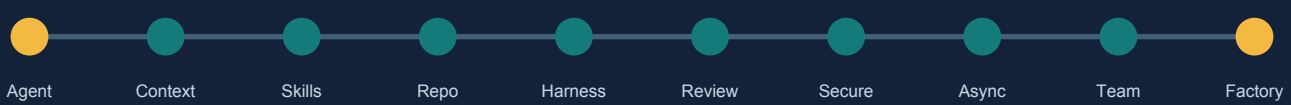
2026 EDITION

Modern Software Engineer

Study Guide 2026

A curated, project-based ten-week guide to building with coding agents

Videos, courses, articles, books, weekly builds, and a capstone



Updated September 4, 2026

Contents

A ten-week, project-based guide to building software with coding agents: how agents work inside, context engineering and MCP, reusable skills, repository readiness, AI code review, security, background agents, team adoption, and the software factory. Each week pairs a capped set of core material with a build and a checkable “done when” list, so you finish with a working system rather than a reading log.

Credit. The ten-week structure and the weekly topics follow the Fall 2026 syllabus of Stanford's CS146S, The Modern Software Developer (themodernsoftware.dev). This guide is independent and not affiliated with the course; every reading, video, build, and checklist is its own selection. All links were checked on September 4, 2026, and every video carries its running time so you can budget honestly.

How to use this guide

Plan on 10–12 hours per week. Split it like this:

- 3–4 hours: the Core material for the week. Each week's core list is capped near four hours; the running times are printed next to each item.
- 5–6 hours: the weekly Build.
- 1–2 hours: document results, failures, costs, and lessons, then tick the week's Done when list.

Everything under Deeper material, Additional video track, and Tools and references is optional. Pick by interest; do not try to clear it.

Use one medium-sized repository throughout the ten weeks. A small SaaS app, developer tool, or API with a UI, tests, and CI works better than ten disconnected toy projects. Each week should improve the same system.

The strongest free backbone is Hugging Face's Context Course (units: Agent Skills, MCP, Plugins, Sub-agents, Hooks, and a bonus Nano Harness). It overlaps unusually well with the first half of this guide. Use the materials below to deepen and extend it.

The guide is language-agnostic; most examples use Python or JavaScript. You need solid programming experience (two or three university-level courses or the equivalent). A machine-learning background helps but is not required; if transformers are new to you, the Week 1 optional foundation covers the gap.

Tooling and budget

Some tools need a subscription or an API key. Decide this before Week 1:

- Pick one primary coding agent and stay with it for at least five weeks so the customization work in Weeks 3–4 compounds. Reasonable choices: Claude Code (subscription or API key), OpenAI Codex CLI (open source, subscription or API key), Cursor, Gemini CLI (open source, has a free tier), or opencode (open source, bring your own key).
- Set a hard monthly spend cap before you start and log spend per week alongside your build notes. Cost is a first-class metric in this guide, not an afterthought.
- Learn prompt caching and use a cheaper model for subagents and review passes. Most runaway bills come from long contexts re-sent on every turn.
- Free path: the Hugging Face courses, the open-source agents above with their free tiers, and every article in this guide cost nothing. Only the API usage for your builds costs money.

The ten weeks at a glance

Week	Topic	You build
1	The internals of coding agents	A 300–500 line terminal agent with four tools and full logging
2	Advanced context engineering	A one-page spec, one feature shipped through RePPIT, and an MCP server with 2–4 tools
3	Agent skills and the CLI	A packaged skill with a helper script, plus a browser-driven web skill
4	Customizing your agent and repository	Repository instructions, two hooks, and a planner / implementer / reviewer split
5	Agent-ready codebases	A scored readiness audit and the fixes that make a fresh agent productive
6	Agentic code review	A severity-ordered review rubric wired to pull requests, measured on five PRs
7	Security	A threat model, SAST / SCA / secret scans in CI, and a prompt-injection test
8	Background agents	An issue-to-PR flow with isolation, budgets, checkpoints, and retries
9	Building an AI-native team	A model gateway, an MCP portal, and a one-page adoption policy
10	The software factory and the future	A traced end-to-end factory with an eval set and a controlled improvement loop

Week 1 — The Internals of Coding Agents

Focus: what an LLM actually is and what the agent loop looks like under the hood; the core tool set (read, write, edit, bash) and how tasks flow through it; how production coding agents structure their system prompts and tool definitions.

Core material (≈ 3 h 20 min)

1. Video: Andrej Karpathy, Intro to Large Language Models (59 min) — the best compact conceptual foundation. If you have the time, watch his longer Deep Dive into LLMs like ChatGPT (3 h 31 min) instead for a fuller foundation.
2. Article + code: Thorsten Ball, How to Build an Agent (≈ 60 min with the code) — a small, legible coding agent with the essential tool loop. Use it as the model for this week's build.
3. Article: Anthropic, Building Effective AI Agents (25 min) — the canonical text on workflows versus agents and the augmented-LLM loop. Barry Zhang's talk below is the video form.
4. Engineering article: OpenAI, Unrolling the Codex agent loop (20 min) — a production-oriented explanation of the loop and its design tradeoffs.
5. Talk: Barry Zhang of Anthropic, How We Build Effective Agents (15 min) — the minimal loop, tool use, risk, verification, and debugging from inside an agent's limited context.

