KAIST

Spring 2025

CS30500:
Introduction to Software Engineering

Chapter 3. Agility and Process 2/2

Prof. In-Young Ko

School of Computing

Last Class

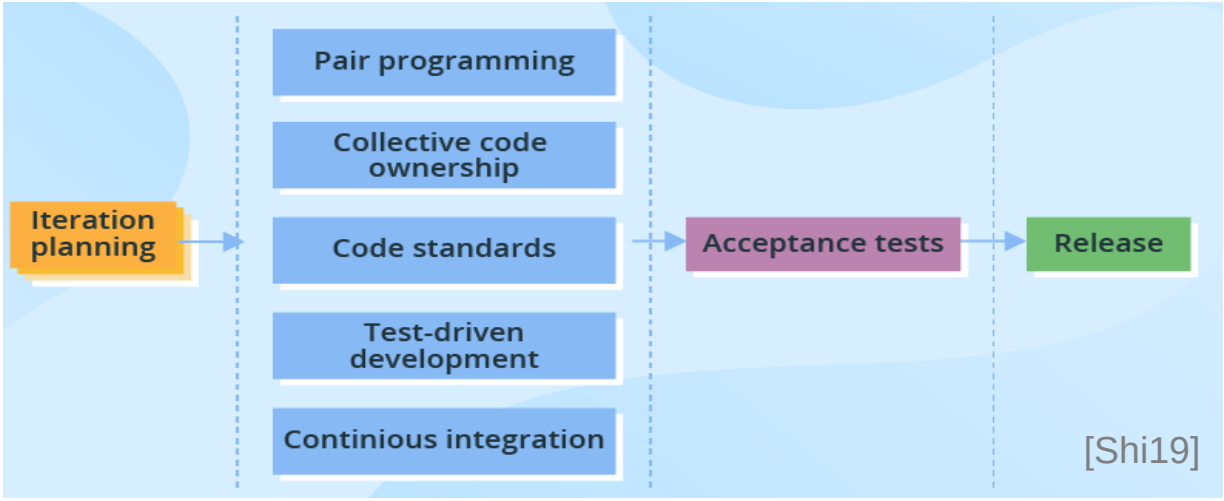
- Agility
- Popular agile methods
  - Scrum
  - Extreme Programming (XP)
  - Kanban

Today's Plan

- Popular agile methods
  - Scrum
  - Extreme Programming (XP)
  - Kanban
- Comparison among process models
- DevOps

What is Extreme Programming (XP)?

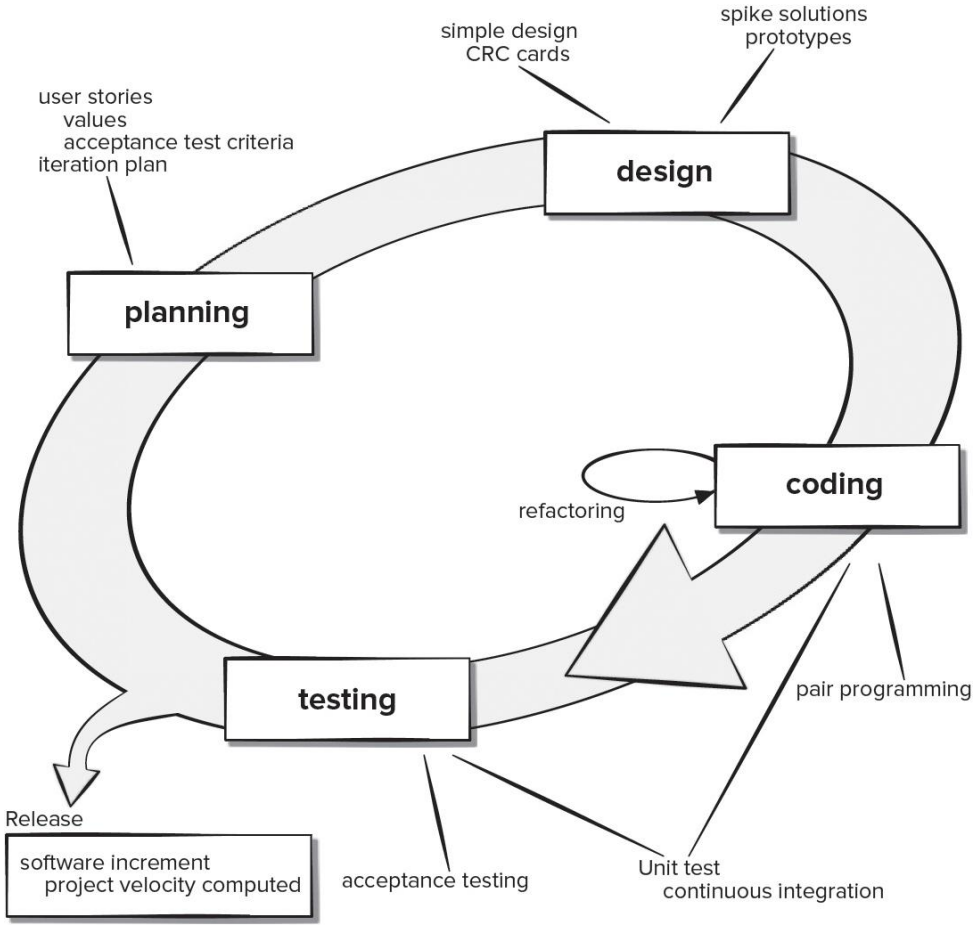
- An iterative process with a set of rules and practices to deal with changing requirements
- Developed by Kent Beck in the late 1990s
- An iteration
  - An iteration of framework activities (planning, design, coding and testing)
  - 1-2 weeks long
  - Requirement changes can be made even after an iteration is launched
  - Use the following practices:
    - Pair programming
    - Test-driven development and test automation
    - Continuous integration (CI)
    - Small releases
    - Simple software design
    - Prescribes to follow the coding standards



XP Activities (1/2)

[PrMa20]

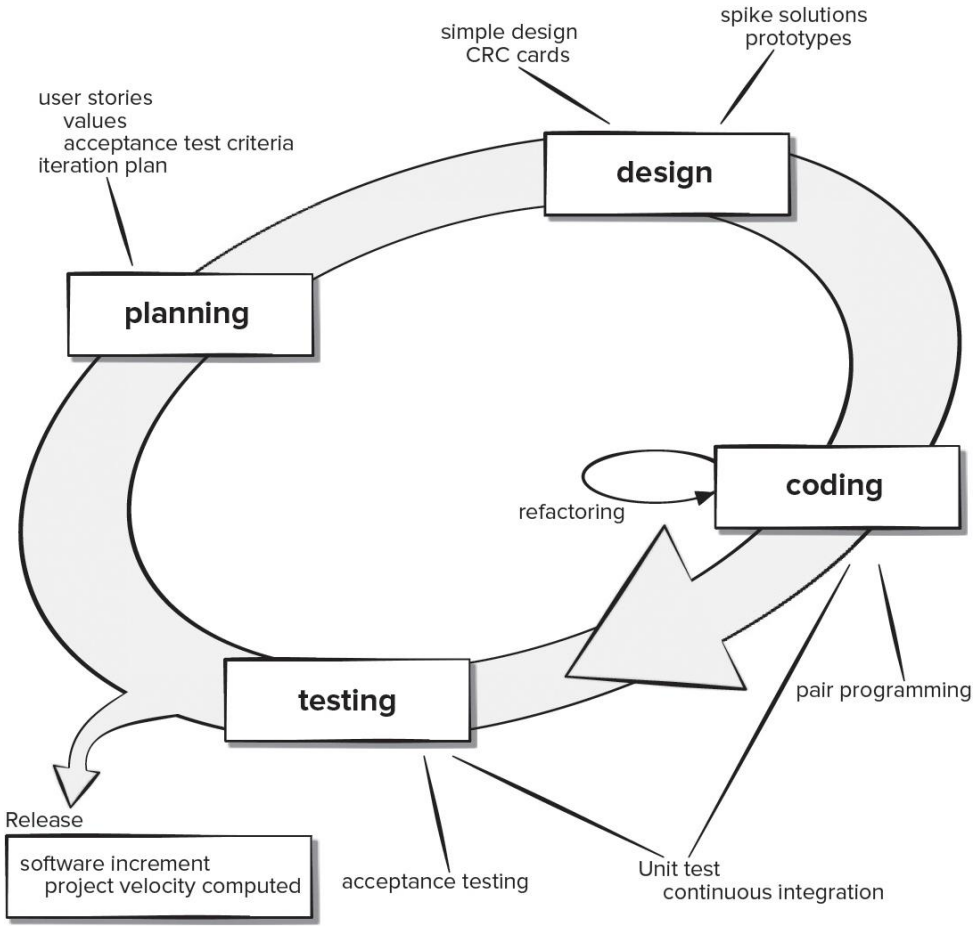
- Planning
  - *Listening* customer requirements and making *user stories*
  - Prioritizing the user stories by the customers
  - Assessing the cost (development weeks) of each story by the team
  - Group stories into the next *software increment* (release) by the team and the customers
- Design
  - Use *CRC (class-responsibility-collaborator) cards* to identify and organize object-oriented classes
  - Develop an *operational prototype* to understand and solve a problem → *refactoring* (modifying and optimizing) the code



