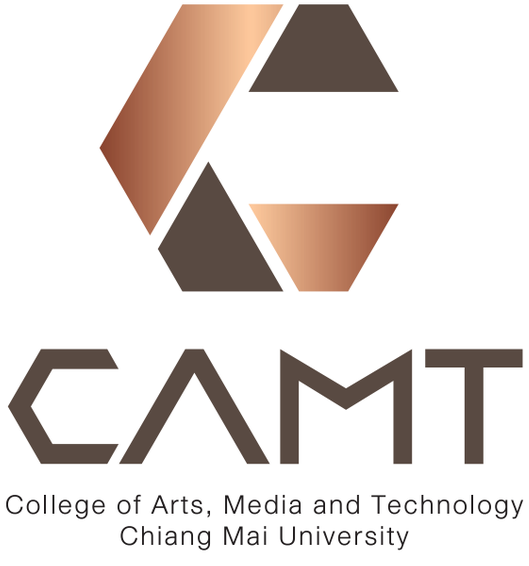
SE 234 Basic Development and Operations

#1 Basic DevOps



Lect Passakorn Phannachitta, D.Eng.

passakorn.p@cmu.ac.th

College of Arts, Media and Technology

Chiang Mai University, Chiangmai, Thailand

Agenda

- What is DevOps
- History
- Principles
- CI & CD overviews

What is DevOps — In short

- DevOps is not a **product**, **software** or a **standard**
- It is more closely related to **culture** or **practice**

Dev Ops = Dev + Ops

- A longer definition
  - DevOps is a set of **practices** intended to **reduce** the time between **committing a change to a system** and the **change being placed into normal production**, while ensuring **high quality**.

(Wikipedia)

Dev Ops = Dev + Ops

- An even longer definition
  - DevOps is the **combination of cultural philosophies, practices, and tools** that **increases an organization's ability to deliver applications and services at high velocity**: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.

    (Amazon web service — AWS)

Cultural philosophies, practices, and tools

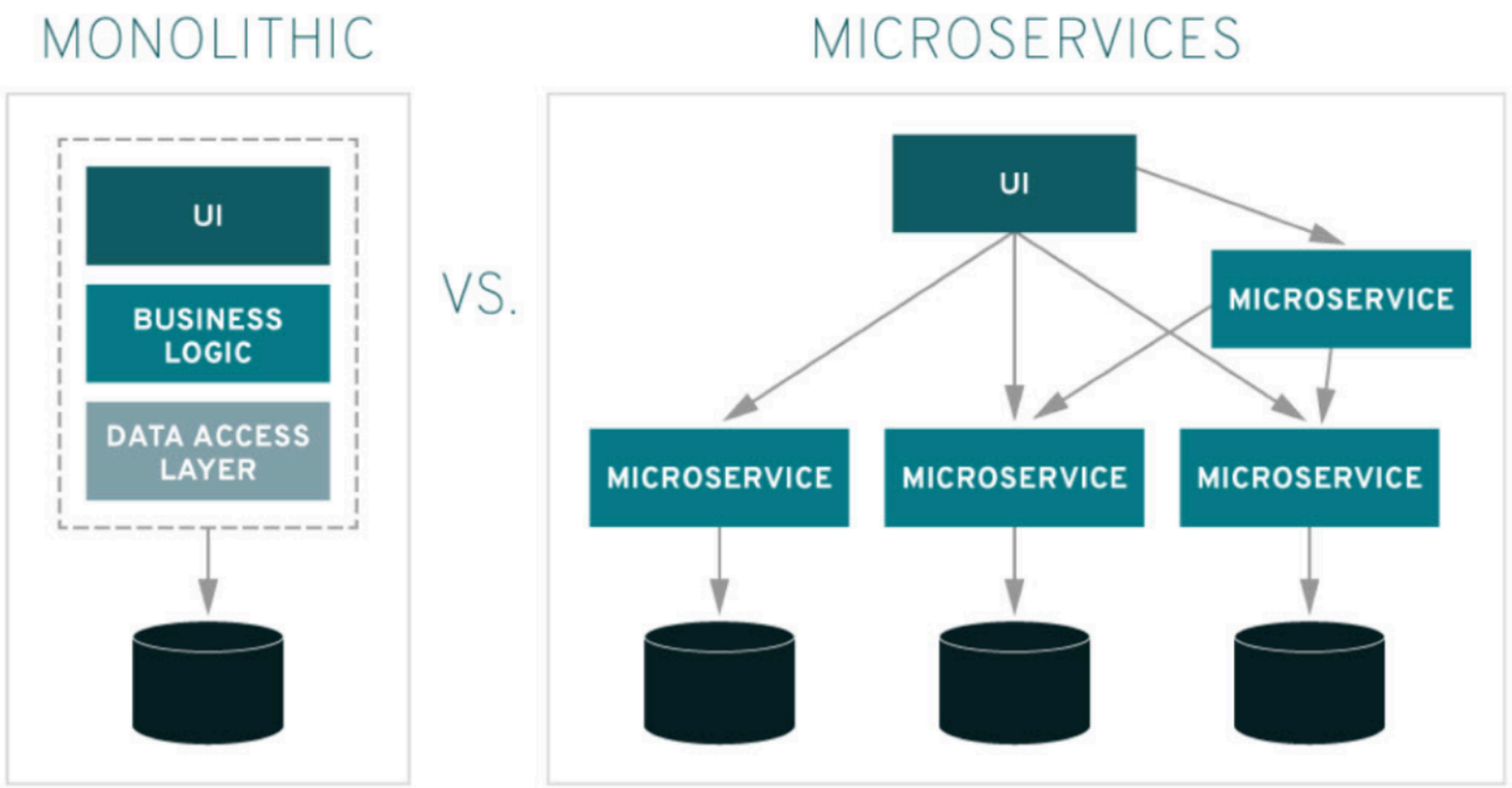
- Strike for
  - minimizing product cycle time
  - minimizing cost
  - maximizing product quality

- Automation toolsets are the essential component facilitating these optimization

Cultural philosophies, practices, and tools

	1970s–1980s	1990s	2000s–Present
Era	Mainframes	Client/Server	Commoditization and Cloud
Representative technology of era	COBOL, DB2 on MVS, etc.	C++, Oracle, Solaris, etc.	Java, MySQL, Red Hat, Ruby on Rails, PHP, etc.
Cycle time	1–5 years	3–12 months	2–12 weeks
Cost	$1M–$100M	$100k–$10M	$10k–$1M
At risk	The whole company	A product line or division	A product feature
Cost of failure	Bankruptcy, sell the company, massive layoffs	Revenue miss, CIO's job	Negligible

Cultural philosophies, practices, and tools



However, it was never easy in practice

- It is said that *Devs are from Venus. Ops are from Mars*

Especially for a typical enterprise

Company	Deploy Frequency	Deploy Lead Time	Reliability	Customer Responsiveness
Amazon	23,000 / day	Minutes	High	High
Google	5,500 / day	Minutes	High	High
Netflix	500 / day	Minutes	High	High
Facebook	1 / day	Hours	High	High
Twitter	3 / week	Hours	High	High
Typical Enterprise	Once every 9 months	Month or Quarters	Low/Med	Low/Med

Note: Lead time is a critical metric

- E.g., measurable time between work begins and ends

- Enterprises' Maturity levels can be defined by Lead time:
  - Less than one day (Elite)
  - Between one day and one week (High)
  - Between one week and one month (Medium)
  - Between one month and size months (Low)

(The state of DevOps report, 2019)

Why was it never easy in practice?

