Transformation to the Agentic SDLC Made Easy

RIDE AGENTS HOLD THE REINS



THE HANDBOOK FOR SOFTWARE TEAMS IN THE AGE OF AI

CHAITANYA

PRAVAAS ENTERPRISE PVT LTD

RIDE AGENTS
HOLD THE REINS



Transformation to the Agentic SDLC Made Easy

RIDE AGENTS HOLD THE REINS

THE HANDBOOK FOR SOFTWARE TEAMS IN THE AGE OF AI

CHAITANYA



PRAVAAS ENTERPRISE PVT LTD

Ride Agents Hold the Reins
Transformation to the Agentic SDLC Made Easy

Copyright © 2026 Pravaas Enterprise Pvt Ltd. All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other noncommercial uses permitted by copyright law.

This book is nonfiction with illustrative fictional scenarios. Maya Iyer, Fernbridge, and every character and event in the running case study are invented; any resemblance to real people or companies is coincidental. Product and company names mentioned remain the property of their respective owners.

The code in this book is provided as a teaching example, without warranty, and may be used under the MIT License.

Published by Pravaas Enterprise Pvt Ltd. First edition, 2026.
ISBN 000-0-00000-000-0 (placeholder)

CONTENTS

Introduction 9

ACT I RIDE *from manual SDLC to agentic SDLC*

PART I GET IN THE SADDLE

1 The Manual SDLC 21
2 Anatomy of an AI Agent 43
3 Choosing Your Agent Toolchain 69

PART II FIRST RIDES

4 Agentic Requirements and Design 89
5 Agentic Coding and Legacy Modernization 105
6 Testing, Review, and Release with Agents 121

ACT II HOLD THE REINS *control, trust, and scale*

PART III MASTER THE REINS

7 Context and Skills Engineering 141
8 Verification and Quality Gates 159
9 Security, Compliance, and Guardrails 177

PART IV LEAD THE HERD

10 Multi-Agent Orchestration 197
11 Roles, Metrics, and ROI 214
12 Your 90-Day Agentic SDLC Roadmap 232

BACK MATTER

Quickstart: Your First Seven Days 250
Prompt and Spec Templates 252
Glossary 257
Appendix: Where Riders Fell 261



THE HORSE DOESN'T CARE WHO'S RIDING

"A rider uses the horse's strength and supplies the judgment."

At 9:41 on a Tuesday morning, Maya Iyer watched a machine do in eleven minutes what her team had quoted at two sprints.

To understand why that moment shook her, you need to know what her Monday looked like.

Maya led engineering for a product team at Fernbridge, a B2B software company with a product customers liked and a codebase that had grown the way most codebases grow: one urgent deadline at a time. Her team worked the way most teams still work. Requirements were written by hand, then rewritten by hand when they turned out to be vague. Designs lived in slide decks. Code was typed line by line. Tests were written last, if there was time. Every pull request waited in a queue for a human with ten other things to do. Releases were a ceremony. Documentation was a promise.

The team was not slow or careless. They were good at their jobs, and they were tired. The manual software development life cycle asks a person to do every step by hand, and every handoff between steps is a place where work sits and waits. Maya kept a list taped to the side of her monitor, and every item on it would be familiar to any engineer, tester, or product manager.



FIGURE 0.1 *Maya's Monday list. If even four of these are true on your team, this book is for you.*

None of these problems was new. Her team had been fighting them for years with the usual tools: more process, more meetings, one more hire, one more dashboard. Each fix helped a little, then the backlog grew back.

Then, on Tuesday, a contractor sent her a demo link. In a moment of skeptical curiosity she pointed an AI coding agent at the thing nobody wanted to touch: the Reports service, two major framework versions behind, with 11% test coverage and a security scanner that flagged forty dependencies every Monday. Then she went to refill her coffee.

When she came back, there was a branch. There were tests. There was a pull request description that was, annoyingly, better than most of the ones her team wrote.

She did not feel excitement. Her first feeling was a cold drop in her stomach. Her second, about four seconds later, was a question that this entire book exists to answer:

"Okay. But who is responsible for this?"

Most of what follows is an attempt to answer it.

WHAT THE MANUAL PROCESS GOT RIGHT

Before we talk about agents, let us be fair to the way we have always built software. The manual SDLC is not stupid. It was designed for a world where producing work was slow and expensive, so it surrounds production with planning, sign-offs, queues, and checkpoints. It protected us from expensive mistakes, and it made us careful.

But it also made waiting normal. If you track one feature from first idea to first customer, you will usually find that people spent far more of the calendar waiting than working: waiting for clarification, for a review, for an environment, for the Friday release window. Each wait made sense on its own. Together, they are the reason a two-week idea takes six weeks to ship.

Agents go straight at that waiting, and they don't wait for the Friday window.

WHAT IS DIFFERENT THIS TIME

If you work in IT, you have lived through shifts before: client-server to web, web to mobile, on-premises to cloud, waterfall to agile, agile to DevOps. Each one rearranged which skills were valuable and which rituals were wasteful. Each one punished the people who waited for certainty and rewarded the people who learned by doing.