XP Activities (2/2)

[PrMa20]

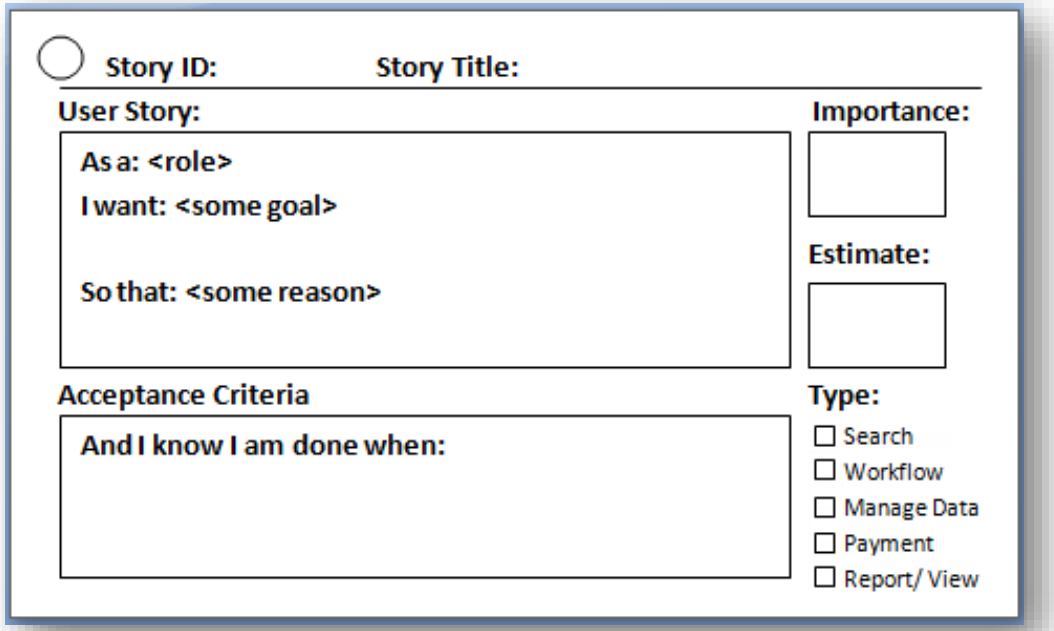
- Coding
  - Develop firstly a series of *unit tests* for testing each of the stories in the release → helps to understand what to implement
  - *Pair programming* – two people work together to implement each story and review code
  - *Continuous integration* of unit code
- Testing
  - Automated unit testing for *regression testing*
  - *Acceptance testing* to test overall system features



- Pair programming and collaboration example: https://www.youtube.com/watch?v=tPxGoE43r_k

User Stories – Kent Beck

- Basic unit of requirements in agile methodologies
- Represent a feature desired by the customer
- Written by the customer on index cards
- Use cases can be derived from them
- User Story Components:
  - Title – written in a present tense, active voice
  - Acceptance test: a method to test the story
  - Priority
  - Story points: estimated time to implement
  - Description – written in one to three sentences



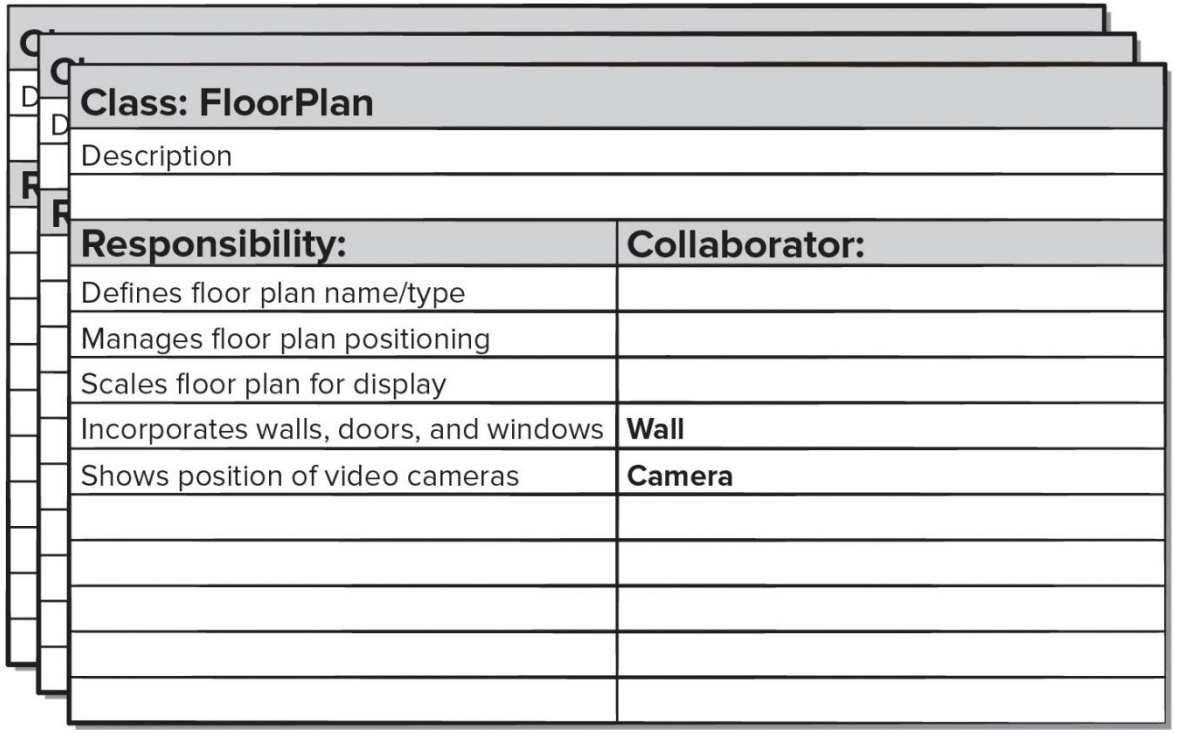
https://www.c-sharpcorner.com/article/what-is-user-story-in-agile-scrum/

Phillip A. Laplante, Requirements Engineering for Software and Systems, 3rd Ed., CRC Press, 2018

