COMP433

Software Engineering

Software processes and models (part 2)

Hisham Ihshaish

Birzeit University

Semester 2 (Spring) 2025/2026 | Sections 5 & 6

What we will cover today

01 **The Agile Manifesto**
Values, principles, and what agile actually means

02 **Scrum**
Empirical process control, accountabilities, artefacts, and events

03 **Extreme Programming and Kanban**
Technical practices and flow-based management

04 **The spiral model and prototyping**
Risk-driven iteration and requirements exploration

05 **Choosing a process model**
A decision framework for your projects

Where we are: the process landscape

Previously, we examined three classical process models. Today we complete the picture with agile methods, risk-driven models, and a decision framework.

Thus far

Waterfall model
Sequential phases; complete requirements up front; late delivery; suits stable, regulated projects (Royce, 1970).

V-model
Extends waterfall by pairing each development phase with a test phase; tests are planned early but executed late.

Incremental development
Delivers software in usable slices; enables early feedback and adaptation; risk of structural degradation.

We'll add these…

Agile Manifesto, Scrum, XP, Kanban
Iterative, feedback-driven approaches grounded in empirical process control; engineering practices; flow management.

Spiral model and prototyping
Risk-driven iteration (Boehm, 1988); prototyping as a technique for reducing requirements uncertainty.

Choosing a process model
A decision framework mapping contextual factors (stability, criticality, team size) to model suitability.

The Agile Manifesto

A reaction to heavyweight processes

Why agile emerged

Historical context

By the late 1990s, many organisations used heavyweight processes: the Capability Maturity Model (CMM), full-scale Rational Unified Process (RUP) configurations, and US Department of Defence standards (DoD-STD-2167A) that required thousands of pages of documentation before a line of code was written. Projects took years to deliver, and by the time the software arrived, the business had moved on.

In February 2001, seventeen software practitioners met at a ski lodge in Snowbird, Utah. They represented diverse approaches -- Scrum, XP, Crystal, DSDM, Feature-Driven Development -- but shared a common frustration: existing processes valued compliance over results. The outcome was the Manifesto for Agile Software Development.

Not anti-process, but anti-bureaucracy

The signatories were not advocating chaos. They were experienced practitioners who had seen the damage caused by treating process as an end in itself. Their argument: process should serve the team, not the other way around. This distinction is frequently misunderstood.

The four values of 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:"

Individuals and interactions *A daily face-to-face conversation catches issues that a 50-page status report misses.*	over	processes and tools
Working software *A deployed prototype with three features teaches more than a 200-page specification.*	over	comprehensive documentation
Customer collaboration *Weekly demos with the registrar reveal misunderstandings that a signed-off SRS cannot.*	over	contract negotiation
Responding to change *When a competitor launches a new feature, the team re-prioritises mid-sprint rather than waiting for the next planning cycle.*	over	following a plan

"That is, while there is value in the items on the right, we value the items on the left more."

The twelve principles (1 of 2)

1

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 for the shorter timescale.

4

Business people and developers must work together daily throughout the project.

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.

The twelve principles (2 of 2)

7

Working software is the primary measure of progress.

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 maximising the amount of work not done -- is essential.

11

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

12

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

What agile is not

Common misconceptions corrected with what the Manifesto and its principles actually say.

"Agile means no documentation"

Principle 7 says working software is the PRIMARY measure, not the ONLY output. Teams still produce documentation -- but only what is useful, not what a process template demands.

"Agile means no planning"

Every Scrum sprint begins with Sprint Planning. Every XP iteration starts with the Planning Game. Agile replaces one big plan with many small plans that adapt to reality.

"Agile means no architecture"

Principle 9 explicitly calls for continuous attention to technical excellence and good design. Architecture emerges iteratively, but it is not ignored.

"Agile is just Scrum"

Scrum is one agile framework. Others include XP, Kanban, Crystal, DSDM, Lean, and SAFe. The Manifesto is a set of values; Scrum is one implementation.

Scrum

An empirical framework for product development

Scrum's empirical foundation

From the 2020 Scrum Guide

Scrum is founded on empiricism and lean thinking. Empiricism asserts that knowledge comes from experience and making decisions based on what is observed. Lean thinking reduces waste and focuses on the essentials.

Scrum rests on three pillars of empirical process control:

1. **Transparency**
   The process and work must be visible to those performing and receiving the work. Decisions are based on the perceived state of artefacts; if those artefacts are opaque, decisions are flawed.

