Agile Methodology — Scrum, Kanban & GitHub Projects

Web Technologies — Lecture 3

Masoud Hamad

State University of Zanzibar (SUZA)

Semester II, 2025/2026

Outline

Methodology vs Architecture vs Patterns

- **Architecture** (last lecture) — *what* we build
- **Design Patterns** — *how* we structure code
- **Methodology** (today) — *how the team works* to build it

Why it matters

Two teams with the same architecture can ship at very different rates. The difference is process, communication, and feedback loops.

Waterfall (the old way)

- One sequential pass; finish each phase before the next
- Big requirements doc up front, then no changes
- **Problems:** users only see software at the end; learning comes too late; cost of change is huge

Agile Manifesto (2001)

We value...

- **Individuals & interactions** over processes & tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Note: "over" — both sides have value. We just value the left more.

12 Principles (Highlights)

- Satisfy the customer through **early and continuous** delivery
- Welcome changing requirements, even late
- Deliver working software **frequently** (weeks, not months)
- Business and developers work together **daily**
- Build projects around motivated individuals — trust them
- Working software is the primary measure of progress
- Continuous attention to technical excellence
- Reflect regularly and **adjust**

Scrum — Three Roles

- **Product Owner** — owns the backlog, decides priorities (the "what" / "why")
- **Scrum Master** — facilitates, removes blockers, protects the team
- **Development Team** — 3–9 people, cross-functional, decides "how"

Key idea: the team is **self-organising**. No traditional "boss" on the team.

Scrum — Three Artefacts

- **Product Backlog** — prioritised list of all desired features (user stories)
- **Sprint Backlog** — subset committed for the current sprint
- **Increment** — working software delivered at end of sprint — must be *potentially shippable*

Scrum — Five Events (per sprint)

Sprint = fixed time-box, typically 1–4 weeks (most teams: 2 weeks)

1. **Sprint Planning** — pick stories, define a sprint goal (4 hrs / 2-week sprint)
2. **Daily Stand-up** — 15 min, three questions:
   - What did I do yesterday?
   - What will I do today?
   - What's blocking me?
3. **Sprint Review** — demo working software to stakeholders
4. **Sprint Retrospective** — "what worked / didn't / will change"
5. **Backlog Refinement** — ongoing grooming of upcoming stories

User Stories

A user story captures a feature from the user's perspective.

```
As a   <type of user>
I want <goal>
So that <benefit>
```

Example:

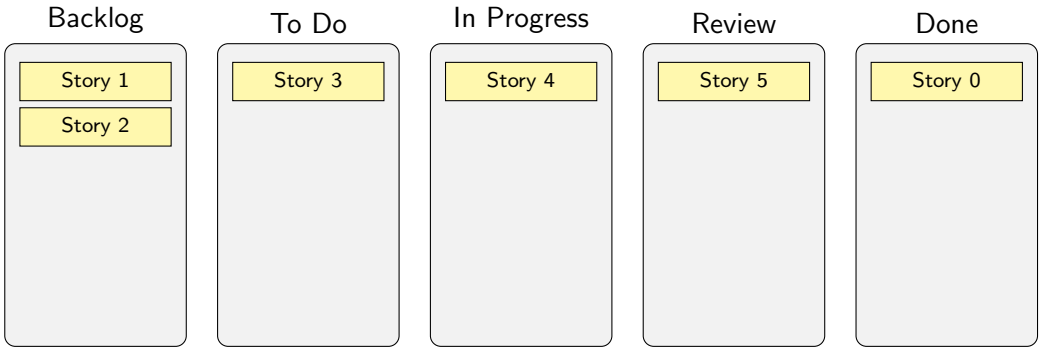
```
As a registered student
I want to download last semester's results as PDF
So that I can share them with my sponsor
```

Add **Acceptance Criteria** that spell out when "done" is achieved.