CRC Modeling

- ***Class-responsibility-collaborator*** *(CRC) modeling* provides a simple means for identifying and organizing the classes that are relevant to system or product requirements

- A CRC model is a collection of standard index cards that represent classes
- The cards are divided into three sections:
  1. Along the top of the card you write the name of the class
  2. List the class responsibilities on the left
  3. List the collaborators on the right



[PrMa20]

CRC Card Example

Communication Infrastructure	
Responsabilities	**Interactions**
Provides a reliable communication mechanism Provides additional services (e.g. naming service)	Resource Provision Service Resource Monitoring Service Parallel Programming Libraries Security Service Scheduling Service Access Agent

Resource Provision Service	
Responsabilities	**Interactions**
Services application execution requests Controls local resource usage Provides resource usage information	Communication Infrastructure Resource Monitoring Service Parallel Programming Libraries Security Service Scheduling Service Access Agent

Resource Monitoring Service	
Responsabilities	**Interactions**
Keeps resource availability and usage information Monitors resource providers	Resource Provision Service Communication Infrastructure Scheduling Service

Security Service	
Responsabilities	**Interactions**
Protects shared resources Manages user identities Previdies secure communication channels	Resource Provision Service Communication Infrastructure

Scheduling Service	
Responsabilities	**Interactions**
Manages Grid resources Schedules application execution requests	Resource Provision Service Communication Infrastructure Resource Monitoring Service Access Agent

Access Agent	
Responsabilities	**Interactions**
Provides an interface for user interaction with the Grid Allows application execution requests submission and monitoring	Resource Provision Service Communication Infrastructure Scheduling Service

Camargo, Raphael & Goldchleger, Andrei & Carneiro, M & Kon, Fabio. (2004). Grid: An Architectural Pattern.

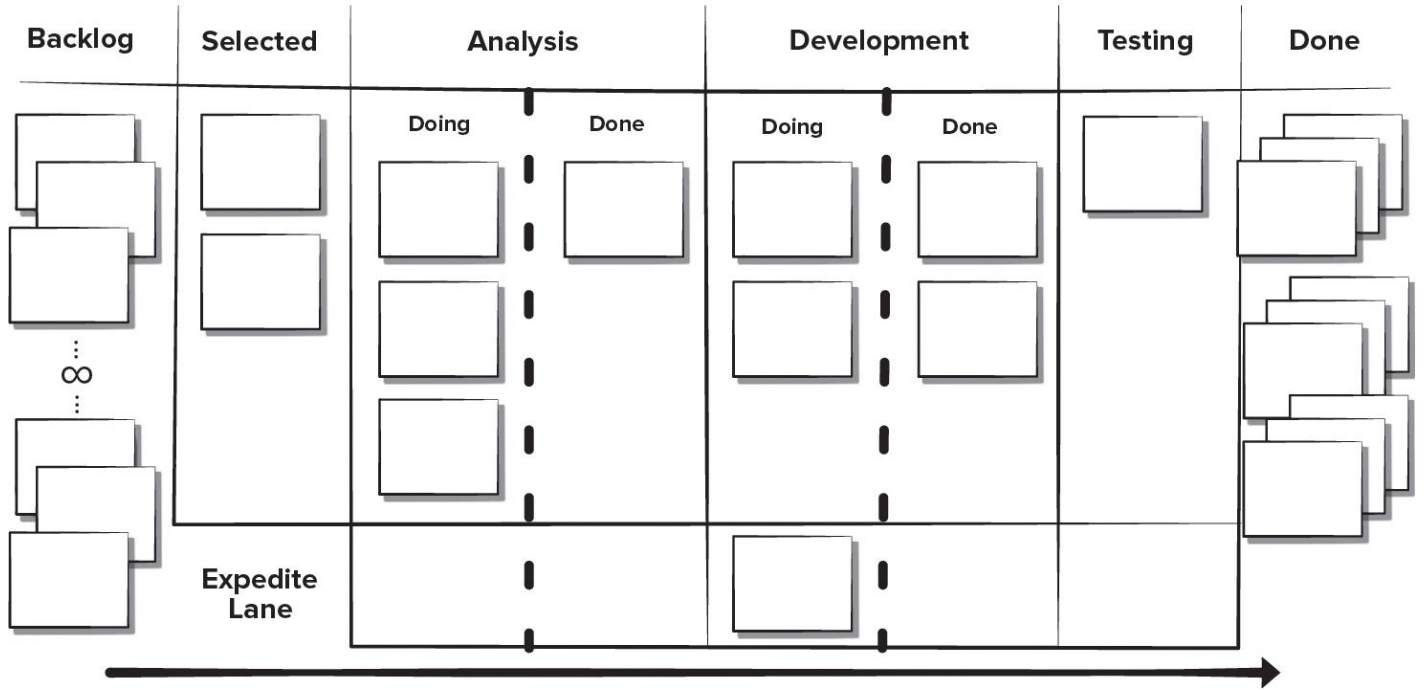
Pros and Cons of XP

- Pros
  - Emphasizes customer involvement
  - Establishes rational plans and schedules
  - High developer commitment to the project
  - Reduced likelihood of product rejection
- Cons
  - Requires frequent meetings about increasing costs
  - Allows for excessive changes
  - Depends on highly skilled team members

[PrMa20]

Kanban

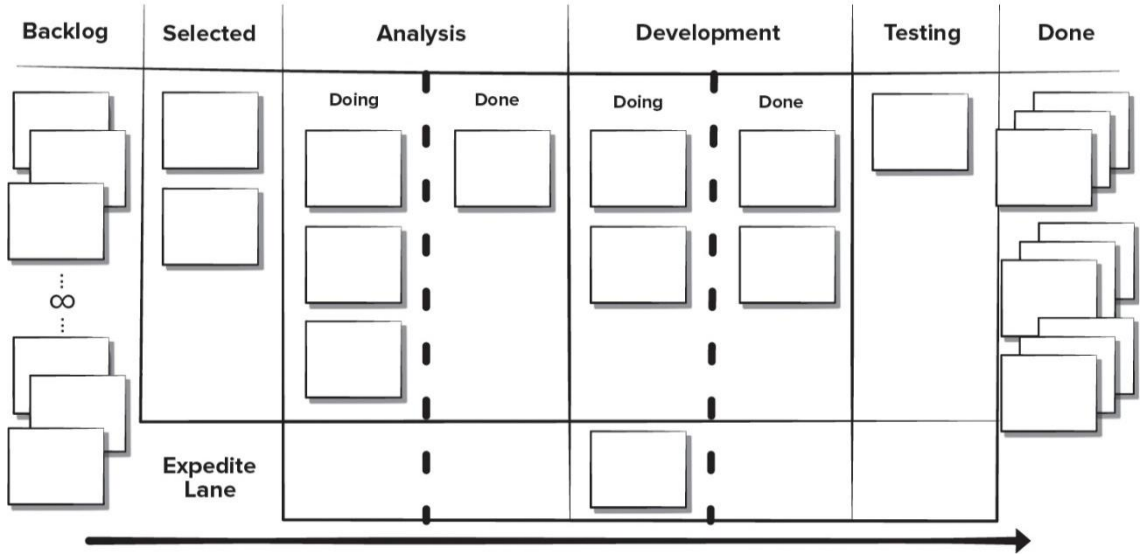
- Methodology for managing changes and service delivery
  - By plan visualization, ongoing communication with customers, and identifying urgent tasks accurately
  - Originated at Toyota and adapted to software development by David Anderson in 2010
  - Daily standup meeting & weekly retrospective meeting
  - Can easily be combined with other agile practices



[PrMa20]