2. **Inspection**
   The artefacts and progress are inspected frequently and diligently to detect potentially undesirable variances or problems. Inspection without adaptation is pointless.

3. **Adaptation**
   If any aspect of the process deviates outside acceptable limits, the process or its materials must be adjusted. Adjustment must be made as soon as possible to minimise further deviation.

Scrum accountabilities

The 2020 Scrum Guide defines three accountabilities (previously called "roles") within a Scrum Team.

1

Product Owner

Accountable for maximising the value of the product. Manages the Product Backlog: ordering items, ensuring clarity, and making trade-off decisions. One person, not a committee. If the team builds the wrong thing, the Product Owner is accountable.

2

Scrum Master

Accountable for the Scrum Team's effectiveness. Coaches the team in self-management and cross-functionality. Removes impediments. Serves the organisation by helping it understand and adopt Scrum. Not a project manager or a team lead.

3

Developers

The people who create any aspect of a usable Increment each Sprint. Self-managing: they decide who does what, when, and how. Cross-functional: collectively possessing all skills needed. Accountable for creating a plan for the Sprint (Sprint Backlog), instilling quality (Definition of Done), and adapting daily.

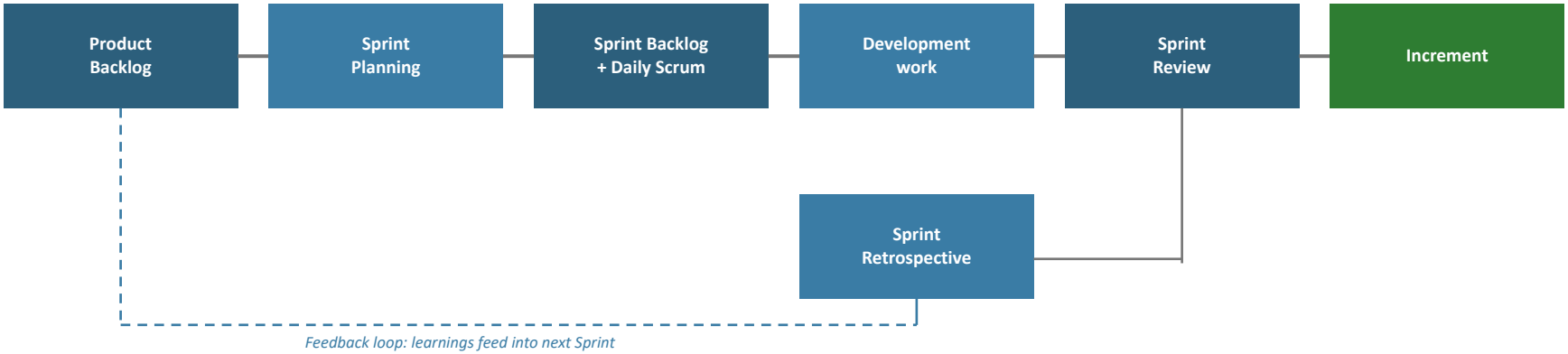
Scrum artefacts and their commitments

Each Scrum artefact contains a commitment that provides transparency and a focus for measuring progress.

Artefact	Purpose	Commitment	Meaning
Product Backlog	An emergent, ordered list of everything needed to develop or improve the product. The single source of work for the Scrum Team.	**Product Goal**	The long-term objective. Each Sprint moves the product closer to the current Product Goal.
Sprint Backlog	The set of Product Backlog items selected for the Sprint, plus the plan for delivering them and achieving the Sprint Goal.	**Sprint Goal**	The single objective for the Sprint. Provides coherence and focus, guiding trade-off decisions during the Sprint.
Increment	A concrete, usable stepping stone toward the Product Goal. Multiple Increments may be created within a Sprint.	**Definition of Done**	A formal description of the quality standard. Work that does not meet the Definition of Done cannot be released.