This shift is different in one important way. Earlier shifts changed *where* we ran software and *how* we organized people. This one changes *who does the typing, the reading, the testing, the triaging, and the summarizing*. Agents do more than assist the software development life cycle; they take part in it. They read tickets, open files, run commands, write code, run tests, read the failures, and try again, in a loop, without you.

That is powerful, and it is a little frightening. Both reactions are reasonable.

THE TWO BAD RESPONSES

Most IT professionals meet this moment in one of two ways, and both are traps.

The Spectator says, "I'll wait until it settles down." Spectators are not foolish. They have seen hype cycles before and they have the scars. But the cost of waiting is not zero. While they wait, their peers are building the one skill that is about to matter most: knowing how to direct, constrain, and verify work done by agents.

The Passenger says, "I'll let the agent handle it." Passengers feel productive. The code appears, the tickets close, the dashboards go green. Then one day a confident-sounding agent deletes the wrong thing, leaks the wrong key, or ships a subtle bug that nobody understands because nobody actually read the code. Passengers tend to fail all at once, and in public.

THE THIRD OPTION: BE THE RIDER

The horse doesn't care who is riding. It will carry a skilled rider to the summit and a careless one off the cliff, with exactly the same enthusiasm. That is both the problem and the opportunity.

A rider doesn't pull the cart or pretend the horse isn't there, and doesn't close their eyes and hope either.

A rider **uses the horse's strength and supplies the judgment**. They choose the route, they know the terrain, and they know when to pull the reins.

That is the stance of this book. Agents bring speed, stamina, and breadth. You bring intent, context, accountability, and the ability to say *no, not that way*. Neither is much use without the other, and the people who learn to ride well will be the ones teams want to hire and promote.

TWO ACTS

The title of this book is a sentence in two halves, and so is the book.

Act I: Ride. You start where Maya started, in the manual SDLC, and move it to an agentic one, phase by phase: requirements, design, code, tests, reviews, releases, operations. Every chapter shows the manual way, then the agentic way, then gives you a hands-on lab and a win you can ship this week. By the end of Act I, your team is moving faster, and you will also have felt where it gets dangerous.

Act II: Hold the Reins. Speed without control is how passengers get hurt. Act II teaches the harness that keeps agents on the trail: **context** (the right information at the right time), **skills** (the tools and domain knowledge each phase needs), and **guardrails** (security, quality, cost, and risk controls). Then it scales the practice to multiple agents, to your whole team, and to your career.

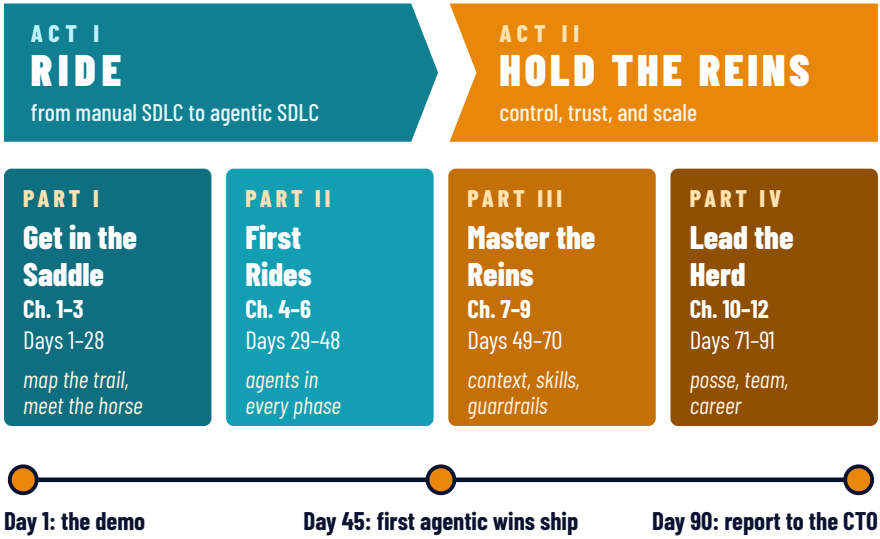


FIGURE 0.2 *The book follows Maya's ninety days. Act I gets you riding; Act II keeps you in control.*

WHAT THIS BOOK IS (AND ISN'T)

It is a field guide. Every chapter follows the same loop:

1. **The Hook:** a scene from Maya's ninety days, ending on a problem.
2. **The Problem:** what breaks in the traditional way of working.
3. **The Shift:** the idea and the mental model that fixes it.
4. **Hands-On Lab:** a practical exercise you can do this week.
5. **Where Riders Fall:** the mistakes teams make most often.
6. **Reins Checklist:** a one-page summary to keep.



FIGURE 0.3 *Every chapter follows the same six-step loop, from a problem you recognize to a practice you can repeat.*

It is tool-agnostic. Tools change monthly; principles last years. I'll show concrete examples, but every technique is explained in a way that transfers to whatever agent you use next quarter.