Production prompts and tool definitions

Read real system prompts, not summaries of them. These are public and readable:

- OpenAI Codex CLI is open source. Read the model prompts in codex-rs/core (files named `gpt_5_2_prompt.md`, `gpt_5_codex_prompt.md`, and similar) and the compaction prompt under `codex-rs/prompts/templates/compact/`.
- Google's Gemini CLI keeps its system prompt in `packages/core/src/core/prompts.ts` and its MCP prompts in `packages/core/src/prompts/`.
- opencode is a third open-source coding agent whose prompts and tool schemas you can diff against the two above.
- Video: How Claude Code Works (1 h 06 min) — an independent workshop on prompt-driven architecture, tool calls, subagents, permissions, and evaluations. The best public walkthrough of a production agent's prompt and tool design.

Optional foundation

- 3Blue1Brown, Transformers, the tech behind LLMs (27 min) — the clearest visual explanation of attention.
- Jay Alammar, The Illustrated Transformer — the written companion.
- Andrej Karpathy, How I use LLMs (2 h 11 min) — practical model use, tools, and limitations.
- Chip Huyen, AI Engineering — read the sections on foundation models, evaluation, and application architecture.

Deeper material

- Geoffrey Huntley, How to build a coding agent: free workshop — a second from-scratch build with a different design.
- mini-swe-agent — a 100-line agent that scores over 74% on SWE-bench Verified. Read it after your own build to see what you over-engineered.
- OpenAI, A practical guide to building agents (PDF, 30 min).
- LangChain, Building Effective Agents with LangGraph (31 min) — routing, parallelization, orchestrator-worker, and evaluator-optimizer patterns.

Additional video track

- Andrew Ng, What's next for AI agentic workflows (13 min) — reflection, tool use, planning, and multi-agent collaboration as the four agentic patterns.
- Robert Brennan of OpenHands, Software Development Agents: What Works and What Doesn't (17 min) — the editor, terminal, browser, sandbox, and action loop.

Build

Build a terminal coding agent in roughly 300–500 lines. Give it four tools: list files, read a file, edit a file, and run a shell command. Log every model response, tool call, result, token count, and stop condition. Test it on three tiny repository tasks and manually classify every failure as model, context, tool, or control-loop failure.

Then read one production system prompt (Codex or Gemini CLI) end to end and annotate it: which lines set persona, which set safety boundaries, which shape tool selection, which handle stopping. Compare it with your own prompt.

Done when

- Your agent completes at least one of the three tasks end to end with all four tools exercised.
- Every run has a log with per-turn token counts and the final stop reason.
- A failure table exists with at least five rows, each classified as model, context, tool, or control loop.
- One annotated production prompt is checked into your notes with at least ten annotations.

Week 2 — Advanced Context Engineering

Focus: advanced prompting techniques and when each applies; RePPIT (Research, Propose, Plan, Implement, Test) and spec-driven development; MCP fundamentals (servers, clients, tools, transport); designing tools for agent ergonomics.

Core material (≈ 3 h 50 min)

1. Video/podcast: The Pragmatic Engineer with Dex Horthy, Context Engineering (1 h 33 min) — context, harnesses, loops, research/plan/implement workflows, compaction, and software factories. The article and transcript are useful for notes.
2. Article: Anthropic, Effective context engineering for AI agents (25 min) — the canonical framing of context as a finite resource: system prompts, tools, examples, retrieval, and compaction.
3. Video: Anthropic, Prompting for Agents (29 min) — how prompting changes when the model runs in a loop with tools.
4. Article: Anthropic, Writing effective tools for agents (25 min) — tool names, descriptions, boundaries, results, and evaluation.
5. Specification: Model Context Protocol, Specification (30 min for the architecture, transports, and tools sections) — read the primary source before any tutorial.
6. Documentation: GitHub, Spec Kit and its Agentic SDD workflow (20 min) — a concrete implementation of spec-driven development.

Deeper material

- Course: Hugging Face, MCP Course (units 0–2, roughly 6–8 h; spread it across Weeks 2 and 3) — free, hands-on foundations, continuing into the end-to-end MCP application unit.
- Manus, Context Engineering for AI Agents: Lessons from Building Manus — KV-cache hit rate, file system as context, and keeping failures in the trace.
- Drew Breunig, How Long Contexts Fail and Chroma, Context Rot — the evidence that more context is not free.
- HumanLayer, Advanced Context Engineering and the 12-Factor Agents principles.
- Ravi Mehta, Specs Are the New Source Code and the Kiro specs docs — the product side of spec-driven development.
- Tooling: MCP Inspector for testing servers; the MCP Registry for publishing and discovery.
- Matt Pocock, How I use Claude Code for real engineering (10 min) — planning, clarifying questions, phased execution, and context-window management.

