

Module Introduction & Software Development Life Cycle (SDLC)

CMP9134: Software Engineering

Dr Francesco Del Duchetto
Lecturer in Robotics and Autonomous Systems

University of Lincoln

6 February 2026

Today's Agenda

- 1 Module Overview
- 2 What is Software Engineering?
- 3 Software Development Life Cycle
- 4 Project Management
- 5 SDLC Models

Agenda

- 1 Module Overview
- 2 What is Software Engineering?
- 3 Software Development Life Cycle
- 4 Project Management
- 5 SDLC Models

Module Learning Outcomes

- **LO1:** Critically apply software engineering principles and techniques to software engineering problems, taking into account recent advances in the field.
- **LO2:** Analyse, develop and evaluate a software artefact from inception to deployment employing professional engineering approaches.
- **LO3:** Apply social, ethical and professional practices and critically analyse their applicability.

Main Topics Covered

- Project **management** principles and practices for software development.
- **Planning and specification** of software projects.
- **Software design** and architectural principles.
- **Software testing** and quality assurance.
- **Software maintenance** and evolution.

Module Delivery Team

Dr Francesco Del Duchetto

- Lecturer in Robotics and Autonomous Systems.
- Research: Human-robot interaction, AI & Robot learning, Robot vision and navigation.
- Office: INB3118. *Best if you contact me before showing up to my office!*
- Email: fdelduchetto@lincoln.ac.uk

Interactions

- **Lectures:** Fridays, 9:00 - 10:00 AM in MB3401.
- **Workshops:** Fridays, 1:30 - 3:30 PM in INB2102.
- **BlackBoard:** Use the Discussion Board for questions regarding material or logistics.

Module Syllabus (Weeks 1-5)

W	Date	Lecture Topic	Workshop
1	06/02/26	Intro & Software Development Life Cycle	Versioning control (GitHub)
2	13/02/26	Agile Frameworks	Agile Setup
3	20/02/26	Software Requirements	Requirement Analysis
4	27/02/26	Software Modelling & OOP	System Architecture
5	06/03/26	Pattern & Reuse	Structural Design
6	13/03/26	HCI & Design Thinking	UI Prototyping
7	20/03/26	Containerisation	Docker & devcontainers
8	27/03/26	Software Testing	Unit Testing
<i>Break - No Lectures!</i>			
12	24/04/26	DevOps & CI/CD	Test Driven Development
13	01/05/26	Continuous Deployment	Automatic Deployment
14	08/05/26	Evolution & Legacy	Refactoring
15	15/05/26	Legal, Ethical, Professional & Social Issues	Project support

Assessment

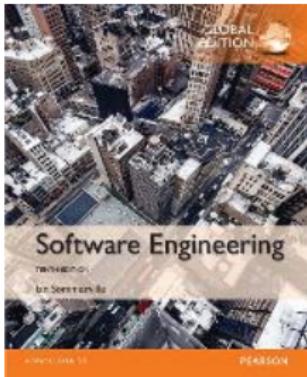
- **Assessment 1 (100%):** Design, develop, evaluate and document a comprehensive web application for remotely monitor and control an autonomous robot.
- **Deliverables:**
 - 1 Public GitHub repository (source code, documentation).
 - 2 Detailed project report (PDF).
 - 3 5-minute video demonstration.
- **Some notes:**
 - This is an individual assessment.
 - Documentation will be provided on BlackBoard *soon*.
 - Workshops tasks will be grounded on the project, so you can have a headstart on the development from the beginning.

Module Pre-requisites

- Basic computer and IT skills (e.g., file management, using a web browser, installing software).
- Understanding of fundamental programming concepts (e.g., variables, control structures, functions).
- Proficiency in coding in any language (e.g., Python, Java, C++).
- Familiarity with basic software development tools (e.g., text editors, IDEs).
- Being proactive and independent in learning new languages and tools as needed.

Reference books

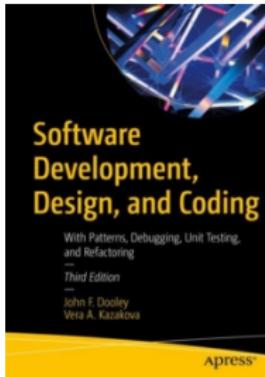
This is a *subset* of relevant books available at the University Library (physical or e-books):



Ian Sommerville

Software Engineering

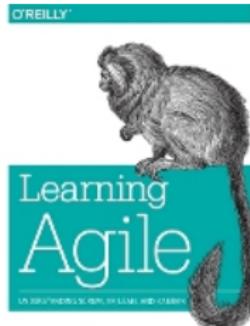
10th Edition, 2016



Dooley, John F.