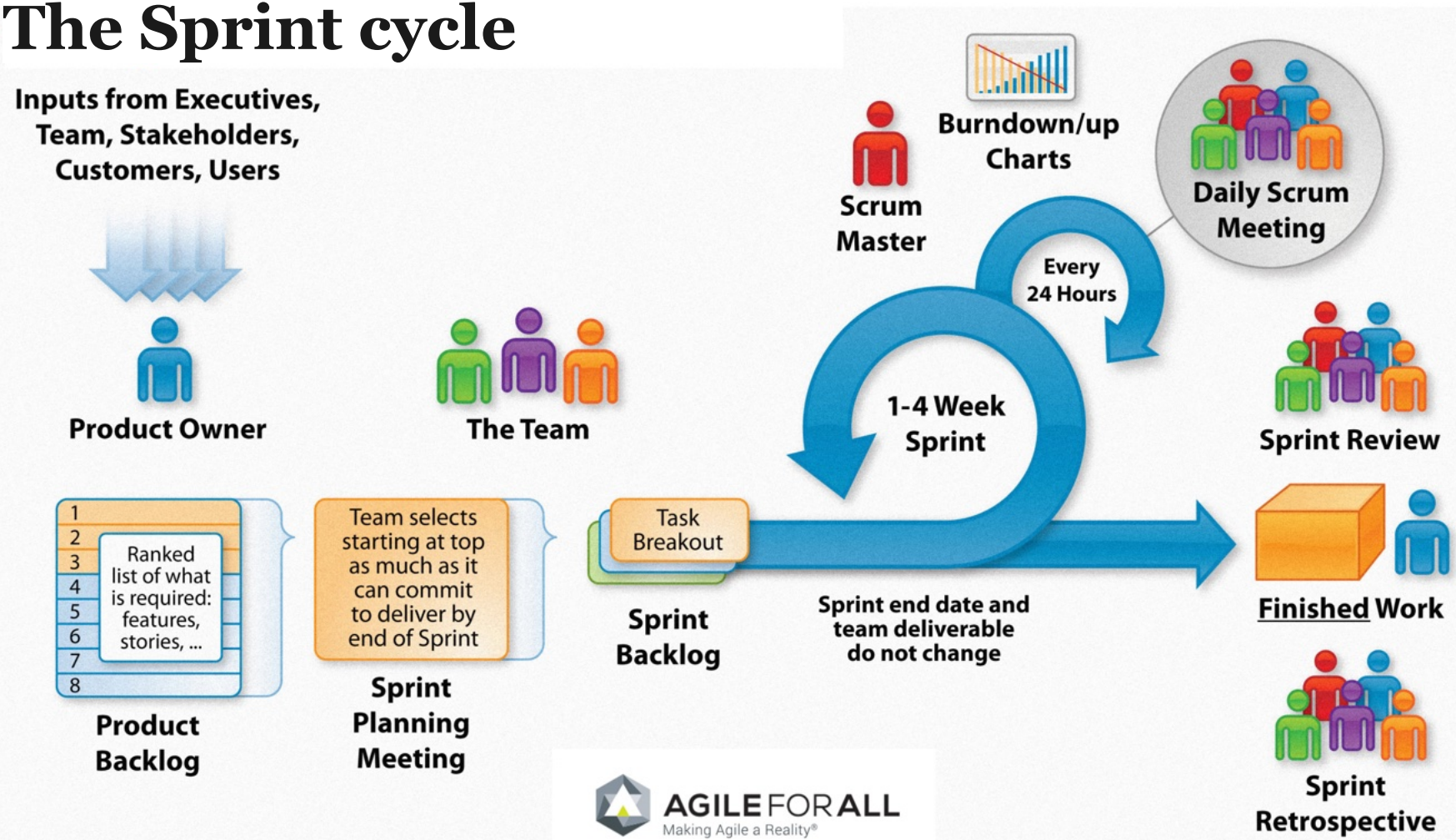
The commitment concept was introduced in the 2020 Scrum Guide. It ensures each artefact has a measurable target against which progress can be inspected.

The Sprint cycle



Product Backlog	Sprint Planning	Daily Scrum	Sprint Review	Sprint Retrospective
An ordered list of everything the product needs. The Product Owner maintains and prioritises it continuously.	The team selects backlog items for this Sprint and creates a Sprint Goal. They plan how to deliver the selected work. Output: Sprint Backlog.	A 15-minute daily event where Developers inspect progress toward the Sprint Goal and adapt their plan for the next 24 hours. Not a status report.	The team demonstrates the Increment to stakeholders. They discuss what was done and what to do next. The Product Backlog is updated based on feedback.	The team reflects on the Sprint: what went well, what could improve, and what actions to take. This is where continuous improvement happens.
	Timebox: 8h / month	*Timebox: 15 min*	*Timebox: 4h / month*	*Timebox: 3h / month*

The Sprint cycle



A sprint in practice: Birzeit library system

A team of four is building a book borrowing system for the Birzeit University library. Sprint length: 2 weeks.