Additional video track

- Advanced prompting: Anthropic, AI Prompt Engineering: A Deep Dive (1 h 16 min), and the shorter Prompting 101 (24 min).
- What still works: Sander Schulhoff on Lenny's Podcast, AI prompt engineering in 2025: What works and what doesn't (1 h 37 min) — evidence-based prompting, few-shot, decomposition, and prompt injection.
- Research, plan, implement: Dex Horthy, No Vibes Allowed: Solving Hard Problems in Complex Codebases (21 min) — targeted research, deliberate compaction, concrete plans, and human review.
- Long-horizon context: Harrison Chase of LangChain, Context Engineering Our Way to Long-Horizon Agents (39 min).

- Spec-driven development: Al Harris of Amazon Kiro, Spec-Driven Development: Agentic Coding at FAANG Scale and Quality (1 h 04 min) — requirements, acceptance criteria, design artifacts, tasks, and steering documents.
- MCP workshop: Mahesh Murag of Anthropic, Building Agents with Model Context Protocol (1 h 44 min) — the full workshop from the protocol's authors.
- MCP video course: DeepLearning.AI and Anthropic, MCP: Build Rich-Context AI Apps (≈ 1 h 30 min) — architecture, servers, clients, tools, resources, prompts, remote deployment, and Inspector-based testing.
- Quick architecture tour: Gaurav Sen, Model Context Protocol: A Deep Dive (9 min).

Build

Choose one real feature for your repository. Write a one-page spec containing objective, constraints, out-of-scope items, acceptance checks, edge cases, and a test plan. Follow Research -> Propose -> Plan -> Implement -> Test without skipping a stage, and keep the artifact from each stage.

Then build one MCP server with two to four narrow tools. Exercise it in MCP Inspector first, then from your agent. Test happy paths, invalid parameters, huge results, timeouts, and permission failures. Revise each description until a fresh agent consistently selects the right tool and passes valid arguments.

Done when

- The spec exists, and each RePPIT stage produced a saved artifact (research notes, proposal, plan, diff, test results).
- The MCP server passes an Inspector session covering every tool with valid and invalid input.
- A fresh agent selects the correct tool on five out of five scripted requests.
- Cost and token counts for the feature are recorded in your notes.

Week 3 — Agent Skills and CLI

Focus: what skills are and how `SKILL.md` plus scripts encode a workflow; web skills and extending agent capability beyond the repository; working effectively from the CLI.

Core material (≈ 3 h 40 min)

1. Course unit: Hugging Face, Agent Skills (≈ 60 min) — the best structured introduction to portable, progressively disclosed skills.
2. Article: Anthropic, Equipping agents for the real world with Agent Skills (20 min), with the public skills repository as worked examples.
3. Specification + tutorial: Agent Skills Quickstart (30 min).
4. Video: Lee Robinson of Cursor, The Beginner's Guide to Coding with Cursor (45 min) — typed languages, linting, formatting, tests, branch review, and parallel background work.
5. CLI foundation: MIT, The Missing Semester 2026: Course Overview and Introduction to the Shell (≈ 60 min with exercises) — shell navigation, composition, scripting, streams, permissions, and safe Bash practices.
6. Standard: AGENTS.md (10 min) — read the examples and conventions now; repository instructions become the main topic in Week 4.

Tools and references

- Command Line Interface Guidelines — how to design CLIs that agents (and humans) can drive reliably. Apply it to your skill's helper script.
- Web skills tooling: Playwright MCP, Chrome DevTools MCP, and browser-use.
- Claude Code, Common workflows — git worktrees, parallel sessions, and resuming conversations from the CLI.

Additional video track

- Hands-on skills workshop: Pedro Rodrigues of Supabase, Skill Issue: How We Used AI to Make Agents Actually Good at Supabase (1 h 19 min) — builds and evaluates `SKILL.md` , supporting scripts, progressive disclosure, and routing behavior.
- Web skills: Paul Klein of Browserbase, Bringing Agents onto the World Wide Web (18 min) — browser-agent harnesses, network interception, WebMCP, Playwright CLI, memory, and reusable website skills.
- Concept explainer: Greg Isenberg, How AI agents & Claude skills work (Clearly Explained) (35 min).
- Long form: Lex Fridman with the Cursor team, Future of Programming with AI (2 h 29 min) — editor design, speculative edits, and how the Cursor founders think about agents.

Build

Find a workflow you have repeated at least three times: release notes, database migration checks, API endpoint scaffolding, dependency upgrades, or UI regression checks. Package it as a skill with:

- A short `SKILL.md` that tells the agent when to use it.
- Progressive disclosure: keep rare details in linked references.
- One deterministic helper script that follows the CLI guidelines above.
- A fixture or sample input.
- A pass/fail verification command.

