A PROMPT ENGINEERING FRAMEWORK

DELTA

Closing the Specification Gap

- what your domain requires
- what the model produces

.NET 9 · JAVA 21 · PYTHON · TYPESCRIPT

The Ten Laws · The Specification Frame · 100+ recipes

Sandeep Dhuri

ACUITY PRESS

DELTA

Closing the Specification Gap

A Prompt Engineering Framework for Enterprise Software Teams

Sandeep Dhuri

Acuity Press

First edition, 2026

Published by Sandeep Dhuri · Acuity Press

ISBN: 979-8-9964807-0-8 (paperback) ISBN: 979-8-9964807-1-5 (ebook, EPUB)

Printed in United States of America

Disclaimer

The information in this book is provided for educational and informational purposes only and reflects the author's professional experience and research. The author and publisher have made every effort to ensure the accuracy of the content at press time but assume no responsibility for errors, omissions, or any losses or damages arising from reliance on the material. This book is not legal, financial, medical, regulatory-compliance, or other professional advice; readers operating in regulated environments (including but not limited to HIPAA, PCI-DSS, GDPR, and SOC 2 contexts) should consult qualified professionals for guidance specific to their circumstances. All code examples should be reviewed and tested in your own environment before any production use. Mentions of specific products, services, vendors, or model providers reflect the state of the field at press time; capabilities, pricing, and availability change frequently and should be verified with the relevant provider before use. The views and opinions expressed in this book are solely those of the author and do not represent the views, positions, or opinions of the author's employer or any of its affiliates. The book was written in the author's personal capacity; nothing in it should be attributed to, or construed as endorsed by, any current or former employer.

1

Why Prompting Is an Engineering Skill

The Specification Gap and the real cost of vague requirements

*"**The model is not wrong. It produced what was statistically most likely. You asked for what was statistically most likely. That is the entire problem, stated precisely.**"*

— Every incident in this chapter was preventable with one constraint clause.

In This Chapter

1. Why AI output quality is a function of specification quality, not model capability
2. The Specification Gap: Domain, Context, and Constraint components
3. Five illustrative failure modes with financial quantification
4. The three-tier prompting skill model and honest self-assessment
5. Laws 1 and 2 of Enterprise AI Prompting

1.1 The Question That Changes Everything

Before we talk about what to put in a prompt, let's be precise about what a prompt actually is.

When you ask an AI coding assistant to 'write a payment service in C#', you aren't asking a question. You are beginning a document. The model's job is to predict the most statistically likely continuation of that document, step by step, word by word, based on patterns in its training data. It's completing, not reasoning. It does this magnificently, at scale, across everything it has ever processed.

And that's the problem. The most likely continuation of 'write a payment service in C#' is, statistically, a tutorial example. Tutorial examples use double for money. They skip idempotency. They don't implement your audit logging pattern. They've never heard of your Money<Currency> type.

The model produces what is most likely. You need what is domain-correct. The gap between those two things is the Specification Gap, and every technique in this book closes one part of it.

STOP AND THINK — Before reading on...

Think about the last AI-generated service your team merged. What was the most important domain constraint it violated or omitted? Write it down before reading on.

LAW 1 OF ENTERPRISE AI PROMPTING

The AI produces what is average in its training data. Your domain requires what is specific. Specificity must come from you.

Frontier code-generation LLMs are dramatically more capable than the models of two or three years ago. They still do not know that a given team uses Money<Currency> instead of decimal, that HIPAA prohibits patient names in log statements, or that a given payment endpoint must be idempotent. These are specific requirements that exist in a specific codebase. No model improvement will ever make them default outputs. They must be specified.

1.2 Five Failure Modes Worth Naming

The incidents that follow are teaching scenarios. They are not based on any specific organization's post-incident write-up, and they do not draw on confidential information. They demonstrate failure mechanisms documented across the published work on AI coding tools. Names, identifying details, and dollar figures are illustrative; the figures indicate realistic ranges. The root cause is identical in every case: a missing constraint.

