IISc CCE · APPLIED AI · WEBINAR

Agentic AI Coding

From Specification to Deployment

How coding agents actually work, why the discipline around them matters,
and a 6-weekend programme to build it properly.

Prof. Deepak Subramani
Associate Professor, Computational and Data Sciences · Indian Institute of Science, Bangalore

ROADMAP

What we'll discuss in this webinar

1

What is agentic AI coding?

The loop these systems run, the harness around the model, and the discipline that makes them safe to point at real work.

2

Why should you study it?

What actually changes in an engineer's job, and what separates the people who thrive from the people who just prompt.

3

The IISc CCE course

Six weekends, the week-by-week plan, what you will need, and exactly how to enrol.

PART 1

What is agentic AI coding?

From a chat window that returns text to a system that reads your repository, edits files, runs your tests, and decides its own next step.

- The loop
- The harness
- Vibe coding
- Spec-driven development
- Where it breaks

THE MECHANICS

One goal in, many self-chosen steps

A chat interface takes a prompt and returns text. A coding agent takes a goal and runs a loop until the work is done.

1 **PERCIEVE**

Gather context

Reads files, searches the codebase, checks git state — before touching anything.

2 **REASON: PLAN & ACT**

Take action

Edits files, runs commands, performs git operations. Each result feeds the next decision.

3 **VERIFY & ITERATE**

Verify results

Runs the tests, inspects the diff — and goes round again if something is wrong.

A system that runs this loop on its own initiative until a Goal is satisfied is a fully autonomous agent.

Every property that makes it powerful also lets it do damage quickly.

Scientific Approach of handling Agentic AI Coding is an essential skill of 2026

VOCABULARY

The harness, not just the model

Agentic harness

The software layer around a model that supplies tools, memory and an execution loop. The model supplies reasoning; the harness decides which tools exist, how results feed back, what counts as “done”, and when to stop.

Where you will meet them

Claude Code and Codex in the terminal · Cursor and Antigravity as agentic IDEs · Devin · VS Code agent mode. The same pattern now wraps non-coding work too.

Why it matters which one

A weaker model's success rate swings widely with the harness it is given. A good harness matters most exactly where the model is weakest.

Memory aid

A horse harness does not make the horse stronger.

It converts raw power into directed work — a specific cart, along a specific road, instead of running wherever it likes.

The harness channels a model's reasoning toward a specific goal, with specific tools, along a checkable path.

