ECE 448/528
Application Software Design

Lecture 2. Software Engineering and SaaS
Spring 2025

Won-Jae Yi, Ph.D.

Department of Electrical and Computer Engineering
Illinois Institute of Technology

Software Engineering and SaaS

Software

- What is the difference between software and programs written for (other) course projects?
- More features? Better quality? Higher performance?
- Software is designed in a way so that it can be improved and reused.
- Software engineering practices attempt to define a process to reduce the overall risk

The Waterfall Model

- A conventional process of software development
  - Stage 1: Requirements analysis and definition
  - Stage 2: System and software design
  - Stage 3: Programming and unit testing
  - Stage 4: Integration and system testing
  - Stage 5: Operation and maintenance
- Waterfall: never go back and revise previous stages
- Advantages
  - Detailed planning for time/personnel/budget within the constraints
  - Goals are well-defined in each stage

The Waterfall Model: Challenges II

- **Demo Availability:** Limited to post-integration and system testing.
  - This timeline poses significant risks if delay occur.
- **Operation and Maintenance**: Can be challenging.
  - Bugs might originate from the operating system or supporting libraries, and updates could disrupt the entire system.
  - Supporting multiple software versions may be necessary, as some users resist upgrading.
- **Flexibility**: Insufficient for today's fast-paced, ever-changing environment.