Failure Mode 1: Float Arithmetic in a Financial Batch Service

PATTERN — The $2,341 Precision Tax PATTERN-3035

A payments platform team was six months into Claude Code adoption. Sprint velocity was up 34%. Then their month-end reconciliation job produced a $2,341.17 shortfall against their bank statement. The discrepancy had been compounding across hundreds of thousands of transactions over 23 days, invisible below monitoring thresholds.

```
THE COST OF THIS MISTAKE
Incident: AI-generated double arithmetic in C# batch aggregation
Total cost: Reconciliation discrepancy + 3 days incident
    ↪ investigation + regulatory disclosure
Time to resolve: 3 days investigation, 15 minutes code fix, 2 weeks
    ↪ regulatory documentation
 Prevention: "Money<TCurrency> for ALL amounts — never decimal,
    ↪ double, or float directly" — 10 words
```

Failure Mode 2: Protected Health Information in Application Logs

PATTERN — The Log Export That Crossed a BAA Boundary PATTERN-4698

A clinical workflow platform generated a medication management service using a frontier code-generation LLM. The output was well-structured, passed code review, and shipped to production on a Thursday. Eleven weeks later, during a quarterly HIPAA security review, their Security Officer searched their Datadog export and found: 'Patient [FULL NAME] updated prescription for [MEDICATION] at [APPOINTMENT TIME] for [ICD-10 CODE].' Every field was Protected Health Information (PHI). All of it was in logs exported to a third-party observability platform with no BAA in place.

```
THE COST OF THIS MISTAKE
Incident: PHI in application logs shipped to third-party analytics
    ↪ platform without HIPAA BAA for 11 weeks
Total cost: Regulatory disclosure obligation + BAA review + complete
    ↪ log purge from third party + HIPAA risk assessment
Time to resolve: 11 weeks undetected, 3 days remediation, 6 weeks
    ↪ regulatory documentation
 Prevention: "PHI (name, DOB, diagnosis, medication) MUST NEVER appear
    ↪  in log statements — correlation ID only"
```

Failure Mode 3: Race Condition Under Flash Sale Load

PATTERN — Hundreds of Orders for Items That Didn't Exist PATTERN-4372

An apparel e-commerce platform launched a limited-edition drop. 31,000 checkout requests arrived in the first 12 seconds. The inventory reservation service, generated with an AI coding assistant three weeks earlier, used a check-then-act pattern: read available stock, verify sufficient, decrement. Under load, hundreds of threads simultaneously read 'stock: 8', all checked 'stock >= requested', and all decremented. Hundreds of customers received order confirmations for items that could not be fulfilled. The cancellation and compensation process ran for four days.

```
THE COST OF THIS MISTAKE
Incident: Race condition in inventory service: 847 oversold items
    ↪ during flash sale (31k concurrent users)
Total cost: hundreds of order cancellations + customer compensation
    ↪ vouchers + 4 days customer service + brand damage
Time to resolve: 4 days customer communications, 1 day code fix +
    ↪ deployment
 Prevention: "@Version on Product entity. @Retryable(
    ↪ OptimisticLockingFailureException.class, maxAttempts=3)"
```

Failure Mode 4: Application Form That Failed Its Accessibility Audit

A consumer financial-services platform built a self-service loan application form using an agentic AI coding tool. The form was functional and mobile-responsive. It failed its WCAG 2.1 AA / ADA Title III accessibility audit on 14 checkpoints: ARIA roles missing on error states, color-only error indicators (no role='alert'), incorrect label associations, sensitive identifier field with autocomplete='on', no programmatic announcement of required fields. Each failure was a separate legal exposure. Without an explicit WCAG 2.1 AA constraint block, the generated form reflected average web form training data, which does not meet enterprise accessibility standards.

Failure Mode 5: Webhook Handler That Double-Charged 340 Customers

A SaaS billing team's Stripe webhook handler processed events and returned 200. When Stripe's infrastructure experienced a 32-second delay, they retried hundreds of webhooks. The handler ran twice. Hundreds of customers were charged twice. The missing constraint: 'Order of operations is mandatory: (1) verify signature, (2) check idempotency key before any processing, (3) execute business logic, (4) mark processed after commit.' Without the explicit order, the generated handler checked idempotency after the business logic, making the check ineffective against retries that arrived during processing.