Company	Deploy Frequency	Deploy Lead Time	Reliability	Customer Responsiveness
Amazon	23,000 / day	Minutes	High	High
Google	5,500 / day	Minutes	High	High
Netflix	500 / day	Minutes	High	High
Facebook	1 / day	Hours	High	High
Twitter	3 / week	Hours	High	High
Typical Enterprise	Once every 9 months	Month or Quarters	Low/Med	Low/Med

Due to the different objectives

- Developers wants to provide as many killing features as possible to make to product always at the market's leading position.
- Operations wants the product to be the most stable, reliable and safe. So, they don't want many fancy features appearing too quickly.

These two objectives are difficult be satisfied at the same time.

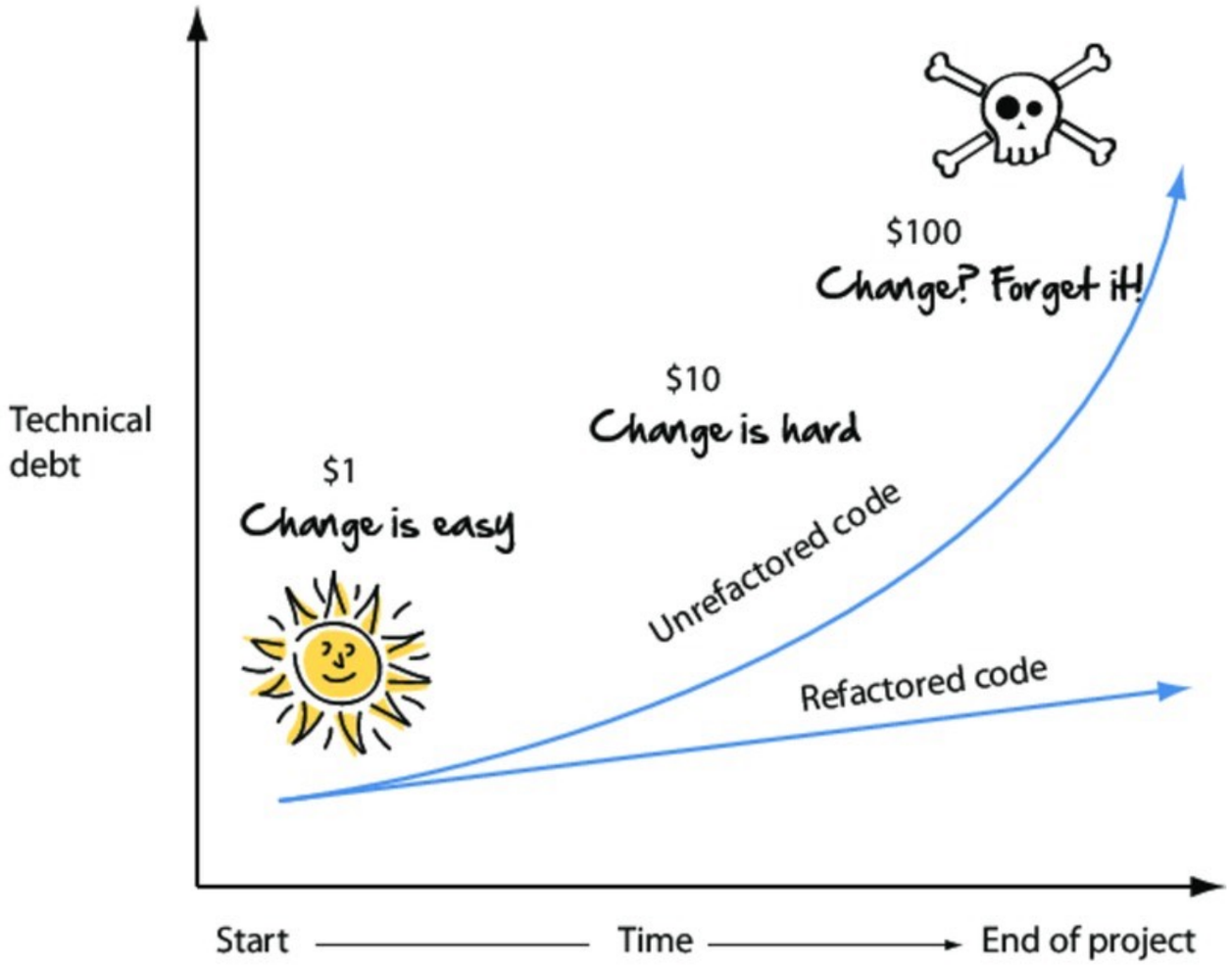
Available options in the past when project is growing

- Spend a lengthy amount of time on testing until Ops have a full confidence that the product has reached its maximum stability, reliability, and security;
- Or, only deliberately test on what Ops do insist that the specific module requires a detailed test.

Most of typical enterprises chose the second options

This introduce the Technical debt problem

- Still remember?



Thus, it is extremely hard to **consistently** hit the market with high quality product

Stated in The state of DevOps report (2016)

- Typical enterprises with DevOps are able to
  - Deploy software faster;
  - Address and fix technical problems faster;
  - Failure rate due to changes is smaller;
  - Time spent of developments is shorter;
  - Security problems are lessen;
  - Employees have more free time;
  - Overall costs are reduced.

