M A X I M I Z I N G

Claude Code

for Software Engineering

Participant Handbook

19 May 2026 · 2-hour workshop

Contents

1. Welcome & how to use this handbook

This handbook accompanies a 2-hour workshop. It is yours to keep, mark up, and refer back to whenever you sit down with Claude Code on a real project.

How to read it

- Each section maps to a workshop segment. You can follow along in real time.
- Code blocks are runnable — copy them straight into your terminal.
- The 5 labs have dedicated sheets in a separate PDF (03_Lab_Exercises.pdf).

> **Tip**
> If your time is limited, read sections 5 (Context), 6 (MCP), and 8 (Skills). They contain ~80% of the practical leverage.

2. Pre-workshop setup checklist

Please complete this 3 days before the workshop. If you hit a problem, ping the workshop trainer — fixing it on workshop day costs everyone time.

Required

1. Node.js 18 or later installed. Verify: node --version
2. Claude Code CLI installed. Install: npm install -g @anthropic-ai/claude-code
3. Claude account, logged in. Run: claude (it will prompt you to log in).
4. Git installed. Verify: git --version
5. A code editor (Cursor recommended).
6. Workshop materials zip downloaded and extracted to a known folder.

Pre-installed MCP (recommended)

Lab 2 (installing the codebase-memory MCP) is much faster if you do it the day before. The lab then becomes a 5-minute verification, not a 20-minute installation race.

```
# 1. Clone the MCP
git clone https://github.com/deusdata/codebase-memory-mcp.git
cd codebase-memory-mcp
npm install

# 2. Register with Claude Code
claude mcp add codebase-memory \
  --command "node $(pwd)/server.js"

# 3. Verify
claude        # then in the prompt:  /mcp
```

3. Glossary

Term	Meaning
Prompt	The instruction you send to the model.
Context	Surrounding information that shapes the answer.
Context window	The total information the model can read at once.
MCP	Model Context Protocol — gives Claude access to external tools and data.
Skill	A reusable, named workflow Claude can invoke.
Agent	An autonomous Claude instance running a multi-step task.
CLAUDE.md	A markdown file in your repo that tells Claude about the project.
RAG	Retrieval-Augmented Generation — fetching relevant info before generating.
Token	The unit AI models read and produce — roughly 0.75 of an English word.
Hallucination	When the model confidently invents something untrue.

4. The reality of AI in software engineering

Workshop opening segment in writing. Read it once, then move on.

The structural shift

Tech layoffs from 2022–2026 are not cyclical. Oracle's announced 30,000+ headcount reductions, Meta's 21,000+ since 2022, Google's 12,000+ — these are restructurings around AI leverage, not temporary belt-tightening.

What this means for you

- The bar moved. "I can code" is no longer the job. "I can multiply my output" is the job.
- AI fluency separates engineers who keep their seat from those who become the saving.
- The good news: the tools are open, cheap, and improving fast.

Mindset
Stop thinking of AI as a faster autocomplete. Think of it as a teammate that needs context, conventions, and judgment from you — and gives back leverage in return.

5. Context engineering 101

Prompt engineering is the visible 10%. Context engineering is the invisible 90% that decides whether the answer is useful.

The four layers

1. Per-prompt context — what you type in the moment. Throwaway.
2. Repo memory (CLAUDE.md) — project conventions, stack, decisions. Permanent.
3. Skills — reusable workflows. Sharable across projects.
4. MCP servers — live data and tools. Bridges Claude to your real systems.

Each lower layer compounds. The bottom is permanent. The top is throwaway.

Example — bare prompt vs context-rich prompt

Bare prompt:

```
Write me a function to validate email.
```

Context-rich prompt:

```
Project: BookShelf (Node.js + Express). CommonJS, not ES modules.
Existing validators live in utils/helpers.js as named exports
(addDays, format_date, calc_fine). Follow that pattern.
Error handling: return { ok: false, error: "..." }. Do NOT throw.
Accept: standard RFC 5322-ish. Reject: empty, no @, no dot, spaces, > 254.
Called from auth.js during register and login.
No new dependencies — pure JS only.

Task: add validate_email(email) to utils/helpers.js. Update exports.
Show me the diff only. Include 5 inline test calls.
```

CLAUDE.md template

Drop this file at the root of any repo. Claude Code reads it automatically.

```
# Project: <name>

## Stack
- Language: <e.g. Node.js 18, TypeScript 5>
- Framework: <e.g. Express, Next.js 14>
- Database: <e.g. PostgreSQL, SQLite>

## Conventions
- Module system: <CommonJS | ESM>
- Error handling: <throw | return { ok, error }>
- Naming: <camelCase | snake_case>
- Tests: <jest | vitest | none>
```

```
## Where things live
- Routes:        <path/to/routes>
- Business logic: <path/to/...>
- Helpers:        <path/to/...>

## Don't do
- <e.g. don't introduce new npm dependencies without asking>
- <e.g. don't break the existing auth flow>
```

6. codebase-memory MCP guide

An MCP that indexes a codebase into a queryable knowledge graph. Once indexed, you can ask Claude natural-language questions and get answers grounded in your real source.

What it tracks

- Functions and their callers
- Modules and their dependencies
- Routes, endpoints, schemas
- Configuration entry points
- Cross-file references

Useful questions

- "How does authentication work in this repo?"
- "Where is the loan flow handled?"
- "What touches the books table?"
- "Show me all callers of calc_fine."
- "Where do we validate user input?"

