# Fundamentals of Software Development for Electronics

Prepared By: Dr. Rony Ibrahim

الجامعة اللبنانية
UNIVERSITE LIBANAISE

# Course outline

- Session 1: Importance of Software Engineering in Electronics
- Session 2: Basic Concepts and Terminologies
- Session 3: Software Development Life Cycle (SDLC)
- Session 4: Integrated Development Environments (IDEs) and Basic Git Workflow
- Session 5 and 6: JavaScript & Introduction to TypeScript
- Session 7 and 8: Building Web Applications with ReactJS
- Session 9: CSS Basics and Responsive Design Principles
- Session 10: Basics of Flutter / React Native / Ionic
- Session 11: Project Presentation & Final Review

# Software Development Life Cycle (SDLC)

Objectives

- Understand the stages of the Software Development Life Cycle (SDLC).
- Learn different SDLC models and methodologies.
- Explore how SDLC applies to software development in electronics.

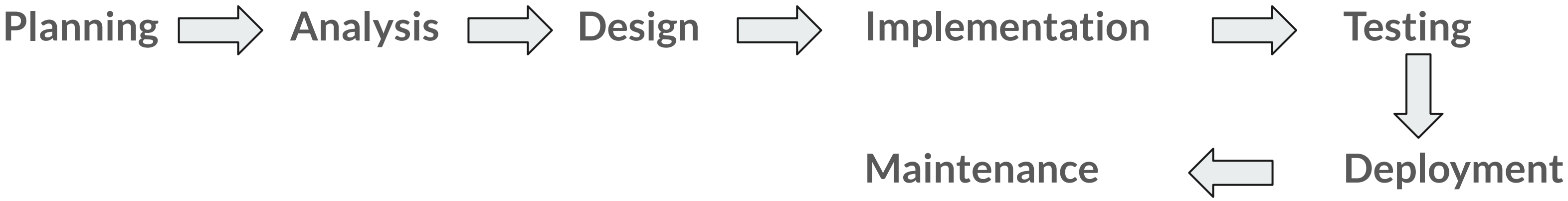
# What is SDLC?

**Definition**: The Software Development Life Cycle (SDLC) is a process used by software developers to design, develop, test, and deploy software.

**Why it's important:**

- Provides a structured framework for delivering high-quality software.
- Ensures all requirements are met efficiently.

**Key Phases:**



**Planning** → **Analysis** → **Design** → **Implementation** → **Testing** → **Deployment** → **Maintenance**

# SDLC Phases in Detail

**Planning**:

- Define the scope and objectives of the project.
- Example in electronics: Planning the integration of software for a smart home system.

**Analysis**:

- Gather and analyze requirements.
- Example: Identifying the type of sensors and data needed.

**Design**:

- Create architectural and detailed designs.
- Example: Designing the software architecture for data collection from Raspberry Pi sensors.

# SDLC Phases in Detail

**Implementation**:

- Develop the actual code.

**Testing**:

- Ensure the software meets requirements and works correctly.

**Deployment**:

- Deliver the software for use (e.g., flashing the firmware).

**Maintenance**:

- Regular updates, fixing bugs, or improving the software.

# Waterfall Model

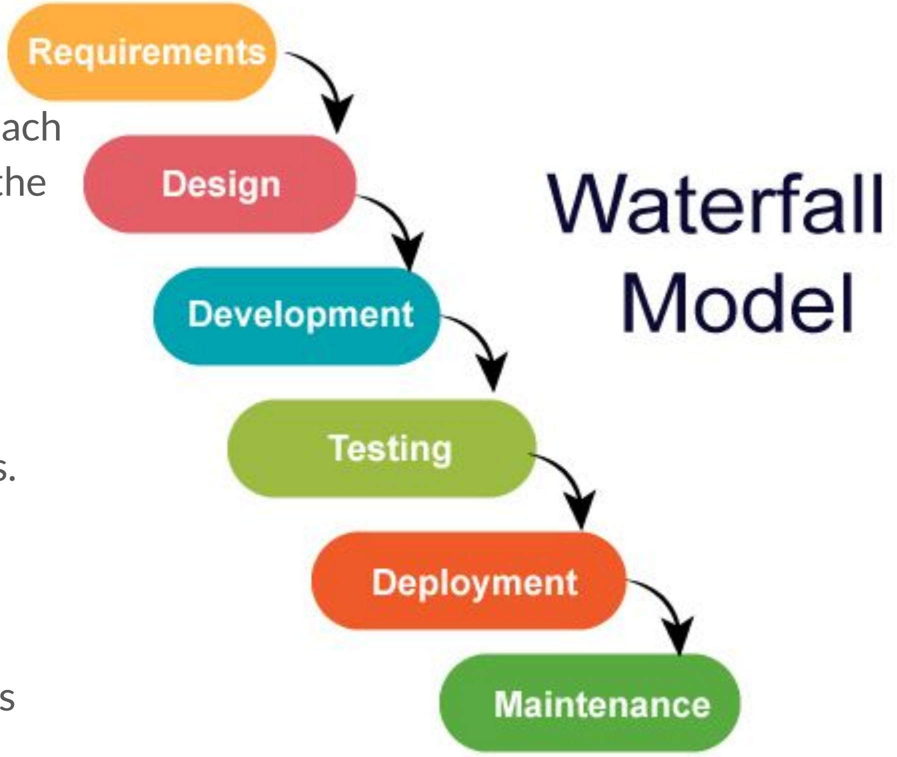
A linear and sequential SDLC model where each phase must be completed before moving to the next.