It is not a hype book. You won't find promises that agents replace engineers, or that you'll 10x anything by Friday. You'll find honest accounting of what works, what fails, and how to tell the difference before it costs you.

It is not only for developers. Architects, QA engineers, DevOps and SRE folks, product owners, engineering managers, and IT leaders all have a seat. The software development life cycle belongs to everyone who touches it.

WHAT YOU WILL HAVE BY THE END

- A map of your own manual SDLC and the places agents help most.
- A working agentic workflow for every phase, tested on a real project.
- A harness of context, skills, and guardrails that your security and compliance teams can review and approve.
- A before-and-after scoreboard you can show your leadership.
- A ninety-day plan for your team, and a clear answer to "what is my role now?"

HOW TO READ IT

- **New to agents?** Read straight through. The chapters build on each other.

- **Already using them daily?** Skim Chapters 1 to 3 and start with Chapter 4, or jump to Act II (Chapter 7) if control is your urgent problem.
- **A leader?** Read Chapters 1, 9, 11, and 12 first, then go back for the rest.

Do the labs. Reading about riding does not teach you to ride. The labs use a single running project, Fernbridge's Reports service, so that by the final chapter you will have built something real and have the instincts to repeat it at your own company.

Maya has ninety days. Let's start with hers.

ACT I · RIDE

PART I

GET IN THE SADDLE



Why the manual trail is straining, what an agent really is, and how to climb on without getting thrown.



01

CHAPTER ONE

THE MANUAL SDLC

"Before you saddle a new horse, walk the old trail once and count the potholes."

THE HOOK

DAY 1

Maya closed the laptop lid halfway, as if the thing might hear her.

By lunch, the demo was all anyone at Fernbridge was talking about. Dev, her most senior backend engineer, shrugged: "Looks like autocomplete on steroids." Priya, the QA lead, went quiet and then asked, "Does it write the tests too? Because if it does, who checks the tests?" Jonas, the newest hire, six months out of university and still afraid to merge to main, said what the others were thinking: "So… are we still needed?"

And then Leena, the product manager, who had been listening from the doorway, said the sentence that made Maya's stomach drop a second time: "Great. So can we get the pricing-page redesign and the mobile offline mode into this quarter?"

Nobody laughed. It was a fair question, and the four people in the room had four different answers to it.

Maya had until Friday to give her CTO a plan. She had no plan. But she had one instinct, and it came from years of watching teams buy tools to fix problems they had never measured. She was not going to decide anything about agents until she could say, in numbers, how her team actually worked today.

She picked the last feature the team had shipped, the scheduled-reports export, and asked everyone to reconstruct its journey from the first chat message to the cus-

tomer's first click. Then she wrote one question on the whiteboard:



FIGURE 1.1 *Maya's whiteboard, Day 1.*

It took an hour to fill the board, and by the end Dev had stopped arguing about the details.

THE MANUAL SDLC, MAPPED

The scheduled-reports export had taken thirty working days from idea to production. Thirteen of those days had someone actually working on it. The other seventeen were spent waiting.

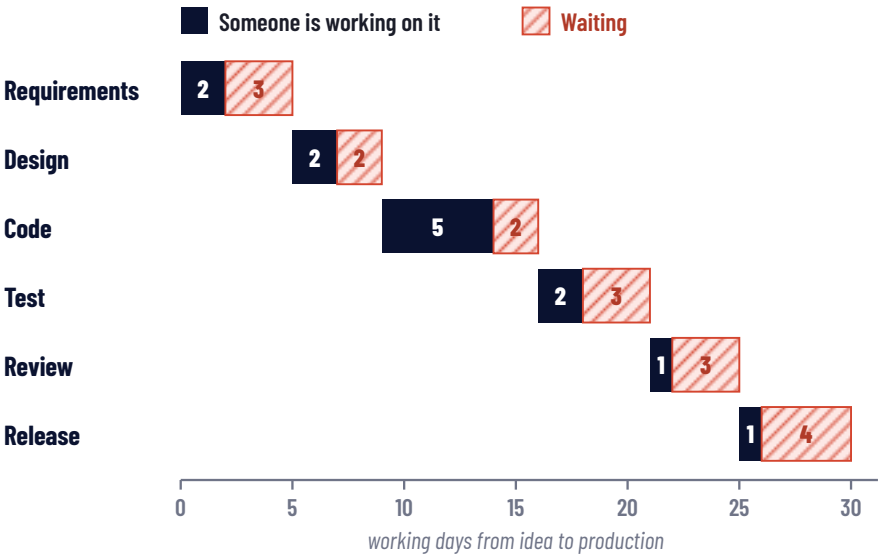


FIGURE 1.2 *One feature, idea to production, in a manual SDLC. The numbers are illustrative of Maya's team; yours will differ, but the shape rarely does.*

Look at the shape of that picture. The work blocks are not where the calendar goes. The gaps between them are: a clarification that arrived on Thursday instead of Monday, a pull request that sat for three days, a regression suite that ran for two, a release window on Friday. Each wait had a reason. Together they cost Maya's team more than the typing ever did.