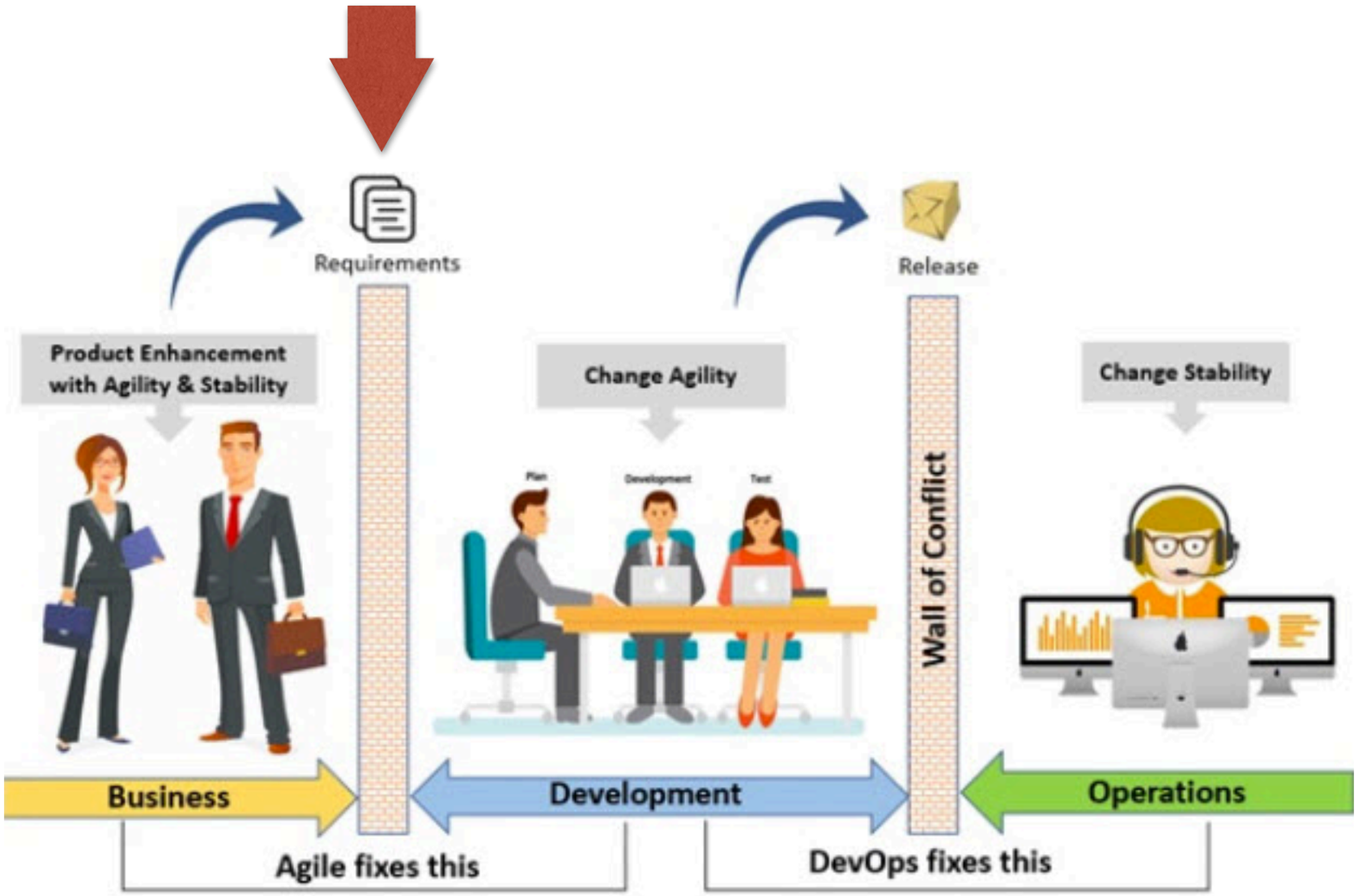
The convergence of DevOps

- Lean movement (1)
  - Started in 1980s to codify Toyota production system with the techniques such as Value Stream Mapping, Kanban boards, and Total Productive Maintenance.
  - Lead time, i.e., the time required to convert raw materials into finished goods, was the best predictor of quality, customer satisfaction, and employee happiness
  - Lean principles focus on **creating value for the customer**—thinking systematically, creating constancy of purpose, embracing scientific thinking, creating flow and pull (versus push), assuring quality at the source, leading with humility, and respecting every individual.

The convergence of DevOps

- Agile movement (2)
  - In 2001, Agile Manifesto was created.
  - The key principle was to **deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale**.
  - Other principles focus on the need for **small, self-motivated teams**, **working** in a **high-trust management model**.

Agile fixes the wall of conflicts between business and devs



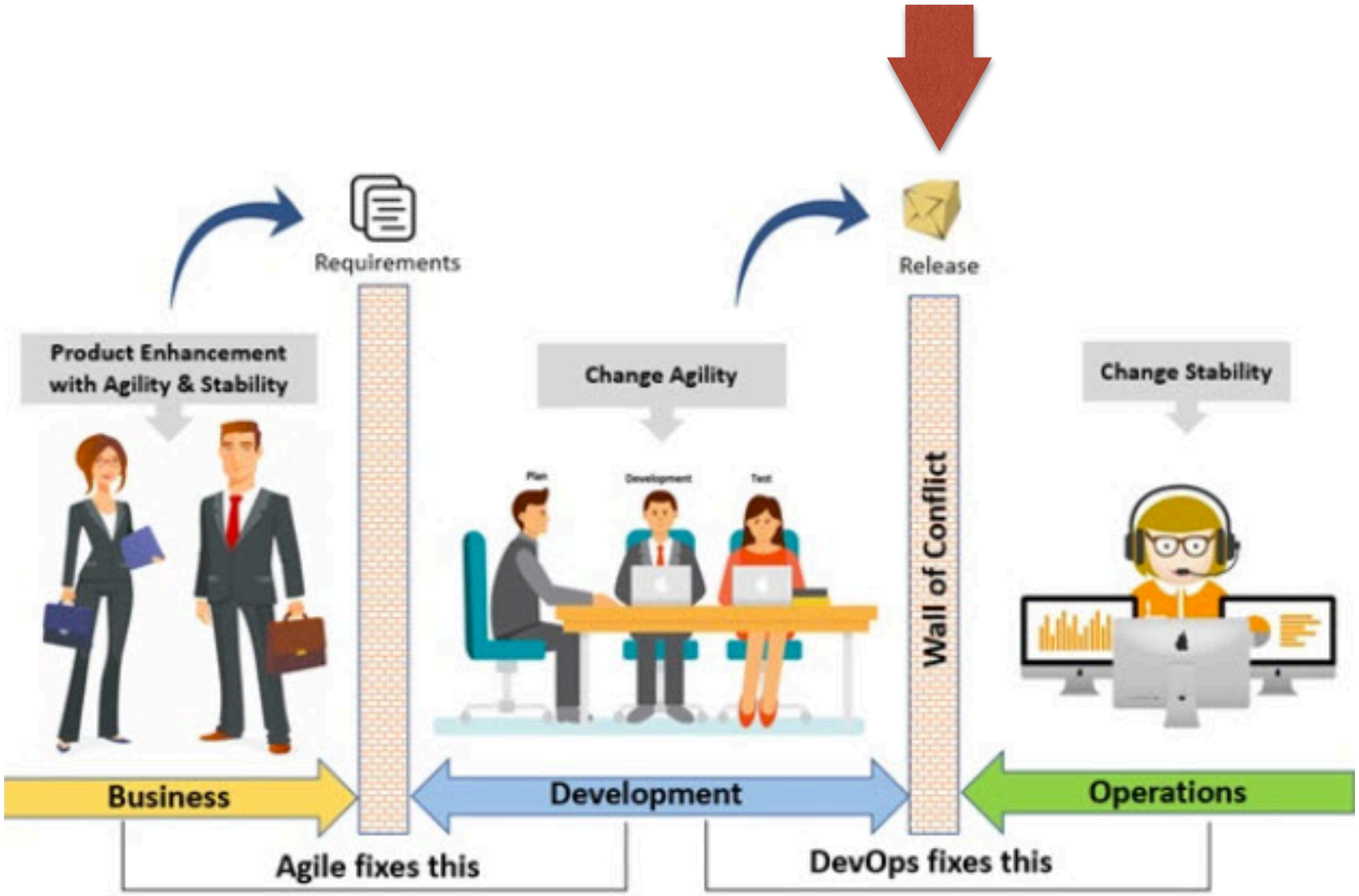
The convergence of DevOps

- The Continuous delivery movement (3)
  - Around the Agile 2006 conference, Tim Fitz posted a blog post titled **Continuous Deployment**.
  - The post motivated Jez Humble and David Farley to develop a **deployment pipeline** (practice?) that ensures the code and infrastructure are **always in a deployable state** and that all code checked in to a repository is directly deployed into production."
  - This is later an essential building block of DevOps.