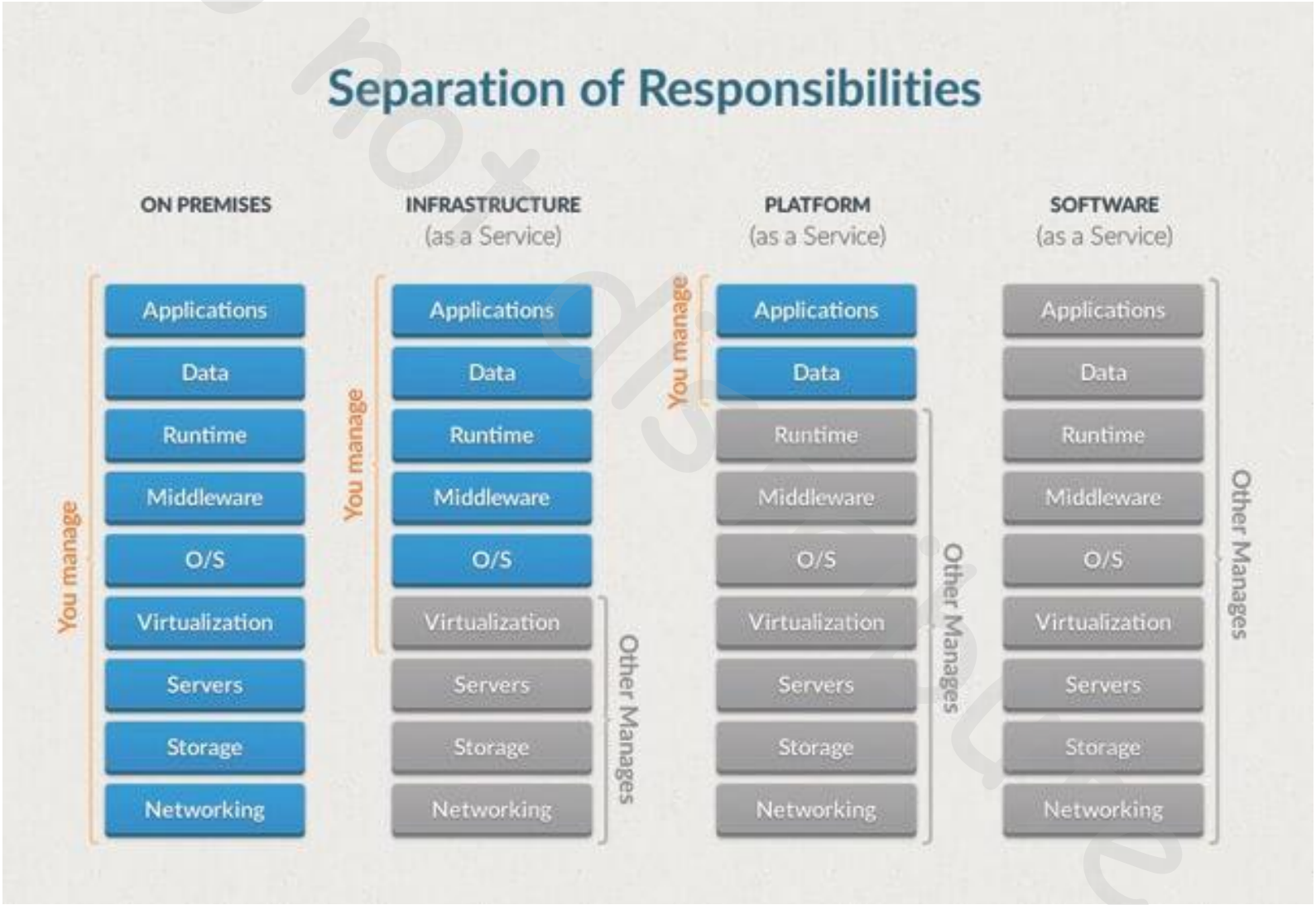
Application Software

- To address the challenges of the waterfall model, it is crucial to analyze the product it applies to—particularly software, especially application software.
- **Common Examples**: Spreadsheets, games, photo editors, finance applications, and more.
  - **Shared Characteristics**: Designed for end-users to be intuitive and require minimal training.
- **Emerging Trends**:
  - Operate from any location.
  - Enable access to data from anywhere.
  - Foster connections with other users.

Cloud Computing

- Scalability, Cost-effective, Backups and High Availability
- Different types of cloud computing platforms available
  - **IaaS (Infrastructure-as-a-Service)**
    - Provides access to computing resources in a VM environment
    - Amazon AWS, MS Azure, Google Compute Engine
  - **PaaS (Platform-as-a-Service)**
    - Provides a platform (runtime) and environment to build applications
    - IBM Watson, Google App Engine, SalesForce
  - **SaaS (Software-as-a-Service)**
    - Provides on-demand software for end users
    - Gmail, Facebook, Office 365, Google Apps
  - **BaaS (Backend-as-a-Service)**
    - Provides a backend for applications (mostly mobile) including APIs and tools for different computer languages to integrate with their backend
    - Google Firebase

Cloud Computing



Software as a Service (SaaS)

- **Cloud Computing for End Users**
- **The Server (Backend)**
  - Hosted on the cloud
  - Managed by professionals to handle application features that end users cannot or prefer not to maintain
- **The Client (Frontend)**
  - Operates on devices owned by end users.
  - Delivered as web or mobile applications.

SaaS Advantages

- **Development**
  - Since the server operates in a predefined environment, concerns about supporting multiple OS versions, databases, etc., are minimized.
  - A centralized server simplifies communication and data sharing among end users.
- **Operation and Maintenance**
  - **Effortless Client Updates**:
    - Web applications are updated instantly.
    - Mobile applications can be updated overnight.
- **Controlled Feature Rollout**:
    - New features can be tested with a small group of clients first.
  - Server updates can be implemented seamlessly, without client awareness.
- In conclusion, SaaS enables the delivery of frequently changing software to meet customer demands.
  - The question remains: How can such adaptable software be developed effectively?

Agile Software Development

- **A set of software development methods that teams may choose to satisfy their needs for a specific project.**
  - Iterative and incremental development (IID)
  - Test-driven development (TDD)
  - Continuous integration (CI)

Iterative and Incremental Development (IID)

- **Definition:** Build a small portion of the system or make a small revision within each cycle
- **Process**:
  - The entire system is assembled and improved over multiple cycles.
  - Demonstrate progress at the end of each cycle to stakeholders.
  - Incorporate new customer requirements that emerge from previous cycles into subsequent iterations.
  - Apply lessons learned from earlier cycles to improve future iterations.
- **Benefits:**
  - Allows progress without a complete understanding of the entire project from the outset.
  - Limits changes, reducing risk compared to applying changes to the entire systems.
- **Combination with Other Models**: Within each cycle, the waterfall model can be used to structure development phases.