Run it from a fresh agent session on two different inputs. Record where the instructions were ambiguous or overloaded the context. Then add one web skill: a browser-driven check (Playwright MCP or Chrome DevTools MCP) that verifies something in your running app.

Done when

- The skill triggers correctly from a fresh session on two inputs without extra coaching.
- The helper script has a documented exit code and runs in under ten seconds.
- The web skill drives a real browser and reports pass/fail on your app.
- Your notes list every ambiguity you fixed in `SKILL.md` and why.

Week 4 — Customizing Your Agent and Repository

Focus: `CLAUDE.md` and `AGENTS.md`, what to put where; hooks for lint gates, test runs, and guardrails; subagent patterns (planner / implementer / reviewer).

Core material (≈ 3 h 40 min)

1. Article: Anthropic, Claude Code best practices (30 min) — the canonical guide to `CLAUDE.md`, tool allowlists, workflows, and multi-Claude patterns.
2. Article: Anthropic, Steering Claude Code: skills, hooks, rules, subagents, and more (15 min) — when to use each mechanism.
3. Course units: Hugging Face, Sub-agents and Hooks (≈ 90 min).
4. Video: Anthropic, Mastering Claude Code in 30 minutes (28 min) — Boris Cherny's own walkthrough of the configuration surface.
5. Video: Boris Cherny, Inside Claude Code With Its Creator (50 min) — terminal design, `CLAUDE.md`, teams, subagents, plan mode, and the future of coding.

Tools and references

- Claude Code docs: Hooks reference and Create custom subagents — the exact event names, matchers, and frontmatter you need for the build.
- Repository instruction files: GitHub, How to write a great agents.md: lessons from over 2,500 repositories; Real Python, How to Write an AGENTS.md File; Agentic AI Foundation, Writing an Effective AGENTS.md.
- The subagent debate: Cognition, Don't Build Multi-Agents versus Anthropic, How we built our multi-agent research system. Read both before designing your three roles.
- Anthropic, How Anthropic teams use Claude Code — internal adoption patterns by team.
- Free courses: Anthropic Academy, Claude Code in Action; DeepLearning.AI, Claude Code: A Highly Agentic Coding Assistant.

Additional video track

- Full workflow: Matt Pocock, AI Coding Workflow: From Product Idea to Tested Implementation (1 h 37 min) — human alignment, PRDs, vertical slices, unattended agents, QA, review, and parallelization.
- Creator, short form: Boris Cherny, Claude Code & the evolution of agentic coding (18 min).
- Engineers' perspective: Every, The Secrets of Claude Code From the Engineers Who Built It (1 h 10 min), and the Latent Space interview Claude Code: Anthropic's Agent in Your Terminal with Boris Cherny and Cat Wu.
- Hooks in practice: IndyDevDan, I'm HOOKED on Claude Code Hooks: Advanced Agentic Coding (30 min) — pre/post tool hooks, logging, and stopping a destructive command.
- Advanced configuration: Anthropic, Claude Code Advanced Patterns: Subagents, MCP, and Scaling to Real Codebases — hooks, orchestration, guardrails, internal tools, and context strategies.
- Practitioner perspective: DHH with Lex Fridman, Future of Programming, AI, Modern Software Engineering, Vibe Coding and Linux (5 h 15 min; watch the programming-with-agents, vibe-coding-versus-engineering, agent setup, model, and harness chapters only). The timestamped transcript makes selective viewing easy.

Build

Add a concise repository instruction file. Treat it as a map, not an encyclopedia: architecture entry points, commands, conventions, safety boundaries, and links to deeper docs. Add one deterministic hook that runs a fast lint or test gate, and one that blocks a dangerous command.

Create three roles (planner, implementer, and independent reviewer) and give each a narrow contract. Run the same feature once with a single agent and once with the three-role workflow. Compare elapsed time, tokens, defects found, and amount of human intervention.

Done when

- The instruction file is under 150 lines and a fresh agent can run setup, tests, and lint from it alone.
- The lint/test hook blocks a deliberately broken commit; the safety hook blocks a deliberately dangerous command.
- The single-agent versus three-role comparison table exists with time, tokens, defects, and interventions.
- You wrote one paragraph on whether the subagent split paid for itself, citing the numbers.

Week 5 — Agent-Ready Codebases

Focus: what makes a repository agent-ready: structure, docs, tests, and checks; scoring and auditing readiness; common gaps that block agents in real repositories.