Sprint Planning (Monday, 2 hours)	Product Owner presents the highest-priority items. Team selects: "search catalogue by title" and "view book availability". Sprint Goal: "Students can find and check availability of any book."
Days 1--4 (Development)	Developers code the search feature. Daily Scrums surface a database indexing issue on day 2. The team adapts: one developer optimises queries while others continue UI work.
Days 5--8 (Development)	Availability display completed. A Developer raises a concern at Daily Scrum: the UI is confusing. Team decides to simplify the layout rather than add more features.
Day 9 (Sprint Review)	Team demos to the librarian. She says: "Search works well, but I need to see which branch has the book, not just a yes/no." This feedback goes into the Product Backlog for Sprint 2.
Day 10 (Retrospective)	Team reflects: "We should sketch the UI before coding next time." Action item added: include a UX sketch step in the Definition of Done for UI stories.

Extreme Programming and Kanban

Technical practices and flow-based management

Extreme Programming: core practices

Technical practices

Test-driven development (TDD)
Write the test before the code. The Red-Green-Refactor cycle ensures every piece of code has automated tests from birth.

Pair programming
Two developers, one keyboard. The driver writes code; the navigator reviews, thinks ahead. Studies (Williams et al., 2000) show pairs produce fewer defects at only 15% higher cost.

Refactoring
Continuously improving code structure without changing its behaviour. Prevents technical debt from accumulating across iterations.

Continuous integration (CI)
Every developer integrates code into the shared repository at least once a day. Automated builds and tests run on each commit.

Simple design
Implement the simplest thing that could work. Do not build for hypothetical future requirements (YAGNI: You Aren't Gonna Need It).

Management practices

Small releases
Release to production frequently. Each release delivers business value and provides feedback.

Planning game
Business (customer) decides scope and priority; developers **estimate** effort. Negotiation replaces dictation.

On-site customer
A real customer sits with the team, available for questions. This eliminates the latency of email and meetings.

Collective code ownership
Any developer can change any code. Nobody owns a module. This spreads knowledge and reduces bus-factor risk.