This is the manual SDLC in one picture, and it explains the list on Maya's monitor. Here is the same list, mapped to the book:

THE PROBLEM	WHAT IT LOOKS LIKE	WHERE WE FIX IT
Backlog that never shrinks	Hundreds of old tickets; every quarter promises more than capacity allows	Ch. 4, 5
Pull requests that wait	Reviews take days; reviewers lose context; PRs grow stale	Ch. 6, 11
Slow regression cycles	A multi-day test freeze before every release	Ch. 6, 8
Bugs from missing tests	The Friday hotfix; the "who touched this?" thread	Ch. 8
The service nobody touches	Low coverage, outdated framework, knowledge in two people's heads	Ch. 5
Stale documentation	The wiki says one thing, the code does another	Ch. 4, 7
Months to onboard	New hires afraid to ship; seniors become full-time guides	Ch. 7, 11
Vague requirements	Rework, because "what we built" is not "what was meant"	Ch. 4
Alert fatigue and toil	The same pager noise, the same manual runbook steps	Ch. 6, 10

That evening Maya did one more thing. She pulled six numbers from the team's dashboards, wrote them in the corner of the whiteboard, and photographed it. She called it her scoreboard.

MAYA'S SCOREBOARD, DAY 1	FERNBRIDGE BASELINE
First review wait on a pull request	2.5 days
Regression cycle before a release	4 days
Idea to production, a typical feature	6 weeks
Test coverage, Reports service	11%
New hire's first merged pull request	Week 6
Open dependency alerts	40

These Fernbridge numbers are fictional, chosen to look like many real teams. Measure your own in Lab 1. You will revisit this scoreboard at the end of each act.

THE PROBLEM

THE OLD SDLC WAS BUILT AROUND HUMAN TYPING SPEED

Why does the manual trail have so much waiting in it? Because every phase of the traditional software development life cycle quietly assumes one thing: **producing work is slow and expensive, and reviewing work is comparatively cheap.**

Look at how we organized ourselves around that assumption:

PHASE	OLD ASSUMPTION	RESULTING RITUAL
Requirements	Writing code is costly, so get it right on paper first	Long specs, sign-offs, change-control boards
Design	Rework is expensive	Heavy up-front design reviews
Coding	Humans type the code	Estimation in person-days; sprint capacity limits
Testing	Testing is a phase after coding	Separate QA cycle, regression suites run nightly
Review	Reading a few hundred lines is quick	Small PRs reviewed by a peer or two
Release	Deployments are risky	Release windows, change advisory boards
Operations	Humans watch the dashboards	On-call rotations and runbooks

Now introduce an agent that can produce a working draft of a feature, a refactor, a test suite, or a runbook in minutes. Something strange happens: **the bottleneck moves.**

Producing is now cheap. What's expensive is everything else:

- **Knowing what to ask for.** Vague intent produces confident, plausible, wrong output at machine speed.
- **Verifying.** Ten times more output means ten times more to check, and checking has not gotten ten times faster.

- **Keeping the system coherent.** Agents optimize the task in front of them, not the long-term shape of your architecture.
- **Owning the consequences.** An agent cannot be paged at 3 a.m. You can.

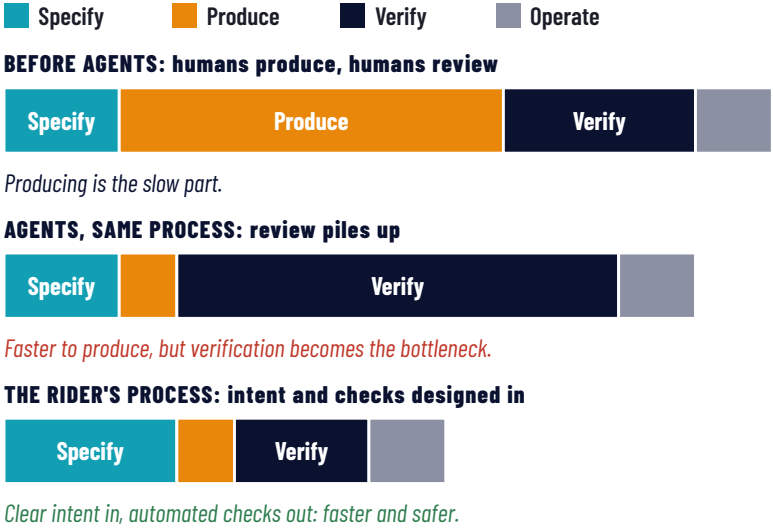


FIGURE 1.3 *The bottleneck moves. The sketch is illustrative, not measured: producing shrinks, while specifying and verifying grow.*

That is the heart of the problem, and it explains why the same tool produces euphoria in one team and chaos in the next. Teams that treat the agent as a faster typist keep their old process and drown in output they can't review. Teams that rebuild their process around the new bottleneck, **intent and verification**, move faster *and* more safely.

