Introduction to Software Engineering

What is Software Engineering?

From Wikipedia:

Software engineering is a systematic engineering approach to software development.

A software engineer is a person who applies the *principles of software engineering* to *design, develop, maintain, test, and evaluate* computer software.

Engineering techniques are used to inform the *software development process* which involves the *definition, implementation, assessment, measurement, management, change, and improvement of the software life cycle process itself*. It heavily uses *software configuration management* which is about systematically controlling changes to the configuration, and maintaining the *integrity and traceability* of the configuration and code throughout the system life cycle. Modern processes use *software versioning*.

What is Software Engineering?

- A systematic engineering approach to software development
- A software engineer is a person who applies the **principles of software engineering** to…
  - **design,**
  - **develop,**
  - **maintain,**
  - **test,**
  - and **evaluate** computer software.

What is Software Engineering?

Engineering techniques are used to inform the **software development process** which involves the

- **definition,**
- **implementation,**
- **assessment, measurement,**
- **management,**
- **change,** and **improvement**
- of the **software life cycle process** itself.

What is Software Engineering?

[The software development process] heavily uses **software configuration management** which is about

- systematically controlling changes to the configuration, and
- maintaining the **integrity and traceability** of the configuration and code throughout the system life cycle.
- Modern processes use **software versioning**.

Principles of Software Eng.

From Barry Boehm's 1983 journal article

1. Manage using a phased life-cycle plan
2. Perform continuous validation
3. Maintain disciplined product control
4. Use modern programming practices
5. Maintain clear accountability for results
6. Use better and fewer people
7. Maintain a commitment to improve the process

