

COURSE OUTLINE

Agentic ADLC with GitHub Copilot — From JIRA Ticket to Merged PR

An advanced program in knowledge, specialised agents, and spec-driven delivery

DURATION

3 days

FORMAT

Instructor-led, hands-on, lab-driven, project-based

DELIVERY

Online Instructor Led Workshop · Onsite Program for your Teams

TRAINER

Muhammed Shakir

Why this course exists

This program turns GitHub Copilot from an individual coding assistant into a repeatable engineering workflow: participants ground agents in repository knowledge, define specialised roles, and carry a ticket through specification, implementation, testing, review, and documentation.

The cost of weak context shows up in familiar places: repeated explanations, invented APIs, changes that drift from the requirement, and tests that confirm the implementation without proving the ticket. Faster code generation only helps when the surrounding workflow makes the result easier to verify.

By the end, participants will know how to make repository knowledge reusable, choose the right level of agent involvement, write specifications an agent can execute, and preserve traceability across JIRA, two repositories, pull requests, and Confluence.

The course culminates in one bounded ticket delivered across a producer and consumer repository, with explicit acceptance criteria, test evidence, independent review, a human merge decision, and documentation that improves the next ticket.

What makes this different

The focus is the complete engineering task, including the context and verification that make generated code useful. **This is a delivery-focused engineering room: every agent action is tied to a requirement, a controlled hand-off, and evidence of completion.** You will:

- **Make the repository the agent's working memory** — organise instructions, architecture notes, decisions, and runbooks so the agent can find the right context without rediscovering the project on every request.

- **Measure the cost of getting a ticket done** — baseline turns, wall-clock time, rework, and available usage measures; compare model and context choices against accepted output.
- **Give specialised agents clear boundaries** — separate design, coding, testing, and review through explicit inputs, outputs, tool permissions, and hand-off contracts.
- **Use specifications to control delivery** — turn a ticket into verifiable acceptance criteria, a challenged technical plan, and small implementation tasks before changing code.
- **Close the knowledge loop** — coordinate changes across repositories, produce evidence-backed pull requests, and update Confluence and repository knowledge so the next task starts with better context.

Every concept is applied to a prepared engineering workspace, so participants leave with a workflow they have exercised and artifacts they can adapt to their own teams.

Skills You'll Walk Away With

- Explain how the agentic development lifecycle (ADLC) changes context preparation, task delegation, verification, and human accountability within the software development lifecycle.
- Baseline a ticket using turns, available usage measures, wall-clock time, and rework; choose modes and models appropriate to the task.
- Structure repository knowledge with scoped instructions, architecture notes, indexes, ownership, and checks for stale guidance.
- Write grounded prompts and reusable skill files with clear goals, constraints, output contracts, done criteria, and stop conditions.
- Define specialised Design, Coding, Test, and Review agents with bounded responsibilities and inspectable hand-off artifacts.
- Translate a JIRA ticket into a specification, technical plan, and task breakdown with measurable acceptance criteria and explicit exclusions.
- Use MCP to retrieve ticket context and prepare controlled JIRA and Confluence updates with traceable human decisions.
- Plan compatible API or schema changes across producer and consumer repositories, including merge order and partial-rollout recovery.
- Evaluate generated changes against requirements, test evidence, review findings, and a representative ticket set.
- Produce a pull request and documentation package that preserves the ticket's intent, risks, validation, and decisions for the next engineer or agent.

Who should attend

This program is for experienced technical professionals who already work with GitHub Copilot and want to use agents across a controlled, collaborative delivery workflow.

-

Software and full-stack developers — taking agent-assisted work from a ticket through implementation, tests, and review.

- **Technical leads and engineering managers** — establishing team conventions, ownership, and measures of delivery improvement.
- **Solution and software architects** — preserving design intent, interface contracts, and repository boundaries during agent-assisted changes.
- **QA and test engineers** — deriving checks from requirements and challenging AI-written code with independent evidence.
- **Developer-experience and platform engineers** — maintaining reusable instructions, agent definitions, skills, and approved tool integrations.

Prerequisites: practical software development experience, Git and pull-request workflows, automated testing, and working familiarity with GitHub Copilot. Participants should be comfortable navigating an existing codebase and reading API contracts. Labs use a prepared two-repository workspace, a configured Copilot environment, and approved JIRA and Confluence MCP connections with training data. Advanced mathematics and model training are not required.

Course Outline

Day 1 — The ADLC Operating Model, Knowledge & Context

Establish a delivery baseline and build the repository knowledge and instruction patterns that help agents produce accurate, reviewable work with less repeated discovery.