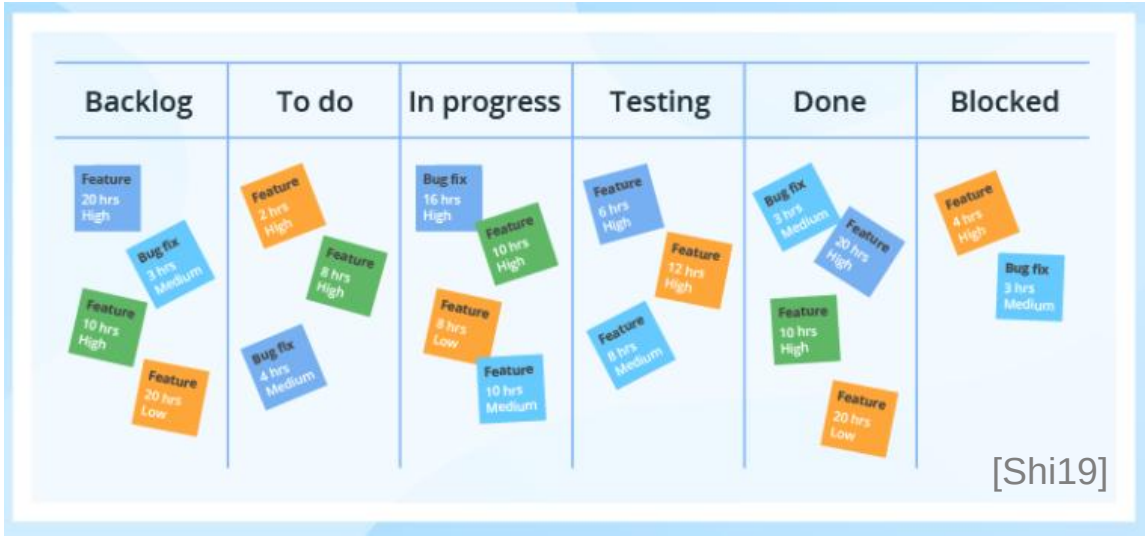
Kanban Practices

- Visualizing workflow using a *Kanban board* – a clear representation of all project activities, responsible persons, and progress [PrMa20]
- Limiting the amount of *work in progress (WIP)* at any given time
- Managing workflow to reduce waste by understanding the current value flow
- Making process policies explicit and the criteria used to define “done”
- Focusing on continuous improvement by creating feedback loops where changes are introduced
- Make process changes collaboratively and involve all stakeholders as needed



Pros and Cons of Kanban

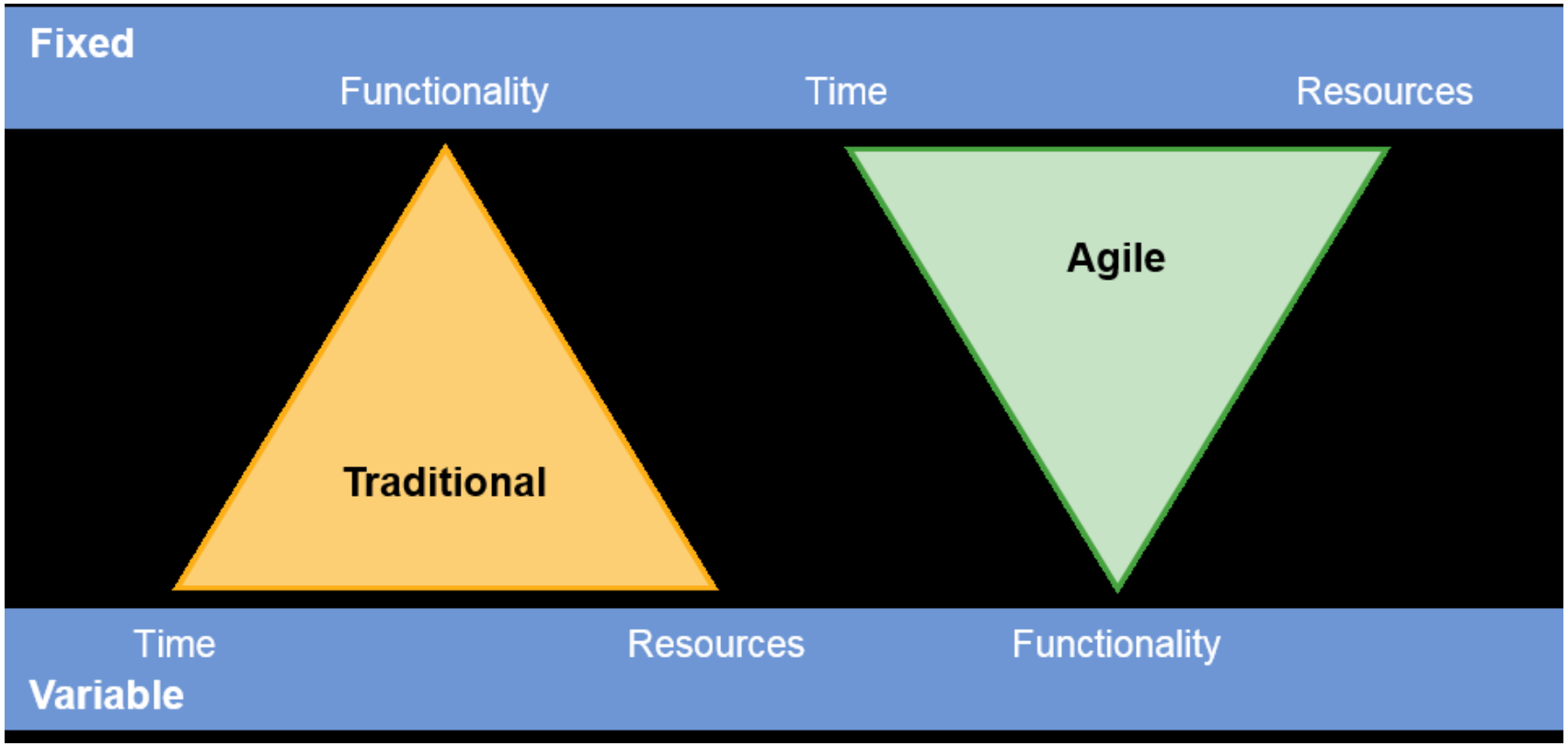
- Pros
  - Lower budget and time requirements
  - Allows early product delivery
  - Process policies written down
  - Continuous process improvement



- Cons
  - Team collaboration skills determine success
  - Poor business analysis can doom the project
  - Flexibility can cause developers to lose focus
  - Developer reluctance to use measurement

[PrMa20]

Traditional Processes Vs. Agile Processes (source: Gartner 2016)



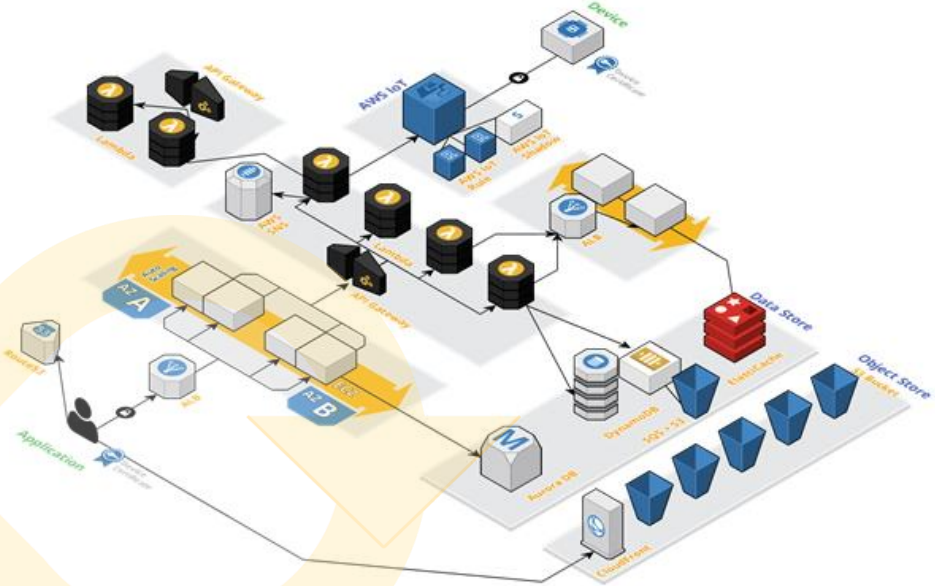
Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

DEVOPS

As a Software Engineer…

- Your task is to design and develop a smart home system where a butler robot understands the user's needs and seamlessly controls and coordinates home appliances.
- How would you enable the software system to evolve dynamically and deployed seamlessly?