+ Use proven patterns of **architecture and design**

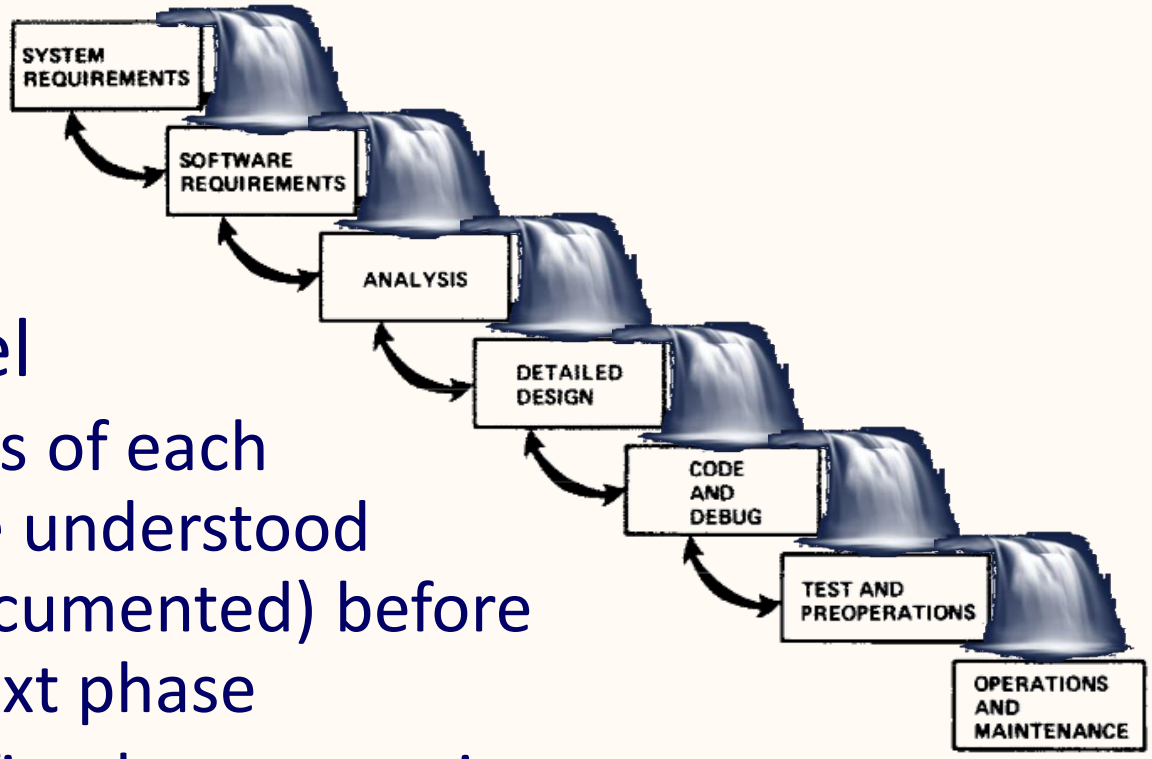
Outline

- Project management
  - Software development processes
  - Validation, accountability, cont. improvement
- Architecture and design
  - Client-server
  - Cloud, microservices
- DevOps

Outline

- Project management
  - Software development processes
    - Waterfall
    - Agile
  - Validation, accountability, cont. improvement
- Architecture and design
- DevOps

Waterfall



- Uses a phased life-cycle model
  - Major products of each phase must be understood (preferably documented) before starting the next phase
  - Iterations confined to successive stages
- Timeline: several months to 1 or 2 years

Accommodating changes

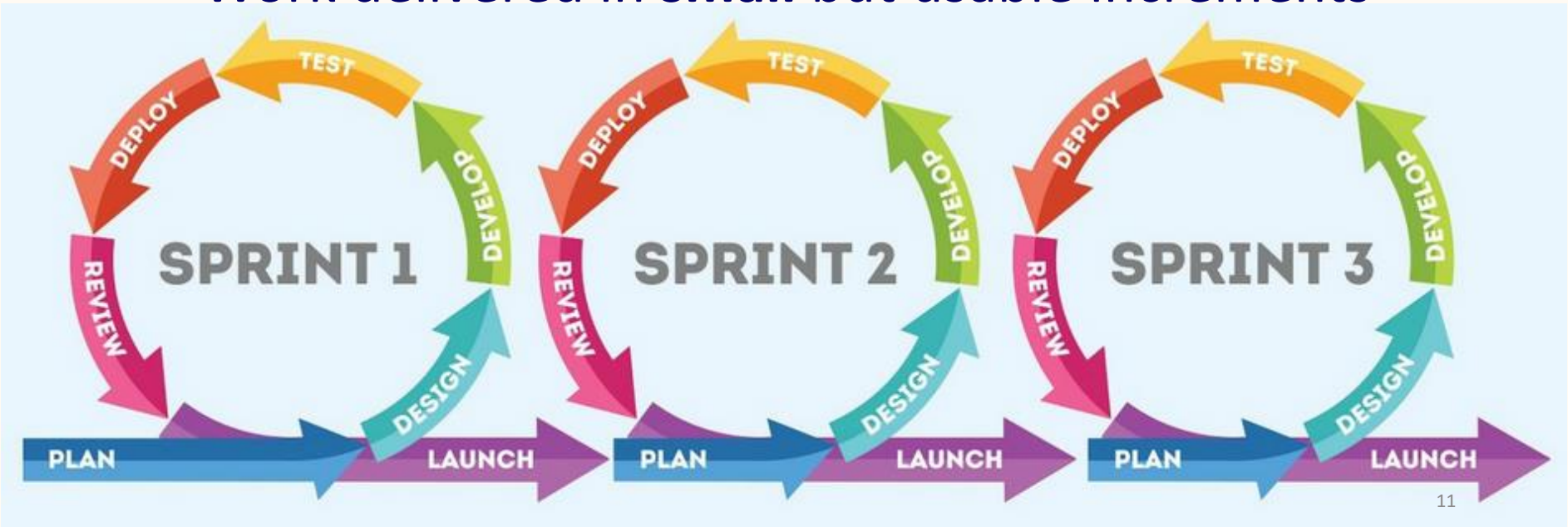
"The waterfall chart shown above is actually an **oversimplified model** of the software development process […]. In fact, any good-sized project must **accommodate changes** in requirements throughout the development cycle.“