Beyond These Five

The five failure modes above span precision, privacy, concurrency, accessibility, and idempotency, but they are not exhaustive. The same missing-constraint pattern produces dozens of distinct failure shapes across enterprise codebases. The Hall of Shame in Appendix G catalogs ten further documented patterns. A short list of recurring categories beyond the five above:

SQL injection in AI-generated dynamic queries: generated repository code that concatenates request parameters into SQL strings instead of using parameterized queries — observed even when the surrounding codebase uses parameterized access exclusively.

Broken authorization in admin endpoints: generated controllers that accept a userId from the request body and act on it without verifying the caller's claim — an IDOR vulnerability disguised as a feature.

Time-zone and DST bugs in scheduled jobs: generated cron expressions and date arithmetic in server local time when the domain requires UTC-anchored boundaries — invisible until a daylight-saving transition double-runs or skips a job.

Cache invalidation failures: generated read-through cache code that stores derived values without invalidating dependents on write — visible as "the dashboard is stale" tickets that no one can reproduce in development.

Resource leaks under retry loops: generated retry-with-backoff code that opens a fresh database connection or HTTP client per attempt, exhausting the connection pool the moment a downstream dependency degrades.

JWT validation gaps: generated authentication middleware that decodes a token but does not verify its signature, issuer, or expiry — a tutorial-pattern that is acceptable for a demo and catastrophic in production.

Backwards-incompatible API changes: generated "refactors" that rename a public field on a request DTO or change a default value, breaking every downstream consumer at the next deployment.

Each of these is one missing constraint clause away from being prevented. The remainder of this book is the toolkit for writing those clauses on purpose, before the model fills them in for you.

1.3 The Three-Tier Skill Model

The pattern is consistent across published work on AI coding tools: AI output quality is predictable from prompting tier more reliably than from team seniority, codebase complexity, or model choice.

Tier	Mental Model	Output Quality	Team Impact	The Tell
Tier 1 Completion	AI = smart search engine	30% excellent, 40% adequate, 30% subtly wrong in domain-specific ways.	Negative — plausible-looking violations of domain standards.	Says 'the AI isn't that useful' while using single-sentence prompts.
Tier 2 Director	AI = competent contractor who needs guidance	60% good, 30% needs significant revision.	Neutral — good individual results, no team leverage.	Gets good results but reinvents every prompt. Nothing is shared.
Tier 3 Spec Engineer	AI = specification executor	~85% production-ready on first attempt (see App. J).	Strongly positive — prompt library compounds quarterly.	Has a reviewed prompt library. Reviews for domain correctness. Trains the team.

55%	***faster task completion for developers using AI coding tools versus a control group writing the same task without — measured in a controlled study of 95 developers writing identical HTTP servers. Structured prompting amplifies this baseline benefit; see Appendix J. Source: GitHub research, 'The Impact of AI on Developer Productivity' (2022) See Research & Statistics Notes (see Appendix J)***

> *"Many engineers spend months at Tier 2, genuinely believing AI tools have a ceiling. The pattern reverses the moment a structured prompt frame is introduced — domain-correct services start arriving on the first attempt, and PR review comments on missed conventions drop sharply. The model's capability is always there. The prompt has been the constraint."* ***— Staff Engineer, London-based RegTech platform***

1.4 The Tuesday Afternoon Test

Apply this to your last AI-assisted pull request right now:

Did the prompt name the interface the generated code must implement?

Did it specify the error handling pattern your team uses (Result<T>, exceptions, or something else)?

Did it name at least one domain-specific constraint — a regulatory requirement, a precision rule, a business invariant?

Did it include the proprietary types your domain uses (Money<T>, Result<T>, custom value objects)?

Is that prompt saved somewhere a colleague can reuse it tomorrow?

Five 'yes' answers: you are at Tier 3. Three or four: approaching Tier 2. Fewer: you are at Tier 1, and the next two chapters will change that permanently.

LAW 2 OF ENTERPRISE AI PROMPTING

A vague specification is not a prompt. It is a request for the model's best guess. Only you know if the guess is right.
There is no feedback loop that tells the AI coding assistant that its output violated your regulatory requirements. It can't see your production incidents. It doesn't know your Money<Currency> type exists. You are the only feedback loop. Specification is how you close it.