THE SHIFT

FROM WRITING THE WORK TO DIRECTING THE WORK

Here is the mental model for the rest of the book.

For decades, an IT professional's value was mostly **doing**: writing the query, writing the function, writing the test, writing the script. In an agentic SDLC, the value migrates toward four things:

1. **Intent:** deciding what should exist and why, precisely enough that it can be built.
2. **Direction:** giving the agent the context, constraints, and examples it needs to succeed.
3. **Verification:** proving, not assuming, that the result is correct, safe, and maintainable.
4. **Accountability:** owning the outcome in production.

Notice that none of these are "typing." All of them are deeply technical. This is why the fear that "agents make technical skill obsolete" is exactly backwards: **the better you understand the system, the better you ride.** A rider who doesn't understand horses is a passenger.

WHERE THE REINS GO ON THE SDLC

Every phase of the life cycle gets agent help, and every phase gets a human checkpoint. The checkpoints are the reins. Without them you have speed; with the checkpoints you have control.

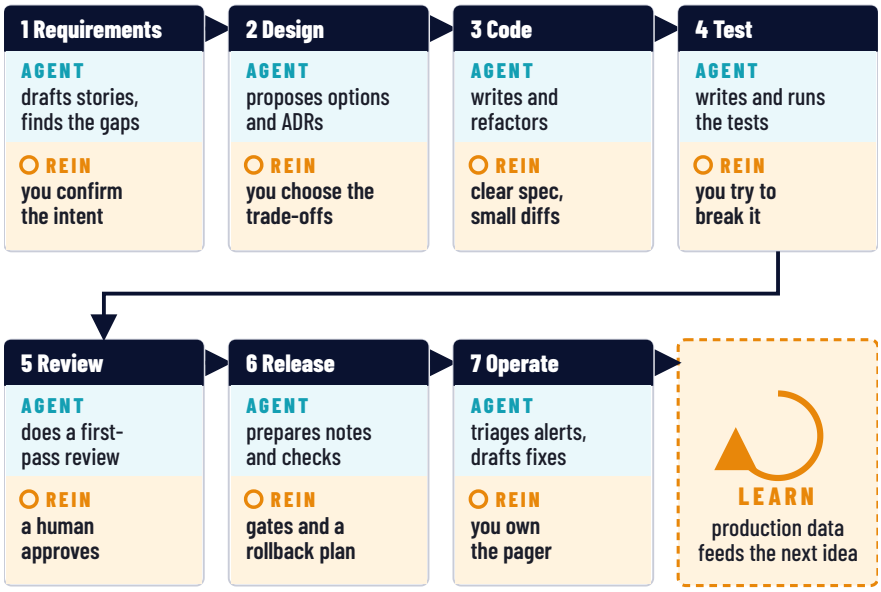


FIGURE 1.4 *The agentic SDLC. Agents do the heavy lifting in each phase; a human holds a rein at every handoff.*

In Act I you will learn to ride this trail fast. In Act II you will learn to hold the reins that make it safe: the context, skills, and guardrails that keep an agent doing what you meant.

THE RIDER'S ANATOMY

You'll see this picture again and again, so learn it now. It is also the map of the whole book:

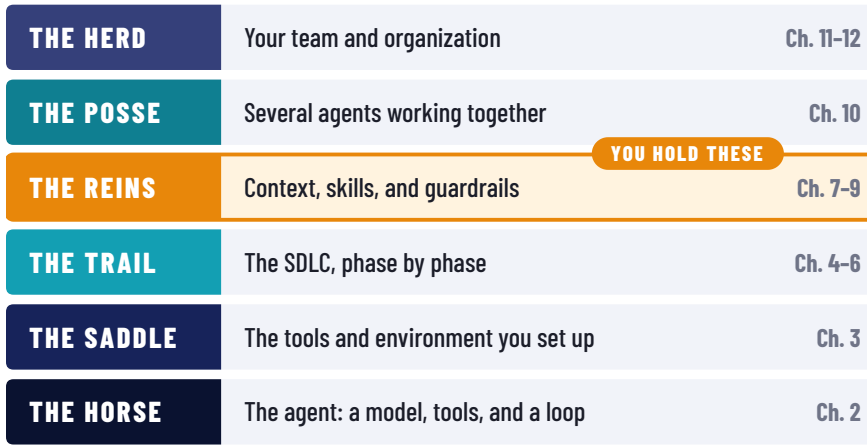


FIGURE 1.5 *The Rider's Anatomy, from the horse at the base to the herd at the top, with the chapters that cover each layer. Act I builds the lower layers; Act II is the reins, the posse, and the herd.*

THE THREE RIDERS

Back at Fernbridge, Maya's team naturally sorted into the three types from the introduction. You will recognize yourself, and probably your colleagues, in them.