(Boehm, 1983)

- Changes using this model are “expensive”
  - Each change may require meetings and document changes at each phase

Agile

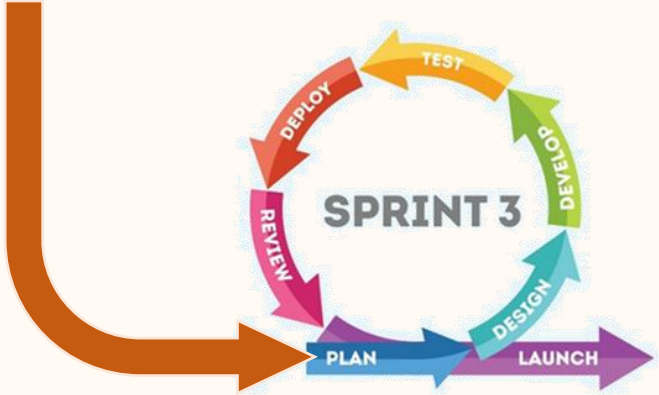
- Iterative approach to project management
  - No “big bang” launch
  - Work delivered in **small** but usable increments



Changes using Agile

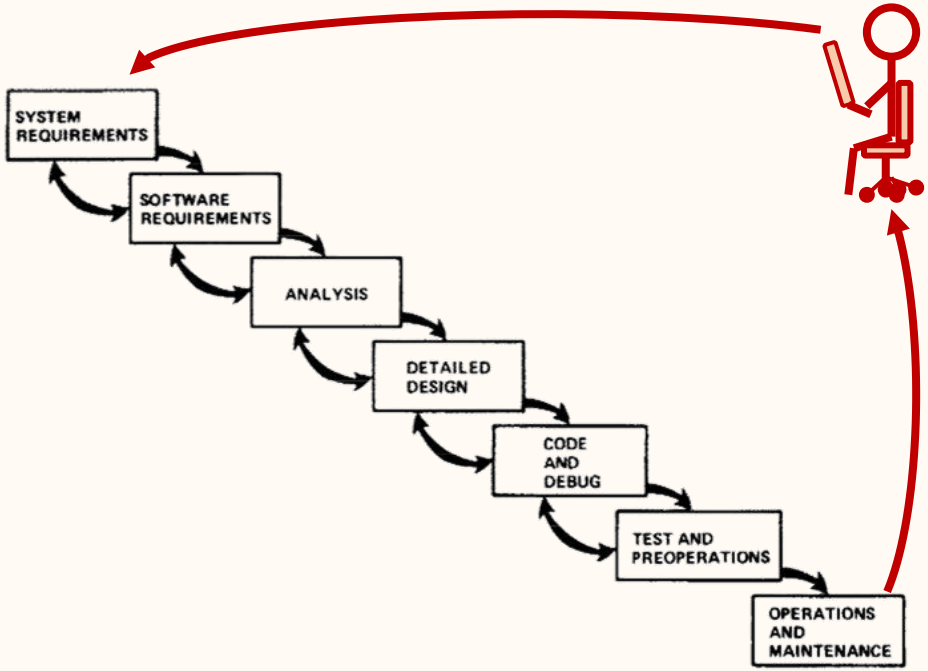
- Requirements, plans, and results are evaluated "continuously"
- Teams have a natural mechanism for responding to change quickly
  - Incorporate **changes** at beginning of each cycle

Every sprint (2-4 weeks)



A big difference? Customer role

- Agile involves the **customer** early/often



Waterfall

Agile

Verification and validation

- Did you build the system right?
  - **Verification** in Software Testing is…

    process of checking documents, design, code, and program in order to check if the software has been built ***according to the requirements*** or not
- Did you build the right system?
  - **Validation** in Software Engineering is…

    dynamic mechanism of determining if the software product ***meets the exact needs of the customer*** or not

Of the 12 Agile principles…

- First two involve customer
- Two address continuous validation
- Three address clear accountability for results

2. Perform continuous validation
5. Maintain clear accountability for results

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
…
7. Working software is the primary measure of progress.