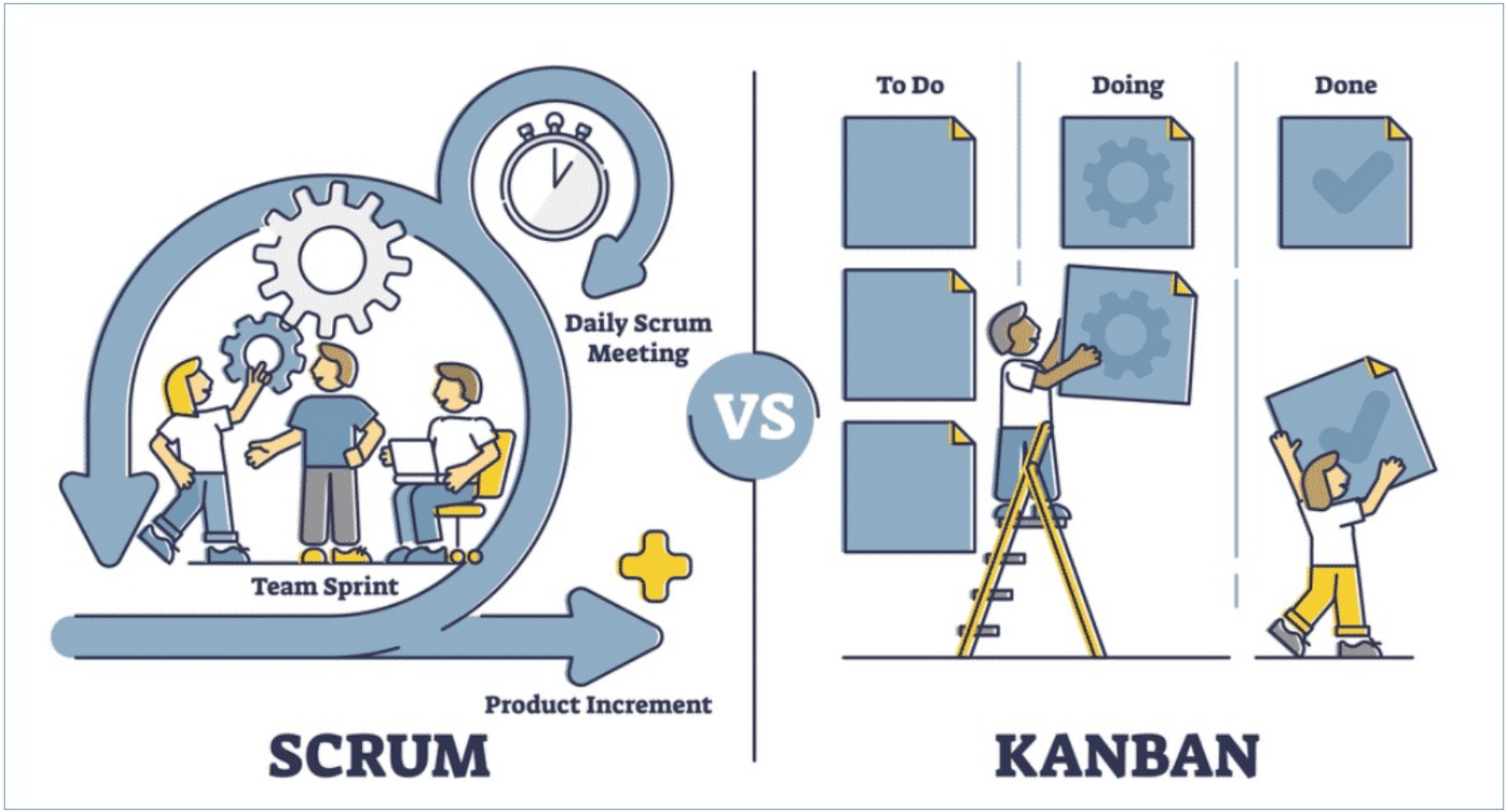
Sustainable pace (40-hour week)
No overtime as a sustained practice. Tired developers write buggy code. Mirrors Agile Principle 8.

The Red-Green-Refactor cycle (TDD)

TDD is not just a testing technique. It is a design discipline that forces you to specify behaviour before implementing it.

*Example: "Students can search books by title." Write a test asserting that searching 'Data Structures' returns matching books. Run it (Red). Implement the search function (Green). Extract a reusable query builder (**Refactor**).*

Kanban


Daily Scrum Meeting
Team Sprint
Product Increment
SCRUM
VS
To Do
Doing
Done
KANBAN

Scrum, XP, and Kanban compared

Dimension	Scrum	XP	Kanban
Cadence	Fixed-length Sprints (1--4 weeks)	Short iterations (1--2 weeks)	Continuous flow; no fixed iterations
Roles / accountabilities	Product Owner, Scrum Master, Developers	Customer, Coach, Programmers	No new roles prescribed; start with current structure
Change policy	No scope changes during a Sprint	Stories can be swapped if not started	Pull new work any time capacity allows
Engineering practices	Not prescribed (teams choose)	TDD, pair programming, CI, refactoring (prescribed)	Not prescribed
Main strength	Team rhythm, accountability, inspect-and-adapt cycle	Technical excellence, code quality, defect reduction	Flow optimisation, bottleneck visibility, low adoption cost
Common pairing	*Scrum + XP practices is the most popular combination.*		

The spiral model and prototyping

Risk-driven iteration and requirements exploration

Boehm's spiral model (1988)

1. Determine objectives

Identify objectives, alternatives, and constraints for this iteration. Define what the loop should achieve.

2. Evaluate risks

Analyse alternatives and identify risks. Resolve risks through prototyping, simulation, benchmarking, or other risk-reduction activities.