ThinQ Architecture on AWS Cloud



https://aws.amazon.com/ko/solutions/case-studies/lg-electronics/

Enabling software to evolve dynamically and deployed seamlessly

ChatGPT's Answer:

To ensure that the **LG smart home butler system** evolves dynamically and is deployed seamlessly, I'd implement a **modular, scalable, and CI/CD-driven** architecture.

...

🚀 Final Thought: Enabling a Living, Adaptive System

- By combining **microservices, AI-driven learning, CI/CD automation, and feature flags**, the **LG smart home butler** will continuously evolve without service disruptions.

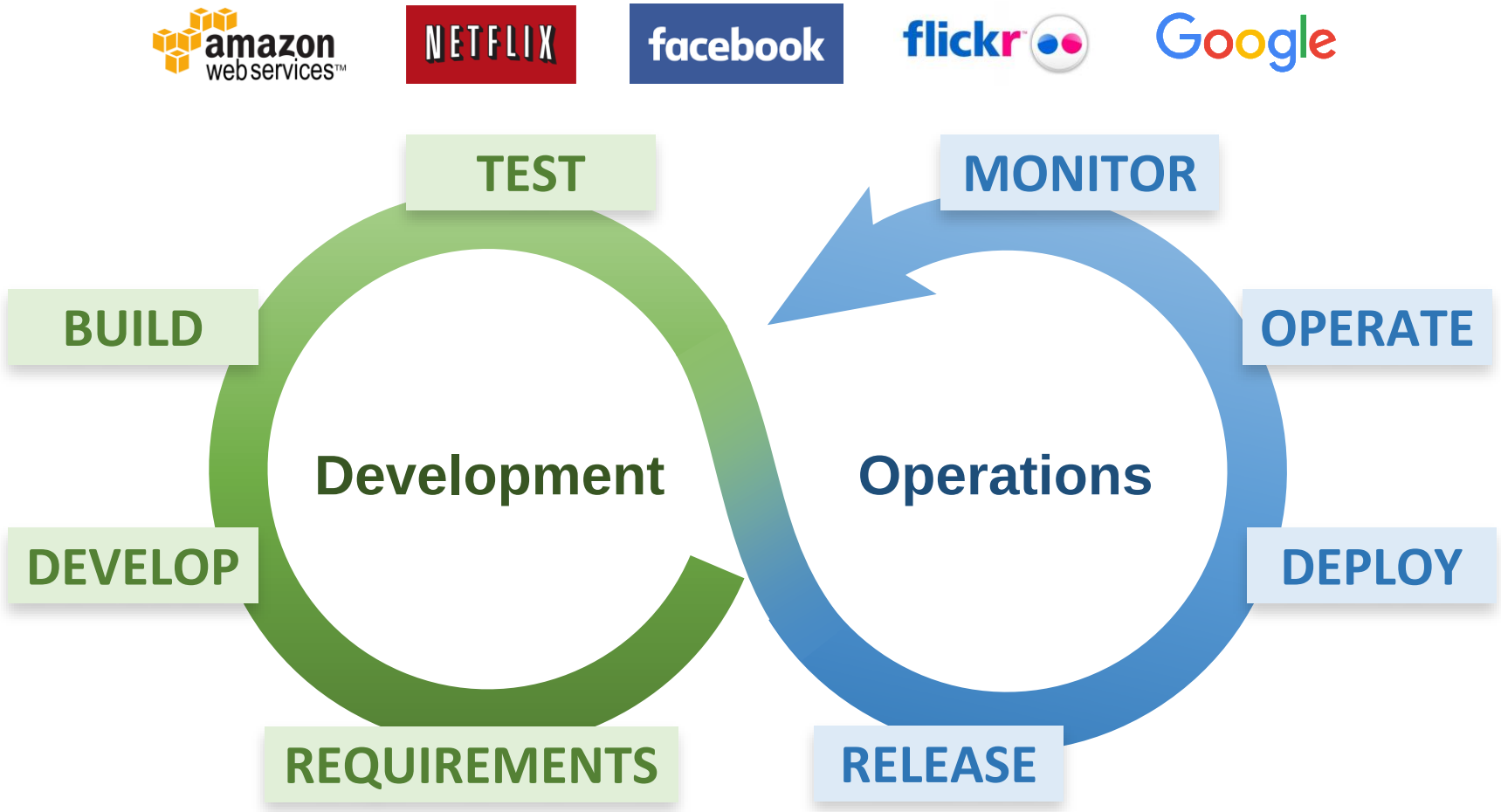
Challenge of Software Delivery

[Ama17] "Practicing Continuous Integration and Continuous Delivery on AWS", AWS White paper, Amazon Web Services, Inc., 2017

- Effectively dealing with rapidly changing competitive landscapes
- Meeting and managing evolving security requirements, and performance scalability
- Bridging the gap between operations stability and rapid feature development
- Many (thousands of) independent software teams should be able to work in parallel to deliver software quickly, securely, and reliably

