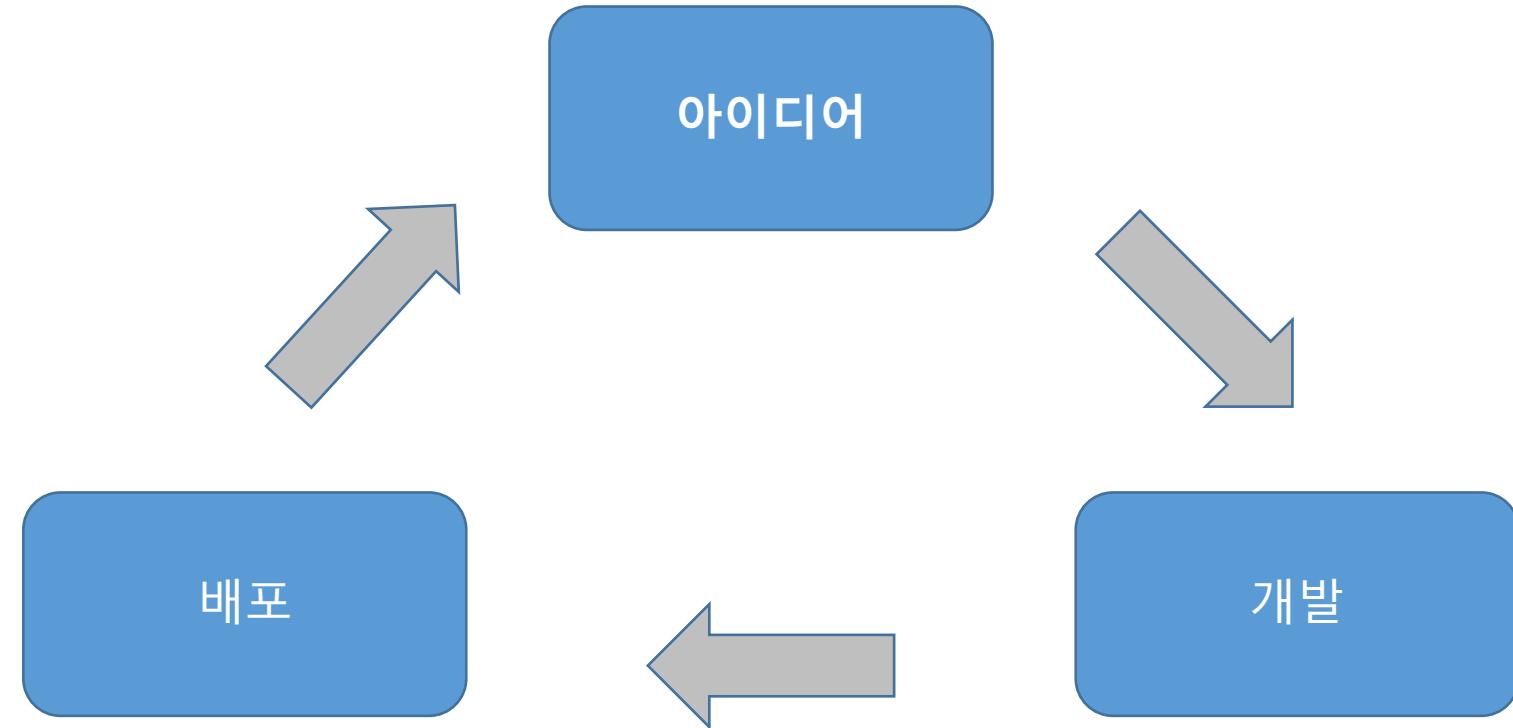


■ Lean

- 도요타의 대표적인 생산방식으로 각 생산 단계에서 인력이나 생산설비 등 생산능력을 필요한 만큼만 유지하면서 생산효율을 극대화하는 방식
 - 린 생산 방법, 디자인 중심 사고, 고객 개발, 애자일 개발 같은 기존 경영 방법 및 제품 개발 방법론의 토대 위에서 만들어진 경영 전략
 - 시장에 대한 가정(market assumptions)을 테스트하기 위해 빠른 프로토타입(rapid prototype)을 만들도록 권함
 - 고객의 피드백을 받아 기존의 소프트웨어 엔지니어링 프랙티스(폭포수 모델 같은) 보다 훨씬 빠르게 프로토타입을 진화시킬 것을 주장
 - 지속적 배포(Continuous Deployment)라는 기법을 사용

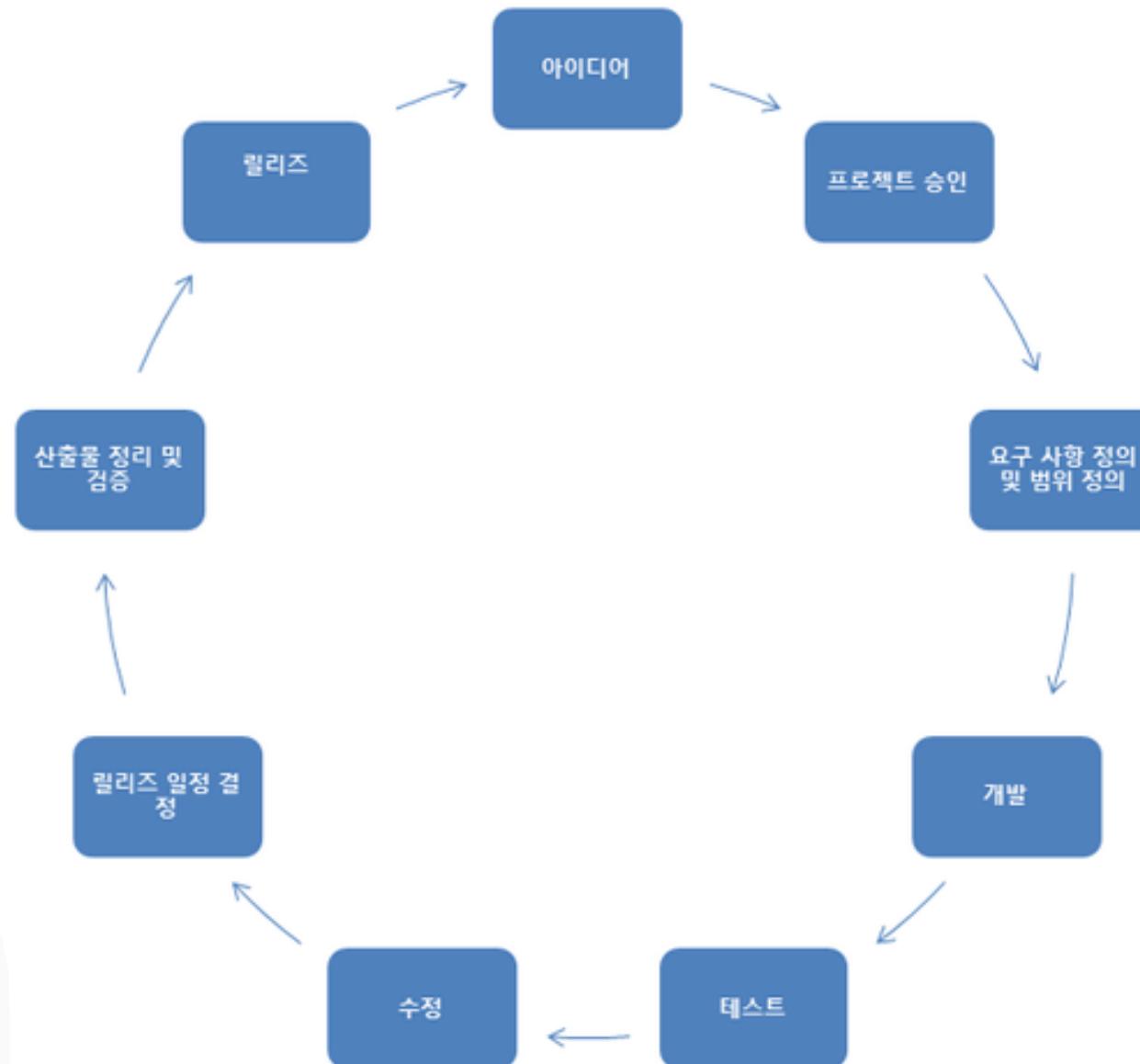
■ 개발 모델

1. 스타트업



■ 개발 모델

2. 체계화된 조직



■ 개발 모델

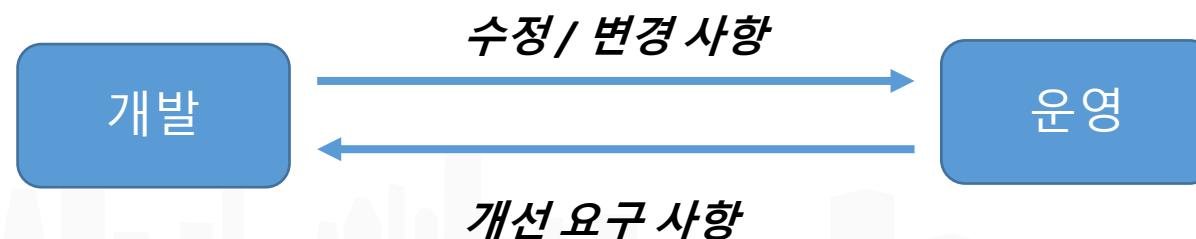
3. CD + Devops 기반

■ CD (Continuous Delivery)

- 운영 시스템에 계속해서 Fix나 새로운 기능을 지속적으로 Release를 하는 개념
- 새로운 FIX나 기능이 추가되면 거의 매일 Release를 하는 개념

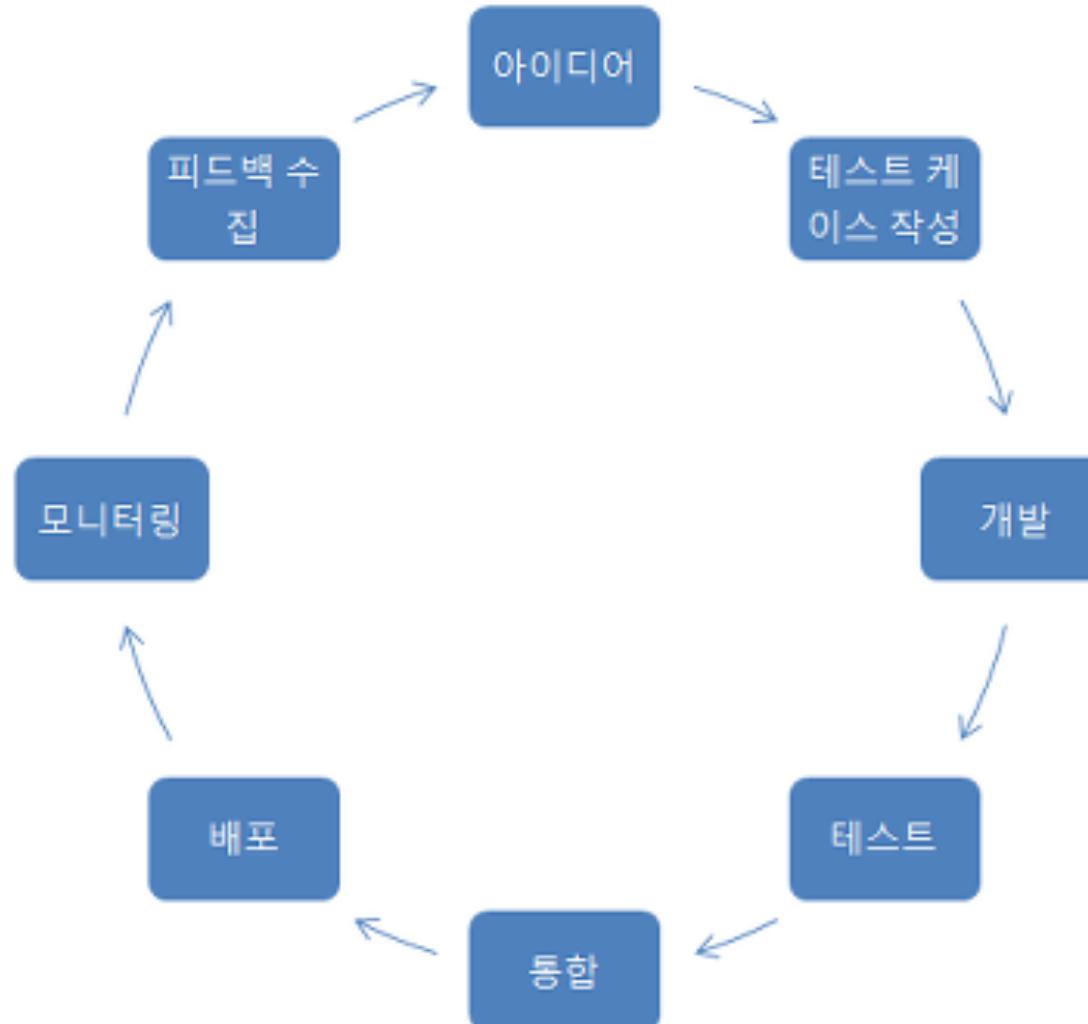
■ Devops (Development + Operations)