Advantages:

- Simple to manage and follow.
- Good for small, clearly defined projects.

Disadvantages:

- Inflexible for handling changes.
- Not ideal for complex, evolving projects like IoT systems.



# Agile Methodology

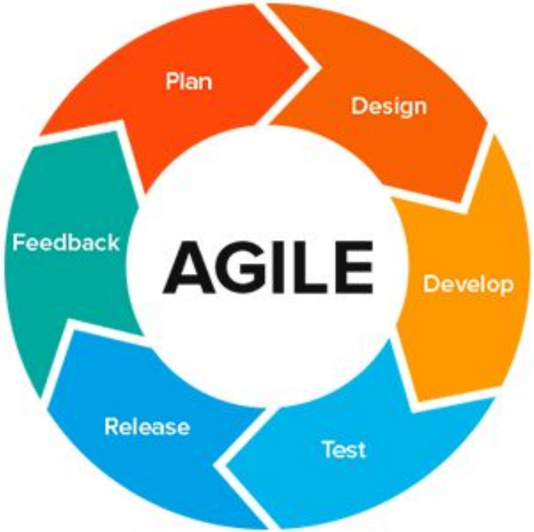
An iterative approach to software development that focuses on collaboration, customer feedback, and small, incremental releases.

Advantages:

- Flexibility to handle changes.
- Frequent iterations and faster delivery.

Disadvantages:

- Requires more involvement from stakeholders.
- More complex project management.



# SDLC in Electronics Projects

- **Planning**: Understanding hardware and software integration.
- **Design**: Designing both the hardware interface and the software.
- **Development**: Coding in Python, JavaScript, TypeScript, etc.
- **Testing**: Testing on Raspberry Pi or microcontrollers.
- **Deployment**: Flashing firmware, deploying software to IoT devices.
- **Maintenance**: Regular updates based on new hardware or software improvements.

# Exercise 1 – Mapping SDLC to an IoT Project

**Task:** Break down the SDLC phases for a project where you are developing software to control smart lights via a mobile app and a Raspberry Pi hub.

**Instructions**:

- Map each SDLC phase to specific tasks involved in developing the software and integrating it with the hardware.
- Identify key deliverables for each phase.

# Exercise 1 – Solution

- **Planning**: Defining the scope (control smart lights via an app).
- **Analysis**: Identifying hardware (lights, Raspberry Pi, mobile app) and user requirements (remote control, scheduling).
- **Design**: Designing the software architecture and communication between app and Raspberry Pi.
- **Implementation**: Writing code for both the mobile app and the Raspberry Pi's software.
- **Testing**: Ensuring lights respond correctly to app commands.
- **Deployment**: Installing the app and flashing the Raspberry Pi with the software.
- **Maintenance**: Adding new features, such as integrating motion sensors.

# Exercise 2 – Agile vs. Waterfall

**Task:** Compare the Agile and Waterfall methodologies for the following project: developing a home security system with cameras and sensors, controlled via a web interface.

**Instructions**:

- Prepare a short discussion comparing the two methodologies and recommend which one you would choose for this project and why.

# Exercise 3 – Creating a Project Backlog (Agile)

**Task:** Create a backlog for an Agile project that aims to develop software to control a home automation system.

**Instructions**:

- Write down at least 5 key user stories for this project (e.g., “As a user, I want to control the lights using my mobile phone”).
- Prioritize the user stories based on their importance and feasibility.
- Define acceptance criteria for at least 2 of the user stories.

# Exercise 3 – Solution

**User Stories:**

1. As a user, I want to control the lights using my mobile phone.
2. As a user, I want to schedule lights to turn on or off at specific times.
3. As a user, I want to receive alerts if a motion sensor is triggered.
4. As a user, I want to control the system from both mobile and desktop devices.
5. As a user, I want to monitor energy usage.

**Prioritization:**

1. Priority 1: Control lights via mobile.
2. Priority 2: Schedule lights.
3. Priority 3: Alerts via motion sensors

# Exercise 3 – Solution

**Acceptance Criteria (for user story 1 -** As a user, I want to control the lights using my mobile phone. )

- The user can control all lights in the house via a simple interface on the mobile app.
- The app provides real-time status (on/off) of each light.