The Spectator (Dev). Experienced, careful, and a little proud. His instinct, "autocomplete on steroids," was half right and half dangerous. Right, because much of the output is pattern completion. Dangerous, because the *loop* (plan, act, observe, retry) is what makes an agent qualitatively different from autocomplete. He was underestimating the horse.

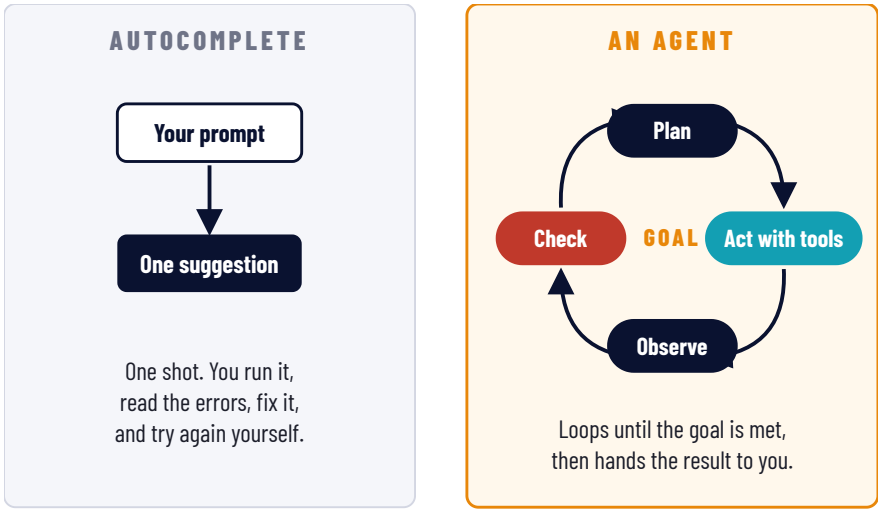


FIGURE 1.6 *Autocomplete answers once. An agent loops: it plans, acts with tools, observes the result, and tries again until the goal is met.*

The Passenger (Jonas). Eager and fast. Within a week he'd be pasting whole tickets into the agent and merging whatever came back if the tests passed. He was overestimating the horse and underestimating himself.

The Rider (who Priya was becoming). Her question, *"who checks the tests?"*, was the best question in the room. It shows the rider's reflex: **never ask only "can it do this?" Always ask "how will I know it did it right?"**

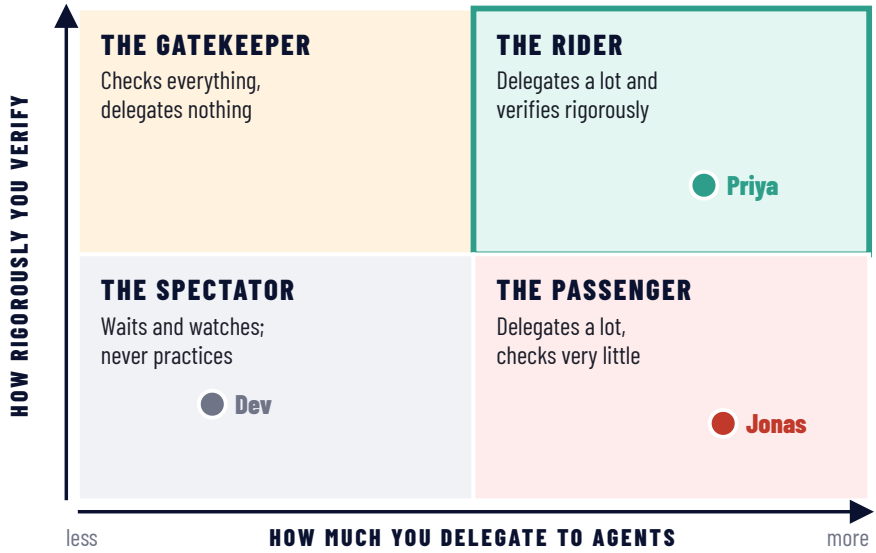


FIGURE 1.7 *Where each type of rider sits. Delegation without verification is the Passenger's trap; verification without delegation leaves the Spectator behind.*

> **Rider's rule #1:** Every time an agent produces something, ask: *How will I know this is correct?* If you have no answer, you don't have a result. You have a guess.

REFRAMING THE QUESTION: "ARE WE STILL NEEDED?"

Jonas's question deserves a straight answer, because your team is asking it too, out loud or silently.

Here is the honest version. The *mix* of work is changing. Routine translation of clear instructions into code is increasingly something agents do well. What grows in importance is the work agents cannot own: understanding messy business reality, making trade-offs among

competing goals, deciding what is safe to automate, carrying accountability, and judging quality. Some tasks will shrink, some roles will be reshaped, and some people will have to learn a lot, quickly.

I won't tell you that nothing will be lost. I will tell you that **the people who learn to direct and verify agents become more valuable, not less**, because they multiply what a single professional can responsibly deliver.