Software Development, Design,

and Coding, 3rd Ed., 2024



Stellman & Greene

Learning Agile

1st Edition, 2014

Agenda

- 1 Module Overview
- 2 What is Software Engineering?
- 3 Software Development Life Cycle
- 4 Project Management
- 5 SDLC Models

Definition

- **Scientific method:** Discovery and organisation of knowledge by means of observation and experimentation.
- **Engineering:** The application of scientific methods to solving real-world problems.
- **Software Engineering:** Applies empirical and scientific approaches to solve practical problems in software.

“A Bad System Will Beat a Good Person Every Time” - W. Edwards Deming.

Scope

Aim

Making our software systems more efficient, scalable, reproducible, economic, accessible, maintainable, reliable, secure, etc.

Software engineering is **concerned with all aspects of software production**, from the early stages of *system specification* through to *maintaining* the system after it has gone into use

Why is it Important?

- Society relies on advanced software systems.
- We need to produce reliable and trustworthy systems economically.
- It is cheaper in the long run to use SE methods than to write programs as personal projects.
- **Diversity:** There is no “Silver Bullet” (universal technique) for all systems (e.g., Stand-alone vs Embedded vs Data collection).

Agenda

- 1 Module Overview
- 2 What is Software Engineering?
- 3 Software Development Life Cycle
- 4 Project Management
- 5 SDLC Models

The SDLC

The Software Development Life Cycle (SDLC) is the process of designing, building, and maintaining software applications.

- 1 Planning
 - 2 Analysis
 - 3 Design
 - 4 Implementation
 - 5 Testing & Integration
 - 6 Maintenance
- Managing complexity in a structured way.
 - Provides specific deliverables at each stage.
 - Frameworks (like Agile) emerge from best practices.

Core Process Activities

All software processes involve these four activities:

- 1 **Software specification:** Defining the software to be produced and constraints.
- 2 **Software development:** Designing and programming the software.
- 3 **Software validation:** Checking that it is what the customer requires.
- 4 **Software evolution:** Modifying software to reflect changing requirements.

Agenda

- 1 Module Overview
- 2 What is Software Engineering?
- 3 Software Development Life Cycle
- 4 Project Management
- 5 SDLC Models

Project Management in SDLC

Planning, organising, and controlling resources to achieve project goals.

- **Planning Phase:** Define goals, identify risks, estimate costs.
- **Analysis/Design:** Create Work Breakdown Structure (WBS) and schedules.
- **Implementation:** Monitor progress and manage stakeholders.
- **Maintenance:** Allocate resources for updates.

Planning: The Project Charter

A document outlining objectives, scope, stakeholders, and high-level requirements.

Components

- Project Background & Objectives
- Scope & Deliverables
- Stakeholders (Sponsors, Managers, Team)
- Assumptions, Constraints, and Risks

Planning: Risk Management

Identifying, assessing, and mitigating potential risks.

- **Monitoring:** Tracking progress vs planned performance.
- **Issue Tracking:** Using tools like Jira or GitHub Issues to track defects.
- **Burndown Charts:** Visualizing work completed vs work remaining.

Analysis/Design: Work Breakdown Structure (WBS)

A hierarchical decomposition of project deliverables into smaller, manageable components.

- Level 1: Product Vision
- Level 2: Major Deliverables/Phases
- Lower Levels: Tangible results requiring decomposition.

Analysis/Design: Scheduling and Estimation

- **Project Schedule:** Timeline identifying tasks, dependencies, and resources.
- **Tools:** Microsoft Project, GanttPRO, Trello (Kanban), Jira.
- **Software Metrics:**
 - Lines of Code (LOC)
 - Function Points (FP)
 - Story Points (Agile complexity metric)

Implementation/Testing: Monitoring Progress

- **Earned Value Analysis (EVA):** Combines scope, schedule, and cost to assess project performance.
- **Key Metrics:**
 - Planned Value (PV)
 - Earned Value (EV)
 - Actual Cost (AC)
- **Formulas:**
 - Schedule Variance (SV) = EV - PV
 - Cost Variance (CV) = EV - AC
 - Schedule Performance Index (SPI) = EV / PV
 - Cost Performance Index (CPI) = EV / AC

Implementation/Testing: Tracking Issues

- **Issue Tracking Tools:** Jira, GitHub Issues, Bugzilla.
- **Burndown Charts:** Visual representation of work completed vs work remaining.
- **Purpose:** Monitor progress, identify bottlenecks, and adjust plans.