3. Develop and validate

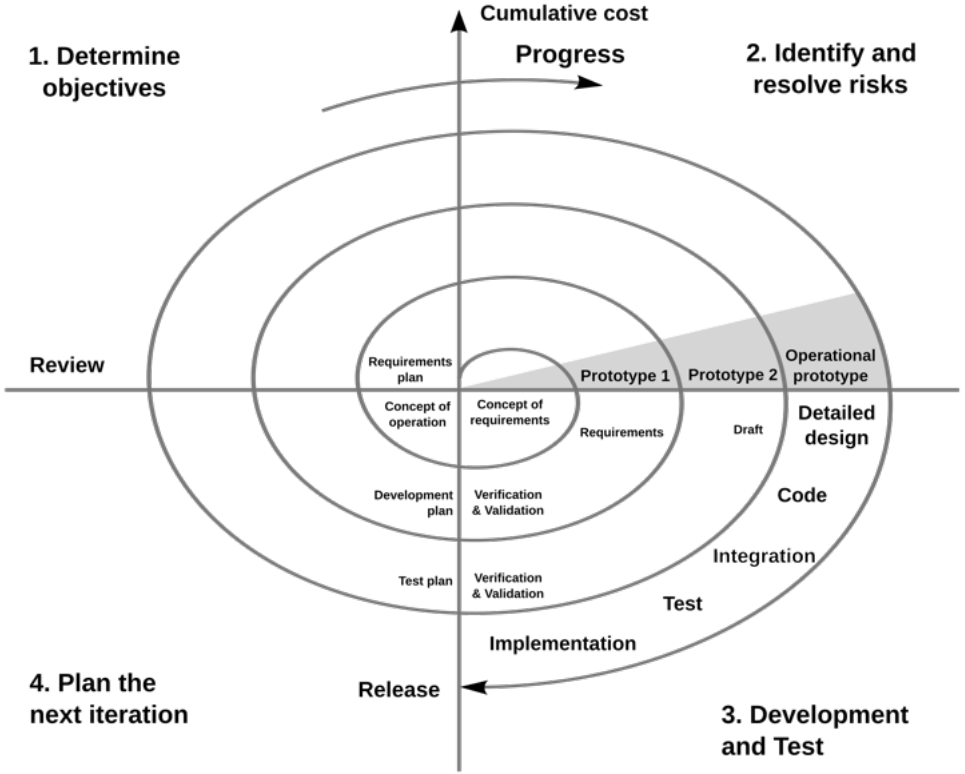
Develop the product for this loop. The development model depends on the risk assessment: could be waterfall, incremental, or prototyping.

4. Plan next iteration

Review results with stakeholders. Plan the next iteration: set scope, schedule, and resources. Decide go/no-go.

Each loop through the four quadrants is one iteration. The spiral expands outward as the project grows. Risk assessment at every loop is the defining characteristic.

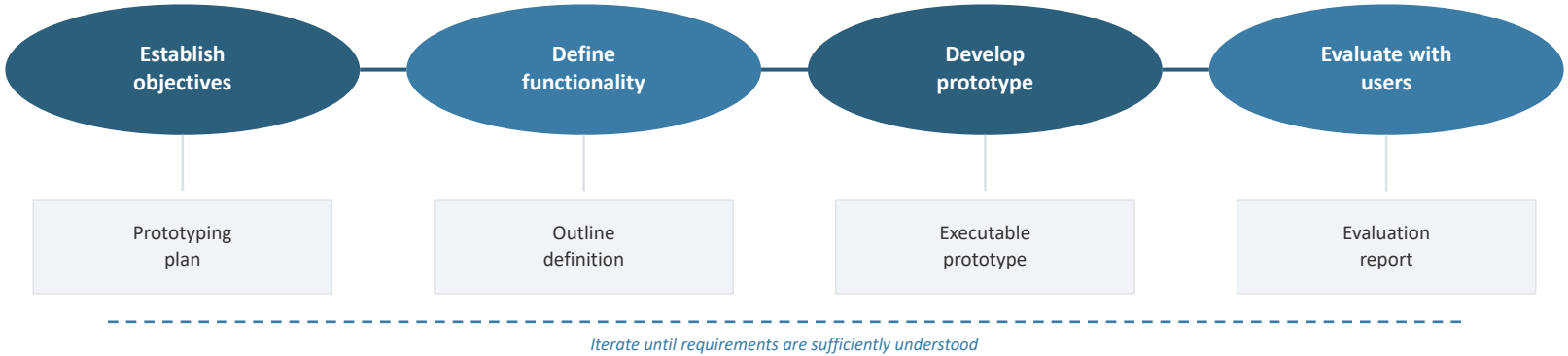
Boehm's spiral model (1988)



Excerpt from Wikipedia. Each loop through the four quadrants is one iteration. The spiral expands outward as the project grows. Risk assessment at every loop is the defining characteristic.