1.5 The Productivity Paradox Is a Specification Paradox

There is a paradox in the 2026 data on AI-assisted development, and it is the reason this book exists. Developers using AI coding tools consistently report feeling faster. When researchers measure what actually ships, the picture is more complicated. In one widely discussed randomized controlled trial of experienced open-source developers, participants given frontier AI tools took longer to complete real tasks than those without them, even though they believed the tools had sped them up. Independent analyses of large code corpora over the same period reported related effects: more code produced, but also more duplication, less refactoring, and a higher defect rate per change. The figures vary by study and will keep moving; the direction is consistent enough to take seriously.

It is worth being careful about what these studies do and do not establish. They are early, their methodologies differ, and vendor-sponsored numbers tend to be rosier than independent ones — a divergence the reader should weigh rather than ignore. This book does not rest its argument on any single statistic, and the specific percentages quoted in industry reports should be treated as snapshots, not constants. (Sources for the studies referenced here are listed in Appendix J.) What is durable is not the number. It is the mechanism underneath it.

That mechanism is the subject of this book. The reported failures share one shape: code that looks correct, passes a quick review, and then behaves wrong for the domain — the float that drifts a financial total, the handler that double-charges under load, the field that leaks into a log. The tool removed the friction of producing code faster than the team could absorb the work of verifying it. The bottleneck moved downstream, from typing to judgment. None of these are model failures. The model produced the most statistically likely code for the prompt it was given. The failure is upstream: the prompt never told the model what the domain actually required.

Stated plainly: the productivity paradox is a specification paradox. A more capable model given a vague specification does not close the gap — it produces a more convincing wrong answer, faster, which is precisely what makes the paradox worse as models improve. The teams that escape it are not the ones that generate the most code; they are the ones whose specifications are precise enough that the output is right the first time and cheap to verify. That discipline — the Specification Frame, the Ten Laws, prompts treated as code — is what the rest of this book teaches. The paradox is the problem. Specification is the way out.

KEY TAKEAWAYS

+ The Specification Gap has three components: Domain Gap (regulatory constraints and proprietary types), Context Gap (your actual codebase), and Constraint Gap (your team's hard-won invariants).
+ Five illustrative failure modes — float-for-money, PHI in logs, missing optimistic locking, inaccessible forms, and non-idempotent webhooks — each demonstrating how a single missing constraint clause produces costs of the kind well documented in the public literature.
+ Tier 3 prompting consistently produces high first-attempt production-readiness — somewhere around 85%, with most observations falling between 80% and 90% in my reading of the published re-

search and post-mortem literature on AI-assisted development. The difference from Tier 1 is structural discipline, not effort.
+ Laws 1 and 2: The AI produces averages. Your domain requires specifics. A vague specification is a request for the model's best guess. These are the two most important truths in this book.

TRY IT NOW

1. Apply the Tuesday Afternoon Test to your last merged AI-assisted PR. Write down the two most important missing constraints.
2. Name the three most expensive invariants in your system — the ones whose violation causes production incidents. These become your first mandatory constraints for every AI interaction in your domain.
3. Show this chapter to one colleague. Ask them to apply the Tuesday Afternoon Test to their last AI-generated PR. Compare your scores.

UP NEXT — Chapter 2: How LLMs Work — Just Enough Theory

The model completes your document. It doesn't answer your question. Chapter 2 explains why that distinction changes everything about how you should write prompts.

You just read Chapter 1.

The full book — 19 chapters, 11 appendices, the Ten Laws, the Specification Frame, and 100+ compilable recipes across .NET 9, Java 21, Python 3.12, and TypeScript — is **free**. No payment, no sign-up, no catch. The author receives no royalties.

Download the complete book

https://acuity.press

Companion code (MIT-licensed)

https://acuity.press/code

Delta: Closing the Specification Gap · Sandeep Dhuri · Acuity Press, 2026
Paperback ISBN 979-8-9964807-0-8 · EPUB ISBN 979-8-9964807-1-5