# Exercise 4 – Implementing Iterative Development for an IoT Project

**Task:** Imagine you're developing an IoT-based smart home security system. Write a detailed plan for implementing the system using an Agile methodology, focusing on iterative development.

**Instructions**:

- Break the project into several sprints (at least 3), each focused on delivering a functional piece of the system (e.g., sensor integration, camera streaming, mobile app).
- Define deliverables for each sprint.
- For Sprint 1, create a detailed task breakdown (including coding, hardware integration, and testing).

# Exercise 4 – Solution

- **Sprint 1** (2 weeks): Sensor integration for motion detection and sending data to a server.

  **Tasks**:

  a. Set up motion sensors.
  b. Write software to capture sensor data.
  c. Implement communication between sensors and server.
  d. Unit testing for sensor readings.
- **Sprint 2** (3 weeks): Camera integration and live video streaming.
- **Sprint 3** (2 weeks): Mobile app interface to control the system and monitor live feeds.

# Exercise 5 – Testing Strategies for Embedded Systems

**Task:** Develop a detailed testing plan for an embedded system that controls a robotic arm. The system should have software for controlling movement based on sensor inputs.

**Instructions**:

- Define testing strategies for each SDLC phase, with a focus on embedded systems.
- Include unit testing, integration testing, system testing, and hardware-in-the-loop (HIL) testing.
- Explain how you would test the robotic arm in a real-world environment, considering both hardware and software components.

# Exercise 5 – Solution

- **Unit Testing**: Test individual functions for reading sensors and controlling motors.
- **Integration Testing**: Verify the communication between sensors and motors (sensor sends data, motor moves).
- **System Testing**: Test the entire system in a simulated environment before actual hardware deployment.
- **HIL Testing:** Connect the software to the real robotic arm and perform real-time testing to ensure the arm moves as expected based on sensor inputs.

# Exercise 6 – Risk Management in SDLC for Electronics

**Task:** Identify potential risks associated with developing software for an autonomous drone (software controls flight, cameras, and data collection) using an SDLC approach.

**Instructions**:

- Define at least 3 major risks at different stages of SDLC (e.g., design, development, deployment).
- Propose mitigation strategies for each risk.

# Exercise 6 – Solution

- **Risk 1 (Design Phase)**: Incorrect hardware-software integration leading to flight instability.

  **Mitigation**: Create detailed design documentation and conduct prototype tests with basic control systems.

- **Risk 2 (Development Phase)**: Real-time sensor data (e.g., from accelerometers) may not be processed quickly enough.

  **Mitigation**: Implement efficient algorithms and optimize code for real-time performance.

- **Risk 3 (Deployment Phase)**: Potential for software malfunction during a flight, causing crashes.

  **Mitigation**: Implement robust fail-safe mechanisms and conduct extensive real-world testing.

# Exercise 7 – Optimizing SDLC for Embedded Systems with Real-Time Constraints

**Task:** For a real-time embedded system (e.g., a real-time medical monitoring device), outline how the SDLC phases would be modified to ensure the system meets real-time constraints.

**Instructions**:

- Explain how design, development, and testing phases would be adjusted for performance optimization.
- Focus on minimizing latency and ensuring the system responds quickly to sensor inputs.

# Exercise 7 – Solution

- **Design Phase:** Use hardware with real-time capabilities (e.g., real-time operating systems, RTOS) and optimize the software design for low-latency communication.

- **Development Phase:** Implement multi-threading or parallel processing to handle concurrent tasks (e.g., data acquisition and processing).

- **Testing Phase:** Perform stress tests to ensure the system maintains performance under high loads and real-time constraints.

# Exercise 8 – Integrating DevOps into SDLC for IoT Systems

**Task:** DevOps plays an essential role in continuous integration/continuous deployment (CI/CD). Develop a DevOps pipeline plan for an IoT system that involves frequent software updates to devices in the field (e.g., smart meters, security cameras).

**Instructions**:

- Explain how continuous integration and deployment could be integrated into the SDLC.

- Include details on automating testing, deploying firmware, and monitoring devices in the field.

# Exercise 8 – Solution

CI/CD Pipeline:

- **Continuous Integration:** Automate testing for every code change (unit tests for sensor functions, integration tests for hardware).

- **Continuous Deployment:** Automate the deployment of updates to IoT devices (over-the-air, OTA updates) without disrupting service.

- **Monitoring:** Implement a monitoring system to track device health and software version in real-time, ensuring that devices update successfully.

# Recap & Key Takeaways

**Key Takeaways:**

- The SDLC provides a structured process for software development.
- Waterfall is a linear model suited for smaller projects with well-defined requirements.
- Agile is more flexible and ideal for projects where requirements evolve, like many IoT systems.
- SDLC principles can be applied to electronics and embedded systems development to ensure reliable, well-structured software.

**Next Session Preview:**

- In the next session, we will discuss Integrated Development Environments (IDEs) and their role in streamlining software development for electronics projects.