Software prototyping

Definition (Sommerville) *A prototype is an initial version of a software system used to demonstrate concepts, try out design options, and find out more about the problem and its possible solutions.*



Throw-away prototype
Built quickly to explore requirements. Discarded after evaluation. The real system is built separately with production quality.

Evolutionary prototype
Built with production quality from the start. Gradually refined into the final system through successive evaluations. Lower risk of rework.

Choosing a process model

Matching the model to the project context

A decision framework

Use this matrix to match contextual factors to process models. There is rarely a single right answer.

Factor	Waterfall	V-model	Incremental	Scrum	Kanban	Spiral
Stable requirements	Strong	Strong	Moderate	Moderate	Moderate	Moderate
Evolving requirements	Poor	Poor	Strong	Strong	Strong	Strong
High criticality	Strong	Strong	Moderate	Moderate	Moderate	Strong
Customer availability	Low need	Low need	Moderate	High need	Moderate	Moderate
Large team (>20)	Moderate	Moderate	Moderate	Needs scaling	Strong	Moderate
Regulatory compliance	Strong	Strong	Moderate	Moderate	Moderate	Strong
High technical risk	Poor	Poor	Moderate	Strong	Moderate	Strong

Green = strong fit; amber = possible with adaptation; red = poor fit. Most projects blend models.

Discussion: your group project process

Using what you have learned across previous lectures, discuss the following with your project group:

1. Are your project requirements stable, or do you expect them to evolve as you learn more?
2. How often can you get feedback from a user or stakeholder? Weekly? After each submission?
3. What is the biggest risk in your project? Technical complexity? Unclear requirements? Team coordination?
4. Based on your answers, which process model (or combination) would you adopt? Justify your choice.

Your project phase 1 submission will include a process justification.
You will need to explain which process model you chose, why it suits your project context, and how you will apply it across the semester. Start thinking about this now.

Take 5 minutes. Be prepared to share one key insight with the class.

Summary: Block 2 complete

The Agile Manifesto (2001) prioritises individuals, working software, customer collaboration, and responsiveness to change -- without rejecting process, documentation, contracts, or plans.

Scrum is grounded in empirical process control (transparency, inspection, adaptation). It defines three accountabilities, three artefacts with commitments, and five events.

XP prescribes engineering practices (TDD, pair programming, CI, refactoring) that complement Scrum's management framework. Most agile teams combine elements of both.

Kanban manages flow through visualisation and WIP limits, justified by Little's Law. It requires no new roles and starts with your current process.

Boehm's spiral model (1988) introduced risk-driven iteration. Prototyping reduces requirements uncertainty within any process model.

No single process model fits all projects. The right choice depends on requirements stability, criticality, team size, customer availability, and technical risk.

Next session

Lecture 4: Requirements engineering (part 1 of 3)

- What are requirements? Why do they matter?
- Types of requirements: functional, non-functional, and domain
- Stakeholder identification and analysis
- The requirements engineering process
- Running example: a university library borrowing system

Preparation

Read Sommerville (R2) Chapter 4, focusing on sections 4.1--4.3 (requirements types and the requirements engineering process). Also read Bruegge & Dutoit (R1) Chapter 4, sections on requirements elicitation. Ensure your project group is formed and registered.

References

Sommerville, I. (2016). Software Engineering. 10th Edition. Pearson.

Bruegge, B. & Dutoit, A. (2013). Object-Oriented Software Engineering Using UML, Patterns, and Java. 3rd Edition. Prentice Hall.

Schwaber, K. & Sutherland, J. (2020). The Scrum Guide. scrumguides.org.

Beck, K. (1999). Extreme Programming Explained: Embrace Change. Addison-Wesley.

Anderson, D. (2010). Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press.

Boehm, B. (1988). A Spiral Model of Software Development and Enhancement. IEEE Computer, 21(5), 61--72.

Beck, K. et al. (2001). Manifesto for Agile Software Development. agilemanifesto.org.

Pressman, R. & Maxim, B. (2020). Software Engineering: A Practitioner's Approach. 9th Edition. McGraw-Hill.

Williams, L. et al. (2000). The Collaborative Software Process. IEEE Software, 17(5), 19--25.

Little, J.D.C. (1961). A Proof for the Queuing Formula L = lambda W. Operations Research, 9(3), 383--387.

IEEE Computer Society (2024). SWEBOK v4. IEEE.