SOFTWARE DEVELOPMENT AND ENGINEERING PRACTICES

Agile Development and Scrum for Engineering Teams

Level: Practitioner • 2 days (expandable to 3) • Virtual, In-person

Overview

Most teams that say they do Scrum are really doing a stand-up and a two-week deadline. The ceremonies are easy to copy; the hard part is the thinking underneath them: working in small increments, making progress visible, and using short feedback loops to change course before a bad plan gets expensive. When a team misses that, agile becomes overhead, and engineers rightly resent it.

This is a hands-on, practitioner course. It starts with why agile exists and the problems it actually solves, because every practice makes more sense once you know what it is protecting you from. From there it layers up gradually: the Scrum framework and its roles, shaping a backlog worth building from, running a sprint well, and then Kanban and flow as an alternative lens. Rather than survey every agile framework, the course goes deep on the small set of practices that make the biggest difference for an engineering team. Every module ends with a lab and builds on the one before.

Who Should Attend

- Developers and testers joining or already working on an agile team
- Technical leads and engineering managers who run sprints and want them to work better
- Product owners and scrum masters who want to understand the engineering side of agile

Prerequisites

- Experience working on a software team in any role
- No prior agile or Scrum experience required
- Familiarity with how software gets planned and delivered in your organization

What You Will Learn

- Explain the agile values and why iterative delivery beats big up-front plans
- Describe the Scrum roles, events, and artifacts and what each one is for
- Write and refine user stories and keep a backlog ordered and healthy
- Plan, run, and review a sprint, including estimation and handling mid-sprint change
- Apply Kanban flow practices: visualizing work, limiting work in progress, and measuring cycle time
- Judge which practices fit your team and lead a useful retrospective

Course Outline

Day one: the thinking behind agile, and the Scrum framework

- Why Agile Exists
  - What goes wrong with big up-front plans: late feedback and expensive surprises
  - The agile values and principles in plain engineering terms
  - Iterative and incremental delivery, and what "done" really means

  - Lab: take a failed waterfall-style project scenario and identify where earlier feedback would have changed the outcome
- The Scrum Framework
  - Roles: product owner, scrum master, and the developers
  - Events: sprint planning, daily scrum, review, and retrospective, and what each protects
  - Artifacts: product backlog, sprint backlog, and the increment
  - Lab: map your own team's current process onto Scrum and spot the gaps
- Backlogs and User Stories
  - Writing user stories that describe value, not tasks
  - Acceptance criteria and a working definition of done
  - Ordering and refining the backlog so planning stays cheap
  - Lab: write and refine a small product backlog for a realistic feature set

Day two: running the work, and improving the system

- Planning and Running a Sprint
  - Estimation that is honest about uncertainty: story points and their limits
  - Sprint planning, forecasting, and committing as a team
  - The daily scrum as coordination, not status theater, and handling mid-sprint change
  - Lab: plan a sprint from the backlog you built, then replay it against a scripted week of surprises
- Kanban and Flow
  - Visualizing work and finding where it actually gets stuck
  - Work-in-progress limits and why finishing beats starting
  - Cycle time, throughput, and when Kanban fits better than Scrum
  - Lab: build a Kanban board for a team's real workflow and set defensible WIP limits
- Inspecting and Adapting
  - Sprint reviews that gather real feedback from real stakeholders
  - Retrospectives that produce one concrete change, not a list of complaints
  - Metrics that help (and the ones that get gamed)
  - Lab: run a structured retrospective on the simulated sprint and commit to one improvement

Extended Version

The three-day version keeps the same gradient and adds depth where teams struggle most:

- Scaling topics: coordinating multiple teams, dependencies, and release planning
- Agile engineering practices in the sprint: continuous integration, testing, and small pull requests
- Working with stakeholders: forecasting, roadmaps, and saying no with data
- A capstone simulation: teams run two full mini-sprints on a shared product, including planning, a review, and a retrospective