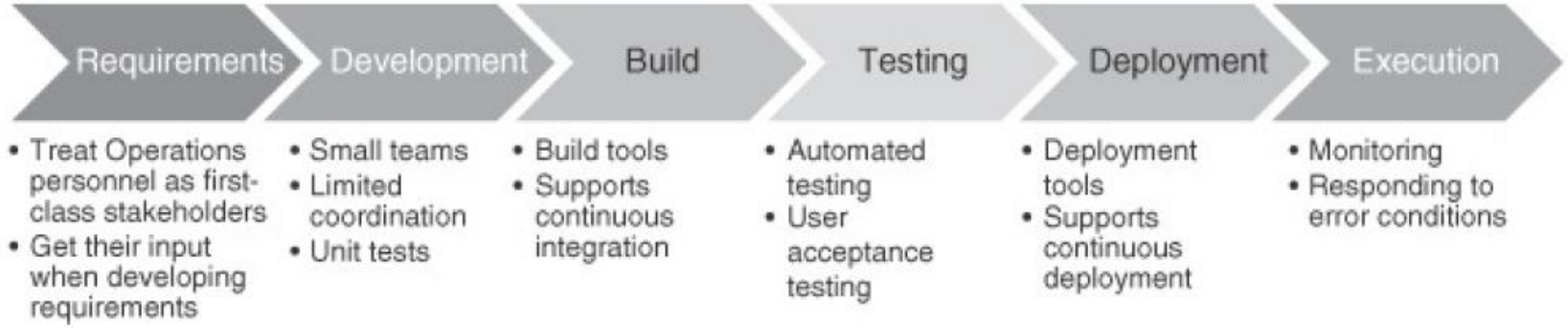
DevOps

*"**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." [Bass et al.]*



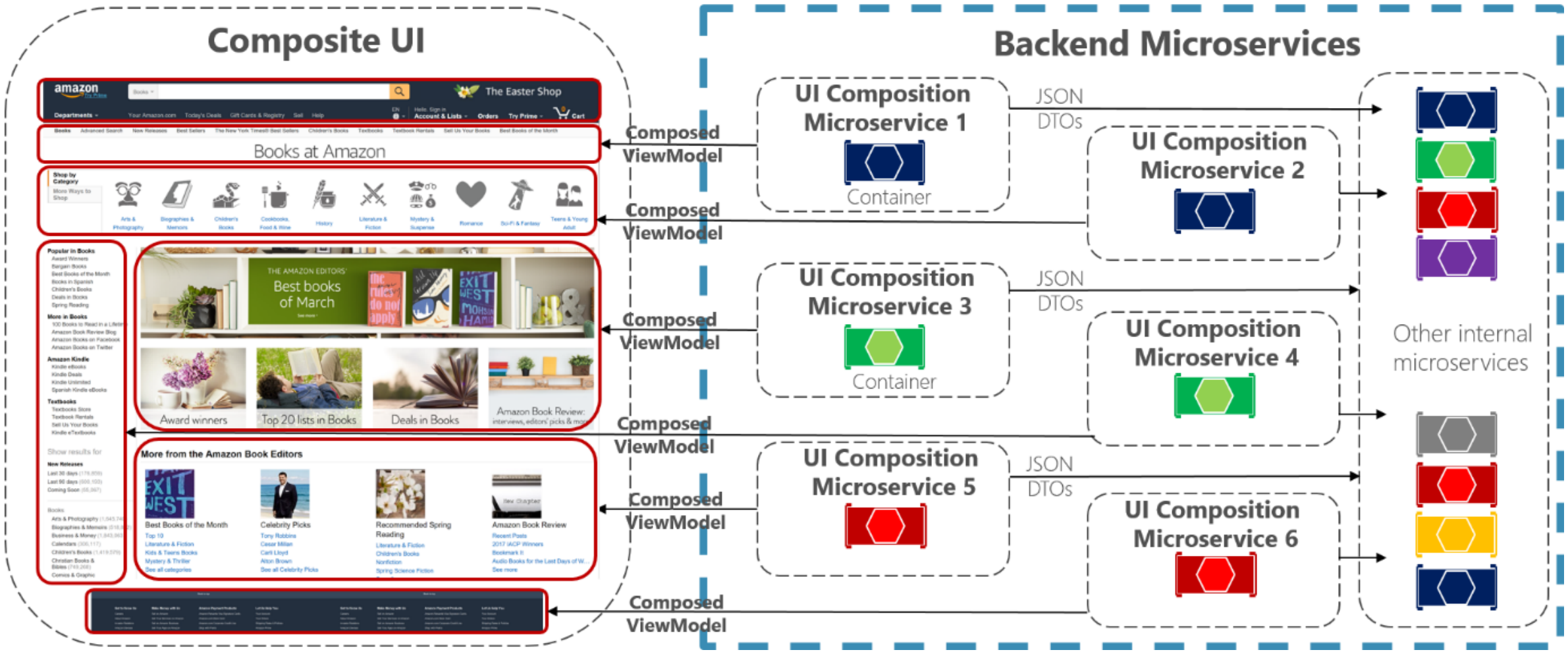
DevOps Practices

- Developed by Patrick DeBios in 2009
- Consider Ops requirements in software development – e.g., logging and monitoring requirements
- Shorten the bug-fixing time (the time between error detection and repair)
- Enforce the deployment process and trace the history of software deployment
- Use *continuous deployment* – shorten the time between code commit and deployment & use of automated tests
- Develop infrastructure code – e.g., deployment scripts



[BWZ15] Len Bass, Ingo M. Weber, Liming Zhu, DevOps: A Software Architect's Perspective, Addison-Wesley, 2015

Microservice Architecture for DevOps



.NET Microservices — Architecture for Containerized .NET Applications (http://aka.ms/MicroservicesEbook)

DevOps Teams

https://entrepreneurshandbook.co/lessons-from-the-billionaire-ceo-who-invented-the-2-pizza-rule-a4b949989c03

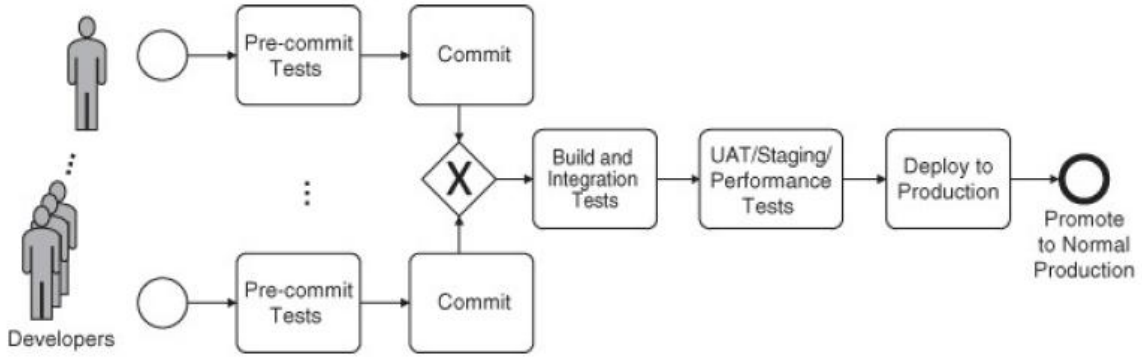
- Relatively small teams – “two pizza rule” (Jeff Bezos, Amazon)
- Advantages of small teams
  - Make decisions quickly
  - Work together in a coherent unit in which everyone understands common goals
  - Express opinions and ideas easily
- Team roles
  - Team lead – project management
  - Team members – developers
  - Service owner – responsible for outside coordination (communicate with stakeholders)
  - Reliability engineer – monitor the service after deployment & diagnose, mitigate and repair problems
  - Gatekeeper (release coordinator) – make decision on deploying services based on test results and with communicating with other roles

[BWZ15] Len Bass, Ingo M. Weber, Liming Zhu, DevOps: A Software Architect’s Perspective, Addison-Wesley, 2015

Building and Testing in DevOps

- Team members can work on different versions concurrently
- Work is not lost if a team member leaves the team
- Code can be easily tested
- Code can be easily integrated with the code produced by other members (other teams)

Deployment pipeline:

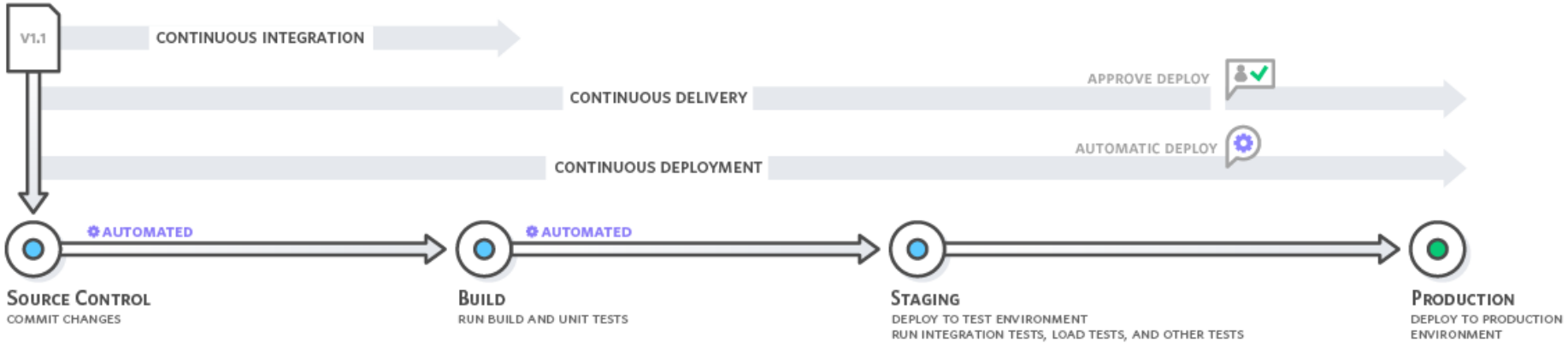


- An integrated version can be easily deployed into various environments (e.g., testing, staging, and production)
- An integrated version can be tested without affecting the production version
- A recently deployed version can be closely supervised
- Older versions are available in case a problem
- Code can be rolled back in the case of a problem

