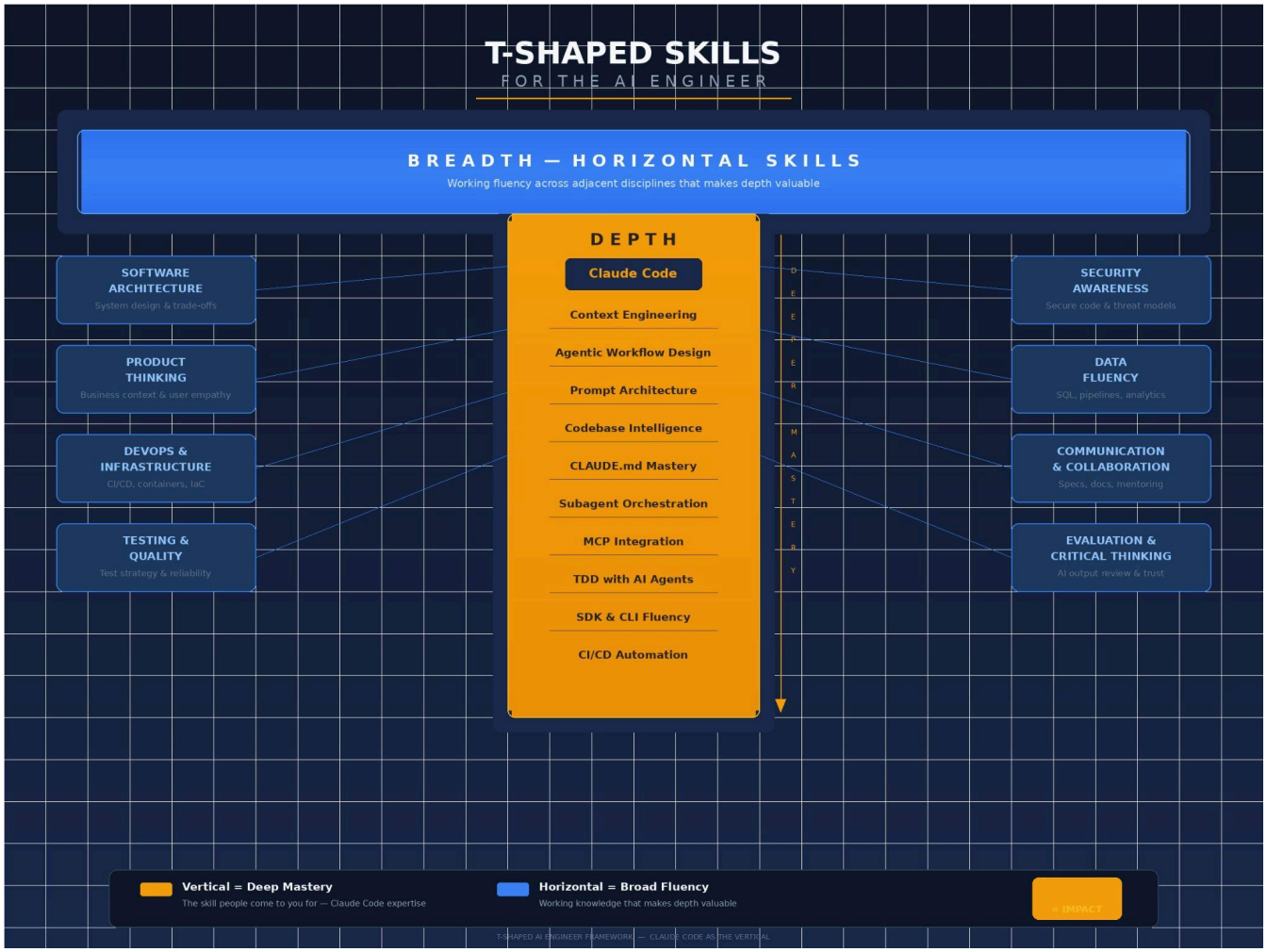

T-SHAPED SKILLS

FOR THE AI ENGINEER

With Claude Code as the Deep Vertical

Created by Ajit Jaokar using Claude Opus 4.6

Background

I created this document for one of my students who asked for reskilling advice towards an AI Engineer. I initially suggested the idea of a T shaped skills professional but then today ie in Feb 2026 I thought the whole role of the AI Engineer could be oriented around the idea of Claude Code. Conceptually, this could apply to any similar platform as well.

I intentionally chose anti patterns as well.

A Framework for Building Mastery in the Age of Agentic AI

February 2026

1. The Concept: Why T-Shaped Skills Matter Now

The T-shaped skills model, first popularised by McKinsey in the early 1990s, describes a professional who combines **deep expertise in one domain** (the vertical bar) with **broad working knowledge across many adjacent disciplines** (the horizontal bar). The vertical gives you credibility, speed, and mastery. The horizontal gives you context, adaptability, and the ability to collaborate across boundaries.