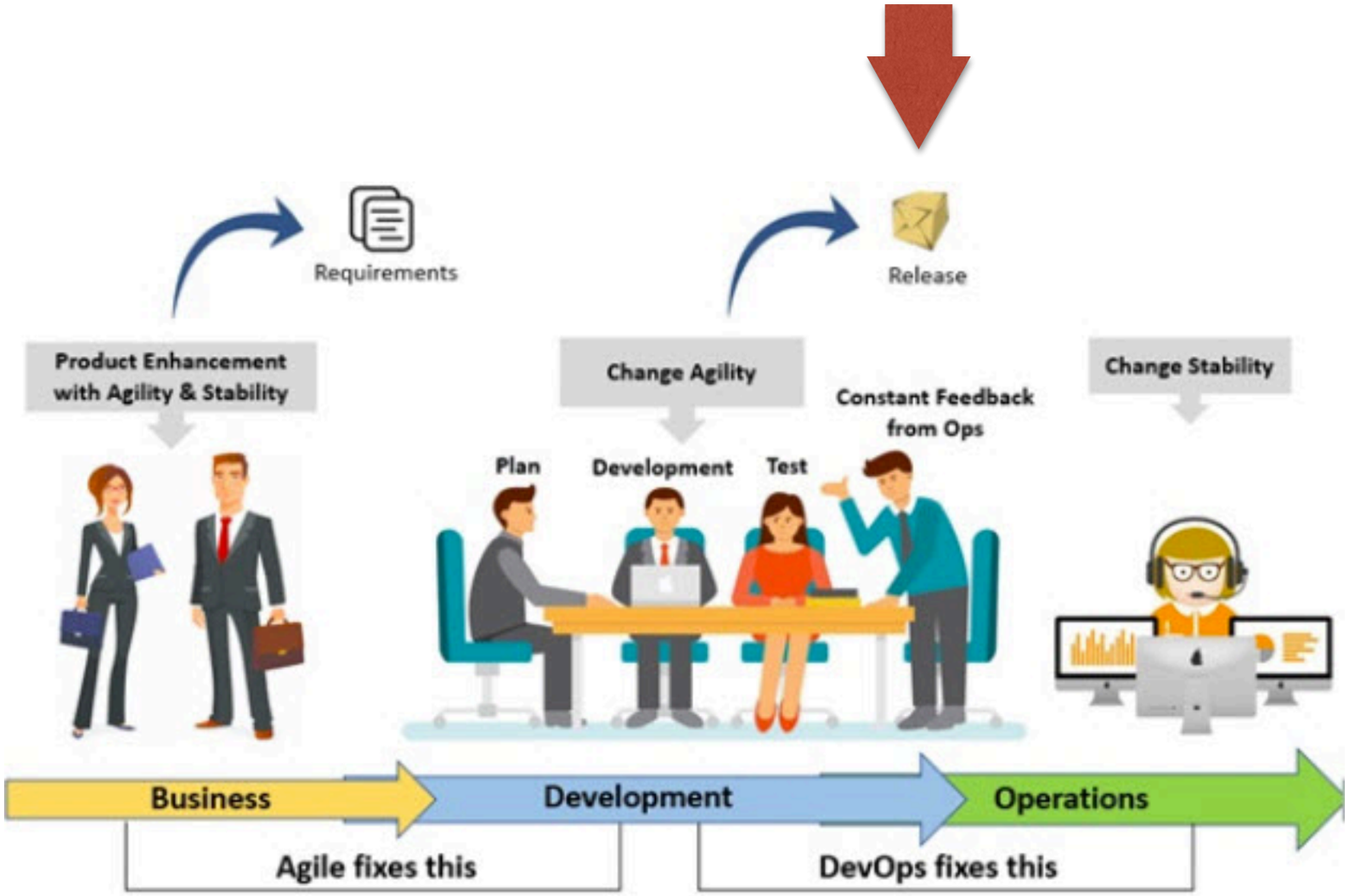
The convergence of DevOps

- The Velocity conference movement (4)
  - In 2009, John Allspaw and Paul Hammond gave a talk entitled “10 Deploys per Day: Dev and Ops Cooperation at Flickr.”, and this enlightened many other ITs on how Dev and Ops can properly coordinate.
  - Soon later in the same year, the word DevOps was coined in a succeeding events named DevOpsDays.

Another wall of conflicts >> Devs and Ops



Fix by DevOps

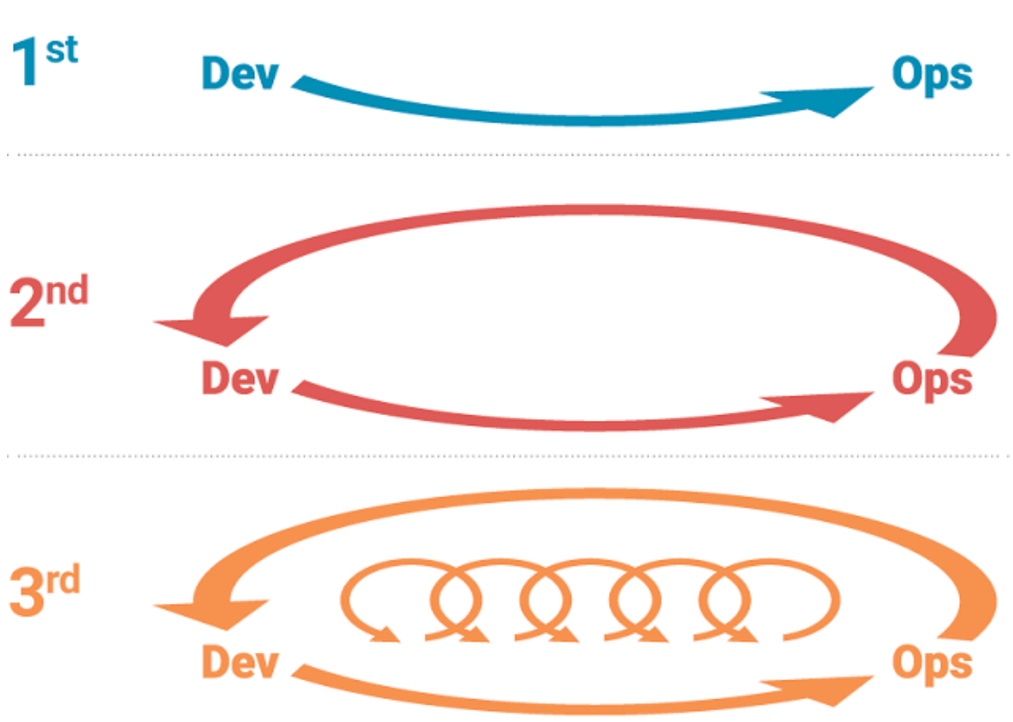


The convergence of DevOps

- The Toyota Kata movement (5)
  - After years, Toyota JIT production and Lean principles were revisited in 2009 based on a more than 20 years of experience of Mike Rother.
  - The conclusion of the study was that the Lean community **missed the most important practice**. The critical factor in Toyota successfully did was to make improvement work habitual, and build it into the daily work of everyone in the organization.