And Leena's question? It deserves the same honesty. Agents will change how fast some work moves, but they will not make a vague requirement clear, a risky release safe, or a quarter's worth of ambition fit into a month. A product manager who understands this becomes the best partner an agentic team can have. We will get to that in Chapter 4.

THE ANATOMY OF A "STAMPEDE MOMENT"

How do you know when agents have hit your organization? Look for these signs:

- A colleague quietly ships in a day what you estimated in a week.
- Pull requests start arriving that are larger, cleaner on the surface, and harder to review.
- Someone asks, "Can we just have the AI do the test cases?"
- Leadership starts using the word *productivity* in meetings where they used to say *quality*.
- Product starts asking why the roadmap can't move faster, now that "AI does the coding."

- People start using agents *without telling anyone*, because they aren't sure it's allowed.

That last one matters most. **Shadow AI** (unsanctioned, unreviewed agent use) is among the biggest risks in the early months. It isn't malice; it's momentum without reins. Chapter 9 covers governance in depth, but the first step is cultural: make it safe to say "I'm using an agent for this."

HANDS-ON LAB 1

Goal: In 45 minutes, see where your own calendar goes, and find out where agents can help you *right now* and where they shouldn't go near.

You need: A notepad or spreadsheet, and your calendar and ticket history from the last two weeks.

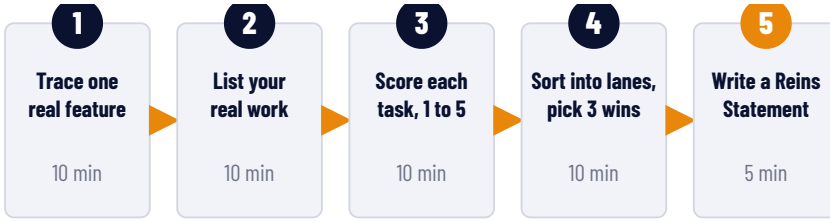


FIGURE 1.8 *The audit in five steps. Start from your real work, not from the tool.*

Step 1: Trace one real feature (10 min). Pick the last feature your team shipped. Reconstruct its journey from first message to production on a timeline, as

Maya did. For each phase, mark the days someone was *working* and the days it was *waiting*. Add them up. This is your own version of Figure 1.2, and the first line of your scoreboard.

Step 2: List your real work (10 min). Write down 15–20 specific tasks you did in the last two weeks. Be concrete: not "development" but "added retry logic to the refund API," "wrote a SQL query to reconcile Q3 invoices," "updated the runbook after the outage," "reviewed three PRs."

Step 3: Score each task (10 min). For each, give a score from 1 to 5 on four questions:

QUESTION	1 MEANS	5 MEANS
Clarity: Could I describe the finished result precisely?	Fuzzy, political, or exploratory	Crisp and testable
Verifiability: Can I quickly check whether the result is right?	Hard to verify	Easy (tests, diff, known answer)
Blast radius (reversed): If it's wrong, how contained is the damage?	Production, money, or customer data	Local, reversible, sandboxed
Context: Is everything needed available in files, docs, or the repo?	Lives in people's heads	Written down or in the code

Step 4: Sort into three lanes and pick your wins (10 min). Add the four scores (max 20):



FIGURE 1.9 *Your total score places each task in a lane.*

- **Green Lane (15–20): Ride now.** Delegate to an agent with light review. *Typical examples:* boilerplate, unit tests for existing code, documentation drafts, log summarization, simple refactors, one-off scripts.
- **Yellow Lane (9–14): Ride with reins.** Agent drafts; you set tight constraints and verify carefully. *Typical examples:* feature work in well-tested areas, migrations, API integrations, performance investigations.
- **Red Lane (4–8): Walk beside the horse.** Use the agent only for research and brainstorming, or not at all, until you've improved clarity, tests, or documentation. *Typical examples:* security-critical logic, ambiguous product decisions, anything touching production money or personal data without strong safeguards.

Then circle the three highest-scoring tasks that you do *often*, ideally ones that sit next to a wait you found in Step 1. Frequency matters: a small win repeated weekly beats a big win once.

Step 5: Write your "Reins Statement" (5 min). For your top task, finish this sentence:

> *"I will let an agent draft* ***__, and I will know it's correct because*** *____."*

Example from Maya's audit:

> *"I will let an agent draft unit tests for the Reports service, and I will know they're correct because they fail when I deliberately break the code, and because Priya reviews the edge cases."*

If you can't finish the second half of that sentence, the task belongs in the Yellow or Red lane until you can.

Bonus: Team version. Run this exercise as a 60-minute team workshop. Compare lists. The disagreements are gold, because they reveal where your team's documentation, test coverage, and shared understanding are weakest. Those are exactly the things that determine how well agents will perform for you.

⚠ WHERE RIDERS FALL

1. Automating the mess. If a manual process is unclear, an agent will do the unclear thing faster. Fix the obvious waste first, then automate what is left.

2. Judging the whole technology by one demo. A flashy success or an embarrassing failure from a single try tells you about that prompt, that day, that tool.