- 개발팀 + 운영팀



■ 개발 모델

3. CD + Devops 기반



- TDD
- 자동화 필수
 - 복잡한 인프라,
미들웨어에 대한 배포
자동화

■ Continuous Deployment, Continuous Delivery

■ 지속적 배포

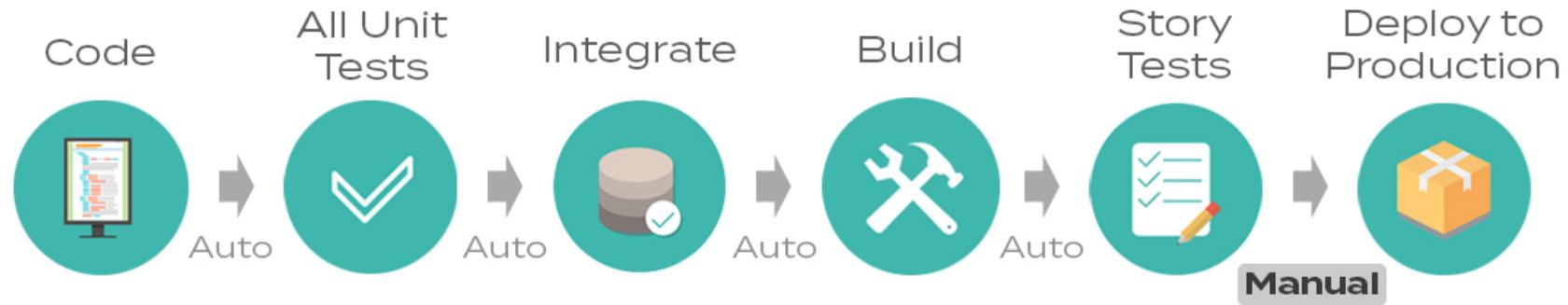
- Continuous Delivery
- Continuous Deployment
- Pipe line

■ 카나리 배포와 블루그린 배포

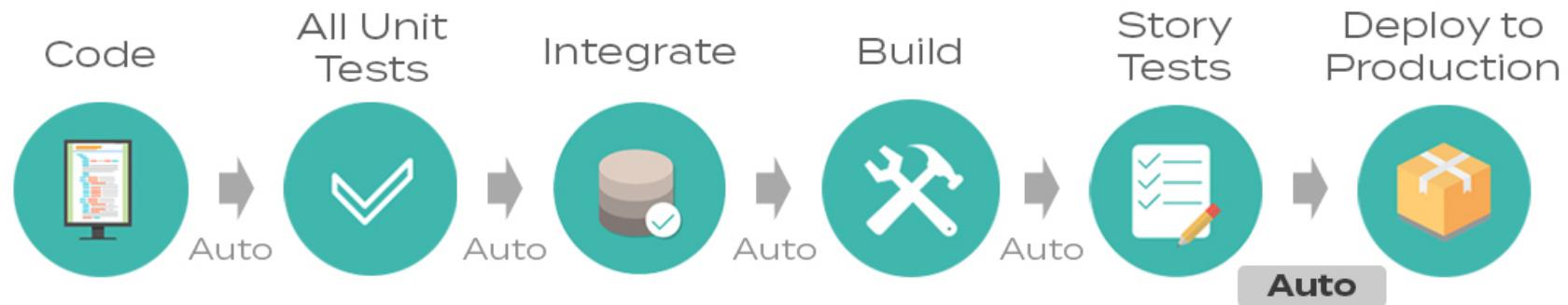


▪ Continuous Deployment, Continuous Delivery

Continuous Delivery

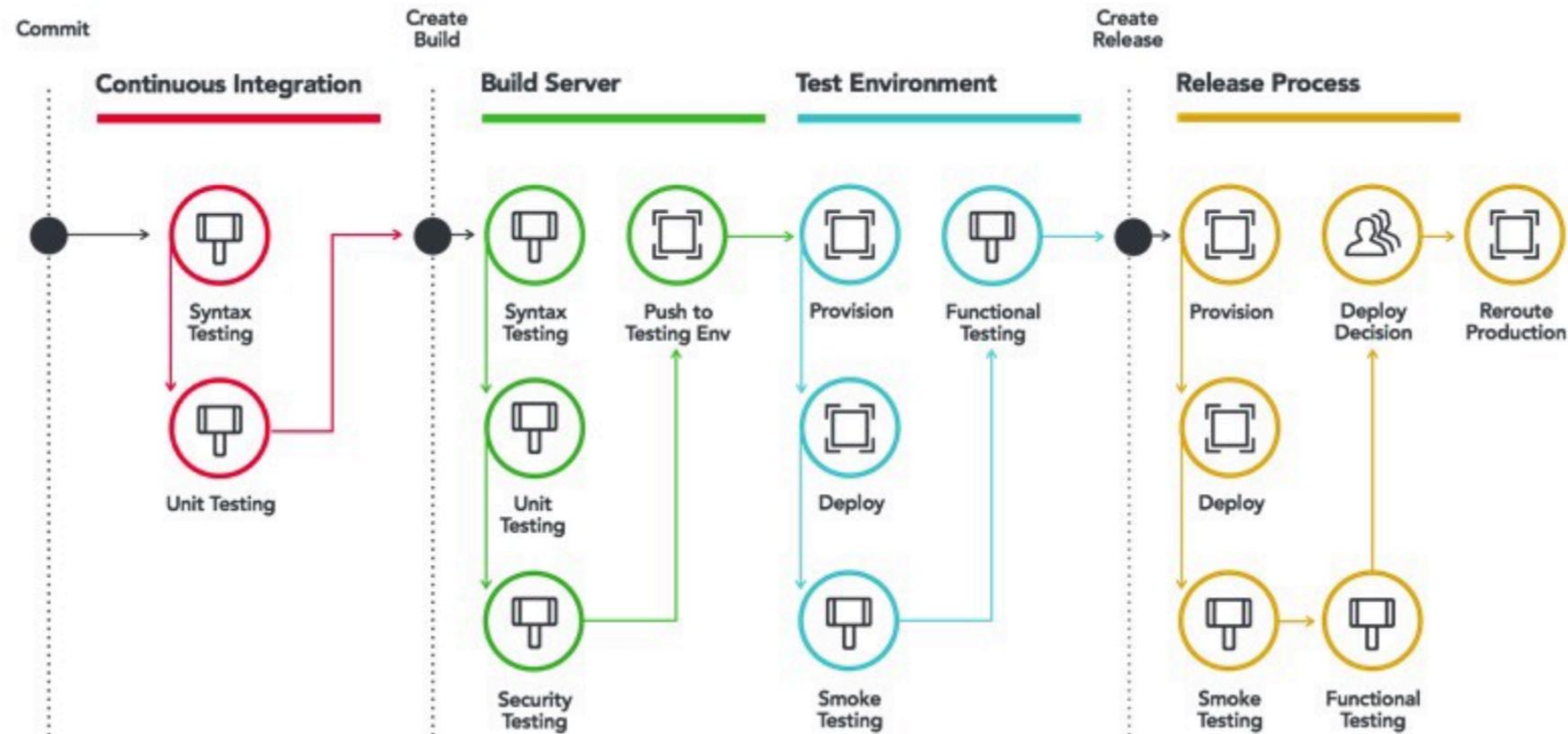


Continuous Deployment



■ Continuous Deployment, Continuous Delivery

- 수동 배포가 아닌, 여러 환경과 수십대의 서버에 배포 해야 하는 경우
- CI + CD (Continuous Delivery, Continuous Deployment)

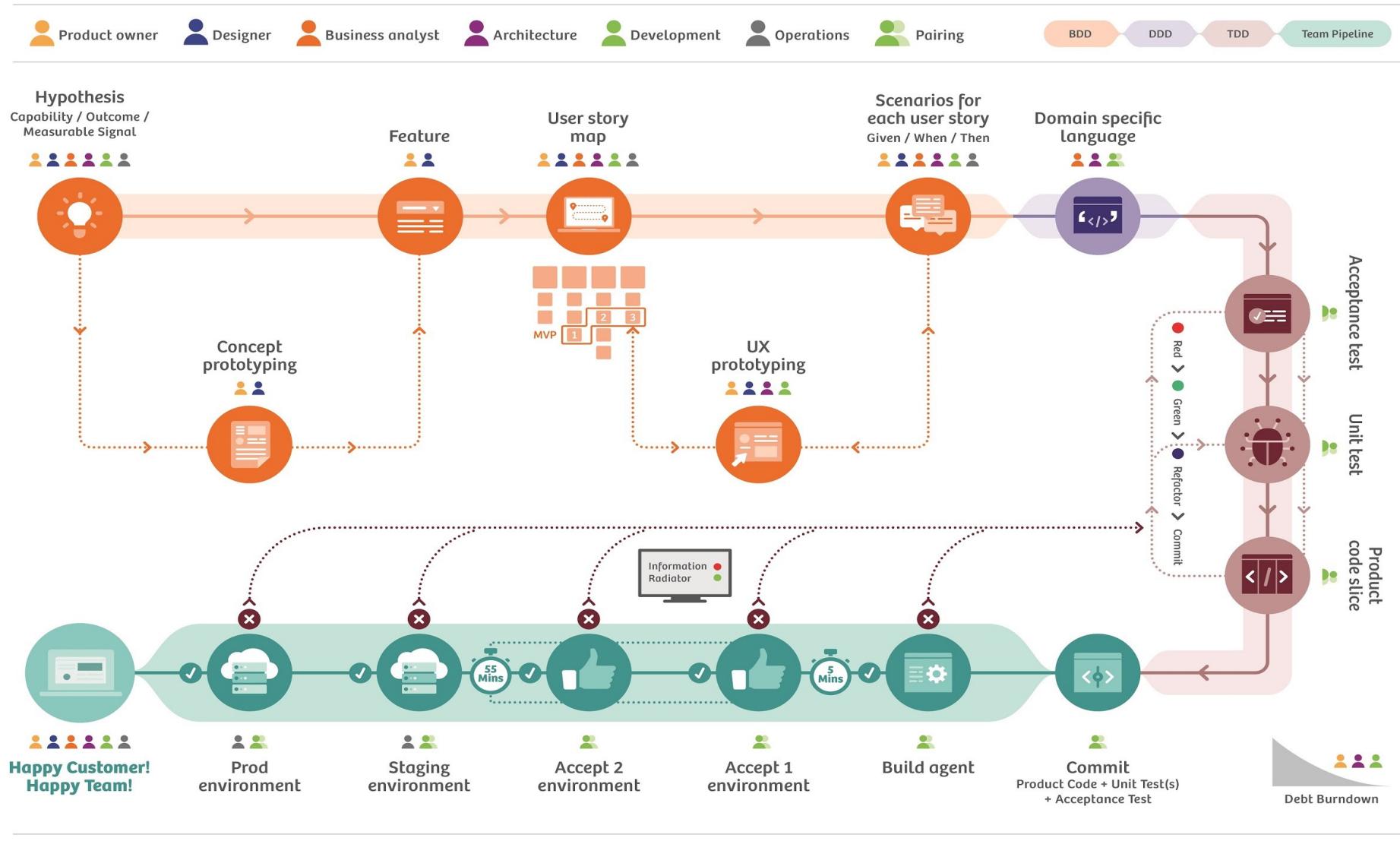


<https://davidtucker.net/devops-101-continuous-delivery-81b66a6bd2d3>

Continuous Deployment, Continuous Delivery



TDD + BDD + DDD + Pairing + Team Pipeline = **Continuous Delivery**



■ *Continuous Deployment, Continuous Delivery*

■ 배포 주기

- 특정 날짜 → 업데이트 주기가 짧아짐 → 배포 자동화 ex) SNS 하루 단위 배포
- 배포 시 문제 발생 → 개발팀 도움 필요
- 배포 후 개발팀으로 부터 확인 필요