Of the 12 Agile principles…

- Three address the way people work together

6. Use better and fewer people

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

…

11. The best architectures, requirements, and designs emerge from self-organizing teams.

Of the 12 Agile principles…

- Four address continuous improvement

7. Maintain a commitment to improve the process

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity–the art of maximizing the amount of work not done–is essential.

…

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Outline

- Project management
  - Software development processes
  - Validation, accountability, cont. improvement
- Architecture and design
  - Client-server
  - Cloud, microservices
- DevOps

What is Software Architecture?

From Wikipedia:

Software architecture refers to the fundamental structures of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations.
The architecture of a software system is a metaphor, analogous to the architecture of a building. It functions as a blueprint for the system and the developing project, laying out the tasks necessary to be executed by the design teams.

What is Software Architecture?

- Refers to…
  - Fundamental **structures** of a software system
  - Discipline of creating such **structures & systems**
- **Structure** is comprised of…
  - Software **elements** (libraries, services, …)
  - **Relations** among them (how they communicate)
  - Properties of **elements** and **relations**

What is Software Architecture?

- Metaphor: analogous to building architecture
- Blueprint for system and developing project
- Lays out tasks for design teams

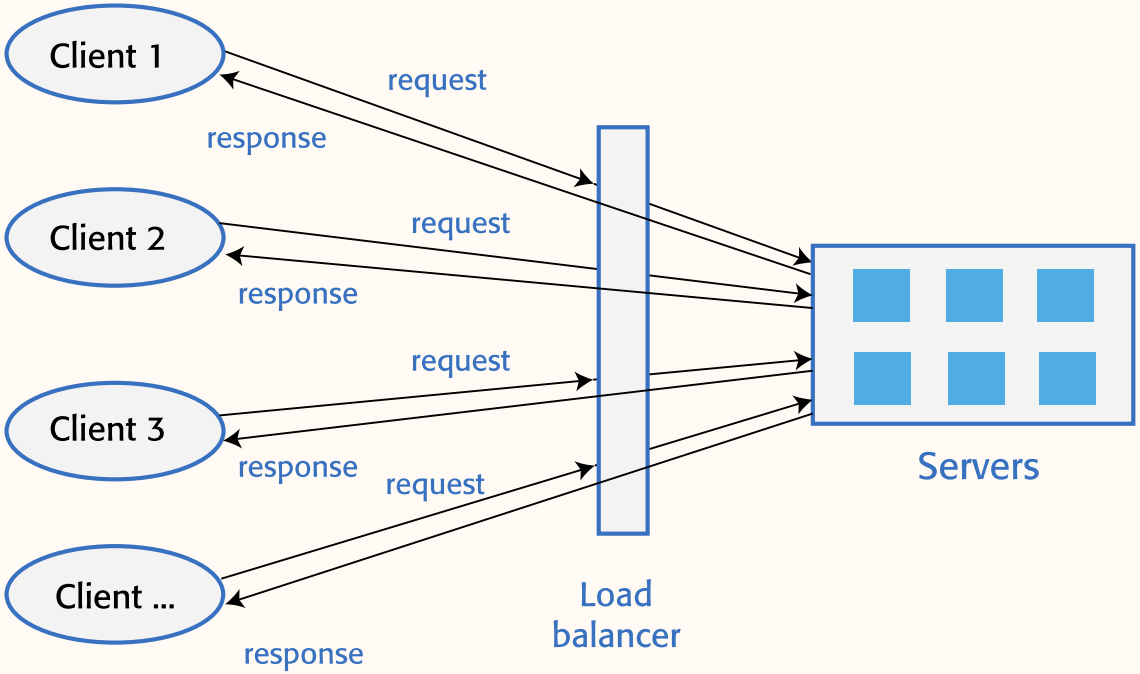
Architectural Design Aspects

Chapter 4

- System decomposition (what are key parts?)
- Distribution architectures
  - What functionality runs on which systems?
  - Examples:
    - Client-server architecture (next slide)
    - Service-oriented architecture
- Technologies (database, platform, server, …)
- Prototyping & product development