Maintenance: Resource Allocation

There are 4 types of software maintenance: **corrective**, **adaptive**, **perfective**, and **preventive**.

- Allocate resources for bug fixes, updates, and enhancements.
- Use historical data to estimate maintenance effort.
- Plan for long-term support and scalability.

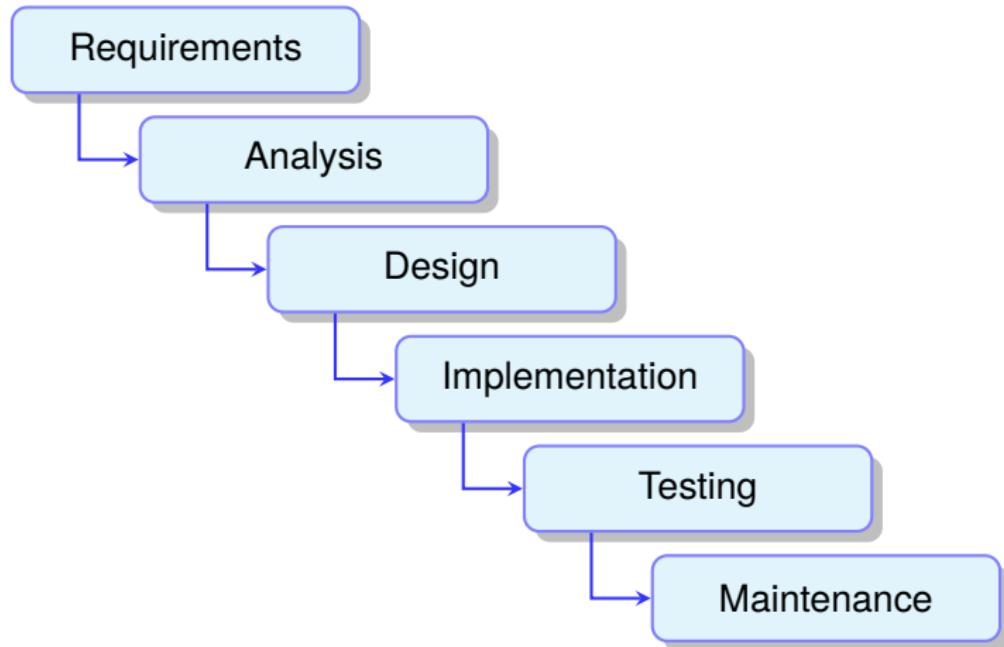
Agenda

- 1 Module Overview
- 2 What is Software Engineering?
- 3 Software Development Life Cycle
- 4 Project Management
- 5 SDLC Models

SDLC Models Overview

- Different projects require different approaches.
- Common models:
 - **Waterfall:** Linear, sequential.
 - **V-Model:** Emphasises verification and validation.
 - **Incremental:** Progressive development.
 - **Spiral:** Risk-driven.
 - **Agile:** Iterative, flexible (e.g., Scrum).

Waterfall Model



First formal Waterfall model is introduced in the 1970s. Emphasises a linear and sequential approach to software development.