■ 수동 배포

- 어느 정도 수동 배포 필요 → 운영 환경 배포

■ 배포 Roll back

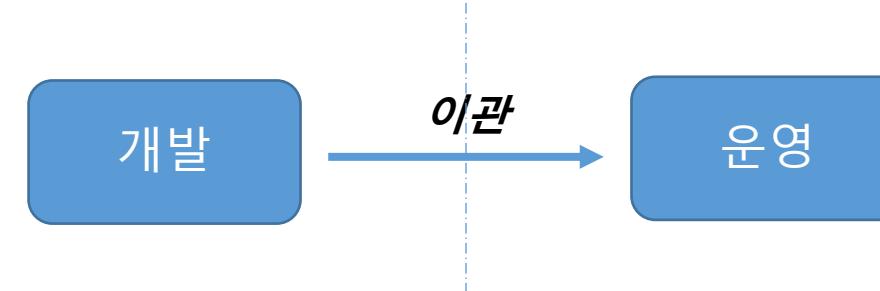
- 배포 시 오류 발생할 경우, 이전 버전으로 배포할 수 있는 기능 필요

■ Release Note

- Release Version, Date, Build Number
- New Features 및 설명
- Bug Number 및 버그 수정 내용

- 안드로이드 릴리즈 노트 : <https://developer.android.com/preview/release-notes>
- Fire Fox 릴리즈 노트 <http://www.mozilla.org/en-US/firefox/67.0.2/releasenotes/>
- Maven 릴리즈 노트 <http://maven.apache.org/release-notes-all.html>

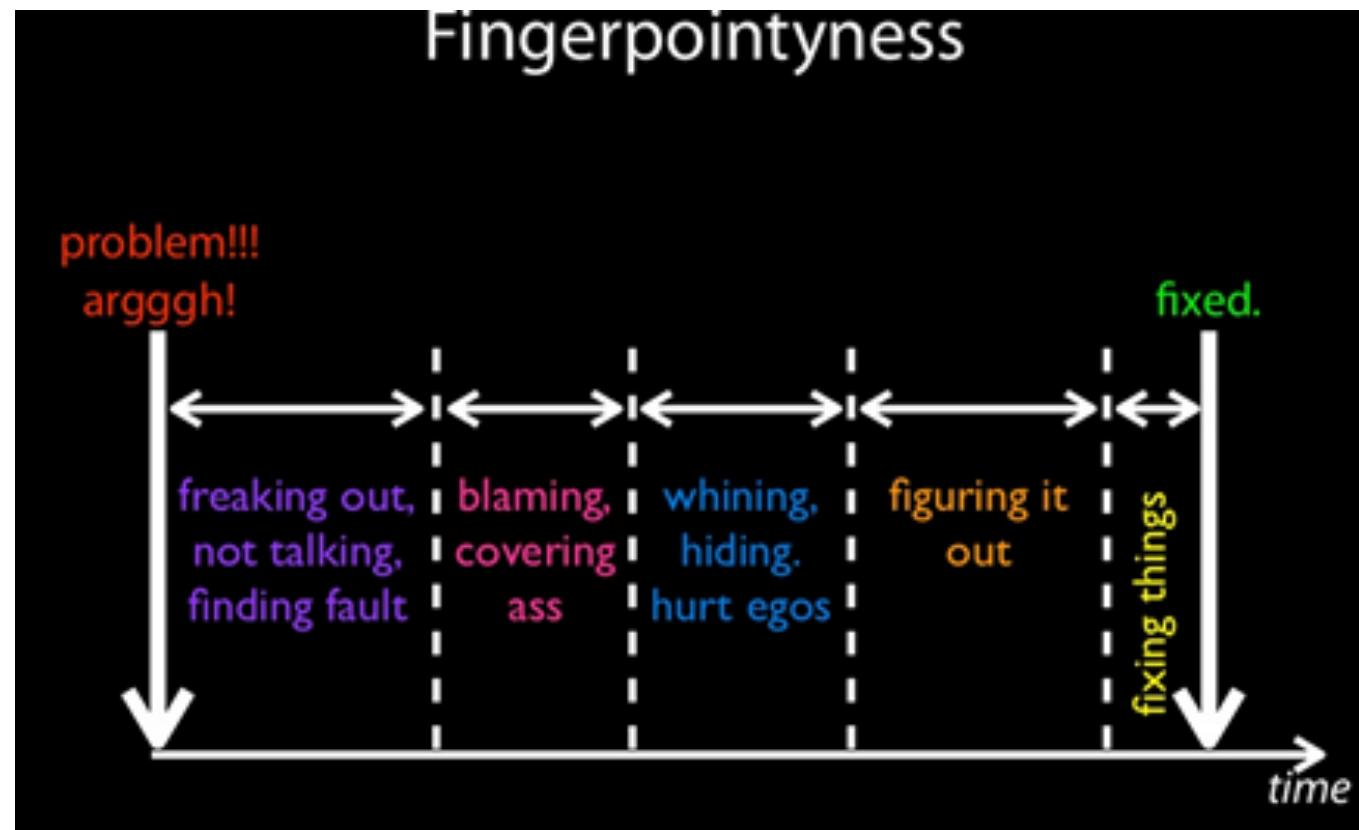
■ 전통적인 개발 운영 체계



- 문제점 1. 장애 발생 → 누구의 잘못?
- 문제점 2. 운영 이슈에 대한 전달 문제
- 문제점 3. 변경 요건

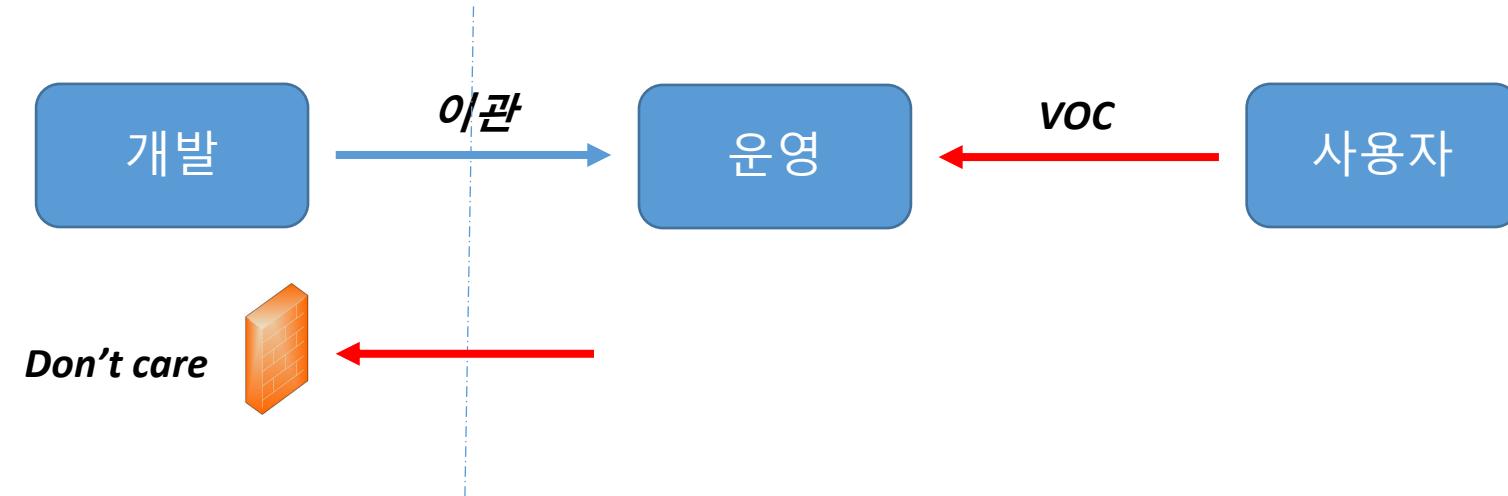
■ 전통적인 개발 운영 체계

- 문제점 1. 장애 발생 → 누구의 잘못?



