Software Engineering 2025–26

Lecture 3: Engineering Process

Or: How do we build it?

Prof. Robert Harle

Department of Computer Science and Technology
University of Cambridge

Easter Term

What is a Software Process?

Software Process

A structured set of activities required to develop a software system.

Every process involves:

- **Specification:** Defining what the system should do.
- **Design and Implementation:** Defining the organization of the system and implementing it.
- **Validation:** Checking that it does what the customer wants.
- **Evolution:** Changing the system in response to changing needs.

The Need for Process

Ad-hoc Development (Code-and-Fix)

- Works fine for 1 person writing a 500-line script.
- Disastrous for 50 people writing a 1,000,000-line banking system.
- Without a process, developers overwrite each other's code, duplicate effort, test inconsistently, and miss deadlines.
- We need a methodology.

The Waterfall Model

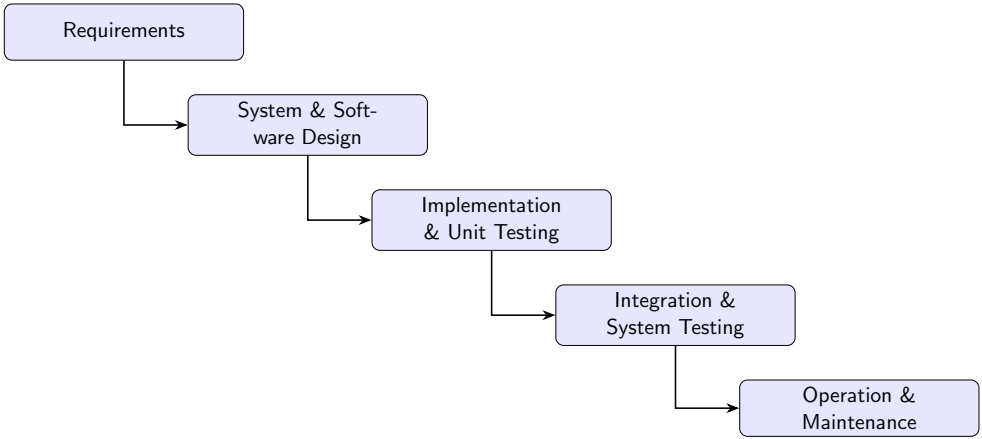
Origin: Winston W. Royce (1970)

- Borrowed heavily from civil and manufacturing engineering.
- You don't build the roof of a house before the foundation is poured and inspected.
- The idea: Software should be built in strict, sequential phases. You cannot move to the next phase until the current one is 100% complete and signed off.

Phases of the Waterfall Model

1. **Requirements Definition:** Write a 500-page specification document.
2. **System and Software Design:** Create exhaustive UML diagrams and database schemas.
3. **Implementation and Unit Testing:** Programmers finally write the code.
4. **Integration and System Testing:** Put the pieces together and see if it works.
5. **Operation and Maintenance:** Deploy to the customer.

Waterfall Model: Visual Flow



Strictly Sequential: No Going Back

The Illusion of Control

Why Management Loved Waterfall

- It looks great on a Gantt chart.
- It provides clear milestones (e.g., "Requirements Sign-off on Oct 1st").
- It creates a paper trail and shifts accountability (e.g., "The developers built exactly what was in the spec, so it's the analyst's fault").

Why Rigid Planning Fails in Software

The Fatal Flaw of Waterfall

- **Software is not a bridge.** If you realize halfway through building a bridge that it needs to be 10 miles to the left, you start over.
- In software, clients change their minds *constantly* because they don't know what they want until they see it.
- Waterfall assumes requirements are frozen in Step 1.
- By Step 5 (2 years later), the business needs have completely changed, and the software is obsolete on arrival.

The Cost of Change

Late Discovery = High Cost

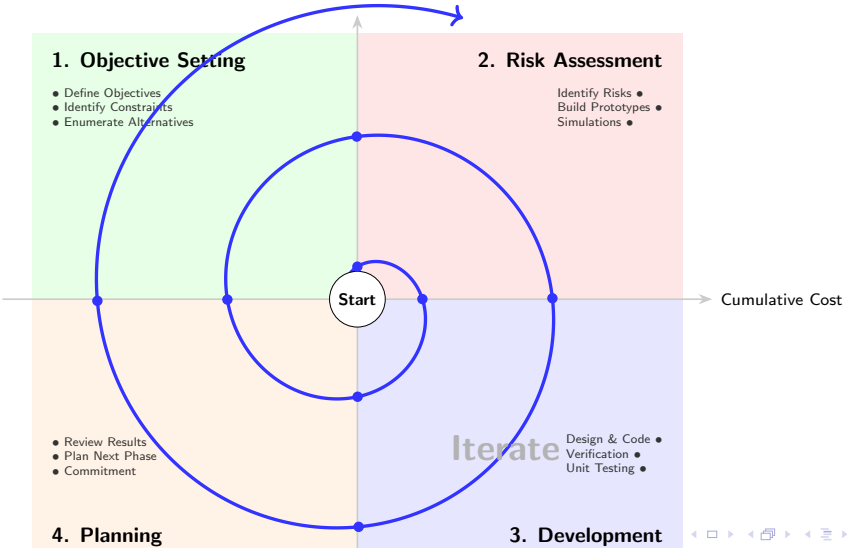
- In Waterfall, you don't do integration testing until phase 4.
- If you find a fundamental architectural flaw then, you have to rewrite the entire system.
- We needed a way to find flaws in week 2, not year 2.