Client-server architecture

Example (Figure 4.12 in text)

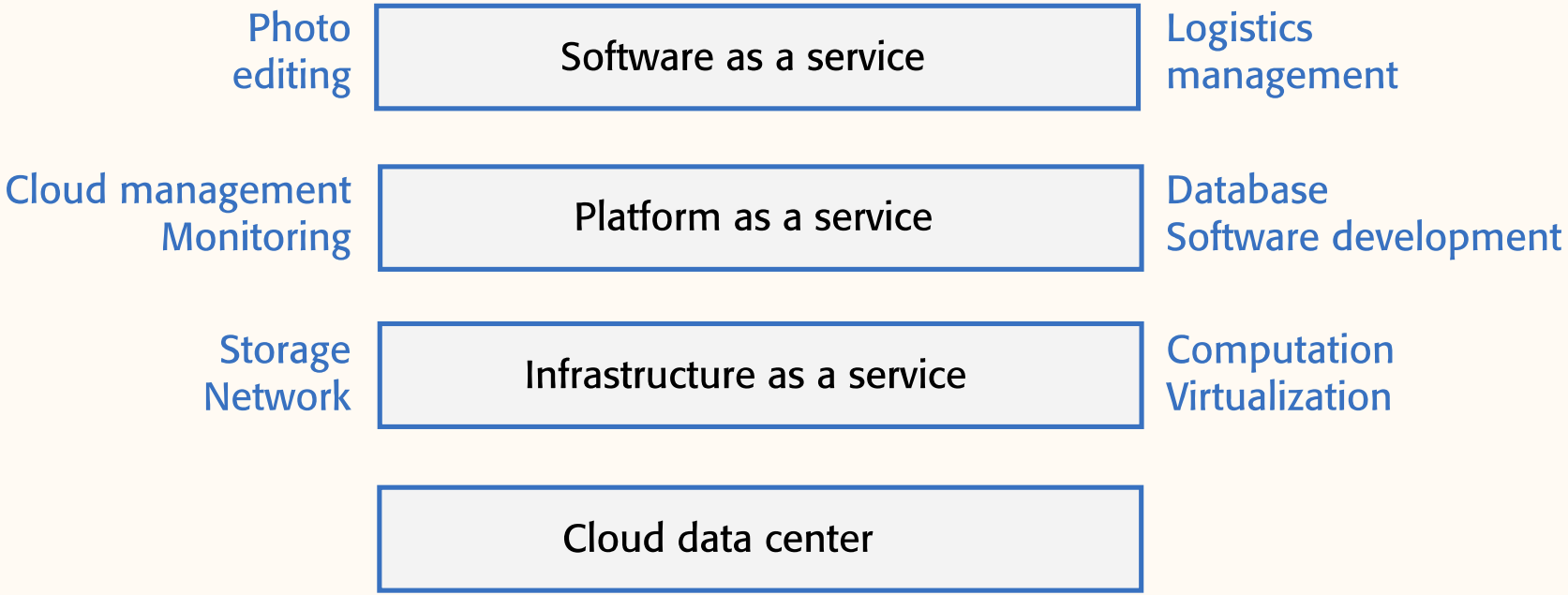


Cloud-based software aspects Chapter 5

- "The Cloud": Internet servers for code/data
- Virtualization and containers (like Docker)
- "Everything as a service"
  - Software (SaaS)—end-user apps
  - Platform (PaaS)
  - Infrastructure (IaaS)—rentable cloud resources
- Cloud software architecture
  - monolithic vs. service-oriented