Story Points & Estimation

- Estimate **relative size**, not hours
- Common scale: Fibonacci 1, 2, 3, 5, 8, 13, 21
- Use **Planning Poker** — everyone picks a number, discuss outliers
- **Velocity** = points completed per sprint — predicts future capacity
- Stories $>$ 13 points should be **split**

Kanban — Visualise the Flow



- **Visualise** all work on a board
- **Limit Work In Progress (WIP)** per column
- **Pull, don't push** — next free worker pulls the next card
- No fixed sprints — continuous flow

Kanban Metrics

- **Lead time** — backlog $\rightarrow$ done (customer-facing)
- **Cycle time** — in-progress $\rightarrow$ done (team-facing)
- **Throughput** — cards completed per week
- **WIP** — cards in progress at one moment

Lower WIP $\Rightarrow$ shorter cycle time (Little's Law).

Scrum vs Kanban

	Scrum	**Kanban**
Cadence	Fixed sprints (1–4 wks)	Continuous flow
Roles	PO, SM, Dev team	No prescribed roles
Planning	Sprint planning	On-demand
Change	Avoid mid-sprint	Anytime
WIP limits	Implicit (sprint scope)	Explicit per column
Best for	New products	Ops, support, steady flow

XP — Extreme Programming

Engineering practices that often pair with Scrum/Kanban:

- **Pair programming** — two devs, one keyboard
- **TDD** (Test-Driven Development) — write the test first
- **Continuous integration** — merge to main every day
- **Refactoring** — continuous improvement of the code
- **Small releases** — ship something every week
- **Collective ownership** — anyone can change anything

Common Anti-Patterns

- **Scrum-but** — "We do Scrum, but no retros / no PO / 6-week sprints..."
- **Story stuffing** — pulling more in mid-sprint
- **Standups as status reports** — to manager, not to teammates
- **No definition of done** — "done" means different things
- **Estimates as deadlines** — crushing the team
- **No retrospective action items** — you didn't actually retrospect

Git — The Core Workflow

```
git clone <repo>                          # get the code
git checkout -b feat/login                # branch per story
# ...edit, save...
git add .  ; git commit -m "Add login form"
git push origin feat/login
# Open a Pull Request on GitHub
```

Branch names: `feat/`, `fix/`, `chore/`, `docs/`.

Pull Requests — Code Review

A PR is a **conversation about a change** before it lands on `main`.

- Author opens PR with description, screenshots, test plan
- At least one teammate reviews and approves
- CI runs tests automatically
- When green + approved, **squash and merge**
- Delete the branch

PRs are **small**: ∼200–400 lines is ideal. Bigger = harder to review well.

Conventional Commits

```
<type>(<scope>): <short description>

feat(auth): add password reset flow
fix(cart): prevent negative quantities
docs(readme): clarify Docker setup
chore(deps): upgrade React to 18.3
test(api): cover 401 path on /students
```

Helps with changelogs, semantic versioning, and reviewing git history.

GitHub Projects — Free Built-in Tool

- Built into every GitHub account — zero setup
- Cards link to issues and PRs — traceability for free
- Multiple views: **Board** (Kanban), **Table**, **Roadmap**
- Custom fields: priority, sprint, story points, assignee, status
- Automation: when a PR is merged, move card to “Done”

Capstone Project Setup (Required)

Each capstone team will run their project on:

- A **public GitHub repository**
- With a **Projects board** (Kanban or Scrum view)
- Stories tracked as **Issues** ("As a. . . , I want. . . , so that. . . ")
- At least **one PR per member per week**, reviewed by a teammate
- Weekly **milestones**; mid-project demo in week 12
- README, CONTRIBUTING.md, architecture diagram in `/docs`

Mixed Teams — Use Each Other's Strengths

Capstone teams will mix **BSc CS** and **BITAM** students.

- BCS strengths: deep technical, algorithms, backend
- BITAM strengths: requirements, stakeholder communication, UX, business case
- **Both must contribute code** — no "manager" role
- Pair-program across the boundary — BCS shows BITAM the API, BITAM shows BCS the user need
- Rotate the Scrum Master role between members

Summary

- Methodology = how the team works (vs architecture = what we build)
- Agile > Waterfall for software in 99% of cases
- Scrum: sprints, ceremonies, three roles
- Kanban: visualise + limit WIP, continuous flow
- Use Git branches + PRs + reviews; small, frequent merges
- Run your capstone on GitHub Projects with weekly cadence

Next: HTML & CSS Basics — selectors, the cascade, specificity, accessibility.

Questions?