The Spiral Model

Barry Boehm (1986)

- An early attempt to fix Waterfall.
- **Risk-Driven:** Instead of doing everything once, develop the system in a series of "spirals".
- Each loop in the spiral represents a phase of the process, but crucially incorporates **prototyping** and risk evaluation before committing to full development.

Spiral Model: Visual Loop



Iterative and Incremental Development (IID)

Breaking the Problem Down

- **Iterative:** Build a rough version of the whole system, then refine it. (Drafting an essay).
- **Incremental:** Build the system piece by piece, fully finishing one piece before moving to the next. (Building Lego).
- Modern software processes combine both: We build incrementally (feature by feature) and iteratively (improving those features over time).

The Shift in Mindset

The 1990s Software Crisis

- Waterfall and heavy, documentation-driven processes were stifling the software industry.
- The internet boom required companies to release software in months, not years.
- A new philosophy was needed.

The Context of the Late 90s

The Rebellion

- Developers were frustrated with "heavyweight" processes where they spent 50% of their time writing documentation that nobody read.
- Small teams were finding success using lightweight, highly adaptable methods.
- They needed a unified voice.

The Snowbird Meeting (2001)

The Birth of Agile

- 17 software pioneers met at a ski resort in Snowbird, Utah.
- They represented various lightweight methodologies (Extreme Programming, Scrum, DSDM).
- They drafted the **Agile Manifesto**.

The Agile Manifesto

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:"

1. **Individuals and interactions** over processes and tools
2. **Working software** over comprehensive documentation
3. **Customer collaboration** over contract negotiation
4. **Responding to change** over following a plan

Value 1: Individuals and Interactions

...over processes and tools.

- A fool with a tool is still a fool.
- The best bug-tracking software in the world cannot replace two engineers talking to each other at a whiteboard.
- Process should serve the team, the team should not serve the process.

Value 2: Working Software

...over comprehensive documentation.

- Documentation is important (as we discussed in Lec 2), but it is a means to an end.
- The primary measure of progress is **working software** delivered to the user.
- If you have a perfect spec doc and no code, you have zero value.

Value 3: Customer Collaboration

...over contract negotiation.

- In Waterfall, you negotiate a rigid contract at the start. When the client wants a change, lawyers get involved.
- Agile assumes the client is part of the development team.
- You show them working software every two weeks, and adapt based on their feedback.

Value 4: Responding to Change

...over following a plan.

- A plan is useless if the market has moved on.
- Agile embraces changing requirements, even late in development.
- Software architecture must be modular enough to allow for pivots without collapsing.

The 12 Principles of Agile

- **1. Satisfy the Customer:** Early and continuous delivery.
- **2. Welcome Change:** Even late in development.
- **3. Frequent Delivery:** Weeks rather than months.
- **4. Daily Cooperation:** Business and developers.
- **5. Trust & Support:** Motivated individuals.
- **6. Face-to-Face:** The most effective communication.
- **7. Working Software:** The primary measure of progress.
- **8. Sustainable Pace:** Constant indefinitely.
- **9. Technical Excellence:** Good design enhances agility.
- **10. Simplicity:** Maximizing work not done.
- **11. Self-Organizing:** Best designs emerge from teams.
- **12. Reflect & Adjust:** Regular process tuning.

What Agile IS NOT

Dispelling the Myths

- Agile is **not** an excuse to write zero documentation.
- Agile is **not** an excuse to skip planning. (Agile teams actually plan more frequently than Waterfall teams).
- Agile is **not** chaos. It requires immense discipline.

Premature Optimization

The Root of All Evil?

- "Premature optimization is the root of all evil (or at least most of it) in programming." – **Donald Knuth**
- **The Agile Perspective:** Don't build a hyper-scalable, distributed architecture for a product that doesn't have a single user yet.
- **YAGNI (You Ain't Gonna Need It):** Only build what is necessary for the current requirement.
- **Focus on Correctness first:** Get it right, then make it fast (if and only if performance becomes a bottleneck).

Extreme Programming (XP)

An early Agile methodology

- Created by Kent Beck in the 1990s.
- Idea: Take recognized good software engineering practices and turn them up to "extreme" levels.
- Focuses heavily on the technical execution of writing code.