Everything as a Service

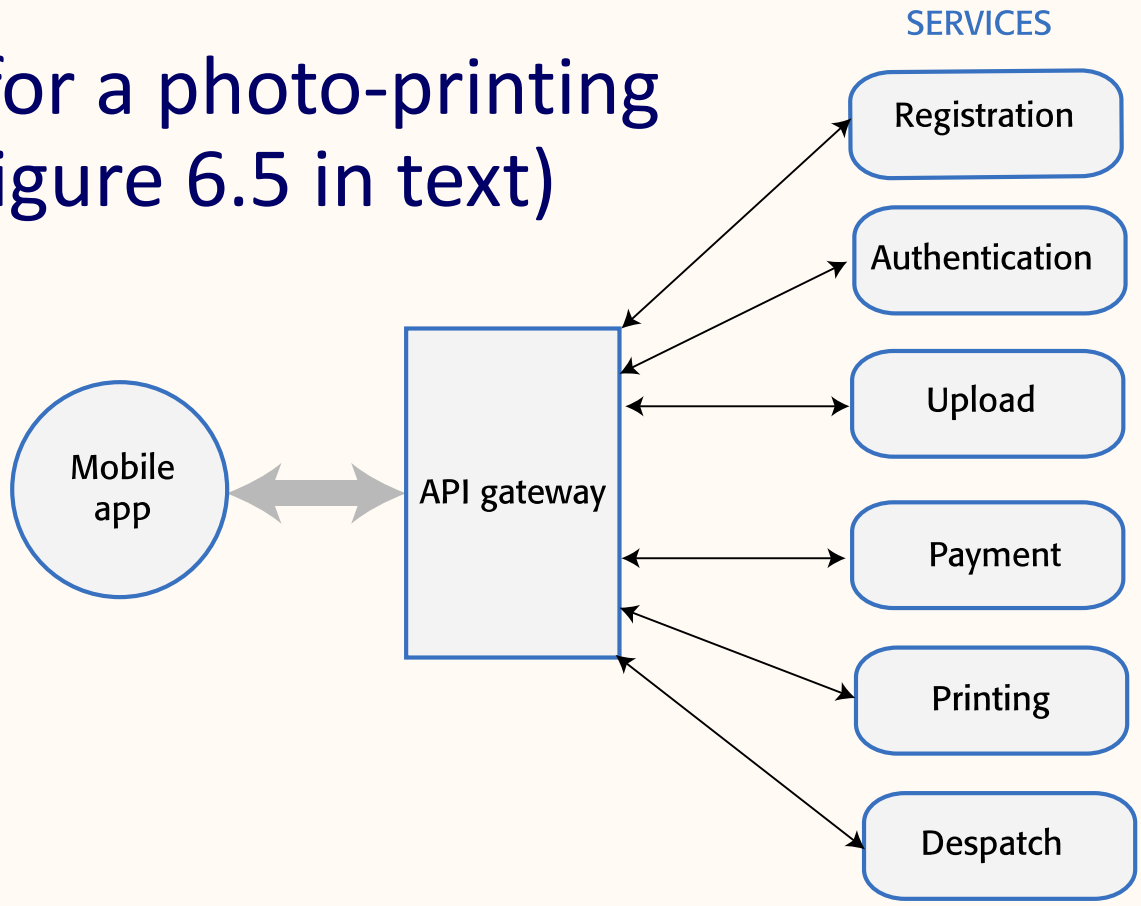
Some examples (Figure 5.5 in text)



Microservices architecture

Chapter 6

Example for a photo-printing system (Figure 6.5 in text)