Waterfall Model

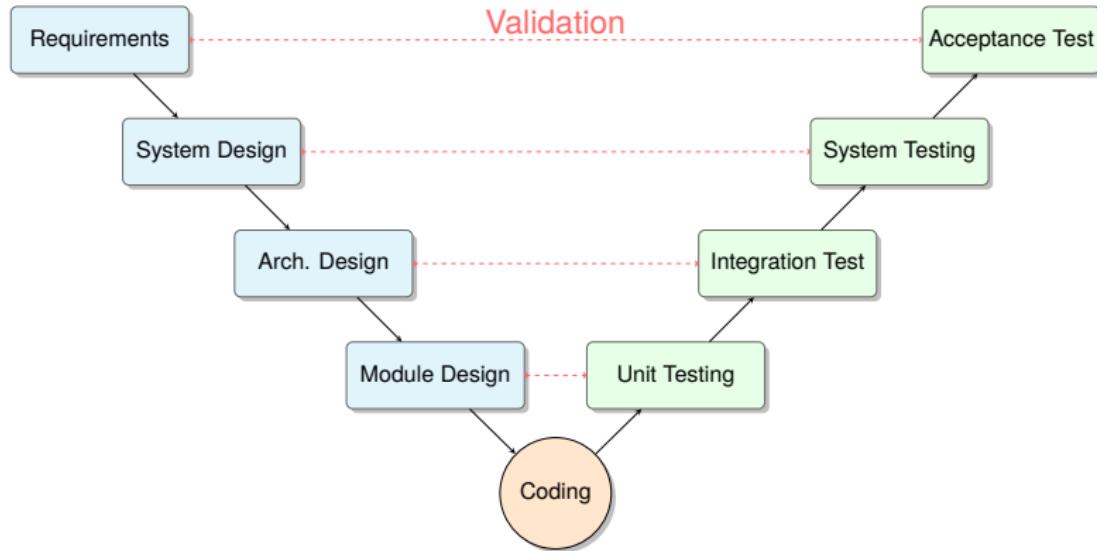
Qualities

- Sequential design process: easy to understand and manage.
- Each phase must be completed before the next begins: easy to organise and coordinate.
- Documentation-centric: suitable for projects with well-defined requirements.

Limitations

- Inflexible to changes: not ideal for projects with evolving requirements.
- Late testing phase: issues may be discovered late in the process.
- Not suitable for complex or long-term projects: lacks iterative feedback.

The V-Model



Introduced in the 1980s as an extension of the Waterfall model.
Emphasises verification and validation at each stage of development.

The V-Model

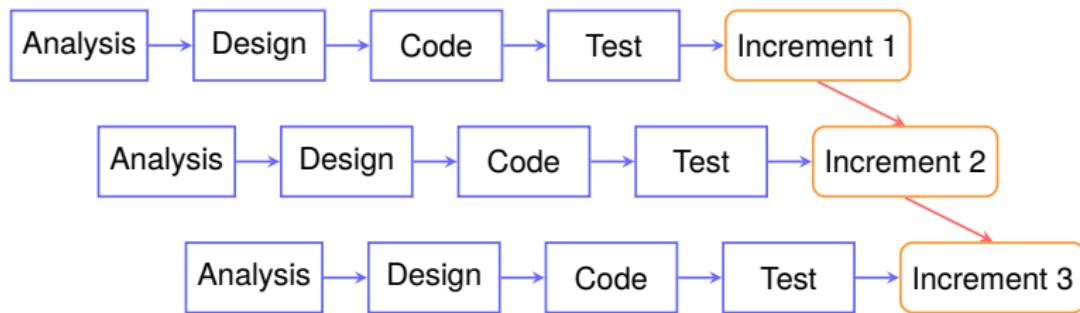
Qualities

- Focus on testing & quality: each development phase has a corresponding testing phase. *Solves waterfall's late testing issue.*
- Sequential and linear phases: easy to organise and coordinate.
- Well suited for projects with clearly defined requirements.

Limitations

- Inflexible to changes: not ideal for projects with evolving requirements.
- Assumes requirements are well understood upfront.
- Not suitable for complex or long-term projects: lacks iterative feedback.

Incremental Model



System is broken down into small, manageable portions. Functional software is produced early.