[BWZ15] Len Bass, Ingo M. Weber, Liming Zhu, DevOps: A Software Architect's Perspective, Addison-Wesley, 2015

Continuous Integration, Delivery & Deployment

[Ama17] "Practicing Continuous Integration and Continuous Delivery on AWS", AWS White paper, Amazon Web Services, Inc., 2017



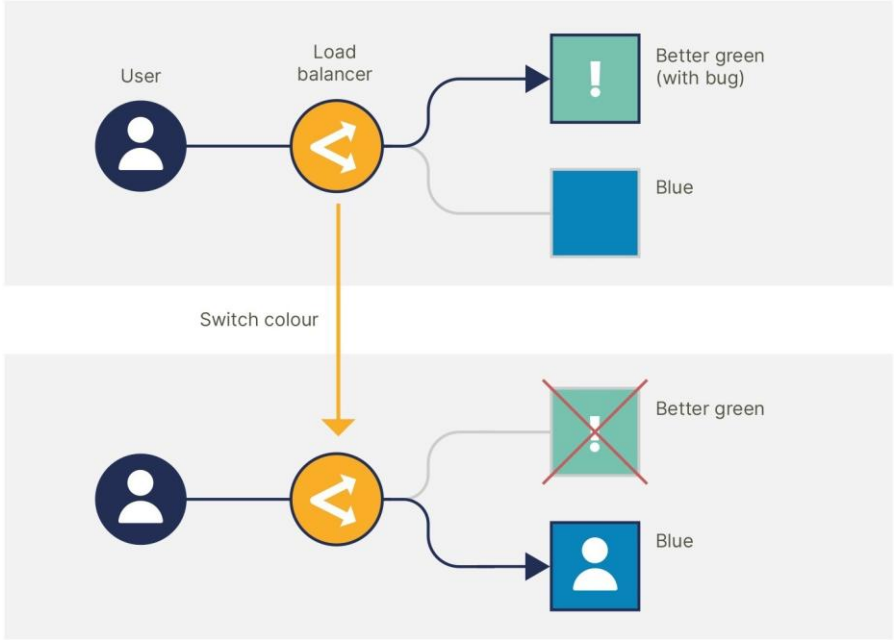
- **Continuous Integration (CI)** – automate the triggers between one phase and the next, up to integration tests
- **Continuous Delivery (CD)** – automate the triggers as far as the staging system (UAT / staging / performance tests)
- **Continuous Deployment** – automate the entire deployment pipeline (i.e., up to the deployment into the production system)
  - Once a service is deployed into production it is closely monitored for a period and then it is promoted into normal production

[BWZ15] Len Bass, Ingo M. Weber, Liming Zhu, DevOps: A Software Architect's Perspective, Addison-Wesley, 2015

Software Architecture for DevOps

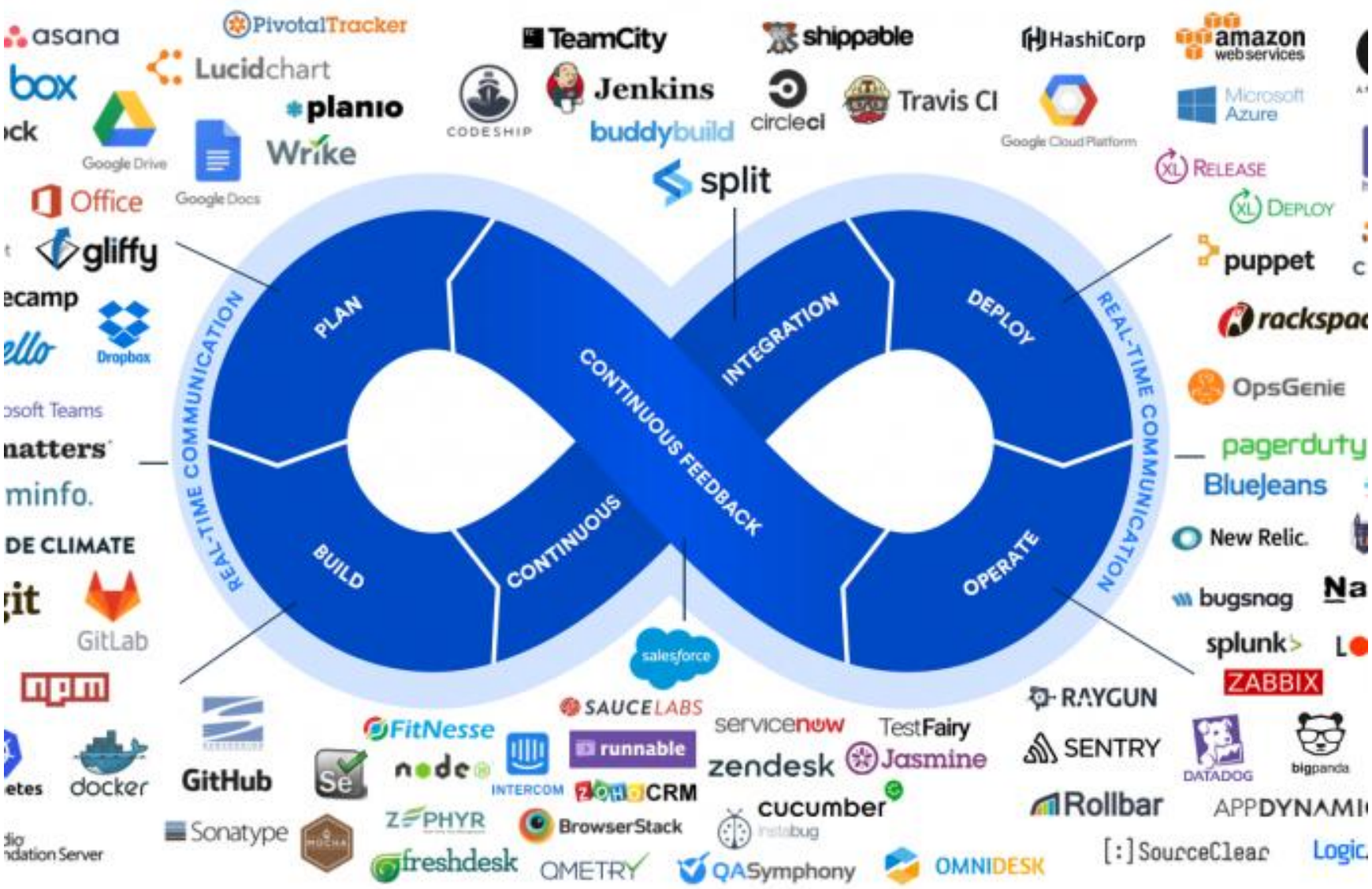
[BWZ15] Len Bass, Ingo M. Weber, Liming Zhu, DevOps: A Software Architect's Perspective, Addison-Wesley, 2015

- Architectural refactoring may be necessary to get the full benefit of DevOps
- Architectural requirements for DevOps:
  - Components should be small enough to be developed by small teams
  - Components should be compatible with other components with which they interact
  - Components should be independently deployable
  - Rolling back a deployment (in the event of errors) should be possible



https://gearset.com/careers/blog/instant-rollbacks-without-interruption-how-we-ship-new-versions-of-gearset-every-day/