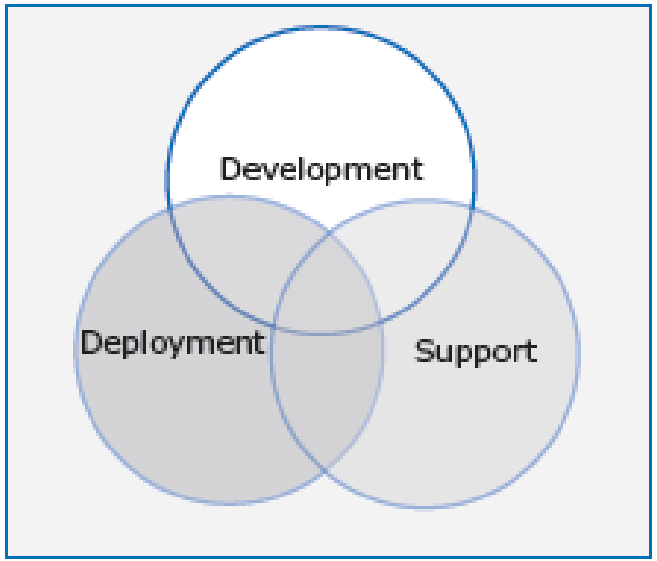
Outline

- Project management
  - Software development processes
  - Validation, accountability, cont. improvement
- Architecture and design
  - Client-server
  - Cloud, microservices
- DevOps

DevOps

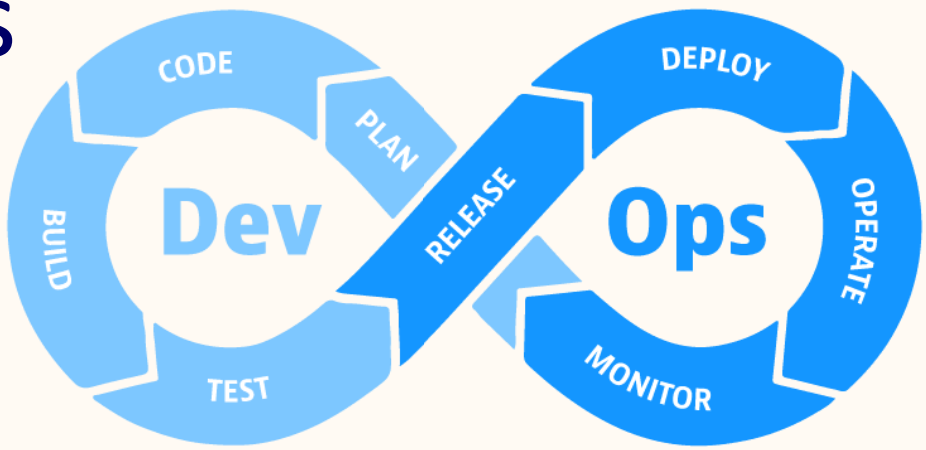
- DevOps = Development + Operations
- Integrates activities on one team
  - Development
  - Deployment
  - Support

(Figure 10.2 in text)



Multi-skilled DevOps team

DevOps activities



- (Agile) Development
  - Requirements gathering
  - Design & architecture
  - Implementation
  - Testing (includes verification and validation)
- Operations
  - Deployment
    - Deploying to the cloud for consumers; configuration; etc.
  - Support
    - Setting up infrastructure; monitoring; bug fixes; etc.

DevOps principles…

From Boehm (1983):

3. Maintain disciplined product control
4. Use modern ~~programming practices~~ development processes and tools

1. Collaboration: "Everyone is responsible for everything"
   - Work as a team - communication, feedback, etc.
   - End-to-end responsibility for software/service
2. Automation: “Everything that can be…”
   - Development, testing, deployment, etc.
3. Continuous (data-driven) improvement: “Measure first, change later”
   - Use DevOps tools to measure and continually improve processes and tools.