Core material (≈ 3 h 30 min)

1. Engineering case study: OpenAI, Harness engineering: leveraging Codex in an agent-first world (30 min) — arguably the single most important reading for this week: repository legibility, structured documentation, machine-enforced invariants, and progressive disclosure.
2. Practical guide: Adobe, Repository Harnesses for AI Coding Agents (≈ 60 min) — a free, systematic guide to making a repository navigable and verifiable by agents.
3. Essay: Martin Fowler, Harness Engineering (20 min).
4. Videos: Eno Reyes of Factory, Making Codebases Agent-Ready (16 min) and Building Reliable Agentic Systems (18 min) — mechanical verification, linters, end-to-end tests, interface documentation, planning, grounding, and human oversight.
5. Evals primer: Hamel Husain, Your AI Product Needs Evals (20 min) — start the evaluation habit here; Week 10 depends on it.
6. Measuring readiness: Beyang Liu of Sourcegraph, The ROI of AI: Why You Need Eval Frameworks (25 min) — rigorous evaluations, repository context, engineering KPIs, and avoiding productivity theater.

Deeper material

- OpenAI, Unlocking the Codex harness.
- Revisit Dex Horthy's Context Engineering interview, particularly the harness, loop, and software-factory chapters.
- Diátaxis — a documentation structure (tutorials, how-tos, reference, explanation) that agents navigate well.
- Chroma, Context Rot — why a lean, well-indexed repository beats dumping everything into context.

Additional video track

- Tests as the harness: Kent Beck with The Pragmatic Engineer, TDD, AI agents and coding (1 h 15 min) — why test-driven development becomes a superpower when agents write the code.
- Harness design: Harrison Chase of LangChain, When to Build Your Own Agent Harness (23 min).

Build

Audit your repository on five dimensions: discoverability, setup, feedback speed, architecture enforcement, and recovery from failure. Score each from 1 to 5 with written evidence. Improve it until a fresh clone can be understood, installed, tested, and changed without tribal knowledge.

At minimum, add a one-command setup, a fast deterministic test path, a short architecture map, local conventions near the code they govern, and a machine-enforced architectural boundary. Give the repository to a fresh agent with one issue and no extra coaching; document every place it gets lost. Re-score afterwards.

Done when

- Before and after readiness scores exist for all five dimensions with evidence.
- One command sets up a fresh clone; one command runs the deterministic test path in under two minutes.
- One architectural boundary is enforced by a check that fails CI when violated.
- The fresh-agent trial is logged with every point of confusion and the fix you applied.

Week 6 — Agentic Code Review

Focus: what AI review catches well and what it misses; review architectures and custom rules; fitting AI review into a team's pull-request workflow.

Core material (≈ 3 h 30 min)

1. Canonical guide: Google, Engineering Practices: Code Review and the Reviewer Guide (60 min).
2. Free book chapter: Software Engineering at Google, Chapter 9: Code Review (45 min).
3. Talk: Tomas Reimers of Graphite, AI-powered entomology: lessons from millions of AI code reviews (10 min), with Graphite's written AI code review implementation and best practices (20 min).
4. Review architecture: Ankit Jain, How to Kill the Code Review (16 min) — a five-layer trust model combining specifications, reusable guardrails, deterministic checks, executable test plans, previews, and human alignment.
5. What automation cannot replace: Geoffrey Litt, Understanding Is the New Bottleneck (20 min) — review as architectural understanding, mentorship, and coordination rather than only correctness checking.
6. Integrations: GitHub, About GitHub Copilot code review (15 min) and Anthropic, Claude Code GitHub Actions (15 min) — the two most common ways to put a reviewer agent on a pull request.

From Cognition

- Cognition, Don't Build Multi-Agents — context sharing and why reviewer agents need the full trace.
- Latent Space with Cognition, DeepWiki: The GitHub Encyclopedia (32 min) — codebase understanding as a product; the same understanding a reviewer needs.
- Cognition, SWE-bench technical report and Devin: Coding Agents 101.

Deeper material

- Research paper: AI-Assisted Assessment of Coding Practices in Modern Code Review and Google Research, Resolving code review comments with ML.
- Rules and feedback loops: Your Coding Agent Doesn't Always Follow Your Rules (10 min) — hooks, deterministic checks, asynchronous verification, reviewer agents, and LLM-as-judge tradeoffs.

Build

Create a review rubric ordered by severity: correctness, security, data loss, concurrency, compatibility, tests, maintainability, then style. Require the reviewer agent to cite exact lines, explain the failure scenario, and propose the smallest fix. Use a model/session that did not write the change. Wire it to your pull requests with one of the integrations above.

Evaluate the reviewer on at least five pull requests, including one deliberately seeded with subtle bugs. Label every comment true positive, false positive, duplicate, or low-value. Track acceptance rate and escaped defects. A human still owns merge approval.

Done when

- The rubric is checked in and the reviewer agent runs automatically on pull requests.
- Five reviewed pull requests have every comment labeled, with precision computed.
- The seeded-bug pull request report states which planted bugs were caught and which escaped.
- One paragraph records what the reviewer systematically misses and what deterministic check now covers it.

Week 7 — Security

Focus: SAST/SCA, dependency and secret-leak vulnerabilities; prompt injection and agent-specific attack surfaces; agent-assisted triage and remediation.