XP Core Practices

- **Pair Programming:** Two developers, one keyboard. One types (driver), one reviews line-by-line (navigator).
- **Test-Driven Development (TDD):** Write the test before you write the code.
- **Continuous Integration:** Merge code into the main branch multiple times a day.
- *Note: While few companies do pure XP today, its technical practices have been universally adopted.*

What is Scrum?

The dominant Agile framework today.

- Unlike XP (which is technical), Scrum is a **management framework**.
- It doesn't tell you how to write code; it tells you how to organize the work.
- Based on Empiricism: Knowledge comes from experience, and making decisions based on what is known.

The Three Pillars of Scrum

- **Transparency:** The emergent process and work must be visible to those performing the work as well as those receiving the work.
- **Inspection:** Artifacts must be frequently inspected to detect undesirable variances.
- **Adaptation:** If an aspect of a process deviates outside acceptable limits, the process or materials must be adjusted as soon as possible.

Scrum Artifacts: The Product Backlog

The Master To-Do List

- An ordered list of everything that is known to be needed in the product.
- It replaces the 500-page Waterfall specification document.
- It is never complete; it constantly evolves as the product and environment change.
- Items at the top are highly detailed; items at the bottom are vague ideas.

Writing Backlog Items: User Stories

Focusing on Value Instead of writing "Add foreign key to users table," we write User Stories:

User Story Format

As a [type of user],
I want [some action or feature],
so that [business value or reason].

Example: "As a student, I want to filter my timetable by term, so that I only see my current lectures."

Acceptance Criteria & Definition of Done

How do we know a story is finished?

- **Acceptance Criteria:** Specific conditions the feature must meet to satisfy the user story. (e.g., "The filter dropdown must include Michaelmas, Lent, and Easter").
- **Definition of Done (DoD):** A universal team checklist. (e.g., "Code is reviewed, unit tests pass, documentation updated, deployed to staging").
- If it doesn't meet the DoD, it is not done. No exceptions.

Scrum Artifacts: Sprint Backlog & Increment

- **Sprint Backlog:** The set of Product Backlog items selected for the current Sprint, plus a plan for delivering them.
- **The Increment:** The sum of all the completed work during a Sprint.
- Crucially: The Increment must be in a **useable condition**. It must be potentially releasable to customers.

Scrum Events: The Sprint

The Heart of Scrum

- A Sprint is a fixed timebox (usually 2 weeks) during which a "Done", useable, and potentially releasable product Increment is created.
- During the Sprint, no changes are made that would endanger the Sprint Goal.
- If the business suddenly needs an urgent new feature, it goes into the Product Backlog for the *next* Sprint.

Scrum Events: Sprint Planning

Starting the Sprint

- The entire Scrum Team collaborates to plan the work for the upcoming Sprint.
- **Why** is this Sprint valuable? (Defining the Sprint Goal).
- **What** can be Done this Sprint? (Pulling items from the Product Backlog).
- **How** will the chosen work get done? (Breaking items down into technical tasks).

Scrum Events: Daily Stand-up (Daily Scrum)

The 15-Minute Sync A daily, time-boxed meeting for the Developers to inspect progress toward the Sprint Goal. Each member answers three questions:

1. What did I do yesterday?
2. What will I do today?
3. Are there any blockers impeding my progress?

Rule: It is NOT a status report for management; it is a synchronization meeting for the engineers.

Scrum Events: Sprint Review

Showing the Work

- Held at the end of the Sprint.
- The Scrum Team presents the results of their work to key stakeholders (clients, business owners).
- It is a working session, not a formal presentation.
- Feedback gathered here is used to update the Product Backlog for the next Sprint.

Scrum Events: Sprint Retrospective

Continuous Improvement

- Occurs after the Review and prior to the next Sprint Planning.
- The team discusses: What went well? What went wrong? How can we improve our process?
- Example: "Our builds were breaking frequently. Let's add a stricter linting check to our CI pipeline next Sprint."
- Process is not static; the team iterates on the *way* they work.

Estimation: Why humans are bad at time

The Estimation Problem

- If I ask you: "How many hours will it take to build a login page?"
- Junior Dev: "2 hours." (Forgets security, database, edge cases).
- Senior Dev: "3 weeks." (Over-engineers the solution).
- Humans are terrible at estimating absolute time (hours/days). We are much better at **relative sizing**.

Estimation: Story Points

Planning Poker

- Agile uses **Story Points** (often Fibonacci numbers: 1, 2, 3, 5, 8, 13) to measure complexity, effort, and risk—not time.
- "Task A is a 2. Task B feels about three times as complex, so it's a 5."
- **Planning Poker:** The team votes simultaneously. If Dev 1 votes an 8 and Dev 2 votes a 1, they discuss the discrepancy to uncover hidden complexities.