DevOps and code management

- **DevOps platforms**
  - Facilitate collaborative software development
  - Popular platforms: Jira, GitHub, **GitLab**, AWS…
  - Typically integrated with **code management** (**version control**) systems
- Source code management (SCM)
  - Tracks history: changes to source code repository
  - Merging: Helps resolve changes that conflict
  - Today's standard: **Git**

DevOps and code management

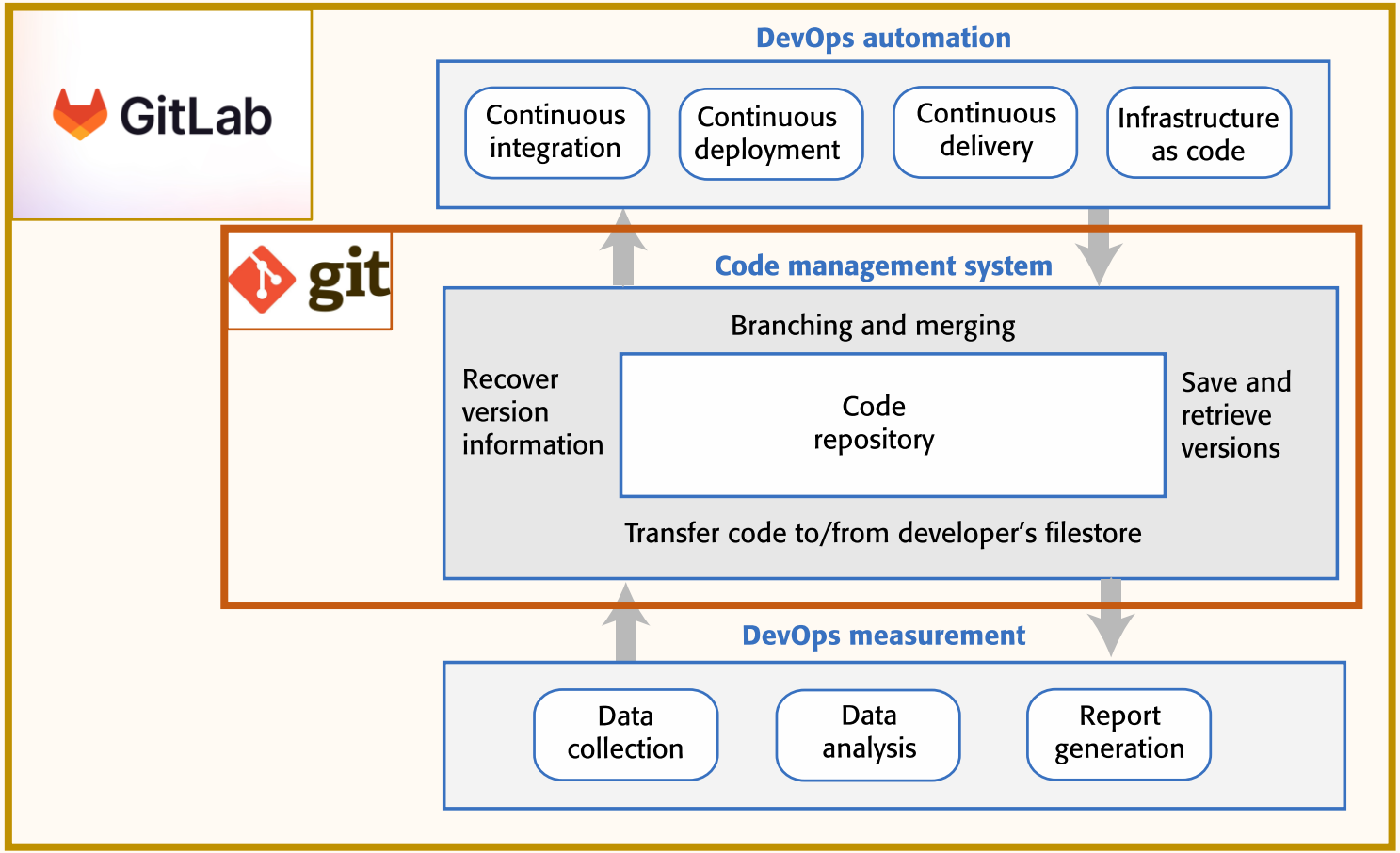


Figure 10.3 in text

Summary

- Project management
  - Software development processes
  - Validation, accountability, cont. improvement
- Architecture and design
  - Client-server
  - Cloud, microservices
- DevOps

Agile

GitLab

Git