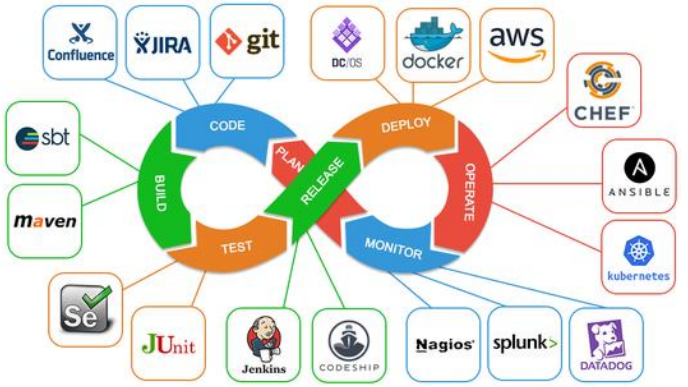
DevOps Tools



https://www.osolabs.com/blog/devops-tools-of-the-trade/

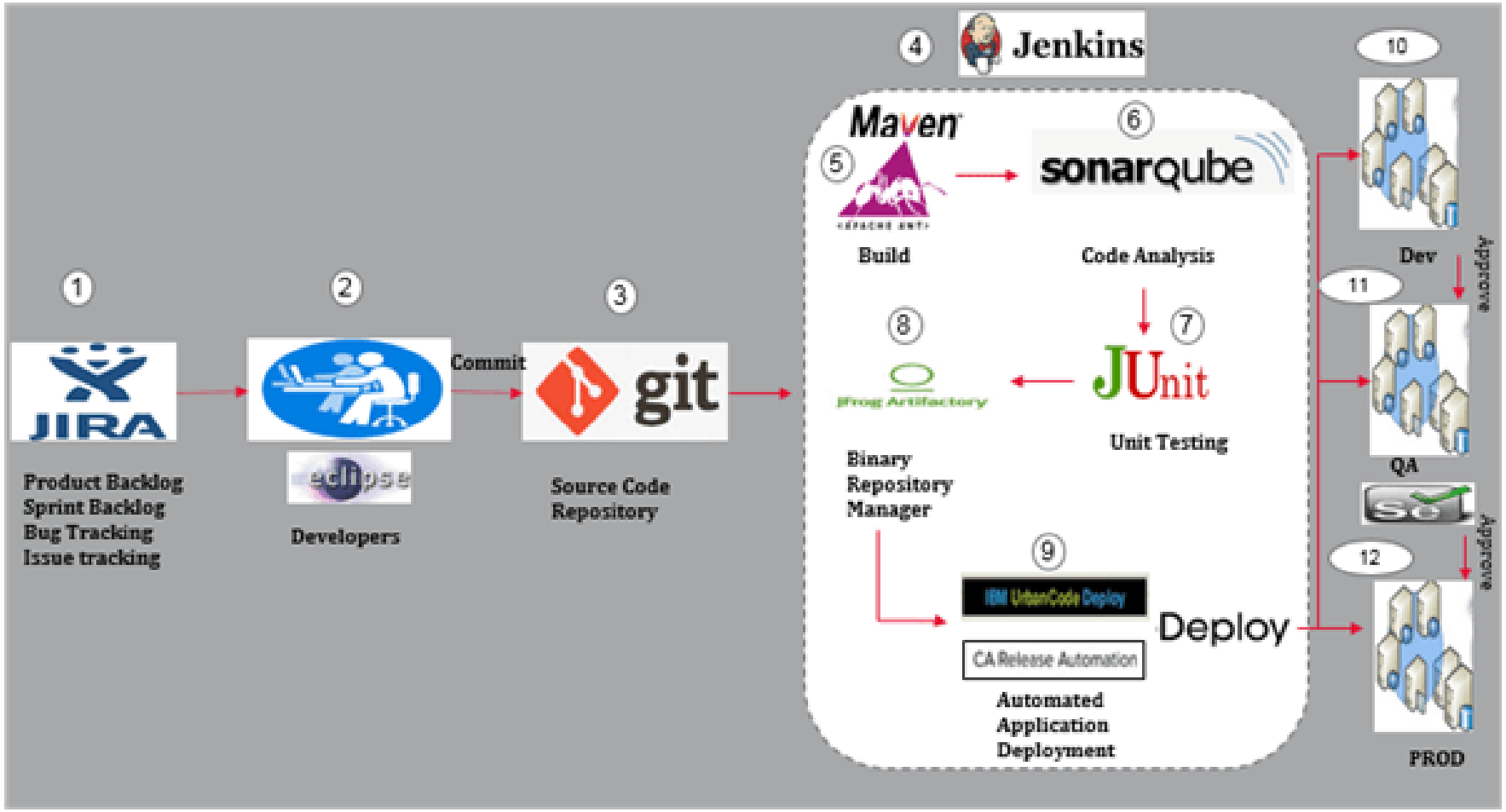
Popular DevOps Tools

- Git – version control & collaboration
- Jira – project management & issue tracking
- Maven – automated build and management
- Selenium – automated testing
- SonarQube – automated code quality management for CI
- Docker – automated deployment
- Jenkins – continuous integration (various plug-ins for building, testing & deploying)
- Chef – configuration management (infrastructure as code)
- Kubernetes – automated deployment, scaling & management
- Splunk – monitoring & visualization
- …



https://www.osolabs.com/blog/devops-tools-of-the-trade/

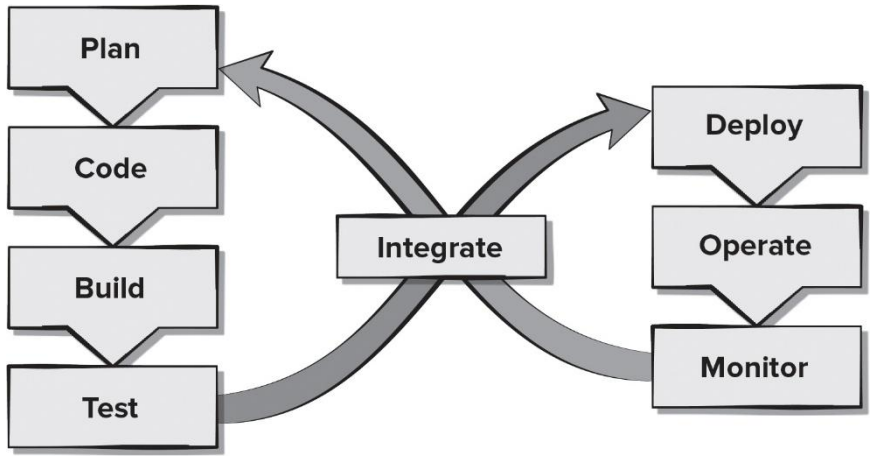
A DevOps Tool Chain



https://www.softwaretestinghelp.com/devops-tools/

Pros and Cons of DevOps

- Pros
  - Reduced time to code deployment
  - Team has developers and operations staff
  - Team has end-to-end project ownership
  - Proactive monitoring of deployed product
- Cons
  - Pressure to work on both old and new code
  - Heavy reliance on automated tools to be effective
  - Deployment may affect the production environment
  - Requires an expert development team



[PrMa20]

Which Process Model?

- Project A: A company had developed a *telephone switching system* before, had been sold to many customers, have software engineers who were involved in the development of the first version and now tries to extend the existing system.

- Project B: A company tries to develop a new *image processing system* in which an image can be stored together with sounds so that when the user looked at the picture taken by a camera, the picture image with sounds can be shown/heard from a digital storage.

- Project C: A company tries to provide a new *web service* by which users can interact with each other in a metaverse environment to sell and buy used cars.

Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

QUESTIONS?