In 2025–2026, the AI engineering landscape has shifted dramatically. Agentic coding tools are no longer experimental—they're infrastructure. Engineers who previously shipped features in weeks now ship in hours. But the engineers who thrive aren't the ones who use AI casually; they're the ones who have gone deep on a specific AI-augmented workflow and built genuine mastery around it.

This document proposes a specific T-shape for the modern AI engineer: Claude Code as the deep vertical, surrounded by a wide horizontal bar of complementary skills that make Claude Code mastery genuinely powerful.

> **The Core Thesis**
> Claude Code is not just a tool—it's an operating environment for software engineering. Going deep on Claude Code means mastering agentic coding, context engineering, prompt architecture, and AI-human collaboration patterns. The horizontal bar ensures you can apply that depth to real problems across the full stack.

2. The T Visualised

The T-shape has two distinct components that reinforce each other:

Component	What It Represents	For the AI Engineer
Vertical Bar (Depth)	Deep, hard-won expertise in one area—the thing people come to you for	Claude Code mastery: agentic workflows, context engineering, prompt architecture, CLAUDE.md configuration, subagent orchestration
Horizontal Bar (Breadth)	Working-level fluency across adjacent disciplines—enough to collaborate, unblock, and see the full system	Software architecture, DevOps/CI-CD, product thinking, security, data fluency, testing strategy, system design, communication

The key insight: without the vertical, you're a generalist who can talk about everything but ship nothing extraordinary. Without the horizontal, you're a power user in a vacuum, unable to connect your Claude Code skills to the problems that actually matter.

3. The Vertical: Claude Code Mastery

Going deep on Claude Code means far more than knowing the CLI commands. It means understanding how to think in an agentic paradigm—where you're not writing code line by line, but orchestrating an AI agent that understands your codebase, executes multi-step tasks, and maintains context across complex projects.

3.1 Core Competencies of the Vertical

Context Engineering

Context engineering is the art of designing the information environment so that Claude produces accurate, aligned outputs. This is the single most important skill for Claude Code mastery and involves structuring CLAUDE.md files, project instructions, and memory configurations so the agent has the right knowledge at the right time.

- **Crafting effective CLAUDE.md files** that encode project conventions, architecture decisions, and coding standards
- **Managing context windows strategically**—knowing what to include, what to exclude, and how to structure information for maximum relevance
- **Designing prompt hierarchies** that layer system-level instructions with task-specific directives
- **Building persistent knowledge systems** through project memory and instruction files that compound over time

Agentic Workflow Design

Claude Code operates as an autonomous agent capable of multi-file operations, codebase analysis, testing, and deployment. Mastery here means knowing how to decompose complex engineering tasks into agent-friendly workflows.

- **Task decomposition:** Breaking large features into logical steps that Claude Code can execute with appropriate checkpoints
- **Subagent orchestration:** Using Claude Code's subagent capabilities to parallelise work across multiple streams
- **Human-in-the-loop design:** Knowing when to give Claude autonomy and when to insert approval gates for critical decisions

- **Recovery and correction patterns:** Efficiently redirecting the agent when it goes off track without losing accumulated context

Prompt Architecture

Beyond basic prompting, this is about building repeatable, composable prompt patterns that produce reliable results across different types of engineering tasks.

- **Writing specification-grade prompts** for feature implementation that include acceptance criteria, edge cases, and constraints
- **Developing slash commands and custom workflows** that encode team practices into reusable patterns
- **Test-driven prompting:** Writing tests first and instructing Claude Code to implement against them, leveraging AI's affinity for TDD
- **Iterative refinement techniques:** Building on Claude Code's outputs through targeted feedback rather than rewriting from scratch

Codebase Intelligence

Claude Code's power scales with its understanding of your codebase. Deep practitioners know how to maximise this understanding.

- **Structuring repositories** so they're legible to AI agents—clear naming, consistent patterns, well-organised modules
- **Leveraging Claude Code for codebase archaeology:** Understanding unfamiliar code, tracing dependencies, and mapping system behaviour
- **Multi-repository workflows:** Working across multiple codebases and coordinating changes that span services

Tool & Integration Mastery

Claude Code connects to the broader development ecosystem through MCP (Model Context Protocol) servers and native integrations.