HOW WE WILL STUDY IT

Five layers of engineering, one discipline

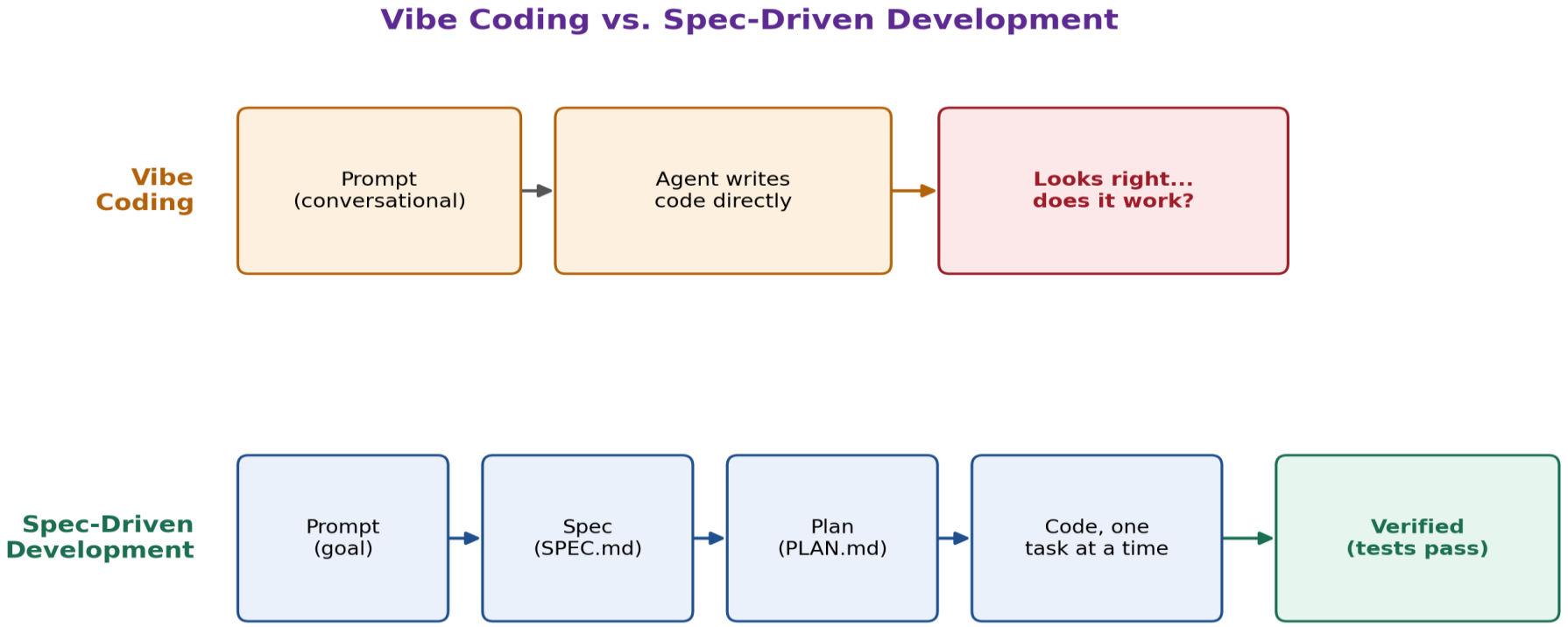
Each layer widens the scope of what you are designing — from a single message to the whole system around the model.

1	**Prompt**	ONE MESSAGE	How you state the goal: wording, examples, output format, what to include and what to leave out.
2	**Context**	ONE SESSION	What the model can see: project memory, retrieved files, tool results — and what must be kept out.
3	**Loop**	ONE TASK	When to act, when to verify, when to stop — and where a human approval gate belongs.
4	**Graph**	MANY STEPS	How agents and tools are wired into a workflow: orchestrator and workers, branches, parallel runs.
5	**Harness**	THE WHOLE SYSTEM	Tools, permissions, hooks, MCP servers, skills — and the pipeline the whole thing plugs into.

Prompt, context, loop, graph, harness. **Together these are the scientific way of working with coding agents** — and they are what the six weekends are built around.

Vibe Check!!!

Vibe coding to Spec-Driven Development



Vibe coding

Describe the intent, accept the code, iterate on what looks wrong. Fast, low-friction, and right for prototypes and throwaway scripts.

Anything meant to last

A real feature carries thousands of unstated requirements. Left to guess, the agent guesses reasonably — and some guesses are wrong in ways nobody sees until production.

SPEC-DRIVEN DEVELOPMENT

Intent becomes the source of truth

- You are the architect:
  - write the spec and review the decisions.
- The agent is the contractor:
  - it plans, builds, and reports back.
- The spec is the contract
  - SKILL TO LEARN: give precise goals, clear pass criteria

THE LANDSCAPE: WHICH TOOL?

Everyone converged on the same shape

GitHub, AWS and Anthropic each shipped a version of this within months of each other. The convergence is the signal — not any one vendor's tool.

GitHub Spec Kit

Four slash commands, three artifacts. Agent-agnostic: a living spec a whole team works from.

AWS Kiro

Requirements written in a constrained syntax borrowed from jet-engine control software, so each one becomes a test.

Claude Code

Conversational, with a plan mode that mechanically blocks edits until a human approves the plan.