Install

```
git clone https://github.com/deusdata/codebase-memory-mcp.git
cd codebase-memory-mcp && npm install
claude mcp add codebase-memory --command "node $(pwd)/server.js"
```

First index

```
cd /path/to/your/project
claude
# inside the Claude Code prompt:
> /codebase-memory index
# wait for it to scan. Ask anything:
> Explain how the loan flow works in this repo.
```

When to re-index
Re-run /codebase-memory index after any significant refactor, when you switch branches, or when new contributors land big changes.

7. Daily SDLC workflow

Before we optimize, let's name what we actually do every day:

Role	Output	Pain point
Software Analyst	Spec, acceptance criteria, edge cases	Specs become stale during build
Backend Developer	API, data, business logic	Re-asks analyst, blocks frontend
Frontend Developer	UI, state, integration	Waits on backend; types out specs by hand
QA	Test cases, regression, sign-off	Discovers context gaps last

The hidden cost

Every handoff loses information. Every loop adds rework. Senior engineers spend more time re-explaining than building.

8. Converting your workflow into a Skill

A skill is a markdown file with YAML front matter. Claude reads it, knows when to use it, and follows the steps. Sharable, versionable, reviewable.

SKILL.md template

```
---
name: bug-fixer
description: |
  Use when fixing a bug ticket. Investigate code, propose fix,
  write test, open PR.
triggers: ["fix bug", "BUG-", "resolve ticket"]
---

# Steps
1. Read the ticket and acceptance criteria.
2. Use codebase-memory to find related code.
3. Propose a minimal patch. WAIT for human approval.
4. Apply the patch and add a regression test.
5. Run tests. Iterate until green.
6. Open a PR with a summary of what changed and why.

# Constraints
- Never modify auth.js without explicit approval.
- Never bump dependencies as part of a bug fix.

# Examples
## Good run
Ticket: BUG-002. Investigated routes/loans.js, found the
decrement was commented out. Re-enabled inside a transaction.
Added a test that asserts available_copies goes to 0 after the
last loan and the next borrow is rejected.
```

Where to put it

- Project-scoped: ./.claude/skills/<skill-name>/SKILL.md
- User-scoped: ~/.claude/skills/<skill-name>/SKILL.md
- Team-shared: check the .claude/ directory into git

Anatomy checklist

- Name — short, descriptive, kebab-case.
- Description — what it does + when to use it. The first 2 lines matter most.
- Triggers — phrases that should auto-activate the skill.
- Steps — numbered, atomic, in order. 5–10 is the sweet spot.
- Constraints — what NOT to do. Explicit beats clever.

- Examples — at least one good run, ideally one bad-run-corrected.

9. agent-dev-team reference

A real skill, built by the workshop trainer, that runs Analyst → Backend → Frontend → QA on one ticket — in sequence. Take it, fork it, replace any role with your team's own conventions.

Where to get it

```
https://uscloud3.dxn2u.net:3000/cipta/agent-dev-team
```

How it's structured

- Stage 1 — Analyst: clarifies acceptance criteria, lists assumptions, flags edge cases.
- Stage 2 — Backend: reads codebase-memory, proposes changes, asks for approval before writing.
- Stage 3 — Frontend: integrates with the backend's contract. Skips this stage if the change is backend-only.
- Stage 4 — QA: writes tests, runs them, reports pass/fail with reproductions.

10. Automation cookbook

A skill is reusable on demand. An automated skill runs without you. Below is the pattern showcased in the last segment of the workshop.

Pattern: email-triggered agent run

1. A watcher (Cowork mode / Hermes agent) polls a designated inbox folder every N minutes.
2. When a new email matches a rule (e.g., subject starts with [FEEDBACK]), it extracts the attachment.
3. The attachment is passed to Claude Code with the agent-dev-team skill as context.
4. Claude Code investigates, makes changes on a feature branch, opens a PR, and runs tests.
5. Upon completion, an auto-reply email is sent with a summary and links to the PR.
6. A human reviews and merges (or rejects).

Guardrails

Guardrail	Why
Human-in-the-loop merge	Auto-commit to a branch, never main.
Audit trail	Every action logged, traceable to a source email.
Cost caps	Daily budget per agent. Hard stop on runaway loops.
Sandboxed write access	Agents work in dedicated repos / branches / containers.
Kill switch	One command to disable all automation.

Smaller variants to start with

- Daily 6am report: "What changed in our repo yesterday?" → posted to Slack.
- PR triage bot: when a PR is opened, run a skill that adds review comments.
- Weekly digest: scan support tickets, group by root cause, draft summary email.

11. Resources & next steps

Links

Resource	URL
Claude Code home	https://claude.com/claude-code
Claude Code docs	https://docs.claude.com/en/docs/claude-code/overview
Claude Code skills	https://docs.claude.com/en/docs/claude-code/skills
MCP overview	https://modelcontextprotocol.io
Prompt engineering guide	https://docs.claude.com/en/docs/build-with-claude/prompt-engineering/overview
codebase-memory MCP	https://deusdata.github.io/codebase-memory-mcp/
agent-dev-team (Cipta)	https://uscloud3.dxn2u.net:3000/cipta/agent-dev-team

Your first week

1. Today — Install Claude Code on your work machine.
2. Day 1 — Index ONE codebase with codebase-memory MCP.
3. Day 2 — Write CLAUDE.md for that repo. Commit it.
4. Week 1 — Convert ONE recurring task into a skill.
5. Week 2 — Share the skill with one teammate.
6. Month 1 — Hook one automation trigger into your inbox or chat.