Evaluate on your own representative tasks, repeatedly.

3. Assuming "tests pass" means "correct." Agents can write the code *and* the tests, and the two can be wrong in agreement. Verification needs independence: separate tests, separate reviewers, deliberate "break it and see" checks (Chapter 8).

4. Starting with the scariest task or the least important one. Too scary, and one failure kills your team's trust. Too trivial, and you learn nothing. Start in the Green Lane with something visible and useful.

5. Treating adoption as an individual sport. If only one person on the team rides, the rest of the process still runs at walking speed, and review becomes the bottleneck. Transformation is a team practice.

6. Hiding it. Silent use creates inconsistent quality, security gaps, and fear. Talk about it openly from day one.

REINS CHECKLIST: CHAPTER 1

- ☐ I can describe where my team's calendar goes: working versus waiting.
- ☐ The bottleneck has moved from *producing* to *intent and verification*.
- ☐ My value is shifting from doing to **intent, direction, verification, accountability**.
- ☐ I can recognize the Spectator, the Passenger, and the Rider, including in myself.
- ☐ For any agent output I can answer: *"How will I know this is correct?"*
- ☐ I've completed my audit and know my Green, Yellow, and Red lanes.
- ☐ I've written at least one Reins Statement and recorded my baseline scoreboard.
- ☐ My team can talk openly about agent use, with no hidden "shadow AI."

◆ ◆ ◆

CLOSING SCENE

MAYA'S DAY 4

By Friday, Maya had her plan, and it fit on one page. It did not say "adopt AI." It said:

1. *Measure how we work today. The scoreboard is on the whiteboard.*
2. *Pick three Green Lane tasks next to our longest waits and run them as supervised experiments.*

3. *Write down how we'll verify each result, before we run it.*
4. *Report back in thirty days with evidence, not opinions.*

The CTO read it twice. "That's not what I expected," he said. "I expected you to tell me we should buy something."

"We might," said Maya. "But first I'd like to know how to ride."

He looked at the page again. "Thirty days," he said. "Don't make me regret it."

She walked back to her desk and opened a fresh terminal. For the first time since Tuesday, she wasn't dreading it.

Next up, Chapter 2: Anatomy of an AI Agent. We'll open the hood, learn what an agent actually is under the marketing, and build a tiny one ourselves in under sixty lines. Once you've seen how the horse thinks, you'll know exactly where to hold the reins.

BACK MATTER

QUICKSTART: YOUR FIRST SEVEN DAYS

You do not need ninety days to begin. You need seven short ones, each under an hour and a half and each leaving one thing you can show someone. Every day points to the lab that does the work, so this page is a map, not a replacement for the chapters. Stay on the Green Lane the whole week; nothing here touches production.

DAY	DO THIS	TIME	YOU WILL HAVE
1	Map the last feature your team shipped, step by step, with the waits (Lab 1)	60 min	A list of delays and six baseline numbers
2	Build and run the tiny agent, then read its trace line by line (Lab 2)	75 min	`agent.py` running and one trace you understand
3	Make one repository ready and write a run card (Lab 3)	60 min	The readiness check printing `READY`
4	Turn one vague ticket into an intent brief that passes the lint (Lab 4)	75 min	A brief, three criteria, one decision record
5	Write the memory file and the consumers document (Lab 7)	75 min	The context lint printing `PASS`
6	Add one gate: the diff guard or the contract check (Labs 6 and 8)	60 min	A check that fails when it should
7	Run your first supervised task from start to finish, then update the scoreboard (Labs 3 and 6)	90 min	One reviewed pull request, with its trace

THREE RULES FOR THE WEEK

- Work on a branch, in a clean tree, with no production credentials anywhere near the agent.
- Write the check first, ask for a plan before allowing an edit, and read the trace before the summary.
- Have someone who did not run the agent review the result.

WHEN A DAY GOES BADLY

IF THIS HAPPENS	TRY THIS	WHERE
The output looks right and you cannot tell if it is	Run a check you did not give the agent, and break the code on purpose	Chapters 6 and 8
Setting up the repository takes a day	That is the finding; fix the tests first and ask again later	Chapter 3
Security or legal asks you to stop	Bring the four gates and a threat map instead of arguing	Chapters 3 and 9
A colleague says it is just autocomplete	Show them a trace and the scoreboard, not a slide	Chapters 2 and 11

AFTER THE SEVEN DAYS

Pick the next three Green Lane tasks beside your longest waits, and write down how you will verify each before you run it. Then read the chapter for the phase that hurts most: Chapter 4 for requirements, Chapter 5 for old code, Chapter 6 for reviews and releases. Chapter 12 turns a working week into a ninety-day plan.

END OF THE SAMPLE

You have read the Introduction, Chapter 1 and the Quickstart. Chapters 2 to 12 each end in a hands-on lab, with tested Python code, and the book closes with templates, a glossary and a failure appendix.

RIDE AGENTS HOLD THE REINS

Get the full book from the page where you found this sample.