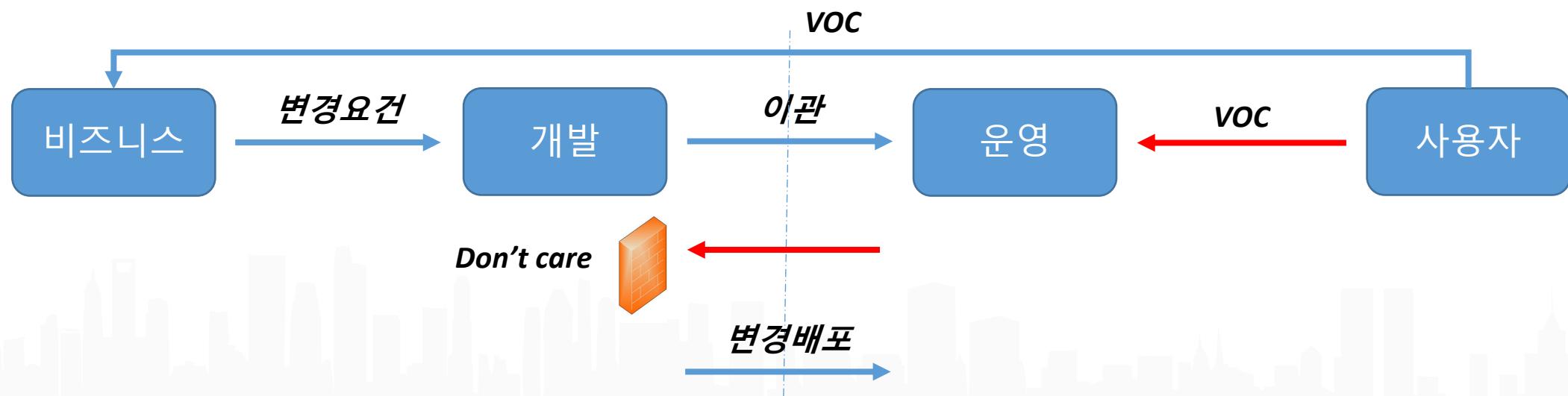
■ 전통적인 개발 운영 체계

- 문제점 2. 운영 이슈에 대한 전달 문제



■ 전통적인 개발 운영 체계

- 문제점 3. 변경 요건
 - 신규 요구 사항
 - 새로운 변경 요건
 - 신규 개발
 - 테스트 배포
 - 지속적인 운영



■ 전통적인 개발 운영 체계

- 해결책은?
 - 개발팀과 운영팀과의 관계 해결
 - 서비스 요구 사항의 신속한 반영의 문제
 - 고객의 요구 사항에 민감한 SW 개발

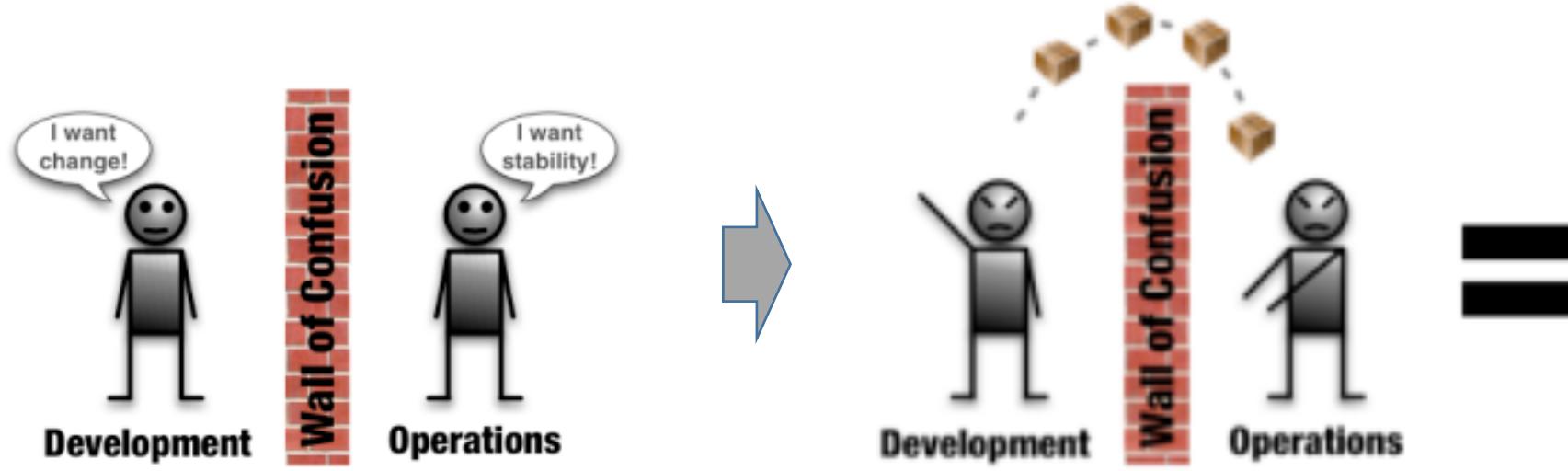
1. 협업

- 기획 + 개발, 운영
- Iterative 개발, Short Release

2. *Devops*

- 개발 + 운영

■ What is DevOps?



<http://dev2ops.org/2010/02/what-is-devops/>

- *Dev*: 변화(변경)
- *Ops*: 안정성
- 동기(*Incentives*) ?
- 프로세스(*Process*) ?
- 도구(*Tools*) ?

■ *What is DevOps?*

■ 2007~2008년 → IT 운영 및 SW 개발에 문제점 대두

- 개발과 배포가 다른 조직에서 관리 (다른 목표를 가짐)

■ 2009년 Belgium에서 첫 DevOps 컨퍼런스

- 인프라로 코드 관리 (Mark Burgess and Luke Kanies)
- 애자일 인프라 스트럭쳐 (Andrew Shafer)
- 애자일 시스템 관리 운동 (Patrick Debois)
- Lean Startup (Eric Ries)
- 지속적인 통합 및 배포 운동 → CI, CD

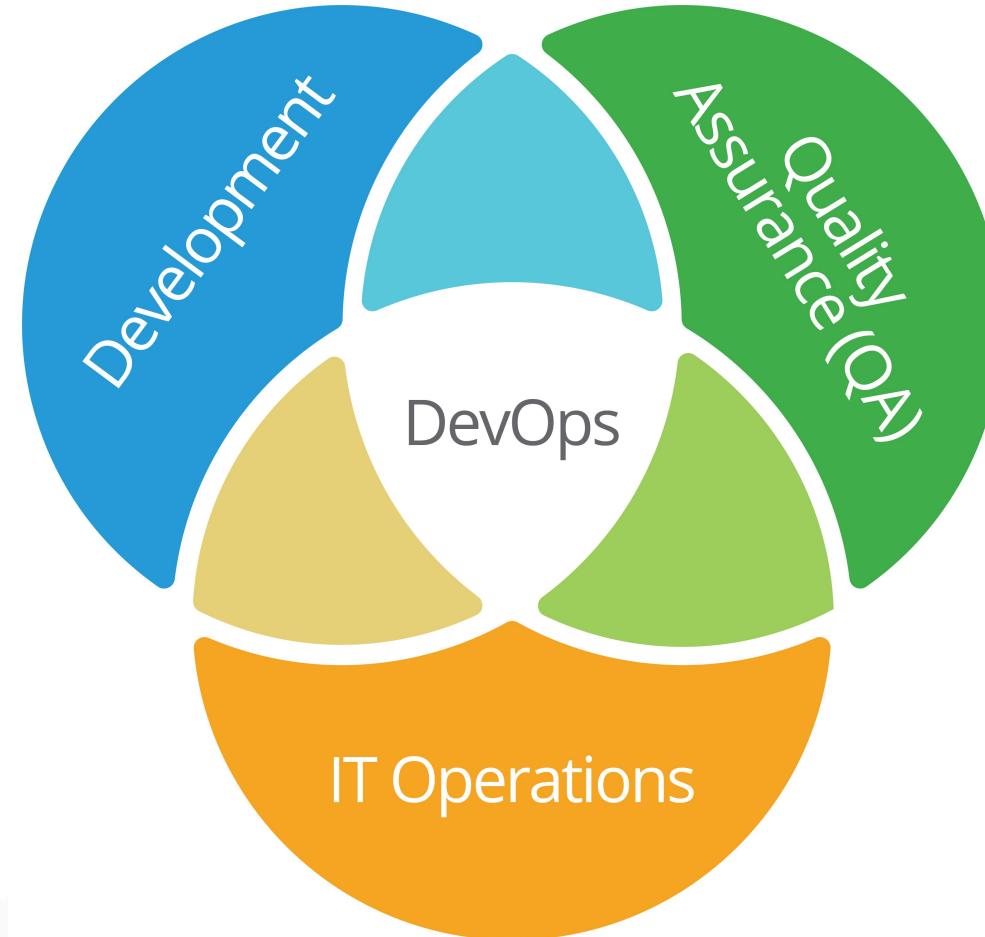
■ What is DevOps?

- Amazon, Google, Netflix과 같은 기업에서 각종 Cloud 기술 확대
- 협업 및 신뢰
- 더 빠른 릴리스, 더 스마트한 업무 진행
- 문제 해결 시간 단축
- 계획되지 않은 업무 쉽게 관리



■ What is DevOps?

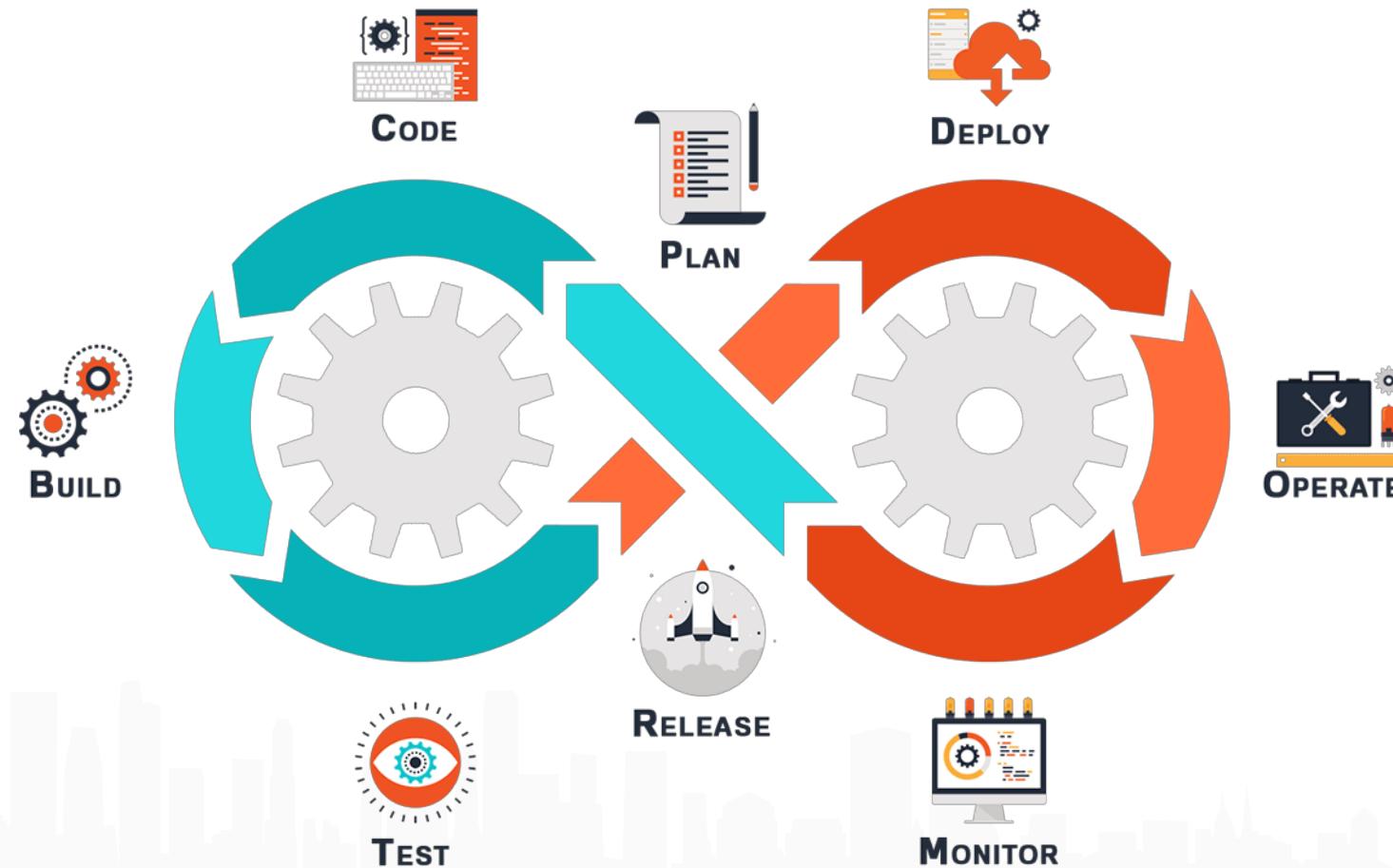
“개발과 운영이 분리되면서 오는 문제점을 해결하기 위해서, **개발과 운영을 하나의 조직으로 합쳐서 팀을 운영하는 문화이자 개발 방법론이다**”



■ What is DevOps?

“엔지니어가, **프로그래밍**하고, **빌드**하고, 직접 시스템에 **배포** 및 서비스를 **RUN**”

“사용자와 **끊임 없이** **Interaction**하면서 **서비스를 개선해** 나가는 일련의 과정, **문화**”



■ What is DevOps?