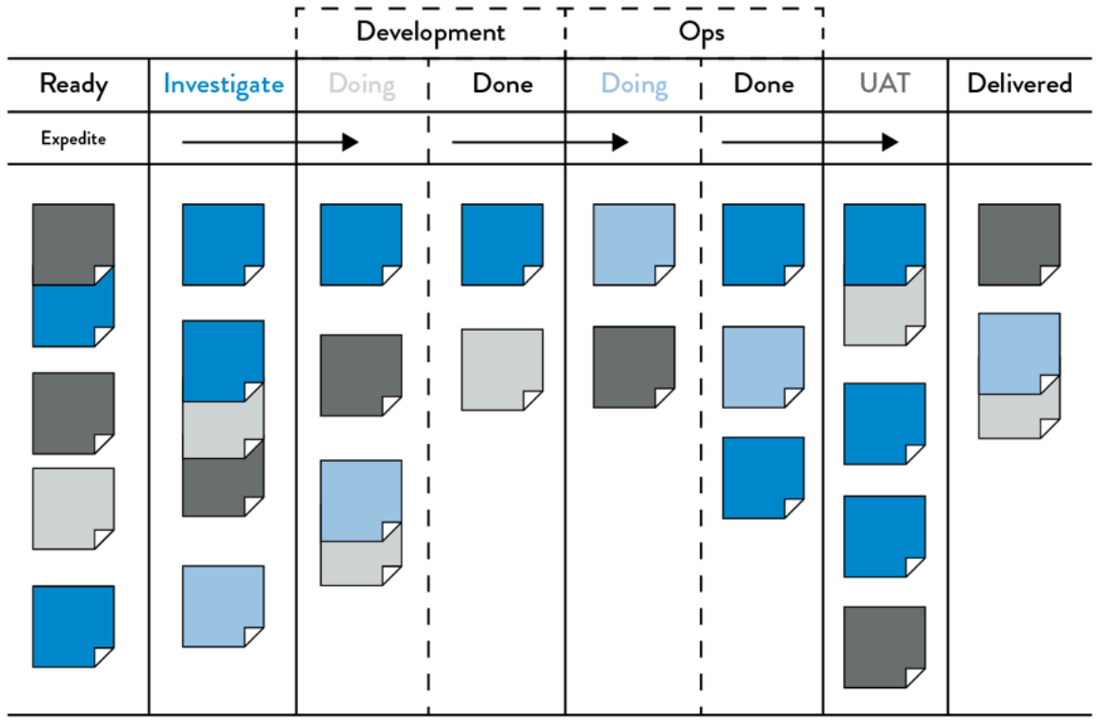
Derived to three DevOps principles

- Flow
- Feedback
- Continual learning and experimentation

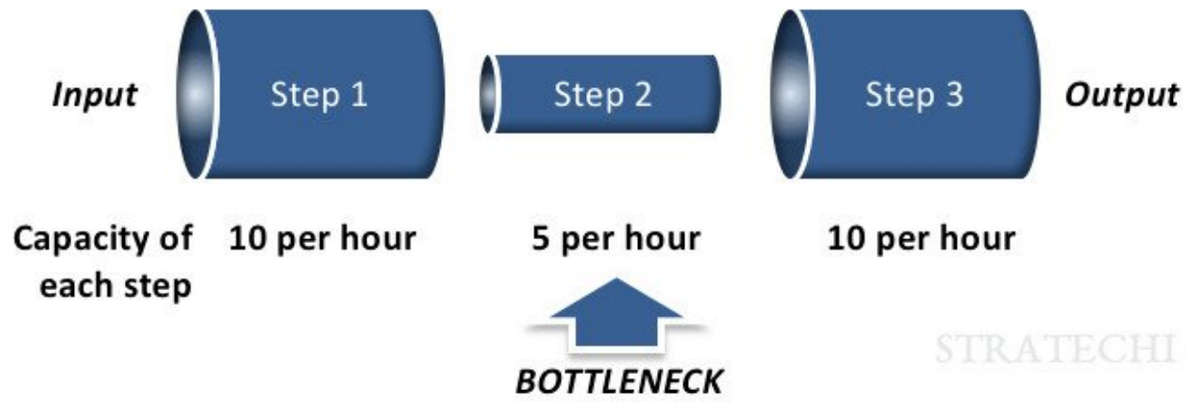


Flow

- The principles of Flow — accelerate the delivery of work from Development to Operations to customers.
  - Make everything transparent and visible
  - Limit the #works in process
  - Make small batches
  - Continually identify and elevate constraints
  - Eliminate possible waste



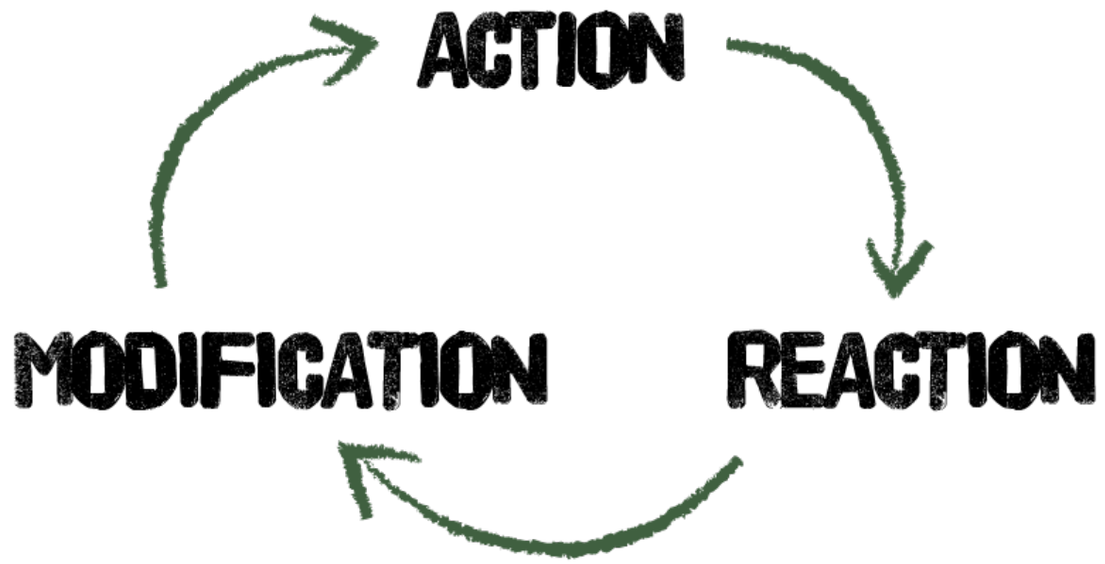
Flow


Input
Step 1
Step 2
Step 3
Output
Capacity of each step
10 per hour
5 per hour
10 per hour
BOTTLENECK
STRATECHI


LARGE BATCHES
F1 F2 F3 F4 F5 I1 I2 I3 I4 I5 Se1 Se2 Se3 Se4 Se5 St1 St2 St3 St4 St5
WAITING
First product ready
SINGLE PIECE FLOW
F1 I1 Se1 St1 F2 I2 Se2 St2 F3 I3 Se3 St3 F4 I4 Se4 St4 F5 I5 Se5 St5
WAITING
First product ready