- **MCP server configuration:** Connecting Claude Code to Git, GitHub, Jira, databases, and internal tools
- **CI/CD integration:** Running Claude Code in automated pipelines for code review, test generation, and deployment tasks
- **SDK and API usage:** Using the Claude Code SDK (Python and TypeScript) to build custom workflows and integrations
- **Headless and background operation:** Running Claude Code non-interactively for batch tasks, automated refactoring, and scheduled maintenance

3.2 Vertical Depth Maturity Model

Mastery doesn't happen overnight. Here's a progression model for Claude Code depth:

Level	Description	Key Indicators
Level 1: User	Uses Claude Code for discrete tasks—ask a question, get an answer, copy-paste the result.	Single-turn interactions, basic prompts, manual file management
Level 2: Practitioner	Uses Claude Code for multi-step workflows. Understands how to give context, iterate on outputs, and leverage agent autonomy.	CLAUDE.md files in projects, multi-turn sessions, uses approval gates intentionally
Level 3: Architect	Designs agentic workflows for the team. Builds prompt libraries, custom commands, and integration patterns.	Team-wide CLAUDE.md standards, custom slash commands, CI/CD integration, measurable productivity gains
Level 4: Expert	Pushes the boundaries of what's possible. Orchestrates subagents, builds novel workflows, contributes to best practices.	Multi-agent orchestration, custom MCP servers, novel workflow patterns, thought leadership

4. The Horizontal Bar: Breadth That Makes Depth Valuable

The horizontal bar is what transforms Claude Code mastery from a party trick into a career-defining capability. Each of these skills amplifies the vertical, and Claude Code amplifies each of them in return.

4.1 Software Architecture & System Design

Why it matters: Claude Code can implement any architecture you describe, but it can't choose the right one for you. Understanding trade-offs between monoliths and microservices, knowing when to use event-driven patterns, and reasoning about data flow and system boundaries—these are decisions that require human judgement.

How Claude Code amplifies it: Once you make the architectural decision, Claude Code can scaffold entire service structures, implement the patterns consistently, and refactor existing code to match the new design—across multiple files simultaneously.

4.2 Product Thinking & Business Context

Why it matters: The most dangerous AI engineer is one who can ship features at incredible speed without understanding what to build. Product empathy—the ability to translate business needs into technical decisions—ensures your Claude Code output creates value rather than just velocity.

How Claude Code amplifies it: When you understand the product context, you can write better specifications for Claude Code. Instead of "add a button," you can say "add a one-click export feature for enterprise users who need to generate compliance reports weekly." The agent produces dramatically better code when given product-aware context.

4.3 DevOps, CI/CD & Infrastructure

Why it matters: Agentic coding doesn't end at writing code. Understanding deployment pipelines, infrastructure-as-code, containerisation, and observability means you can use Claude Code across the full software lifecycle, not just the coding phase.

How Claude Code amplifies it: Claude Code can write Dockerfiles, configure GitHub Actions, generate Terraform modules, and debug deployment issues—but only if you know enough about the infrastructure to guide it correctly and validate its outputs.

4.4 Testing Strategy & Quality Engineering

Why it matters: AI-generated code needs more testing, not less. Understanding testing pyramids, property-based testing, integration test design, and mutation testing ensures that the code Claude Code produces is actually reliable.

How Claude Code amplifies it: Test-driven development has had a renaissance with Claude Code. Write the tests first, describe the expected behaviour, and let the agent implement. Claude Code is remarkably good at generating comprehensive test suites when given clear specifications.

4.5 Security Awareness

Why it matters: Claude Code sends code context to external servers and can access environment variables and configuration files. Understanding supply chain security, secrets management, access control, and common vulnerability patterns is essential for responsible use.

How Claude Code amplifies it: Security-aware engineers can instruct Claude Code to follow secure coding patterns, audit generated code for vulnerabilities, and configure sandboxed environments that limit exposure.

4.6 Data Fluency

Why it matters: Knowing how to clean, structure, query, and reason about data is essential for building anything that touches databases, analytics, or ML pipelines. Data fluency also means understanding bias, data quality, and the limitations of AI training data.

How Claude Code amplifies it: Claude Code excels at writing SQL queries, building data pipelines, and transforming data—but only produces correct results when you can validate the logic and ask the right questions about the data.

4.7 Communication & Collaboration

Why it matters: As AI handles more of the code writing, the engineer's role shifts toward specification, review, and communication. The ability to write clear technical specs, explain trade-offs to non-technical stakeholders, and mentor others on AI-augmented workflows becomes the differentiator.

How Claude Code amplifies it: Claude Code can generate documentation, draft RFCs, write release notes, and help prepare presentations—but the strategic communication decisions remain fundamentally human.

4.8 Evaluation & Critical Thinking

Why it matters: The ability to evaluate AI-generated code is what distinguishes a productive AI engineer from a reckless one. This means reading generated code critically, understanding when the agent has made subtle errors, and knowing when to trust and when to verify.

How Claude Code amplifies it: Strong evaluators can move faster because they can give Claude Code more autonomy with confidence, reviewing outputs efficiently rather than re-doing the work.