Cursor · Devin

A disposable plan you approve and throw away — or a ticket that is itself treated as the spec.

Whichever tool your team standardises on, the same five phases are underneath the branding.

That was the map, not the territory

Every box on that map is a working session in its own right. The parts that decide whether this works on real code are the parts we have not opened:

1. What makes a spec an agent can actually build against
2. Acceptance criteria a machine can check, not a human's opinion
3. Project memory, skills, subagents, hooks and MCP servers
4. Orchestrating many agents in one session without collisions
5. Running the whole workflow headless, inside a CI pipeline
6. Evaluating a coding agent: outcomes and trajectories

PART 2

Why study this, and why now?

The tooling is free and the demos are everywhere. What is scarce is the judgment to use it on work that matters.

- What changes in your job
- The understanding gap
- Why not by osmosis
- Questions to sit with

THE SHIFT IN YOUR WORK

The job moves up a level — it does not disappear

~~Writing code~~	→ **Specifying outcomes**	The scarce skill becomes stating precisely what “done” means, in terms a machine can check.
~~Reviewing lines~~	→ **Reviewing trajectories**	Reading a plan before it runs catches problems a diff review finds far too late.
~~Personal habits~~	→ **Enforced discipline**	The rules that matter stop being things you remember and become things the tooling holds you to.
~~Shipping features~~	→ **Owning what shipped**	Whoever commits agent-written code answers for it. That responsibility did not move.

THE UNCOMFORTABLE PART

Working code is not evidence of understanding

A session can pass every test while the person who approved each plan never truly follows why the model landed on that architecture.

Passing tests confirms the code behaves correctly. It says nothing about whether the reviewer could have written or debugged it unaided.

An ACM report on computing degree programmes records the same worry from instructors: people who lean on generated code without engaging with it lose exactly the abilities that assessments — and employers — test for.

There are ways to close this gap

Two concrete techniques turn “I should probably understand this” into something you can actually check — one that reconstructs the reasoning behind a finished codebase, and one that tests whether you can defend it unaided.

Neither needs new tooling. Both are things you will practise in the course, on code you wrote with an agent that week.

WHY THIS, WHY NOW

Four reasons to learn it deliberately

1 The adoption is already here

Software is, by a wide margin, where agentic systems have seen their fastest real-world adoption. This is not a bet on the future.

2 Demos are not workflows

A short tour shows you a tool. It does not teach you what to do when the plan is wrong, the spec is ambiguous, or the diff spans forty files.

3 The discipline transfers

Spec, plan, verify, iterate outlives any one vendor. Learn it once and it survives whichever harness your employer standardises on next.

4 Judgment is the differentiator

Knowing when to vibe-code and when not to, and being able to defend what an agent built, is what separates a practitioner from a prompt-user.

A FAIR TEST

Could you answer these?

1. A task is halfway between a prototype and production. Do you vibe-code it or write a spec first — and what tells you which?
2. An agent has just touched forty files and every test passes. What do you review, and in what order?
3. A test fails because the spec was ambiguous. What exactly do you change, and what must you not change?
4. Your agent needs a rule that must hold every single time. Where does that rule live so the model cannot skip it?
5. You are asked to defend an architecture an agent chose for you. How do you find out whether its explanation is the real reason?

If these feel like the right questions but not yet answerable ones, that is precisely the gap the next six weekends are built to close.

PART 3

The course: six weekends

Agentic AI Coding: From Specification to Deployment — IISc Centre for Continuing Education.

- At a glance
- Week by week
- What you'll need
- How to enrol

IISC CCE

Agentic AI Coding: From Specification to Deployment

6

weekends

Sept 5 – Oct 10, 2026

12

live sessions

Sat & Sun, 10:30am–12pm

100%

online

Live on Zoom, sessions recorded

A hands-on programme for people who want to build, ship and operate real AI agents — from writing a spec to deploying a full-stack application.

Certificate from IISc CCE for participants who finish the course and the final project.