Incremental Model

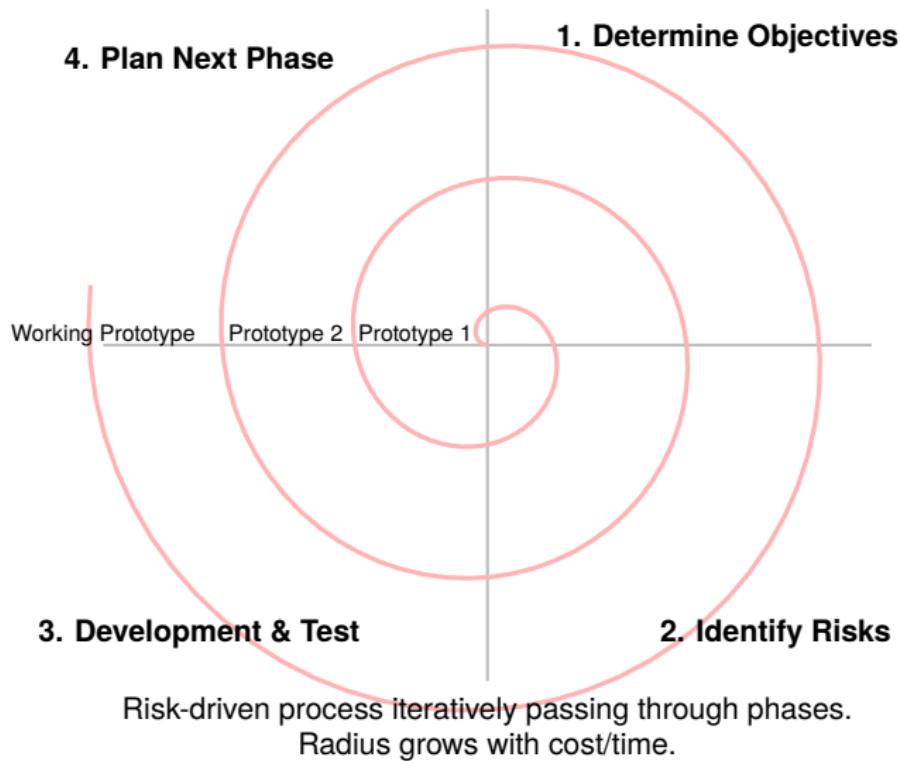
Qualities

- Each increment delivers a functional part of the system: allows for early delivery of useful software. *Solves waterfall's late delivery issue.*
- Allows for flexibility and adaptability as each increment can be adjusted based on feedback. *Solves waterfall's and v-model's inflexibility issue.*
- Easier to manage risks by breaking down the project into smaller parts.

Limitations

- Requires careful planning and design to ensure increments integrate well.
- May lead to higher overall costs due to repeated phases for each increment. Hence, not suitable for very small projects where overhead may outweigh benefits.

Spiral Model



Spiral Model

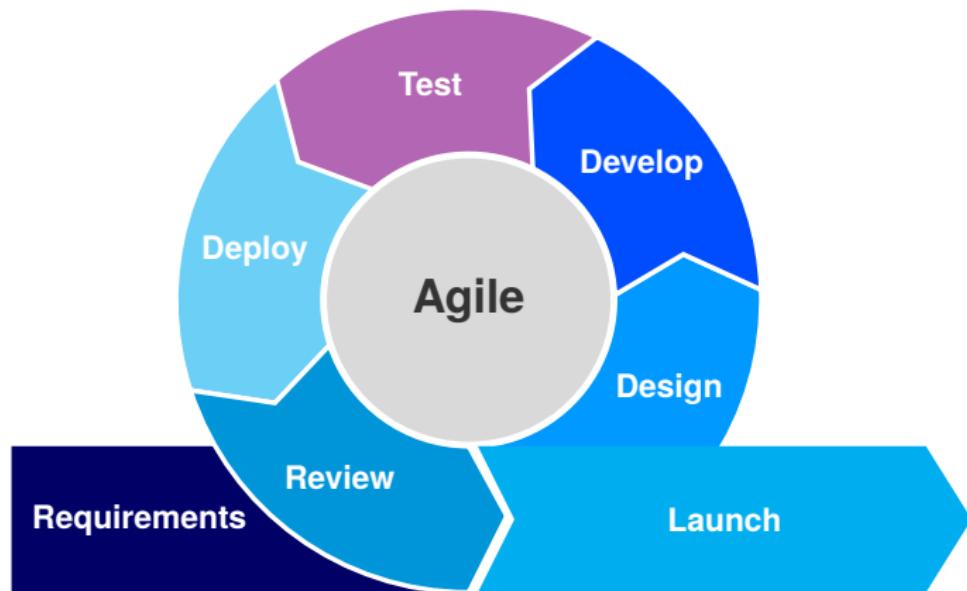
Qualities

- Focus on early risk identification and mitigation: suitable for high-risk projects.
- Allows for iterative development and refinement based on customer's feedback.
- Flexible and adaptable to changing requirements at various stage of development.

Limitations

- Requires careful planning and design to ensure increments integrate well.
- May lead to higher overall costs due to amount of risk assessments and iterations. Hence, not suitable for very small projects where overhead may outweigh benefits.
- Risk assessment and management require expertise and experience.

Agile Process



Commonly described as a set of principles or a phylosophy rather than a strict methodology. Introduced in the early 2000s as a response to the limitations of traditional SDLC models.

Agile Process

Qualities

- Focus on customer collaboration and responsiveness to change: suitable for projects with evolving requirements.
- Allows for iterative development and frequent delivery of working software.
- Emphasises teamwork, communication, and continuous improvement.

Limitations

- Requires active customer involvement throughout the project.
- May lead to scope creep if not properly managed.
- Less emphasis on documentation may lead to challenges in knowledge transfer and maintenance.

Next Steps

- **This Week's Workshop:** Learning Git (Version Control).
- **Next Week:** Agile Frameworks.

Any Questions?

fdelduchetto@lincoln.ac.uk