Module 1 • The ADLC Operating Model & Token Economics

- Where the agentic development lifecycle differs from the SDLC — delegation, context preparation, verification, and accountability
- The re-discovery tax: what the agent re-learns on each request, and the effect on usage, latency, and accuracy
- Baseline a ticket end to end — turns, wall-clock time, rework, and premium requests or other usage measures available in the configured environment
- Mode and model selection per task: mechanical changes, reasoning, and review — when a lower-cost model is sufficient
- The context window as a budget: what belongs in active context, in a file, or behind a tool call; token use versus product billing measures
- Interactive versus delegated asynchronous work — choosing an approach based on ambiguity, task boundaries, and review needs

Module 2 • Knowledge Harnessing — the Repository as the Agent's Memory

- The layered knowledge model — organisation, repository, path-scoped, and task-scoped guidance; checking what the selected Copilot surface actually loads
- Repository instructions, architecture notes, ADRs, conventions, and runbooks as reusable context
- Signal versus bloat — what to record, what the agent can infer, and what actively misleads it
- Structure for retrieval: scoping rules, indexes, module maps, and navigable naming
- Keep knowledge current — ownership, review cadence, staleness detection, same-PR updates, and targeted CI checks for drift
- Protect sensitive information — secrets, credentials, and client data; leak paths through prompts, files, tool results, and generated documentation

Module 3 • Prompt & Context Engineering for Accurate Output

- Anatomy of a useful instruction: goal, constraints, inputs, output contract, done criteria, and stop conditions
- Reusable skill files — turn ad-hoc prompts into versioned, reviewed assets with explicit task inputs
- Grounding patterns: cite the file, inspect before changing, acknowledge missing evidence, and ask rather than invent
- Failure taxonomy — invented APIs, silent scope drift, confidently wrong refactors, and tests written only to pass; the checks that expose each
- Context hygiene in a long session — preserve decisions and evidence, compact selectively, and recognise when to start clean
- Prove an instruction improves results — use representative tickets, an acceptance rubric, and a baseline pass rate

Day 1 Labs — baseline a ticket · compare mode and model choices · build scoped repository knowledge · detect stale or unsafe context · turn a prompt into a reusable skill · evaluate grounding on representative tickets

Day 2 — Specialised Agents, Specifications & the Ticket as the Unit of Work

Give agents clear roles and hand-offs, then turn a JIRA ticket into a verifiable specification and plan before any code changes.

Module 4 • Specialised Agents — Design, Coding, Test & Review

- When narrow roles help — context pollution, conflicting goals, reviewer independence, and the coordination cost of extra agents
- Define an agent: role, goal, allowed tools, inputs, outputs, guardrails, hand-off contract, and prohibited actions
- Four roles in practice: Design explores options; Coding follows the architecture; Test proves requirements; Review challenges the change independently

- Hand-off artifacts as interfaces — specification, plan, diff, test report, and review report; give reviewers the requirements and evidence without anchoring them on the coder's rationale
- Orchestration patterns — sequential hand-offs, parallel drafts, review-before-merge gates, and explicit human decisions
- Anti-patterns — agent sprawl, overlapping responsibilities, stale definitions, and treating multiple agreeing agents as proof of correctness

Module 5 · Spec-Driven Development

- Spec-first delivery — a durable statement of intent, constraints, and acceptance that survives the chat session
- The chain: principles ' specification ' technical plan ' task breakdown ' implementation, with a human checkpoint between stages
- Specifications an agent can execute and a test can check — verifiable acceptance criteria, explicit out-of-scope work, and stated assumptions
- Challenge the plan before code exists — missing dependencies, unsupported assumptions, unnecessary changes, and testability
- Keep specification, tests, and code aligned; detect spec drift during pull-request review
- When a full specification is excessive — use a short acceptance checklist and bounded plan for small, clear changes

Module 6 · The Ticket as the Unit of Work — JIRA over MCP [MCP]

- MCP in practice — what the agent can see, tool-call budgets, and retrieving current ticket context instead of relying on a pasted snapshot
- Make a ticket agent-ready — acceptance criteria, definition of ready, linked context, and a scope the agent can finish
- Ticket ' specification ' plan ' tasks, with the ticket key connecting branch, commit, pull request, and documentation
- Read the surroundings — epics, linked issues, prior implementations, and earlier design decisions; retrieve only what the task needs
- Under-specified tickets — surface questions and missing evidence before implementation
- Write-back boundaries — permitted updates, human approval points, least-privilege access, untrusted ticket content, and an auditable record

Day 2 Labs — define and exercise the four specialist roles · review hand-off artifacts without anchoring · write and challenge a specification · retrieve and clarify a JIRA ticket over MCP · create a traceable implementation plan and task breakdown

Day 3 — Multi-Repo Delivery, Review & the Capstone: JIRA 'Code' PR 'Confluence

Carry one bounded ticket through coordinated repository changes, independent review, linked pull requests, and a reusable knowledge update.