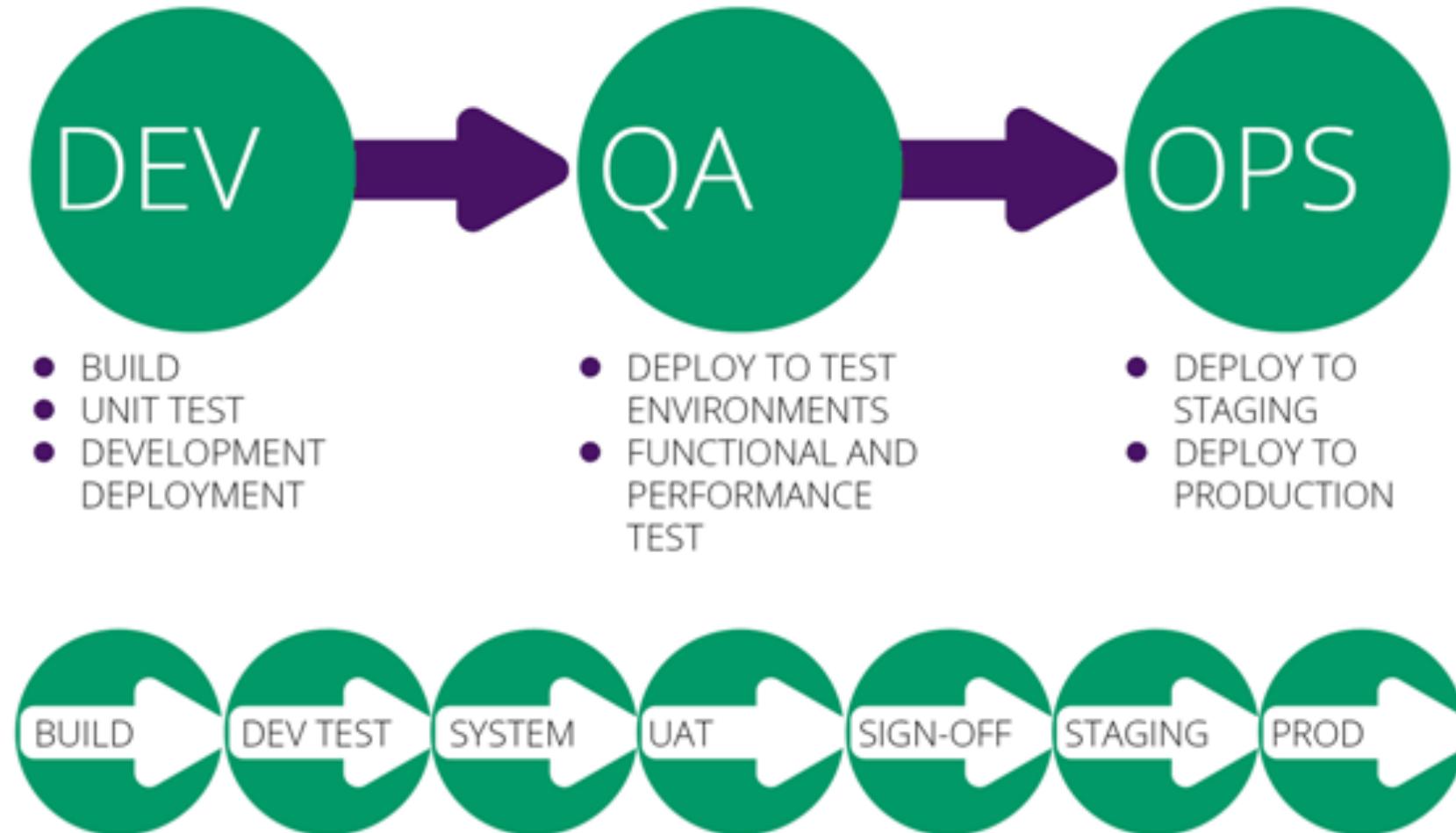
■ DevOps 적용 시

- 30배 더 자주 Deployment
- Deployment 실패 비율도 50% 감사

Company	Deploy Frequency	Deploy Lead Time
Amazon	23,000 / day	MINUTES
Google	5,500 / day	MINUTES
Netflix	500 / day	MINUTES
Facebook	1 / day	HOURS
Twitter	3 / week	HOURS
Typical Enterprise	Once every 9+ months	MONTHS or QUARTERS

■ What is DevOps?

■ DevOps cycle



■ DevOps 관점

■ DevOps 가 성공하기 위한 기초

- 신뢰의 문화
- 동료의식
- “us vs. them” 문화 vs “**we**” 문화?
- DevOps 에서 가장 중요한 것은 **공유**
 - 아이디어, 이슈, 프로세스, 툴, 그리고 목표
- Devops 에서 중요한 것은 **문화와 사람**
 - 그 다음이 프로세스와 도구

■ DevOps 관점

■ DevOps를 어떻게 판단할 수 있는가?

- **측정지표** 관점

- quality, testing, 공유된 동기의 정도

- **프로세스** 관점

- 합동, 빠른 피드백을 위한 흐름, 전체적인 프로세스 구성

- **기술** 관점

- 자동화를 통한 빠른 피드백

- “인프라를 코드로 관리하기” 와 같은 자동화된 Release

■ DevOps 사례

■ Facebook

- 하루 2번 Minor 업데이트, 일주일에 한번 Major 업데이트
- 내부 사용자를 위한 H1 배포
- 실사용자 중 1%의 사용자를 위한 H2 배포
- 전체 사용자를 위한 H3 배포
- 빌드에 15분, 배포에 15분
- IRC를 통해 배포 상황 및 에러 대응 (응답 없는 개발자 코드는 빼고 배포)
- 배포 후 내부 모니터링 외에 외부 SNS를 스크롤링 하여 모니터링
- 배포 시 문제가 생기면 바로 롤백

■ DevOps 사례

■ Flickr

- 하루에 10번 배포
- 개발 코드, QA 코드, 인프라 설정 스크립트 등을 하나의 SCM으로 결합하여 모든 팀이 공유

■ Netflix

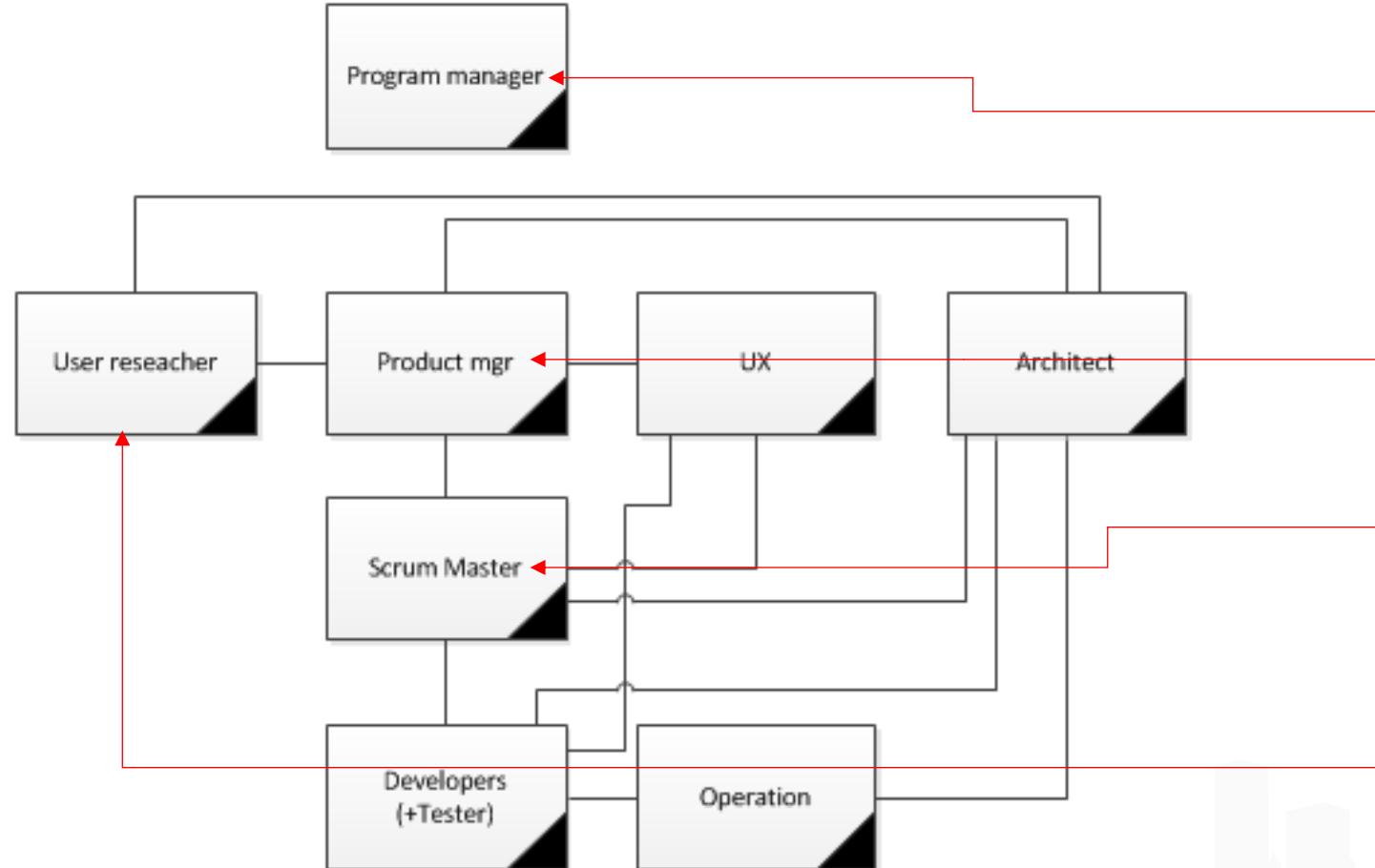
- 배포 팀에 의해 Test, Release, Prod Branch 운영
- Canary analytics
 - 운영 환경과 같은 환경 (Pre Prod)으로 1000개의 모니터링 지표 테스트 후 배포
- 기존 인스턴스가 아닌 새 버전용 인스턴스에 배포 후, 문제 없으면 기존 인스턴스 삭제

■ Google

- SRE (Site Reliability Engineering) 팀 운영
- 개발 완료 후 6개월 정도 개발팀이 운영 → 운영 매뉴얼을 첨부하여 SRE 팀에 리뷰 → 운영 팀

■ DevOps 기반의 개발팀

- End to End
- <https://www.gov.uk/service-manual/the-team>

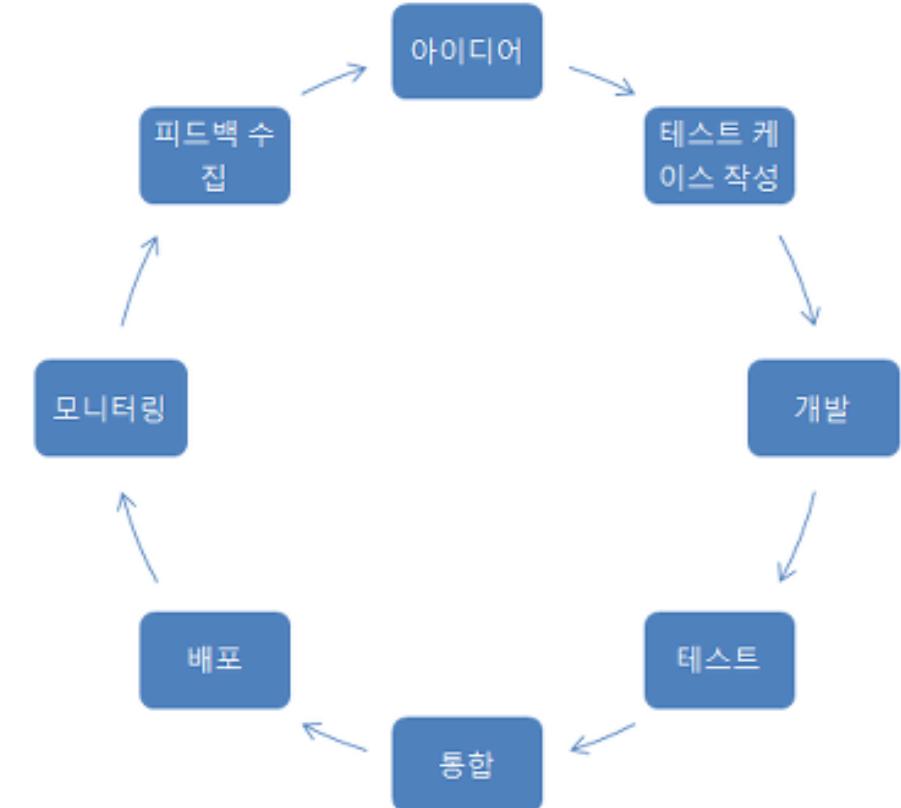


- **전체 서비스 관장**
 - 개발, 운영, 서비스, 기획, Stakeholder, Communication 등
- **서비스 기획, 요구 사항 정의, 우선 순위 결정**
- **Project Leader**
 - 일정관리, 개발 리소스 관리
- **서비스 전략**
 - 서비스의 방향, 시장 분석
 - 수익 구조, 비즈니스 모델 정의, 로드맵 정의

■ DevOps 기반의 개발팀

■ Development cycle

- ① 사용자의 요구사항 분석. VoC 수집
- ② 사용자 스토리 작성 (요구 사항 작성)
- ③ 사용자 스토리에 대한 업무 범위 정의 및 우선순위 지정
- ④ 이해 관계자에 대한 리포팅 및 관리 (내부 영업, 보고 등)
- ⑤ 다른 프로젝트와 관리
- ⑥ 필요의 경우 솔루션 (오픈소스 또는 상용) 평가 및 도입
- ⑦ 개발 (디자인, 빌드, 테스트, 데모, Iterative way)
- ⑧ 테스트 (UAT)
- ⑨ 서버에 배포
- ⑩ Security 관리, Compliance 관리 (개인 정보 보호, 국가별 법적 사항 확인 등)
- ⑪ 서비스 운영, 모니터링.