Feedback

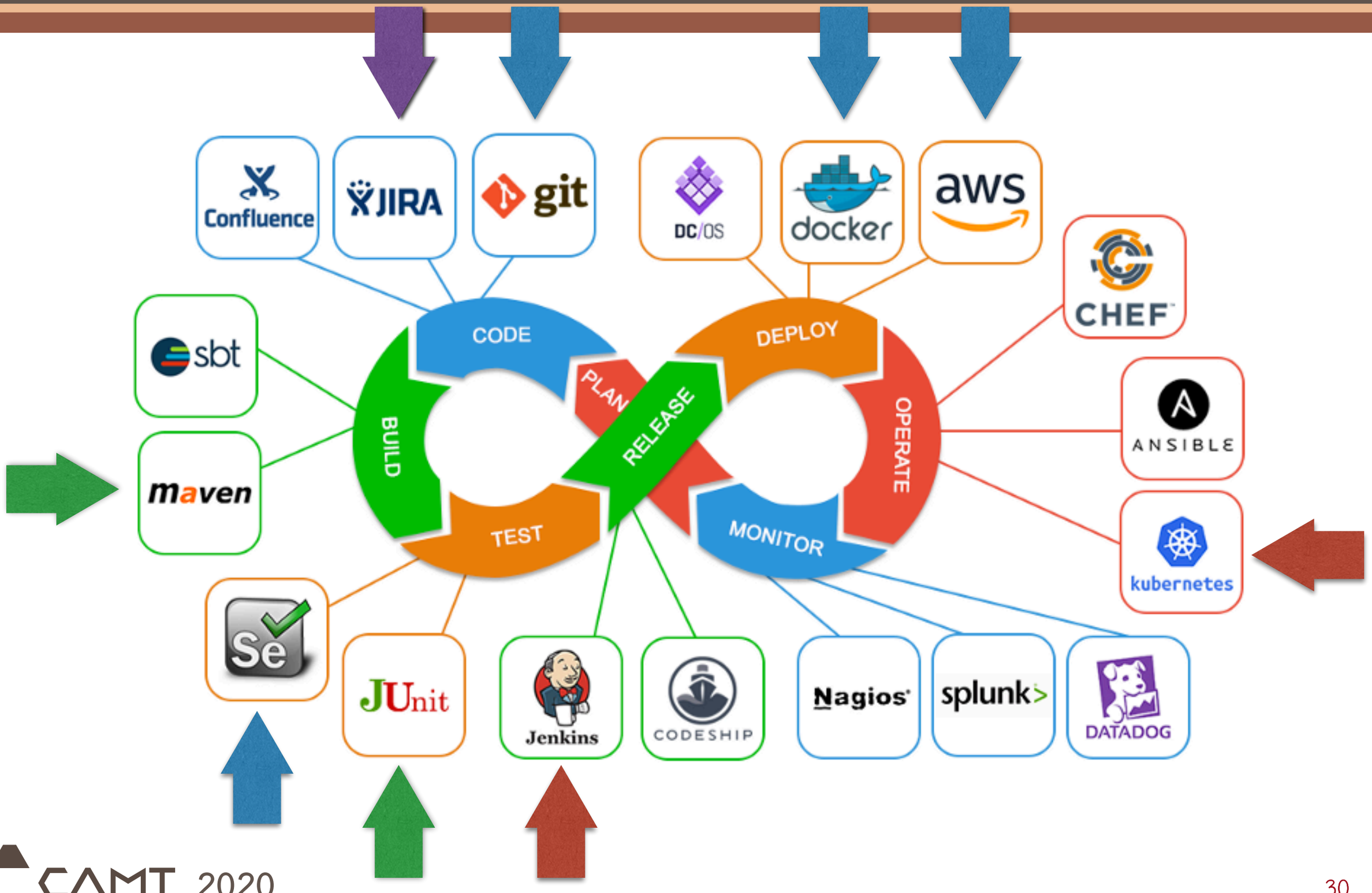
- The principles of Feedback — enable the team to create safer systems of work.
  - Feedback can make everyone improve and similar problems are less likely to occur again.
  - Keep ensuring the quality at the source



Feedback

- The principles of Continual Learning and Experimentation — foster a high-trust culture and a scientific approach to organizational improvement risk-taking as part of our daily work.
- Possibility to improve the daily work routine.
- Possibility to transform local deliveries to global improvements.
- Fail fast and win big