Test-Driven Development (TDD)

- **Unit Testing:**
  - Focuses on small units, such as a class.
  - Serves as an executable specification of your code, offering precision and consistency superior to plain English documentation.
- **Integration Testing:**
  - Tests the entire system as a whole.
  - Provides an opportunity to demonstrate progress to clients and facilitates communication to clarify and understand their requirements.
- **Acceptance Testing:**
  - Verifies whether the system meets the specified requirements.
  - Server updates can be implemented seamlessly, without client awareness.
- **Test-Driven Development (TDD):**
  - Tests are written before the code is developed.
  - Decomposes the entire system into smaller, testable components.
  - The course project requires unit testing with reasonable coverage, and project grading is based on acceptance testing outcomes.

Continuous Integration (CI)

- **Manual Unit Testing:**
  - Focuses on small units, such as a class.
  - Serves as an executable specification of your code, offering precision and consistency superior to plain English documentation.
- **Integration Testing:**
  - Tests the entire system as a whole.
  - Provides an opportunity to demonstrate progress to clients and facilitates communication to clarify and understand their requirements.
- **Acceptance Testing:**
  - Verifies whether the system meets the specified requirements.
  - Server updates can be implemented seamlessly, without client awareness.
- **Test-Driven Development (TDD):**
  - Tests are written before the code is developed.
  - Decomposes the entire system into smaller, testable components.
  - The course project requires unit testing with reasonable coverage, and project grading is based on acceptance testing outcomes.

Course Project Introduction

Smart Hub for Internet of Things

- An application that allows end users to monitor and control their IoT devices.
  - We will focus on the software part.
  - Open-source solutions like Home Assistant exist but we would like to build our own to learn application software development.
- Goals
  - End users can use the application on most devices.
  - End users may choose to send device information to the cloud so the devices can be access from almost anywhere, or
  - End users may choose to keep device information local for
  - privacy reasons.
    - In such case, the application should work without Internet connections.
  - The application can manage any IoT devices and may interoperate with other software.

The IoT Hub

- **Our solution is based on SaaS**
  - Server Backend (Projects 4 and 5)
    - Communicates with (hypothetical) IoT devices
    - Can operate in the cloud or on-premises for enhanced privacy
  - Client (Project 6)
    - Interfaces with the server backend rather than directly communicating with the individual devices.
    - Implements a frontend web application:
      - Accessible via a browser, eliminating the need for installations.
      - Avoids the need to develop platform-specific mobile apps.
  - Open Protocols
    - Utilizes MQTT middleware for communication between the server and IoT devices.
    - Provides RESTful services to enable clients and other software to monitor and control connected devices.

The IoT Simulator

- **IoT Devices for the IoT Hub:**
  - Actual IoT devices are not preferred for the following reasons:
    - Impractical for coursework.
    - May result in a server compatible with only specific IoT devices rather than being universally applicable.
  - Simulator Approach:
    - Create a custom simulator to replicate the behavior of desired IoT devices.
    - Support only open protocols commonly used by many IoT devices.
    - Simplify by simulating smart plugs that can be turned on/off and measure power consumption.
    - Include a standalone web application for the simulator, enabling functionality without the IoT hub
  - Course Preparation
    - Projects 1 to 3 serve as a practical exercises to build skills in Java, networking, and HTML development.

Extensions

- **Project 7 will be optional for ECE 448 but mandatory for ECE 528.**
- Details will be released in the future about Project 7…

Hints for Project 1

- **src/main/java/ece448/iot_sim/PlugSim.java**
  - See what member variables and methods are provided.
  - Think what they are supposed to do.
- **src/test/java/ece448/iot_sim/PlugSimTests.java**
  - Add unit tests here.
- **src/main/java/ece448/grading/GradeP1.java**
  - This contains the grading test cases.
  - You are not supposed to modify this file.
  - But you may learn more about the class PlugSim here.

Practicing TDD for Project 1

- Create more unit tests in PlugSimTest.java
  - Using testInit and testSwitchOn as examples.
  - Each unit test is a method annotated with @Test.
  - Create a PlugSim object and call some of its methods.
  - Use assertFalse or assertTrue to verify the results.
  - Feel free to choose meaningful method names.
- Modify PlugSim.java to make unit tests pass.
  - Locate // P1: add your code here and add code there.
- Run grading test cases using gradle grade_p1.
  - For any failing grading test case, refer to GradeP1.java to understand why.
  - Create more unit tests to isolate the issue and then make them pass (refer to the items 1. and 2. above).
  - See if the grading test case passes now.
- These are simplified but typical red, green, refactor cycles that many developers use daily to move their projects forward.