■ DevOps 기반의 개발팀

■ 필요 역량

- 코딩능력

- Devops 엔지니어는 기본적으로 개발자를 기본
- 개발을 위한 기본적인 코딩 능력. 자동화를 위한 스크립트 작성 능력 등

- 다른 사람과 잘 협업하고 커뮤니케이션할 수 있는 능력

- Devops는 협업 문화
- 개발과 운영 간의 소통 문제를 해결
- 오픈 마인드 기반의 커뮤니케이션 능력이 매우 중요

- 프로세스를 이해하고 재 정의할 수 있는 능력

- 테스트 자동화, 배포, 그리고 요구 사항에 대한 수집 및 정의
- 해당 팀의 모델이나 서비스의 성격에 따라서 프로세스 구축

■ DevOps 프레임워크

■ CALMS



“DevOps를 활용하는 팀은 30배 더 자주 배포할 수 있고, 실패는 60배 줄일 수 있으며, 160배 빠르게 복구할 수 있습니다.”

Puppet Labs 2016 DevOps 상태 보고서

■ *DevOps 핵심*

- 가치와 목적
- 프로세스
- 도구



■ DevOps 핵심

■ 가치와 목적

- 서로간의 존중
- 공유된 목적에 대한 합의
- 공동의 소유권(ownership)
- 가치의 공유
- 전체 차원의 “동기, 목적 ” 조정
 - “quality” 정의
 - 가시성, 투명성
 - 하나의 팀

■ DevOps 핵심

■ 프로세스

- Dev와 Ops 간에 동일 **프로세스를 공유**
- 두 그룹이 초점을 맞춤
 - 최대의 빈도와 품질로 사용자에게 애플리케이션 변경을 **배포**
- 통합된 프로세스를 통해서 ***cycle time* 을 줄임**

■ DevOps 핵심

■ 도구

- **DevOps 의 능률은 자동화**에 의존
 - 빌드 시스템 / 소스코드 관리 시스템 / 기술적, 기능적 acceptance 테스트 / 패키징 / 제품 배포
 - 모든 단계에서 자동화된 적절한 도구가 필요
 - 코드와 스크립트
 - 모두 버전 관리 시스템에 저장
 - **코드와 스크립트는** 빌드, 테스팅, 배포 및 puppet 등의 **설정 관리 프로그램에 대한 내용도 포함**
 - 인프라를 코드로 관리

■ DevOps 구성

■ 측정기준

- **cycle time**
- 하나의 operation, function, process를 완료하는데 필요한 기간
- Batch size를 줄이면서 **더 자주 배포**하여 cycle time을 줄임

■ 전통적인 측정 기준

- 숫자 / 정적 코드 분석, 테스트 coverage / 기능(function points) 등
- 이슈 10개 처리한 사람과 100개 처리한 사람을 비교하기 위한 방법은?

■ 애자일 측정 기준

- 지속적으로 가치가 있는 소프트웨어를 배포할 수 있도록 고객 피드백, 지속적인 테스팅, 반복적인 개발을 보장할 수 있는지를 접근방법을 필요
- 테스팅이 완료되지 않거나, 대상 시스템에 설치가 안 되었거나, 품질이나 모니터링을 보장하지 못하면 끝나지 않음

■ DevOps 구성

■ Dev와 Ops 간에 사용되는 지표

- 변경

■ 변경

- *Application code*
- *Middleware*
- *Infrastructure (OS, Servers, Switches, Routers)*



■ DevOps 구성

■ 배포 개선 및 가속화

- 자동화된 *Release*
- *Deployment* 와 *Release* 분리

- 지속적인 통합 (Continuous integration)
- Automatic tests
- Build, test, releasing 소프트웨어를 이용 → 반복 가능한 프로세스
- 가상 머신, 미들웨어, 애플리케이션 코드 등을 “반복” 가능하도록 Provisioning, deploy
- 기술적 관점이 아닌 비즈니스적인 관점에서 자동화
- 릴리즈를 자주, 반복적으로 함
- 자동화된 릴리즈 → 모니터링

■ DevOps 구성

■ 배포 개선 및 가속화

- 자동화된 Release
- Deployment 와 Release 분리

- Deployment : 특정 환경에 소프트웨어 Release 를 설치
- Release : 사용자가 사용할 수 있는 특정 소프트웨어 버전
- Deployment 와 Release 분리하는 방법
 - Branch by Abstraction
 - Feature Toggles : Runtime 시 사용할 기능을 선택
 - Dark Launching : 전체 사용자에게 배포하기 전 prod에 일부만 배포
 - Blue-Green Deployment : Old version, New version 을 함께 운영하면서 Load Balancer 등을 이용

■ *Top 11 Things You Need To Know About DevOps*

- 1) *DevOps는 무엇이며 어디에서 왔나?*
- 2) *DevOps와 애자일의 차이점은?*
- 3) *DevOps와 ITIL 또는 ITSM과의 차이점은?*
- 4) *How does DevOps fit with VisibleOps?*
- 5) *DevOps의 기초 원리는?*

System Thinking → Amplify Feedback Loops → Culture of continual experimentation and learning
- 6) *DevOps 패턴의 영역*
 - Area 1: Extend Development into Production
 - Area 2: Create Production feedback into Development
 - Area 3: Embed Development into IT Operations
 - Area 4: Embed IT Operations into Development

■ *Top 11 Things You Need To Know About DevOps*

7) *DevOps의 가치는?*

- 시장에 대한 빠른 대응
- " Quality" 향상
- 조직적 효율성 증대

8) *보안과 QA는 어떻게 DevOps에 합류할 수 있는가?*

- 기능, 통합, 보안 테스트를 개발의 일일 운영에 통합하여 더 빠르게 문제를 찾아냄

9) *My DevOps Favorite Pattern #1*

- 환경을 최대한 일찍 만듬
- 자동화 된 환경 생성 프로세스, Dev, QA, Prod
- 각 환경의 차이를 최대한 줄임

■ *Top 11 Things You Need To Know About DevOps*

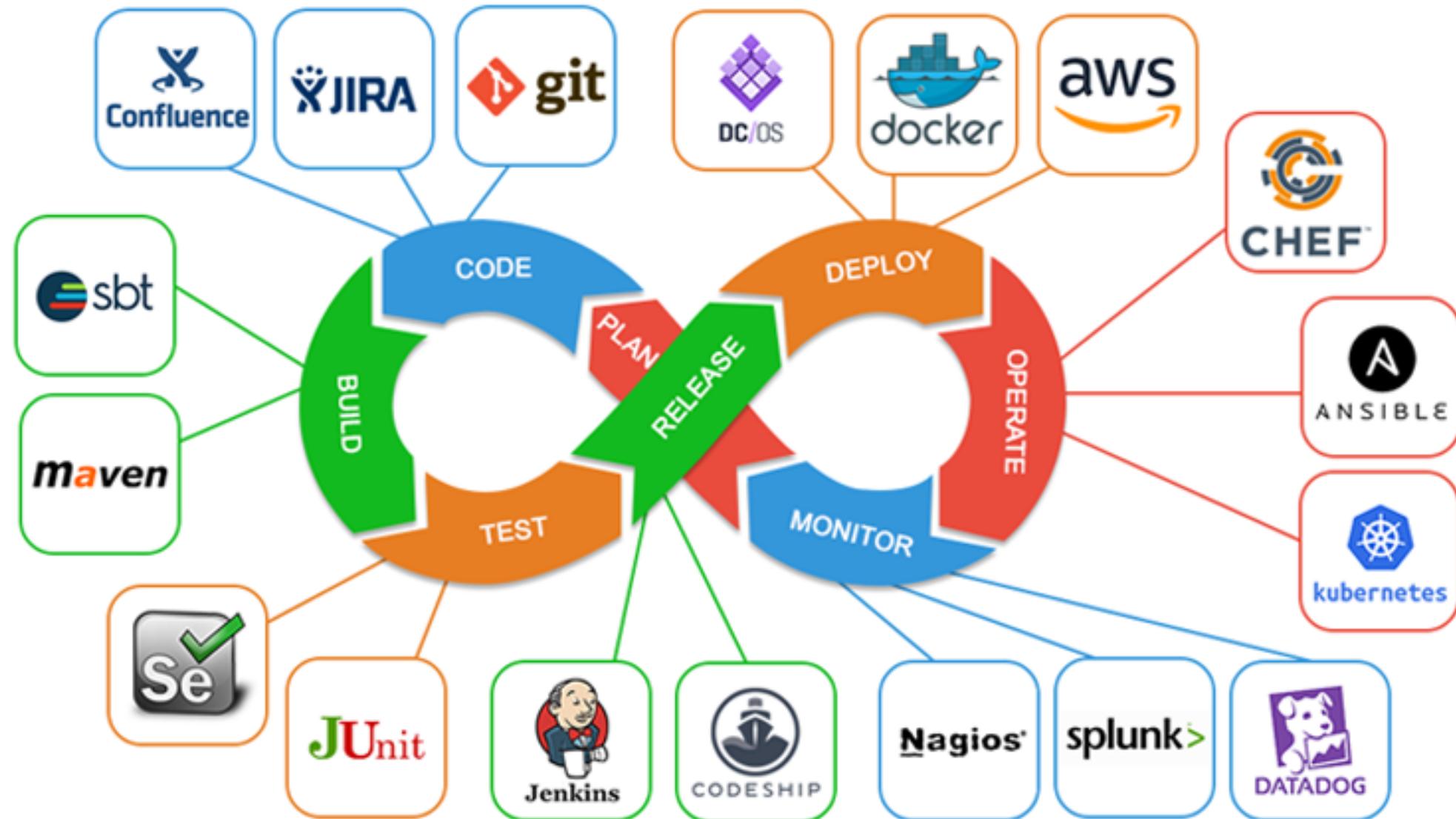
10) *My DevOps Favorite Pattern #2*

- Dev를 고객의 경험과 최대한 밀접하게 연결시킴
- Dev를 Level 3 기술지원에 포함 또는 코드 배포, 롤백 등에 함께 책임을 가지도록 함
- 목표는 Dev가 Ops를 대체하는 것이 아니라, 개발을 통한 변경 영향을 직접 확인하고 공동의 목표를 달성하기 위해서 Ops 와 함께 문제를 빠르게 고치기 위한 것