Module 7 · Multi-Repo Engineering with Agents

- Why cross-repository work is harder — fragmented context, limited search visibility, and contract changes arriving out of order
- Workspace strategies — multi-root workspaces, per-repository instructions, and a shared knowledge repository
- Contract-first change — API or schema first, producer and consumer sequencing, a compatibility window, and incremental merges
- Coordinated pull requests — linked changes, merge order, validation, and recovery when only one side lands
- Keep cross-repository context focused — indexes, interface summaries, and generated clients instead of loading entire repositories into context
- Shared libraries and versioning — ownership boundaries, compatibility checks, and changes that require an explicit maintainer decision

Module 8 · Closing the Loop — Review, Pull Requests & Documentation [MCP]

- Run the Review agent on a real change set — useful findings, blind spots, and the human's responsibility for acceptance
- Review AI-written code against the ticket — missing edge cases, invented dependencies, weakened tests, and claims unsupported by the diff
- Build a pull request from evidence — specification links, test results, unresolved risks, merge order, and rollback notes
- Confluence write-back over MCP — publish approved decisions, interface contracts, and runbooks with source links and controlled permissions
- Feed documentation back into repository knowledge — update the guidance the next ticket will retrieve
- Roll out the workflow in a team — ownership of agent and skill files, review standards, and comparison with the initial delivery baseline

Day 3 Labs — coordinate a producer–consumer change · test against acceptance criteria · run independent review and resolve findings · prepare linked PRs and make the training merge decision · update Confluence and repository knowledge, then compare with the baseline

Capstone — One Ticket, Two Repositories, a Complete Delivery Trail

Participants deliver a bounded feature change across prepared producer and consumer repositories using training tickets and data:

- Retrieve the JIRA ticket and relevant linked context through an approved MCP connection
- Produce a specification with verifiable acceptance criteria, assumptions, and exclusions
- Challenge the technical plan and approve the task breakdown before implementation
- Use specialised Design, Coding, Test, and Review roles with explicit hand-off artifacts
- Implement compatible changes across both repositories with a documented merge order
- Derive tests from the acceptance criteria and retain the actual execution evidence
- Resolve review findings and prepare linked pull requests with risk and rollback notes
- Make a human acceptance decision and merge approved changes in the training repositories
- Update Confluence and repository knowledge with the decisions and contracts needed by the next ticket
- Compare turns, available usage measures, wall-clock time, and rework with the initial baseline

Acceptance depends on satisfying the ticket, passing the relevant checks, resolving material review findings, and preserving a complete delivery trail. A confident agent response or polished PR description is insufficient evidence.

What participants take away

- A task, mode, and model selection rubric
- A ticket baseline and delivery comparison scorecard
- Scoped repository instructions and a maintainable knowledge index
- Reusable prompt and skill patterns with grounding and stop conditions
- Design, Coding, Test, and Review agent definitions with hand-off contracts
- Specification, technical-plan, and task-breakdown templates
- JIRA and Confluence MCP workflows with write-back boundaries
- A cross-repository contract, merge-order, and rollback checklist
- An evidence-backed pull-request and documentation package
- A completed training ticket demonstrating the workflow from intake to merged code and updated knowledge

A note on delivery

This is a build-and-verify program. Each day combines focused instruction, guided demonstrations, individual or pair labs, review, and troubleshooting. The labs are short, connected exercises that build toward the capstone; the capstone integrates their artifacts on Day 3, with time for review and a baseline comparison.

To keep the scope teachable, the repositories, development environment, access permissions, and MCP connections are prepared before delivery. Day 1 establishes the baseline, knowledge, and context; Day 2 adds specialised roles, specifications, and ticket intake; Day 3 uses all of it for implementation, review,

and delivery. Model and integration exercises use the capabilities available in the cohort's approved environment.

The course ends at merged training pull requests and updated documentation. SRE and operations agents are outside scope; existing team CI checks provide validation, while CI/CD pipeline design, deployment, and production operations remain with the teams that own them.

Participants leave with a completed delivery workflow and reusable engineering artifacts they can inspect, measure, and adapt to their own repositories.