WEEK BY WEEK

Six weekends, spec to deployment

1

Sept 5–6

What is Agentic AI?

Levels of autonomy: augmented LLM calls, agentic workflows, single- and multi-agent systems, fully autonomous agents.

2

Sept 12–13

Harness & spec-driven development

Writing high-quality specs, vibe coding vs. SDD, and writing correct tests.

3

Sept 19–20

Full-stack build

An end-to-end web application with agentic coding — front-end and back-end specs. Self-project begins.

4

Sept 26–27

Testing, CI/CD, deployment

Full unit tests, continuous integration and cloud deployment of the Week 3 application.

5

Oct 3–4

Principled evaluation

Evaluating the agentic coding lifecycle — the AI-driven software development lifecycle.

6

Oct 10–11

System design & demos

High-level system design for new applications, and final project presentations.

OUTCOMES

What you will learn — and what you will use

By the end of six weekends

1. How coding agents plan, execute and self-correct across multi-step tasks — not just single-shot completions
2. Real workflows with today's agentic coding tools, beyond the demo-level tour
3. Tool use and orchestration: what an agent needs to reliably read, write, run and test code
4. Building a coding agent of your own — from prototype to something you would trust on a real task
5. Debugging and evaluating agent behaviour: where these systems fail, and how to catch it

Tools you will use

- Claude Code · Claude Sonnet 5 & Opus 5 · Anthropic API
- ChatGPT & OpenAI API · Gemini & Gemma
- LangChain & LangSmith · Agent Development Kit
- Cursor · Model Context Protocol (MCP)
- Python, React, FastAPI, SQL, Redis
- Docker · Vercel · Vector databases

FIT & PREREQUISITES

Who it is for, and what you will need

Who this is for

- Software professionals in product or services
- Engineers wanting to supercharge their work with AI
- Start-up teams, national lab and academic researchers
- UG, PG and research students or staff
- Career-break returners, hobby developers, and anyone aspiring to join the AI industry

Hardware

- A laptop (Windows, macOS or Linux) with at least 8GB RAM
- A stable internet connection
- No GPU required, and no Zoom account needed — just a browser

Eligibility

- B.Tech, BCA or B.Com minimum — students welcome to apply
- A genuine interest in software development
- No prior AI/ML background required

Subscriptions & accounts

- A GitHub account
- Claude Pro or ChatGPT Plus (about ₹2,000 + taxes/month) — either works
- A free-tier cloud account (AWS/GCP/Azure) and a free Vercel account

NEXT STEP

How to enrol

1	**Create an account**	cce.iisc.ac.in/registration
2	**Update your profile**	with biodata information
3	**Search the course**	“Agentic AI Coding”
4	**Click Apply Now**	on the course page
5	**Follow the instructions**	on screen, and pay to confirm

Seats

First come, first served

Confirmed on payment. Enrolling early gives you more time with the setup material.

Questions

Course content
questlabiisc@gmail.com

Admissions & payment
office.cce@iisc.ac.in

Post Enrollment

What will you get on enrolling?

1. Access to the class communication channel
2. Study Material – Textbook by Prof. Deepak Subramani; Hands-on guides
3. Recording, notes and highlights/summary after every class
4. Weekly Practice Problems – To stay on top of things
5. Advice on your project

Project

Bring Your Own Project

Or do a project from the list that we provide

Office Hours with the Professor

AI Clinic

You can book 1-on-1 or many-on-1 office hours with Prof. Deepak during the course duration for advice and doubt clearance

SEPT 5 – OCT 10, 2026 · SAT & SUN, 10:30AM–12PM · ONLINE

Ready to build with agentic AI?

The tools are already in your terminal. The discipline is what turns them into engineering.

Course page
deepaksubramani.in/agentic-ai-coding

Register
cce.iisc.ac.in/registration

Ask us
questlabiisc@gmail.com

Thank you — questions welcome.