11) *My DevOps Favorite Pattern #3*

- 표준화가 안 되어 있으면 문제가 생김
- 사용 가능한 배포 프로세스 정의
- 환경을 표준화

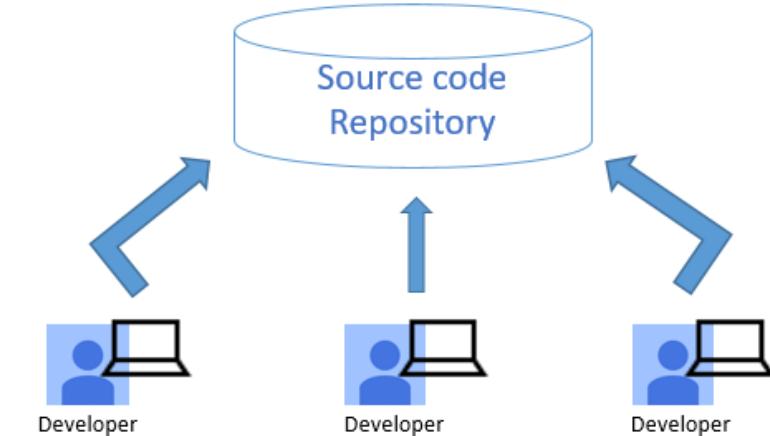
▪ DevOps 도구



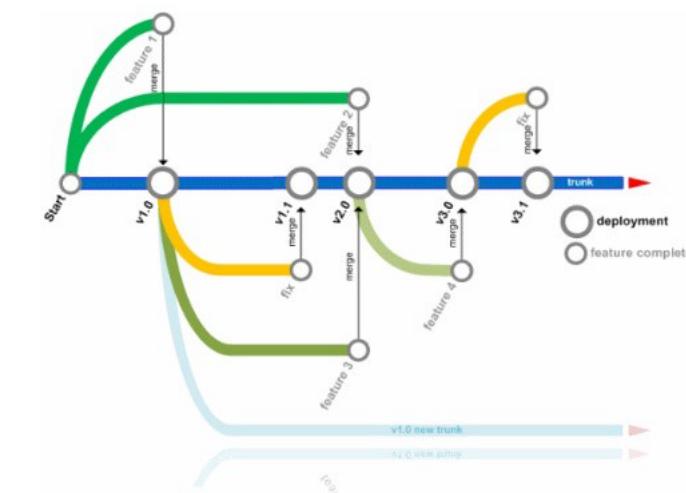
■ DevOps 도구

■ 버전 관리

- 소스 코드
- 요건 정의서, 설계서 등의 문서
- 데이터베이스 스키마 및 데이터
- 미들웨어 등의 설정 파일
- 라이브러리 등의 의존 관계 정의
- 시스템 설정 및 인프라
- 브랜치, 태그 등을 사용한 효율적인 개발



Branch and Merge



■ DevOps 도구

■ 퍼켓 관리

- 이슈 처리
- 버전 관리 시스템의 연계

The screenshot shows the JIRA Software interface for the 'Teams in Space' project. The dashboard includes a sidebar with links like Backlog, Agile board, Releases, Reports, All Issues, Components, and Add-ons. The main area displays version information (Version 6.3.3, UNRELEASED), start date (10 Aug 2015), release date (9 Oct 2015), and release notes. A progress bar indicates 28 days left. Key statistics are shown: 12 Warnings, 106 Issues in version, 73 Issues done, 4 Issues in progress, and 29 Issues to-do. Below this is a table of 106 issues, each with a key, summary, assignee, status, and development status. The table shows various tasks such as Afterburner reporting capability, revision VI automation, and video chat interface, many of which are marked as 'DONE'.

P	T	Key	Summary	Assignee	Status	Development
↑	✓	TIS-111	The revolutionary Afterburner reporting capability	Jeff	DONE	UNDER REVIEW
↑	✗	TIS-110	Afterburner revision VI automation	Bryan	DONE	
↑	✓	TIS-109	Afterburner revision VI script	Sherri	DONE	MERGED
↑	✓	TIS-108	Afterburner revision VI demo	Brandon	DONE	MERGED
↓	✓	TIS-107	Afterburner revision VI prototype	Jay	DONE	
↑	✓	TIS-106	Add video chat interface	Kellie	DONE	1 commit
↑	✗	TIS-105	Create video of launch	Sara	DONE	
↑	✓	TIS-104	Write blog post for launch	Carlos	DONE	3 commits
↑	✓	TIS-103	Review pre-launch checklist	Kelly	DONE	
↓	✓	TIS-102	Afterburner revision VI redundant test	Karen	DONE	

■ DevOps 도구

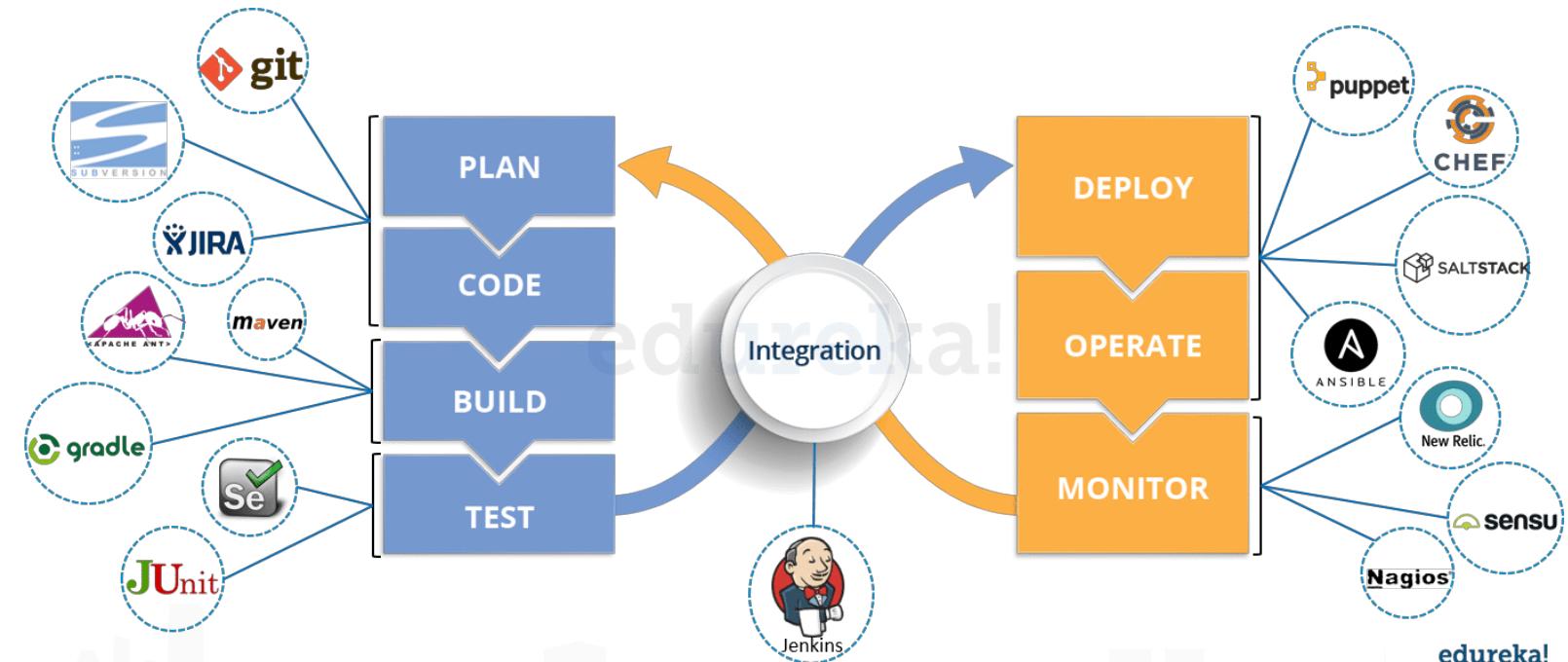
■ 지속적인 통합

- Continuous Integration
- 애자일 개발 방법의 하나로, eXtreme Programming 방법론으로 고안
 - 애자일 개발 중에서도 가장 기본이자 핵심 방법론
- 빌드, 테스트를 실시하는 프로세스를 상시 실시
- 가치 있는 소프트웨어를 고품질로 신속하게 전달하는 것을 목표
- 통합 작업을 미루지 말고 개발 중에라도 실시해서 소프트웨어 개발의 복잡성을 제거
- CI 장점
 - 비용 면에서의 이점
 - 시장 변화 속도
 - 속도와 품질 둘 다 확보

■ DevOps 도구

■ 지속적인 통합

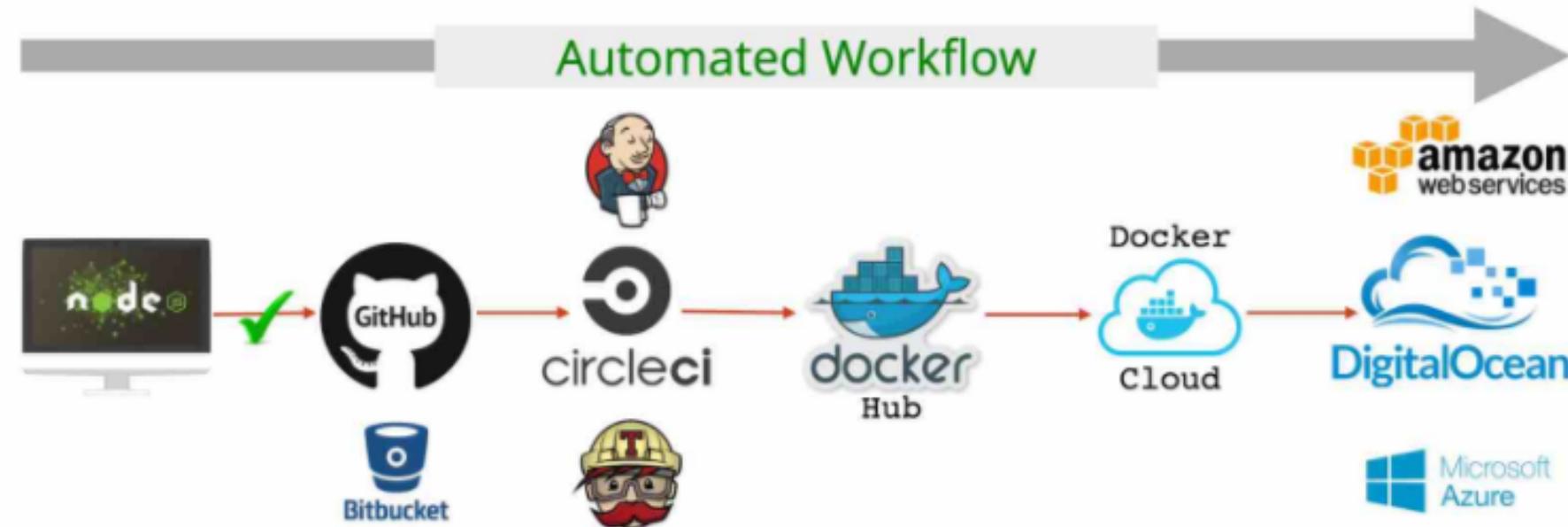
- 버전관리 시스템: Subversion, Git
- 빌드 도구: Make, Scons, Ant, Maven, Gradle, Rake
- 테스트 도구: TDD, BDD
 - Unit Test
 - Integration Test
 - User Acceptance Test
 - Regression Test
- CI 도구: Jenkins, Travis CI



■ DevOps 도구

■ 배포 자동화(CD)