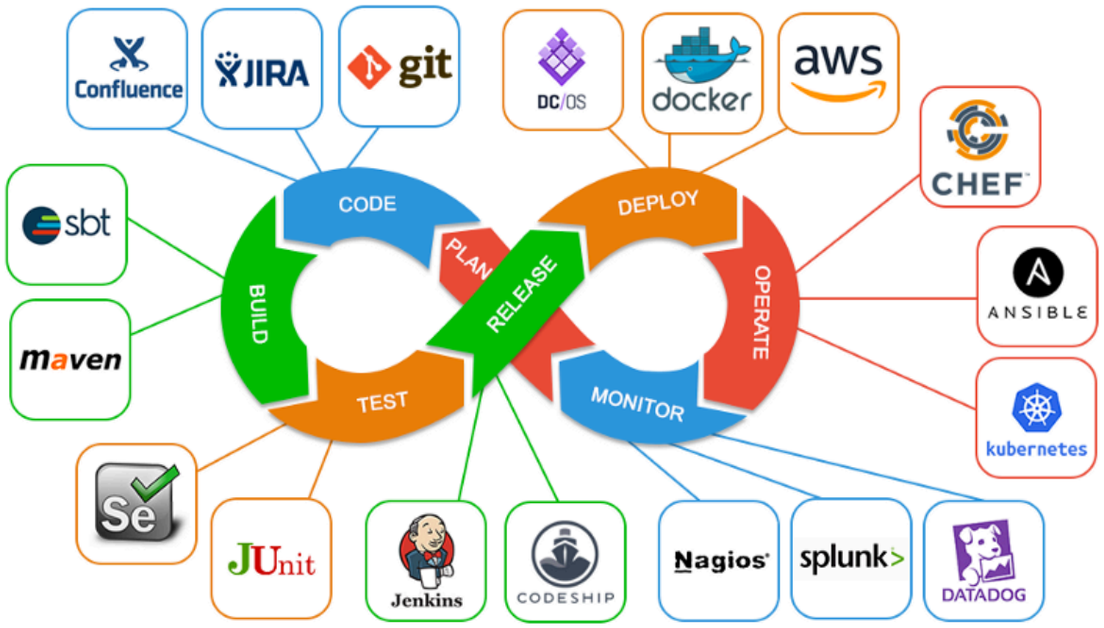
In software development

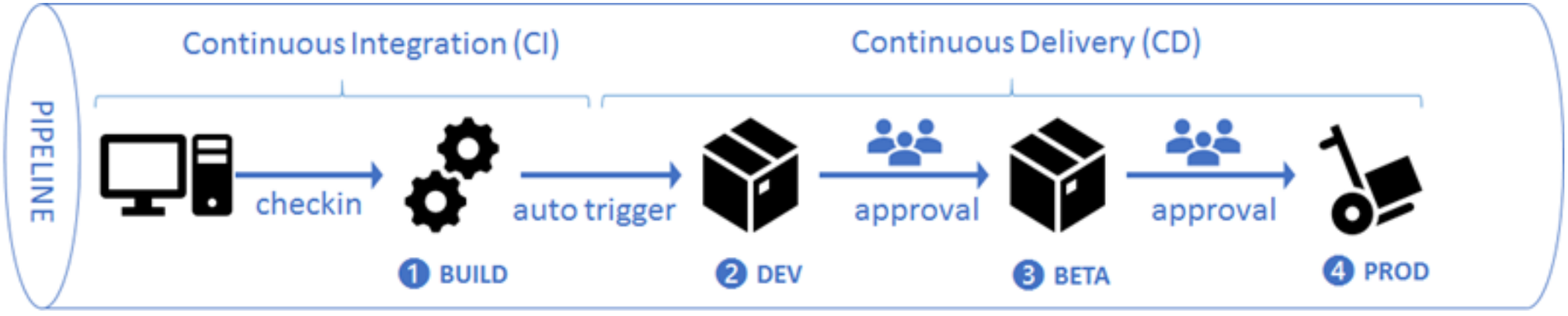


As the project continuously grows

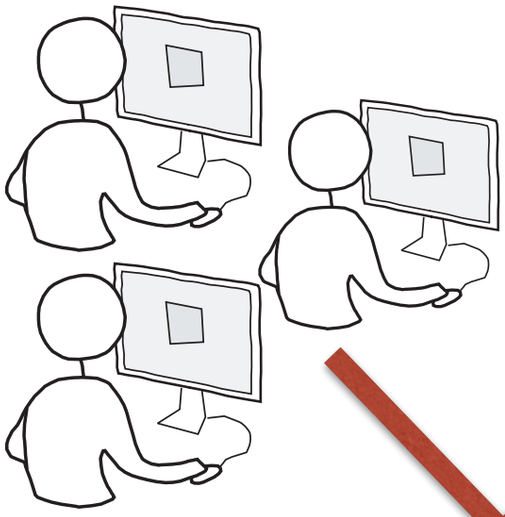
- The effort required to integrate, build, and deploy may increase exponentially with
  - The number of components.
  - The number of defects.
  - Time since the previous integration.

CI & CD — aiming at minimizing Lead time

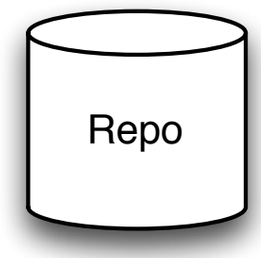




Development without CI & CD



Infrequent commits

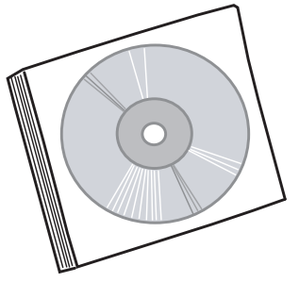


Late and difficult integration

Lots of Bugs

Testing happens late

Infrequent releases



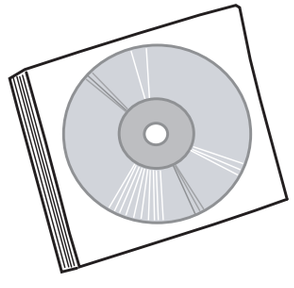
Development without CI & CD

- Insufficient testing
- Issue raised later are harder to fix
- Slow release process