5. The Synergy: How the T-Shape Compounds

The real power of the T-shape isn't in either dimension alone—it's in the multiplication effect between them. Here are concrete examples of how vertical depth and horizontal breadth create outsized impact:

Scenario	Vertical Only	Vertical + Horizontal
Feature Implementation	Quickly generates working code, but may miss architectural implications or edge cases in production	Writes product-aware specs, generates architecturally sound code, includes comprehensive tests, and produces deployment-ready output
Bug Investigation	Uses Claude Code to grep and trace, but may not understand the system-level context	Combines codebase intelligence with system design knowledge to identify root causes and fix them holistically
Technical Debt Reduction	Can refactor files quickly but may introduce inconsistencies across services	Plans refactoring with architectural coherence, uses Claude Code for execution, validates with integration tests
Incident Response	Good at reading logs and suggesting fixes	Combines infrastructure knowledge, security awareness, and agentic debugging to resolve incidents end-to-end
Team Onboarding	Sets up Claude Code for new engineers	Builds CLAUDE.md files encoding team practices, creates onboarding workflows, and establishes shared prompt libraries

6. Building Your T: A Practical Roadmap

Building a T-shape is a multi-year journey, but you can make meaningful progress in months. Here's a phased approach:

Phase 1: Foundation (Weeks 1–4)

Get comfortable with Claude Code as your daily driver. Install it, configure your first CLAUDE.md file, and start using it for real work—not toy examples.

- Install Claude Code and use it for at least 2–3 hours daily on real tasks
- Create a CLAUDE.md for your primary project with coding standards and architecture notes
- Practice the feedback loop: prompt, review, refine, learn
- Read the official Claude Code documentation thoroughly

Phase 2: Deepen the Vertical (Months 2–3)

Move from user to practitioner. Start designing workflows, not just executing tasks.

- Build custom slash commands for your most common workflows
- Experiment with test-driven prompting: write tests first, let Claude implement
- Configure MCP servers to connect Claude Code to your team's tools
- Start tracking what works: maintain a personal prompt library
- Use Claude Code for increasingly complex tasks: multi-file refactors, feature branches, code reviews

Phase 3: Widen the Horizontal (Months 3–6)

Identify the horizontal skills that would most amplify your Claude Code depth, and invest deliberately.

- If you lack architecture knowledge: study system design, practice trade-off analysis
- If you lack DevOps skills: use Claude Code to learn—have it generate Dockerfiles and CI configs, then study what it produces
- If you lack product thinking: sit in on product meetings, read user research, understand the business context of your code
- If you lack testing depth: explore property-based testing, mutation testing, and integration test design

Phase 4: Multiply (Month 6+)

Become the person who makes the whole team better at AI-augmented engineering.

- Establish team-wide CLAUDE.md standards and best practices
- Build and share reusable prompt patterns and workflows
- Mentor others on effective human-AI collaboration
- Contribute to the broader community: write about what you've learned, share your workflows
- Push boundaries: experiment with subagent orchestration, headless automation, novel agentic patterns

7. Anti-Patterns to Avoid

Anti-Pattern	The Risk	The Fix
All Vertical, No Horizontal	You become a Claude Code power user who can't make sound architectural,	Deliberately invest in at least 3–4 horizontal skills. Use Claude Code itself as a learning tool for adjacent domains.

Anti-Pattern	The Risk	The Fix
	product, or security decisions. Speed without direction.	
All Horizontal, No Vertical	You dabble in AI tools but never develop the deep expertise that produces exceptional results. Average at everything, excellent at nothing.	Commit to Claude Code as your primary tool for at least 3 months. Go deep before going wide.
Prompt Jockey	You treat Claude Code as a magic oracle rather than a collaborator. Copy-paste prompts without understanding the output.	Always read and understand generated code. Build the habit of review-first workflows.
Automation Absolutist	You try to automate everything with Claude Code, including decisions that require human judgement and nuance.	Develop strong instincts for when to delegate and when to decide. The human-in-the-loop is a feature, not a limitation.
Context Hoarder	You dump entire codebases into context without curation, resulting in confused, generic outputs.	Practice deliberate context engineering. Less but better information produces better results.

8. The Future of the T

The T-shaped model isn't static. As Claude Code evolves—with more capable models, better agentic reasoning, and deeper integrations—the vertical deepens and the horizontal widens. What was expert-level today may become baseline tomorrow.

But the fundamental shape endures: the engineers who will define the next era of software development are those who combine deep mastery of AI-augmented workflows with the breadth of knowledge needed to apply that mastery to problems that matter.

Claude Code isn't just a tool in your toolkit. It's a new way of engineering—one that rewards deep investment, systematic thinking, and the kind of cross-disciplinary fluency that the T-shaped model has always championed.

Start Here

Install Claude Code. Create a CLAUDE.md. Ship something real. Then keep going deeper while spreading wider. The T reveals itself through practice, not planning.