- Continuous Deployment, Continuous Delivery



■ DevOps 도구

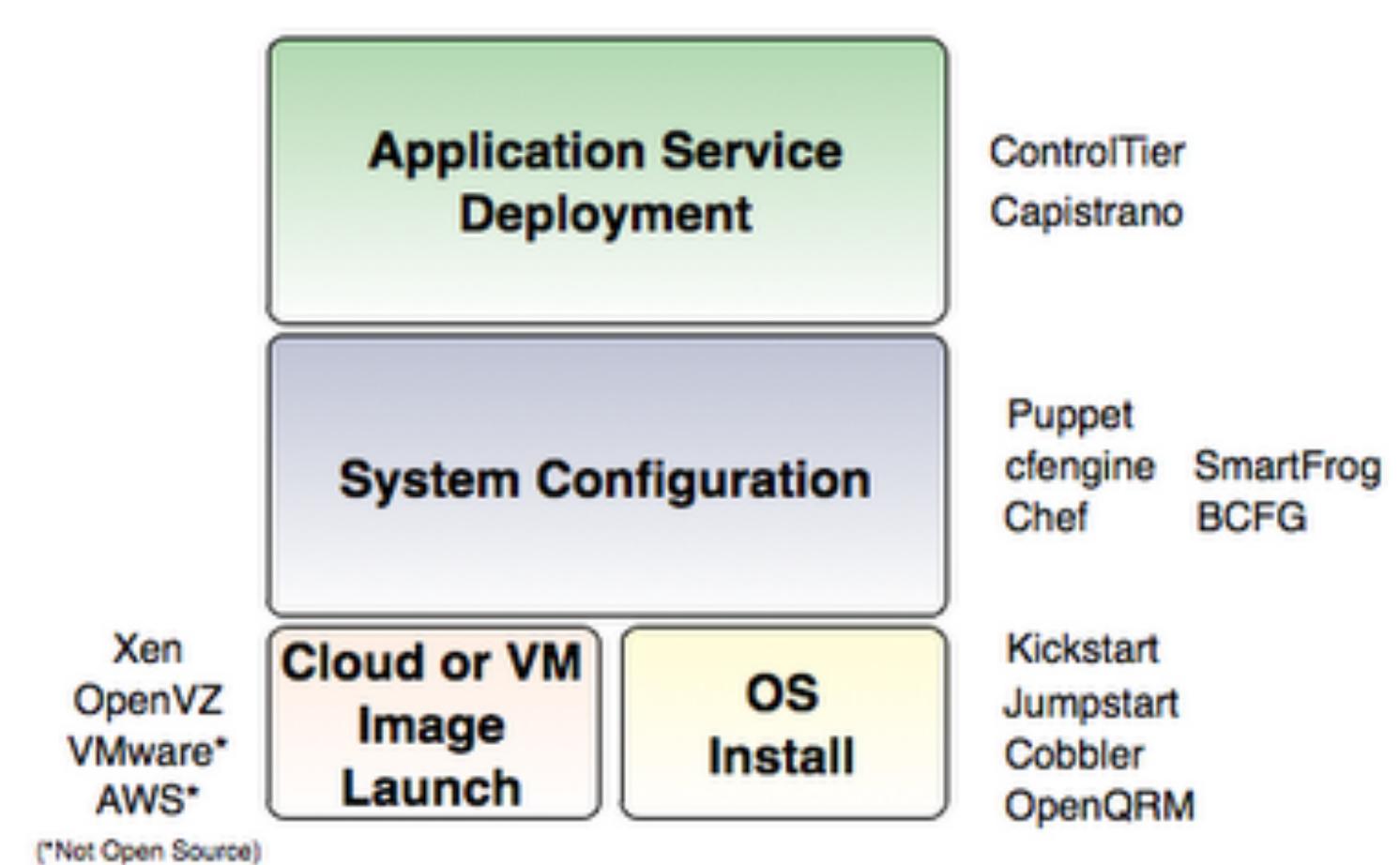
■ 배포 자동화(CD)

- 작은 단위로 나누어 자주 배포하면 위험 제어 가능
- 피드백을 빨리 받을 수 있으며, 조직을 안정화
- 배포 자동화
 - **개발부터 상용 환경**에 배포하기까지 일관된 처리를 필요
 - 모든 것을 버전 관리, **모든 환경을 같은 방법으로 구축**
 - 배포 작업을 자동화하고 사전 검증, 반복 테스트
 - 배포 파이프라인
 - 애플리케이션 빌드/배포/테스트/배포라는 일련의 프로세스를 자동화
 - 자동화를 통해서 배포 속도를 가속화, 정확도
 - 누구든지 배포

▪ DevOps 도구

▪ Provisioning Tool

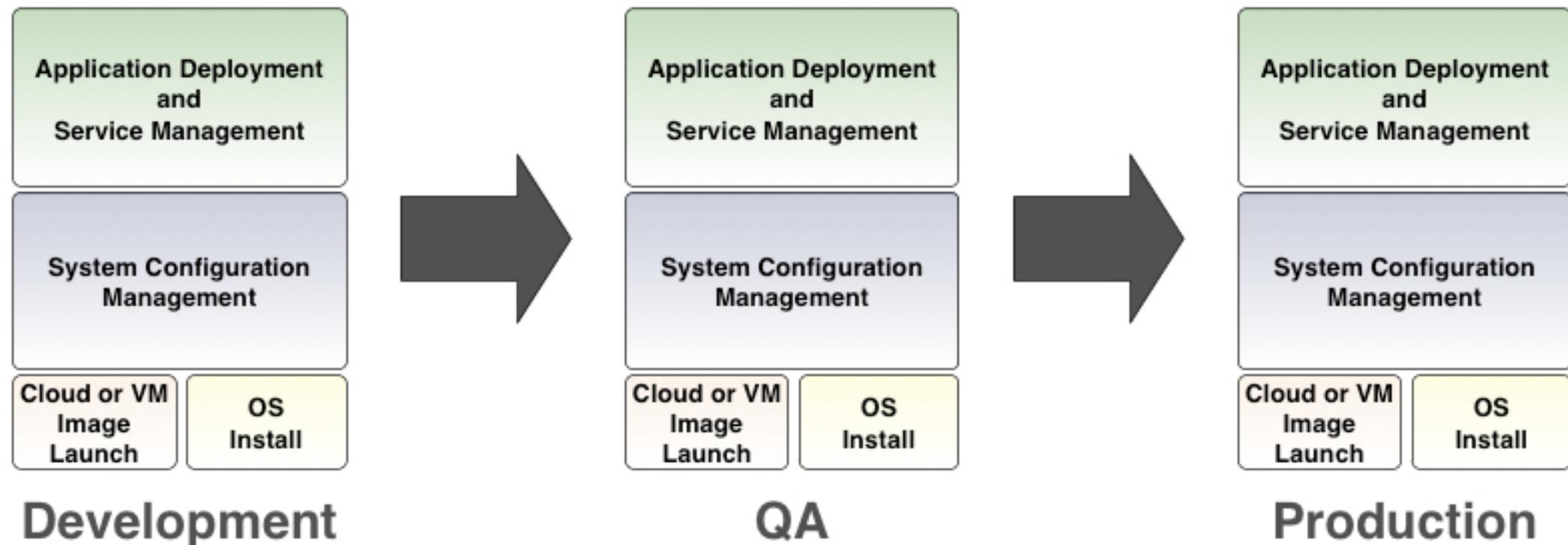
- Abstraction over Shell Commands
- Reusable
- Templating
- Collaboration
- Extensible



▪ DevOps 도구

▪ Provisioning Tool Chain

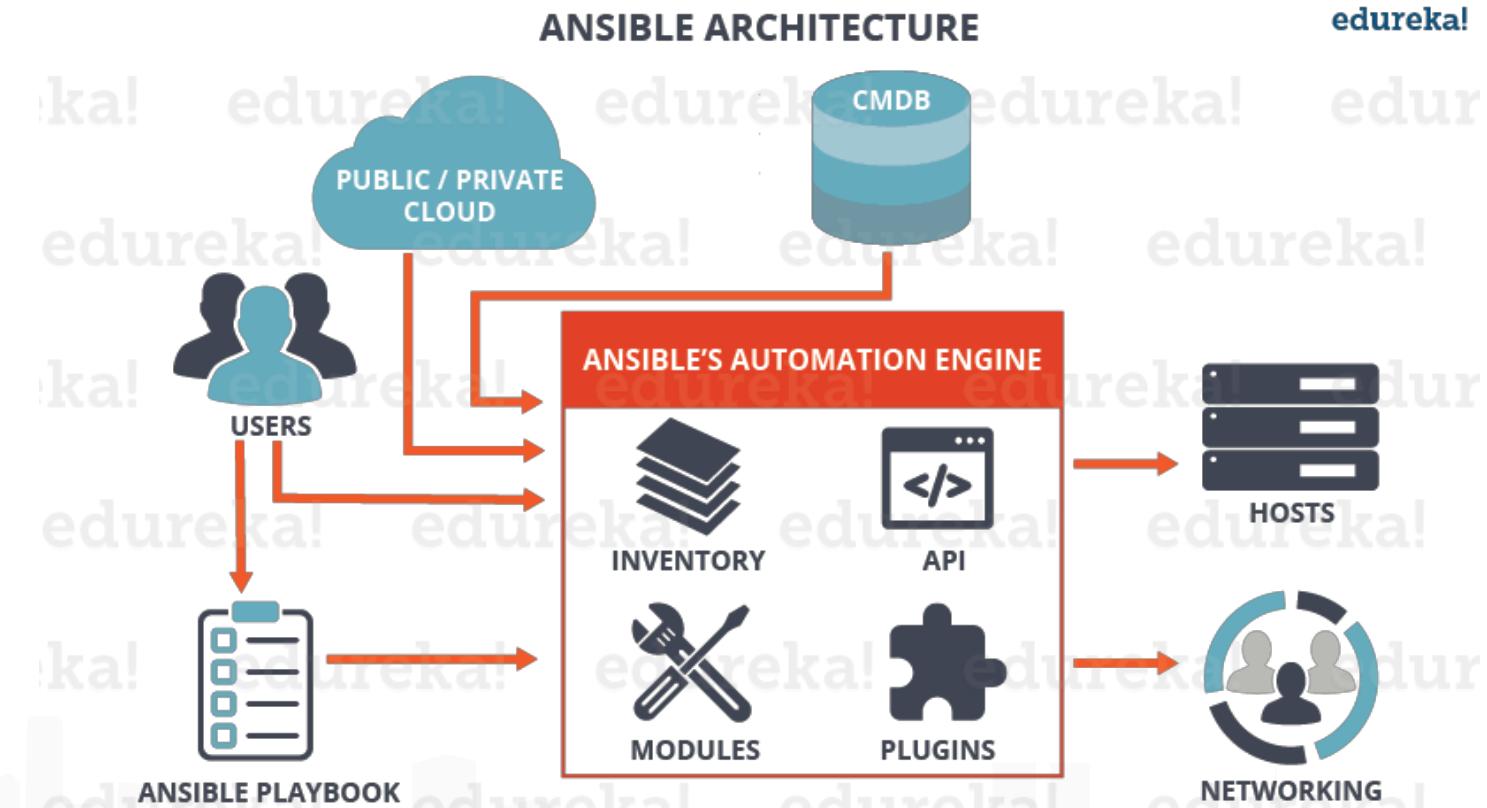
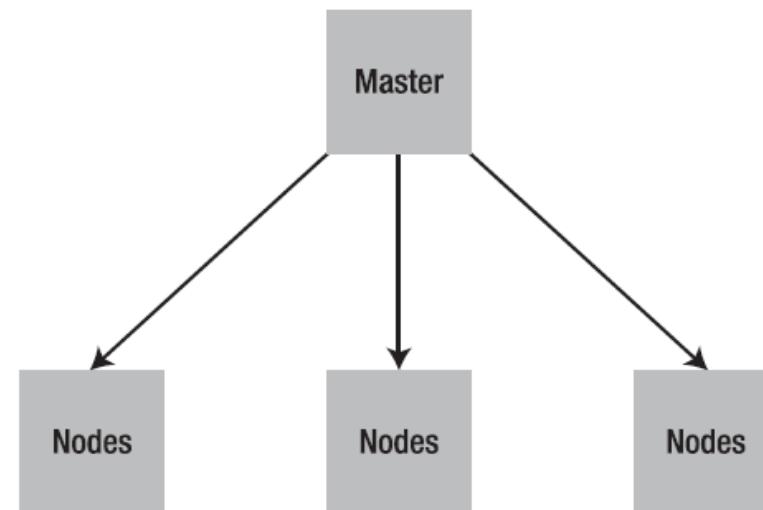
- Automated Provisioning + Automated Lifecycle



<http://dtosolutions.com/fully-automated-infrastructure/fully-%20automated-provisioning/>

▪ DevOps 도구

- Provisioning Tool Chain – Configuration Management
 - Puppet vs Chef vs Ansible vs Salt vs Cfengine vs Etc



■ DevOps 시작하기

■ *Microservice Team Structure*

- Two Pizza team
- Teams communicating through API contracts
- Develop, test and deploy each service independently
- Consumer Driven Contract



■ DevOps 시작하기

- 같은 목적을 향해 달려가고 있는가?
- 같은 프로세스를 사용하고 있는가?
- 같은 도구를 사용하고 있는가?

- Study → 도구부터 접근
- 시스템 설정은 Puppet
- VM OS 설치는 Vagrant or Docker
- 모든 설정 파일/스크립트는 VCS
- 공통의 배포 프로그램, Workflow 프로그램 사용
- QA/Staging 환경을 함께 구축
- Dev + Ops 공통 사용. 최대한 실제 환경과 비슷하게 구현