Process Roles: The Scrum Master

The Process Protector

- The Scrum Master is a **servant-leader**.
- They do not assign tasks or tell people how to code.
- Their job is to facilitate Scrum events, remove blockers (e.g., getting a slow server fixed), and protect the team from external interruptions.

Process Roles: The Product Owner

Maximizing Value

- The Product Owner (PO) represents the business and the customer.
- They are the sole person responsible for managing the Product Backlog.
- They decide *what* gets built next by prioritizing the backlog based on business value.

Process Roles: The Developers

The Execution Engine

- The people who do the work (programmers, designers, QA engineers).
- The team must be **cross-functional** (having all skills necessary to create a Product Increment).
- The team is **self-organizing**. Nobody (not even the Scrum Master) tells them how to turn Product Backlog items into Done increments.

Technical Program Manager (TPM)

The Cross-Team Coordinator

- A role common in large tech companies (Google, Meta).
- What happens when Team A needs an API built by Team B, but Team B is blocked by Team C?
- The TPM manages complex dependencies across multiple teams and aligns engineering work with broad company timelines.

When Scrum Fails: "Scrum-but"

Fake Agile

- "We do Scrum, *but* our Sprints are 6 weeks long."
- "We do Scrum, *but* management injects urgent tasks every day."
- "We do Scrum, *but* we skip retrospectives because we are too busy."
- This results in Waterfall disguised with Agile terminology, leading to burnout and failed projects.

Alternative: Kanban

Continuous Flow

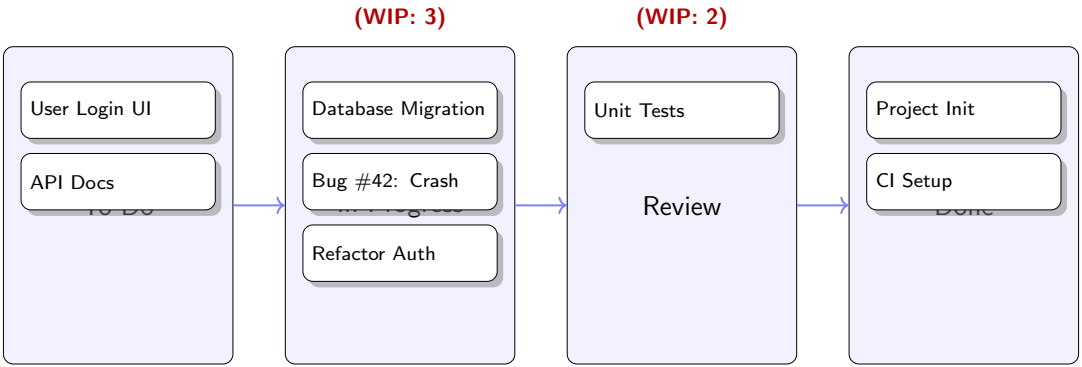
- Originates from Toyota manufacturing.
- Visualizes work on a board (To Do, In Progress, In Review, Done).
- No time-boxed sprints. Items are pulled continuously from a backlog.
- Best for operations/maintenance teams handling continuous incoming tickets.

Kanban Core Principle: WIP Limits

Work In Progress (WIP)

- Kanban enforces strict limits on columns. (e.g., "Only 3 items allowed in 'In Progress' at one time").
- If the limit is reached, developers must stop pulling new work and help their teammates finish existing tasks to clear the bottleneck.
- Optimizes for flow and fast delivery over batched features.

Kanban Board: Visual Flow



Pull-based system: Work flows to the right

Scaling Agile

What if you have 1,000 developers? Scrum works for teams of 5-9 people. How do we scale it?

- **SAFe (Scaled Agile Framework):** Highly structured, popular in enterprise and government. Often criticized as being "Waterfall in disguise."
- **LeSS (Large-Scale Scrum):** Keeps Scrum principles but applies them across multiple teams working on the same product backlog.
- **Spotify Model:** Guilds, Tribes, and Squads (though Spotify themselves admit they don't strictly use it anymore).

Shipping a Product: Convergence of Roles

It takes a village. To build modern software, you need:

- **Product Managers** to define the *Why*.
- **UX Designers** to craft the *Human Interface*.
- **Software Engineers** to architect and write the *Logic*.
- **QA/SDETs** to verify the *Quality*.
- **DevOps/SREs** to ensure the *Reliability*.

Summary of Lecture 3

- Waterfall is a rigid, document-heavy process that struggles with changing requirements.
- Agile values working software, adaptability, and human interaction.
- Scrum is an empirical framework using Sprints, Backlogs, and specific roles to deliver iterative increments.
- Story Points provide a relative measure of complexity over raw time estimation.
- Process requires discipline; skipping steps leads to "Fake Agile."