Core material (≈ 3 h 50 min)

1. Threat catalogs: OWASP MCP Top 10 (30 min) and OWASP Top 10 for LLM Applications (30 min).
2. Agent-specific framing: Simon Willison, The lethal trifecta for AI agents (10 min) and Prompt injection explained (talk with transcript, 15 min).
3. Real exploit: Embrace The Red, GitHub Copilot: Remote Code Execution via Prompt Injection (CVE-2025-53773) (15 min) — a coding-agent attack chain end to end.
4. Video: Isaac Evans of Semgrep, When AI Writes Code: Rethinking App Security (45 min) — AI-generated vulnerabilities, SAST, security assistants, feedback loops, and risks created by coding agents.
5. Hands on: Semgrep, Finding vulnerabilities in modern web apps using Claude Code and OpenAI Codex (20 min) — agent-assisted triage in practice.
6. Coding-agent threat model: Fouad Matin of OpenAI, Safety and Security for Code-Executing Agents (14 min) — remote code execution, prompt injection, exfiltration, containers, network restrictions, approvals, and OS-level sandboxing.
7. Hands-on labs: PortSwigger, Web Security Academy — do the Web LLM attacks labs, including indirect prompt injection (≈ 60 min for two labs).

Tools and references

- SAST: Semgrep and CodeQL. Secrets: gitleaks. Dependencies/SCA: OSV-Scanner. Run all three classes in the build.
- Anthropic, Making Claude Code more secure and autonomous with sandboxing and the claude-code-security-review GitHub Action.
- OWASP, MCP Tool Poisoning and the original Invariant Labs disclosure, MCP Security Notification: Tool Poisoning Attacks.
- OWASP, Agentic AI Threats and Mitigations and the broader OWASP GenAI Security Project.
- Research: Design Patterns for Securing LLM Agents against Prompt Injections; NIST, Strengthening AI Agent Hijacking Evaluations.

Additional video track

- Practical sandboxing: Cloudflare's Harshil Agrawal, Why and How You Need to Sandbox AI-Generated Code (38 min) — capabilities, secrets, networking, cleanup, isolation surfaces, and indirect prompt injection.
- Lab companion: Rana Khalil, Web Security Academy series introduction (12 min).

Book

- Adam Shostack, *Threat Modeling: Designing for Security* — use the first edition; the second edition, retitled for an AI world, is announced for February 2027.

Build

Threat-model the entire agent workflow: user input, repository content, retrieved web pages, tools, credentials, MCP servers, generated commands, logs, and deployment. Apply least privilege, explicit allowlists, secret isolation, sandboxing, and human approval for irreversible actions.

Run SAST (Semgrep or CodeQL), dependency/SCA (OSV-Scanner), and secret scans (gitleaks) in CI. Plant a harmless indirect-prompt-injection string in an untrusted fixture and verify that the agent treats it as data rather than instructions. Write a short incident playbook for credential exposure, malicious tool output, and runaway cost.

Done when

- A threat model document lists every trust boundary in the agent workflow with a mitigation per boundary.
- SAST, SCA, and secret scanning run in CI and block on high-severity findings.
- The planted injection string is demonstrably ignored, with the transcript saved.
- The incident playbook covers the three scenarios with an owner and first three steps for each.

Week 8 — Background Agents

Focus: asynchronous, cloud-delegated agents; managing fleets of parallel agents; issue-to-PR pipelines and triggers from Slack, Linear, and GitHub.

Core material (≈ 3 h 30 min)

1. The products: OpenAI, Codex cloud; Anthropic, Claude Code on the web; Cursor, Cloud Agents; GitHub, Managing issues and pull requests with the Copilot coding agent (≈ 20 min each; compare their trigger, isolation, and approval models).
2. Orchestration case study: OpenAI, Open-sourcing Symphony and the Symphony specification (≈ 60 min) — issue-to-PR orchestration as a specification.
3. Video: Rajesh Bhatia of Cloudflare, Cloudflare's AI Engineering Stack: Assisted to Delegated (37 min) — how Cloudflare moved from assisted coding to delegated agents on its own platform.
4. Durable execution: Preeti Somal of Temporal, Scaling AI Agents Without Breaking Reliability (15 min) — persistent state, orchestration, visibility, automatic recovery, human interaction, and parallel work.
5. Fleet operations: Kyle Jaejun Lee, I Run a Fleet of AI Agents Across Three Machines: Here's What Broke (9 min) — attention limits, durable context, blocking approvals, routing, conflicting work, and multi-machine recovery.

Tools and references

- Cloudflare, Agents SDK and Sandbox SDK — a platform for stateful agents and isolated execution.
- Temporal, Durable AI Agent tutorial and the OpenAI Agents integration concepts.
- Cloudflare, Code Mode: give agents an entire API in 1,000 tokens — a tool-efficiency technique for fleets.
- Google, Jules — a third cloud agent to compare.
- Claude Code, Common workflows — git worktrees for parallel local agents.