Lots of Bugs

- Integration hell
- Project delays
- Unhappy clients
- Higher maintenance costs

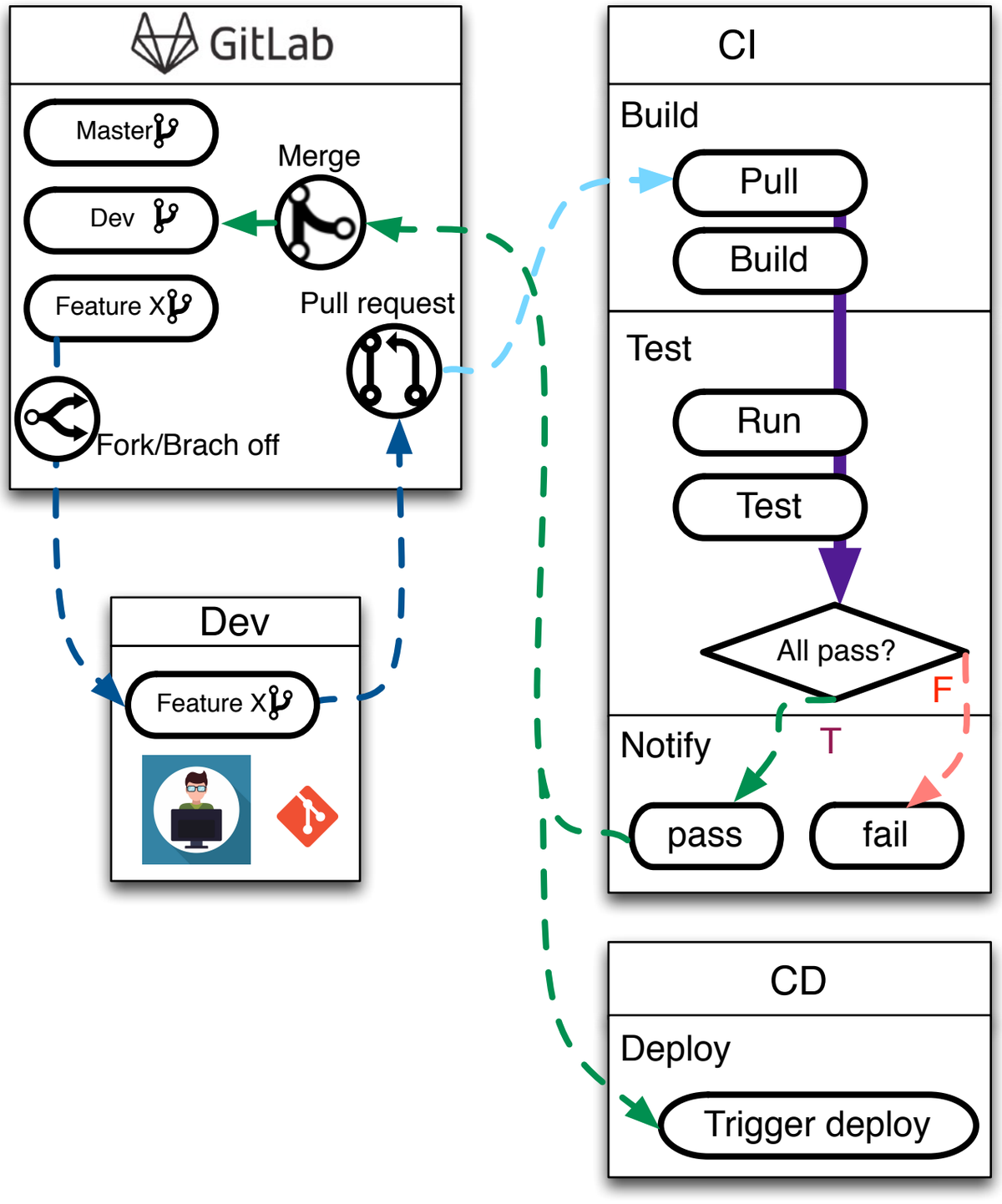
Infrequent releases



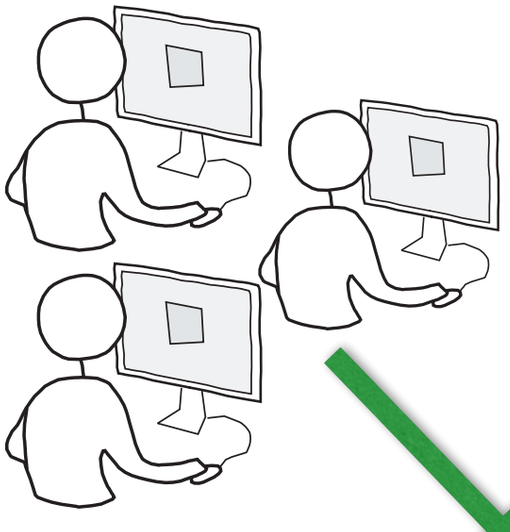
Starter kits

- A source code repository, e.g., Git
- An automated build, i.e., build scripts
- An automated testing suite
- An automated code quality measurement
- A continuous build service or server.
- A deployment and release helper

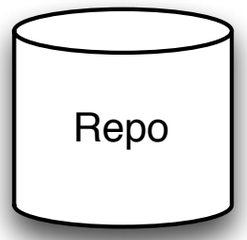
Some example flow



Development with CI & CD



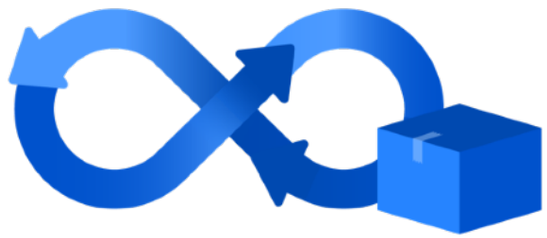
Regularly commits



Automated build

Testing is automated and happens early and often

Fewer bugs



Regular and automated releases

Core Practices

- **Single source repository**
  - Everyone's codes are stored in the same place
  - Single point of truth
  - Ownership is shared

Core Practices

- **Automate Build**
  - Using IDE to build is not automating
    - Eliminate the work-in-my-local problem
      - E.g., file and environment variable is not in the same structure for the entire team.
  - Use a build tool to compile, package, and test

Core Practices

- **Automate Testing**
  - Let the failed test fail the build — Avoid technical debt
  - Integrate not only unit test but also integration test.
  - See the failed test as an urgent task.
    - Fix it or you will even waste more time

Core Practices

- **Publish the Latest Distributable**
  - The current state of product will be the same as what is deployable and distributable.
  - Only built the deployable only if it passed the test.

Core Practices

- **Commit More Often**
  - We will be notified early that our code is not going to work.
  - Aim for at least once an hour
    - Practice task separation and prioritization
  - This will make you update frequently as well.

Core Practices

- **Build every Commit**
  - To get the fastest feedback possible
  - Automate build will help and enable you this practice
  - Less to merge and fix each time.

Core Practices

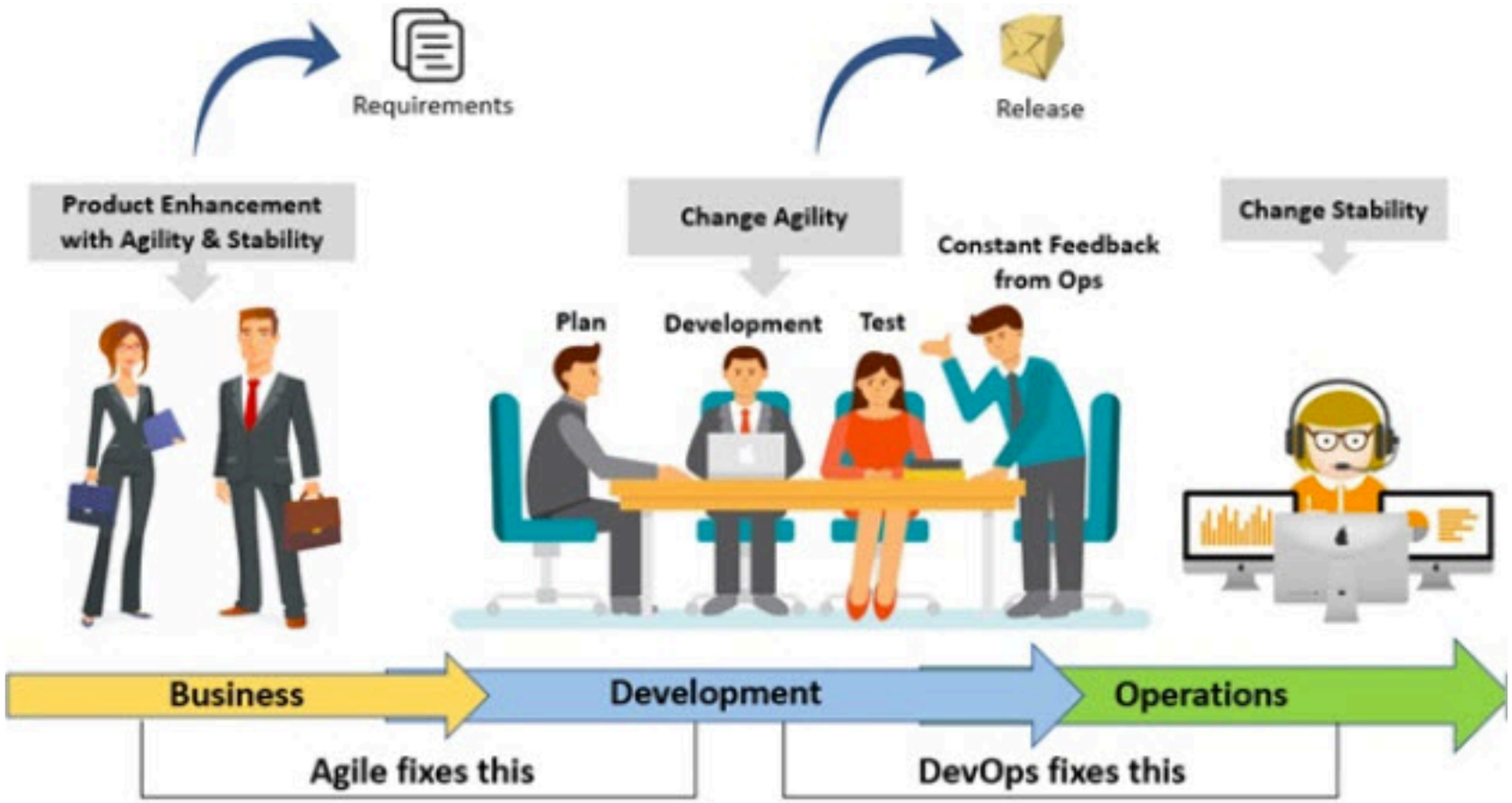
- **Test is also applied to the production clone**
  - Multithreaded issues, e.g., race conditions, can also be captured
  - Database and other system architectures can also be tested.

Core Practices

- **Everyone sees what is happening**
  - Reduce time to fix
  - Promote team work

Summary

- DevOps is not a **product**, **software** or a **standard**
- It is more closely related to **culture** or **practice**



Discussion Time