Additional video track

- Build your own harness: Thariq Shihipar of Anthropic, Claude Agent SDK: Full Workshop (1 h 52 min) — the SDK that Claude Code itself runs on, for building a custom background agent.
- Where agents are going: Anthropic, Building the future of agents with Claude (22 min).
- Infrastructure economics: Neil Movva, Why AI Is About to Get 1000x Cheaper (1 h 23 min; the relevant chapters are long-running agents at `05:32` , the inference stack at `15:12` , throughput versus latency at `20:03` , and transformer hardware at `33:19`). Treat “1000x” as an ambition, not a demonstrated forecast.

Build

Build an issue-to-pull-request background flow. Each job gets an isolated worktree or container, a strict budget, a checkpoint, deterministic validation, and a resumable state. Add retries with bounded backoff and prevent two agents from editing the same work item simultaneously.

Run three jobs in parallel: a small feature, a bug fix, and a documentation task. Interrupt one midway and demonstrate that it can resume or fail cleanly. Agents may open pull requests; only a human may merge.

Done when

- Three parallel jobs ran from three issues and each opened a pull request without editing the others' files.
- The interrupted job either resumed from its checkpoint or failed with a clear state, and the log proves which.
- Each job stayed under its budget, and the budget breach path was tested at least once.
- A comparison note ranks the hosted products above on trigger, isolation, and approval model.

Week 9 — Building an AI-Native Team

Focus: MCP portals and centralized, permissioned tool access; LLM gateways, model routing, and cost optimization; organization-wide adoption patterns.

Core material (≈ 3 h 40 min)

1. Talk: Uber, Agentic SDLC: Building Blocks for Uber's Software Factory (18 min) — model gateways, agent identity, PII redaction, MCP gateways, tool access, remote environments, context graphs, and pre-CI validation.
2. MCP portals: Anthropic's Karan Sampath, Gateways Are All You Need (18 min) and Tobin South, What Does Enterprise-Ready MCP Mean? (14 min) — identity, access policy, observability, provisioning, oversight, and data-loss prevention.
3. Architecture: Cloudflare, Scaling MCP adoption: reference architecture for enterprise deployments (25 min).
4. Authorization: MCP, Enterprise-Managed Authorization and the technical specification (20 min).
5. Evidence on adoption: DORA, 2025 State of AI-assisted Software Development (60 min) — AI amplifies the strengths and weaknesses of the surrounding engineering system. Pair it with the counter-evidence: METR, Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity (20 min).
6. Video: Amjad Masad of Replit, The Future of Software Creation (42 min) — agent infrastructure, secure execution, autonomy levels, and how cheap software creation may reshape teams.

Tools and references

- Gateways for the build: LiteLLM AI Gateway (open source) and Cloudflare AI Gateway. Routing primer: Vercel, Six LLM routing strategies.
- Internal registries: the MCP Registry.
- Adoption evidence: DORA, AI Capabilities Model; Stack Overflow, 2025 Developer Survey: AI; OpenAI, How OpenAI uses Codex (PDF); Anthropic, How Anthropic teams use Claude Code.

Additional video track

- Adoption playbook: Building an Autonomous Engineering Org (18 min) — a maturity model covering champions, repository readiness, delegation, review bottlenecks, isolation, and organizational impact.

Book

- Nicole Forsgren, Jez Humble, and Gene Kim, *Accelerate* — useful for evaluating whether AI adoption improves delivery rather than merely increasing output volume.

Build

Put your model calls behind one gateway or proxy (LiteLLM or Cloudflare AI Gateway). Add per-agent identity, budgets, model routing, fallback behavior, cost and latency logs, and redaction rules. Create a small MCP portal with allowlisted tools, explicit scopes, audit logs, and revocable credentials.

Write a one-page team adoption policy covering approved data, mandatory human decisions, incident ownership, review requirements, and what metrics matter. Prefer change-failure rate, lead time, review burden, cost, and verified task success over lines of code.

Done when

- Every model call in your system goes through the gateway; a direct call is blocked or logged as a violation.
- Routing sends at least one class of request to a cheaper model, and the cost log shows the saving.
- The MCP portal exposes only allowlisted tools, and revoking a credential cuts off access within one run.
- The adoption policy fits on one page and names an owner for incidents.

Week 10 — The Software Factory + The Future

Focus: self-running, self-improving software systems; running and securing agents post-deployment; where AI software engineering goes next.

Core material (≈ 3 h 30 min)

1. Keynote: Andrej Karpathy, Software Is Changing (Again) (39 min).
2. Current case study: Warp, The self-improvement loop in a software factory (20 min).
3. Critical counterargument: Dex Horthy, Harness Engineering Is Not Enough: Why Software Factories Fail (19 min) — brownfield maintainability, weak review, incidents, and architectural decay under high-volume agent output.
4. Post-deployment operations: Always-on agents run production without the on-call tax (25 min), and Google SRE, Postmortem Culture: Learning from Failure (30 min) — the operating discipline that agent-run systems still need.
5. Tracing and evals: Arize AI, How to Debug AI Agents: Tracing, Observability & Evals (19 min) and Arize Phoenix, Your First Traces (30 min hands-on).
6. System design: Revisit OpenAI's Symphony article and specification with the factory build in mind (20 min).

Tools and references

- Evals course: DeepLearning.AI, Evaluating AI Agents (≈ 2 h) — tracing, component and trajectory evaluation, LLM judges, and production monitoring.
- Observability: Langfuse (open source alternative to Phoenix) and the OpenTelemetry GenAI semantic conventions.
- Controlled improvement loops: DSPy — optimize prompts against your eval set instead of editing them by hand.
- Benchmarks to borrow task design from: SWE-bench and Terminal-Bench.
- Google SRE, Introduction.

Additional video track

- Evals, in depth: Hamel Husain and Shreya Shankar on Lenny's Podcast, Why AI evals are the hottest new skill for product builders (1 h 46 min) — error analysis, LLM-as-judge validation, and the eval workflow.
- Software-factory thesis: Eno Reyes, The Software Factory — a timestamped segment on agent readiness, deterministic systems, harnesses, feedback loops, and organizations as capital allocators for agent work.
- The decade of agents: Andrej Karpathy with Dwarkesh Patel, "We're summoning ghosts, not building animals" (2 h 26 min) — why agents still cannot plan, remember, or compound knowledge, and what that means for the next decade.
- Broad practitioner synthesis: Revisit DHH's Future of Programming and Modern Software Engineering, especially `3:59:24` on the future of programming. Use it for perspective, not as a technical authority.

Build

Connect the pieces into a minimal factory:

```
issue -> aligned spec -> isolated agent run -> tests/checks -> independent review -> human merge
```

Trace every run. Create a fixed evaluation set of 10–20 representative tasks and record success, regressions, cost, latency, retries, and human interventions. Add one controlled improvement loop that may propose changes to prompts, skills, or tools, but cannot deploy those changes without evaluation and human approval. "Self-improving" must not mean "silently rewrites its own controls."

Write one blameless postmortem for the worst failure the factory produced during the ten weeks.

Done when

- Every factory run is traceable from issue to merge decision in Phoenix or Langfuse.
- The evaluation set has baseline and final results for at least ten tasks.
- The improvement loop has proposed at least one change, and that change was evaluated before a human approved or rejected it.
- One postmortem exists with a timeline, root cause, and a change that prevents recurrence.

Suggested capstone

Build an agentic maintenance system for a real repository. It should accept a well-specified issue, create an isolated workspace, research and plan the change, implement it, run repository checks, perform an independent AI review, and open a pull request for human approval.

Deliverables

- A repository with one-command setup and a clear architecture map.
- A concise `AGENTS.md` or equivalent repository guide.
- One reusable agent skill and one MCP integration.
- Deterministic hooks, tests, and CI gates.
- A threat model and least-privilege permission design.
- A durable issue-to-PR background workflow with retries.
- Model gateway telemetry for cost, latency, routing, and failures.
- A fixed evaluation suite and a small results dashboard or report.
- A five-to-ten-minute demo and a retrospective describing failures and improvements.

Self-imposed completion gates

- A fresh clone can be set up and validated with one documented command.
- At least ten fixed evaluation tasks have baseline and final results.
- One interrupted background job demonstrably resumes or fails cleanly.
- Generated changes cannot bypass tests, security checks, or human merge approval.
- Every run is traceable to its issue, spec, prompts/context, tool calls, outputs, costs, and final review decision.

The short bookshelf

Do not try to read a dozen books cover to cover. These five cover the durable ideas behind this guide:

1. Chip Huyen, *AI Engineering* — building and evaluating applications with foundation models.
2. Jay Alammar and Maarten Grootendorst, *Hands-On Large Language Models* — practical LLM foundations with an accompanying code repository.
3. Titus Winters, Tom Manshreck, and Hyrum Wright, *Software Engineering at Google* — free online; especially code review, testing, dependency management, and large-scale change.
4. Adam Shostack, *Threat Modeling: Designing for Security* — security reasoning that remains useful when the attack surface includes agents and tools.
5. Nicole Forsgren, Jez Humble, and Gene Kim, *Accelerate* — measuring whether a development system actually improves.

If you only have half the time

Keep the builds and reduce passive material. Use this minimum path:

1. Thorsten Ball's How to Build an Agent, then one production prompt file from the Codex repository.
2. Anthropic's Effective context engineering and Writing effective tools, plus the MCP specification.
3. Dex Horthy's Context Engineering interview.
4. Anthropic's Claude Code best practices and the hooks reference.
5. OpenAI's Harness Engineering case study.
6. Google's Code Review guide.
7. OWASP MCP Top 10, the lethal trifecta, and two PortSwigger labs.
8. OpenAI Symphony and the Uber Agentic SDLC talk.
9. DeepLearning.AI's Evaluating AI Agents course.

The practical work is the guide. Watching everything without building the cumulative system will teach vocabulary; implementing it will teach engineering judgment.